Training Support Vector Machines via Adaptive Clustering

Daniel Boley* Dongwei Cao†

Abstract

Training support vector machines involves a huge optimization problem and many specially designed algorithms have been proposed. In this paper, we proposed an algorithm called *ClusterSVM* that accelerates the training process by exploiting the distributional properties of the training data, that is, the natural clustering of the training data and the overall layout of these clusters relative to the decision boundary of support vector machines. The proposed algorithm first partitions the training data into several pair wise disjoint clusters. Then, the representatives of these clusters are used to train an initial support vector machine, based on which we can approximately identify the support vectors and non-support vectors. After replacing the cluster containing no support vectors with its representative, the number of training data can be significantly reduced, thereby speeding up the training process. The proposed *ClusterSVM* has been tested against the popular training algorithm SMO on both the artificial data and the real data, and a significant speedup was observed. The complexity of *ClusterSVM* scales with the square of the number of support vectors and, after a further improvement, it is expected that it will scales with square of the number of non-boundary support vectors.

1 Introduction

Support vector machines (SVM) (Vapnik [19]) have been successfully applied in a variety of domains, including handwritten digit recognition, text document classification and microarray data analysis. In training these SVMs, one needs to maximize a convex objective function subjecting to box constraints. This kind of optimization problem has been extensively studied and many software packages have been developed. However, the off-the-shelf packages typically require the entire Gram matrix be stored in the main memory and, knowing the fact that the size of the Gram matrix scales with the square of the number of training data, the memory requirement of these packages quickly makes them impractical even for a moderate problem [6]. Thus, many specially tailored optimization algorithms have been proposed. The first class of such algorithms tries to solve the entire optimization problem by solving a series of small problems. The basic techniques include chunking and decomposition, which were discussed by Boser et al. [4], Osuna et al. [15], Kaufman et al. [11] and Joachims [10]. Especially noteworthy is the SMO (Sequential Minimal Optimization) algorithm by Platt [16] that sequentially optimizes over a subset of size two, for which we can perform the optimization analytically. The success of these algorithms depends on an appropriate criterion for the active set selection and an efficient strategy to cache the Gram matrix. A second class of algorithms tries to approximate the Gram matrix by a smaller matrix either using the low-rank representation (Fine et al. [9]) or by sampling (Williams et al. [20], Achlioptas et al. [1]), thereby reducing the size of the optimization problem and speeding up the training process. However, the price for such speedup is some loss of optimality. Keerthi et al. [12] proposed an algorithm based on observations about the geometrical properties of support vector machines.

In this paper, we proposed a fast training algorithm called *ClusterSVM* whose idea is to speed up the training process by reducing the number of training data. This is accomplished by partitioning the training data into pair wise disjoint clusters, each of which consists of either only support vectors or only non-support vectors, and replacing the cluster containing only non-support vectors by a representative. In order to identify the cluster that contains only non-support vectors, the training data is first partitioned into several pair wise disjoint clusters and an initial support vector machine is trained using the representatives of these clusters. Based on this initial SVM, we can judge whether a cluster contains only non-support vectors or not. For the cluster that contains both support vectors and non-support vectors, based on the decision boundary of the initial SVM, we can split it into two subclusters that approximately contain either only non-support vectors or only support vectors. This process is then repeated if one of the subclusters contains both support vectors and non-support vectors. The training time of this strategy scales with the square of the number of support vectors and, as shown by experiments, an approximate can be found even faster.

*Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455. Email: boley@cs.umn.edu

†Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455. Email: dcao@cs.umn.edu

Further, based on the theory underlying *ClusterSVM*, it is expected that the training time will scale with the number of boundary support vectors after some straightforward extensions to the current work.

The rest of the paper is organized as follows. Section 2 briefly introduces the optimization problem involved in training SVM, followed by the theoretical results underlying *ClusterSVM*. In section 3, the experimental results were reported on both the artificial data and the real data. Finally, section 4 concludes the paper with further research topics.

2 ClusterSVM

2.1 Support vector machines In a two-class classification problem, given a training data set $\mathcal{D}$ of size n

$$\mathcal{D} = \left\{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathcal{R}^N,\ y_i \in \{1, -1\}\right\} \tag{2.1}$$

where $i = 1, 2, \cdots, n$ and y_i indicates the class membership of the object i represented by vector $\mathbf{x}_i$, the support vector classifier $f(\mathbf{x})$ is defined as [19]

$$f(\mathbf{x}) = sign\left(d(\mathbf{x})\right) = \begin{cases} 1 & : \quad d(\mathbf{x}) \geq 0 \\ -1 & : \quad d(\mathbf{x}) < 0 \end{cases} \tag{2.2}$$

where $d(\cdot)$ is call the functional margin and it is defined as

$$d(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle_{\mathcal{H}} + b \tag{2.3}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the dot product of the reproducing kernel Hilbert space $\mathcal{H}$ generated by a symmetric positive definite kernel $K(\cdot, \cdot)$ satisfying the Mercel condition, and $\Phi(\cdot)$ is the mapping associated with $K(\cdot, \cdot)$ [19]. The optimal parameter $\mathbf{w}^*$ and b^* corresponding to the optimal classifier $f^*(\mathbf{x})$ can be obtained by solving the following optimization problem [19]

$$\text{Minimize} : g(\mathbf{w}, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i \tag{2.4a}$$

$$\text{Subject to} : y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle_{\mathcal{H}} + b) \geq 1 - \xi_i \tag{2.4b}$$

$$\xi_i \geq 0$$

With the help of Lagrange multipliers, the Wolfe dual form of the above minimization problem is [19]

$$\text{Maximize} : W(\alpha) = \alpha^{\mathbf{T}}\mathbf{1} - \frac{1}{2}\alpha^{\mathbf{T}}\mathbf{H}\alpha \tag{2.5a}$$

$$\text{Subject to} : 0 \leq \alpha \leq C \tag{2.5b}$$

$$\alpha^{\mathbf{T}}\mathbf{y} = 0.$$

where $\alpha_i \geq 0$ $(i = 1, 2, ..., n)$ are the Lagrange multipliers, $\mathbf{1}$ is a vector of ones and $\mathcal{H}$ is the Gram matrix with component $H_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. The necessary and sufficient condition for a weight vector $\mathbf{w}$ and Lagrange multiplier α to be optimal is the KKT condition [19], which are the primal and dual feasibility constraints plus the following complementarity's conditions

$$\alpha_i \left(y_i \left(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle_{\mathcal{H}} + b\right) - 1 + \xi_i\right) = 0 \tag{2.6a}$$

$$\xi_i \left(\alpha_i - C\right) = 0 \tag{2.6b}$$

Based on the optimal solution α, the functional margin $d(\cdot)$ can also be written as

$$d(\mathbf{x}) = \sum_{x_i \in \mathcal{D}_{SV}} K(\mathbf{x}_i, x) + b \tag{2.7}$$

where D_{SV} is the set of support vectors, which are the subset of training data that have nonzero α's, that is, $0 < \alpha \leq C$. It is the set of support vectors that determines the decision boundary and all the other training data, that is, non-support vectors, can be removed without influencing the decision boundary.

2.2 ClusterSVM Figure 1 shows the training data of a two-dimensional two-class classification problem. The training data in the positive class are partitioned into two disjoint clusters and those in the negative class are partitioned into three clusters. The motivation of *ClusterSVM* is to reduce the number of training data by replacing a cluster with an appropriately defined representative. However, not all clusters can be replaced with a representative while yielding the same the SVM as the SVM that would be obtained using the original training data set $\mathcal{D}$. It follows from the following Proposition 2.1 that there are two kinds of clusters that can be replaced without influencing the solution, including the cluster that contains only non-support vectors ($\alpha = 0$) and the cluster that contains only boundary support vectors ($\alpha = C$).

PROPOSITION 2.1. *Let $\mathcal{D}$ denote the training data set consisting of two disjoint sets $\mathcal{D}_1$ and $\mathcal{D}_2$ and, without loosing generality, assume $\mathcal{D}_1$ is a subset of the training data in class 1. Further, define the representative of cluster $\mathcal{D}_1$ be a point $\mathbf{x}_0$ such that $\Phi(\mathbf{x}_0)$ is the mean of the images of all data in $\mathcal{D}_1$ under $\Phi(\cdot)$, that is,*

$$\Phi(\mathbf{x}_0) = \frac{1}{n_1}\sum_{i=1}^{n_1} \Phi(\mathbf{x}_i) \tag{2.8}$$

where n_1 is the number of data in $\mathcal{D}_1$. Then the optimization problem obtained by replacing the set $\mathcal{D}_1$ with its representative $\mathbf{x}_0$ and setting the upper bound of α_0, which is the Lagrange multiplier of $\mathbf{x}_0$ (c.f.(2.5)),

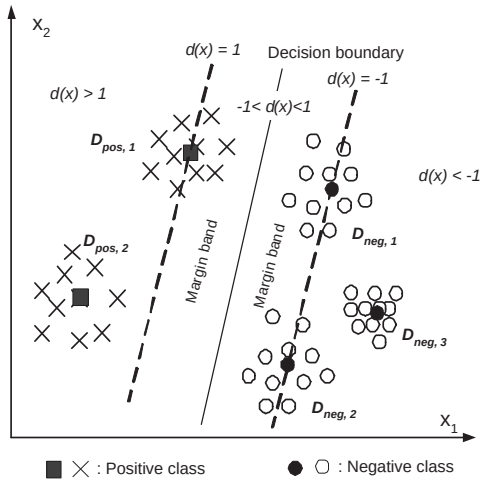

Figure 1: A toy example. The representative of a cluster is labeled with a solid square/circle. The decision boundary of the SVM trained using the representatives of 5 initial clusters is shown.

to n_1C is equivalent to the one obtained by adding a constraint to (2.5) *that requires all Lagrange multipliers corresponding to the data in $\mathcal{D}_1$ be equal.*

Proof. Let α_1 and α_2 be the Lagrange multipliers of the data in $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. The optimization problem (2.5) can be written as

$$\begin{aligned} \text{Maximize} : W(\alpha) &= \left(\alpha_1 \mathbf{1}_1 - \frac{1}{2}\alpha_1^T \mathbf{H}_{11}\alpha_1\right) \\ &+ \left(\alpha_2 \mathbf{1}_2 - \frac{1}{2}\alpha_2^T \mathbf{H}_{22}\alpha_2\right) - \alpha_1^T \mathbf{H}_{12}\alpha_2 \quad (2.9a) \\ \text{Subject to} : & 0 \le \alpha_{1,i} \le C,\ \forall i = 1, 2, ..., n_1 \\ & 0 \le \alpha_{2,j} \le C,\ \forall j = 1, 2, ..., n_2 \\ & \alpha_1^T \mathbf{y}_1 + \alpha_2^T \mathbf{y}_2 = 0. \quad (2.9b) \end{aligned}$$

Upon replacing data in $\mathcal{D}_1$ with $\mathbf{x}_0$, the objective function in (2.9) can be written as,

$$\begin{aligned} W(\alpha) &= \left(\alpha_0 - \frac{1}{2}\alpha_0 H_{00}\alpha_0\right) \\ &+ \left(\alpha_2 \mathbf{1}_2 - \frac{1}{2}\alpha_2^T \mathbf{H}_{22}\alpha_2\right) - \alpha_0 \mathbf{H}_{02}\alpha_2 \quad (2.10) \end{aligned}$$

Using equation (2.8), we have

$$\begin{aligned} \alpha_0 H_{00}\alpha_0 &= \alpha_0\alpha_0 y_0 y_0 \langle \Phi(\mathbf{x}_0), \Phi(\mathbf{x}_0)\rangle \\ &= \alpha_0\alpha_0 y_0 y_0 \frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_1}\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle \quad (2.11) \end{aligned}$$

Let α_1^* be a vector of length n_1 with all components being equal to α_0/n_1, equation (2.11) can be written as

$$\begin{aligned} \alpha_0 H_{00}\alpha_0 &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_1}\alpha_{1,i}^*\alpha_{1,j}^* y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \alpha_1^{*T}\mathbf{H}_{11}\alpha_1^* \quad (2.12) \end{aligned}$$

where $y_i = y_0$ $(i = 1, 2, ..., n_1)$ and $\sum_{i=1}^{n_1}\alpha_{1,i}^* = \alpha_0$. Since $0 \le \alpha_0 \le n_1 C$, we have $0 \le \alpha_{1,i}^* \le C$ $(i = 1, 2, ..., n_1)$. Similarly, $\alpha_0 \mathbf{H}_{02}\alpha_2$ can be written as

$$\alpha_0 H_{02}\alpha_2 = \alpha_1^{*T}\mathbf{H}_{12}\alpha_2 \quad (2.13)$$

After substituting equations (2.12) and (2.13) into equation (2.10), we arrive the following optimization problem

$$\begin{aligned} \text{Maximize} : W(\alpha_1^*, \alpha_2) &= \left(\alpha_1^* \mathbf{1}_1^* - \frac{1}{2}\alpha_1^{*T}\mathbf{H}_{11}\alpha_1^*\right) \\ &+ \left(\alpha_2 \mathbf{1}_2 - \frac{1}{2}\alpha_2^T \mathbf{H}_{22}\alpha_2\right) - \alpha_1^{*T}\mathbf{H}_{12}\alpha_2 \quad (2.14a) \\ \text{Subject to} : & 0 \le \alpha_{1,i}^* \le C,\ \forall i = 1, 2, ..., n_1 \\ & 0 \le \alpha_{2,j} \le C,\ \forall j = 1, 2, ..., n_2 \\ & \boldsymbol{\alpha_1^{*T}\mathbf{y}_1 + \alpha_2^T\mathbf{y}_2 = 0.} \\ & \alpha_{1,i_1}^* = \alpha_{1,i_2}^*,\ \forall i_1, i_2 = 1, 2, ..., n_1 \quad (2.14b) \end{aligned}$$

The proposition follows by comparing equation (2.9) and equation (2.14). □

It should be pointed out that, in general, a point $\mathbf{x}_0$ satisfying equation (2.8) may not exist (e.g., when K is a Gaussian kernel). However, assuming the existence of such $\mathbf{x}_0$ is only for the convenience of stating and proving this proposition. In practice, replacing a cluster with $\mathbf{x}_0$ can be implemented by replacing corresponding rows/columns with there average in the Gram matrix. The idea of Proposition 2.1 is illustrated in Figure 2 for a toy problem that has two points in class 1 with Lagrange multipliers α_1 and α_2, and one point in class -1 with Lagrange multiplier α_3. The cube $pqst - ovwu$ is the feasible region of the original optimization problem (2.9). After replacing two data in class 1 by a point whose image is the mean of the images of these two data, the feasible region of the resulting optimization problem (c.f. (2.14)) is the rectangle $opsw$. Thus, the feasible region of the problem (2.14) is a subset of that of the problem (2.9). It is not hard to show that the optimal solution of (2.14) is also the optimal solution of (2.9) if $\mathcal{D}_1$ satisfies either of the following conditions. As in Proposition 2.1, $\mathcal{D}_1$ is a subset of the data in class 1 and it will be replaced by its representative defined by equation (2.8).

- **Condition 1** All data in $\mathcal{D}_1$ are non-support vectors, which means the corresponding Lagrange

multiplier $\alpha_1 = \alpha_2 = 0$. With reference to Figure 2, this means the feasible region of the problem (2.9) and that of the problem (2.14) coincides at line *op*.

- **Condition 2** All data in $\mathcal{D}_1$ are boundary support vectors, which means the corresponding Lagrange multipliers $\alpha_1 = \alpha_2 = C$. With reference to Figure 2, this means that the feasible region of the problem (2.9) and that of the problem (2.14) coincides at line *ws*.

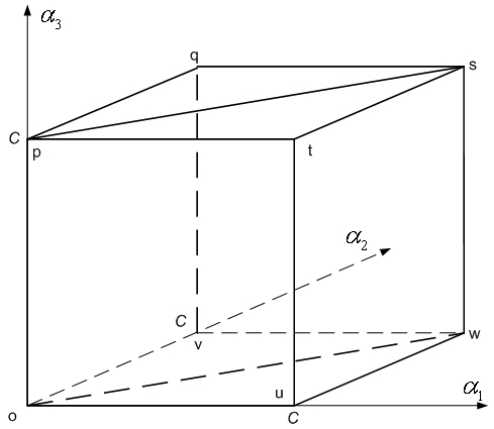

Figure 2: Illustration of Proposition 2.1

Thus, we can safely replace clusters of above two types with a representative defined in equation 2.8 without influencing the solution, and this will make the complexity of the resulting training algorithm scales with the square of the number of non-boundary support vectors. In the current algorithm, only the first condition is used, which gives out one more simplification. Since the Lagrange multipliers of non-support vectors are all zeros, we can choose a cluster's representative to be one of its members, instead of the one defined in equation (2.8).

The next issue is to identify clusters that contain only non-support vectors. However, there is a cycle here because the set of support vectors is unknown before training is finished. The solution is to first partition the training data into pair wise disjoint clusters, then train an initial SVM using the representatives of these clusters. Based on this initial SVM, we can approximately tell the position of each cluster relative to the decision boundary, thereby approximately identifying the clusters containing no support vectors. For the cluster that is believed to contain both support vectors and non-support vectors, it is split into two subclusters, one of which is expected to contain only support vectors and the other is expected to contain only non-support vectors. For example, in Figure 1, all data in the cluster $\mathcal{D}_{pos,2}$ have functional margin larger than one and it is estimated not to contain any support vectors. However, the cluster $\mathcal{D}_{pos,1}$ contains data with functional margin $d(\mathbf{x}) \leq 1$ and these data are likely to be support vectors. Thus, the cluster $\mathcal{D}_{pos,1}$ needs to be split into two subclusters. It is believed that the subcluster having data with $d(\mathbf{x}) > 1$ contains no support vectors, while the other subcluster is believed to contain only support vectors. After splitting cluster $\mathcal{D}_{pos,1}$, $\mathcal{D}_{neg,1}$ and $\mathcal{D}_{neg,2}$ and replacing some clusters with their representatives, the resulting training data set $\mathcal{D}_{reduced}$ is shown in Figure 3, from which we can see a significant reduction on the number of training data. Algorithm 1 summarized the proposed *ClusterSVM*, where the pseudocenter $\mathbf{x}^p(\mathcal{D}_i)$ of a cluster $\mathcal{D}_i$ is defined as

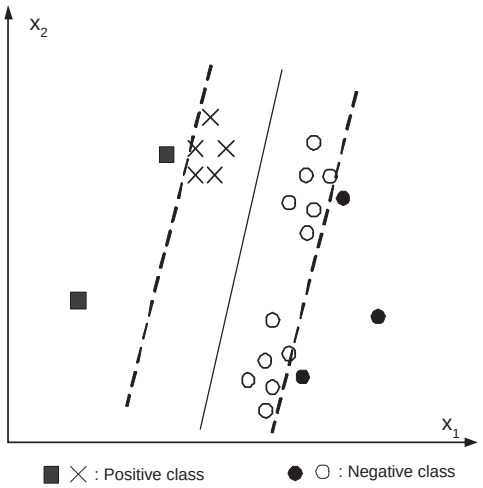

Figure 3: The reduced training data set $\mathcal{D}_{reduced}$ after replacing some clusters with their representatives. For clarity purpose, the decision boundary shown in Figure 1 is kept here.

$$\mathbf{x}^p(\mathcal{D}_i) = \underset{\mathbf{x}\in\mathcal{D}_i}{\operatorname{argmin}} \left\| \mathbf{x} - \frac{1}{n_i}\sum_{k=1}^{n_i} \mathbf{x}_k \right\|_2 \tag{2.15}$$

where $\|\cdot\|_2$ means 2-norm and n_i is the number of data in $\mathcal{D}_i$. The properties of Algorithm 1 (*ClusterSVM*) is summarized in Proposition 2.2.

PROPOSITION 2.2.

1. *Algorithm 1 will converge in a finite number of steps.*
2. *The SVM_{new} obtained using only $\mathcal{D}_{reduced}$ is the same as the SVM that would be obtained using $\mathcal{D}$ when Algorithm 1 terminates, that is, when the following condition is satisfied*

$$yd(\mathbf{x}) > 1, \quad \forall \mathbf{x} \in \mathcal{D}_{unused} \tag{2.16}$$

where $\mathcal{D}_{unused}$ contains data in $\mathcal{D}$ but not in $\mathcal{D}_{reduced}$.

Proof. From Algorithm 1, we can see that the size of the reduced training data set $\mathcal{D}_{reduced}$ is strictly increasing, that is,

$$|\mathcal{D}^{old}_{reduced}| < |\mathcal{D}_{reduced}| \tag{2.17}$$

Since there is a finite number of training data in $\mathcal{D}$, $\mathcal{D}_{reduced}$ will be the same as $\mathcal{D}$ in a finite number of steps, which means that Algorithm 1 will converge in a finite number of steps.

For the second part of the proposition, we need only to show that, for the weight vector $\mathbf{w}$ of SVM_{new}, the KKT conditions are satisfied for all training data in $\mathcal{D}_{unused}$, that is, the Lagrange multiplier is zero. For a given $\mathbf{x}_i \in \mathcal{D}_{unused}$, we have

$$y_i d(\mathbf{x}_i) > 1 \Longrightarrow y_i \left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle_{\mathcal{H}} + b\right) > 1 \tag{2.18}$$

Knowing the fact that $\xi_i \geq 0$, this means that the constraint specified by equation (2.4b) will not be active, thus $\alpha_i = 0$. □

3 Experiments

3.1 Implementations Due to its popularity, the training algorithm $\mathbb{A}$ we use in Algorithm 1 is Platt's SMO [16], and the *ClusterSVM* is compared with SMO. An implementation of SMO by Chang et al. [5] and its Matlab® wrapper by Ma et al. [14] were used in this paper. However, it should be pointed out that, being used as a meta-algorithm, *ClusterSVM* could accelerate *any* training algorithm. The clustering algorithm $\mathbb{C}$ used here is the PDDP (Principal Direction Divisive Partition) by Boley [3] because it is one the most efficient clustering algorithms.

The number of initial clusters k^+ (k^-) can be any number between one and the number of training data n^+ (n^-) in $\mathcal{D}^+$ ($\mathcal{D}^-$). However, knowing the fact that the initial SVM will be trained using the representatives of the initial clusters and all subsequent partitions will depend on this SVM, the number of initial clusters should be large enough so that the initial SVM can approximate the true SVM reasonably well. At the same time, it should not be too large since letting $k^+ = n^+$ and $k^- = n^-$ would make $\mathcal{D}_{reduced} = \mathcal{D}$, and there will be no speedup. Another reason for preferring small k^+ (k^-) is that both clustering the training data $\mathcal{D}$ and training the initial SVM can be performed very quickly. In this paper, the following square root heuristic is suggested

$$k^+ = round(\sqrt{n^+}) \text{ and } k^- = round(\sqrt{n^-}) \tag{3.19}$$

Algorithm 1 ClusterSVM: Two class SVM

Require: A SVM training algorithm $\mathbb{A}$; A clustering algorithm $\mathbb{C}$; Training data set $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$, where $\mathcal{D}^+$ ($\mathcal{D}^-$) is the set of the training data in class 1 (-1); The number of initial clusters k^+ (k^-) into which $\mathcal{D}^+$ ($\mathcal{D}^-$) is partitioned; The maximum number of passes NP_{max} through the $WHILE$ loop.

1: Call the clustering algorithm $\mathbb{C}$ to partition $\mathcal{D}^+$ ($\mathcal{D}^-$) into k^+ (k^-) clusters, that is

$$\mathcal{D}^+ = \bigcup_{i=1}^{k^+} \mathcal{D}^+_i \quad \text{and} \quad \mathcal{D}^- = \bigcup_{i=1}^{k^-} \mathcal{D}^-_i$$

2: Define the set $\mathcal{G}$ of clusters as

$$\mathcal{G} \leftarrow \{\mathcal{D}^+_1, \cdots, \mathcal{D}^+_{k^+}, \mathcal{D}^-_1, \cdots, \mathcal{D}^-_{k^-}\}$$

3: Define the reduced training data set $\mathcal{D}_{reduced}$ as (c.f. (2.15))

$$\mathcal{D}_{reduced} \leftarrow \{\mathbf{x}^p(\mathcal{D}'), \mathcal{D}' \in \mathcal{G}\}$$

4: $Flag \leftarrow 1$, $NP \leftarrow 0$
5: **while** $Flag = 1$ and $NP < NP_{max}$ **do**
6: $\quad Flag \leftarrow 0$, $NP \leftarrow NP + 1$
7: $\quad$ Train SVM_{new} using $\mathcal{D}_{reduced}$ and the training algorithm $\mathbb{A}$
8: $\quad$ Remove from $\mathcal{D}_{reduced}$ the datum that is the representative of any cluster in $\mathcal{G}$
9: $\quad \mathcal{G}^{old} \leftarrow \mathcal{G}$ and $\mathcal{G} \leftarrow \emptyset$
10: $\quad$ **for all** $\mathcal{D}' \in \mathcal{G}^{old}$ **do**
11: $\quad\quad$ **if** $\exists \mathbf{x} \in \mathcal{D}'$ such that $yd(\mathbf{x}) \leq 1$ according to SVM_{new}, where y is the label of $\mathbf{x}$ **then**
12: $\quad\quad\quad Flag \leftarrow 1$
13: $\quad\quad\quad$ Split $\mathcal{D}'$ into $\mathcal{D}'_{sv}$ and $\mathcal{D}'_{nsv}$

$$\mathcal{D}'_{sv} = \{\mathbf{x} | \mathbf{x} \in \mathcal{D}' \quad \text{and} \quad yd(\mathbf{x}) \leq 1\}$$
$$\mathcal{D}'_{nsv} = \{\mathbf{x} | \mathbf{x} \in \mathcal{D}' \quad \text{and} \quad yd(\mathbf{x}) > 1\}$$

14: $\quad\quad\quad \mathcal{D}_{reduced} \leftarrow \mathcal{D}_{reduced} \cup \mathcal{D}'_{sv} \cup \{\mathbf{x}^p(\mathcal{D}'_{nsv})\}$
15: $\quad\quad\quad \mathcal{G} = \mathcal{G} \cup \{\mathcal{D}'_{nsv}\}$
16: $\quad\quad$ **else**
17: $\quad\quad\quad \mathcal{D}_{reduced} \leftarrow \mathcal{D}_{reduced} \cup \{\mathbf{x}^p(\mathcal{D}')\}$
18: $\quad\quad\quad \mathcal{G} = \mathcal{G} \cup \{\mathcal{D}'\}$
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end while**
22: Return the SVM_{new}.

There are primarily two motivations for this heuristic. First, knowing the fact that the time for clustering

typically scales linearly with the number of training data [8], the square root heuristic can make the total time to obtain the initial SVM scale linearly with the number of training data. The second motivation is that this heuristic has been suggested in the study of clustering algorithms (e.g. [7]). The effectiveness of this heuristic was demonstrated experimentally. Since the initial SVM can approximate the true SVM quite well and each pass through the outer *WHILE* loop (line 5 to 21 in Algorithm 1) involves training a SVM, the next issue is how many times the *WHILE* loop should be performed. Based on the experiments, it is enough to carry out the *WHILE* loop once.

The last implementation issue is the strategy for the multi-class classification problem. There are many strategies for multi-class classification problem and, in this paper, the "one versus the rest" strategy is used. In this strategy, assuming there are m classes, m classifiers are trained and each of them discriminates one class from all the other classes. A test data is classified to the class that has the maximum functional margin. In order to avoid the repeated clustering, the clustering algorithm is applied to the data of each class before any classifier is trained. Then, to train the classifier that discriminates the class i from the remaining $m-1$ classes, the clusters corresponding to class i are used as the partition of the data in class i, and the clusters corresponding to the remaining $m-1$ classes are put together and used as the partition for the data in that $m-1$ classes. All experiments were run on a PC running Windows 2000 Server with one Pentium 4 2.8GHz processor and 1GB RAM, and the algorithm was implemented using Matlab® [18].

3.2 Data sets There are three data sets examined in this paper.

- **Artificial data set** This is a three-class classification problem and each class consists of data drawn from a 2D normal distribution with covariance matrix being identity matrix. The centers of three classes are $(0, \sqrt{3})$, $(-1, 0)$ and $(1, 0)$. The same number of training data are drawn for each class and the size of the training data $\mathcal{D}$ varies from 300 to 6000. The test data set is of the same size as the training data set and is constructed in the same way. All 3 classifiers are obtained using the regularization coefficient $C = 10000$ (c.f. equation (2.4a)) and the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.

- **USPS data set** This is the US Postal Service (USPS) handwritten zip code recognition data set and there are 7291 training data and 2007 test data, all of which were collected from mail envelopes in Buffalo [13]. Each digit is represented as a 16×16 matrix whose entry ranges from -1 to 1. As suggested by [17], a smoothing operation using a Gaussian kernel with width 0.75 was applied to the image as a preprocessing step. The regularization coefficient $C = 10$ and the kernel is a homogeneous polynomial kernel of degree 3, that is, $K(\mathbf{x}_i, \mathbf{x}_j) = \left(\frac{\mathbf{x}_i^T \mathbf{x}_j}{256}\right)^3$.

- **Isolet data set** This data set was downloaded from UCI machine learning repository [2] and the goal is to recognize 26 spoken letters. There are 6238 training data and 1559 test data. Each datum has 617 attributes and each attribute is a real number between -1 and 1. All 26 classifiers were obtained using the regularization coefficient $C = 0.02$ and the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.

3.3 Experimental results The effect of the number of initial clusters k was studied through the artificial data set. There 2000 training data in each class (6000 total) and the number of initial clusters k varies from 1 to 81 with an interval of 2. For each value of k, ten randomly generated training data set were tried. Figure 4 compares the relative difference between the error rate of the initial SVM with that of the true SVM for different values of k. The relative difference RD is defined as

$$RD = \frac{|ER_{initial} - ER_{true}|}{ER_{true}} \tag{3.20}$$

where $ER_{initial}$ and ER_{true} are the error rate on the same test data set. Figure 5 shows the time to obtain the initial SVM $T_{Initial\ SVM}$ as a function of the number of initial clusters. $T_{Initial\ SVM}$ consists of time for clustering and the time for training the initial SVM. From Figure 4 and Figure 5, we can see that the square root heuristic, corresponding to $k = 45$ in this experiment, gives a reasonable good trade-off between the accuracy and the complexity, although it is a rather gross heuristic.

With the number of initial clusters being specified by the square root heuristic, the effect of the number of passes NP through the *WHILE* loop (line 5 to 21 in Algorithm 1) is shown in Table 1. The SVM trained after 3 passes is the same as the SVM that would be obtained using the original training data set, thus the corresponding error rate can be taken as the reference. From Table 1, it can be seen that one pass through the *WHILE* loop is enough to give a good performance. Thus, the maximum number of passes is set to 1 in Algorithm 1, that is, $NP_{max} = 1$. In addition, it can be seen from Table 1 that, with $NP = 0$, the initial SVM

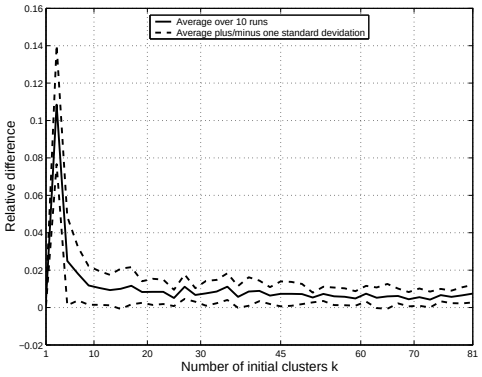

Figure 4: The performance of the initial SVM compared to that of true SVM. The square root heuristic corresponds $k = 45$. The seemingly good performance of $k = 1$ comes from the symmetry of this problem and it has no general implications.

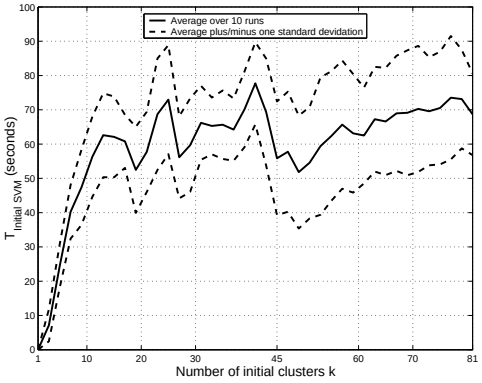

Figure 5: Time to obtain the initial SVM as a function of the number of initial clusters k. The artificial data set is used and the square root heuristic corresponds $k = 45$.

also gives pretty good result.

Table 1: Effects of NP on the artificial data set (6000 training data). NP is the number of passes through the *WHILE* loop in Algorithm 1.

NP	0	1	2	3
Error rate (%)	25.87	25.43	25.48	25.47

Table 2 through 4 compares the performance of *ClusterSVM* with that of SMO, where the number of initial clusters is specified by the square root heuristic and the maximum number of passes through the *WHILE* loop $NP_{max} = 1$. In these tables, the speed up is defined as

$$Speedup = \frac{T_{SMO}}{T_{ClusterSVM}} \tag{3.21}$$

where T_{SMO} is the training time of SMO and $T_{ClusterSVM}$ is the training time of *ClusterSVM*. The clustering time is the time used for the initial clustering. Based on these tables, we have the following observations.

- $N_{train,i}$ $(i = 1, 2, \cdots, m)$ is the actual number of training data used to train the i-th classifier. For SMO, this number is the number of training data of all classes and it is independent of which classifier is being trained. For *ClusterSVM*, $N_{train,i}$ is the number of training data after replacing every cluster containing no support vectors with its representative. For the artificial data set shown in Table 2, $N_{train,i}$ is almost the same for all three classifiers and this is within our expectations, since this is a symmetric problem. However, for the USPS data set shown in Table 3, $N_{train,i}$ varies from one classifier to another and this reflects the fact that all ten classifiers are inherently different. For example, discriminating digit 1 from the other digits is different from discriminating digit 0 from the other digits. Similarly, for the Isolet data set shown in Table 3, different classifier has different number of training data. Thus, the *ClusterSVM* reduces the number of training data in a task dependent way.

- N_{train} is the average number of training data over all k classifiers and, comparing *ClusterSVM* with SMO, it can be seen that *ClusterSVM* reduce the number of training data significantly. It is this data reduction that help accelerating the training process.

- The speed up of *ClusterSVM* over SMO is 3.2 for the artificial data set, 1.5 for the USPS data set and 1.9 for the Isolet data set.

- Comparing the error rate of SMO and that of *ClusterSVM*, it can be seen that the speedup of *ClusterSVM* sacrifices little performance. This nice property is attributed to the good initial clusters, whose overhead is only a small faction of total training time.

Finally, the scaling performance of *ClusterSVM* was shown in Figure 6 for the artificial data set, which shows the average training time over 10 runs. It can be seen that *ClusterSVM* scales better than SMO.

Table 2: Artificial data set. $N_{train,i}$ is the actual number of training data to train the i-th classifier.

	SMO	*ClusterSVM*
$N_{train,1}$	6000	3260
$N_{train,2}$	6000	3026
$N_{train,3}$	6000	3022
N_{train}	6000	3103
Training time (sec.)	8344	2588
Clustering time (sec.)	NA	3
Speedup	3.2	
Error rate (%)	25.47	25.43

Table 3: USPS data set. $N_{train,i}$ is the actual number of training data to train the i-th classifier.

	SMO	*ClusterSVM*
$N_{train,1}$	7291	788
$N_{train,2}$	7291	2364
$N_{train,3}$	7291	1975
$N_{train,4}$	7291	1544
$N_{train,5}$	7291	2259
$N_{train,6}$	7291	1621
$N_{train,7}$	7291	1206
$N_{train,8}$	7291	2407
$N_{train,9}$	7291	1560
$N_{train,10}$	7291	2638
N_{train}	7291	1836
Training time (sec.)	105	68
Clustering time (sec.)	NA	18
Speedup	1.5	
Error rate (%)	5.43	5.28

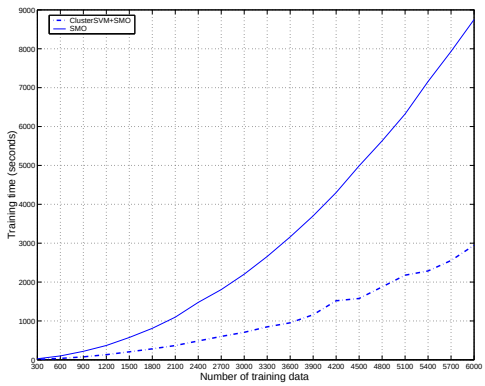

Figure 6: Comparison of the scaling performance of *ClusterSVM* and that of SMO on the artificial data set. The solid line represents SMO and the dashed line represents *ClusterSVM*

4 Conclusions

An efficient SVM training algorithm *ClusterSVM* was proposed in this paper and a significant speedup over

Table 4: Isolet data set. $N_{train,i}$ is the actual number of training data to train the i-th classifier.

	SMO	*ClusterSVM*
$N_{train,1}$	6238	1102
$N_{train,2}$	6238	1237
$N_{train,3}$	6238	840
$N_{train,4}$	6238	1242
$N_{train,5}$	6238	1123
$N_{train,6}$	6238	1102
$N_{train,7}$	6238	1016
$N_{train,8}$	6238	932
$N_{train,9}$	6238	957
$N_{train,10}$	6238	1039
$N_{train,11}$	6238	1048
$N_{train,12}$	6238	914
$N_{train,13}$	6238	945
$N_{train,14}$	6238	1159
$N_{train,15}$	6238	946
$N_{train,16}$	6238	1435
$N_{train,17}$	6238	1018
$N_{train,18}$	6238	883
$N_{train,19}$	6238	762
$N_{train,20}$	6238	1225
$N_{train,21}$	6238	980
$N_{train,22}$	6238	1346
$N_{train,23}$	6238	1258
$N_{train,24}$	6238	837
$N_{train,25}$	6238	852
$N_{train,26}$	6238	864
N_{train}	6238	1041
Training time (sec.)	278	144
Clustering time (sec.)	NA	24
Speedup	1.9	
Error rate (%)	4.55	4.55

SMO was observed on both the artificial data set and the real data set. The possible extensions to *ClusterSVM* are the follows.

- The second sufficient condition mentioned after the Proposition 2.1 has not been used in *ClusterSVM*. It is not hard to incorporate this condition into *ClusterSVM* and this would make the training time scale with the number of non-boundary support vectors. This would definitely speed up the SVM training further.

- With the help of a clustering algorithm, *ClusterSVM* effectively incorporate the distributional property of the training data into the training process. It is expected that the similar idea can be used

to improve other supervised learning algorithm like neural networks.

References

[1] D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. In S. B. Thomas, G. Dietterich, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, 2002.

[2] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[3] D. L. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers, 1992. Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, ACM.

[5] C. C. Chang and C. J. Lin. Libsvm: a library for support vector machines, 2001. Version 2.33.

[6] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[7] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *15th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 318–329, 1992.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2000.

[9] S. Fine and K. Scheinberg. Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

[10] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[11] L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[12] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1), January 2000.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[14] J. Ma, Y. Zhao, and S. Ahalt. OSU SVM classifier matlab toolbox 3.00.

[15] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In A. Island, editor, *IEEE NNSP*, 1997.

[16] J. C. Platt. 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*. MIT Press, 1999.

[17] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *First International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1995.

[18] The Mathworks Inc. Matlab 6.1. http://www.mathworks.com.

[19] V. Vapnik. *Statistical Learning Theory*. Wiley, NY, 1998.

[20] C. K. I. Williams and M. Seeger. Using the nystrom method to speed up kernel machines. In T. K. Leen, T. G. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.