Fast Pattern Selection Algorithm for Support Vector Classifiers: Time Complexity Analysis

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

Sungzoon Cho
Department of Industrial Engineering, Seoul National University,
San 56-1, Shillim-Dong, Kwanak-Gu, 151-742, Seoul, Korea
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. The time complexity of the proposed algorithm is much smaller than that of the naive M^2 algorithm.

1. Introduction

In SVM QP formulation, the dimension of kernel matrix ($M \times M$) is equal to the number of training patterns (M). A standard QP solver has time complexity of order $O(M^3)$: 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: Chunking, SMO, SVM^{light} and SOR [5][8]. The general time complexity of those methods is approximately (the number of iterations) $\cdot O(Mq+q^3)$ where q is the size of the working set. Of course, "the number of iterations" is supposed to increase as M increases.

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. Therefore, selecting those patterns (potential support vectors) prior to SVM training is quite desirable (see Fig. 1).

Up to now, there have been considerable researches about pattern selection near the decision boundary. Shin and Cho selected the clean patterns near the decision boundary based on the bias and variance of outputs of a network ensemble [10]. Lyhyaoui et al. implemented R classifiers, somewhat like SVMs, by selecting patterns near the decision boundary. They proposed *1*-nearest neighbor method in opposite class after class-wise clustering [6]. To lessen the burden of the MLP training, Choi and Rockett used *k*NN 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]. Yet another pattern selection approach, focusing mostly on SVMs, was proposed by Almeida et al [1]. They conducted k-means clustering on the entire training set. All patterns were selected for heterogeneous clusters, while only centroids were selected for homogeneous clusters.

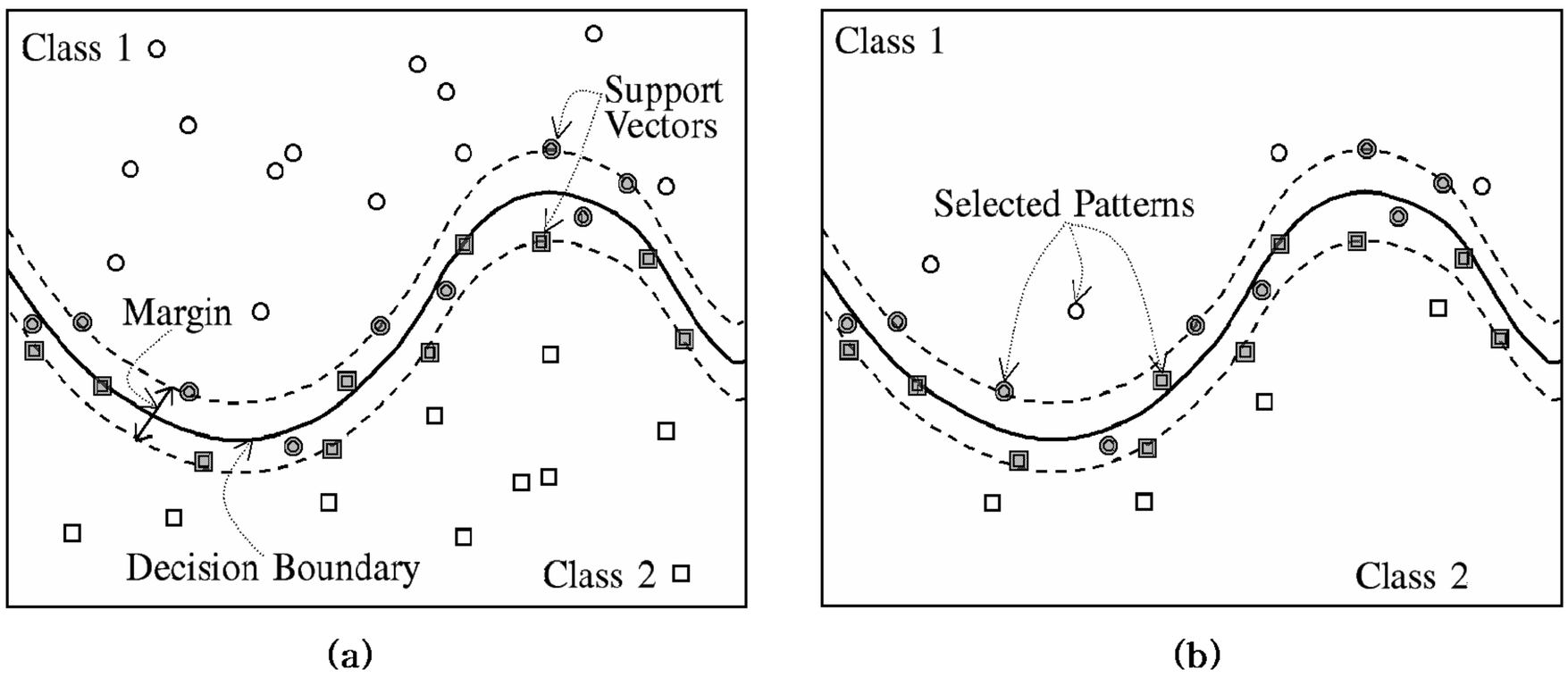


Fig. 1. Pattern selection: a large training set shown in (a) is condensed to a small training set (b) which is composed of only potential support vectors.

Recently, we proposed to select the patterns near the decision boundary based on the neighborhood properties [9]. We utilized k nearest neighbors to look around the pattern's periphery. *The first property* dictates that a pattern located near the decision boundary tends to have more heterogeneous neighbors. The degree of proximity to the decision boundary can be estimated by neighbors' label entropy. A larger entropy value indicates that the pattern is close to the decision boundary. *The second property* dictates that a pattern on the correct side of the decision boundary tends to belong to the same class as its neighbors. The ratio of the neighbors whose label matches that of the pattern can be used as an estimate on how noisy the pattern is. In other words, the smaller the ratio value, the more likely the pattern is noisy. Thus, among the patterns with a positive entropy value, potential noisy patterns are identified and then eliminated. The approach reduced the number of patterns significantly, thus reduced the training time. However, a naive algorithm evaluating kNNs for all patterns took $O(M^2)$, so the pattern selection process itself was time consuming.

In this paper, we propose a fast algorithm. Here, we just compute the kNNs of the patterns near the decision boundary, not all training patterns. The idea comes from another 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 approximately $O(bM)$, where b is the number of patterns in the "overlap" region around decision boundary. In most practical problems, $b \ll M$ holds.

This paper is structured as follows. In section 2, we propose the fast algorithm which selects the patterns near the decision boundary. In section 3, we provide the time complexity analysis of the algorithm. In section 4, we present the empirical results confirming the time complexity of our algorithm. In the last section, we conclude the paper with the discussion of the limitations.

2. Fast Algorithm based on Neighborhood Properties

The first neighborhood property is that a pattern located near the decision boundary tends to have heterogeneous neighbors. Thus, the degree of pattern $\vec{x}$'s proximity to the decision boundary can be estimated by "**Neighbor_Entropy($\vec{x}$)**", which is defined as the entropy of pattern $\vec{x}$'s k-nearest neighbors' class labels (see Fig. 2). A pattern with a positive Neighbor_Entropy($\vec{x}$) 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 correct side of the decision boundary tends to belong to the same class as its neighbors. If a pattern's own label is very different from those of its neighbors, it is likely to be incorrectly labeled. The measure "**Neighbor_Match($\vec{x}$)**" is defined as the ratio of $\vec{x}$'s neighbors whose label matches that of $\vec{x}$. Only those pattern $\vec{x}$s are selected that satisfy Neighbor_Match($\vec{x}$) $\geq \beta \cdot 1/J$ (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 the following selecting criteria.

[Selecting Criteria] Neighbor_Entropy($\vec{x}$) > 0 and Neighbor_Match($\vec{x}$) $\geq \beta \cdot 1/J$

LabelProbability($\vec{x}$) {
/* For $\vec{x}$, calculate the label probabilities of kNN($\vec{x}$) over J classes, $\{C_1, C_2, \ldots, C_J\}$, where kNN($\vec{x}$) is defined as the set of k nearest neighbors of $\vec{x}$. */
$k_j = |\{\vec{x}' \in C_j \mid \vec{x}' \in \boldsymbol{kNN}(\vec{x})\}|, \quad j = 1, \cdots, J.$
return $\left(P_j = \frac{k_j}{k}, \forall j\right)$.
}

Neighbor_Entropy($\vec{x}$) {
/* Calculate the neighbors-entropy of $\vec{x}$ with its nearest neighbors' labels. In all calculations, $0 \log_J \frac{1}{0}$ is defined to be 0. */
Do **LabelProbability($\vec{x}$)**.
return $\left(\sum_{j=1}^{J} P_j \cdot \log_J \frac{1}{P_j}\right)$.
}

Neighbor_Match($\vec{x}$) {
/* Calculate the neighbors-match of $\vec{x}$. j^* is defined as the label of $\vec{x}$ itself. */
$j^* = \arg_j \{C_j \mid \vec{x} \in C_j, \quad j = 1, \cdots, J\}$.
Do **LabelProbability($\vec{x}$)**.
return (P_{j^*}).
}

Fig. 2. Neighbor_Entropy and Neighbor_Match functions

A naive algorithm was presented in [9] where the kNNs of all patterns were evaluated. This algorithm is easy to implement and also runs in a reasonable amount of time as long as the size of training set, M, is relatively small. However, when the size of training set is large, the computational

cost increases in proportion to the size. Let us assume that the distance between any two points in d-dimensional space can be computed in $O(d)$. Then finding the nearest neighbors for each pattern takes sum of distance computation time ("DT") and search time ("ST") (see table 1). The total time complexity of the naive algorithm, therefore, is $O(M \cdot (DT+ST))$. Roughly speaking, it is $O(M^2)$.

Table 1. DT and ST

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

There is a considerable amount of literature on efficient nearest neighbor searching algorithms for large data sets of a high dimension. Most approaches focus on reducing DT or ST. See [4] and [11] for reducing DT, and [2] and [7] for reducing ST. Our approach, on the other hand, focuses on reducing the first M of $O(M \cdot M)$. The idea comes from yet another neighborhood property that the neighbors of a pattern located near the decision boundary tend to be located near the decision boundary as well. Given a set of randomly selected patterns, we examine the patterns near the decision boundary and their neighbors only. This successive "neighbors" only evaluation of the "current" pattern set is repeated until all the patterns near the decision boundary are chosen and evaluated. A pattern is "expanded" or a pattern's neighbors are evaluated when its Neighbor_Entropy is positive. This "selective kNN expanding" procedure is shown in Fig. 3 using notations displayed in table 2.

[Expanding Criteria] Neighbor_Entropy($\vec{x}$) > 0

Table 2. Notation

Symbol	Meaning
$\boldsymbol{D}$	the original training set whose cardinality is M
$\boldsymbol{D_e}^i$	the evaluation set at i-th step
$\boldsymbol{D_o}^i$	a subset of $\boldsymbol{D_e}^i$, the set of patterns to be "expanded" from $\boldsymbol{D_e}^i$, each element of which will compute its k nearest neighbors to constitute the next evaluation set, $\boldsymbol{D_e}^{i+1}$
$\boldsymbol{D_x}^i$	a subset of $\boldsymbol{D_e}^i$, the set of patterns "not to be expanded" from $\boldsymbol{D_e}^i$, $\boldsymbol{D_x}^i = \boldsymbol{D_e}^i - \boldsymbol{D_o}^i$
$\boldsymbol{D_s}^i$	the set of "selected" patterns from $\boldsymbol{D_o}^i$ at i-th step
$\boldsymbol{S_o}^i$	the accumulated set of expanded patterns, $\bigcup_{j=0}^{i-1} \boldsymbol{D_o}^j$
$\boldsymbol{S_x}^i$	the accumulated set of non-expanded patterns, $\bigcup_{j=0}^{i-1} \boldsymbol{D_x}^j$
$\boldsymbol{SS}^i$	the accumulated set of selected patterns, $\bigcup_{j=0}^{i-1} \boldsymbol{D_s}^j$, the last of which $\boldsymbol{SS}^N$ is the reduced training pattern set
$\boldsymbol{kNN}(\vec{x})$	the set of k nearest neighbors of $\vec{x}$
$\boldsymbol{B}$	the set of patterns located in the "overlapped" region characterized by Neighbor_Entropy $(\vec{x}) > 0$
$\boldsymbol{B}^+$	the set of k nearest neighbors of patterns belonging to $\boldsymbol{B}$

Selective-*k*NN-Expanding( ) {

[0] Initialize $D_e^{\,i}$ with randomly chosen patterns from D.

Constants k and J are given. Initialize i and various sets as follows:

$i \leftarrow 0, \quad S_o^{\,0} \leftarrow \varnothing, \quad S_x^{\,0} \leftarrow \varnothing, \quad SS^0 \leftarrow \varnothing.$

while $D_e^{\,i} \neq \varnothing$ **do**

[1] Choose $\vec{x}$ satisfying [Expanding Criteria].

$D_o^{\,i} \leftarrow \{\vec{x} \mid \text{Neighbor\_Entropy}(\vec{x}) > 0, \ \vec{x} \in D_e^{\,i}\}.$

[2] Select $\vec{x}$ satisfying [Selecting Criteria].

$D_s^{\,i} \leftarrow \{\vec{x} \mid \text{Neighbor\_Match}(\vec{x}) \geq \beta/J, \ \vec{x} \in D_o^{\,i}\}$

[3] Update the pattern sets: the expanded, the non-expanded, and the selected.

$S_o^{\,i+1} \leftarrow S_o^{\,i} \cup D_o^{\,i}, \quad S_x^{\,i+1} \leftarrow S_x^{\,i} \cup D_x^{\,i}, \quad SS^{i+1} \leftarrow SS^i \cup D_s^{\,i}.$

[4] Compute the next evaluation set $D_e^{\,i+1}$

$D_e^{\,i+1} \leftarrow \bigcup_{\vec{x} \in D_o^{\,i}} kNN(\vec{x}) - (S_o^{\,i+1} \cup S_x^{\,i+1}).$

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

end

return SS^i

}

Fig. 3. Selective-*k*NN-Expanding algorithm

3. The Time Complexity Analysis of the Fast Algorithm

In this section, we show that the proposed algorithm terminates within a finite number of steps and that its time complexity is significantly smaller than that of the naive algorithm.

Lemma 1. Different evaluation sets are disjoint:

$$D_e^{\,i} \cap D_e^{\,j} = \varnothing, \quad \forall i \neq j. \tag{1}$$

proof. Consider step [4] of the algorithm shown in Fig. 3,

$$D_e^{\,i} = \left(\bigcup_{\vec{x} \in D_o^{\,i-1}} kNN(\vec{x}) \right) - (S_o^{\,i} \cup S_x^{\,i}). \tag{2}$$

Since $S_o^{\,i}$ and $S_x^{\,i}$ are defined as $\left(\bigcup_{j=0}^{i-1} D_o^{\,j} \right)$ and $\left(\bigcup_{j=0}^{i-1} D_x^{\,j} \right)$ respectively, their union results in

$$\boldsymbol{S_o}^{i} \cup \boldsymbol{S_x}^{i} = \bigcup_{j=0}^{i-1} \left(\boldsymbol{D_o}^{i} \cup \boldsymbol{D_x}^{i} \right) = \left(\bigcup_{j=0}^{i-1} \boldsymbol{D_e}^{j} \right). \tag{3}$$

By replacing $\left(\boldsymbol{S_o}^{i} \cup \boldsymbol{S_x}^{i} \right)$ in Eq.(2) with Eq.(3), we get

$$\boldsymbol{D_e}^{i} = \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{i-1}} \boldsymbol{kNN}(\vec{x}) \right) - \left(\bigcup_{j=0}^{i-1} \boldsymbol{D_e}^{j} \right). \tag{4}$$

Eq.(4) clearly shows that $\boldsymbol{D_e}^{i}$ does not share patterns with any of its earlier sets $\boldsymbol{D_e}^{j}$, $j=0, \cdots, i-1$. ¶

Lemma 2. The union of all $\boldsymbol{D_e}^{i}$'s is equivalent to the set of kNN's of the union of all $\boldsymbol{D_o}^{i}$'s.

$$\left(\bigcup_{i=1}^{n} \boldsymbol{D_e}^{i} \right) = \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{0} \cup \boldsymbol{D_o}^{1} \cup \cdots \cup \boldsymbol{D_o}^{n-1}} \boldsymbol{kNN}(\vec{x}) \right). \tag{5}$$

proof. From Eq.(4) in Lemma (1), we get

$$\bigcup_{i=1}^{n} \boldsymbol{D_e}^{i} = \bigcup_{i=1}^{n} \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{i-1}} \boldsymbol{kNN}(\vec{x}) \right) - \bigcup_{i=1}^{n} \left(\bigcup_{j=0}^{i-1} \boldsymbol{D_e}^{j} \right). \tag{6}$$

Since in general

$$\left(\bigcup_{\vec{x} \in A_1} \boldsymbol{kNN}(\vec{x}) \right) \cup \left(\bigcup_{\vec{x} \in A_2} \boldsymbol{kNN}(\vec{x}) \right) = \left(\bigcup_{\vec{x} \in A_1 \cup A_2} \boldsymbol{kNN}(\vec{x}) \right) \tag{7}$$

holds, we get

$$\left(\bigcup_{i=1}^{n} \boldsymbol{D_e}^{i} \right) = \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{0} \cup \boldsymbol{D_o}^{1} \cup \cdots \cup \boldsymbol{D_o}^{n-1}} \boldsymbol{kNN}(\vec{x}) \right) - \left(\bigcup_{i=0}^{n-1} \boldsymbol{D_e}^{i} \right). \tag{8}$$

Eq.(8) can be rewritten in the form of

$$\left(\bigcup_{i=1}^{n} \boldsymbol{D_e}^{i} \right) = \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{0} \cup \boldsymbol{D_o}^{1} \cup \cdots \cup \boldsymbol{D_o}^{n-1}} \boldsymbol{kNN}(\vec{x}) \right) \cap \left(\bigcup_{i=0}^{n-1} \boldsymbol{D_e}^{i} \right)^{C}. \tag{9}$$

If we union $\left(\bigcup_{i=0}^{n-1} \boldsymbol{D_e}^{i} \right)$ to both sides of Eq.(9), then

$$\left(\bigcup_{i=0}^{n} \boldsymbol{D_e}^{i} \right) = \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{0} \cup \boldsymbol{D_o}^{1} \cup \cdots \cup \boldsymbol{D_o}^{n-1}} \boldsymbol{kNN}(\vec{x}) \right) \cup \left(\bigcup_{i=0}^{n-1} \boldsymbol{D_e}^{i} \right) \tag{10}$$

results. Since $\boldsymbol{D_e}^{i} \subseteq \bigcup_{\vec{x} \in \boldsymbol{D_o}^{i-1}} \boldsymbol{kNN}(\vec{x})$, $i=1, \cdots, n$, $\left(\bigcup_{i=1}^{n-1} \boldsymbol{D_e}^{i} \right)$, the last $n-1$ components of the second factor of the right hand side may vanish. Then, we finally have

$$\left(\bigcup_{i=1}^{n} \boldsymbol{D_e}^{i} \right) \cup \boldsymbol{D_e}^{0} = \left(\bigcup_{\vec{x} \in \boldsymbol{D_o}^{0} \cup \boldsymbol{D_o}^{1} \cup \cdots \cup \boldsymbol{D_o}^{n-1}} \boldsymbol{kNN}(\vec{x}) \right) \cup \boldsymbol{D_e}^{0}. \tag{11}$$

If we consider only the relationship after the first iteration, then { D_{ e}^{0}} from both sides of Eq.(11) is not to be included. Now, the lemma is proved. ¶

Lemma 3. Every expanded set $\boldsymbol{D}_o^{\,i}$ is a subset of $\boldsymbol{B}$, the set of patterns in the overlapped region.

$$\boldsymbol{D}_o^{\,i} \subseteq \boldsymbol{B}, \ \forall i. \tag{12}$$

proof. Recall that in the proposed algorithm, $\boldsymbol{D}_o^{\,i}$ is defined as

$$\boldsymbol{D}_o^{\,i} = \{\vec{x} \mid \text{Neighbor\_Entropy}(\vec{x}) > 0, \ \vec{x} \in \boldsymbol{D}_e^{\,i}\}. \tag{13}$$

Compare it with the definition of $\boldsymbol{B}$

$$\boldsymbol{B} = \{\vec{x} \mid \text{Neighbor\_Entropy}(\vec{x}) > 0, \ \vec{x} \in \boldsymbol{D}\}. \tag{14}$$

Since $\boldsymbol{D}_e^{\,i}$'s are subsets of $\boldsymbol{D}$, $\boldsymbol{D}_o^{\,i}$'s are subsets of $\boldsymbol{B}$. ¶

Lemma 4. Different expanded sets $\boldsymbol{D}_o^{\,i}$'s are disjoint.

$$\boldsymbol{D}_o^{\,i} \bigcap \boldsymbol{D}_o^{\,j} = \varnothing, \ \forall i \neq j. \tag{15}$$

proof. Every expanded set is a subset of the evaluation set by definition (see step[1] in the algorithm)

$$\boldsymbol{D}_o^{\,i} \subseteq \boldsymbol{D}_e^{\,i}, \ \forall i. \tag{16}$$

By Lemma 1, $\boldsymbol{D}_e^{\,i}$'s are disjoint from others for all i's. Therefore, their respective subsets are disjoint, too. ¶

Theorem 1. (Termination of the Algorithm)
If the while loop of the proposed algorithm exits after N iterations, then N is finite.

proof. We show that $N < \infty$. Inside the while-loop of the algorithm (Fig. 3), condition $\boldsymbol{D}_e^{\,i+1} \neq \varnothing$ holds. Therefore, $\boldsymbol{D}_o^{\,i} \neq \varnothing$, $i = 0, \cdots, N-1$. That means $n(\boldsymbol{D}_o^{\,i}) \geq 1$, $i = 0, \cdots, N-1$. Since $\boldsymbol{S}_o^{\,i}$ is defined as $\bigcup_{j=0}^{i-1} \boldsymbol{D}_o^{\,j}$, and $\boldsymbol{D}_o^{\,j}$'s are disjoint (Lemma 4), we get

$$n(\boldsymbol{S}_o^{\,i}) = \sum_{j=0}^{i-1} n(\boldsymbol{D}_o^{\,j}). \tag{17}$$

Since $n(\boldsymbol{D}_o^{\,i}) \geq 1$, $i = 0, \cdots, N-1$, $n(\boldsymbol{S}_o^{\,i})$ is monotonically increasing. In the meantime, the union of all the $\boldsymbol{D}_o^{\,j}$'s generated in the while loop is bounded by $\boldsymbol{B}$ (Lemma 3). So, we obtain

$$\bigcup_{j=0}^{N-1} \boldsymbol{D}_o^{\,j} \subseteq \boldsymbol{B}. \tag{18}$$

Now, Lemma 4 leads us to

$$\sum_{j=0}^{N-1} n(\boldsymbol{D}_o^{\,j}) \leq n(\boldsymbol{B}). \tag{19}$$

Combination of Eq.(17) and Eq.(19) results in

$$n(\boldsymbol{S}_o^N) \leq n(\boldsymbol{B}). \tag{20}$$

Since $n(\boldsymbol{B}) \ll M$ and finite, $n(\boldsymbol{S}_o^N)$ is finite. Thus, N is finite.¶

Theorem 2. (Termination of the Algorithm)

The number of patterns whose kNNs are evaluated is $(r \cdot n(\boldsymbol{B}^C) + n(\boldsymbol{B}^+))$, where $\boldsymbol{B}^C$ is the complement set of $\boldsymbol{B}$ or $\boldsymbol{D}-\boldsymbol{B}$, and r is the proportion of initial random sampling, $(0 < r < 1)$.

proof. The number of patterns whose kNNs are evaluated is denoted as $\sum_{i=0}^{N} n(\boldsymbol{D}_e^i)$. Let us first consider cases from $i=1$ to N. We have

$$\begin{aligned}
\sum_{i=1}^{N} n(\boldsymbol{D}_e^i) &= n\left(\bigcup_{i=1}^{N} \boldsymbol{D}_e^i\right) && \text{by Lemma 1} \\
&= n\left(\bigcup_{\vec{x} \in \boldsymbol{D}_o^0 \cup \boldsymbol{D}_o^1 \cup \cdots \cup \boldsymbol{D}_o^{N-1}} \boldsymbol{kNN}(\vec{x})\right) && \text{by Lemma 2} \\
&\leq n\left(\bigcup_{\vec{x} \in \boldsymbol{B}} \boldsymbol{kNN}(\vec{x})\right) && \text{by Lemma 3} \\
&= n(\boldsymbol{B}^+).
\end{aligned} \tag{21}$$

Let us include the case of $i=0$.

$$\sum_{i=0}^{N} n(\boldsymbol{D}_e^i) \leq n(\boldsymbol{D}_e^0) + n(\boldsymbol{B}^+), \tag{22}$$

where $n(\boldsymbol{D}_e^0)$ is approximately $r \cdot n(\boldsymbol{D})$ because $\boldsymbol{D}_e^0$ is randomly chosen from $\boldsymbol{D}$. In the meantime, some patterns of $\boldsymbol{D}_e^0$ are already counted in $n(\boldsymbol{B}^+)$. The number of those pattern amounts to $n(\boldsymbol{D}_o^0)$ since $\boldsymbol{D}_o^0 = \{\vec{x} \mid \text{Neighbor\_Entropy}(\vec{x}) > 0,\ \vec{x} \in \boldsymbol{D}_e^0\}$ and $\boldsymbol{D}_o^0 \subseteq \boldsymbol{B} \subseteq \boldsymbol{B}^+$. To get a tighter bound, therefore, we exclude the duplicated count from Eq.(22):

$$\sum_{i=0}^{N} n(\boldsymbol{D}_e^i) \leq n(\boldsymbol{D}_e^0 - \boldsymbol{D}_o^0) + n(\boldsymbol{B}^+), \tag{23}$$

where $n(\boldsymbol{D}_e^0 - \boldsymbol{D}_o^0)$ denotes the number of the patterns which do not belong to $\boldsymbol{B}$. It amounts

$$\begin{aligned}
n(\boldsymbol{D}_e^0 - \boldsymbol{D}_o^0) &= n(\boldsymbol{D}_e^0) - n(\boldsymbol{D}_o^0) \\
&\approx n(\boldsymbol{D}_e^0) - n(\boldsymbol{D}_e^0) \cdot \frac{n(\boldsymbol{B})}{n(\boldsymbol{D})} \\
&= r \cdot n(\boldsymbol{D}) - r \cdot n(\boldsymbol{D}) \cdot \frac{n(\boldsymbol{B})}{n(\boldsymbol{D})} \\
&= r \cdot n(\boldsymbol{B}^C),
\end{aligned} \tag{24}$$

where $\boldsymbol{D}_o^0 \subseteq \boldsymbol{D}_e^0$. Thus, we get the following bound by Eq.(21) and Eq.(24):

$$\sum_{i=0}^{N} n(\boldsymbol{D}_e^i) \leq r \cdot n(\boldsymbol{B}^C) + n(\boldsymbol{B}^+). \tag{25}$$

¶

The time complexity of the fast algorithm is $(r \cdot b^C + b^+) \cdot M$ where $b^C = n(\boldsymbol{B}^C)$ and $b^+ = n(\boldsymbol{B}^+)$. Practically, b^C is almost as large as M, i.e., $b^C \approx M$. But the initial sampling ratio r is usually quite small, i.e., $r \ll 1$. Thus the first term $r \cdot b^C M$ may be assumed to be insignificant. In most real world problems, b^+ is just slightly larger than b, thus the second term $b^+ M$ can be approximated to bM. In short, $(r \cdot b^c + b^+)M$ can be simplified as bM, which of course is much smaller than M^2 since $b \ll M$.

Fig. 4 depicts the theoretical relationship between the computation time of the fast algorithm (with that of the naive algorithm) and b. The computation time of the naive algorithm M^2does not change as long as M is fixed. Meanwhile, that of the fast algorithm is linearly dependent on b. Therefore, the fast algorithm is always faster than the naive algorithm except the case of $b \approx M$.

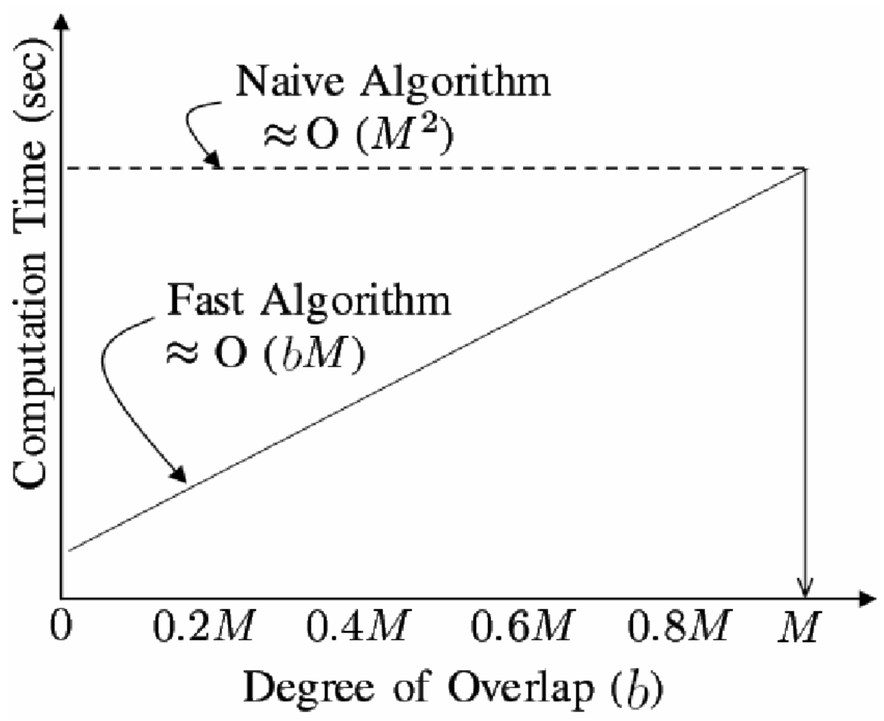


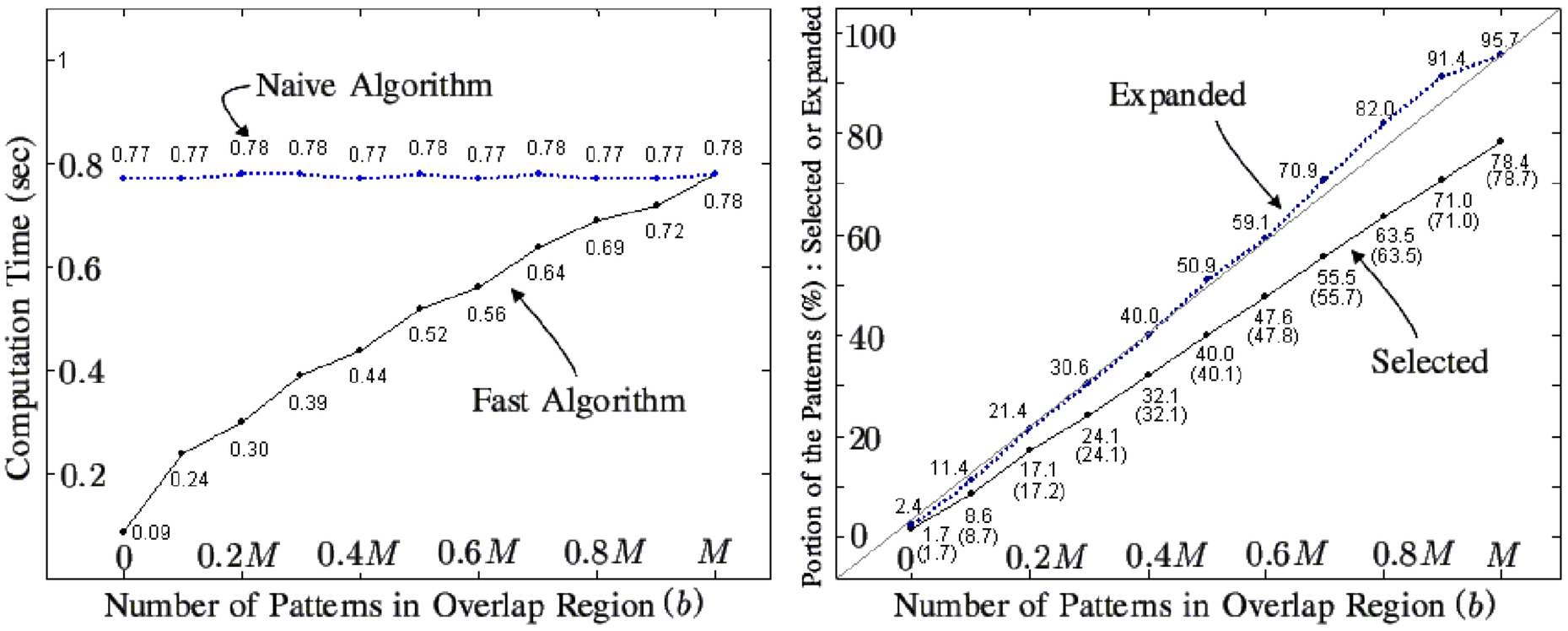
Fig. 4. Theoretical relationship between the computation time and b.

4. Experimental Results

The fast algorithm runs in bM, roughly speaking. We now show whether the complexity stands in the practical situations through experiments. A total of M patterns, half of M from each class, were randomly generated from a pair of two-dimensional uniform distributions:

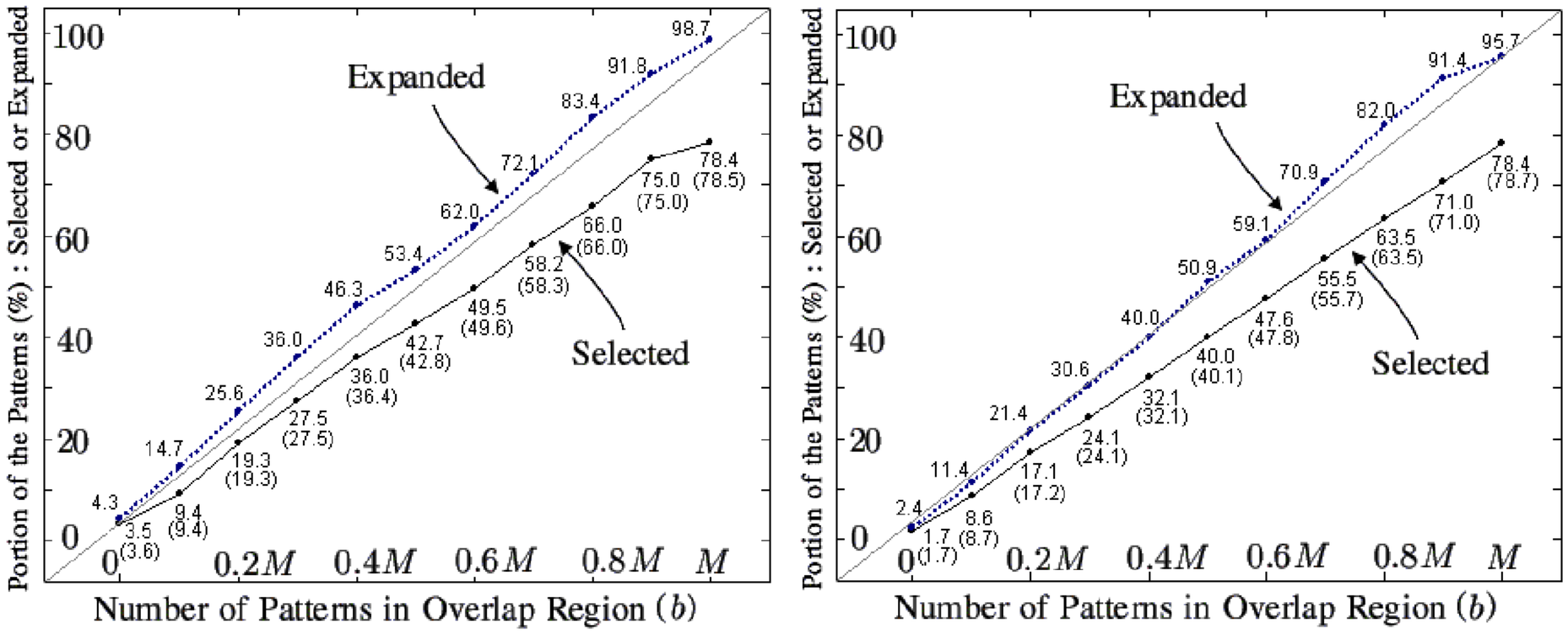
$$C_j = \left\{ \vec{x} \mid U\left(\left\lfloor \left(\frac{-1+(-1)^{j+1}}{2} + \frac{(-1)^j b}{2M} \right) \right\rfloor < \vec{x} < \left\lfloor \left(\frac{1+(-1)^{j+1}}{2} + \frac{(-1)^j b}{2M} \right) \right\rfloor \right) \right\}, \quad j = 1, 2.$$

We set b to every decile of M, i.e. $b = 0, 0.1M, 0.2M, \cdots, 0.9M, M$. Fig. 5 shows the actual computation time for various values of b when (a) $M = 1,000$ and (b) $M = 10,000$, respectively. They clearly show that computation time is exactly proportional to b. Compare with the Naive algorithm's computation time that is constant regardless of b. Fig. 6 also gives almost identical pictures: the percentage of selected and expanded patterns is proportional to b.



(a) M=1,000 **(b)** M=10,000

Fig. 5. Actual computation time for various values of b.



(a) M=1,000 **(b)** M=10,000

Fig. 6. Actual percentage of expanded/selected patterns for various values of b.

5. Conclusion

We proposed a fast pattern selection algorithm which evaluates and selects only the patterns near the decision boundary based on the neighborhood properties. We also proved that it roughly takes $O(bM)$ time and empirically confirmed it.

Currently, there are two limitations. First, the proposed algorithm was developed under the assumption that the classes are overlapped (a non-separable case). Therefore, if one class is remote and clearly separable from the other, an empty set will be returned as a selected pattern set. Of course, such cases rarely happen in real world applications. Second, the number of neighbors, k, was empirically set to 4 in the experiment. A more scientific method needs to be employed. It is currently under investigation.

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 Silvean, 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] 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.

[7] 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 Knowledge-Based Intelligent Information Engineering Systems*, Adelaide, Australia, pp. 443-446.

[8] 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.

[9] Shin, H. J. and Cho, S., (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.

[10] Shin, H. J. and Cho, S., (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.

[11] 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.