Automatic Model Selection for Support Vector Machines

Jen-Hao Lee and Chih-Jen Lin

Department of Computer Science and

Information Engineering

National Taiwan University

Taipei 106, Taiwan

Abstract

Automatic model selection is an important issue to make support vector machines (SVM) practically useful. Most existing approaches use the leave-one-out (loo) related estimators. As finding the loo rate is time consuming, researchers exploit different techniques to approximate it. Therefore, they develop methods mainly from a machine learning point of view while numerical experiments are usually not seriously treated. Up to now, most existing SVM software cannot do effective automatic model selections. We think that if the calculation of loo is a difficult numerical issue, more investigation from a numerical point of view should be conducted. In this paper, we study the relation between the loo rate and the stopping criteria of the decomposition method for SVM. For each particular model, the computation of loo involves a sequence of standard SVM problems. We show that using a very loose stopping criteria for the decomposition method, the best model can still be obtained. Such an observation leads us to be able to design simple and practical automatic model selection software.

1 Introduction

Given training vectors $x_i \in R^n, i = 1, \ldots, l$ in two classes, and a vector $y \in R^n$ such that $y_i \in \{1, -1\}$, the support vector machine requires the solution of the following quadratic programming problem

$$(D) \qquad \begin{aligned} &\min \frac{1}{2}\alpha^T Q \alpha - e^T \alpha \\ &y^T \alpha = 0, 0 \le \alpha_i \le C, i = 1, \ldots, l, \end{aligned} \tag{1}$$

where e is the vector of all ones, C is the upper bound of all variables, Q is an l by l positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel. Some popular kernels are, for example, $e^{-\gamma\|x_i - x_j\|^2}$, $(x_i^T x_j/\gamma + \delta)^d$, etc.

A practical difficulty of using SVM is the selection of parameters such as C and kernel parameters γ and d. Users would like to choose parameters which leads to the best generalization performance. As the estimation of the generalization error usually takes excessive computational time, up to now most SVM software cannot do effective model selections.

The most used estimator is the leave-one-out cross validation, which is an almost unbiased estimator. Its procedure is as follows: For $j = 1, \dots, l$, we solve the following problem:

$$(D_j) \qquad \begin{aligned} &\min \frac{1}{2}(\alpha^j)^T Q^j \alpha^j - \sum_{i=1}^{l-1} \alpha_i^j \\ &(y^j)^T \alpha^j = 0, \\ &0 \le \alpha_i^j \le C, i = 1, \dots, l-1, \end{aligned}$$

where α^j is a vector of length $l-1$, Q^j is a sub-matrix of Q by removing the jth column and row and y^j is a sub-vector of y by removing y_j.

That is, the jth training data is removed and the remaining data are trained. We then test x_j using

$$\operatorname{sign}(\sum_{i=1}^{l-1} \alpha_i^j y_i^j K(x_i, x_j) + b^j), \tag{2}$$

where b^j is the Lagrange multiplier associated to the constraint $\sum_{i=1}^{l-1} y_i \alpha_i^j = 0$ of (D_j). The loo rate is

$$L(x_1, y_1, \dots, x_l, y_l) \equiv 1 - \sum_{j=1}^{l} \frac{|\operatorname{sign}(\sum_{i=1}^{l-1} \alpha_i^j y_i^j K(x_i, x_j) + b^j) - y_j|}{2l}.$$

It has been known that if α is an optimal solution of (D) with $\alpha_j = 0$, by removing the jth component, α^j is already a solution of (D_j). Since $\alpha_j = 0$ implies the value of (2) to be nonnegative, we do not have to solve all $(D_1), \dots, (D_l)$ but only those with $\alpha_j > 0$. Therefore, we must retrain the data #SV times. This is very time consuming so up to now the only available strategy is to upper bound or approximate this estimator with an easily computable quantity. Some examples are (Chapelle and Vapnik 2000; Chapelle et al. 2000; Jaakkola and Haussler 1999; Joachims 2000; Vapnik and Chapelle 2000), and a short survey in (Campbell 2000, Section 2.6). Most of these approaches focus on studying theoretical bounds of loo so numerical experiments and comparisons might not be seriously treated. For example, some papers compare their approaches with direct calculation of loo rates without even mentioning their settings such as stopping criteria, initial solutions, cache size, etc.

We think that if the calculation of loo is a difficult numerical issue, more investigation from a numerical point of view should be conducted. Currently a major method to solve (D) and $(D_j), j = 1, \dots, l$ is the decomposition method. When people say that the calculation of loo is difficult, it is under the assumption that loo rate is exactly evaluated. However, practical issues such as the stopping criteria affect the precision of loo rates as well as the computational time. In addition, as the decomposition method does not use the full first and second order information, one of its main drawbacks is the slow local convergence. Thus different stopping criteria may affect a lot on the number of iterations and the computational time. We wonder that if a loose stopping criterion is used, probably the obtained loo rates are already good enough to find the best parameters. Then the computational time can be dramatically reduced.

This paper is organized as follows. Section 2 describes the decomposition method for SVM. The main experiments on finding loo rates with different stopping tolerances are in Section 3. Results conclude that a loose stopping criterion is enough to find parameters with nearly the best loo rate. In Section 4 we discuss the implementation of caches between different models. Then in Section 5 we exploit some methods to further reduce the search space. We make discussions and future directions in Section 6.

We implement the proposed method as the software looms (leave-one-out model selection) which is available at the authors' homepage.[‡]

2 Decomposition Methods, Stopping Criteria, and the Implementation for loo

The difficulty of solving (1) is the density of Q because Q_{ij} is in general not zero. Thus traditional optimization algorithms such as Newton, Quasi Newton, etc., cannot be directly applied due to memory requirements. Several authors (for example, Osuna et al. (1997), Joachims (1998), Platt (1998), and Saunders et al. (1998)) have proposed decomposition methods to conquer this difficulty. Basically they separate the index $\{1,\dots,l\}$ of the training set to two sets B and N, where B is the working set and $N=\{1,\dots,l\}\backslash B$. If we denote α_B and α_N as vectors containing corresponding elements, the objective value is equal to $\frac{1}{2}\alpha_B^T Q_{BB}\alpha_B-(e_B-Q_{BN}\alpha_N)^T\alpha_B+\frac{1}{2}\alpha_N^T Q_{NN}\alpha_N-e_N^T\alpha_N$. At each iteration, α_N

[‡]looms is available at http://www.csie.ntu.edu.tw/~cjlin/looms

is fixed and the following sub-problem with the variable α_B is solved:

$$
\begin{aligned}
\min \quad & \frac{1}{2}\alpha_B^T Q_{BB}\alpha_B - (e_B - Q_{BN}\alpha_N)^T\alpha_B \\
& 0 \le (\alpha_B)_i \le C, i = 1, \dots, q, \\
& y_B^T\alpha_B = -y_N^T\alpha_N,
\end{aligned} \tag{3}
$$

where $\begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$ is a permutation of the matrix Q and q is the size of B. The process continues until an approximated solution is obtained. For implementation usually a special storage using the idea of a cache is used to store recently used elements of Q. Hence the computational cost of later iterations can be reduced.

In this paper we consider the variant of (4) used by the software BSVM (Hsu and Lin 1999):

$$
(\bar{D}) \qquad
\begin{aligned}
& \min \frac{1}{2}\alpha^T(Q + yy^T)\alpha - e^T\alpha \\
& 0 \le \alpha_i \le C, i = 1, \dots, l,
\end{aligned} \tag{4}
$$

This formulation was first used by (Mangasarian and Musicant 1999) and (Friess et al. 1998). Thus (D_j) becomes

$$
(\bar{D}_j) \qquad
\begin{aligned}
& \min \frac{1}{2}(\alpha^j)^T(Q^j + y^j(y^j)^T)\alpha^j - \sum_{i=1}^{l-1}\alpha_i^j \\
& 0 \le \alpha_i^j \le C, i = 1, \dots, l-1.
\end{aligned}
$$

The stopping criteria of BSVM (for $(\bar{D})$) is

$$
\begin{aligned}
-\epsilon \le ((Q + yy^T)\alpha)_i - 1 \le \epsilon, &\quad \text{if } 0 < \alpha_i < C, \\
((Q + yy^T)\alpha)_i - 1 \ge -\epsilon, &\quad \text{if } \alpha_i = 0, \\
((Q + yy^T)\alpha)_i - 1 \le \epsilon, &\quad \text{if } \alpha_i = C.
\end{aligned} \tag{5}
$$

In the next section we will compare results using different ϵ (0.1 and 0.001) for finding loo (i.e. solving all $(\bar{D}_j)$, where x_j is a support vector of $(\bar{D})$). We call this part the "loo" phase. Before this calculation, to correctly identify support vectors of $(\bar{D})$, $\epsilon = 0.001$ is used for both cases. We name the procedure for solving $(\bar{D})$ the "training" phase.

If the decomposition methods is used to obtain the loo rate, some methods can be immediately applied to save the computational time:

1. As each $(\bar{D}_j)$ is a minor modification of (D), we can expect that $\alpha_i^j, i = 1, \ldots, l-1$ may be quite close to $\alpha_i, i = 1, \ldots, l, i \neq j$. Thus if $\alpha_i, i = 1, \ldots, l, i \neq j$ is used as an initial solution, iterations of the decomposition methods can be saved. Note that $\alpha_i, i = 1, \ldots, l, i \neq j$ is already a feasible solution to $(\bar{D}_j)$ so it can be used as an initial solution. This is not the case for (D) and (D_j) due to the linear constraint $(y^j)^T \alpha^j = 0$. We have to slightly modify $\alpha_i, i = 1, \ldots, l, i \neq j$ in order to have a feasible solution of (D_j) (for example, (Joachims 2000)). Experiments on this strategy for finding loo are discussed in (DeCoste and Wagstaff 2000).

2. After $(\bar{D})$ is solved, some elements of Q have been calculated and stored in computer memory. If they can be used for solving $(\bar{D}_j)$, the number of kernel evaluations can be reduced.

In the rest of this section we discuss how to easily implement these methods. Here we give an example of using BSVM. For other software, the concepts are similar. If we consider $(\bar{D}_j)$ as the following problem:

$$\begin{aligned} &\min \frac{1}{2}\alpha^T (Q + yy^T)\alpha - e^T\alpha \\ &0 \leq \alpha_i \leq C, i = 1, \ldots, l, \\ &\alpha_j = 0, \end{aligned} \tag{6}$$

then we can still use the same programs of BSVM with very few modifications to solve $(\bar{D}_j), j = 1, \ldots, l$. These few changes are used to keep $\alpha_j = 0$:

(a) The decomposition method needs the gradient $(Q + yy^T)\alpha$ of the objective function in the beginning. After solving $(\bar{D})$, if α is an optimal solution, $(Q + yy^T)\alpha$ is also returned. Thus the gradient must be adjusted before solving each $(\bar{D}_j)$ as α_j is assigned to be zero.

(b) We modify the working set selection so that the index j is never chosen.

(c) For the stopping criteria (5), the jth index cannot be considered.

The implementation of (a) to (c) is very simple. For example, the working set selection involves some loops with indices from 1 to l. In each loop we just need an additional statement to skip the index j. Because the same programs are used, the contents of the cache after solving $(\bar{D})$ is automatically passed to the first (D_j). Similarly, once a $(\bar{D}_j)$ is solved, its cache will be used for for the next $(\bar{D}_{\bar{j}})$.

However, interestingly we can use an even simpler implementation in which (b) and (c) are automatically taken care of. Note that if q is the size of the working set, BSVM selects $q/2$ elements of the working set by considering free variables and the other $q/2$ elements by solving the following problem:

$$
\begin{aligned}
\min \quad & \nabla f(\alpha^k)^T d \\
& -1 \le d_i \le 1, i = 1, \dots, l, \\
& d_i \ge 0, \text{ if } \alpha_i = 0, \qquad d_i \le 0, \text{ if } \alpha_i = C, \\
& |\{d_i \mid d_i \ne 0\}| \le q/2,
\end{aligned} \tag{7}
$$

where $f(\alpha) \equiv 1/2\alpha^T(Q + yy^T)\alpha - e^T\alpha$ and $|\{d_i \mid d_i \ne 0\}|$ means the number of components of d which are not zero. Problem (7) follows the working set selection of the software SVM^{light} (Joachims 1998). Many existing implementation use a similar selection (e.g. (Keerthi et al. 1999; Rüping 2000; Collobert and Bengio 2000)).

While solving $(\bar{D})$, there is an array to store the gradient $\nabla f(\alpha) = (Q + yy^T)\alpha - e$. Since $\alpha_j = 0$ from the beginning, if we set the jth element of this gradient array to be ∞, j will never be chosen while solving (7). Otherwise, if $d_j > 0$, the optimal objective value satisfies

$$\nabla f(\alpha)^T d = \infty \cdot d_j + \text{ a finite number } > 0.$$

However, zero is an feasible point of (7) so the optimal objective value of (7) is less than or equal to zero. Therefore, we do not even need to modify the working set selection procedure of BSVM. In addition, $\infty \ge -\epsilon$ so (5) always holds and the code for stopping criterion can remain the same.

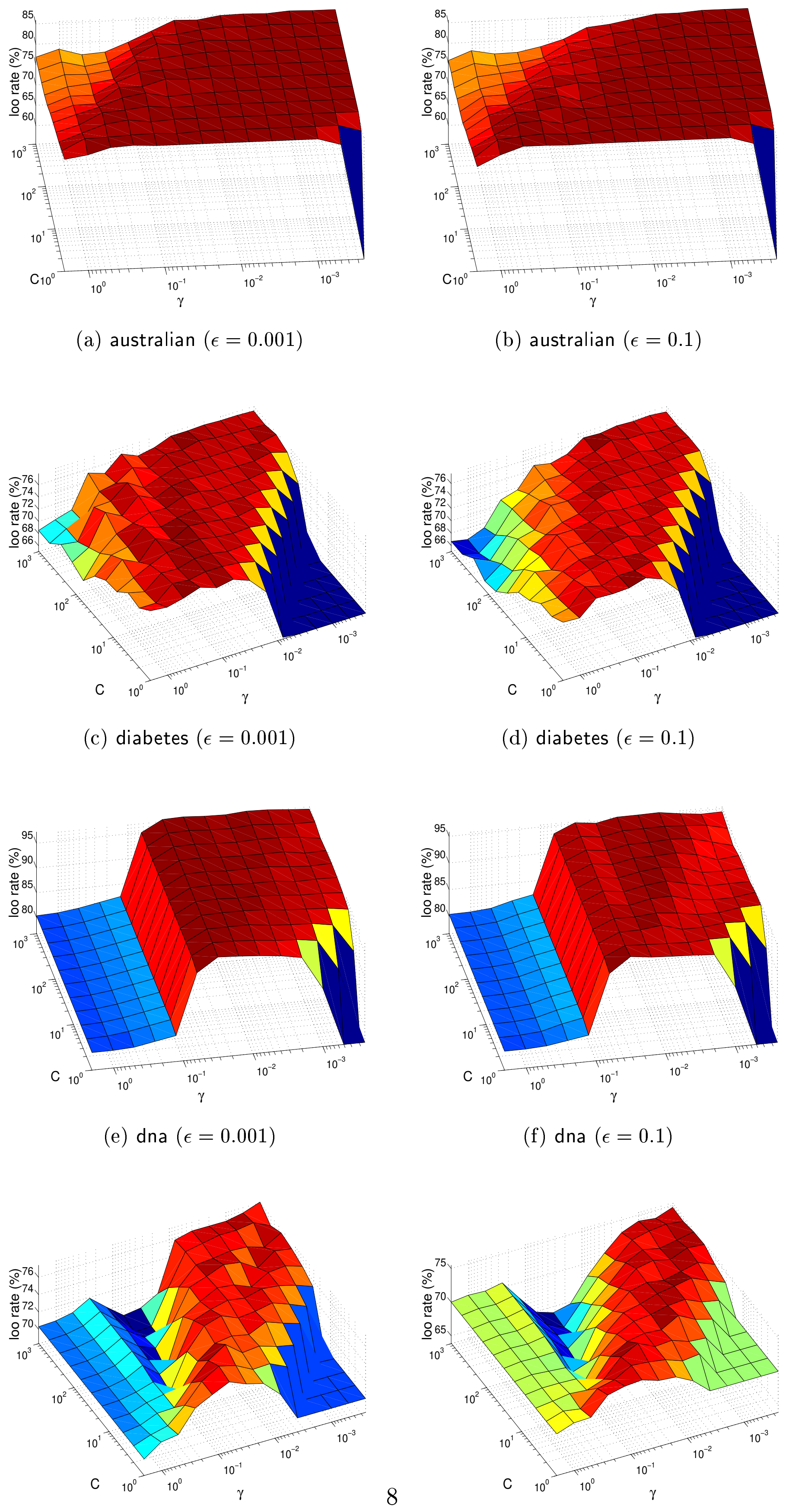
Therefore, the only work is to assign ∞ to the jth component of the gradient array. Practically we can use the largest number or the infinity defined under the computer's floating point systems. Note that assigning the jth component as ∞ can be included in (a).

The above arguments are also true for (D) and (D_j). Therefore, if a similar selection like (7) is used, allowing users to specify an initial solution and the initial gradient is the only essentially requirement an SVM software for (D) should fulfill so that it can directly find the loo.

3 Experiments

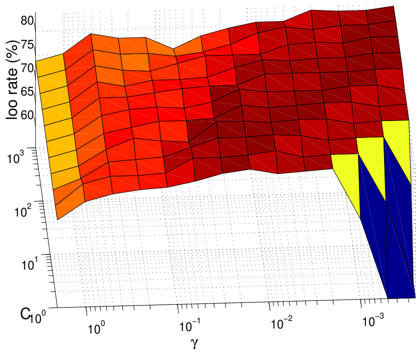
We consider eight problems: australian, diabetes, dna, german, heart, satimage, segment and vehicle. They are all from the Statlog collection (Michie et al. 1994). The number of training data (l) and the number of attributes (n) are shown in the second and third columns of Table 3.2, respectively. Data are scaled to $[-1, 1]$ except dna which is already in a binary representation. Some of these problems have more than 2 classes so we treat all data not in the first class as in the second class. For each problem, we use the RBF kernel $K(x_i, x_j) = e^{-\gamma\|x_i - x_j\|^2}$ with $C = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]$ and γ= [0.00025, 0.0005, 0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128, 0.256, 0.512, 1.024, 2.048]. Therefore, for each problem 154 parameter sets are tried. The computational experiments for this section were done on a Pentium III-800 with 128MB RAM running FreeBSD. The gcc compiler and BSVM version 1.0 with default options are used. The loo rate using different parameters are presented in three dimensional surfaces (Figures 3.1 and 3.2) where x, y, and z axes are C, γ, and loo rates, respectively. For each row of figures, the left one uses the stopping tolerance $\epsilon = 0.001$ while the right part uses $\epsilon = 0.1$. In other words, if we assume figures using $\epsilon = 0.001$ are already the theoretical surfaces, we are interested in knowing whether using $\epsilon = 0.1$, surfaces have similar shapes as those of using $\epsilon = 0.001$. If this is the case, practically using $\epsilon = 0.1$ may be enough to find the best parameters. We consider results using $\epsilon = 0.001$ as a standard for comparison because 0.001 is the default stopping tolerance of many SVM software (e.g. SVM^{light} (Joachims 1998), BSVM (Hsu and Lin 1999), etc). Users basically consider their output values as the real solution.

It can be clearly seen that figures on the right-hand side have very similar shapes as those on the left. To further confirm that $\epsilon = 0.1$ is good enough, we make more observations in Table 3.1. The second column shows the best loo rate using $\epsilon = 0.001$ which we consider as the theoretical value. Then in the third column, we present the loo rate using $\epsilon = 0.001$ and the best parameters from the relaxed case ($\epsilon = 0.1$). For example, for problem vehicle, the best loo rate happens at $C = 256$ and $\gamma = 0.128$ when using $\epsilon = 0.1$. As practically they will be the parameters for future use, we would like to check if their theoretical loo value is close to the best generalization performance. This is done by calculating their loo rate with $\epsilon = 0.001$ (i.e. 87.12) and then comparing with the best loo rate using $\epsilon = 0.001$ (i.e. 87.59). Similarly, the fourth column is by considering the best two parameters from

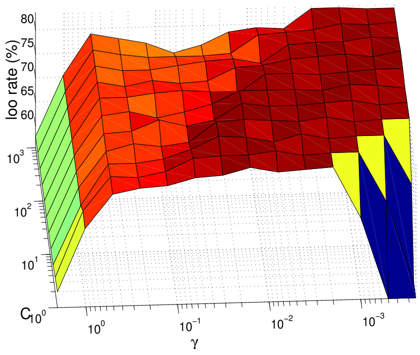


(a) australian ($\epsilon = 0.001$)
(b) australian ($\epsilon = 0.1$)
(c) diabetes ($\epsilon = 0.001$)
(d) diabetes ($\epsilon = 0.1$)
(e) dna ($\epsilon = 0.001$)
(f) dna ($\epsilon = 0.1$)
(g) german ($\epsilon = 0.001$)
(h) german ($\epsilon = 0.1$)

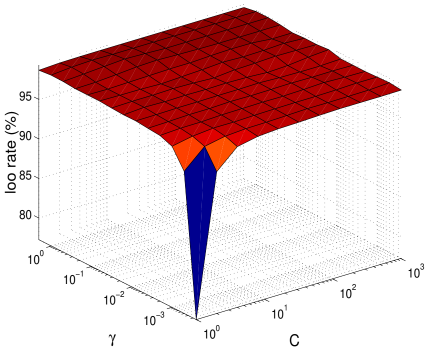
Figure 3.1: Comparison of loo rates using $\epsilon = 0.001$ (left) and $\epsilon = 0.1$ (right)



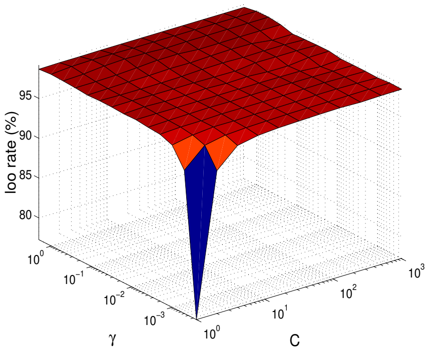
(a) heart ($\epsilon = 0.001$)



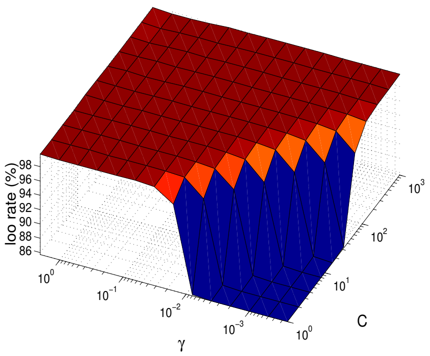
(b) heart ($\epsilon = 0.1$)



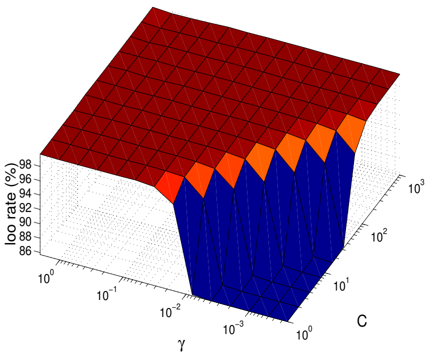
(c) satimage ($\epsilon = 0.001$)



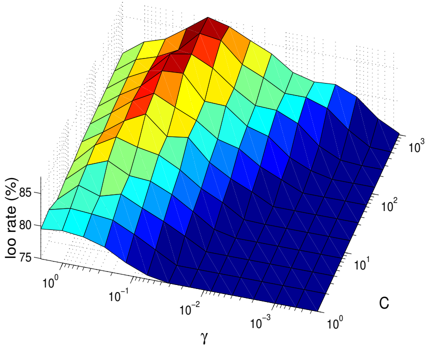
(d) satimage ($\epsilon = 0.1$)



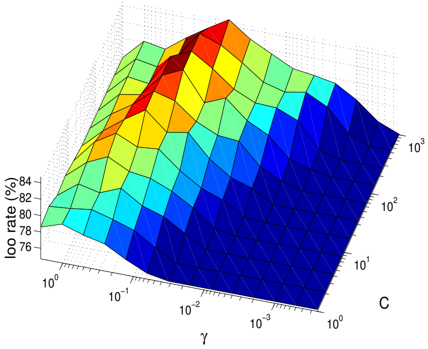
(e) segment ($\epsilon = 0.001$)



(f) segment ($\epsilon = 0.1$)



(g) vehicle ($\epsilon = 0.001$)



(h) vehicle ($\epsilon = 0.1$)

Figure 3.2: Comparison of loo rates using $\epsilon = 0.001$ (left) and $\epsilon = 0.1$ (right)

using $\epsilon = 0.1$. After checking their loo rates with $\epsilon = 0.001$, the larger one is presented. This means practically after all parameters with $\epsilon = 0.1$ are tried, we can pick the best two parameters and find their loo rates by a smaller ϵ. Then the one with a larger loo rate will be for future use. Furthermore, the last column is by checking the best three parameters from using $\epsilon = 0.1$. Table 3.1 clearly indicates that using a loose stopping criterion has already generated good parameters.

Table 3.1: Accuracy of best parameters using a loose stopping criteria: "loo:(C, γ)"

Problem	best loo(%) $\epsilon = 10^{-3}$	non-relaxed rate of top relaxed parameters top 1	top 2	top 3
australian	$85.94 : (32, 0.064)$	$85.65 : (512, 0.002)$	$85.80 : (1, 0.256)$	$85.80 : (2, 0.032)$
diabetes	$77.86 : (1, 0.256)$	$77.73 : (512, 0.004)$	$77.73 : (1, 0.064)$	$77.73 : (64, 0.016)$
dna	$97.30 : (64, 0.016)$	$96.85 : (32, 0.004)$	$97.05 : (64, 0.004)$	$97.05 : (1, 0.032)$
german	$77.40 : (1024, 0.00025)$	$76.50 : (32, 0.001)$	$77.10 : (4, 0.032)$	$77.10 : (128, 0.0005)$
heart	$84.07 : (2, 0.008)$	$83.70 : (4, 0.032)$	$84.07 : (2, 0.008)$	$84.07 : (16, 0.016)$
satimage	$99.35 : (8, 0.512)$	$99.35 : (8, 0.512)$	$99.35 : (16, 0.512)$	$99.35 : (4, 1.024)$
segment	$99.83 : (16, 2.048)$	$99.78 : (1024, 0.032)$	$99.78 : (64, 0.064)$	$99.78 : (128, 0.064)$
vehicle	$87.59 : (512, 0.128)$	$87.12 : (256, 0.128)$	$87.12 : (1024, 0.064)$	$87.59 : (512, 0.128)$

Table 3.2: Total computational time (in seconds)

			Without relaxation		With relaxation	
Problem	l	n	training($\epsilon = 10^{-3}$)	loo($\epsilon = 10^{-3}$)	training($\epsilon = 10^{-3}$)	loo($\epsilon = 10^{-1}$)
australian	690	14	95.89	4776.06	94.77	251.84
diabetes	768	8	167.14	19082.74	167.98	845.65
dna	2000	180	1607.02	33179.04	1596.61	1749.85
german	1000	24	362.20	31951.09	363.85	1368.30
heart	270	13	13.68	408.27	13.59	42.38
satimage	4435	36	1066.98	11809.77	1068.07	1032.18
segment	2310	19	201.16	407.61	201.38	131.19
vehicle	846	18	170.56	9493.26	170.73	500.73

Regarding the total computational time for calculating loo rates of all parameter sets, we present them in Table 3.2. It is not surprising that using $\epsilon = 0.1$ is much faster than $\epsilon = 0.001$. We can clearly see that for problems with just hundreds or thousands of training data, if $\epsilon = 0.001$, a prohibitive amount of computational time is required. However, if $\epsilon = 0.1$ is used instead, the computational time is acceptable.

4 Caches between Different Models

It can be clearly seen in Table 3.2 that when $\epsilon = 10^{-3}$, comparing to the loo time, the training time is negligible. However, this is not the case for $\epsilon = 0.1$. As the stopping criterion is relaxed while calculating the loo rate, the training time is a significant part of the total computational time. In this section we seek for methods for reducing the training time.

The training time is for solving $(\bar{D})$ of each model. For results in Section 3, this is implemented by giving an empty cache with an initial solution zero. For two models with similar parameter sets, the solutions as well as contents of caches after solving $(\bar{D})$ should be alike. Thus such information of one model can be passed to another model as initial guesses.

When γ is fixed and C is increased to δC, it has been studied in (DeCoste and Wagstaff 2000) that $\delta\alpha$, where α is an optimal solution of $(\bar{D})$ using C and γ, is a better initial solution than zero. Here we adopt this strategy. In addition, the change of C does not affect the value of Q so all elements in the cache can be reused. For practical implementation, we also have to change the the gradient $(Q + yy^T)\alpha$ to $(Q + yy^T)\delta\alpha$.

Table 4.3: Total computational time using caches between different models

	training	loo
australian	42.63	224.40
diabetes	95.30	784.55
dna	193.54	1639.22
german	137.88	1269.08
heart	6.05	36.17
satimage	183.51	868.34
segment	16.60	95.51
vehicle	65.52	458.57

When γ is changed, $Q_{ij} = y_i y_j e^{-\gamma\|x_i - x_j\|^2}$ (if the RBF kernel is used) are different. However, if γ is changed from γ_1 to γ_2, then we have

$$Q_{ij} = y_i y_j e^{-\gamma_2\|x_i - x_j\|^2} = y_i y_j (e^{-\gamma_1\|x_i - x_j\|^2})^{\gamma_2/\gamma_1}. \tag{8}$$

Thus by a power operation on each element in the cache, the new contents can be used for the case of γ_2.

We implement an S-shaped strategy to solve all models with different C and γ. First the smallest γ is considered and C is increased. After the model with the largest C is solved, we increase γ. At this time contents of the cache are updated by (8). For our implementation, when parameters are updated from (C, γ_1) to (C, γ_2), the final solution of using (C, γ_1) may not be an applicable initial guess of using (C, γ_2). This is because our implementation also needs the initial gradient. As Q has been changed, there is no way to immediately obtain the gradient unless required elements of Q are calculated right away. Hence at this situation we set the initial solution to be zero. Then C is decreased until the smallest value has been solved. Continually we increase γ, update the cache, and use zero as the initial solution. Such a process is continued until the largest γ has been considered.

In Table 4.3, we present the computational time using the above techniques. We can see that the training time is largely improved.

5 Reducing the Search Space

In this section we demonstrate a simple method to reduce the search space. Remember that the loo rate is calculated by retraining the data #SV times. Hence while calculating the loo of one fixed parameter set, if the rate is already lower than that of the best parameter so far, we can stop earlier and move to the next parameter set. Therefore, we may not have to retrain the data #SV times.

For support vectors x_j obtained by solving $(\bar{D})$, some will be wrongly classified by the following decision function of $(\bar{D}_j)$:

$$\text{sign}(\sum_{i=1}^{l-1} \alpha_i^j y_i^j (K(x_i, x_j) + 1)). \tag{9}$$

Note that (9) is different from (2) due to the use of $(\bar{D})$ instead of (D). We would like to quickly identify such x_j as early as possible. Hence after solving $(\bar{D})$, we sort the value of the decision function at each support vector x_j:

$$y_j(\sum_{i=1}^{l} \alpha_i y_i (K(x_i, x_j) + 1) = ((Q + yy^T)\alpha)_j$$

in the increasing order. We conjecture that if the value is smaller, there is a higher probability that x_j is wrongly classified in (9).

Of course the order of trying different parameter sets affects the performance. If a parameter set with a high loo rate is tested first, more computational time can be saved.

Currently we have not investigated how to choose a good initial parameter set. We simply try C and γ from small to large. Using this trick, for the same problems, the computational time is in Table 5.4. Comparing to Table 4.3, we can clearly see that the loo time is reduced.

Furthermore, users can use the following method to speed up the process. By predetermining a rate r, during the calculation of the loo of a parameter set, if the rate is less than r or the best rate so far, we terminate this loo calculation. Though it is not easy to predetermine this rate, practically if users have known a lower bound of the best rate, an effective r can be chosen to save the total computational time. In addition, users could check if an area of parameters produce a satisfactory accuracy or not. In other words, users can set a large r in the beginning and efficiently find out if this accuracy is achievable.

Table 5.4: Total computational time by early termination of loo

	training	loo
australian	42.77	159.09
diabetes	94.95	524.59
dna	192.83	687.48
german	139.52	921.20
heart	6.03	23.49
satimage	184.10	623.00
segment	17.20	33.42
vehicle	66.46	353.80

6 Discussions and Future Directions

In this paper, we demonstrate a simple method for implementing automatic model selections. It is suitable for problems with up to thousands of variables. There are many possible improvements. Some future directions are as follows:

1. We need more techniques to reduce the search space of parameters. As the loo rate is not a differentiable function of parameters, some non-differentiable optimization techniques may be useful.

2. For polynomial kernel $(x_i^T x_j/\gamma + \delta)^d$, there are more parameters. It will be impossible to try all combination of four parameters. If we can check whether three parameters (e.g. $((x_i^T x_j + 1)/\gamma)^d$) are enough, with techniques mentioned earlier, we think it may still be possible to find the best parameter in a reasonable amount of time.

3. Currently we develop some implementation techniques specifically for the RBF kernel. (For example, the use of Eq. (8).) We will have to investigate other techniques which are either general to most kernels or specific to some of them.

Though the loo calculation for model selections is difficult, in this paper we have shown that by some numerical tricks, the calculation may not be very time consuming. This experience also tells us that for any machine learning algorithm, we cannot judge its real difficulty without conducting some careful numerical investigation.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 89-2213-E-002-013. The authors thank Chih-Wei Hsu for many helpful discussions when we tried to modify BSVM for this work.

References

Campbell, C. (2000). An introduction to kernel methods. In R. J. Howlett and L. C. Jain (Eds.), *Radial basis function networks: design and applications*, Berlin. Springer Verlag.

Chapelle, O. and V. Vapnik (2000). Model selection for support vector machines. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems 12.* Cambridge, Mass: MIT Press.

Chapelle, O., V. Vapnik, O. Bousquet, and S. Mukherjee (2000). Choosing kernel parameters for support vector machines. Technical report. Submitted to Machine Learning.

Collobert, R. and S. Bengio (2000). SVMTorch: A support vector machine for large-scale regression and classification problems. Available at `http://www.idiap.ch/learning/SVMTorch.html`.

DeCoste, D. and K. Wagstaff (2000). Alpha seeding for support vector machines. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD-2000).*

Friess, T.-T., N. Cristianini, and C. Campbell (1998). The kernel adatron algorithm: a fast and simple learning procedure for support vector machines. In *Proceeding of 15th*

Intl. Conf. Machine Learning. Morgan Kaufman Publishers.

Hsu, C.-W. and C.-J. Lin (1999). A simple decomposition method for support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. Submitted to Machine Learning.

Jaakkola, T. S. and D. Haussler (1999). Probabilistic kernel regression models. In *Proceedings of the 1999 Conference on AI and Statistics*.

Joachims, T. (1998). Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA. MIT Press.

Joachims, T. (2000). Estimating the generalization performance of a SVM efficiently. In *Proceedings of the International Conference on Machine Learning*, San Francisco. Morgan Kaufman.

Keerthi, S., S. Shevade, C. Bhattacharyya, and K. Murthy (1999). Improvements to Platt's SMO algorithm for SVM classifier design. Technical report, Department of Mechanical and Production Engineering, National University of Singapore. To appear in Neural Computation.

Mangasarian, O. L. and D. R. Musicant (1999). Successive overrelaxation for support vector machines. *IEEE Trans. Neural Networks 10*(5), 1032–1037.

Michie, D., D. J. Spiegelhalter, and C. C. Taylor (1994). *Machine Learning, Neural and Statistical Classification*. Englewood Cliffs, N.J.: Prentice Hall. Data available at anonymous ftp: `ftp.ncc.up.pt/pub/statlog/`.

Osuna, E., R. Freund, and F. Girosi (1997). Training support vector machines: An application to face detection. In *Proceedings of CVPR'97*.

Platt, J. C. (1998). Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA. MIT Press.

Rüping, S. (2000). mySVM - another one of those support vector machines. Software available at `http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/`.

Saunders, C., M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola (1998). Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal

Holloway, University of London, Egham, UK.

Vapnik, V. and O. Chapelle (2000). Bounds on error expectation for SVM. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans (Eds.), *Advances in Large Margin Classifiers*, Cambridge, MA, pp. 261–280. MIT Press.