Fast SVM Training Algorithm with Decomposition on Very Large Data Sets

Jian-xiong Dong, Adam Krzyżak, *Senior Member*, *IEEE*, and Ching Y. Suen, *Fellow*, *IEEE*

Abstract—Training a support vector machine on a data set of huge size with thousands of classes is a challenging problem. This paper proposes an efficient algorithm to solve this problem. The key idea is to introduce a parallel optimization step to quickly remove most of the nonsupport vectors, where block diagonal matrices are used to approximate the original kernel matrix so that the original problem can be split into hundreds of subproblems which can be solved more efficiently. In addition, some effective strategies such as kernel caching and efficient computation of kernel matrix are integrated to speed up the training process. Our analysis of the proposed algorithm shows that its time complexity grows linearly with the number of classes and size of the data set. In the experiments, many appealing properties of the proposed algorithm have been investigated and the results show that the proposed algorithm has a much better scaling capability than Libsvm, SVM^{light}, and SVMTorch. Moreover, the good generalization performances on several large databases have also been achieved.

Index Terms—Support vector machines (SVMs), algorithm design and analysis, algorithm efficiency, machine learning, handwritten character recognition.

1 Introduction

Learning from the data is one of the basic ways humans perceive the world and acquire the knowledge. Nowadays, there are massive amounts of data available at an astonishingly increasing pace on the Internet and in industrial applications. There are classification tasks with a large number of classes such as the recognition of handwritten Chinese characters with more than 6,000 classes. In a problem of categorization of Web documents, gigabytes of data with high dimension are processed. Support vector machine (SVM) [1], [2], [3] has emerged as a good classification technique and has achieved excellent generalization performance in a wide variety of applications, such as handwritten digit recognition [4], [5], [6], categorization of Web pages [7], and face detection [8]. For a large-scale classification task, SVM has some advantages from an algorithmic perspective over other methods such as neural networks [9] and probability-based density estimation methods such as the Gaussian mixture models [10]. These models are usually trained by gradient-based methods and Expectation Maximization [11] and they are sensitive to the parameter initialization and get stuck easily in local minima. In the case of a large data set, it is difficult to choose good structures in order to control the complexity of these models unless we have rich prior knowledge. In a support vector machine, the optimization is done using a convex quadratic programming and the solution is not only explicitly defined, but also unique and sparse.[1] Once a suitable kernel for SVM is chosen, the structure of support vector classifier is data-driven and automatically determined. However, since the training kernel matrix grows quadratically with the size of the data set, training support vector machines on large data sets is a very slow process and it has become a bottleneck. Therefore, it is important to develop fast algorithms for training SVM to solve large-scale classification problems.

Although many methods for solving the optimization problem of support vector machines are available [3], we list here only the prominent ones which can be used to train SVMs on a large data set, such as Chunking [1], [8], Sequential Minimal Optimization (SMO) (see [12], [13]) and SVM^{light} [14]. The chunking algorithm starts with an arbitrary subset (chunk of data, working set) which can fit in the memory and solves the optimization problem on it by the general optimizer. Support vectors (SVs) remain in the chunk while other points are discarded and replaced by a new working set with gross violations of KKT (Karush-Kuhn-Tucker) conditions [15]. The rationale of this operation is that only support vectors contribute to the final form of a decision function. In addition, the chunking algorithm is based on the sparsity of SVM's solution. That is, support vectors actually take up a small fraction of the whole data set. But, one of the problems for the chunking algorithm is that there may be many more active candidate support vectors during the optimization process than the final ones so that their size can go beyond the chunking space. The method of selecting a new working set by evaluating KKT conditions without efficient kernel caching may lead to a high computational cost [8].

SMO, introduced by Platt [12] and improved by Keerthi et al. [13], further takes the decomposition idea to

- *J.-x. Dong and C.Y. Suen are with Centre for Pattern Recognition and Machine Intelligence, the Department of Computer Science and Software Engineering, Concordia University, 1455 de Maisonneuve Blvd. West, Suite GM-606, Montréal, Québec, Canada H3G 1M8. E-mail: {jdong, suen}@cenparmi.concordia.ca.*
- *A. Krzyżak is with the Department of Computer Science and Software Engineering, Concordia University, 1455 de Maisonneuve Blvd. West, Montréal, Québec, Canada. E-mail: krzyzak@cs.concordia.ca.*

Manuscript received 23 July 2003; revised 21 Sept. 2004; accepted 22 Sept. 2004; published online 10 Feb. 2005.
Recommended for acceptance by H. Shum.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-0187-0703.

1. Many kernel weights are equal to zero.

an extreme and optimizes the subsets of two points at each iteration. The power of SMO draws from the fact that no extra optimization package is required and an analytical solution for a two-point optimization problem can be given explicitly. Several heuristics have been suggested to select the working set. Keerthi et al. [13] further enhance the performance of SMO by pointing out the inefficiency of updating one-thresholded parameters in Platt's algorithm and replacing it with two-thresholded parameters. The important contribution of Keerthi et al.'s modification is that the pair of patterns chosen for optimization is theoretically determined by two-thresholded parameters and the optimization on this subset leads to a considerable advancement in the objective function. In practice, when the size of a data set grows bigger, the problem of determining the optimal pair at a low cost still exists.

SVM^{light} [14] is a general decomposition algorithm, where a good working set is selected by finding the steepest feasible direction of descent with q nonzero elements. The q variables that correspond to these elements compose the working set. When q is set equal to 2, Chang and Lin [16] pointed out that the selected working set corresponds to the optimal pair in Keerthi el al.'s modification of SMO. SVM^{light} caches q rows of kernel matrix (row caching) to avoid kernel reevaluations and LRU (Least Recently Used) is applied to update the rows in the cache.[2] However, when the size of the training set is very large, the number of cached rows becomes small due to the limited memory. As a result, the number of active variables is not large enough to achieve a fast optimization.

Flake and Lawrence [17] applied SMO to a large-scale regression. They pointed out that caching plays an important role in the speed-up of training and inefficiency of LRU policy. Moreover, they suggested that the training of SVM using SMO with iterative decomposition is more advantageous and proposed an initial prototype of sequential iterative optimization. Although some issues such as the construction of new working sets and stopping conditions were addressed there, the solutions to these problems were not efficient. The authors' proposed a strategy which was used to construct new subproblems by evaluating KKT condition violations is inefficient and will have a strong impact on the overall performance. The weakness of this strategy has been addressed in Osuna et al.'s remarks [8] and in our previous work [6].

Recently, some authors proposed the mixture of models to solve SVM's learning problem on a very large data set. Collobert et al. [18] proposed a parallel mixture of SVMs. The model first trained many SVMs on small subsets and then combined their outputs using a gater such as linear hyperplane or multilayer perceptron. Although the idea of the mixture of local experts is quite old, it differs from other mixture models by automatically assigning the training samples to different SVMs according to the prediction made by the gater of how well each local SVM performs on each example. The authors showed that on the UCI Forest database with 581,012 samples, their model was not only faster than a single SVM such as $\mathrm{SVMTorch}$ [19] when running on a platform with 50 CPUs, but also achieved a higher accuracy. However, there are two problems with their model. One is how to determine the optimal number of local SVMs, which is similar to the problem of determining the optimal number of centers in clustering techniques. The problem is usually very hard. The second problem is that in their model, no regularization term is used to control the complexity of the model globally, which will play a key role on the generalization performance. Although the authors claimed that their model achieved a higher accuracy on the UCI Forest database, unfortunately, they did not test their model on the MNIST handwritten digit database. On MNIST, the accuracies of different classifiers are well-known. For support vector machines, the good parameters for polynomial and radial basis kernels are also known. Rida et al. [20] proposed a similar model but without automatically assigning training samples as above. They did not test the performance of their model on a very large data set.

The other approach to train SVM on large data sets is to apply the Bayesian committee machine (BCM) [21] to the support vector machine resulting in Bayesian committee support vector machine (BC-SVM) [22]. In the BCM, the data set is divided into M subsets of the same size and M models are derived from the individual sets. The predictions of the individual models are combined using a weight scheme which is derived from a Bayesian perspective in the context of Gaussian process regression. That is, the weight for each individual model is the inverse covariance of its prediction. A good approximation requires that M subsets be pairwisely independent. Although the Bayesian committee support vector machine performs better than uniform mixture of individual SVMs on subsets, it has a slightly higher error rate than the full SVM on some data sets [22]. Also, the sparse property does not hold for the BC-SVM. As a result, its prediction is more time consuming than that of a standard SVM.

By reviewing the behavior of the above algorithms, we have concluded that they are inefficient on large data sets due to three key factors. The computational cost of training SVM primarily depends on kernel evaluations. Efficient kernel caching can reduce or avoid kernel reevaluations. The LRU caching policy may fail because elements of kernel matrix are usually accessed irregularly [17]. In addition, evaluating kernel elements on the fly is not efficient at all because data access is not temporally local. Massive evaluation of kernel elements via the blocking algorithm will reduce hardware cache misses and speed up the computation. The second problem originates from frequent access of portions of noncontiguous memories, which potentially can lead to high cache misses. Training SVM on a large data set usually requires the access of a substantial amount of memory. In a virtual memory system, accessing this memory irregularly will cause high Translation Look-aside Buffer (TLB) misses [23]. Consequently, memory access will take more time. Finally, although the existing algorithms can be used to train SVMs on a large data set with multiclasses, the computational cost is high. For example, with respect to the one-against-the-others training strategy [4] for multiclasses, the training cost for m classes is about m times as high as that for two classes.

The main contribution of this paper is to present efficient solutions to the above problems. A two step procedure has been designed to train support vector machines. The first step is called parallel optimization, in which the kernel matrix of a support vector machine is approximated by block diagonal matrices so that the original optimization problem can be decomposed into hundreds of subproblems, which can be easily and efficiently solved. The great

2. Here, cache means a part of memory, not hardware cache.

advantage of this step is to remove most nonsupport vectors quickly and collect training sets for the next step, called sequential working set algorithm. Some effective strategies such as kernel caching and good selection of working set are integrated to these two steps to speed up the training process. In addition, the Block Linear Algebra Subprogram (BLAS) [24], [25], which is optimized on Intel P4, can be used to efficiently calculate the kernel matrix. Experiments on the large MNIST handwritten digit database have shown that the proposed method has achieved a speed-up factor of one magnitude of order, compared with existing algorithms such as SVM^{light} and LIBSVM [16]. Although the proposed algorithm provides an approximation to the optimal solution, it does it without sacrificing the generalization performance when the size of working set is large enough to contain support vectors.

This paper is organized as follows: First, support vector machines are introduced. Then, in Section 3, a fast training algorithm for SVM is presented. A discussion of efficient implementation strategies is given in Section 4. In Section 5, we analyze the space and runtime complexity of the proposed algorithm. Section 6 presents extensive experiments which have been conducted to investigate the properties of the proposed algorithm on very large databases and summarizes its generalization performance on those databases. Finally, we conclude this paper with some remarks.

2 Support Vector Machine

Let $\{\mathbf{x}_i, y_i\},\ i = 1, \ldots, l,\ y_i \in \{-1, 1\}$, and $\mathbf{x}_i \in \mathbb{R}^n$ be the training samples where $\mathbf{x}_i$ is the training vector and y_i is its corresponding target value. In order to make notation consistent, we use lowercase bold letters, for example, $\mathbf{x}$, to denote column vectors and uppercase bold letter to denote matrices. The notation $(\mathbf{A})_{ij}$ is used to denote the ith row and the jth column element of matrix $\mathbf{A}$. Boser et al. [26] showed that training support vector machine for a pattern recognition problem can be formulated as the quadratic optimization problem:

$$\begin{aligned} &\text{maximize}: \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\alpha^T \mathbf{Q}\alpha \\ &\text{subject to}: 0 \le \alpha_i \le C,\ \ i = 1, \ldots, l \\ &\qquad\qquad \sum_{i=1}^{l} y_i \alpha_i = 0, \end{aligned} \tag{1}$$

where α is a vector of length l and its component α_i corresponds to a training sample $\{\mathbf{x}_i, y_i\}$, $\mathbf{Q}$ is an $l \times l$ semidefinite kernel matrix, and C is a parameter chosen by the user. A larger C assigns a higher penalty to the training errors. The training vector $\mathbf{x}_i$ whose corresponding α_i is nonzero is called support vector. Support vector machine maps training vector $\mathbf{x}_i$ into a high-dimensional feature space by the function $\mathbf{\Phi}(\mathbf{x})$ such that $(Q)_{ij} = y_i y_j (K)_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}^T(\mathbf{x}_i)\mathbf{\Phi}(\mathbf{x}_j)$. When the above optimization problem is solved, we can obtain an optimal hyperplane in a high-dimensional feature space to separate the two-class samples. The decision function is given by

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) - b\right), \tag{2}$$

where

$$\operatorname{sgn}(u) = \begin{cases} 1, & \text{for } u > 0, \\ -1, & \text{for } u < 0. \end{cases} \tag{3}$$

In the latter algorithm, we resort to a technique proposed by Keerthi et al. [13], [27] to select two variables for optimization and determine the stopping conditions. First, we split training patterns into five sets:

1. $I_0 = \{i : 0 < \alpha_i < C\}$,
2. $I_1 = \{i : y_i = 1,\ \alpha_i = 0\}$,
3. $I_2 = \{i : y_i = -1,\ \alpha_i = C\}$,
4. $I_3 = \{i : y_i = 1, \alpha_i = C\}$, and
5. $I_4 = \{i : y_i = -1,\ \alpha_i = 0\}$.

Then, we define

$$b_{\text{up}} = F_{i\_\text{up}} = \min\{F_i : i \in I_0 \cup I_1 \cup I_2\}, \tag{4}$$

$$b_{\text{low}} = F_{i\_\text{low}} = \max\{F_i : i \in I_0 \cup I_3 \cup I_4\}, \tag{5}$$

where $F_i = \sum_{j=1}^{l} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) - y_i$. Keerthi et al. [13] showed that conditions for an optimal solution to (1) hold at some $\boldsymbol{\alpha}$ if and only if $b_{\text{low}} \le b_{\text{up}}$. Moreover, the worst violating pair of patterns $(i\_\text{up}, i\_\text{low})$ are chosen for optimization, which can lead to a large increase in the objective function [3].

3 A Fast Algorithm for Training SVM

For the optimization described in (1), the key problem is that the dense kernel matrix Q cannot be stored into memory when the number of training samples l is very large. We can observe the fact that the optimal solution to (1) still holds if any nonsupport vector is removed. Moreover, numerous experiments [4], [6] have shown that support vectors actually constitute only a small fraction of the training samples. If most nonsupport vectors can be removed quickly at the first step, SVM training can be accelerated dramatically. Divide-and-conquer[3] is a general principle for solving complex problems. Thus, we can divide the original problem (1) into small subproblems which can be solved easily. Since kernel matrix $\mathbf{Q}$ is symmetric and semipositive definite, its block diagonal matrices are semipositive definite, and can be written as

$$\mathbf{Q}_{\text{diag}} = \begin{bmatrix} \mathbf{Q}_{11} & & & \\ & \mathbf{Q}_{22} & & \\ & & \ddots & \\ & & & \mathbf{Q}_{kk} \end{bmatrix}, \tag{6}$$

where $l_i \times l_i$ square matrices $\mathbf{Q}_{ii}$, $i = 1, \ldots, k$, $\sum_{i=1}^{k} l_i = l$ are called block diagonal. Thus, we obtain k optimization subproblems:

3. A good example in computer science is the **quick-sort** algorithm.

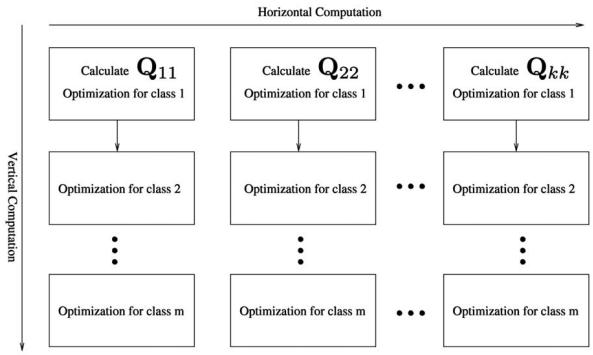

Fig. 1. Parallel optimization diagram.

$$\begin{aligned} &\text{maximize} : \sum_{j=1}^{l_i} \alpha_j^{(i)} - \frac{1}{2}\alpha^{(i)T}\mathbf{Q}_{ii}\alpha^{(i)} \\ &\text{subject to} : 0 \le \alpha_j^{(i)} \le C, \quad j = 1, \ldots, l_i \\ &\qquad \sum_{j=1}^{l_i} y_j^{(i)}\alpha_j^{(i)} = 0, \end{aligned} \tag{7}$$

where $i = 1, \ldots, k$ and $\boldsymbol{\alpha}^{(i)}$ is a vector of length l_i. Optimization of each subproblem in (7) is equivalent to that in (1) with the constraints that l_i components of $\boldsymbol{\alpha}$ are free and the rest are set to zero. Optimization of k subproblems can be used to remove nonsupport vectors quickly in (1) based on the assumption that the set of nonsupport vectors in (7) is a subset of those in (1). Further, we extend this approach to train support vector machine with multiclasses, where the one-against-the-others classification strategy is used. The computational diagram is depicted in Fig. 1.

The computation in Fig. 1 is efficient due to three aspects. First, kernel matrix can be effectively divided into block diagonal matrices such that each of them can fit into the memory. Many off-the-shelf algorithms can be used to solve this problem. Second, for vertical computation, all classes share the same block kernel matrix,[4] which needs to be calculated once. Finally, after the calculation of the block matrices in the first row, optimizations from the second class to the mth class are independent. Also, computations on different columns are independent. The computation framework is suitable for parallel optimization on the architecture of multiprocessors since optimization independence and data locality can maximize parallelism and reduce the cost of long latency remote communication in using multiprocessors [23].

In the above parallel optimization, only principal diagonal block matrices are used and the classification information on the off-diagonal block matrices of $\mathbf{Q}$ is completely ignored. Therefore, in the next step, a solution must be found to recover such information. After the above parallel optimization, most nonsupport vectors for each class will be removed from the training set. Then, a new training set for each class can be obtained by collecting support vectors from the optimization subproblems in the same row, as shown in Fig. 1. In the kernel submatrix which is constructed from this new training set of small size, most of the useful classification information on the off-diagonal block matrices of $\mathbf{Q}$ will be restored. So, the step of parallel optimization can be regarded as an effective data filter. Although the size of the new training set is much smaller than that of the original one, the memory may not be large enough to store the kernel matrix, especially when dealing with a large data set. Therefore, a fast sequential working set algorithm for training SVM[5] is summarized as follows:

Fast Sequential Optimization for Training SVM

Input: Training set is S, and the fixed size of the working set is d, where $d \le l$ and l is the size of the training set. Also, square kernel caching matrix with the dimension d is provided.

Output: α_i, $i = 1, \ldots, l$.

Initialization: Shuffle the training set; set α_i to zero and select a working set B such that $B \subseteq S$.

Optimization:

Repeat

1. Apply modified SMO to optimize a subproblem in working set B, in combination with some effective techniques such as kernel caching, efficient computation of kernel matrix, then update α_i.

2. Select a new working set with a queue technique.

Until the specified stopping conditions are satisfied.

The above algorithm can also be used as the optimizer for parallel optimization in Fig. 1, where the size of the working set is the same as that of training subset since each block diagonal matrix can be stored into the memory, and Step 2 is skipped.

4 Strategies of Implementation

Section 3 provides a general computational framework for training support vector machines. For efficient computation, one needs to take into account some issues such as kernel caching, the computation of kernel matrix, selection of a new working set, and stopping conditions.

4.1 Kernel Caching

Kernel cache in this paper is defined as a part of contiguous memory that stores $d \times d$ square kernel matrix on the working set. The size of the working set d should be large enough to contain all support vectors in the whole training set and small enough to satisfy the memory constraint. Since kernel matrix on the working set is completely cached, each element of the kernel matrix needs to be evaluated only once and must be calculated via fast method presented later before starting the optimization, so that all kernel elements are available during the optimization.

4.2 Optimization on the Working Set

In our optimizer, the worst violating pair of patterns $(i\_\text{up}, i\_\text{low})$ is always updated according to (4), (5), instead of choosing it from I_0, or the previous i_up or i_low (Keerthi et al. SMO [13]). The strategy of choosing the worst violating pair for optimization is also used in SVMTorch [19]. Since kernel elements are completely cached, the computational cost of evaluating F_i in (4), (5) is low. Updating F_i can be done efficiently as follows:

4. In fact, the shared kernel matrix is K without multiplication of target values.

5. The initial prototype of this algorithm has already been suggested in [17].

$$\begin{aligned}\Delta F_i = {} & y_{i\_\mathrm{up}} \Delta\alpha_{i\_\mathrm{up}} K(\mathbf{x}_i, \mathbf{x}_{i\_\mathrm{up}}) \\ & + \; y_{i\_\mathrm{low}} \Delta\alpha_{i\_\mathrm{low}} K(\mathbf{x}_i, \mathbf{x}_{i\_\mathrm{low}}),\end{aligned} \tag{8}$$

where $i = 1, \ldots, d$, $\Delta\alpha_{i\_\mathrm{low}} = \alpha_{i\_\mathrm{low}}^{\mathrm{new}} - \alpha_{i\_\mathrm{low}}^{\mathrm{old}}$, and

$$\Delta\alpha_{i\_\mathrm{up}} = \alpha_{i\_\mathrm{up}}^{\mathrm{new}} - \alpha_{i\_\mathrm{up}}^{\mathrm{old}}.$$

F_i is also cached. We use the following pseudocode to illustrate the simple optimization procedure.

Optimization on the Working Set

1. **Initialization**: $F_i = -y_i$, $i = 1, \ldots, d$.
2. **Loop**
   - 2.1 Select worst violating pair $(i\_\mathrm{up}, i\_\mathrm{low})$ and calculate b_{up} and b_{low}.
   - 2.2 If $b_{\mathrm{low}} < b_{\mathrm{up}} + 2\tau$, then goto 3.
   - 2.3 Update α_{i_up}, α_{i_low}.
   - 2.4 Update F_i in terms of (8), where $i = 1, \ldots, d$.
   - 2.5 Update I_0, I_1, I_2, I_3, I_4.
3. Load a new working set, update F_i and kernel matrix.
4. If global stopping conditions are satisfied, the algorithm terminates; otherwise goto 2.

Here, τ is a positive tolerance parameter. In the above loop, the number of multiplication operations for updating F_i is of order $O(d)$; the number of comparison operations for selecting the worst violating pair is of order $O(d)$.

4.3 Selection of a New Working Set

After optimization on the current working set is finished, a new data set will be loaded to replace nonsupport vectors by queue operations. Two operations associated with a queue data structure are defined by

1. **Enqueue**$(Q_S, id(\mathbf{x}))$: Append the index[6] of a sample $\mathbf{x}$ to the rear of he queue Q_S,
2. **Dequeue**(Q_S): Remove the index of a sample from the front of the queue S and return it.

Operation $id(.)$ returns the index of a sample in set S. Each operation above takes $O(1)$ time. The queue initially stores the indices of all training samples S. An important step prior to that is to shuffle the training set randomly before starting parallel optimization such that distribution of training samples from different classes is balanced. Suppose that the working set is stored in an array $\mathbf{B}[1, \ldots, i, \ldots, d], i = 1, \ldots, d$, where each array element is an n-dimensional vector. The index set of nonsupport vectors is a subsequence $\{i_1, \ldots, i_k, \ldots, i_{\mathrm{nsv}}\}$, $k = 1, \ldots, \mathrm{nsv}$, where nsv is the number of nonsupport vectors of the current working set. Then, the algorithm of selecting a new working set is summarized as follows:

Algorithm for Selecting a New Working Set

Initialization: $\mathbf{B}[i] \leftarrow S[\mathbf{Dequeue}(Q_S)]$, $i = 1, \ldots, d$.

Selection:

$k \leftarrow 0$.

Repeat

1. $\mathbf{Enqueue}(Q_S, id(B[i_k]))$.

2. $\mathbf{B}[i_k] \leftarrow S[\mathbf{Dequeue}(Q_S)]$.

3. $k \leftarrow k + 1$.

Until $k = \mathrm{nsv}$.

6. Here, the index denotes the sequential number of a sample in the total training set S.

After a new working set is loaded, the kernel matrix on the new working set must be updated via the method in Section 4.4. In addition, only F_i for these new training samples need to be calculated. Moreover, for a large data set, only the working set is required to be in the memory. One of the good properties of the above method is that I/O (Input/Output) access is efficient since training samples in the data file are accessed sequentially when a new working set is loaded.

4.4 Calculation of Kernel Matrix

When a kernel can be represented as a function of dot product, matrix multiplication can be used to calculate kernel elements efficiently. Obviously, three kernels such as linear, polynomial kernel, and radial basis function (RBF) belong to this type. Here, we describe the details of computing RBF kernel matrix, which can be easily extended to other types of kernel. RBF kernel can be written as

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(- \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 / (2\sigma^2)), \tag{9}$$

where $\parallel . \parallel$ is the Euclidean norm and

$$\parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2 = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j - 2\mathbf{x}_i^T \mathbf{x}_j, \tag{10}$$

$i, \; j = 1, \ldots, d$. Terms $\mathbf{x}_i^T \mathbf{x}_i$ can be calculated by calling CBLAS function **cblas_sdot**. Let $\mathbf{A} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d)$. The Gram matrix $\mathbf{A}^T\mathbf{A} = (\mathbf{x}_i^T \mathbf{x}_j)$, $i, \; j = 1, \ldots, d$ can be easily calculated by calling CBLAS function **cblas_ssyrk**. Due to symmetry of the kernel matrix, only elements in the upper triangle are calculated for the implementation of **cblas_ssyrk**.

The kernel matrices in Fig. 1 can be calculated via the above method, where $d = l_i$ in (6). To update the kernel matrix when a new working set is loaded, we do not need to compute the total kernel matrix since some elements can be reused. Let the new working set be split into two sets: support vector set represented by the array $\mathbf{B}_{\mathrm{sv}}$ and nonsupport vector set $\mathbf{B}_{\mathrm{nsv}}$. Updating the total kernel matrix requires $\mathbf{B}_{\mathrm{sv}}^T \mathbf{B}_{\mathrm{sv}}$, $\mathbf{B}_{\mathrm{nsv}}^T \mathbf{B}_{\mathrm{nsv}}$ and $\mathbf{B}_{\mathrm{sv}}^T \mathbf{B}_{\mathrm{nsv}}$. Since kernel elements $K(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i \in \mathbf{B}_{\mathrm{sv}}$ and $\mathbf{x}_j \in \mathbf{B}_{\mathrm{sv}}$, can be reused, $\mathbf{B}_{\mathrm{sv}}^T \mathbf{B}_{\mathrm{sv}}$ does not need to be recalculated. $\mathbf{B}_{\mathrm{nsv}}^T \mathbf{B}_{\mathrm{nsv}}$ and $\mathbf{B}_{\mathrm{sv}}^T \mathbf{B}_{\mathrm{nsv}}$ can be evaluated by calling CBLAS function **cblas_ssyrk** and **cblas_sgemm**.

The rationale of the usage of the BLAS package is its computation efficiency, portability, and maintenance. The key computational kernel of the BLAS package such as matrix multiplication is implemented by hardware vendors in assembly language, which makes efficient use of cache, memory, and instructions such as single instruction and multidata (SIMD) [28] on Intel Pentium series or vector instructions in vector processors [23]. Moreover, BLAS has been efficiently implemented on different platforms, which enables the proposed SVM algorithm to perform well across platforms. Furthermore, the performance of the proposed method improves with the increase of computational power of a processor in the future when a new BLAS package is plugged in. Consequently, the cost of software maintenance will be reduced.

4.5 Reduction of Cache and TLB Misses

In Fig. 1, CBLAS function **cblas_ssyrk** is used to calculate kernel matrices $\mathbf{Q}_{ii}$. **cblas_ssyrk** stores results into the upper triangle of the symmetric kernel matrix. Thus, we need to copy the elements in the upper triangle into the lower triangle so that SMO can access contiguous memory

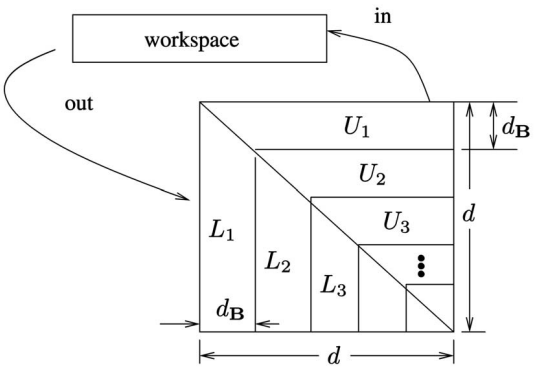

Fig. 2. Copy upper-triangle elements into lower-triangle via a workspace.

during the optimization. Crude copy operations will result in high TLB misses and cache thrashing, which lead to a high cost. The reason is that kernel matrix takes a large portion of paged memory and the limited TLB does not contain all memory page numbers for the kernel matrix. In order to solve this problem, the upper-triangle of the kernel matrix is broken up into contiguous blocks of a fixed size $d_{\mathbf{B}}$ and a workspace with the size of $d_{\mathbf{B}}d$ is allocated.[7] Fig. 2 shows how to copy elements into the lower-triangle.

The workspace is used to copy blocks of the upper-triangle into the symmetric blocks of lower-triangle. $d_{\mathbf{B}}$ can be determined by the size of the second-level cache (L2) if available. Similarly, the method can be used to copy elements of $\mathbf{B}_{\text{nsv}}^T\mathbf{B}_{\text{nsv}}$ and $\mathbf{B}_{\text{sv}}^T\mathbf{B}_{\text{nsv}}$ into the kernel matrix during the optimization on the sequential working sets.

4.6 Insert Positive Samples into the Working Set

In Fig. 1, it is possible that there is no positive sample[8] on some working set since the size of negative samples is much larger than that of positive samples for one-against-the-other multiclass training strategy. As a result, SMO will lead to an incorrect solution. In order to tackle this problem, we first randomly collect one sample from each class. During the training, if no positive sample on a working set is found, we replace the first negative sample by the positive sample from the collected set.

4.7 Stopping Conditions

With respect to the stopping conditions, most algorithms evaluate KKT conditions at each training sample. If none violates KKT, the algorithm stops. On a large data set, the computational cost of this step is high. Instead, heuristic stopping rules are suggested. We track the variations of b_{up}, b_{low} and the number of support vectors on the two successive working sets. If all these variations are small and a sufficient number of training samples have been learned, the algorithm terminates. The heuristic stopping rules are given by

$$\begin{aligned}&(\ |\Delta \text{sv}| < 20 \ \text{ and } \ |\Delta b_{\text{up}}| < 2\tau \ \text{ and } \ |\Delta b_{\text{low}}| < 2\tau \\ &\text{and Number of learned samples} > l\ ) \\ &\text{or } \ (\ \text{sv} \geq d - 1\ ) \\ &\text{or } \ (\text{Number of learned samples} > T \cdot l),\end{aligned} \tag{11}$$

where $|\ .\ |$ computes the absolute value and l is the size of training set. Adding the constraint on the number of learned samples is necessary. Otherwise, in an extreme case, if the loaded samples are all nonsupport vectors, the algorithm is likely to terminate too early. As a result, the optimization is incomplete and SVM will suffer from the risk of poor generalization performance. T is a user-defined parameter which controls the maximal number of loops to allow the iterative algorithm to go through the entire training set. T must be greater than 1.0. Furthermore, although the above stopping conditions and parallel optimization may lead to a "not-too-precise" optimal solution, they are essential to the control of the computational cost within a reasonable bound.

At the second stage, when the number of support vectors grows above the maximum size of the working set, which depends on the size of memory available, the data set collected from the parallel optimization stage is divided into several subsets again and we train one SVM on each subset. Finally, the outputs of these SVMs are uniformly combined to make the final decision.

5 Analysis of Space and Runtime Complexity

Before analyzing the space and runtime complexity[9] of the proposed algorithm, we introduce some notation:

- l: size of original training set.
- d: size of the working set.
- n: dimension of the feature vector.
- m: number of classes.
- k: number of working sets in Fig. 1 and $k = l/d$.[10]
- $h_{ij},\ i = 1, \ldots, m;\ j = 1, \ldots, k$: number of support vectors on the working set at the ith row, jth column in Fig. 1.
- $P_i,\ i = 1, \ldots, m$: the size of training set for class i at the second stage, where $P_i = \sum_{j=1}^{k} h_{ij}$.
- $\mu_i,\ i = 1, \ldots, m$: the factor that shows how many nonsupport vectors are removed at the parallel optimization stage, where $P_i = \mu_i l$.
- $r_{ij},\ i = 1, \ldots, m;\ j = 1, \ldots, T_i$: number of support vectors on the jth working set of class i, where T_i denotes the number of optimized working sets of class i when the optimization is terminated at the second stage. For the sake of analysis, let $r_{i0} = 0$.
- $\beta_i,\ i = 1, \ldots, m$: the factor that shows how many samples have been learned when the optimization is terminated at the second stage, where

$$\beta_i P_i = \sum_{j=0}^{T_i - 1} (d - r_{ij}).$$

For simplicity, the size of working set d is used in both parallel and sequential optimizations. In addition, we assume that the major computational cost comes from multiplication operators without considering the cost of other operators such as memory access. The following assumptions are made in the analysis:

- Only general kernel type $K(\mathbf{x}_i^T\mathbf{x}_j)$ is considered. The cost of each multiplication operation is c_1 and the cost of evaluating $K(.)$ is c_2.

7. Align the starting address so that it begins at a cache-line boundary.
8. Its corresponding desired output is 1.0.
9. Analysis is based on a single processor.
10. For the sake of analysis, we assume that $l \bmod d$ is zero.

- The size of the working set is large enough and the final number of support vectors of SVM for each class is not larger than d.
- During the optimization on each working set, the number of support vectors is monotonically increasing according to a negatively accelerated function $\mathrm{sv} = d(1 - \exp(-\lambda t)),\ \lambda > 0$, where t is the iterative step.

The simplified support vector growing model can be deduced using the idea similar to that in the estimation of part reliability in industrial engineering [30]. Let $\mathrm{nsv}(t)$ be the number of nonsupport vectors at time $t\ (\geq 0)$. The probability $F(t)$ of the number of support vectors at time t is represented by

$$F(t) = \frac{d - \mathrm{nsv}(t)}{d}. \tag{12}$$

Thus,

$$\mathrm{nsv}(t) = d(1 - F(t)), \tag{13}$$

then growing rate $h(t)$ of support vectors is given by

$$h(t) = -\frac{d\mathrm{nsv}(t)}{dt}/\mathrm{nsv}(t). \tag{14}$$

Substitution of (13) into (14) yields

$$h(t) = \frac{\dot{F}(t)}{1 - F(t)}. \tag{15}$$

We integrate (15) on both sides over the interval $[0, t]$. Using initial condition $F(0) = 0$, we obtain

$$F(t) = 1 - \exp\left(-\int_0^t h(x)dx\right). \tag{16}$$

For the sake of analysis, let the growing rate $h(t)$ be a constant function λ. Since $\mathrm{sv}(t) = F(t)d$, we have

$$\mathrm{sv}(t) = d(1 - \exp(-\lambda t)). \tag{17}$$

The negatively accelerated function is widely used in psychology to explain the physiological response to the physical stimuli [29] and in the first-order goal-seeking negative feed-back control system. In Ben-Hur et al.'s recent work [31], the continuous computation model is portrayed as solution of ordinary differential equations, which converges to the attracting fixed points that become the system outputs. In the vicinity of the attracting fixed point, the model has exponential convergence rate. From the theoretical perspective, the number of support vectors in SVM optimization can be regarded as a state variable of an autonomous system that evolves with the gradient field of the cost function. Since SVM optimization is a convex programming problem, the local property may hold globally. Intuitively, at the beginning, the objective cost in (1) is increasing fast so that support vectors are growing rapidly. When the objective cost is close to the optimal maximum, the number of support vectors increases slowly.

5.1 Space Complexity

The requirements of storage space in the two steps are different. In both steps, only a working set is loaded into the memory, rather than the total training set. Usually, n is much smaller than d. The predominant storage comes from the kernel matrix. In Fig. 1, its size is estimated as $4d^2$ bytes.[11] In the second algorithm, since some kernel elements are reused, we cannot totally overwrite the kernel matrix and a new matric with the same size is required to calculate $\mathbf{B}_{\mathrm{nsv}}^T\mathbf{B}_{\mathrm{nsv}}$ and $\mathbf{B}_{\mathrm{sv}}^T\mathbf{B}_{\mathrm{nsv}}$. Its total storage space for sequential working set optimization is estimated as $8d^2$ bytes.

5.2 Analysis of Runtime Complexity

It is difficult to analyze precisely the runtime complexity of the algorithm. Therefore, we just estimate dominant computational cost based on the above assumptions. In Fig. 1, the computation cost of kernel matrices is given by

$$g_{\mathrm{ker}}^{(1)} = k\left(\frac{1}{2}nd^2c_1 + \frac{1}{2}d^2c_2\right). \tag{18}$$

The factor $1/2$ appears because **cblas_ssyrk** updates only the upper-triangle of the kernel matrix. Now, we estimate the cost of optimization on the working set. For each iteration, updating F_i in (8) contributes to the main cost, which is $4dc_1$. The number of iterative steps on the working set is

$$t = \frac{1}{\lambda}\ln\left(\frac{d}{d - \mathrm{sv}}\right). \tag{19}$$

Then, the cost of optimization can be approximated by

$$\begin{aligned} g_{\mathrm{op}}^{(1)} &= \sum_{i=1}^{m}\sum_{j=1}^{k}\frac{4dc_1}{\lambda}\ln\left(\frac{d}{d - h_{ij}}\right) \\ &= \frac{4dc_1}{\lambda}\sum_{i=1}^{m}\sum_{j=1}^{k}\ln\left(1 + \frac{h_{ij}}{d - h_{ij}}\right) \\ &\leq \frac{4dc_1}{\lambda}\sum_{i=1}^{m}\sum_{j=1}^{k}\frac{h_{ij}}{d - h_{ij}}. \end{aligned} \tag{20}$$

After the parallel optimization step, we estimate the cost of the second stage. The costs of computing $\mathbf{B}_{\mathrm{sv}}^T\mathbf{B}_{\mathrm{nsv}}$ and $\mathbf{B}_{\mathrm{nsv}}^T\mathbf{B}_{\mathrm{nsv}}$ on the jth working set of class i are $r_{i(j-1)}(d - r_{i(j-1)})(c_1n + c_2)$ and $\frac{1}{2}(d - r_{i(j-1)})^2(c_1n + c_2)$, respectively. Thus, the total cost of updating kernel matrices can be approximated by

$$\begin{aligned} g_{\mathrm{ker}}^{(2)} &= \sum_{i=1}^{m}\sum_{j=0}^{T_i-1}(r_{ij}(d - r_{ij})(c_1n + c_2) \\ &\quad + \frac{1}{2}(d - r_{ij})^2(c_1n + c_2)) \\ &= (c_1n + c_2)\sum_{i=1}^{m}\sum_{j=0}^{T_i-1}(d - r_{ij})\frac{d + r_{ij}}{2} \\ &\leq (c_1n + c_2)\sum_{i=1}^{m}\sum_{j=0}^{T_i-1}(d - r_{ij})d \\ &= (c_1n + c_2)d\sum_{i=1}^{m}\beta_iP_i. \end{aligned} \tag{21}$$

The iterative step on the jth working set of class i is given by

11. The data type of a kernel element is float. The size of a float data type is 4 bytes.

$$t_{ij} = \frac{1}{\lambda}\ln\left(\frac{d}{d - r_{ij)}}\right) - \frac{1}{\lambda}\ln\left(\frac{d}{d - r_{i(j-1)}}\right) = \frac{1}{\lambda}\ln\left(\frac{d - r_{i(j-1)}}{d - r_{ij}}\right). \quad (22)$$

The cost of optimization on the working sets is

$$\begin{aligned} g_{\text{op}}^{(2)} &= \sum_{i=1}^{m}\sum_{j=1}^{T_i} t_{ij} 4d \\ &= \sum_{i=1}^{m} \frac{4c_1 d}{\lambda}\ln\left(\frac{d}{d - r_{iT_i}}\right) \\ &= \sum_{i=1}^{m} \frac{4c_1 d}{\lambda}\ln\left(1 + \frac{r_{iT_i}}{d - r_{iT_i}}\right) \\ &\le \frac{4c_1 d}{\lambda}\sum_{i=1}^{m}\frac{r_{iT_i}}{d - r_{iT_i}}. \end{aligned} \quad (23)$$

By adding expressions (20) and (23), the upper bound of the total cost for SMO optimization is given by

$$\begin{aligned} g_{\text{op}} &\le \frac{4c_1 d}{\lambda}\left(\sum_{i=1}^{m}\sum_{j=1}^{k}\frac{h_{ij}}{d - h_{ij}} + \sum_{i=1}^{m}\frac{r_i T_i}{d - r_{iT_i}}\right) \\ &\le \frac{4c_1 d}{\lambda}\left(\sum_{i=1}^{m}(d-1) + \sum_{i=1}^{m}\sum_{j=1}^{k}(d-1)\right) \\ &< \frac{4c_1 d^2}{\lambda}(m + km) \\ &= \frac{4c_1 d}{\lambda}\left(1 + \frac{d}{l}\right)ml. \end{aligned} \quad (24)$$

Similarly, by adding (18) and (21), the upper bound of the total computational cost of updating kernel matrix is

$$\begin{aligned} g_{\text{ker}} &\le \frac{1}{2}kd^2(c_1 n + c_2) + (c_1 n + c_2)d\sum_{i=1}^{m}\beta_i P_i \\ &= (c_1 n + c_2)d\left(\frac{1}{2}kd + \sum_{i=1}^{m}\beta_i P_i\right) \\ &= (c_1 n + c_2)dl\left(\frac{1}{2} + \sum_{i=1}^{m}\beta_i \mu_i\right) \\ &\le (c_1 n + c_2)dl\left(\frac{1}{2} + \sum_{i=1}^{m} T\right) \\ &= \left(T + \frac{1}{2m}\right)d\left(c_1 + \frac{c_2}{n}\right)lmn, \end{aligned} \quad (25)$$

where T is the parameter in stopping conditions (11). Comparing expression (24) with (25), it can be seen that updating kernel matrix dominates the computational cost and the factors that influence the overall performance appear in the upper bound, such as float multiplication, kernel function evaluation, and the dimension of input data. Moreover, for a fixed working set size d, m, and n, the upper bound of cost scales linearly with the size of training set l. Finally, we conclude the section with the following remarks:

- The nonasymptotical cost of updating kernel matrix is calculated precisely. Its upper bound does not depend on the assumption that the number of SVs is increasing according to the negatively accelerated function.
- During the procedure of deducing the growing model of support vectors, we assume that $h(t)$ is a constant function and not precise. A better model for $h(t)$ may be required in order to obtain a good estimation of SMO optimization cost.
- In (25), we can see that, when the size of training set l and the number of classes m are very large and the number of support vectors, which is controlled by the size of working set d, is relatively small, the above upper bound shows that the proposed algorithm is computationally efficient.

6 Experimental Results

In this section, we investigate various properties of the proposed algorithm and compare it with the existing algorithms on large data sets. Our algorithm shows excellent performance on two very large commercial data sets: Hanwang handwritten digits and Hanwang handwritten Chinese characters.

The code was compiled by Microsoft visual C++6.0. All experiments were conducted on a PC with single Intel P4 1.7Ghz processor with 256K L2 (second-level) cache, SDRAM[12] of 1.5 Gigabytes and 200 G hard disk (7200 RPM). The operating system was Windows 2000 Professional.

MNIST [32] handwritten digit database consists of 60,000 training samples and 10,000 testing samples, which originate from NIST database. The preprocessing was done by LeCun's research group [32] and a linear transform was performed such that all patterns were centered at 28×28 window while keeping the aspect ratio. The pixel values of resulting gray-scale images were scaled to fall in the range from -1.0 to 1.0.

Hanwang handwritten digit database is a large commercial database from Hanwang, an OCR company in China. It consists of 1,321,718 training samples and 300,000 testing samples. The samples are in binary format. Some samples in this database are illustrated in Fig. 3.

Hanwang handwritten Chinese character database is a large commercial database from Hanwang company. It consists of four subsets A, B, C, and D with characters of varying image quality. There are 3,755 categories, each of which has about 800 samples. In our experiment, we randomly split the data set into two parts: 2,144,489 for training and 542,122 for testing. Some samples from this database are shown in Fig. 4.

For character recognition, discriminative feature extraction is an important step to enhance the generalization performance of a learning algorithm. In our experiments, features for handwritten digits and handwritten Chinese characters are extracted based on the gradients of an image. For handwritten digits, a 576-dimensional feature vector [6] is obtained from the normalized patterns of size 22×22 and a 400-dimensional feature vector from those of size 18×18. On Hanwang handwritten digit database, a 576-dimensional feature vector is first extracted. Then, its dimension is

12. Single Data Rate Memory.

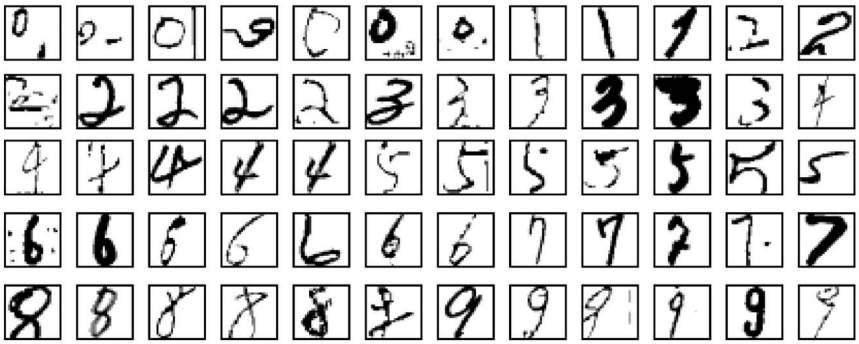

Fig. 3. Some samples from Hanwang digit database.

Fig. 4. Some samples from Hanwang handwritten Chinese character database.

reduced to 120 by principal component analysis [33]. For handwritten Chinese characters, a feature vector of size 1,296 is generated [34]. Then, its dimension is reduced to 392 by multiple discriminant analysis [35].

The parameters C in (1) and τ are set to 10.0 and 0.01, respectively. T in stopping conditions (11) is set to 6. ATLAS [36] (Automatically Tuned Linear Algebra Software)[13] optimized on P4 is the default BLAS package for computing the kernel matrix. The σ^2 in (9) is set to 0.3, unless mentioned otherwise. This value is chosen using a cross-validation method on a subset [34].

6.1 Algorithm Properties and Comparisons of Training Performance

6.1.1 Computation of Kernel Matrix

Three methods of computing the kernel matrix are tested on Hanwang handwritten digit database. One is simple C implementation of **cblas_sgemm** and **cblas_ssyrk**; the other two methods are BLAS packages optimized on P4: ATLAS and MKL6.0.[14] The RBF kernel is used. The size of the working set d is set to 8,000. Training performance measures are shown in Table 1, where μ_i, β_i, and T_i are defined in Section 5, and BSV denotes the number of bounded support vectors. Note that the size of the working set for class 3 is set to 8,500 at the second step. It can be observed from Table 1 that μ_i is less than 0.1, which implies that most nonsupport vectors are removed at the step of parallel optimization. In addition, most β_i are close to 1.0. That is, during sequential optimization step, the algorithm usually goes through the training set once and stops. In order to see how the kernel computation influences the training speed, in Table 2 we compare the performance of the three methods listed above.

In Table 2, the kernel computation dominates the cost for C implementation, which is consistent with the complexity analysis in Section 5, while it does not seem to be true for ATLAS and MKL6.0. In fact, ATLAS and MKL6.0 are both optimized on P4 and efficiently make use of SIMD instructions, where several computations are done with a single instruction in parallel [37]. That is, parallelism exists in ATLAS and MKL6.0. When we analyze the computational complexity, kernel elements are assumed to be calculated sequentially. In addition, compared with C implementation, ATLAS achieves a speed-up factor of about 3 on P4.

13. Available from http://math-atlas.sourceforge.net/.

14. Available from http://developer.intel.com/software/products/mkl/index.htm.

TABLE 1
Performance Measures for ATLAS on Hanwang Handwritten Digit Database

Class	0	1	2	3	4	5	6	7	8	9
SV	7208	3256	7639	8255	6541	7221	5066	5506	7245	5876
BSV	65	441	30	88	89	47	114	452	35	354
μ_i	0.107	0.040	0.112	0.114	0.099	0.108	0.073	0.076	0.091	0.055
β_i	1.00	1.96	1.00	1.00	1.00	1.00	1.08	1.09	1.00	1.59
T_i	69	18	101	102	46	67	24	28	57	34

6.1.2 The Size of Working Set

In Section 5, we claim that there exists an optimal size of the working set which achieves the minimal cost. Therefore, we continued to test the proposed method on Hanwang digit database and to observe how the size of the working set affected the performance. Fig. 5 shows the training time and average number of support vectors with the growing size of the working set. It can be seen that the training time and average number of support vectors are growing with the size of the working set ($\geq 8,000$), but the growth rate is slow. That is, training time is not sensitive to this parameter in a large range such that the users do not need to tune this parameter. Moreover, although the average number of support vectors is increasing, the substitution error rate on Hanwang testing set remains unchanged (0.5 percent). This fact indicates that a good setting of working set size can remove the redundant support vectors to some extent. For the optimal training speed, it is better to set different working sizes at these two steps. Moreover, at the second step, the size of the working set is chosen to be close to the number of support vectors in each class.

6.1.3 Testing Different Kernels and Comparing Performance with Existing Algorithms

In this experiment, we tested the proposed method on MNIST database with RBF and polynomial kernel and on Hanwang handwritten digit database with RBF kernel, and compared its performance with that of existing SVM packages such as SVM^{light},[15] LIBSVM,[16] and SVMTorch2.[17] For RBF kernel, a 576-dimensional discriminative feature vector was extracted [6]. With respect to the polynomial kernel, we directly applied SVM on 28×28 pixel images. Since patterns in the original MNIST are not truly centered, further preprocessing was performed by first enclosing the pattern in the rectangular bounding box, and then by translating this rectangle to the center of a 28×28 box. Subsequently, patterns were smoothed using 3×3 mask: The center element is set to $1/2$; the rest $1/16$.

Next, we used DeCoste and Schölkopf [5] idea to normalize each pattern by its Euclidean-norm scalar value such that the dot product was always within $[-1, 1]$. We used the polynomial kernel $(\mathbf{x}_1 \cdot \mathbf{x}_2)^7$. The dimension of input vectors was 784 (28×28). The size of the working set was still kept at $8,000$. There exists a fast method for computing general polynomial kernel $(\mathbf{x}_1 \cdot \mathbf{x}_2)^q$, where q is a positive integer. Let $u_- = \lfloor \log_2 q \rfloor$ and $u_+ = \lceil \log_2 q \rceil$. Then,

$$u = \begin{cases} u_- & \text{if } u_- + q - 2^{u_-} < u_+ + 2^{u_+} - q, \\ u_+ & \text{otherwise.} \end{cases} \tag{26}$$

The polynomial kernel can be calculated by

$$(\mathbf{x}_1 \cdot \mathbf{x}_2)^q = \overbrace{((((\mathbf{x}_1 \cdot \mathbf{x}_2)^2)^2)^{\cdots})^2}^{u} (\mathbf{x}_1 \cdot \mathbf{x}_2)^{q-2^u}. \tag{27}$$

The number of multiplication operations is $u + |q - 2^u|$. For example, $(\mathbf{x}_1 \cdot \mathbf{x}_2)^7 = (((\mathbf{x}_1 \cdot \mathbf{x}_2)^2)^2)^2 (\mathbf{x}_1 \cdot \mathbf{x}_2)^{-1}$.

In order to ensure that the comparisons are fair, the following preconditions are assumed:

- dense feature vector that is stored in a contiguous memory,
- similar size of cache that stores kernel elements,
- the fixed stopping tolerance,
- the fixed kernel parameters and C in (1), and
- the fixed experimental platform.

The post condition is that each method should achieve a similar error rate on the test set. Before we compare the performance, some facts on Libsvm and SVM^{light} are provided. Libsvm and SVMTorch2 both efficiently implement Keerthi et al's SMO and use a simple LRU row caching strategy, which is the same as SVM^{light}s. The different shrinking strategies are used in all three software packages. The stopping conditions for Libsvm and SVMTorch2 are

$$b_{\text{low}} < b_{\text{up}} + 2\tau, \tag{28}$$

and termination criteria for SVM^{light} are

$$\begin{array}{lcll} y_i(F_i + b) & \geq & -\tau & \forall i \;\; \alpha_i = 0, \\ |y_i(F_i + b)| & < & \tau & \forall i \;\; 0 < \alpha_i < C, \\ y_i(F_i + b) & \leq & \tau & \forall i \;\; \alpha_i = C, \end{array} \tag{29}$$

where τ is a stopping tolerance and it is set to 0.01. In addition, SVM^{light} skips checking KKT conditions of inactive variables before it terminates. The cache sizes on MNIST and Hanwang handwritten digit databases are set to 250 M[18] and 400 M, respectively. The working set size for the proposed method is set to 8,000. Table 3 compares the total training time of three methods on MNIST and Hanwang handwritten digit databases. It can be seen that the proposed method performs best on both databases and its training time scales well with the size of the training set. Also, the above table indicates that SVM^{light} and Libsvm perform much better when the one-against-one training strategy for multiclasses is used, rather than the one-against-others strategy. The

15. Available from http://svmlight.joachims.org/.
16. Available from http://www.csie.ntu.edu.tw/~cjlin/libsvm/.
17. Available from http://www.idiap.ch/learning/SVMTorch.html.
18. Megabytes. $1M = 2^{20}$ bytes.

TABLE 2
Performance Comparisons of Three Methods for Kernel Computation

Methods	ATLAS	MKL6.0	C
$g_{\mathrm{Ker}}^{(1)} + g_{\mathrm{Ker}}^{(2)}$(seconds)	1957	2265	7401
$g_{\mathrm{op}}^{(1)} + g_{\mathrm{op}}^{(2)}$ (seconds)	755	1169	758
Total training time (seconds)	2712	3434	8159

reason is that for one-against-others strategy, the limited cache only stores a small number of rows of kernel matrix when the size of the training set is large so that cache thrashing occurs frequently and LRU caching policy is prone to failure. In addition, although the same training method was used in Libsvm and SVMTorch2, SVMTorch2 performed better than Libsvm since SVMTorch2 had a better implementation of the LRU caching method and better shrinking strategy. Moreover, we can see that the computational cost of SVM^{light} and Libsvm is very high when the training strategy for multiclasses is one-against-others. It is worth mentioning that four methods have achieved almost the same substitution error rate on MNIST test set (0.6 percent with RBF kernel, 1.24 percent with polynomial kernel) and 0.5 percent on Hanwang test set. More details can be found on HeroSvm2 Web page.[19]

6.1.4 The Cost Gap between the Approximated Solution and Precise One

It is important to investigate how accurate the approximated solution is since some training samples are removed at the parallel stage and the heuristic stopping rules in (11) are used in the proposed method. At present, an analytic answer to the problem is not available. So, we can observe the gap in the experiments. The experiment was carried out on MNIST handwritten digit database. Libsvm and the proposed method were used to calculate the dual cost in (1). The parameter settings are the same, as in Section 6.1.3. The SVM training method for multiclasses is one-against-the-rest. We calculated the absolute cost gap and relative one by

$$\mathrm{gap} = |\mathrm{cost}_a - \mathrm{cost}_b|$$
$$\text{relative gap} = \frac{|\mathrm{cost}_a - \mathrm{cost}_b|}{\mathrm{cost}_a} \times 100\%,$$

where cost_a and cost_b specify the dual cost of the Libsvm and the proposed method, respectively. The results of experiments are shown in Table 4. It can be seen in Table 4 that the relative gap is small when SVMs are trained with RBF kernel. Although the gap is relatively large with polynomial kernel, it is still surprising to see that the proposed method achieved the same test error rate (1.24 percent) as others.

6.1.5 Scaling Behavior with Training Size

Although we have shown in the section dealing with algorithm complexity that the proposed algorithm linearly scales with the size of training set, it is important to validate this claim experimentally. We first divide the training set of each class on Hanwang handwritten digit database into 10 equal parts. Then, we randomly choose several parts from the set of each class to construct 10 training sets, which grow by 10 percent. Then, we fix the kernel parameters and the size of the working set, and train SVMs on each training set. The results are shown in Fig. 6. It can be seen that the algorithm scales linearly with the size of training set, which is consistent with the theoretical analysis.

6.1.6 IO Efficiency

With respect to the proposed method, only the working set, rather than the whole training set, is required to be stored in memory. A new working set is loaded into memory from a data file by queue operations such that data are accessed sequentially most of the time. As a result, the IO buffer for file reading operations is efficiently utilized and file access time is reduced considerably. On Hanwang handwritten digit database, when the size of the working set is set to 8,000, the total IO time is only 14.39 seconds, which is much less than the total training time of 2,712 seconds.

6.1.7 Performance on a Large Database with Thousands of Classes

This experiment was performed on Hanwang handwritten Chinese character database. The σ^2 parameter of RBF kernel in (9) can be obtained according to the method in [34] and is set to 0.8. For the proposed algorithm, the size of the working set at two steps was set to 8,000 and 3,000, respectively. The total training time was about 19 hours. With respect to Libsvm and SVM^{light}, their computational cost of Libsvm on this database was prohibitively high based on a one-against-other training strategy for multiclasses. Therefore, we only tested SVM^{light} using a one-against-one strategy. Its training time was approximately 644 hours. Also, in our experiment, the training speed of SVM^{light} was high on the data set of one pair of classes since the number of samples for each class was small, just about 800. However, its time complexity grows quadratically with the number of classes (m), whereas the dominant cost of the proposed algorithm proves to scale linearly with m in (25) for sufficiently large m. This experimental result indicates that the proposed algorithm's advantage over Libsvm and SVM^{light} is more obvious on databases of huge size with a large number of classes.

6.1.8 Performance on UCI Forest Database

The above databases are mostly noise-free so that the number of support vectors does not grow above the maximum size of the working set. In this section, we test the proposed algorithm on UCI forest data set [42][20] to observe its behavior when the above restriction is violated. The misclassification rate on this data set is high on the previous benchmarks [42], [18], [39]. The data set contains 581,012 samples with seven classes. The dimension of the feature vector is 54. The details about this data set are given in [42]. We randomly split the samples into the training set and testing set by class (75 percent for training, 25 percent for testing). As a result, the sizes of training and testing sets are 435,756 and 145,256, respectively. Before the training, we linearly scale each component of the feature vector into

19. Available from http://www.cenparmi.concordia.ca/.

20. Available from http://www.ics.uci.edu/ mlearn/MLSummary.html.

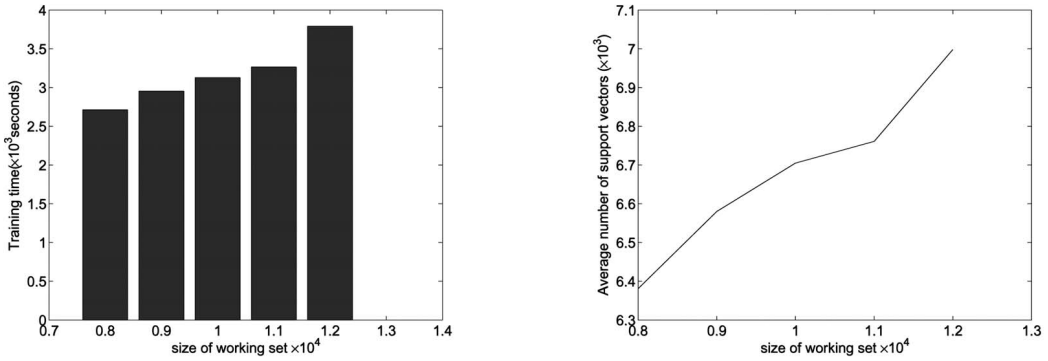

Fig. 5. (a) Training time versus the size of working set. (b) The average number of support vectors versus the size of working set.

TABLE 3
Comparisons of the Total Training Time of Three Methods (Hours)

Database		Proposed method	SVM^{light}(v5.0)		Libsvm(v2.4)		SVMTorch2
			A	B	A	B	A
MNIST	RBF	0.064	1.11	0.37	3.04	0.39	0.68
	POLY	0.075	2.03	0.54	4.52	0.54	1.18
Hanwang digit	RBF	0.75	-	7.63	-	16.21	16.1

TABLE 4
Cost Gap between the Approximated and Precise Solutions

Kernel	Method	0	1	2	3	4	5	6	7	8	9
RBF	Libsvm	282	411	507	539	607	519	387	689	706	927
	Proposed	282	424	506	536	600	517	385	675	702	933
	Gap	0	13	1	3	7	2	3	13	4	6
Relative gap (%)		0	3.16	0.2	1.12	1.15	0.39	0.78	1.89	0.57	0.65
POLY	Libsvm	617	1108	1203	1544	1656	1499	967	1926	2061	2723
	Proposed	604	1166	1157	1494	1478	1386	1008	1807	1807	2792
	Gap	13	58	46	50	178	113	41	119	254	69
Relative gap (%)		2.12	5.23	3.82	3.24	10.7	7.5	4.24	6.18	12.3	2.53

The digits in the first row denote different classes.

the interval $[0, 1]$, then the power transformation $x^{0.4}$ is applied to these components so that the data distribution is Gaussian-like [33]. In order to get a good parameter for RBF kernel, we first choose two small subsets: one for training and the other for validation. The parameter is tuned on the validation set. Then, the kernel parameter $\sigma^2 = 0.1$ in (9) is obtained. The size of working set at the parallel optimization stage is equal to 8,000.

After the parallel optimization ended, we found that the sizes of collected training sets for class "Spruce-Fir" and " Lodgepole" at the second stage were 212,975 and 225,628, respectively, which were quite large and almost half size of the original training set. In addition, most support vectors are bounded. This indicates that the samples for two classes may contain noises or the classification boundary is very complex. When we trained SVM for class "Spruce-Fir" at the second stage and the size of working set was set to 8000, the number of support vectors growed above the size of working set. Therefore, we divided the training set of each class into several equal-size subsets and set the new size of working set to 12,000. Then, we trained SVMs on these subsets. Finally, we uniformly combined the outputs of these SVMs to make a final decision. The average number of support vectors on these subsets for each class are shown in Table 5. The training times of two stages were 1,319 and 4,943 seconds, respectively. So, the total training time was about 104 minutes. The classification accuracy is shown in Table 6. It can be seen that

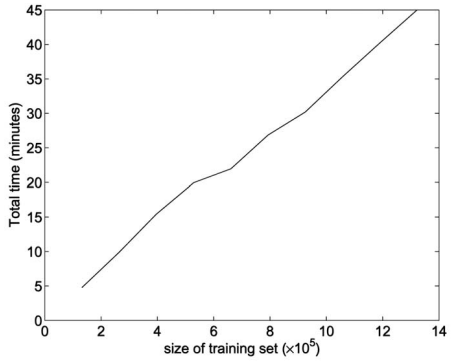

Fig. 6. The total training time versus the size of training set.

TABLE 5
Average Number of Support Vectors for Each Class ($\overline{\text{sv}}$)

Class	SF	LP	PD	WL	AP	DF	KH
$\overline{\text{sv}}$	11866	11916	11901	1970	9482	11851	8108
$\overline{\text{bsv}}$	8844	8927	8564	928	7963	10432	6573
Number of SVMs	13	16	2	1	1	2	1

SF: Spruce-Fir; LP: Lodegpole; PD: Ponderosa; WL: Willow; AP: Aspen; DF: Douglas-Fir; KH: Krummholz. $\overline{\text{bsv}}$: the average number of bounded support vectors.

TABLE 6
Classification Confusion Matrix on the Setting Sets

Class	SF	LP	PD	WL	AP	DF	KH
SF	44012	8356	8	0	104	20	460
LP	6390	63434	251	4	346	328	73
PD	7	337	8038	51	19	487	0
WL	0	3	72	591	0	21	0
AP	52	704	50	0	1556	12	0
DF	6	298	454	26	6	3552	0
KH	427	33	0	0	0	0	4668
Testing error(%)	16.9	10.4	10.7	13.9	34.4	18.2	8.9

SF: Spruce-Fir; LP: Lodgepole; PD: Ponderosa; WL: Willow; AP: Aspen; DF: Douglas-Fir; KH: Krumholz.

most misclassifications occur in classes "Spruce-Fir," "Lodgepole," "Aspen," and "Douglas-Fir."

Since different divisions of training and testing sets are used in the benchmark test, it is not easy to compare the performance fairly. But, it is worth mentioning the Collobert et al's results [39]. They considerd only one binary classification for class "Lodgepole" using one-against-all strategy. Their training and testing sets consist of 100,000 and 50,000 samples, respectively. The experiments were conducted on the cluster with 50 Athlon 1.2Ghz CPUs. The testing error rate was about 9.28 percent for the hard mixture of SVMs and the total time was 37 minutes. The training time was 119 minutes on a single CPU. But, when the size of the training set was increased to 400,000 and local experts were changed to multilayer perceptrons, other than SVM, the hard probability mixture of MLPs achieved 5.61 percent testing error rate on the binary classification for class "Lodgepole" and the training time was 17 minutes.

6.2 Performance on a Large Artificial Data Set

The purpose of this section is to test how well the proposed algorithm can handle a very large size of training set. In this experiment, the proposed method is tested on an artificially generated data set, called *ringnorm* [38], which has 20 dimensions and two classes. Class 1 is multivariate normal with a mean of zero and a covariance matrix four times the identity. Class 2 has a unit covariance matrix and mean (a, a, ..., a), where $a = \frac{2}{\sqrt{20}}$.[21] According to Fukunaga and Krile's method [41], the precise Bayes' error rate for the above two-class classification problem can be computed by numerical integration yielding 1.24 percent. The generated training and testing sets consist of 100 million and 3 million samples, respectively, where two classes have the same *priori* probability.[22] The σ^2 of RBF kernel in (9) and C are set to 198.0 and 20.0, respectively. At the parallel optimization step, the size of the working set is 8,000. After parallel optimization ends, the size of training set for the sequential optimization is 5,807,025. Then, we divide the set into many subsets, each of which consists of 14,000 samples. The size of the working set remains at 8,000. At this stage, we observe that, on each subset, the number of support vectors is about 7,900 and among these support vectors the number of bounded ones is about 7,700. The total training time and IO time are 15.89 hours and 316 seconds, respectively. The final decision is made using SVMs on these subsets in a majority vote method: For each unseen pattern, the output of each SVM is uniformly added together. This strategy is similar to that in the ensemble methods such as Bagging [40]. Using the above rule, the error rate on the test set is 1.23 percent, a little lower than the Bayes' error rate. Theoretically, the classification accuracy of any classifier cannot exceed that of the optimal Bayes classifier. The bias may come from the fact that the data generated by a computer does not exactly characterize the two-class multivariate Gaussian distributions. Moreover, we test each SVM independently on the test set and observed that their error rates are almost the same. Their mean and standard deviation are 0.0126 and 7.0×10^{-5}, respectively. This indicates that statistically each subset is sufficient for the classification.

6.3 The Role of Parallel Optimization

In Section 4.7 and experiments on UCI forest and artificial databases, we proposed an approximated solution when the number of support vectors grows above the maximum size of the working set. It is necessary to investigate its classification accuracy, compared with the main algorithm and how it relates to the solution of uniform SVM mixture models on the subsets. Those experiments have been conducted on MNIST database. We first shuffle the training set and divide it into subsets with equal size and combine the SVM solutions on these subsets uniformly. The sequential optimization algorithm has been used to train SVMs on the subsets without the step of parallel optimization. For the different sizes of subsets,

21. In [38], a is $\frac{1}{\sqrt{20}}$.
22. Data source is referred to http://www.cs.toronto.edu/~ delve/data/ringnorm.

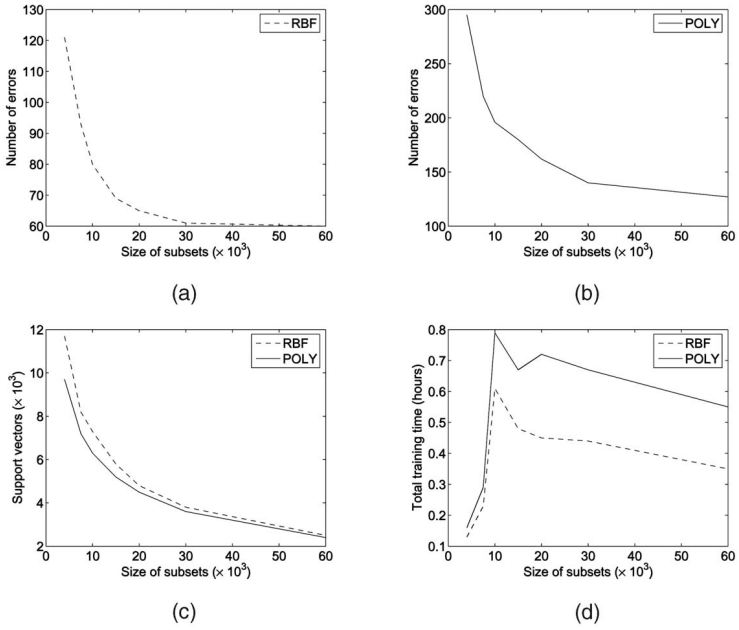

Fig. 7. Properties of uniform mixture of SVMs scale with the size of subsets without parallel optimization step. (a) Classification accuracy for RBF, (b) classification accuracy for POLY, (c) support vectors, and (d) total training time.

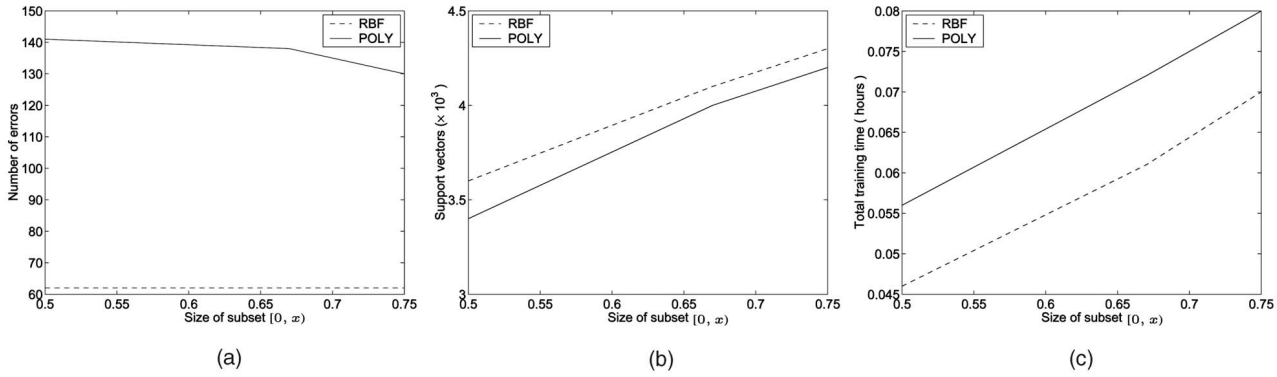

Fig. 8. Performance measures of uniform mixture of two SVMs versus x, the size of the set $[0, x)$ after parallel optimization is applied. (a) Classification accuracy, (b) support vectors, and (c) total training time.

the overall classification errors, the average number of merged support vectors and the total training time are shown in Fig. 7.

In Fig. 7, when the size of subsets is fixed, we add the number of support vectors together on subsets for each class. Then, the average number of support vectors over all classes is calculated and plotted. It can be observed that the classification accuracy is increasing with the growing size of subsets and the average number of support vectors for each class is decreasing. For RBF kernel, the uniform mixture of SVMs has achieved the optimal performance when the size of subsets is set to the half size of the total training set. That is, for each class there are only two SVM experts. But, the whole training time is about 0.44 hours, which is much higher than the training time (0.064 hours) of the proposed method shown in Table 3. In the experiments, when parallel optimization step is applied, we proportionally split the collected training set into two sets defined by the two intervals: $[0, x)$ and $[1 - x, 1]$, where $0.5 \leq x \leq 1$. The greater the x (i.e., the size of $[0, x)$ is, the more overlap exists between the two sets. The degree of overlap can be measured by the size of $[0, x) \cap [1 - x, 1]$, i.e., $2x - 1$, $0.5 \leq x \leq 1$. For example, when x equals 0.5, 0.75, and 1, the degree of overlap between the two subsets is 0 percent, 50 percent, and 100 percent, respectively.

The uniform mixture of SVMs on two subsets is used to make the final decision. Some of performance measures are shown in Fig. 8. The results are plotted in terms of x, which is proportional to the degree of overlap between the two subsets. From the comparisons of performance measures in Fig. 7 and Fig. 8, we can observe that parallel optimization step boosts the performance dramatically. In addition, the classification accuracy for a uniform mixture of SVMs can be improved when the size of subsets is increased. For the main proposed method, instead of uniform mixture, SVM is trained on the whole collected training set. Performance measures are summarized in Table 7. Among the above three measures, the main proposed method performs best. When parallel optimization step is applied and the number of support vectors grows above the maximum size of the working set, the uniform mixture of SVMs provides a good solution, but not the optimal one.

TABLE 7
Performance Measures of the Main Proposed Method

Database		Training time (hours)	Errors	$\overline{sv}$
MNIST	RBF	0.064	60	2447
	POLY	0.075	124	2428

Average number of support vectors for each class ($\overline{sv}$).

7 Conclusions

We have presented an efficient training algorithm for support vector machines on several databases as well as a huge database with thousands of classes. This algorithm consists of two steps: parallel optimization and sequential optimization. At the parallel optimization step, most nonsupport vectors are quickly removed so that the training time for sequential optimization can be reduced dramatically. In addition, some effective strategies, such as kernel caching and efficient computation of kernel matrix, are integrated to speed up the training process. Further, the space and runtime complexity of the proposed algorithm are analyzed and we show that its runtime complexity linearly scales with the number of classes and the size of the data set.

Extensive experiments have been conducted to study various appealing properties of the proposed algorithm. Compared with Libsvm and SVM^{light}, the proposed algorithm has a much higher training speed, especially on databases of a huge size with thousands of classes. In addition, we tested the generalization performance of feature-based SVM on several handwritten character databases trained by the proposed algorithm. Good generalization performances have also been achieved on these databases.

Acknowledgments

The authors are grateful for support from NSERC and FCAR. They would like to thank the anonymous reviewers for their constructive comments and suggestions. In addition, they are indebted to Dr. Changping Liu of the Hanwang company for providing them with Hanwang databases for research, Dr. S.S. Keerthi for his comments on a draft of this paper and to Beverley Abramovitz for her help in proofreading this draft.

References

[1] V.N. Vapnik, *Statistical Learning Theory.* New York: Wiley, 1998.
[2] N. Cristianini and J.S. Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge, U.K.: Cambridge Univ. Press, 2000.
[3] B. Schölkopf and A.J Smola, *Learning with Kernels.* Cambridge, Mass.: MIT Press, 2002.
[4] B. Schölkopf, C.J.C. Burges, and V.N. Vapnik, "Extracting Support Data for a Given Task," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining,* pp. 252-257, 1995.
[5] D. DeCoste and B. Schölkopf, "Training Invariant Support Vector Machines," *Machine Learning,* vol. 46, nos. 1-3, pp. 161-190, 2002.
[6] J.X. Dong, C.Y. Suen, and A. Krzyżak, "A Fast SVM Training Algorithm," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 17, no. 3, pp. 367-384, 2003.
[7] T. Joachims, "Text Categorization with Support Vector Machine: Learning with Many Relevant Features," *Proc. ECML-98, 10th European Conf. Machine Learning,* pp. 137-142, 1998.
[8] E. Osuna, R. Freund, and F. Girosi, "Training Support Vector Machines: An Application to Face Detection," *Proc. IEEE. Conf. Computer Vision and Pattern Recognition,* pp. 130-136, 1997.
[9] C.M. Bishop, *Neural Networks for Pattern Recognition.* Oxford, U.K.: Clarendon Press, 1995.
[10] J. Verbeek, N. Vlassis, and B. Kröse, "Efficient Greedy Learning of Gaussian Mixture Models," *Neural Computation,* vol. 15, no. 2, pp. 469-485, Feb. 2003.
[11] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Royal Statistical Soc., Series B,* vol. 39, no. 1, pp. 1-38, 1977.
[12] J.C. Platt, "Fast Training of Support Vector Machines Using Sequential Minimal Optimization," *Advances in Kernel Methods: Support Vector Machines,* B. Schölkopf, C.J.C. Burges, and A. Smola, eds., pp. 185-208, Cambridge, Mass.: MIT Press, Dec. 1998.
[13] S.S. Keerthi, S.K. Shevade, C. Bhattachayya, and K.R.K. Murth, "Improvements to Platt's SMO Algorithm for SVM Classifier Design," *Neural Computation,* vol. 13, pp. 637-649, Mar. 2001.
[14] T. Joachims, "Making Large-Scale Support Vector Machine Learning Practical," *Advances in Kernel Methods: Support Vector Machines,* B. Schölkopf, C.J.C. Burges, and A. Smola, eds., pp. 169-184, Cambridge, Mass.: MIT Press, Dec. 1998.
[15] H. Kuhn and A. Tucker, "Nonlinear Programming," *Proc. Second Berkeley Symp. Math. Statistics and Probabilistics,* pp. 481-492, 1951.
[16] C.C. Chang and C.J. Lin, "Libsvm: A Library for Support Vector Machines," technical report, Dept. of Computer Science and Information Eng., Nat'l Taiwan Univ., 2003.
[17] G.W. Flake and S. Lawrence, "Efficient SVM Regression Training with SMO," *Machine Learning,* vol. 46, nos. 1-3, pp. 271-290 Mar. 2002.
[18] R. Collobert, S. Bengio, and Y. Bengio, "A Parallel Mixture of SVMs for Very Large Scale Problems," *Neural Computation,* vol. 14, no. 5, pp. 1105-1114, 2002.
[19] R. Collobert and S. Bengio, "SVMTorch: Support Vector Machines for Large-Scale Regression Problems," *J. Machine Learning Research,* vol. 1, pp. 143-160, 2001.
[20] A. Rida, A. Labbi, and C. Pellegrini, "Local Experts Combination through Density Decomposition," *Proc. Seventh Int'l Workshop AI and Statistics,* D. Heckerman and J. Whittaker, eds., Jan. 1999.
[21] V. Tresp, "A Bayesian Committee Machine," *Neural Computation,* vol. 12, no. 11, pp. 2719-2741, 2000.
[22] A. Schwaighofer and V. Tresp, "The Bayesian Committee Support Vector Machine," *Proc. Int'l Conf. Artificial Neural Networks,* pp. 411–417, 2001.
[23] D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach,* second ed. San Fransisco: Morgan Kaufmann, 1996.
[24] J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Software,* vol. 16, pp. 1-17, 1990.
[25] R.C. Whaley, A. Petitet, and J.J. Dongarra, "Automated Empirical Optimization of Software and the ATLAS Project," technical report, Dept. of Computer Science, Univ. of Tennessee, 2000.
[26] B.E. Boser, I.M. Guyon, and V.N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," *Proc. Fifth Ann. ACM Workshop Computational Learning Theory,* D. Haussler, ed., pp. 144-152, 1992.
[27] S.S. Keerthi and E.G. Gilbert, "Convergence of a Generalized SMO Algorithm for SVM Classifier Design," *Machine Learning,* vol. 46, no. 3, pp. 351-360, Mar. 2002.
[28] Intel Corporation, "The IA-32 Intel Architecture Software Developer's Manual," vol. 1: basic architecture, order number: 245470, 2002.
[29] J.E.R. Staddon, *Adaptive Behavior and Learning.* Cambridge, U.K.: Cambridge Univ. Press, 1983.
[30] B.V. Gnedenko, Y.K. Belyayev, and A.D. Solovyev, *Mathematical Methods of Reliability Theory.* New York: Academic Press, 1969.
[31] A. Ben-Hur, H.T. Siegelmann, and S. Fishman, "Complexity for Continuous Time Systems," *J. Complexity,* vol. 18, no. 1, pp. 51-86, 2002.
[32] Y. LeCun, L.D. Jackel, L. Bottou, J.S. Denker, H. Drucker, I. Guyon, U.A. Müller, E. Sackinger, P. Simard, and V.N. Vapnik, "Comparison of Learning Algorithms for Handwritten Digit Recognition," *Proc. Int'l Conf. Artificial Neural Network,* F. Fogelman and P. Gallinari, eds., pp. 53-60 1995.
[33] K. Fukunaga, *Introduction to Statistical Pattern Recognition,* second ed. New York: Academic Press, 1990.

[34] J.X. Dong, A. Krzyżak, and C.Y. Suen, "High Accuracy Handwritten Chinese Character Recognition Using Support Vector Machine," *Proc. Int'l Workshop Artificial Neural Networks on Pattern Recognition,* 2003.
[35] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification,* second ed. New York: Wiley, 2001.
[36] R.C. Whaley and J.J. Dongarra, "Automatically Tuned Linear Algebra Software (ATLAS)," *Proc. High Performance Networking and Computing,* 1998.
[37] Intel Corporation, "Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual," order number: 248966, 2002.
[38] L. Breiman, "Bias, Variance, and Arcing Classifiers," Technical Report 460, Statistics Dept., Univ. of California, Berkeley, Apr. 1996.
[39] R. Collobert, Y. Bengio, and S. Bengio, "Scaling Large Learning Problems with Hard Parallel Mixtures," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 17, no. 3, pp. 349-365, 2003.
[40] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 24, no. 1, pp. 49-64, 1996.
[41] K. Fukunaga and T.F. Krile, "Calculation of Bayes' Recognition Error for Two Multivariate Gaussian Distributions," *IEEE Trans. Computers,* vol. 18, no. 3, pp. 220–229, Mar. 1969.
[42] J.A. Blackard and D.J. Dean, "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables," *Computers and Electronics in Agriculture,* vol. 24, pp. 131-151, 1999.

Jian-xiong Dong received the master's degree in computer science from the Institute of Automation, Chinese Academy of Sciences in 1999 and the PhD degree at CENPARMI, Concordia University, Canada, in 2003. Currently, he is doing research on large-scale cursive word recognition, supported by industrial research fellowship of Natural Sciences and Engineering Research Council of Canada. His research interests include pattern recognition, machine learning, and handwritten character recognition.

Adam Krzyżak received the MSc and PhD degrees in computer engineering from the Technical University of Wroclaw, Poland, in 1977 and 1980, respectively, and the DSc degree (habilitation) in computer engineering from the Warsaw University of Technology, Poland, in 1998. In December 2003, he received the title of professor from the President of the Republic of Poland. Since 1983, he has been with the Department of Computer Science, Concordia University, Montréal, Canada, where he is currently a professor. In 1983, he held the International Scientific Exchange Award in the School of Computer Science, McGill University, Montréal, Canada, in 1991, the Vineberg Memorial Fellowship at Technion-Israel Institute of Technology, and, in 1992, the Humboldt Research Fellowship at the University of Erlangen-Nürnberg, Germany. He visited the University of California at Irvine, the Information Systems Laboratory at Stanford University, the Riken Frontiers Research Laboratory, Japan, Stuttgart University, Technical University of Berlin, and the Technical University of Szczecin, Poland. He has published more than 200 papers on neural networks, pattern recognition, image processing, computer vision, control, and nonparametric estimation. He has been associate editor of the *IEEE Transactions on Neural Networks* and is presently on the editorial board of the *Pattern Recognition Journal* and the *Journal of Neural, Parallel and Scientific Computations*. He was coeditor of the book *Computer Vision and Pattern Recognition* (Singapore: World Scientific, 1989) and is coauthor of the book *A Distribution-Free Theory of Nonparametric Regression* (Springer-Verlag, 2002). He has served on the program committees of the Vision Interface Conference, the International Conference on Document Processing and Applications, the International Conference on Computer Vision, Pattern Recognition, and Image Processing, and the World Congress of Nonlinear Analysts. He is a senior member of the IEEE.

Ching Y. Suen received the MSc (Eng.) degree from the University of Hong Kong and the PhD degree from the University of British Columbia, Canada. In 1972, he joined the Department of Computer Science of Concordia University where he became Professor in 1979 and served as chairman from 1980 to 1984, and as associate dean for Research of the Faculty of Engineering and Computer Science from 1993 to 1997. He has guided/hosted 65 visiting scientists and professors, and supervised 65 doctoral and master's graduates. Currently, he holds the distinguished Concordia Research Chair in artificial intelligence and pattern recognition, and is the director of CENPARMI, the Centre for PR & MI. Dr. Suen is the author/editor of 11 books and more than 400 papers on subjects ranging from computer vision and handwriting recognition, to expert systems and computational linguistics. He is the founder of *The International Journal of Computer Processing of Oriental Languages* and served as its first editor-in-chief for 10 years. Presently, he is an associate editor of several journals related to pattern recognition. He is a fellow of the IEEE, the IAPR, and the Academy of Sciences of the Royal Society of Canada. He has given 150 seminars at major computer industries and various government and academic institutions. He has been the principal investigator of 25 industrial/government research contracts, and is the recipient of several prestigious awards.