Fast Pattern Selection for Support Vector Classifiers

Hyunjung Shin and Sungzoon Cho

Department of Industrial Engineering, Seoul National University,
San 56-1, Shillim-Dong, Kwanak-Gu, 151-742, Seoul, Korea
{hjshin72, zoon}@snu.ac.kr

Abstract. Training SVM requires large memory and long cpu time when the pattern set is large. To alleviate the computational burden in SVM training, we propose a fast preprocessing algorithm which selects only the patterns near the decision boundary. Preliminary simulation results were promising: Up to two orders of magnitude, training time reduction was achieved including the preprocessing, without any loss in classification accuracies.

1 Introduction

One of the strongest points of Support Vector Machine (SVM) theory is the fact that SVM quadratic programming (QP) problem is quite simple [14]. In SVM QP formulation, the dimension of kernel matrix ($M \times M$) is equal to the number of training patterns(M). Unfortunately, however, this innate characteristic yields a difficulty when a large scale real-world problem is given. This enormous matrix cannot easily fit into the memory of a standard computer. Moreover, training cannot be finished in a reasonable time even if we manage to load the matrix on the memory.

A standard QP solver has time complexity of order $O(M^3)$ [10]. Here, for the sake of simplicity, we do not consider the dimension of input pattern. A small training set can be solved with such standard QP solvers: MINOS, CPLEX, LOQO and MATLAB QP routines. In order to attack the large scale SVM QP problem, the decomposition methods or iterative methods have been suggested which break down the large QP problem into a series of smaller QP problems. They include the Chunking algorithm, Sequential Minimal Optimization (SMO) algorithm, SVM Light algorithm and Successive Over Relaxation (SOR) algorithm [5]. The general time complexity of those decomposition methods is approximately (*the number of iterations*)$\cdot$ $O(Mq+q^3)$: if q is the size of the working set, each iteration costs $O(Mq)$ to calculate q columns of the kernel matrix, about $O(q^3)$ for solving the subproblem, and $O(Mq)$ for finding the next working set and updating the gradient. "The number of iterations" is supposed to increase as the number of training patterns increases.

K.-Y. Whang, J. Jeon, K. Shim, J. Srivatava (Eds.): PAKDD 2003, LNAI 2637, pp. 376–387, 2003.

One way to circumvent this computational burden is to select only the training patterns, in advance, that are more likely to be support vectors. In a classification problem, the support vectors are distributed near the decision boundary and participate in the constitution of the margins thereabout. Therefore, selecting those patterns (potential support vectors) prior to SVM training is quite desirable (see Fig. 1).

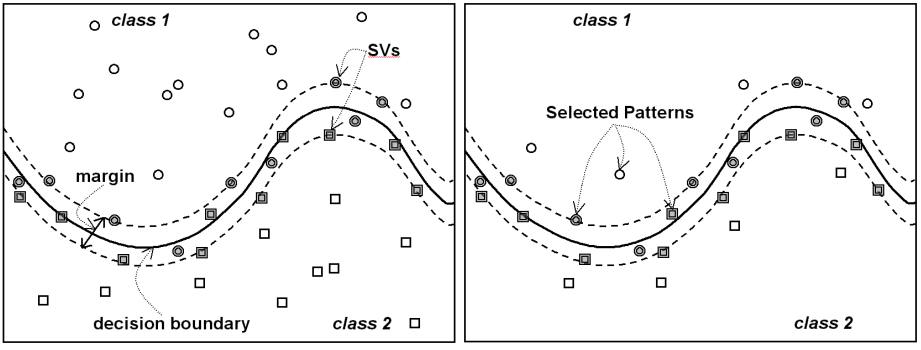


Fig. 1. Pattern selection: a large training set shown in (a) is condensed to a small training set (b) which is composed of only potential support vectors near the decision boundary.

Up to now, there have been considerable researches about pattern selection near the decision boundary for classification problem. Shin and Cho selected the clean patterns near the decision boundary based on the bias and variance of outputs of a network ensemble [12]. Lyhyaoui et al. implemented RBF classifiers, somewhat like SVMs, by selecting patterns near the decision boundary. They proposed 1-nearest neighbor method in opposite class after class-wise clustering [7]. To lessen the burden of the MLP training, Choi and Rockett used kNN classifier to estimate the posterior probabilities for pattern selection. But one major drawback is that it takes approximately $O(M^2)$ to estimate the posterior probabilities [3]. A pattern selection approach, more focused on SVMs, was proposed by Almeida et al. They conducted k-means clustering on the entire training set. All patterns were selected for heterogeneous clusters in their class membership, while only centroids were selected for homogeneous clusters [1].

Recently, we proposed to select the patterns near the decision boundary based on the neighborhood properties [11]. We utilized k nearest neighbors to look around the pattern's periphery. The first idea stems from the fact that a pattern located near the decision boundary tends to have more heterogeneous neighbors with respect to their class membership. The degree of proximity to the decision boundary is estimated by Neighbors_Entropy. A larger Neighbors_Entropy value indicates that the pattern is close to the decision boundary. The second idea arises from the fact that a pattern on the right side of the decision boundary tends to belong to the same class as its neighbors. Among the patterns with

positive Neighbors_Entropy value, potential noisy patterns are eliminated by the ratio of the neighbors whose label matches that of the pattern. A smaller ratio indicates that the pattern is potentially incorrectly labeled. The approach reduced the number of patterns significantly, thus reduced the training time. However, the pattern selection process evaluating the kNNs for all patterns, was time consuming. A naive algorithm took $O(M^2)$, so the pattern selection process became a near bottleneck.

In this paper, we propose a fast version of the algorithm. Here, we just compute the kNNs of the patterns near the decision boundary, not all training patterns. The idea comes from another simple neighborhood property that the neighbors of the pattern located near the decision boundary tend to be located near the decision boundary as well. The time complexity of the fast algorithm is $O(bM)$, where b is the number of patterns in the "overlapped" region around decision boundary.

Now, we briefly summarize the contents of the paper. In section 2, we present our basic idea of selecting the patterns near the decision boundary. In section 3, we propose the fast algorithm. In section 4, we show the experimental results, involving two synthetic problems and one real world problem. In the last section, we conclude the paper with the discussion of the limitations.

2 Selection Criteria Based on Neighborhood Properties

The first neighborhood property is that a pattern located near the decision boundary tends to have heterogeneous neighbors with respect to their class labels. Thus, the degree of pattern $\boldsymbol{x}$'s proximity to the decision boundary can be estimated by "*Neighbors_Entropy($\boldsymbol{x}$)*", which is defined as the entropy of pattern $\boldsymbol{x}$'s k-nearest neighbors' class labels (see Fig. 2). A pattern with a positive Neighbors_Entropy($\boldsymbol{x}$) value is assumed to be located near the decision boundary. Note that the measure considers the pattern's neighbors only, not the pattern itself.

The second neighborhood property is that a pattern on the right side of the decision boundary tends to belong to the same class as its neighbors. If a pattern's own label does not match the majority label of its neighbors', it is likely to be incorrectly labeled. The measure "*Neighbors_Match($\boldsymbol{x}$)*" is defined as the ratio of $\boldsymbol{x}$'s neighbors whose label matches that of $\boldsymbol{x}$. Only those pattern $\boldsymbol{x}$s are selected whose Neighbors_Match($\boldsymbol{x}$) $> \beta \cdot \frac{1}{J}$ (J is the number of classes and $0 < \beta \leq 1$). In other words, potential noisy patterns are eliminated. The parameter β controls the selectivity. We have empirically found a value of 0.5. Based on those neighborhood properties we propose following selection criteria.

[Selecting Criteria] Neighbors_Entropy($\boldsymbol{x}$) > 0 and Neighbors_Match($\boldsymbol{x}$) $> \beta \cdot \frac{1}{J}$.

3 Fast Implementation of the Proposed Method

A naive algorithm was represented in [11] where the kNNs of all patterns were evaluated. This algorithm is easy to implement and also performs well as long as the size of training set, M, is relatively small.

LabelProbability($\boldsymbol{x}$) {

/* For $\boldsymbol{x}$, calculate the label probabilities of $k\mathbf{NN}(\boldsymbol{x})$ over J classes, $\{C_1, C_2, \ldots, C_J\}$, where $k\mathbf{NN}(\boldsymbol{x})$ is defined as the set of k nearest neighbors of $\boldsymbol{x}$. */

$k_j = |\{\boldsymbol{x}' \in C_j | \boldsymbol{x}' \in k\mathbf{NN}(\boldsymbol{x})\}|, \quad j = 1, \ldots, J.$

return $\left(P_j = \frac{k_j}{k}, \forall j\right)$.

}

Neighbors_Entropy($\boldsymbol{x}$) {

/* Calculate the neighbors-entropy of $\boldsymbol{x}$ with its nearest neighbors' labels. In all calculations, $0\, log_J \frac{1}{0}$ is defined to be 0. */

Do **LabelProbability($\boldsymbol{x}$)**.

return $\left(\sum_{j=1}^{J} P_j \cdot log_J \frac{1}{P_j}\right)$.

}

Neighbors_Match($\boldsymbol{x}$) {

/* Calculate the neighbors-match of $\boldsymbol{x}$. j^* is defined as the label of $\boldsymbol{x}$ itself.*/

$j^* = \arg_j \{C_j \mid \boldsymbol{x} \in C_j, j = 1, \ldots, J\}$.

Do **LabelProbability($\boldsymbol{x}$)**.

return (P_{j^*}).

}

Fig. 2. Neighbors_Entropy and Neighbors_Match functions

However, when the size of training set is large, the computational cost increases in proportion to the size. Let's assume that the distance between any two points in d-dimensional space can be computed in $O(d)$. Then to find the nearest neighbors for each pattern takes sum of "distance computation time (DT)" and "search time (ST)". The total time complexity of the naive algorithm, therefore, is $O\,(\ M \cdot (DT + ST)\ )$. Roughly speaking, it is $O(M^2)$.

DT : Distance Computation Time	$O\,(\ d \cdot (M-1)\ )$
ST : Search(query) Time	$\min\{O\,(\ (M-1) \cdot \log(M-1)\ ), O\,(\ k \cdot (M-1)\ )\}$
Total Time Complexity	$O\,(\ M \cdot (DT + ST)\ ) \approx O(M^2)$

There is a considerable literature on nearest neighbor searching for the case of the huge data size and the high dimensional. Most approaches focus on reducing DT or ST. See [4] and [13] for reducing DT, and [2] and [8] for reducing ST.

Our approach, on the other hand, focuses on reducing M of $O\,(M \cdot (DT + ST))$. The idea comes from yet neighborhood property that the neighbors of the pattern located near the decision boundary tend to be the patterns near the decision boundary themselves. We start with a set of randomly

selected patterns. Once we find some of the patterns near the decision boundary, we examine the its neighbors. This successive evaluation of the neighbors of a current set of patterns will stop when all the patterns near the decision boundary are chosen and evaluated. By following [Spanning Criteria], it is determined whether the next evaluation will be spanned to the pattern's neighbors or not. This "selective kNN spanning" procedures is shown in Fig. 3.

[Spanning Criteria] Neighbors_Entropy($\boldsymbol{x}$) > 0

4 Experiments

The proposed algorithm was tested on two artificial binary-class problems and one well-known real world problem. All simulations were carried on a Pentium 800 MHz PC with 256 MB memory.

4.1 Synthetic Problems

The two problems were devised for the performance proof according to the different decision boundary characteristics. We used Gunn's SVM Matlab Toolbox [15].

Continuous XOR Problem with Four Gaussian Densities: The first problem is a continuous XOR problem. From the four Gaussian densities, a total of 600 training patterns were generated. There are about 10% patterns in the overlapped region between the densities. The characteristic of this problem is that the density of patterns gets sparser when it goes closer to the decision boundary.

$$C_1 = \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in C_{1A} \cup C_{1B},\ \begin{bmatrix} -3 \\ -3 \end{bmatrix} \leq \boldsymbol{x} \leq \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right\},$$
$$C_2 = \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in C_{2A} \cup C_{2B},\ \begin{bmatrix} -3 \\ -3 \end{bmatrix} \leq \boldsymbol{x} \leq \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right\}$$

where C_{1A}, C_{1B}, C_{2A} and C_{2B} were

$$C_{1A} = \left\{ \boldsymbol{x} \mid N\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.5^2 \end{bmatrix} \right) \right\},\ C_{1B} = \left\{ \boldsymbol{x} \mid N\left(\begin{bmatrix} -1 \\ -1 \end{bmatrix}, \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.5^2 \end{bmatrix} \right) \right\},$$
$$C_{2A} = \left\{ \boldsymbol{x} \mid N\left(\begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.5^2 \end{bmatrix} \right) \right\},\ C_{2B} = \left\{ \boldsymbol{x} \mid N\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 0.5^2 & 0 \\ 0 & 0.5^2 \end{bmatrix} \right) \right\}.$$

$\mathbf{D}$	The original training set.
$\mathbf{D}_{\mathbf{e}}^{i}$	The set to be evaluated (spanned or non-spanned) at i th step.
$\mathbf{D}_{\mathbf{o}}^{i}$	The set of the patterns to be spanned at i th step from $\mathbf{D}_{\mathbf{e}}^{i}$. Each element will find its k nearest neighbors to constitute the next evaluation set, $\mathbf{D}_{\mathbf{e}}^{i+1}$.
$\mathbf{D}_{\mathbf{x}}^{i}$	The set of the patterns not to be spanned at i th step from $\mathbf{D}_{\mathbf{e}}^{i}$.
$\mathbf{D}_{\mathbf{s}}^{i}$	The set of the selected patterns at i th step from $\mathbf{D}_{\mathbf{o}}^{i}$.
$\mathbf{S}_{\mathbf{o}}^{i}$	The accumulated set of the spanned patterns after subsequent evaluations till (i-1) th step.
$\mathbf{S}_{\mathbf{x}}^{i}$	The accumulated set of the non-spanned patterns after subsequent evaluations till (i-1) th step.
$\mathbf{SS}^{i}$	The accumulated set of the selected patterns till (i-1) th step, the final set will be returned as a aimed set.
$k\mathbf{NN}(\boldsymbol{x})$	The set of k-nearest neighbors of $\boldsymbol{x}$, whose cardinality is k.

SelectiveKnnSpanning() {

[0] Initialize $\mathbf{D}_{\mathbf{e}}^{0}$ with randomly chosen patterns from $\mathbf{D}$.
K and J are global constant variables, and $P_1, \cdots, P_J$ are global variables.
$i \leftarrow 0, \quad \mathbf{S}_{\mathbf{o}}^{0} \leftarrow \emptyset, \quad \mathbf{S}_{\mathbf{x}}^{0} \leftarrow \emptyset, \quad \mathbf{SS}^{0} \leftarrow \emptyset.$

while $\mathbf{D}_{\mathbf{e}}^{i} \neq \emptyset$ **do**

[1] Calculate **Neighbors_Entropy($\boldsymbol{x}$)** and **Neighbors_Match($\boldsymbol{x}$)** of $\boldsymbol{x} \in \mathbf{D}_{\mathbf{e}}^{i}$.

[2] Choose $\boldsymbol{x}$ satisfying [Spanning Criteria].
$\mathbf{D}_{\mathbf{o}}^{i} \leftarrow \{\boldsymbol{x} \mid Neighbors\_Entropy\,(\boldsymbol{x}) > 0,\ \boldsymbol{x} \in \mathbf{D}_{\mathbf{e}}^{i}\}.$
$\mathbf{D}_{\mathbf{x}}^{i} \leftarrow \mathbf{D}_{\mathbf{e}}^{i} - \mathbf{D}_{\mathbf{o}}^{i}.$

[3] Select $\boldsymbol{x}$ satisfying [Selecting Criteria].
$\mathbf{D}_{\mathbf{s}}^{i} \leftarrow \{\boldsymbol{x} \mid Neighbors\_Match\,(\boldsymbol{x}) > \beta \cdot \frac{1}{J},\ \boldsymbol{x} \in \mathbf{D}_{\mathbf{o}}^{i}\}.$

[4] Update the pattern sets: the spanned, the non-spanned and the selected.
$\mathbf{S}_{\mathbf{o}}^{i+1} \leftarrow \mathbf{S}_{\mathbf{o}}^{i} \cup \mathbf{D}_{\mathbf{o}}^{i}$, $\mathbf{S}_{\mathbf{x}}^{i+1} \leftarrow \mathbf{S}_{\mathbf{x}}^{i} \cup \mathbf{D}_{\mathbf{x}}^{i}$, $\mathbf{SS}^{i+1} \leftarrow \mathbf{SS}^{i} \cup \mathbf{D}_{\mathbf{s}}^{i}$.

[5] Constitute the next evaluation set $\mathbf{D}_{\mathbf{e}}^{\mathbf{i+1}}$.
$\mathbf{D}_{\mathbf{e}}^{i+1} \leftarrow \bigcup_{\boldsymbol{x} \in \mathbf{D}_{\mathbf{o}}^{i}} k\mathbf{NN}(\boldsymbol{x}) - (\mathbf{S}_{\mathbf{o}}^{i+1} \cup \mathbf{S}_{\mathbf{x}}^{i+1}).$

[6] $i \leftarrow i + 1.$

end

return $\mathbf{SS}^{i}$

}

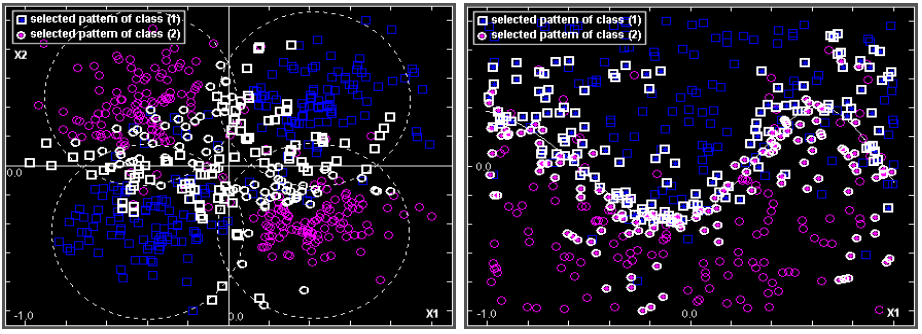
Fig. 3. Selective kNN spanning algorithm

Sine Function Problem: In the second problem, the input patterns were generated from a two-dimensional uniform distribution, and then the class labels were determined by whether the input is located above or below a sine decision function. To make the density near the decision boundary thicker, four different Gaussian noises were added along the decision boundary, i.e., $N(\boldsymbol{\mu}, \boldsymbol{s}^2)$ where $\boldsymbol{\mu}$ is an arbitrary point on the decision boundary and $\boldsymbol{s}$ is a Gaussian width parameter(0.1, 0.3, 0.8, 1.0). Among 500 training patterns, 20% are located in the overlapped region. Therefore, the characteristic of this problem is that the density of patterns increases near the decision boundary.

$$C_1 = \left\{ \boldsymbol{x} \mid x_2 > \sin\left(3x_1 + 0.8\right)^2, \begin{bmatrix} 0 \\ -2.5 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 2.5 \end{bmatrix} \right\},$$

$$C_2 = \left\{ \boldsymbol{x} \mid x_2 \leq \sin\left(3x_1 + 0.8\right)^2, \begin{bmatrix} 0 \\ -2.5 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 2.5 \end{bmatrix} \right\}.$$

Fig. 4(a) and Fig. 4(b) shows the selected patterns against the original ones after normalization ranging from -1 to 1. The value of k was set as 5.



(a) (b)

Fig. 4. The selected patterns against the original ones: (a) Continuous XOR Problem and (b) Sine Function Problem. The selected patterns, shown as outlined circles or squares, are scattered against the original ones.

Only 180(30%) and 264 (52.8%) of the original patterns were selected, respectively. The number of the selected patterns by fast algorithm was identical to that by the naive algorithm. The reduction ratio in Sine Function Problem is rather smaller when it is compared with that of Continuous XOR problem. That is due to the difference in densities near the decision boundary.

A total of 300 test patterns were generated from the statistically identical distribution as its training set for both problems. We built 50 SVMs with all patterns and 50 SVMs with the selected patterns according to every combination of hyper-parameters. Five RBF kernels with different width parameters(σ=0.25, 0.5, 1, 2, 3) and five polynomial kernels with different degree parameters(p=1, 2, 3, 4, 5) were adopted. As for the tolerance parameter, five levels (C= 0.1,

1,10, 100, 1000) were used. This optimal parameter searching is laborious and time-consuming if individual training takes a long time.

SVM performances "with all patterns" (A) and "with the selected patterns" (S) were compared in terms of the test error rate, the execution time, and the number of support vectors. Fig. 5 shows SVM test error rates in all parameter combinations we adopted. In each cell, upper number indicates test error rate for A and lower for S. Gray cell means that S performed better than or equal to A, while white cell means that A performed better. An interesting fact is that SVM performance with S is more sensitive to parameter variation. This phenomenon is shown substantially with ill-posed parameter combination (white cells). For the parameters where SVM with A performs well, SVM with S also works well.

Kernel / C	Rbf (Width Parameter)					Polynomial (Degree Parameter)				
	0.25	0.5	1	2	3	1	2	3	4	5
0.1	11.33	11.67	9.67	25.33	34.33	51.67	10.67	11.67	10.00	10.33
	11.33	29.33	50.00	50.00	50.00	50.00	50.00	32.67	11.67	10.67
1.0	10.00	12.00	11.00	10.67	34.33	34.00	10.00	10.67	11.33	11.33
	10.33	12.00	31.33	50.00	50.00	50.00	15.67	12.33	11.33	11.00
10	10.00	11.33	10.00	12.00	10.00	31.67	9.67	10.33	11.67	12.00
	10.00	10.33	10.00	35.33	50.00	50.00	10.00	9.67	10.33	10.00
100	10.00	11.00	9.67 *	10.67	10.33	31.67	10.00	10.00	11.67	12.33
	9.00	10.33	9.67	10.33	22.67	50.00	9.00	10.00	10.33	10.67
1000	10.67	10.33	11.33	10.33	10.00	31.67	10.33	9.67	12.33	12.67
	11.00	10.67	10.33	10.33	10.00	50.00	9.00	10.00	11.33	11.67

(a)

Kernel / C	Rbf (Width Parameter)					Polynomial (Degree Parameter)				
	0.25	0.5	1	2	3	1	2	3	4	5
0.1	16.33	18.33	20.00	20.67	20.33	20.67	21.00	20.67	16.67	16.00
	16.67	18.33	20.00	33.33	55.33	22.67	19.33	19.67	16.33	15.67
1	13.67	16.33	19.33	20.00	20.67	20.33	19.67	17.33	14.00	13.67
	13.33	15.67	19.33	20.33	22.33	22.00	19.33	16.67	14.00	13.67
10	14.67	13.33	16.67	19.00	20.67	20.33	20.00	15.67	14.00	14.33
	14.67	12.67	14.67	19.67	20.67	21.67	20.00	15.33	13.67	14.33
100	16.67	14.33	13.33	18.33	19.67	20.33	20.00	13.67	13.33 *	14.33
	15.33	13.33	13.67	18.67	19.33	21.67	18.67	13.67	13.33	14.33
1000	18.00	13.33	14.00	15.00	19.00	20.33	20.00	15.00	14.33	14.67
	18.00	13.33	14.33	14.67	17.67	20.33	18.67	12.33	14.33	14.00

(b)

Fig. 5. Test error comparison for SVM experimental results: (a) Continuous XOR Problem and (b) Sine Function Problem. In each cell, upper number indicates test error rate for A and lower for S.

Table 1 compares the best performance of each SVM, A and S. The corresponding parameter combination is marked '*' in Fig. 5. The SVM with the selected patterns was trained much faster thanks to fewer training patterns. Less important support vectors were eliminated in building the margins. Employing fewer SVs results in a smaller recall time. All this happened while the generalization ability was not degraded at al.

Table 1. Best Result Comparison

	Continuous XOR Problem		Sine Function Problem	
	A(All)	S(Selected)	A(All)	S(Selected)
Execution Time (sec.)	454.83	3.85	267.76	8.79
Margin	0.0612	0.0442	0.1904	0.0874
No. of Training Patterns	600	180	500	264
No. of Support Vectors	167	84	250	136
Training Error(%)	10.50	13.89	19.00	19.32
Test Error(%)	9.67	9.67	13.33	13.33

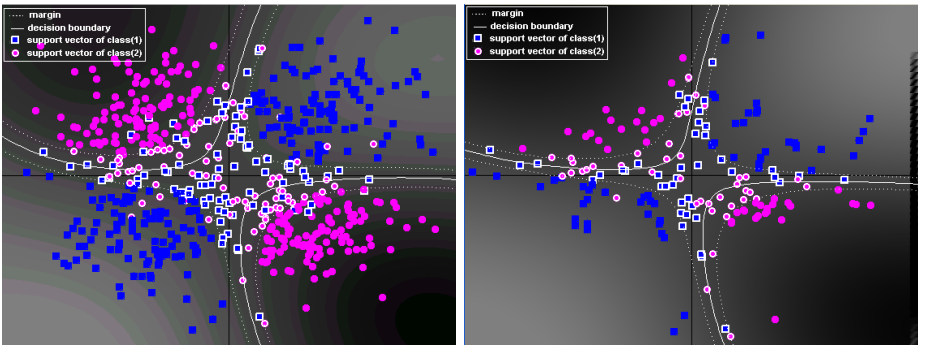
Table 2 shows the execution time from Fig. 5. The average SVM training time of A was 472.20(sec) for Continuous XOR problem and 263.84(sec) for Sine

Function Problem. It is quite comparable 4.45(sec) and 9.13(sec) of S. Pattern selection time of S took 0.88(sec) and 0.66(sec) by the naive algorithm. The proposed fast algorithm reduced the selection time to 0.21(sec) and 0.17(sec), respectively. Therefore, by conducting pattern selection procedure just once, we could shrink SVM training time by one or two orders of magnitude.

Table 2. Execution Time Comparison

		No. of patterns	No. of SVs(avg.)	Pattern Selection CPU time	SVM Avg. CPU time
Continuous XOR	A (All)	600	292.20	-	472.20
	S (Selected)	180	120.72	0.88 (Naive)	4.45
				0.21 (Fast)	
Sine Function	A (All)	500	293.60	-	263.84
	S (Selected)	264	181.76	0.66 (Naive)	9.13
				0.17 (Fast)	

Fig. 6 shows the decision boundaries and margins of the SVMs both with all patterns and with the selected patterns of Continuous XOR problem. (Because of page limitation, we omit the figures for Sine Function problem.) Decision boundary is depicted as a solid line and the margins are defined by the dotted lines in both sides of it. Support vectors are outlined. Note that in the aspects of generalization performance, the decision boundaries are all but similar in both plots.



(a) (b)

Fig. 6. Patterns and decision boundaries between (a) SVM with all patterns and (b) SVM with the selected patterns for continuous XOR Problem.

4.2 Real Problem: MNIST Dataset

For real world data benchmarking test, we used MNIST dataset of handwritten digits from [15]. It consists of 60,000 patterns for training and 10,000 patterns for testing. All binary images are size-normalized and coded by gray-valued 28x28 pixels in the range between -1 and 1, therefore input dimension is 784.

Nine binary classifiers of MNIST were trained: class 8 is paired with each of the rest. We used OSU SVM Classifier Matlab Toolbox since the kernel matrix was too large for the standard QP rountine to fit into memory [15]. A Fifth-order polynomial kernel (p=5), C value of 10, and KKT(Karush-Kuhn-Tucker) tolerance of 0.02 were used same as in [10], and the value of K was set to 50.

The results for the training runs both with all patterns and with the selected patterns are shown in Table 3. First, we compare the results with regard to the number of training patterns and also the number of support vectors. The pattern selection chose an average 16.75% of patterns from the original training sets.

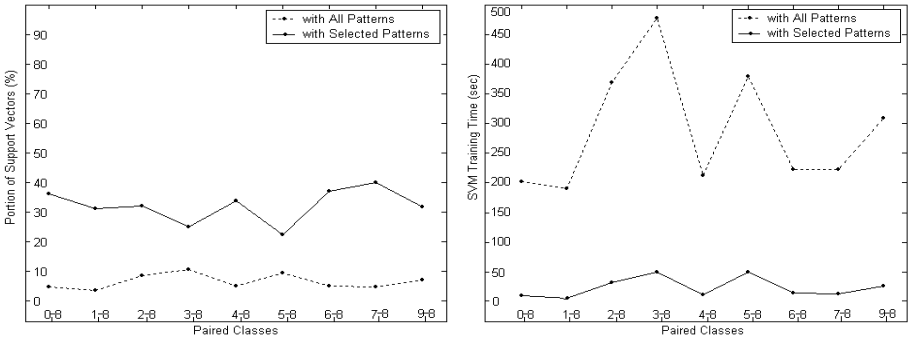
Table 3. MNIST Result Comparison

	Paired Classes								
	0-8	**1-8**	**2-8**	**3-8**	**4-8**	**5-8**	**6-8**	**7-8**	**9-8**
A(All Patterns)									
No. Of Patterns	11774	12593	11809	11982	11693	11272	11769	12116	11800
No. Of SVs	538	455	988	1253	576	1039	594	578	823
SVM Exe. Time	201.74	190.76	368.82	477.25	212.18	379.59	222.84	222.40	308.73
Test Error(%)	0.51	0.09	0.34	0.50	0.10	0.16	0.31	0.10	0.41
S(Selected Patterns)									
No. Of Patterns	1006	812	2589	4089	1138	3959	1135	999	1997
No. Of SVs	364	253	828	1024	383	882	421	398	631
SVM Exe. Time	11.11	5.92	33.03	49.84	12.11	49.74	14.69	13.55	26.61
Pattern Sel. Time	46.11	43.33	78.81	97.89	48.13	93.42	44.27	40.14	59.62
Test Error(%)	0.41	0.95	0.34	0.45	0.15	0.21	0.31	0.18	0.43

When all patterns were used, only 6.44% of the training patterns were used as support vectors. With selected patterns, 32.09% of its training patterns were support vectors. In terms of utilization, the pattern selection did a good job (see Fig. 7(a)).

Second, SVM execution time was significantly reduced from 287.15(sec) to 24.07(sec) on average after the pattern selection procedure. Including the pattern selection time, the total time was, on average, 85.37(sec). Consider that the SVM training is performed several times to find the optimal parameters. On the other hand, the pattern selection procedure is performed only once (see Fig. 7(b)).

Now we finally compare the SVM test error rates between A (with all patterns) and S (with the selected patterns). First, average test error rate over nine classifiers were 0.28% for the former and 0.38% for the latter. But note that most SVM classifiers of S performed all but same as the ones of A except 1-8 classifier. This exception is mainly due to the unique characteristic of digit '1' images. Individual pattern belongs to '1' class is a sparse vector. Only a few fields of a



(a) (b)

Fig. 7. MNIST Result Comparison: (a) The portion of support vectors in training set, and (b) SVM execution time.

vector have significant gray-scaled value, and the rest of the fields have value of '0'. Hence, it might affect the finding the nearest neighbors in opposite class(8 class) during the pattern selection procedure. This conjecture is accompanied by the number of selected patterns of '1' digit class. Only 95 patterns were selected from 6742 patterns in '1' digit class. In the meantime, average test error rate for the rest was 0.30% for A and 0.31% for S.

5 Conclusion

To alleviate the computational burden in SVM training, we proposed a fast pattern selection algorithm which selects the patterns near the decision boundary based on the neighborhood properties. Through several simulations, we obtained promising results: SVM with the selected patterns was trained much faster with fewer redundant support vectors, without any loss in test performance.
The time complexity analysis of the proposed algorithm will be carried out in the near future.

Related to the proposed algorithm, we would like to address two limitations here. First, it is not easy to find the right value for β in [Selecting Criteria]. β controls the selectivity by kNN classifier. In small dimensional space, β can be set to a large value(i.e. $\beta = 1$), since kNN classifier performs well. But in large dimensional space, a large value of β is somewhat risky. Because kNN classifier's accuracy is degraded. Empirically, up to about 20 dimensional space, $\beta = 1$ was not problematic. This coincides with the recommendation in [9].

Second, the proposed algorithm works efficiently under the following circumstances: when the classes are overlapped and when the pattern is stored as a non-sparse vector. Neither condition is met for '1' digit class of MNIST data set. It is known as a linearly separable class from all other digit classes. It is also the most sparse matrix among 10 digit classes [6] [10].

References

[1] Almeida, M.B., Braga, A. and Braga J.P.(2000). SVM-KM: speeding SVMs learning with a priori cluster selection and k-means, *Proc. of the 6th Brazilian Symposium on Neural Networks*, pp. 162–167.

[2] Arya, S., Mount, D.M., Netanyahu, N.S. and Silverman, R., (1998). An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions, *Journal of the ACM*, vol. 45, no. 6, pp. 891–923.

[3] Choi, S.H. and Rockett, P., (2002). The Training of Neural Classifiers with Condensed Dataset, *IEEE Transactions on Systems, Man, and Cybernetics – PART B: Cybernetics*, vol. 32, no. 2, pp.202–207.

[4] Grother, P.J.,Candela, G.T. and Blue, J.L, (1997). Fast Implementations of Nearest Neighbor Classifiers, *Pattern Recognition*, vol. 30, no. 3, pp. 459–465.

[5] Hearst, M.A., Scholkopf, B., Dumais, S., Osuna, E., and Platt, J., (1998). Trends and Controversies - Support Vector Machines, *IEEE Intelligent Systems*, vol. 13, pp. 18–28.

[6] Liu C.L., and Nakagawa M., (2001). Evaluation of Prototype Learning Algorithms for Nearest-Neighbor Classifier in Application to Handwritten Character Recognition, *Pattern Recognition*, vol.34, pp. 601–615.

[7] Lyhyaoui, A., Martinez, M., Mora, I., Vazquez, M., Sancho, J. and Figueiras-Vaidal, A.R., (1999). Sample Selection Via Clustering to Construct Support Vector-Like Classifiers, *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1474–1481.

[8] Masuyama, N., Kudo, M., Toyama, J. and Shimbo, M., (1999). Termination Conditions for a Fast *k*-Nearest Neighbor Method, *Proc. of 3rd International Conference on Knoweldge-Based Intelligent Information Engineering Systems*, Adelaide, Australia, pp. 443–446.

[9] Mitchell, T.M., (1997). *Machine Learning*, McGraw Hill, See also Lecture Slides on Chapter8 for the Book at http://www-2.cs.cmu.edu/ tom/mlbook-chapter-slides.html.

[10] Platt, J.C. (1999). Fast Training of Support Vector Machines Using Sequential Minimal Optimization, *Advances in Kernel Methods: Support Vector Machines*, MIT press, Cambridge, MA, pp. 185–208.

[11] Shin, H.J. and Cho, S.Z., (2002). Pattern Selection For Support Vector Classifiers, *Proc. of the 3rd International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*, Manchester, UK, pp. 469–474.

[12] Shin, H.J. and Cho, S.Z., (2002). Pattern Selection Using the Bias and Variance of Ensemble, *Journal of the Korean Institute of Industrial Engineers*, vol. 28, no. 1, pp. 112–127, 2002.

[13] Short, R., and Fukunaga, (1981). The Optimal Distance Measure for Nearest Neighbor Classification, *IEEE Transactions on Information and Theory*, vol. IT–27, no. 5, pp. 622–627.

[14] Vapnik, V., (2000). *The Nature of Statistical Learning Theory*, Springer-Verlag New York, Inc. 2nd eds.

[15] http://www.kernel-machines.org/