Speeding-up Model Selection for Support Vector Machines

DucDung Nguyen and TuBao Ho

Japan Advanced Institute of Science and Technology
1-1, Asahidai, Tatsunokuchi, Ishikawa 923-1292 Japan
{dungduc, bao}@jaist.ac.jp

Abstract

One big difficulty in the practical use of support vector machines is the selection of a suitable kernel function and its appropriate parameter setting for a given application. There is no rule for the selection and people have to estimate the machine's performance based on a costly multi-trial iteration of training and testing phases. In this paper, we describe a method to reduce the model selection training time for support vector machines. The main idea is, in the process of trying a series of models, the support vectors in previously trained machines are used to initialize the working set in training a new machine. This initialization helps to reduce the number of required optimization loops, thus reducing the training time of the model selection process. Experimental results on real-life datasets show that the training time for each subsequent machine can be reduced effectively in a variety of situations in the model selection process. The method is applicable to different model search strategies and does not affect the model selection result.

Introduction

In a tutorial paper on support vector machines (SVMs) for pattern recognition, Burges (Burges 1998) pointed out three main limitations of the support vector learning approach. The biggest limitation lies in the choice of the kernel and its parameter setting. The second limitation is speed and size, in both the training and testing phases, and the third limitation is dealing with discrete data. Unfortunately, solving the model selection problem for SVMs means we must solve the first two major limitations mentioned above.

The model selection (MS) problem asks the following question (Scheffer and Joachims 1999; Kearns et al. 1995; Forster 2000): given an observed data set, which model or learning algorithm and with which parameter setting will perform best on the unseen data? For SVMs, it is necessary to select the most suitable kernel, its parameter value(s), and the appropriate error cost. To answer the question, candidate models should be tried, and the model with the highest estimated performance should be selected. This is a costly task because it requires multiple trials of training and testing models.

In this paper we describe a method to speed up the model selection process by reducing the training time of the models being considered. The main idea is, in the sequence of trying models, the results of previously trained machines are reused to train new machines. More concretely, the support vectors in previously trained machines are used to initialize the working set in training each new machine. This initialization of the working set helps to reduce the required number of optimization loops, so the optimization process can converge more quickly. Experimental results on three real-life datasets *sat-image*, *letter recognition*, and *shuttle* in the StatLog collection (Michie, Spiegelhalter, and Taylor 1994) show that the training time for each subsequent machine can be reduced from 28.8% to 85.5% depending on situations in the MS process. The method is applicable to common model search strategies like grid search (Hsu and Lin 2002), pattern search (Momma and Bennett 2002), or gradient-based (Chapelle and Vapnik 2000; Keerthi 2002), and does not change the result of model selection.

Model Selection for Support Vector Machines

Support Vector Learning

The support vector learning approach (Vapnik 1995) finds the separating hyperplane that maximizes the distance from the plane to the training data, either in the input space or in the feature space, depending on the choice of kernel. For a two-class classification problem, suppose that we are given a set of training data S = $\{(x_i, y_i)\}_{i=1,\ldots,N}$, where $x_i \in R^d$ and $y_i \in \{-1, 1\}$; support vector learning finds the optimal hyperplane by maximizing the following function:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

under constraints

$$\sum_{i=1}^{N} y_i \alpha_i = 0$$

$$0 \le \alpha_i \le C, i = 1, \ldots, N$$

The norm of the plane is determined by a linear combination of vectors x_i with associated coefficient $\alpha_i > 0$, called *support vectors* (SVs). The parameter C controls the trade-off between the complexity of the decision function and the number of training errors. The decision function then takes the form:

$$f(x) = sign\left(\sum_{\alpha_i > 0} y_i \alpha_i K(x_i, x) + b \right)$$

where $K(x, y)$ is a function calculating the dot product of two vectors in the feature space. Different kernel functions will produce different types of machines. For example, the machine might be a polynomial (polynomial kernel), a radial basis function (Gaussian RBF kernel), or a particular type of two-layer sigmoidal neural network (sigmoidal kernel) (Burges 1998). Common kernels are listed in Table 1.

Table 1: Common kernel functions.

Linear	$K(x, y) = x.y$
Gaussian RBF	$K(x, y) = \exp(-\gamma \|x - y\|^2)$
Polynomial	$K(x, y) = ((x.y) + 1)^d$
Sigmoidal	$K(x, y) = \tanh(\kappa(x.y) + 1)$

Model Selection for Support Vector Machines

k-fold cross validation is one of the most widely used methods for performance evaluation and model selection, not only for SVMs but also for other learning approaches. In *k*-fold cross validation, the available dataset S is divided into k disjoint parts $S = S_1 \cup S_2 \cup \ldots \cup S_k$. Models are trained on the k-1 parts $S_1 \cup S_2 \cup \ldots S_{i-1} \cup S_{i+1} \cup \ldots S_k$ and tested on the remaining part S_i. The performances of k validation tests are then averaged. Though the *k*-fold cross validation method is simple, consistent (Shao and Tu 1995), almost unbiased (Efron and Tibshirani 1993), and works well in many applications (Kohavi 1995); the main drawback of *k*-fold cross validation is its high computational cost. For evaluating one model, 10-fold cross validation needs 10 times of training and testing. Another approach to estimating the performance of a SVM is to determine its upper-bound of error; users then look for the machine that has the lowest bound, instead of the lowest error estimated by the cross validation. In this way each model is trained and tested just one time. Various kinds of bounds have been calculated as a function of the training error and the complexity of the machine (such as the VC-dimension of the machine (Vapnik 1998), the radius-margin ratio (Vapnik and Chapelle 2000; Keerthi 2002), and the span of support vectors (Chapelle and Vapnik 2000)).

For a given application, the user must choose the type of kernel, its parameter value(s), and the error cost for the SVM training program. For example, when the Gaussian RBF kernel is chosen, the user must select the parameter determining the width γ of the function and the penalty cost C. Because these two parameters take values in R, then there is an infinite number of possible pairs of parameters, or infinite models which could be taken into consideration. Even when the grid search method is applied (Hsu and Lin 2002), the number of candidate models is still very large. To reduce this number, one can apply a pattern search (Momma and Bennett 2002) in the parameter space, or when the derivation of the bounds can be calculated or approximated, the gradient method (Chapelle and Vapnik 2000; Keerthi 2002) can be used to reduce the number of models considered.

All the above model selection methods require trying a series of models, or running the SVM learning program many times with different parameter values. To speed up this process we can apply a data filtering technique to reduce the size of the problem (Ou et al. 2003), or seeding the initial solution using the solution of another machine of the same kernel (DeCoste and Wagstaff 2000). In this paper we propose a new method to speed up the training phase in MS for SVMs by using SVs in previously trained machines to build up the working set in training a new machine. This helps to reduce the number of optimization loops, and thus can reduce the training time.

Speeding-up Model Selection

The General Decomposition Algorithm for SVM Training

Support vector learning solves a constrained quadratic optimization of N variables, where N is the number of training data. This becomes a heavy task when N is big, and there exist a large number of proposed solutions to this problem. One common framework that has been described in literature, as well as implemented in most free and commercial software, is the decomposition algorithm described in Table 2. The original big quadratic programming (QP) problem is decomposed into a sequence of smaller QPs (Osuna, Freund, and Girosi 1997). The result of the decomposition method is ensured to be consistent with the original problem due to the convexity and the uniqueness of the global solution. In this method the size of the working set (WS) is specified in advance, to be big enough to cover all SVs and small enough not to exceed the capacity of the computer (e.g. RAM memory). Each small QP on the WS is solved using existing techniques, most frequently the sequential minimal optimization (SMO) algorithm (Platt 1999), with a faster speed and a much smaller memory requirement. After that, the WS is updated by replacing vectors with zero

Table 2: Decomposition Algorithm for SVM Training.

```
SVM Learning Algorithm

Input:
        a set $S$ of $N$ training examples $\{(x_i,y_i)\}_{i=1...N}$
        the size of the working set $l$
Output:
        a set of $N$ coefficient $\{\alpha_i\}_{i=1...N}$

// Initialization
1. Set all $\alpha_i$ to zero
2. Select a working set $B$ of size $l$
// Optimization
3. Repeat
4.        Solve the local optimization on $B$
5.        Update the working set $B$
6. Until the global optimization condition is satisfied
```

coefficient by other vectors (not in the WS) that does not satisfy the optimization conditions (the Karush-Kuhn-Tucker conditions). The process is repeated until there is no vector violating the KKT conditions, or the global optimization conditions are met.

Building the Working Set

The selection of the WS in the above algorithm has a big impact on the convergence of the decomposition method. If all support vectors (those with the corresponding $\alpha_i > 0$) are selected in the initial WS, then the optimization loop in the decomposition method is performed just one time. In other words, a better initial working set with as many support vectors as possible will produce a closer local solution to the global solution, and will lead to a faster global convergence. However, in practice we don't know in advance which training vector will be the support vector, and in practice there is no way but to randomly initialize the working set, e.g. (Dong, Krzyzak, and Suen 2003). We can also increase the number of support vectors to be included by increasing the size of the WS, but the side effect is that this will also increase the optimization time on the WS, and therefore will increase the training time (Dong, Krzyzak, and Suen 2003). Moreover, the size of the WS is limited by the memory of the computer, due to the requirement of the kernel matrix.

Our method starts with the fact that support vectors are training examples that lie close to the border between two classes, and two different machines may have many of them in common. This property has been reported in literature; for example, on the USPS hand-written digit dataset, two different machines trained by two different kernels, RBF and polynomial, share more than 80% of support vectors (Vapnik 1998). To reconfirm this argument we have conducted intensive experiments on three datasets, *sat-image*, *letter recognition*, and *shuttle*, from the StatLog collection (Michie, Spiegelhalter, and Taylor 1994). Details are reported in Figure 1. The results of these experiments show that two different machines trained by different parameter settings might still share a large number of support vectors. In MS context, many models must be tried, and we can benefit from using the information of previously trained models in training new ones.

One simple yet effective way is to select the SVs in trained machines as the initial working set for training a new machine. This method faces two difficulties. First, when the number of classes in the dataset is more than two, then the number of SVM required to build-up a classifier is m or $m*(m-1)/2$ binary classifiers depending on whether the selected strategy is one-versus-one or one-versus-rest, where m is the number of classes. In order to retain the information of previously trained machines, we need m or $m*(m-1)/2$ different sets of SVs. The second problem is that because the size of the working set is given in advance then the total number of SVs may exceed this limitation. In our experiments, a FIFO (First In First Out) queue structure with the same size as the working set was used to store the SVs of previously trained (binary) machines. With this structure, all the SVs of the latest trained machine (supposed to be closest to the next machine) are kept in the initial working set.

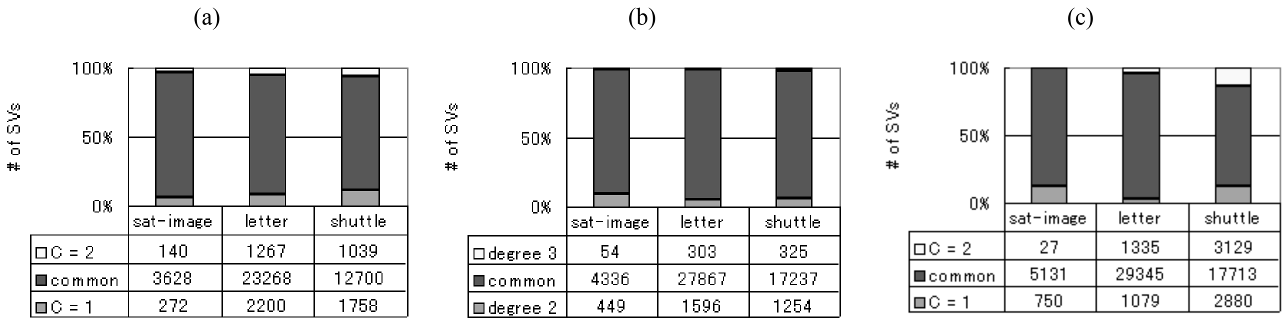
Experiments

We conducted experiments on three datasets in the StatLog collection: *sat-image*, *letter recognition*, and *shuttle*. These datasets are summarized in Table 3. They were chosen for their generality in dimension, size, number of classes, and the class distribution.

Table 3: Datasets used in experiments

Dataset	# Attribute	# Class	Size
Sat-image	36	6	4,435
Letter recognition	16	26	15,000
Shuttle	9	7	43,500

To see the effect of the method in real situations, we conducted experiments in different scenarios, including fixing the kernel and varying the cost parameter, fixing the cost and changing the kernel, and changing both kernel and cost parameter. In the first scenario we fixed the kernel to be linear and varied the cost parameter from 1 to 10 (Figure 2: a, b, c). The second scenario was to fix the cost parameter at 1 and train the machines with polynomial kernels of degree from 2 to 9 (Figure 2: d, e, f). The third scenario used Gaussian RBF kernels of the width γ changing from 0.01 to 0.1 with a step of 0.01 and varied the cost parameter from 1 to 10 (Figure 2: g, h, i). The kernel cache sizes were 2000 (*sat-image*), 2000 (*letter*



(a)

	sat-image	letter	shuttle
□ C = 2	140	1267	1039
■ common	3628	23268	12700
▨ C = 1	272	2200	1758

(b)

	sat-image	letter	shuttle
□ degree 3	54	303	325
■ common	4336	27867	17237
▨ degree 2	449	1596	1254

(c)

	sat-image	letter	shuttle
□ C = 2	27	1335	3129
■ common	5131	29345	17713
▨ C = 1	750	1079	2880

Figure 1: Support vectors in two different machines learned from three datasets *sat-image*, *letter recognition*, and *shuttle*: (a) linear machines learned with different error cost $C = 1$ and $C = 2$, (b) polynomial machines of degree two and three learned with the same error cost $C = 1$, (c) RBF machines learned with different cost penalties $C = 1$ and $C = 2$. Two different machines trained by two different parameter settings may share a big portion of SVs.

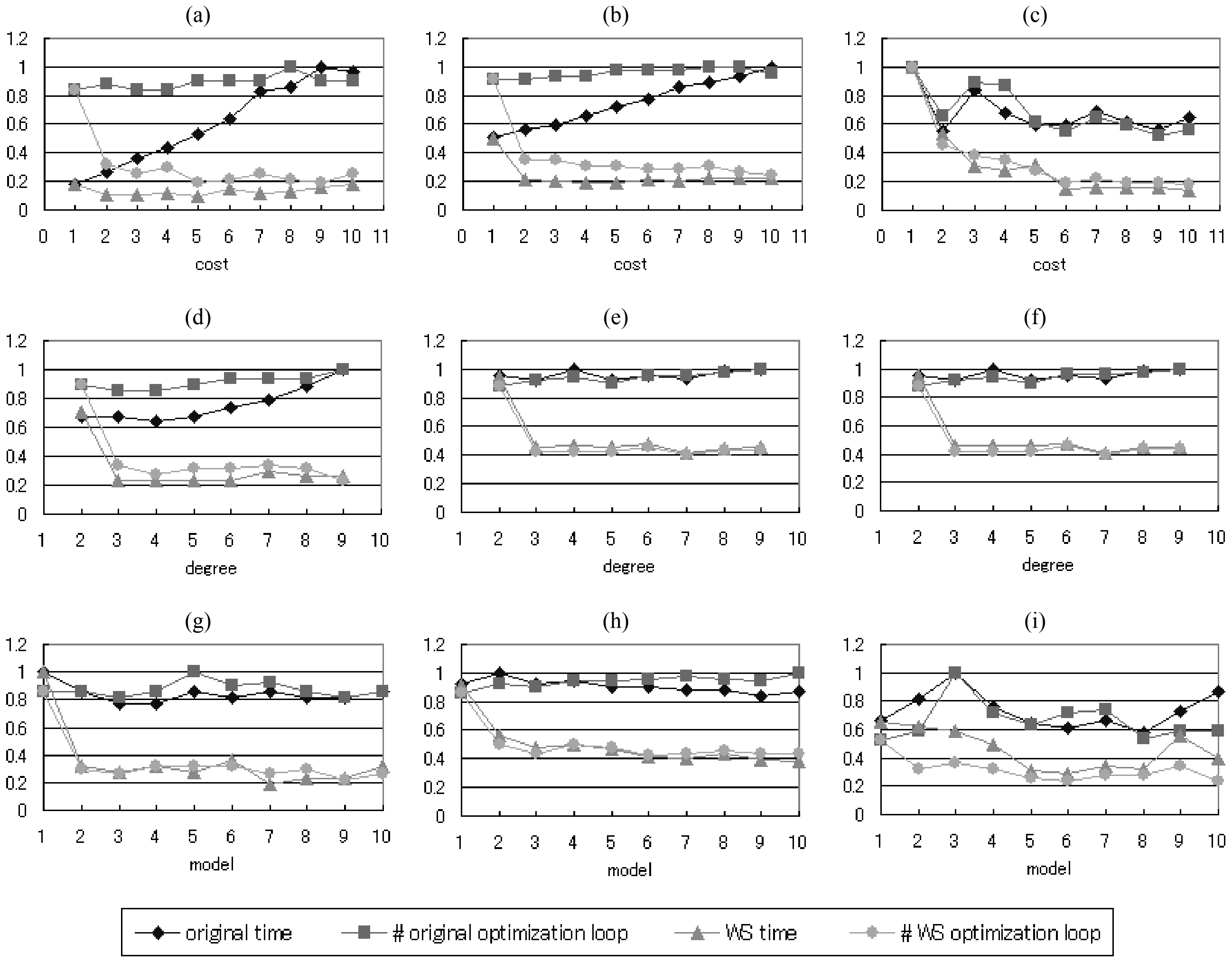


Figure 2: Reduction in number of required optimization loops and training time on three datasets *sat-image* (a-d-g), *letter recognition* (b-e-h), and *shuttle* (c-f-i), and in different situations: the same linear kernel with different cost (a-b-c), polynomial kernels of different degree with the same cost, and different RBF kernels with different costs. All measures (average number of loops and average training time) are normalized into (0,1]. "WS" stands for the proposed working set selection method.

recognition), and 10,000 (*shuttle*). The optimization program was an implementation of the SMO algorithm and its improvement (Platt 1999; Keerthi et al. 2001). Experiments were conducted on a PC Windows XP with 2.99 GH, 2GB RAM.

Experimental results are reported in Figure 2. In this figure, we compare the number of optimization loops and running time for each parameter setting on the three datasets. Because the numbers of classes are greater than 2 for all three datasets then for each parameter setting the training program had to run *m* times, where *m* is the number of classes (using one-versus-rest strategy). For comparison purpose, the number of optimization loops and running time in Figure 2 were averaged and normalized into (0,1] (divided by the maximum value). From the results we can see that in every situation the training time for each subsequent machine was reduced significantly, from 22.8% (*shuttle* dataset, RBF kernel, error cost 2) to 85.5% (*sat-image* dataset, linear kernel, cost 7).

Conclusion

We have described a method to speed up the training phase in model selection for support vector machines. The method utilizes the support vectors of previously trained machines to initialize the working set in training a new machine. This initialization scheme makes the training process converge more quickly. Experiment results on real life datasets show that the training time of subsequent machines can be reduced significantly.

In comparing with other methods, the proposed one has two main advantages. First, it does not change the result of model selection. This is because the proposed method aims at initializing a better working set, leading to a faster convergence in training. In (Ou et al. 2003), a data filtering method is used to reduce the number of data in the dataset, or to reduce the size of the optimization problem. The data reduction makes the model selection process run faster, but the result is not the same as working on the entire available dataset due to the distortion of the training data. Moreover, the data-filtering algorithm has its own parameter *k* (in *k*-NN classification), so for each application it is necessary to do another model selection job in order to find the best value of *k*. The second advantage is the applicability of the proposed method in different situations and for different model search strategies like grid search (Hsu and Lin 2002), pattern search (Momma and Bennett 2002), and gradient-based methods (Chapelle and Vapnik 2000; Keerthi 2002). The alpha-seeding method in (DeCoste and Wagstaff 2000) is limited to one kind of kernel and with a limited scheme of varying cost parameter. Experiment results on the *adult* dataset in the UCI corpus with linear kernel show the effectiveness of the alpha-seeding method (a reported of 5 times faster), but for machines with different kernels and different cost values, this method is not applicable.

The future work of this research is to enhance the way we utilize previously trained machines in initializing the working set, for example, using not only the support vectors (those with a distance to the separating hyperplane smaller than or equal one), but also the vectors that lie close to the separating plane (those with a distance to the separating hyperplane greater than one).

References

Burges C. J. C. 1998. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining* 2(2):121-167.

Chapelle O. and Vapnik V. 2000. Model selection for support vector machines. *Advances in Neural Information Processing Systems 12*, ed. S.A. Solla, T.K. Leen and K. R. Muller. Cambridge, MA.: MIT Press.

DeCoste D. and Wagstaff K. 2000. Alpha Seeding for Support Vector Machines. *International Conference on Knowledge Discovery & Data Mining (KDD-2000).*

Dong J. X., Krzyzak A., and Suen C. Y. 2003. A fast SVM training algorithm. *International Journal of Pattern Recognition and Artificial Intelligence* 17(3): 367-384.

Efron B. and Tibshirani R. 1993. *An Introduction to the Bootstrap*, London : Chapman & Hall.

Forster M. R. 2000. Key Concepts in Model Selection: Performance and Generalizability. *Journal of Mathematical Psychology* 44:205-231.

Hsu C. W. and Lin C. J. 2002. A comparison on methods for multi-class support vector machines. *IEEE Transactions on Neural Networks* 13:415-425.

Kearns M., Mansour Y., Andrew Y. Ng. and Ron D. 1997. An Experimental and Theoretical Comparison of Model Selection Methods. *Machine Learning* 27(1):7-50.

Keerthi S., Shevade S., Bhattacharyya C., and Murthy K. 2001. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation* 13: 637–649.

Keerthi S.S. 2002. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. *IEEE Transactions on Neural Networks* 13: 1225-1229.

Kohavi R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1137-1143, San Mateo, CA: Morgan Kaufmann.

Michie D., Spiegelhalter D. J., and Taylor C. C. 1994. *Machine Learning, Neural and Statistical Classification*. N.Y.: Ellis Horwood.

Momma M. and Bennett K.P. 2002. A pattern search method for model selection of support vector regression. In *Proc. of SIAM Conference on Data Mining*.

Osuna E., Freund R., and Girosi F. 1997. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII - Proceedings of the 1997 IEEE Workshop*, 276 - 285, New York.

Ou Yu-Yen, Chen Chien-Yu, Hwang Shien-Ching, and Oyang Yen-Jen. Expediting Model Selection for Support Vector Machines Based on Data Reduction. In *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics.* Washington D.C.

Platt J. 1999. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, 185-208. Cambridge, MA.: MIT Press.

Scheffer T. and Joachims T. 1999. Expected error analysis for model selection. In *Proceedings of ICML-99, 16th International Conference on Machine Learning*. San Francisco: Morgan Kaufmann.

Shao J. and Tu D. 1995. *The Jackknife and Bootstrap*. New York: Springer-Verlag.

Vapnik V. 1995. *The Nature of Statistical Learning Theory*. N.Y.: Springer.

Vapnik V. 1998. *Statistical Learning Theory*. N.Y.: John Wiley & Sons.

Vapnik V. and Chapelle O. 2000. Bounds on Error Expectation for Support Vector Machines. *Neural Computation* 12(9):2013-2036.