Training Support Vector Machine using 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 only non-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 scale with square of the number of non-boundary support vectors.

Keywords: support vector machine, PDDP, clustering, optimization.

1 Introduction

Support vector machines (SVM) (Vapnik [32]) have been successfully applied in a variety of domains, including handwritten digit recognition [4], text document classification [15] and microarray data analysis [5]. In training a support vector machine, 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 [7]. 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. [25], Kaufman et al. [17] and Joachims [16]. Especially noteworthy is the SMO (Sequential Minimal Optimization) algorithm by Platt [27] 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. [10]) or by sampling (Williams et al. [33], Achlioptas et al. [1]), thereby reducing the size of the optimization problem and speeding up the training process. Using the fact that it is the support vectors that determine the optimal solution, a third class of algorithms tries to identify the support vectors efficiently, and this is the approach taken in this paper. Shin et al. [30] proposed a fast training algorithm based on quick identification of support vectors, however, their algorithm appears to have some difficulties in dealing with linearly separable training data set. In addition, Keerthi et al. [18] proposed an algorithm based on observations about the geometrical properties of support vector machines.

Mangasarian and his colleagues proposed several variations of standard support vector machines by modifying the objective function, together with several very efficient training algorithms ([22], [24], [20], [13], [23], [11], [14] and [12]). Since the optimization problems given by these modifications are different from that of standard support vector machines, we will not address them in this paper.

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

*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

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 such that, approximately, one contains only non-support vectors and the other contains 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 solution can be found even faster. 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.

Enhancing the SVM training process with clustering or similar techniques has been examined with several variations in [34], [26] and [29]. Based on a hierarchical micro-clustering algorithm, Yu et al. [34] proposed a scalable algorithm to train support vector machines with a linear kernel. However, their algorithm currently works for the linear kernel only and uses the fact that the original space and the feature space are the same under a linear kernel. It is not immediately obvious how to generalize their algorithm to more popular nonlinear kernels. In [29], Shih et al proposed a technique called text bundling, where the training data are partitioned into clusters based on a Rocchio score, and data within a cluster are replaced by their mean. Pavlov et al. [26] proposed a strategy to speed up the training process by squashing. Both [29] and [26] used only linear kernels in their experiments. In this paper, we will demonstrate both theoretically and experimentally that the proposed method is applicable to both the linear kernel and nonlinear kernels.

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, we are 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 y_i indicates the class membership of the object i represented by vector $\mathbf{x}_i$, and $i = 1, 2, \cdots, n$. The support vector classifier $f(\mathbf{x})$ is defined as [32]

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

where $d(\cdot)$ is called the *functional margin*,

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

Here, $\Phi(\cdot)$ is a mapping from $\mathcal{R}^N$ to a (usually) higher dimension Hilbert space $\mathcal{H}$, and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the dot product in $\mathcal{H}$. Generally, the mapping $\Phi(\cdot)$ is specified implicitly by using a kernel $K(\cdot, \cdot)$ that calculates the dot product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, that is,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle_{\mathcal{H}}. \tag{2.4}$$

To guarantee the existence of $\Phi(\cdot)$, the kernel $K(\cdot, \cdot)$ must satisfy certain conditions [32]. The optimal parameters $\mathbf{w}^*$ and b^* corresponding to the optimal classifier $f^*(\mathbf{x})$ can be obtained by solving the following optimization problem [32]

$$\text{Minimize} : g(\mathbf{w}, \xi) = \frac{1}{2} ||\mathbf{w}||^2_{\mathcal{H}} + C \sum_{i=1}^{n} \xi_i \tag{2.5a}$$

$$\text{Subject to} : y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle_{\mathcal{H}} + b) \geq 1 - \xi_i \quad \xi_i \geq 0. \tag{2.5b}$$

With the help of Lagrange multipliers, the Wolfe dual form of the above minimization problem is [32]

$$\text{Maximize} : W(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \tag{2.6a}$$

$$\text{Subject to} : 0 \leq \alpha_i \leq C,\ \forall i = 1, 2, ..., n \quad \boldsymbol{\alpha}^T \mathbf{y} = 0, \tag{2.6b}$$

where $\boldsymbol{\alpha}$ is a vector with components α_i that are the Lagrange multipliers, $\mathbf{1}$ is a vector of ones, and $\mathbf{H}$ is the Gram matrix with component $H_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. The necessary and sufficient conditions for a weight vector $\mathbf{w}$ and Lagrange multipliers $\boldsymbol{\alpha}$ to be optimal are the KKT conditions [32], which consist of primal and dual feasibility constraints plus the following complementarity conditions

$$\alpha_i \left(y_i \left(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle_{\mathcal{H}} + b \right) - 1 + \xi_i \right) = 0 \tag{2.7a}$$

$$\xi_i \left(\alpha_i - C \right) = 0. \tag{2.7b}$$

Based on the optimal solution $\boldsymbol{\alpha}$, the functional margin $d(\cdot)$ can also be written as

$$d(\mathbf{x}) = \sum_{x_i \in \mathcal{D}_{SV}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b, \tag{2.8}$$

where D_{SV} is the set of support vectors, which are the subset of training data that have nonzero α's, that is, $0 < \alpha_i \leq C$. 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 Review of Available Training Algorithms In this subsection, we will informally review the methods based on chunking and decomposition, and their relationships to the method proposed in this paper.

Chunking [4] is based on the observation that it is set of support vectors that determine the classifier (c.f. equation (2.8)). It works as follows: First, train a support vector machine using a subset of the training data and discard the non-support vectors; Second, train next support vector machine using the support vectors and some of the unused training data. These two steps are repeated until all training data are used and they all satisfy the KKT conditions. Training each support vector machine is accomplished using an off-the-shelf optimization package. This method will suffer if the number of support vectors is very large.

Another class of training algorithms is based on the idea of decomposition ([25], [17] and [16]). In these algorithms, a fixed number of training data are selected as active set for each iteration, and we optimize $W(\cdot)$ with respect to the α's corresponding to active set, while keeping the α's of all the other training data frozen. The algorithm terminates when the KKT conditions are satisfied for all training data. This kind of algorithm can be used in many different formulations of support vector machines, and their success depends on the effectiveness of the criterion used in active set selection.

Sequential Minimum Optimization (SMO) [27] is an extreme case of decomposition-based methods. In SMO, the size of the active set is 2. Assuming α_1 and α_2 correspond to the training data in the active set and the α's corresponding to the other training data are frozen, the optimization problem in equation (2.6) can be written as

$$\begin{aligned} \text{Maximize}: W(\boldsymbol{\alpha}_\mathbf{a}) &= \left(\boldsymbol{\alpha}_\mathbf{a}^T \mathbf{1}_\mathbf{a} - \frac{1}{2}\boldsymbol{\alpha}_\mathbf{a}^T \mathbf{H}_{\mathbf{aa}} \boldsymbol{\alpha}_\mathbf{a}\right) \\ &\quad + F_1 - \boldsymbol{\alpha}_\mathbf{a}^T \mathbf{H}_{\mathbf{ac}} \boldsymbol{\alpha}_\mathbf{c} \qquad (2.9a) \\ \text{Subject to}: & \; 0 \leq \alpha_1 \leq C \\ & \; 0 \leq \alpha_2 \leq C \\ & \; \alpha_1 y_1 + \alpha_2 y_2 = F_2, \qquad (2.9b) \end{aligned}$$

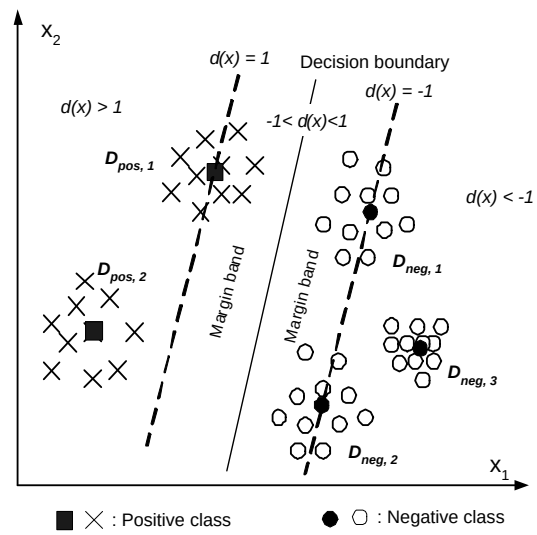

Figure 1: A toy example. The representative of a cluster is labeled with a solid square/circle. The decision boundary of the initial SVM trained using the representatives of 5 initial clusters is shown.

where subscript $\mathbf{a} = [1\ 2]^T$ denotes the data within the active set, subscript $\mathbf{c} = [3\ 4\ \cdots\ N]^T$ denotes the data outside the active set, and F_1 and F_2 depend only on the data outside the active set and are constant while solving (2.9). After expressing α_1 as a function of α_2 using the equality constraint in (2.9b) and substituting the resulting formula into (2.9a), we can obtain an explicit formula for the α_2 that maximizes $W(\cdot)$. After applying the inequality constraints in (2.9b), we will have the updated α_1 and α_2. The active set is selected using two heuristics whose goal is to maximize the growth of the objective function $W(\cdot)$ at each iteration.

All these methods have the common property that they do a local search in the space of α's, and this can often be very inefficient. In the proposed *ClusterSVM*, we first partition the training data into disjoint clusters, then train an initial SVM using representatives of these clusters. This initial SVM gives us a *global* picture of the solution, such as the position and shape of decision boundary, the relative position between each datum and the decision boundary. Based on this information, we can approximately identify the support vectors and non-support vectors, and the training process is accelerated by replacing non-support vectors with few data.

2.3 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 disjoint clusters. The motiva-

tion 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 ($\alpha = C$).

Since each training datum corresponds to one row and column in the Gram matrix $\mathbf{H}$, replacing data in a cluster with a representative corresponds to replacing the rows and columns associated with these data with a single row and column. Let $\mathcal{D}$ denote the training data set consisting of two disjoint sets $\mathcal{D}_1$ and $\mathcal{D}_2$ and, without losing generality, assume $\mathcal{D}_1$ is a subset of the training data in class 1. Let $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ be the Lagrange multipliers of the data in $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, then the optimization problem (2.6) is equivalent to

$$\begin{aligned} \text{Maximize}: W(\boldsymbol{\alpha}) &= \left(\boldsymbol{\alpha}_1^T \mathbf{1}_1 - \frac{1}{2}\boldsymbol{\alpha}_1^T \mathbf{H}_{11}\boldsymbol{\alpha}_1\right) \\ &+ \left(\boldsymbol{\alpha}_2^T \mathbf{1}_2 - \frac{1}{2}\boldsymbol{\alpha}_2^T \mathbf{H}_{22}\boldsymbol{\alpha}_2\right) - \boldsymbol{\alpha}_1^T \mathbf{H}_{12}\boldsymbol{\alpha}_2 \qquad (2.10a) \\ \text{Subject to}: & 0 \le \alpha_{1,i} \le C,\ \forall i = 1,2,...,n_1 \\ & 0 \le \alpha_{2,j} \le C,\ \forall j = 1,2,...,n_2 \\ & \boldsymbol{\alpha}_1^T \mathbf{y}_1 + \boldsymbol{\alpha}_2^T \mathbf{y}_2 = 0, \qquad (2.10b) \end{aligned}$$

where $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1^T \ \ \boldsymbol{\alpha}_2^T]^T$. Let the index to the row and column that will replace the rows and columns associated with $\mathcal{D}_1$ be 0 and the label $y_0 = 1$, we have the following optimization problem

$$\begin{aligned} \text{Maximize}: W(\boldsymbol{\alpha}) &= \left(\alpha_0 - \frac{1}{2}\alpha_0 H_{00}\alpha_0\right) \\ &+ \left(\boldsymbol{\alpha}_2^T \mathbf{1}_2 - \frac{1}{2}\boldsymbol{\alpha}_2^T \mathbf{H}_{22}\boldsymbol{\alpha}_2\right) - \alpha_0 \mathbf{H}_{02}\boldsymbol{\alpha}_2 \qquad (2.11a) \\ \text{Subject to}: & 0 \le \alpha_0 \le n_1 C \\ & 0 \le \alpha_{2,j} \le C,\ \forall j = 1,2,...,n_2 \\ & \alpha_0 + \boldsymbol{\alpha}_2^T \mathbf{y}_2 = 0, \qquad (2.11b) \end{aligned}$$

where α_0 is the Lagrange multiplier corresponding to the representing row/column. Here, H_{00} is defined as

$$H_{00} = \frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_1} H_{ij}, \qquad (2.12)$$

and $\mathbf{H}_{02}$ is a row vector of length n_2 with entries defined as

$$H_{0j} = \frac{1}{n_1}\sum_{i=1}^{n_1} H_{ij}, \qquad (2.13)$$

where $j = 1, 2, \cdots, n_2$. Then, we have the following proposition.

PROPOSITION 2.1. *The optimization problem defined by equations* (2.11), (2.12) *and* (2.13) *is equivalent to the one obtained by adding a constraint to* (2.6) *that requires all Lagrange multipliers corresponding to the data in* $\mathcal{D}_1$ *be equal.*

Proof. Using equation (2.12), we have

$$\begin{aligned} \alpha_0 H_{00}\alpha_0 &= \alpha_0\alpha_0 \frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_1} H_{ij} \\ &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_1} \frac{\alpha_0}{n_1}\frac{\alpha_0}{n_1} H_{ij}. \qquad (2.14) \end{aligned}$$

Let $\boldsymbol{\alpha}_1^*$ be a vector of length n_1 with all components being equal to α_0/n_1, equation (2.14) can be written as

$$\begin{aligned} \alpha_0 H_{00}\alpha_0 &= \sum_{i=1}^{n_1}\sum_{j=1}^{n_1} \alpha_{1,i}^*\alpha_{1,j}^* H_{ij} \\ &= \boldsymbol{\alpha}_1^{*T}\mathbf{H}_{11}\boldsymbol{\alpha}_1^*. \qquad (2.15) \end{aligned}$$

Using the similar arguments, $\alpha_0\mathbf{H}_{02}\boldsymbol{\alpha}_2$ can be written as

$$\alpha_0 H_{02}\boldsymbol{\alpha}_2 = \boldsymbol{\alpha}_1^{*T}\mathbf{H}_{12}\boldsymbol{\alpha}_2. \qquad (2.16)$$

After substituting equations (2.15) and (2.16) into equation (2.11) and using the fact that $\sum_{i=1}^{n_1}\alpha_{1,i}^* = \alpha_0$, we arrive the following optimization problem

$$\begin{aligned} \text{Maximize}: W(\boldsymbol{\alpha}_1^*, \boldsymbol{\alpha}_2) &= \left(\boldsymbol{\alpha}_1^{*T}\mathbf{1}_1^* - \frac{1}{2}\boldsymbol{\alpha}_1^{*T}\mathbf{H}_{11}\boldsymbol{\alpha}_1^*\right) \\ &+ \left(\boldsymbol{\alpha}_2^T \mathbf{1}_2 - \frac{1}{2}\boldsymbol{\alpha}_2^T \mathbf{H}_{22}\boldsymbol{\alpha}_2\right) - \boldsymbol{\alpha}_1^{*T}\mathbf{H}_{12}\boldsymbol{\alpha}_2 \qquad (2.17a) \\ \text{Subject to}: & 0 \le \alpha_{2,j} \le C,\ \forall j = 1,2,...,n_2 \\ & 0 \le \alpha_{1,1}^* \le C \\ & \boldsymbol{\alpha}_1^{*T}\mathbf{y}_1 + \boldsymbol{\alpha}_2^T \mathbf{y}_2 = 0. \\ & \alpha_{1,1}^* = \alpha_{1,i}^*,\ \forall i = 2,...,n_1, \qquad (2.17b) \end{aligned}$$

where $0 \le \alpha_{1,i}^* \le C$ ($i = 1, 2, ..., n_1$) follows from the fact that $0 \le \alpha_0 \le n_1 C$. The proposition follows by comparing equation (2.10) and equation (2.17). □

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 (2.10). After replacing two data in class 1 by a representative, the feasible region of the resulting optimization

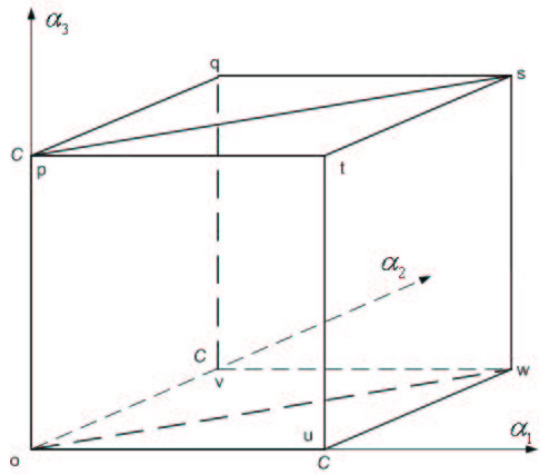

Figure 2: Illustration of Proposition 2.1.

problem (c.f. (2.17)) is the rectangle *opsw*. Thus, the feasible region of the problem (2.17) is a subset of that of the problem (2.10) and, by replacing clusters of training data with their representatives, the solution of the resulting optimization problem is an approximation to the solution of the original optimization problem. Further, it is not hard to show that the optimal solution of (2.17) is exactly the optimal solution of (2.10) 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 to be replaced by a representative.

- **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 (2.10) and that of the problem (2.17) 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 (2.10) and that of the problem (2.17) coincides at line *ws*.

Thus, we can safely replace clusters of above two types with a representative without influencing the solution and such replacement will reduce the size of the optimization problem. In addition, for the cluster $\mathcal{D}_i$ satisfying the above **Condition 1**, we can replace the corresponding rows and columns with the row and column associated with the pseudocenter of this cluster because the Lagrange multiplier of a non-support vector is zero, where the pseudocenter $\mathbf{x}^p(\mathcal{D}_i)$ of the cluster $\mathcal{D}_i$ is defined as

$$\mathbf{x}^p(\mathcal{D}_i) = \underset{\mathbf{x}\in\mathcal{D}_i}{\operatorname{argmin}} \left\| \mathbf{x} - \frac{1}{n_i}\sum_{k=1}^{n_i} \mathbf{x}_k \right\|_2, \tag{2.18}$$

where $\|\cdot\|_2$ means 2-norm and n_i is the number of data in $\mathcal{D}_i$. Ties are broken arbitrarily in (2.18). As a pilot study, only the cluster satisfying the **Condition 1** is replaced by its representative in the current algorithm.

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 only non-support vectors. A cluster is estimated to contain non-support vectors if it contains data with functional margin $d(\cdot)$ larger than 1. A cluster is estimated to contain support vectors if it contains data with functional margin $d(\cdot)$ less than 1. Clusters that are estimated to contain both support vectors and non-support vectors are 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. This idea is illustrated in Figure 1 and 3. Figure 1 shows the training data of a two-class classification problem and they are partitioned into 5 pair-wise disjoint clusters ($\mathcal{D}_{pos,1}$, $\mathcal{D}_{pos,2}$, $\mathcal{D}_{neg,1}$, $\mathcal{D}_{neg,2}$ and $\mathcal{D}_{neg,3}$). The representatives (that is, pseudocenters defined in equation 2.18) of these clusters are labeled with solid squares and solid cycles. An initial SVM was trained using these representatives, and its decision boundary ($d(\mathbf{x}) = 0$) and supporting hyper-planes ($d(\mathbf{x}) = \pm 1$) were shown in Figure 1. For the cluster belonging to the positive class, it is believed to contain only non-support vectors if the functional margin ($d(\mathbf{x})$) of its data are all larger than 1 (e.g. $\mathcal{D}_{pos,2}$) and, hence, it can be replaced by its pseudocenter without being split. However, a cluster belonging to the positive class is believed to contain both support vectors and non-support vectors if it contains some data with functional margin $d(\mathbf{x}) \leq 1$, which are likely to be support vectors, and some data with functional margin $d(\mathbf{x}) > 1$, which are believed to be non-support vectors. This kind of cluster is partitioned into two subclusters and it is believed that the subcluster having data with $d(\mathbf{x}) > 1$ contains only non-support vectors and can be replaced with its pseudocenter, while the other subcluster ($d\mathbf{x} \leq 1$) is believed to contain only support vectors. An example of such cluster is $\mathcal{D}_{pos,1}$ in Figure 1, which is partitioned into two subclusters along $d(\mathbf{x}) = 1$. Similar arguments

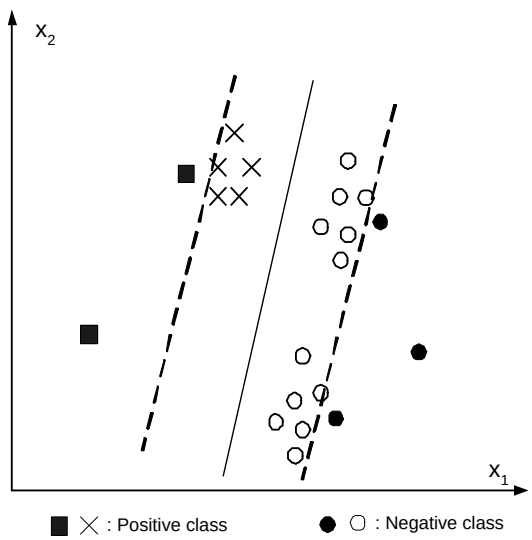

Figure 3: The reduced training data set $\mathcal{D}_{reduced}$ after splitting some clusters and replacing clusters and subclusters containing only non-support vectors with their representatives. The representatives are labeled with solid squares and solid cycles. For clarity purpose, the decision boundary shown in Figure 1 is kept here.

apply to the cluster belonging to the negative class. A cluster is believed to contain only non-support vectors if its data all satisfy $d(\mathbf{x}) < -1$, and a cluster is believed to contain both non-support vectors and support vectors if some of its data satisfy $d(\mathbf{x}) < -1$ and the other data satisfy $d(\mathbf{x}) \geq -1$. Using this criterion, cluster $\mathcal{D}_{neg,3}$ needs not to be split, while clusters $\mathcal{D}_{neg,1}$ and $\mathcal{D}_{neg,2}$ need to be split into two subclusters. After splitting some clusters and replacing the clusters and subclusters containing only non-support vectors with a representative, 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.

The proposed training algorithm *ClusterSVM* is detailed in Algorithm 1 and its properties are summarized in Proposition 2.2.

Proposition 2.2. *With reference to Algorithm 1 (*ClusterSVM*, see next page) and setting $NP_{max} = \infty$, we have*

1. *Algorithm 1 will converge after a finite number of passes through the* WHILE *loop (line* 5 *through* 18*).*
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*

$$yd(\mathbf{x}) > 1, \quad \forall \mathbf{x} \in \mathcal{D}_{unused}, \tag{2.19}$$

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

Proof. From line 13 and 14 in Algorithm 1, we can see that the size of the reduced training data set $\mathcal{D}_{reduced}$ is strictly increasing after each pass through the *WHILE* loop. Since there is a finite number of training data in $\mathcal{D}$, $\mathcal{D}_{reduced}$ will be the same as $\mathcal{D}$ after a finite number of passes through the *WHILE* loop, which means that Algorithm 1 will converge after a finite number of passes.

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

$$y_i d(\mathbf{x}_i) > 1 \Longrightarrow y_i \left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle_{\mathcal{H}} + b\right) > 1. \tag{2.20}$$

Knowing the fact that $\xi_i \geq 0$, this means that the constraint specified by equation (2.5b) will not be active, thus $\alpha_i = 0$. □

It should be noted that the second conclusion of Proposition 2.2 does not depend on how to partition $\mathcal{D}$ into $\mathcal{D}_{reduced}$ and $\mathcal{D}_{unused}$. As long as the condition specified in equation (2.19) is satisfied for all data in $\mathcal{D}_{unused}$, the SVM obtained using $\mathcal{D}_{reduced}$ is the same as that would be obtained using $\mathcal{D}$.

3 Experiments

3.1 Implementations Due to its popularity, the underlying training algorithm $\mathbb{A}$ we use in Algorithm 1 is Platt's SMO [27], and the *ClusterSVM* is compared with SMO. An implementation of SMO using C++ by Chang et al. [6] and its Matlab® [31] "mex" wrapper by Ma et al. [21] were used in this paper. However, it should be pointed out that, 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 PDDP and all the other codes were implemented using Matlab®. All experiments were run on a PC running Windows 2000 Server with one Pentium 4 2.8 GHz processor and 1GB RAM.

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 the initial 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

Algorithm 1 ClusterSVM: Two class SVM

Require: An underlying 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. (2.18))

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

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: $\quad \mathcal{G}^{old} \leftarrow \mathcal{G}$ and $\mathcal{G} \leftarrow \emptyset$
9: $\quad$ **for all** $\mathcal{D}' \in \mathcal{G}^{old}$ **do**
10: $\quad\quad$ **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**
11: $\quad\quad\quad Flag \leftarrow 1$
12: $\quad\quad\quad$ Split $\mathcal{D}'$ into $\mathcal{D}'_{sv}$ and $\mathcal{D}'_{nsv}$

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

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

time, it should not be too large since letting $k^+ = n^+$ and $k^- = n^-$ would make $\mathcal{D}_{reduced} = \mathcal{D}$, and there would be no speedup. Another reason for preferring small k^+ (k^-) is that both clustering the training data $\mathcal{D}$ and training the initial SVM needs to be performed very quickly. In this paper, the following square root heuristic is suggested

$$k^+ = round(\sqrt{n^+}) \text{ and } k^- = round(\sqrt{n^-}). \tag{3.21}$$

There are primarily two motivations for this heuristic. First, knowing the fact that the time for clustering typically scales linearly with the number of data [9], the square root heuristic can make the total time to obtain the initial SVM scale linearly with the number of training data. The second reason is that this heuristic has been suggested in the study of clustering algorithms (e.g. [8]). The effectiveness of this heuristic is demonstrated experimentally in Section 3.3. Since the initial SVM can approximate the true SVM quite well and each pass through the outer *WHILE* loop (line 5 to 18 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 $d(\cdot)$. In order to avoid repeated clusterings, 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 those $m-1$ classes.

3.2 Data sets There are three data sets examined in this paper. The kernel $K(\cdot,\cdot)$ and the regularization coefficient C used for each data set are chosen based on some experiments and experiences from the literatures if any.

- **Artificial data set** As shown in Figure 4, this is a three-class classification problem and each class consists of data drawn from a 2D normal distribution with covariance matrix being the 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. The kernel $K(\cdot,\cdot)$ and the regularization coefficient C used in

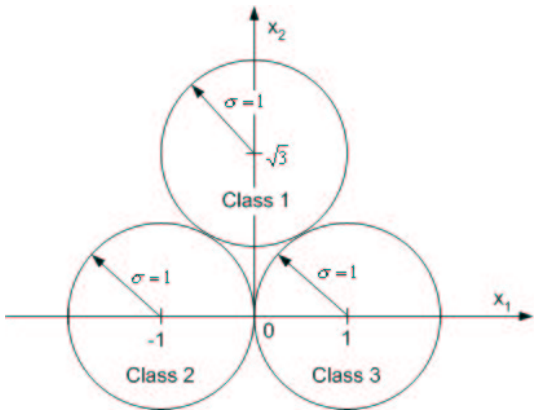

Figure 4: Artificial data set.

all 3 classifiers are

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \text{ and } C = 10000. \tag{3.22}$$

- **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 [19]. Each digit is represented as a 16×16 matrix whose entry ranges from -1 to 1. As suggested by [28], a smoothing operation using a Gaussian kernel with width 0.75 was applied to the image as a preprocessing step. With reference to [28] and after some initial experiments, the kernel $K(\cdot, \cdot)$ and the regularization coefficient C used in all 10 classifiers are

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\frac{\mathbf{x}_i^T \mathbf{x}_j}{256} \right)^3 \text{ and } C = 10. \tag{3.23}$$

- **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. After some initial experiments, the kernel $K(\cdot, \cdot)$ and the regularization coefficient C used in all 26 classifiers are

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \text{ and } C = 0.02. \tag{3.24}$$

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, 10 randomly generated training data set were tried.

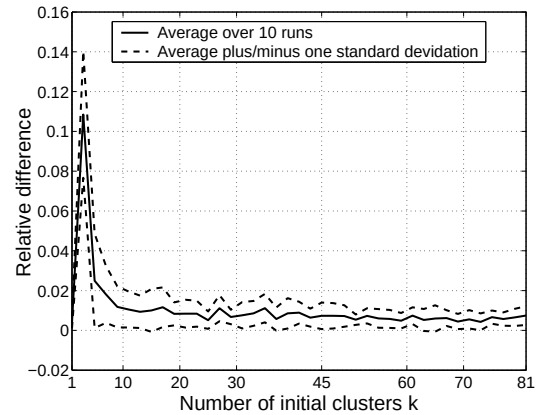

Figure 5: The performance of the initial SVM compared to that of true SVM as a function of the number of initial clusters k. There are 2000 training data in each class (6000 in total) and the square root heuristic corresponds to $k = 45$. The seemingly good performance of $k = 1$ comes from the symmetry of this problem and it has no general implications.

Figure 5 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

$$RD = \frac{|ER_{\text{initial}} - ER_{\text{true}}|}{ER_{\text{true}}}, \tag{3.25}$$

where ER_{initial} and ER_{true} are the error rate of the initial SVM and the true SVM on the same test data set. Figure 6 shows the time to obtain the initial SVM $T_{\text{Initial SVM}}$ as a function of k. $T_{\text{Initial SVM}}$ consists of time for clustering and the time for training the initial SVM. From Figure 5 and Figure 6, 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 18 in Algorithm 1) is shown in Table 1 for the artificial data set with 6000 training data. The SVM trained after 3 passes is the true SVM, which 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 NP_{max} in Algorithm 1 is set to 1. In addition, it can be seen from Table 1 that, with $NP = 0$, the initial SVM also gives pretty good result.

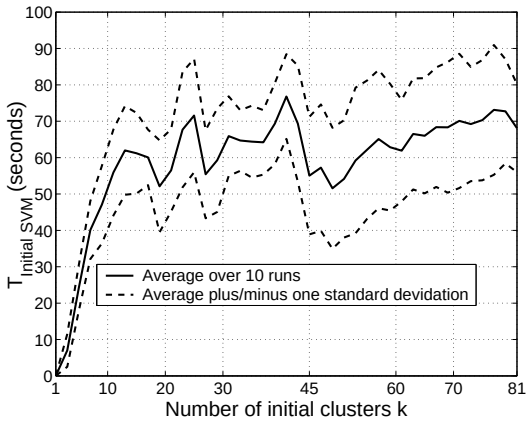

Figure 6: Time to obtain the initial SVM as a function of the number of initial clusters k. There are 2000 training data in each class (6000 in total) and the square root heuristic corresponds to $k = 45$.

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

$$Speedup = \frac{T_{SMO}}{T_{ClusterSVM}}, \tag{3.26}$$

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 clustering all training data. 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 only non-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 when *ClusterSVM* is used. This is within our expectations because of the symmetry of the artificial data set. However, for the USPS data set shown in Table 3, $N_{train,i}$ varies from one classifier to another when *ClusterSVM* is used. This is reasonable because 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 when *ClusterSVM* is used. 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 helps 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 clustering that makes the initial SVM approximate the true SVM quite well. At the same time, as shown in these tables, the overhead induced by clustering is only a small faction of total training time.

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

Finally, the scaling performance of *ClusterSVM* was shown in Figure 7 for the artificial data set, which shows the average training time over 10 runs. It can be seen that *ClusterSVM* scales better than SMO.

4 Conclusions

An efficient SVM training algorithm *ClusterSVM* was proposed in this paper and a significant speedup over SMO was observed on both the artificial data set

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

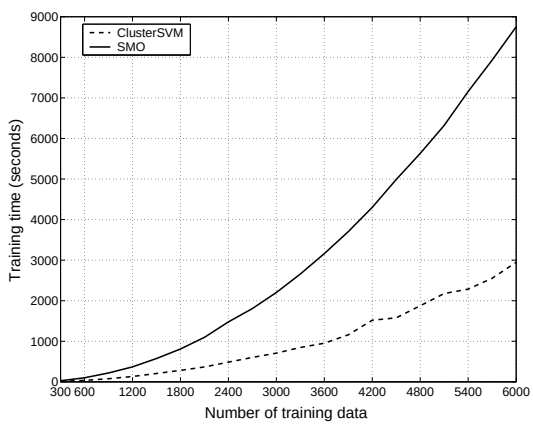

Figure 7: Comparison of the scaling performance of *ClusterSVM* and that of SMO on the artificial data set. The solid line represents SMO and the dashed line represents *ClusterSVM*. The number of training data varies from 300 to 6000 with an interval of 300. For clarity purpose, labels on the horizontal axis only show every 600.

and the real data set. The role of the clustering algorithm $\mathbb{C}$ in the proposed method is to provide a reasonable partition of the training data within the same class, based on which we can approximately identify the support vectors and nonsupport vectors. This is different from the usual goal of a clustering algorithm, which is to group similar data into the same cluster while separating different data into different clusters. Assuming we have a set of data drawn according to some probability distribution, we do not want the algorithm $\mathbb{C}$ to give clusters that reflect the "modes" of this distribution, but to give a reasonable partition of the data. The possible extensions to *ClusterSVM* are the follows.

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

- 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. Further, in the current implementation of *ClusterSVM*, all Lagrange multipliers are initialized to zeroes for SVM training within the *WHILE* loop (line 5 to 18 in Algorithm 1). A better initialization would be based on the solutions of the initial SVM. These two improvements would

definitely speed up the SVM training further, and they would especially benefit problems having large number of boundary support vectors where SMO usually converges slowly. A two-class problem will have a large number of boundary support vectors if the probability distributions of two classes are highly overlapped.

- 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.

Acknowledgements

This research was supported by NSF grant IIS-0208621. The authors would like to thank anonymous reviewers for their valuable comments.

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] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, M. Ares, and D. Haussler. Support vector machine classification of microarray gene expression data. Technical report, University of California, Santa Cruz, 1999.

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

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

[8] 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.

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

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

[11] G. M. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In *KDD 2001: The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

[12] G. M. Fung and O. L. Mangasarian. A feature selection Newton method for support vector machine classification. Technical Report 02-03, University of Wisconsin, Data Mining Institute, 2002.

[13] G. M. Fung and O. L. Mangasarian. Finite Newton method for Lagrangian support vector machine classification. Technical Report 02-01, University of Wisconsin, Data Mining Institute, 2002.

[14] G. M. Fung, O. L. Mangasarian, and A. J. Smola. Minimal kernel classifiers. *Journal of Machine Learning Research*, 3:303–321, 2002.

[15] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*. Springer, 1998.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Y.-J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational Optimization and Applications*, pages 5–22, October 2001.

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

[22] O. L. Mangasarian and D. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, March 2001.

[23] O. L. Mangasarian and D. R. Musicant. Active support vector machine classification. In T. K. Lee, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, pages 577–583, 2000.

[24] O. L. Mangasarian and D. R. Musicant. RSVM: Reduced support vector machines. In *SIAM International Conference on Data Mining*, 2001.

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

[26] D. Pavlov, D. Chudova, and P. Smyth. Towards scalable support vector machines using squashing. In

Knowledge Discovery and Data Mining, pages 295–299, 2000.

[27] 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.

[28] 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.

[29] L. Shih, J. D. M. Rennie, Y.-H. Chang, and D. R. Karger. Text bundling: Statistics-based data reduction. In *Twentieth International Conference on Machine Learning*, 2003.

[30] H. Shin and S. Cho. Fast pattern selection for support vector classifiers. In K.-Y. Whang, J. Jeon, K. Shim, and J. Srivastava, editors, *PAKDD*, volume 2637 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2003.

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

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

[33] 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.

[34] H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. In *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.