International Journal of Neural Systems, Vol. 6, No. 4 (2006) 1–11

World Scientific
www.worldscientific.com

PRUNING AND MODEL-SELECTING ALGORITHMS IN THE RBF FRAMEWORKS CONSTRUCTED BY SUPPORT VECTOR LEARNING*

PEI-YI HAO
Department of Information Management,
National Kaohsiung University of Applied Sciences,
Kaohsiung 807, Taiwan, R.O.C
haupy@cc.kuas.edu.tw

JUNG-HSIEN CHIANG
Department of Computer Science and Information Engineering,
National Cheng Kung University,
Tainan 701, Taiwan, R.O.C
jchiang@mail.ncku.edu.tw

Received 2 December 2005
Revised 24 April 2006
Accepted 6 June 2006

This paper presents the pruning and model-selecting algorithms to the support vector learning for sample classification and function regression. When constructing RBF network by support vector learning we occasionally obtain redundant support vectors which do not significantly affect the final classification and function approximation results. The pruning algorithms primarily based on the sensitivity measure and the penalty term. The kernel function parameters and the position of each support vector are updated in order to have minimal increase in error, and this makes the structure of SVM network more flexible. We illustrate this approach with synthetic data simulation and face detection problem in order to demonstrate the pruning effectiveness.

Keywords: Support vector machine; model selection; network pruning; kernel-based learning; RBF network.

1. Introduction

In this paper we address the problem of redundant support vectors in implementing support vector machine (SVM), a new pattern classification and function approximation technique recently developed by Vapnik.[8,28] Traditional techniques for pattern classification are based on the minimization of empirical risk — that is, on the attempt to optimize the performance on the training set. In contrast, SVMs minimize the structural risk — that is, the probability of misclassifying yet-to-be-seen patterns for a fixed but unknown probability distribution of the data.[27,28] This new induction principle relies on the theory of uniform convergence in probability, and it is equivalent to minimizing an upper bound on the generalization error. Furthermore, the SVM can be considered as an alternative training technique for polynomial, radial basis function, and multilayer perceptron classifiers,[17] in which the weights of the network are obtained by solving a quadratic programming (QP) problem with linear inequality and equality constraints, rather than by solving a non-convex, unconstrained optimization problem. In this paper we consider the problem of selecting

*Corresponding author.

basis functions in the radial basis function (RBF) framework constructed by support vector learning.

When solving the QP problem in SVM, we occasionally obtain support vectors corresponding to weights close to zero. Those support vectors (SVs) do not significantly affect the final sample classification and function approximation results and thus are redundant in the SVM decision procedure. It is an important issue to reduce the number of support vectors. In this paper we focus on developing an appropriate methodology to eliminate unnecessary support vectors. The related works includes eliminating the redundant SV in a post-processing step of support vector training,[3,4,17] some other methods use another training algorithm with a modified criterion instead of SVM.[10,15] For Least-Squares SVM, Suykens *et al.*[25,26] first proposed a pruning algorithm to obtain the sparse representation. Nevertheless, several variations have been proposed. In *nu*-tube SVM[23] one can control the lower bound of support vectors and the upper bound of errors; while in fixed-size SVM[26] one can fix the desired number of support vectors beforehand. The benefits of pruning redundant support vectors include: increasing the speed of classification and regression, reducing hardware or storage requirements, and in some cases enabling meaningful support vector extraction. Furthermore, removing unimportant weights in a trained neural network may improve its generalization ability.[20,22] This hypothesis could also be supported by the "bound of risk" introduced by Ref. 28. Since the Vapnik-Chervonenkis (VC) dimension of the learning machine is small, the bound of difference between true risk and empirical risk is tight. Intuitively, VC dimension can be adopted as a measure of the capacity of a learning machine, and a learning machine with few weights may have low VC dimension.[3] Therefore, eliminating redundant support vectors reduces the VC dimension and so improves the generalization ability.

In original support vector learning, parameters such as the regular constant C and the kernel function parameter q must be determined in advance. And the support vectors must be selected from the training samples. However, how to determining a set of proper parameter is still suboptimal and computationally extensive. With different parameter sets $\{C,q\}$, the SVM approach may result in different optimum solutions under the same training data set. Ref. 9 shows that it is not straightforward to select those parameters properly. Several validation approach approaches have been proposed for verifying the validation of the selection parameter set such as k-fold cross-validation, VC bounds, Xi-Alpha bond, and radius-margin bound. Those approaches still suffer from various problems mentioned in Ref. 9. In this research, instead of developing algorithms to find suitable selections for these parameters, we propose to employ traditional network pruning approaches to improve the learning performance. In addition, the positions of support vectors are also adjusted during the network pruning process, and this leads to a flexible formulation in the SVM approximation function.

The organization of this paper is as follows. We first give an outline of the SVM classification and regression in Sec. 2, and then describe the redundant support vectors problem. In Sec. 3, we provide both the penalty term and the sensitivity measure-based pruning algorithms. Experiments are then discussed.

2. Support Vector Learning Approach

2.1. *SVM classification*

Let $\{(x_1, y_1), \ldots, (x_l, y_l)\} \subset \aleph \times \{-1,\ 1\}$ be a given training data set, where $\aleph$ denotes the space of input points. The SVM classification is to map data points x_i into a high dimensional feature space via a nonlinear transform Φ and to classify them by a hyperplane $w \cdot \Phi(x) + b$ in feature space. The task of SVM classification is therefore to

$$\begin{aligned} \underset{w,b,\xi_i}{\text{minimize}} \quad & \|w\|^2/2 + C\sum_i^l \xi_i \\ \text{subject to} \quad & y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall\, i \end{aligned} \tag{1}$$

where the constant $C > 0$ determines the trade off between the maximum of margin and the amount up to which deviations are tolerated. Its dual quadratic programming classification problem is

$$\begin{aligned} \underset{\lambda_i}{\text{maximize}} \quad & -\frac{1}{2}\sum_{i,j=1}^{l} \lambda_i\lambda_j \left\langle \Phi(x_i) \cdot \Phi(x_j) \right\rangle + \sum_{i=1}^{l} \lambda_i \\ \text{subject to} \quad & \sum_{i=1}^{l} y_i\lambda_i = 0 \\ & \lambda_i \in [0, C] \quad \forall\, i \end{aligned} \tag{2}$$

where λ_i are the Lagrange multipliers. The functional form of the mapping $\phi(x_i)$ does not need to be known since it is implicitly defined by the choice of kernel

$$K(x_i, x_j) = \langle \phi(x_i) \cdot \phi(x_j) \rangle \tag{3}$$

The vector w has the form:

$$w = \sum_{i=1}^{l} y_i \lambda_i \Phi(x_i)$$

and therefore

$$\begin{aligned} f(x) &= \sum_{i=1}^{l} y_i \lambda_i \langle \Phi(x_i) \cdot \Phi(x) \rangle + b \\ &= \sum_{i=1}^{l} y_i \lambda_i K(x_i, x) + b \end{aligned} \tag{4}$$

The points with $\lambda_i \neq 0$ were identified as support vectors since only those points determine the final decision result among all training points.

2.2. *SVM regression*

Let $\{(x_1, y_1), \ldots, (x_l, y_l)\} \subset \aleph \times R$ be a given training data, where $\aleph$ denotes the space of input points. The SVM regression is to map data points x_i into a high dimensional feature space via a nonlinear transform Φ and to regress them by a linear function $f(x) = w \cdot \Phi(x) + b$ in feature space. By introducing an ε-insensitive loss function $|\xi|_\varepsilon$

$$|\xi|_\varepsilon := \begin{cases} 0 & \text{if } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$

The task of SVM regression is therefore to

$$\begin{aligned} \underset{w,b,\xi_i,\xi_i^*}{\text{minimize}} \quad & \|w\|^2/2 + C \sum_{i}^{l} (\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - (w \cdot \Phi(x_i) + b) \leq \varepsilon - \xi_i \\ & (w \cdot \Phi(x_i) + b) - y_i \leq \varepsilon - \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \quad \forall i \end{aligned}$$

where the constant $C > 0$ determines the trade off between the flatness of f and the amount up to which deviations larger than ε are tolerated.[28] Its dual quadratic programming-regression problem is

$$\underset{\lambda_i, \lambda_i^*}{\text{maximize}} \begin{cases} -\frac{1}{2} \sum_{i,j=1}^{l} (\lambda_i - \lambda_i^*)(\lambda_j - \lambda_j^*) \\ \quad \times \langle \Phi(x_i) \cdot \Phi(x_j) \rangle \\ -\varepsilon \sum_{i=1}^{l} (\lambda_i + \lambda_i^*) + \sum_{i=1}^{l} y_i (\lambda_i - \lambda_i^*) \end{cases} \tag{6}$$

$$\text{subject to} \begin{cases} \sum_{i=1}^{l} (\lambda_i - \lambda_i^*) = 0 \\ \lambda_i, \lambda_i^* \in [0, C] \end{cases}$$

where λ_i and λ_i^* are the Lagrange multipliers. The vector w has the form: $w = \sum_{i=1}^{l} (\lambda_i - \lambda_i^*) \Phi(x_i)$ and therefore

$$\begin{aligned} f(x) &= \sum_{i=1}^{l} (\lambda_i - \lambda_i^*) \langle \Phi(x_i) \cdot \Phi(x) \rangle + b \\ &= \sum_{i=1}^{l} (\lambda_i - \lambda_i^*) K(x_i, x) + b \end{aligned} \tag{7}$$

The points with $(\lambda_i - \lambda_i^*) \neq 0$ were identified as support vectors since only those points determine the final decision result among all training points.

Some notes on the support vector learning approach

Throughout this paper, we use the Gaussian kernel function $K(x, y) = \exp(-q \|x - y\|^2)$. According to Eqs. (4) and (7) the final decision function of SVM has the following formulation

$$f(x) = \sum_{i=1}^{n} \alpha_i \exp(-q \|x - \bar{x}_i\|^2) + b \tag{8}$$

where $\bar{x}_i$ are support vectors chosen from the training points and $\alpha_i \in [-C, C]$ for $i = 1, \ldots, n$. The support vector machines can be represented as a RBF network architecture, as shown in Fig. 1, where each hidden unit in the RBF network architecture corresponds to one support vector in SVM.

The motivating idea behind the SVM network architecture is that decision function has the following two properties:

(1) According to Eq. (8) we observe that only those points corresponding to α_i that differ from zero will affect the final decision result. The value of α_i is obtained by solving the quadratic programming

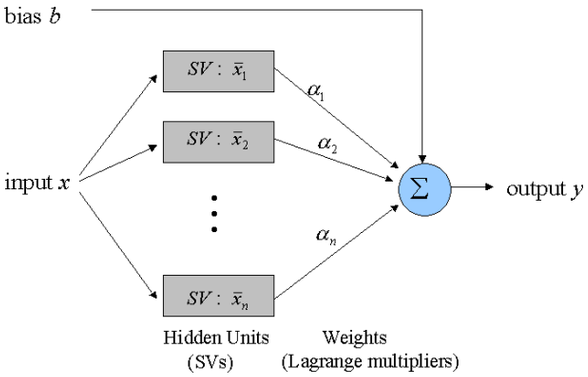


Fig. 1. The SVM network architecture representation.

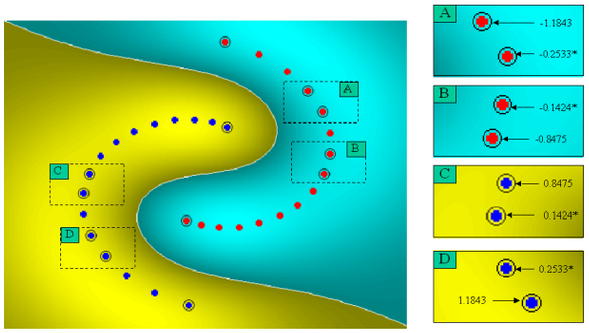


Fig. 2. An example of the occurrence of redundant support vectors in SVM classification model. The support vectors are marked with circles.

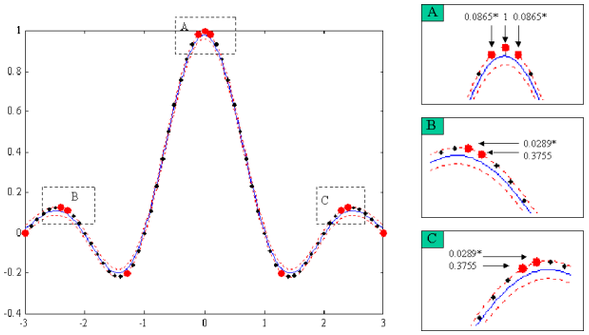


Fig. 3. An example of the occurrence of redundant support vectors in SVM regression model. The support vectors are represented as red dots. A larger epsilon value typically leads to less support vectors.

problem in Eqs. (2) and (6). Occasionally, we will obtain some points where α_i is close to zero. Clearly, those points will not significantly affect the final decision result, and we consider them as the redundant support vectors. However, we can not eliminate those points arbitrarily because this could lead to bad results. We propose a suitable mechanism to prune those redundant support vectors without significantly increasing the errors in Sec. 3.

(2) According to Eqs. (2) and (6) we observe that the learnable parameters during SVM learning procedure are α_i and b. Parameter $\bar{x}_i$ is the support vector coordinate, and it has to choose from training points. Parameters q, the Gaussian kernel width, and C, the regular constant, are predefined constant. If parameters $\bar{x}_i$, C, and q are learnable, i.e., the positions of support vectors can be located arbitrarily, the upper and lower bond of α_i can be different, and the width of Gaussian kernel corresponding to each support vector can be different, then this leads to a flexible formulation in the SVM approximation function. In other words, we can represent the final decision function with fewer support vectors. We illustrate the redundant support vectors in following example.

An example of redundant support vectors

Here we illustrate examples that redundant support vectors occur in the original SVM learning in sample classification and function regression cases. By solving the quadratic programming in Eqs. (2) and (6) we occasionally obtain some support vectors with α_i close to zero. As illustrated in Figs. 2 and 3 for SVM classification and regression model, respectively, the support vectors are identified with a red circle. We enlarge the detailed support vector surrounding picture with its value of α_i in the right side, and the redundant support vectors are marked with superscript "*". We observe an interesting phenomenon that the redundant support vectors usually do not occur alone. They usually follow significant support vectors (i.e., support vectors with α_i differ from zero significantly). This is because by the construction of SVM is not flexible enough, since the support vectors must be chosen by training points and the width of Gaussian kernel corresponding to each support vectors must be the same. Those redundant support vectors with α_i close to zero need to be eliminated appropriately. We will describe our eliminating approach in next section.

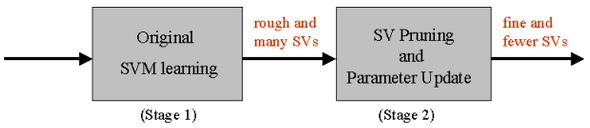


Fig. 4. Two-stage procedure for SVM training and redundant SV pruning

3. Support Vector Pruning and Model-Selecting Algorithm

In this work, we propose a two-stage learning algorithm to prune unimportant support vectors, as shown in Fig. 4 The first stage is the original SVM learning, and in the second stage we attempt to prune redundant support vectors. At the same time, we will also update parameters in order to have minimal increase in error.

Here we propose a methodology to prune redundant support vectors in a well-trained SVM network obtained after stage 1. We remove redundant support vectors by setting their parameter α_i equal to zero. Meanwhile, we adjust other parameters to minimize the increase in error after pruning them. At this stage, the learnable parameters are α_i, b, $\bar{x}_i$, and q_i, where q_i is the width of Gaussian kernel corresponding to ith support vector, and α_i are not bond in $[-C,C]$. Furthermore, we use the actual output obtained from the well trained SVM network in stage 1 as the desired training target in stage 2 in order to preserve the generalization ability of the original SVM learning.

There are two well-known pruning methods in traditional neural network pruning algorithms [11, 20]. One, called the penalty term method, adds terms to the objective function, thus rewarding the network for driving the weights of unnecessary elements to zero and, in effect, removing them during back-propagation learning. The other is the sensitivity calculation method. It estimates the sensitivity of the error function when an element is removed; the element with least effect can then be removed. We modify the above two pruning methods as the basis for support vector pruning algorithms.

3.1. *Method I (penalty term method)*

The first method modifies the error function so that the normal back-propagation algorithm effectively prunes the network by driving weights to zero during training. The weights may be removed when they decrease below a certain threshold. The modified error function provides an appropriate tradeoff between reliability of the training data and complexity of the model. This tradeoff can be realized by minimizing the total risk expressed as:

$$E = E_p + \lambda E_c \tag{9}$$

The first term E_p is the standard performance measure, which is typically defined as a mean-square error

$$E_p = \frac{1}{2}\left(f^{[k]} - d^{[k]}\right)^2 \tag{10}$$

where

$$f^{[k]} = \sum_{i=1}^{n} \alpha_i \exp(-q_i \left\| x^{[k]} - \bar{x}_i \right\|^2) + b \tag{11}$$

is the actual output, and $d^{[k]}$ is the desired output for current input $x^{[k]}$. E_c is the complexity penalty, which depends on the network (model) alone. Since a hidden unit (SV) with parameter α_i that is almost zero will not affect the final decision result, we can reasonably remove it. Here we adopt the complexity term proposed by Ref. 29, and apply it to our SVM pruning algorithm. The complexity penalty term is

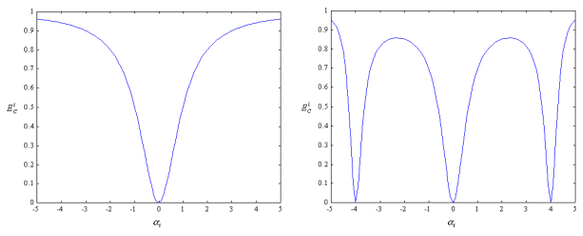
$$E_c = \sum_{i=1}^{n} \frac{(\alpha_i/\alpha_0)^2}{1 + (\alpha_i/\alpha_0)^2} \tag{12}$$

where α_0 is a predefined constant. For $|\alpha_i| \gg \alpha_0$, the cost of a weight approaches λ. For $|\alpha_i| \ll \alpha_0$, the cost is nearly 0. However, the complexity penalty term introduced above has an undesirable property: it forces the weight α_i of each hidden unit to approach zero. A better way is to drive the weights of unnecessary hidden units to zero but to retain the weights of significant hidden units, those with significant α_i values. Therefore, we propose an alternative design which defines the complexity penalty term as

$$E_c = \sum_{i=1}^{n} \frac{(\alpha_i(\alpha_i - C)(\alpha_i + C))^2}{b_0 + (\alpha_i(\alpha_i - C)(\alpha_i + C))^2} \tag{13}$$

where b_0 is a constant. It can be seen that for the cases of $\alpha_i \approx 0$, $\alpha_i \approx -C$, and $\alpha_i \approx C$, the cost of weight is nearly 0. Otherwise, the cost approaches λ. We design this complexity term because, according to the quadratic programming in Eqs. (2) and (6), the parameter α_i of each hidden unit is located in the interval $[-C, C]$. Figure 5 shows the difference between those two complexity penalty terms.

From Eq. (9), the λ value determines the tradeoff between reliability of the training data and complexity of the model, and it requires some tuning.



(a) (b)

Fig. 5. Two penalty strategies: (a) $E_c^i = \frac{\alpha_i^2}{1+\alpha_i^2}$; (b) $E_c^i = \frac{(\alpha_i(\alpha_i-C)(\alpha_i+C))^2}{b_0+(\alpha_i(\alpha_i-C)(\alpha_i+C))^2}$ with $C = 4$ and $b_0 = 100$.

When λ is infinitely large, all the weights will be driven to zero. When λ is zero, the back-propagation learning process is unconstrained, with the network being completely determined from the training points. Here we assume

$$\lambda = \lambda_0 \exp(-\rho \cdot n_p^2) \tag{14}$$

where λ_0 is the initial value of λ when the pruning algorithm starts, n_p represents the numbers of hidden units (i.e. SVs) that have been pruned, and ρ is a predefined constant. At the beginning of pruning redundant support vectors, the λ value set as a larger value in order to prune more hidden units. As the number of pruned units increases, the λ value should decrease in order to minimize the increase in error. The flowchart of the penalty term-based redundant support vector pruning method is shown in Fig. 6.

Now we use gradient descent method to update the parameters in the SVM. The basic updating rule for weight is

$$W(t+1) = W(t) + \eta\left(-\frac{\partial E}{\partial W}\right).$$

(A) The updating rule for parameter α_i:

$$\frac{\partial E}{\partial \alpha_i} = \frac{\partial E_p}{\partial \alpha_i} + \lambda \frac{\partial E_c}{\partial \alpha_i} \tag{15}$$

where

$$\begin{aligned}\frac{\partial E_p}{\partial \alpha_i} &= \frac{\partial E_p}{\partial f}\frac{\partial f}{\partial \alpha_i} \\ &= \left(f^{[k]} - d^{[k]}\right) \cdot \exp\left(-q_i \left\|x^{[k]} - \bar{x}_i\right\|^2\right)\end{aligned}$$

and

$$\frac{\partial E_c}{\partial \alpha_c} = \frac{2\alpha_i/(\alpha_0)^2}{\left(1 + (\alpha_i/\alpha_0)^2\right)^2}$$

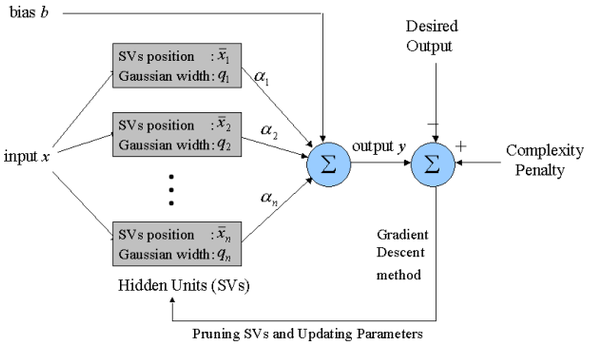


Fig. 6. Flowchart of redundant support vectors pruning using penalty term.

if we use Eq. (10) or.

$$\frac{\partial E_c}{\partial \alpha_i} = \frac{2b_0(\alpha_i(\alpha_i - C)(\alpha_i + C))(3\alpha_i^2 - C^2)}{(b_0 + (\alpha_i(\alpha_i - C)(\alpha_i + C))^2)^2}$$

if we use Eq. (13).

(B) The updating rule for parameter q_i:

$$\frac{\partial E}{\partial q_i} = \frac{\partial E_p}{\partial q_i} = \frac{\partial E_p}{\partial f}\frac{\partial f}{\partial q_i} \tag{16}$$

where

$$\frac{\partial E_p}{\partial f} = (f^{[k]} - d^{[k]})$$

and

$$\frac{\partial f}{\partial q_i} = \alpha_i \exp\left(-q_i \left\|x^{[k]} - \bar{x}_i\right\|^2\right) \cdot \left(-\left\|x^{[k]} - \bar{x}_i\right\|^2\right).$$

(C) The updating rule for parameter $\bar{x}_i$:

Let $\bar{x}_i = (\bar{x}_{i1}, \bar{x}_{i2}, \ldots, \bar{x}_{id})^T$ be the position of the ith support vector and $x^{[k]} = (x_1^{[k]}, x_2^{[k]}, \ldots, x_d^{[k]})^T$ be the position of the current input. Then the updating rule for parameter $\bar{x}_{ij}$ is

$$\frac{\partial E}{\partial \bar{x}_{ij}} = \frac{\partial E_p}{\partial \bar{x}_{ij}} = \frac{\partial E_p}{\partial f}\frac{\partial f}{\partial \bar{x}_{ij}} \tag{17}$$

where

$$\frac{\partial E_p}{\partial f} = (f^{[k]} - d^{[k]})$$

and

$$\frac{\partial f}{\partial x_{ij}} = \alpha_i \exp(-q_i \left\|x^{[k]} - \bar{x}_i\right\|^2) \cdot (2q_i) \cdot \left(x_j^{[k]} - \bar{x}_{ij}\right).$$

(D) The updating rule for parameter b is

$$\frac{\partial E}{\partial b} = \frac{\partial E_p}{\partial b} = \frac{\partial E_p}{\partial f}\frac{\partial f}{\partial b} \tag{18}$$

where

$$\frac{\partial E_p}{\partial f} = (f^{[k]} - d^{[k]})$$

and

$$\frac{\partial f}{\partial b} = 1.$$

3.2. *Method II (sensitivity calculation method)*

The basic idea of this approach is to use information on second-order derivatives of the error surface in order to make a trade-off between network complexity and training performance.[12,14] In a well trained SVM network, the functional Taylor series of the error with respect to the weights (or parameters) is

$$\delta E_p = \left(\frac{\partial E_p}{\partial w}\right)^T \cdot \delta w + \frac{1}{2}\delta w^T \cdot H \cdot \delta w + O\big(\|\delta w\|^3\big) \tag{19}$$

where $H \equiv \partial^2 E_p/\partial w^2$ is the Hessian matrix. For a network trained to minimum error, the first (linear) term vanishes. We ignore the third and all higher order terms.

The purpose of the sensitivity calculation method is to set to zero one of the weights that minimizes the increase in error given in Eq. (19). Eliminating w_q is expressed as $e_q^T \cdot \delta w + w^T = 0$ where e_q is the unit vector in weight space corresponding to weight w_q. The above task is therefore to

$$\min_q \left\{ \min_{\delta w} \left(\frac{1}{2}\delta w^T \cdot H \cdot \delta w \big| s.t.\, e_q^T \cdot \delta w + w_q = 0 \right) \right\}. \tag{20}$$

The Lagrangian can be formulated as:

$$L = \frac{1}{2}\delta w^T \cdot H \cdot \delta w + \beta(e_q^T \cdot \delta w + w_q)$$

where β is a Lagrangian multiplier. Taking the derivative of L with respect to δw and setting to zero, the "optimal weight change" and the "resulting change in error" can be obtained as:

$$\delta w = -\frac{w_q}{\left[H^{-1}\right]_{qq}} H^{-1} \cdot e_q \tag{21}$$

$$L_q = \frac{1}{2}\frac{w_q^2}{\left[H^{-1}\right]_{qq}} \tag{22}$$

where L_q is the "saliency" of weight q – the increase in error when the weight q was pruned.[12]

Now we apply this scheme to our pruning algorithm for redundant support vectors. The weight vector of the SVM network is $w = [\Phi, \Gamma, \Xi, b]^T$ where $\Phi = [\alpha_1, \ldots, \alpha_n]$, $\Gamma = [q_1, \ldots, q_n]$, and $\Xi = [\bar{x}_1, \ldots, \bar{x}_n]$. We denote L_q^Φ as the increase in error when the q-th parameter of Φ is pruned. The second pruning algorithm for redundant support vectors is summarized as follows:

Pruning Algorithm for Redundant Support Vectors using Sensitivity Calculation.

Step 1. Train an SVM network
Step 2. Compute H^{-1}
Step 3. Find the q-th SV that gives the smallest "saliency" L_q^Φ
Step 4. IF the increased error after adding L_q^Φ is below the tolerance threshold, THEN proceed to Step 5; [the q-th SV should be deleted] ELSE stop. [no more SVs can be deleted]
Step 5. Update all SVs using Eq. (21). Go to Step 2.

Since we start with a well trained SVM network, the computation of Hessian matrix and its inverse can simplify as[12]

$$H \cong \frac{1}{P}\sum_{k=1}^{P}\left(\frac{\partial f^{[k]}}{\partial w}\right)\cdot\left(\frac{\partial f^{[k]}}{\partial w}\right)^T \tag{23}$$

and the inverse of the Hessian matrix can be computed as

$$\begin{aligned} H^{-1} &= H_P^{-1} \\ &= H_{P-1}^{-1} + \frac{H_{P-1}^{-1}\cdot\left(\frac{\partial f^{[P-1]}}{\partial w}\right)\cdot\left(\frac{\partial f^{[P-1]}}{\partial w}\right)^T \cdot H_{P-1}^{-1}}{P + \left(\frac{\partial f^{[P-1]}}{\partial w}\right)^T \cdot H_{P-1}^{-1}\cdot\left(\frac{\partial f^{[P-1]}}{\partial w}\right)} \end{aligned} \tag{24}$$

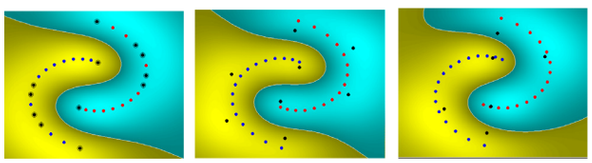
where $H_0^{-1} = (\eta I)^{-1}$ and η $(10^{-8} < \eta < 10^{-4})$ is a small constant in order to make H^{-1} meaningful.

4. Experiments

In this section, we present examples to investigate the utility of the proposed redundant support vector pruning algorithm, demonstrating that it can offer several advantages in different scenarios. First we present the synthesis examples and benchmark dataset to illustrate different pruning algorithms. We then apply support vector pruning algorithm to the face detection problem as the second experiment.

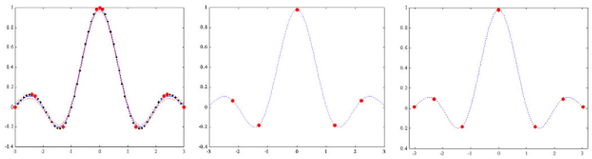
4.1. *Synthesis experiment*

In this experiment we apply the proposed redundant support vectors pruning algorithm to original SVM learning for both sample classification and function regression examples presented in Sec. 2. In the classification case, we use a small two-spiral data set where the parameters are set to be $C = 10$ and $q = 1.65$. In the function regression case, we regress the function $sinc(x)$ where the parameters are set to be $C = 10$, $q = 2$, and $\varepsilon = 0.02$. In the classification experiment 12 SVs were obtained by original support vector learning. The penalty term support vector pruning algorithm eliminated 4 redundant SVs while the sensitivity calculation support vector pruning algorithm eliminated 6 redundant SVs. The locations of remaining significant support vectors after pruning algorithm using penalty term and sensitivity calculation methods pruning algorithms are illustrated in Figs. 7(b) and (c), respectively. In the function regression experiment, 11 SVs were obtained by original support vector learning. The penalty term support vector pruning algorithm eliminated 6 redundant SVs while the sensitivity calculation support vector pruning algorithm eliminated 4 redundant SVs. The pruning results using penalty term and sensitivity calculation methods are illustrated in Figs. 8(b) and (c), respectively. In this experiment we eliminate about 30%∼50% unnecessary support vectors in the proposed pruning algorithms. In the function regression experiment, the root mean square-error (RMSE) in original support vector learning is 0.014201. The RMSEs obtained from both the penalty term and sensitivity calculation pruning method are 0.014233 and 0.014254, respectively. The increment in RMSE is small than 0.00005, and it is small enough to ignore.



(a) (b) (c)

Fig. 7. Classification experiment for two winded spiral where (a) the original SVM for classification. The redundant SVs pruning using: (b) penalty term method and (c) sensitive calculation method. Each support vectors was represented as a black dot.



(a) (b) (c)

Fig. 8. Regression experiment for *sinc*(x) function where (a) the original SVM regression result. The redundant SVs pruning results using: (b) penalty term method and (c) sensitive calculation method. Each support vectors was represented as a red dot.

4.2. *Benchmark experiment*

In this section we apply the proposed redundant support vectors pruning algorithm on a set of neural network benchmark problems, PROBEN1.[19] From PROBEN1 we choose the following datasets: cancer, card, diabetes, and heart. The information of collected datasets is given in Table 1.

For original SVM training, the parameters are set to be $C = 1$, $q = 0.7$, and $\varepsilon = 0.01$. The validation performance is measured by training 70% of the training set and testing the other 30%. Table 2 shows the numbers of SVs, training/testing accuracy

Table 1. Quick overview of the dataset.

Dataset	#in	#out	#example	Description
Cancer	9	2c	699	Wisconsin breast cancer diagnosis problem from UCI machine learning database
Card	51	2c	690	Credit card approval problem from UCI machine learning database
Diabetes	8	2c	768	Diabetes diagnosis problem from UCI machine learning database
Heart	35	2c	920	Heart-disease diagnosis problem from UCI machine learning database
Heartc	35	2c	303	The same as Heart
Hearta	35	1a	920	The same as Heart
Heartac	35	1a	303	The same as Heart

c = class output(0/1), a = analog output($0\cdots1$).

Table 2. Experiment result.

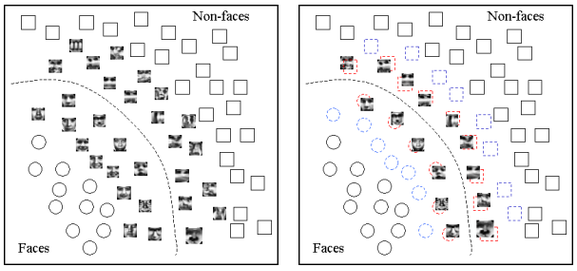
Dataset	Original SVM			After pruning					
	#SVs	Acc. (train/test)		Method I — Penalty term method			Method II — Sensitivity calculation method		
				#SVs	Acc(train/test)		#SVs	Acc(train/test)	
Classification (*Acc. Ratio*)									
Cancer	64	96.99	97.85	38	96.57	97.85	43	96.57	**98.28**
Card	315	93.99	85.02	187	92.13	**85.99**	191	93.17	85.51
Diabetes	298	79.69	76.95	103	79.49	**76.95**	121	78.28	76.78
Heart	441	91.77	81.16	239	91.15	81.16	262	90.29	**81.88**
Heartc	149	92.45	80.22	76	92.45	80.22	78	91.04	**82.42**
Regression (*MSE*)									
Hearta	391	0.0148	**0.0529**	231	0.0149	0.0531	235	0.0162	0.0540
Heartac	129	0.0124	**0.0559**	74	0.0128	0.0562	86	0.0136	0.057

performance in original SVM learning, penalty term method and sensitivity calculation method, respectively. In summary, we eliminate about 35%–65% unnecessary support vectors in the proposed pruning algorithms with a slight increment in training errors. For penalty term method, the validation performance is equal to or better than original SVM in cancer, card, diabetes, heart and heartc; while for sensitivity calculation method, the validation performance is equal to or better than original SVM in cancer, card, heart and heartc.

4.3. *Face detection experiment*

We now apply proposed pruning algorithms to a real world application — face image detection. Speed is an important factor in face detection problem, and eliminating redundant SVs will definitely increase the speed of face detection. We utilize the MIT CBCL face database as training and test data (http://www.ai.mit.edu/projects/cbcl/). The training set was generated by Ref. 24 and the non-faces training set was generated by Ref. 13. There are 2,429 faces and 4,548 non-faces in the final training set. The test set we used is a subset of the CMU Test Set 1,[21] relevant information about how the subset was chosen can be found in Ref. 13. There are 472 faces and 23,573 non-faces in the test set. This is the same data set that was used in Ref. 1. Each image in the database is of size 19 by 19, which either contains a human face or not.

In original support vector learning we extract 1,464 support vectors from the training data where the parameters are set to be $C = 10$ and $q = 0.02$. The percentage ratio of SVs is about 20.9% among training data. The classification accuracy is 100% in training set and 97.812% in test set, respectively in original SVM approach. Using penalty term method we eliminated 709 redundant SVs while the classification accuracy is 100% in training set and 97.812% in test set, respectively. Using sensitivity calculation method we eliminated 676 redundant SVs while the classification accuracy is 100% in training set and 97.775% in test set, respectively. Since about 50% SVs were pruned in both pruning algorithms, the final face detection performs almost two times faster than the original one. Besides, the SVs in face detection have another geometrical interpretation as shown in Fig. 9(a). Those SVs are images (contained both faces and non-faces) that selected from training set to support the final decision function, which are called the support faces.[16] After pruning redundant SVs, the image pixels in remaining support faces change to new values, and this makes image deviation from original location. We denoted the remaining SVs as extended support faces, and these extended support faces do not belong to the original training images, as shown in Fig. 9(b). Figure 10 shows some of face detection results in the CMU test set.



(a) (b)

Fig. 9. Geometrical interpretation of how the SVM model separates the face and non-face classes. (a) The support faces are obtained after SVM training. (b) After pruning, the square and circular symbols drawn with blue dot line denote the redundant support faces that have been eliminated. The remaining support faces deviate from the original support faces while the square and circular symbols drawn with red dot line denote the original support faces.

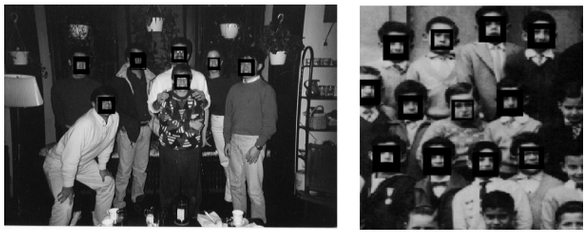


Fig. 10. Examples of face detection results in CMU test images.

5. Discussion

In this paper we prune redundant support vectors in the second-stage training. Hence, the overall training time is longer than original SVM training. However, eliminate redundant support vectors can reduce the testing time and storage requirement. And it is more important to reduce testing time than training time in real applications. Moreover, pruning can improve generalization ability. The computational cost in the second-stage training is worthy. Beside, the model-selection task is employed in the second-stage pruning, and it is faster than training a SVM several times with different model-parameters. In comparison with the two redundant support vectors pruning methodologies, the penalty term method is much slower than sensitivity calculation method because the learning structure of gradient descent manner. It also needs some prior knowledge to determine the values of some predefined constants (e.g., the learning rate and the tradeoff parameter). Furthermore, one benefit of sensitivity calculation method is that it estimates the sensitivity of the error function after removing support vectors, and this can efficiently control the increase in error value so that the error is always below a tolerance threshold. However, the major drawback in sensitivity calculation method is the demand for memory. The Hessian matrix is of size n^2 where n is the numbers of weights that need to learn. A decomposition method utilized in the face detection experiment is that it only takes into account a subset of weights at each iteration. How to improve the speed of pruning algorithm when the number of support vectors is huge will be our further work.

Acknowledgments

This work was partially supported by National Science Council Taiwan under grant. NSC-94-2218-E-151-016.

References

1. M. Alvira and R. Rifkin, An empirical comparison of SNoW and SVMs for face detection, Center for Biological and Computational Learning, MIT, *A.I. memo*, Cambridge, MA, no. 2001-004 (2001).
2. V. Blanz, B. Schölkopf, H. Bulthoff, C. J. C. Burges, V. N. Vapnik and T. Vetter, Comparison of view-based object recognition algorithms using realistic 3D models, in *Proc of ICCANN'96*, LNCS, Vol. 1112 (1996), pp. 251–256.
3. C. J. C. Burges, A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery* **2**(2) (1998) 955–974.
4. C. J. C. Burges and B. Scholkopf, Improving the accuracy and speed of support vector learning machines, in M. Mozer, M. Jordan and T. Petsche (eds.), *Advances in Neural Information Processing Systems*, Vol. 9 (MIT Press, Cambridge, MA, 1997), pp. 375–381.
5. C. J. C. Burges, Simplified support vector decision rules, in *Proc. 15th Inter. Conf. on Machine Learning* (San Mateo, CA, 1996, Morgan Kaufmann), pp. 71–77.
6. J.-H. Chiang and P.-Y. Hao, A new kernel-based fuzzy clustering approach: Support vector clustering with cell growing, *IEEE Trans. on Fuzzy Systems* **11**(4) (2003) 518–527.
7. J.-H. Chiang and P.-Y. Hao, Support vector learning mechanism for fuzzy rule-based modeling: A new approach, *IEEE Trans. on Fuzzy Systems* **12**(1) (2004) 1–12.
8. C. Cortes and V. N. Vapnik, Support vector network, *Machine Learning* **20** (1995) 1–25.

9. K. Duan, S. S. Keerthi and A. N. Poo, Evaluation of simple performance measures for turning SVM hyperparameters, *Neurocomputing* **51** (2003) 41–59.
10. T. Graepel, R. Herbrich, B. Scholkopf, A. J. Smola, P. L. Bartlett, K.-R. Muiller, K. Obermayer and R. C. Williamson, Classification on proximity data with LP-machines, in *Proc. of 9th Int. Conf. on Artificial Neural Networks* (1999), pp. 304–309.
11. S. Haykin, *Neural Networks — A Comprehensive Foundation* (Prentice Hall, New Jersey, 2nd edn., 1999).
12. B. Hassibi, D. G. Stork and G. J. Wolff, Optimal brain surgeon and general network pruning, *IEEE International Conference on Neural Networks* **1** (San Francisco) (1992), pp. 293–299.
13. B. Heisele, T. Poggio and M. Pontil, Face detection in still gray images, A.I. Memo/C.B.C.L Paper 1687, Center for Biological and Computational Learning, MIT, Cambridge, MA (2000).
14. Y. LeCun, J. S. Denker and S. A. Solla, Optimal brain damage, in *Advances in Neural Information Processing Systems* **2** (San Mateo, CA: Morgan Kaufmann) (1990), pp. 598–605.
15. S. Mika, G. Ratsch and K.-R. Muller, A mathematical programming approach to the Kernel Fisher algorithm, in T. K. Leen, T. G. Dietterich and V. Tresp (eds.), *Advances in Neural Information Processing Systems*, Vol. 13 (MIT Press, 2001), pp. 591–597.
16. B. Moghaddam and M.-H. Yang, Learning gender with support faces, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **24**(5) (2002) 707–711
17. E. Osuna, R. Freund and F. Girosi, Training support vector machines: An application to face detection, in *Proceedings, CVPR* (IEEE Computer Society Press, 1997), pp. 130–136.
18. E. Osuna and F. Girosi, Reducing run-time complexity in SVMs, in *Proceedings of the 14th International Conf. on Pattern Recognition*, (Brisbane, Australia) (1998) [In B. Scholkopf, C. Burges and A. Smola (eds.), *Advances in Kernel Methods — Support Vector Learning*, pp. 271–283, 1999. Cambridge, MA: MIT Press].
19. L. Prechelt, PROBEN 1 — A set of neural network benchmark problems and benchmarking rules. Technical Report Nr. 21/94, Fakultat fur Informatik, Universitat Karlsruhe, D-76128, Karlsruhe, Germany (1994).
20. R. Reed, Pruning algorithms — A survey, *IEEE Trans. Neural Networks*, Vol. 4 (1993), pp. 740–747.
21. H. A. Rowley, S. Baluja and T. Kanade, Neural network-based face detection, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **20**(1) (1998) 23–38.
22. D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1 (Cambridge, MA: MIT Press, 1986).
23. B. Scholkpof, A. J. Smola, R. C. Williamson and P. L. Bartlett, New support vector algorithms, *Neural Computation* **12** (2000) 1083–1121.
24. K.-K. Sung, *Learning and Example Selection for Object and Pattern Recognition*, Ph.D. thesis, MIT, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Cambridge, MA (1996).
25. J. A. K. Suykens, J. De Brabanter, L. Lukas and J. Vandewalle, Weighted least squares support vector machines: Robustness and sparse approximation, *Neurocomputing, Special Issue on Fundamental and Information Processing Aspects of Neurocomouting*, **48**(1) (2002a) 85–105.
26. J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor and J. Vandewalle, *Least Squares Support Vector Machines*, World Scientific, Singapore (2002b) (ISBN 981-238-151-1).
27. V. N. Vapnik, *Estimation of Dependencies Based on Empirical Data* (Springer-Verlag, New York, 1982).
28. V. N. Vapnik, *The Nature of Statistical Learning Theory* (Springer-Verlag, New York, 1995).
29. A. S. Weigend, D. E. Rumelhart and B. A. Huberman, Generalization by weight-elimination with approach to forecasting, *Advances in Neural Information Processing Systems*, Vol. 3 (San Mateo, CA: Morgan Kaufmann, 1991), pp. 875–882.