A Geometric Approach to Train Support Vector Machines

Ming-Hsuan Yang and Narendra Ahuja
Department of Computer Science and Beckman Institute
University of Illinois at Urbana-Champaign, Urbana, IL 61801
{mhyang,ahuja}@vision.ai.uiuc.edu
http://vision.ai.uiuc.edu

Abstract

Support Vector Machines (SVMs) have shown great potential in numerous visual learning and pattern recognition problems. The optimal decision surface of a SVM is constructed from its support vectors which are conventionally determined by solving a quadratic programming (QP) problem. However, solving a large optimization problem is challenging since it is computationally intensive and the memory requirement grows with square of the training vectors. In this paper, we propose a geometric method to extract a small superset of support vectors, which we call guard vectors, to construct the optimal decision surface. Specifically, the guard vectors are found by solving a set of linear programming problems. Experimental results on synthetic and real data sets show that the proposed method is more efficient than conventional methods using QPs and requires much less memory.

1 Introduction

The Support Vector Machine (SVM) is a novel machine learning algorithm based on statistical learning theory that can be applied to pattern recognition and regression problems [15] [3]. One distinct characteristic of SVMs is that it aims to find the optimal hyperplane from a set training samples such that the expected recognition error for the unseen test samples is minimized. According to the structural risk minimization inductive principle, a function that describes the training data well and belongs to a set of functions with lowest VC (Vapnik-Chervonenkis) dimension will generalize well regardless of the dimensionality of the input space [3]. Based on this principle, SVM adopts a systematic approach to find a linear function that belongs to a set of functions with lowest VC dimension. The SVM algorithm also provides non-linear function approximations by projecting the input vectors to a high dimensional feature space in which a linear hyperplane is constructed to separate all the projected vectors. One novelty of the nonlinear SVMs is the use of kernel functions to avoid the expensive computations imposed by the nonlinear projection. Although there is no guarantee that the patterns in a high dimensional space can always be linearly separated by a hyperplane, in practice it is usually feasible to find one such linear hyperplane in the projected space.

SVMs have recently attracted much attention because of the rigorous theoretical derivations from statistical learning theory and excellent empirical results in many classification tasks. Training a SVM is equivalent to solving a quadratic optimization problem in which the support vectors (SVs) are identified to construct the optimal hyperplane. However, many researchers have found that SVMs are not only computationally expensive but also memory intensive.

Consider a training set of m examples, $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, $\ldots$, $(\mathbf{x}_m, y_m)$, where $\mathbf{x}_i$ is an input vector and y_i is its class label, one can solve the following quadratic programming (QP) problem to find the optimal hyperplane:

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i Q_{ij}\alpha_j - \sum_{i=1}^{m}\alpha_i \\ \text{Subject to} & 0 \le \alpha_i \le C \quad \text{and} \quad \sum_{i=1}^{m} y_i\alpha_i = 0 \end{array}$$

where C is a parameter for soft margin classifier and Q is an $m \times m$ matrix that depends on the training inputs $\mathbf{x}_i$, the label y_i, and the kernel function of a SVM. Note that a training set of 50,000 examples will yield a Q matrix with 2.5 billion elements, which cannot easily fit into the memory of a standard computer. Solving a large QP problem is usually computationally and memory intensive, nontrivial to implement, and can suffer from numerical instability.

Although several methods have been developed to efficiently solve the quadratic optimization problem [10] [11] [7], none use the *structure* information of the training vectors. Since the support vectors (SVs) form a subset of training vectors which have equal minimum perpendicular distance to the optimal hyperplane and the optimal hyperplane is a weighted linear combination of these SVs, we can construct *exactly* the same optimal hyperplane if we are given only the SVs.

In this paper, we propose a method to extract a superset of the SVs based on the structural information of the training vectors. Note that for each support vector $\mathbf{x}_s$ with label y_s, we can always find a hyperplane that passes through $\mathbf{x}_s$ and separates the vectors with label y_s and those with the opposite class label. In other words, a training vector may be a support vector if there exists such a hyperplane that linearly separates all the vectors according to their labels.

The duality theory in computational geometry specifies that a point or a vector in any dimensional space has a unique corresponding hyperplane in the dual space [12], and vice versa. We are interested in determining whether there exits a hyperplane through a vector $\mathbf{x}_i$ in the primal space such that all the points are linearly separated according to their labels in the primal space. Since all the points in the primal space have dual hyperplanes in the dual space and their normal directions are determined by their labels, the existence of a separating hyperplane at $\mathbf{x}_i$ in the primal space is equivalent to feasibility of a linear program in the dual space. If there exists such a hyperplane, there exists a corresponding point in the dual space. Since the hyperplane through $\mathbf{x}_i$ can linearly separate all the points, $\mathbf{x}_i$ may be a support vector. We will call such vectors the *guard vectors* in the rest of this paper. For an optimal hyperplane with unit normal $\mathbf{w}$, the support vectors must lie on hyperplanes with normalized distance $\frac{1}{||\mathbf{w}||}$ to the optimal hyperplane. Therefore, every support vector must be a guard vector. However, a guard vector may not be a support vector since it may not have minimum distance to the optimal hyperplane.

For a set of m training examples, the superset of SVs are found by solving a set of m linear programming problems. Each LP aims at determining whether a point can be a support vector on the optimal hyperplane. Note that we are interested in the feasibility of each LP rather than the optimal objective value of that LP. After we extract the set of guard vectors, we can construct the same optimal hyperplane by solving a quadratic programming problem with these vectors. Since the guard vectors usually form a small superset of support vectors and linear programming problems can be solved more efficiently than quadratic programming problems, the proposed method is an efficient algorithm to train SVMs. Our experimental results on several benchmarks show that the proposed method has low time and space complexity.

This paper is organized as follows. Section 2 describes previous approaches to train SVMs. In Section 3, we present a method to extract guard vectors using linear programming and then construct an optimal hyperplane. In Section 4, we present experimental results on several synthetic datasets and real data sets. We conclude this paper with comments on current and future work in Section 5.

2 Related Work

For large learning problems with many training examples, the constrained quadratic optimization approach to solving SVMs quickly becomes intractable in terms of time and memory requirements. A training set of 50,000 examples will yield a Q matrix with 2.5 billion elements, which cannot easily fit into the memory of a standard computer. Consequently, traditional optimization algorithm such as Newton, Quasi Newton, etc., cannot be directly applied (since these methods usually involve the whole Hessian matrix of Q). Several researchers have proposed decomposition methods to solve this optimization problem [2] [15] [10] [8] [6] [7] [11]. Vapnik et al. describe a "chunking" method [2] using the fact that the solution of a QP problem is the same if we remove the rows and columns of the matrix Q that correspond to zero Lagrange multipliers. Thus, a large QP problem can be decomposed into a series of smaller QP subproblems in which all of the nonzero Lagrange multipliers are identified and all the zero Lagrange multipliers are discarded. After all the nonzero Lagrange multipliers in Q have been identified, the last step then solves the remaining QP problem. Osuna et al. propose a novel decomposition algorithm for solving the SVM QP problem [10]. As a variation of the active set methods, a large QP problem is decomposed into a series of subproblems by maintaining a small working set. The algorithm works by moving the samples that violate the Karush-Kühn-Tucker condition to the working set in each iteration, and solve the subproblems. Since the working set is usually small, this method does not have memory problems. However, a numerical QP solver is required. Joachims improves Osuna's methods with a strategy to select good working sets [6]. Platt develops Sequential Minimal Optimization (SMO) which further improves Osuna's method by making all the working sets of size 2 with a set of heuristics [11]. One feature of SMO is that all the subproblems are solved analytically.

An analogous problem to SVM has been investigated in the Statistical Mechanics literature, which has resulted in several perceptron-alike algorithms. These algorithms, e.g., Adatron [1], aim to find maximal margin hyperplanes in the input space. By introducing kernels into Adatron, Friess et al. propose

Kernel-Adatron which is able to maximize the margin in the feature space [5]. Ruján describes a stochastic approximation method to to estimate the optimal Bayes classifier for linearly separable problems [13] and later extends to support vector machines using kernels [14]. Using a projection method different fromwhat has been used in conventional SVMs, Freund and Shcapire develop an algorithm to find maximal margin perceptron in a high dimensional feature space [4]. Compared to SVMs, this method is much simpler to implement (i.e., no QP optimization) and more efficient in terms of computational complexity. However the performance is close to, but not as good as, the performance of SVM on the same problems.

Recently Keerthi et al. treat SVM classification in terms of finding the nearest points between pairs of convex polytopes in the feature space, and solve them with an iterative nearest point algorithm. Although this approach does not require numerical QP library, one drawback is that it needs to consider $\frac{m(m-1)}{2}$ pairs of points for a problem with m points.

3 Extracting Guard Vectors for Support Vector Machines

In this section we first describe a method to extract the guard vectors by solving a set of linear programming problems. These guard vectors are then used to construct hyperplane for linear SVMs. For nonlinear SVMs, a projection method is described to project vectors onto a higher dimensional space in which a linear hyperplane is constructed.

3.1 Determining Guard Vectors Using Linear Programming

A point in a plane has two parameters: its x-coordinate and its y-coordinate. A line in a plane has also two parameters: its slope and its intersection with the y-axis. The duality transform in computational geometry specifies that every point in a plane has a unique dual hyperplane representation [12]. In other words, there exists a one-to-one mapping between points and lines such that certain properties of points translate to certain other properties for the set of lines. For instance, the mapping is incidence and order preserving. The dual of a point $p = (p_x, p_y)$ in a plane is a line defined by $y = p_x x - p_y$. Figure 1 shows each point in the primal plane corresponds to a line in the dual plane. Also note that three points on a line becomes three lines through a point.

For each point p_i, we can use its class label to determine whether there exists a line l_s through p_i in the primal plane such that all the points of the same label are on the same side of l_s. If all the points of the same

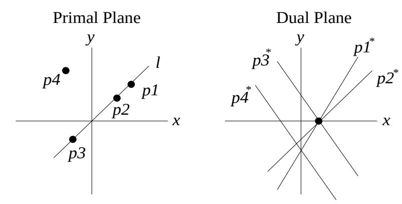


Figure 1: Dual representation of a point. Each point in the primal plane corresponds to a line in the dual plane.

label fall on the same side of l_s, it means l_s can linearly separate all the points. We call such point p_i a guard vector. Since each point on a plane corresponds to a line, we can find all the dual line representations for all the points. Therefore, the problem of determining the existence of l_s is equivalent to solving a linear program in the dual space where the constraints are the dual lines of the points in the primal plane. If there exists a feasible point to the LP in the dual plane, it means there exists a line in the primal plane that linearly separates all the points.

Figure 2 shows three points with their duals (i.e., lines) in a plane in which points p_1 and p_2 belong to class '+' while point p_3 belongs class '-'. Consider point p_1 first, Figure 2 (a) shows one LP that we use the dual of point 1 ,i.e., line l_1, and specifies that all the points belonging to class '+' should be on the right hand side of or on l_1 and the points belonging to class '-' should be on the left hand side of l_1. The corresponding LP for point p_1 is

$$\begin{aligned} p_{11}x_1 - x_2 &= p_{12} \quad (l_1) \\ p_{21}x_1 - x_2 &\geq p_{22} \quad (l_2) \\ p_{31}x_1 - x_2 &\leq p_{32} \quad (l_3) \end{aligned}$$

Note the equality and inequality constraints are set according to the class labels of the point considered in this LP and the other points. This means that any solution $p^* = (x_1, x_2)$ must be on l_1 and p^* must satisfy the constraints in l_2 and l_3. In other words, if there exists a solution for this LP, l_2 should be on the right hand side of l_1 and l_3 should be on the left hand side of l_1. Clearly, there is no feasible solution to satisfy all the constraints since p^* does not satisfy the constraint in l_3. Figure 2 (b) shows another possible formulation for l_1. The constraint of l_2 cannot be satisfied and thus there is no feasible solution for this formulation. Since there is no feasible solution for either formulation of l_1, p_1 is not a guard vector. Similarly for p_2, Figure 2 (c) shows one case that the one LP that has no feasible solution. However, Figure 2 (d) shows one LP of p_2 that has feasible solution because l_1 is on the right left hand side of l_2 and l_3 is on the right hand side of l_2. In fact, the shaded area in Figure 2 shows the all the feasible solutions for this LP. Consequently, p_2 is a guard vector. This example illustrates the use of LP in which a feasible solution means the intersection of the half planes specified by the constraints is nonempty. It also shows that we are only interested in the feasibility of a LP rather than its optimal value.

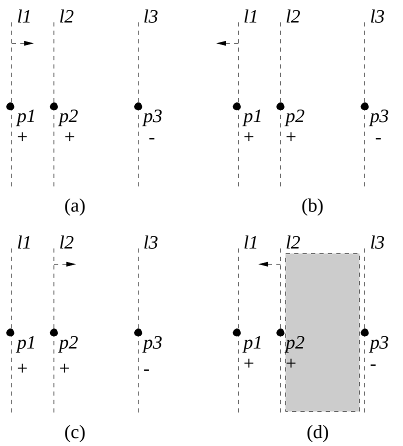


Figure 2: Using linear programming to extract guard vectors. Each point and its dual are shown in the figure. Points p_1 and p_2 belong to class '+' while point p_3 belongs to class '-'.

Formally, for a set of m points with q positive examples and $m-q$ negative examples, we can formulate a LP problem for point p_1:

$$
\begin{array}{llcccc}
\text{Maximize } x_1 & & & & & \\
\text{Subject to} & p_{11}x_1 & - & x_2 & = & p_{12} \\
 & p_{21}x_1 & - & x_2 & \geq & p_{22} \\
 & & & & \cdots & \\
 & p_{q1}x_1 & - & x_2 & \geq & p_{q2} \\
 & p_{(q+1)1}x_1 & - & x_2 & \leq & p_{(q+1)2} \\
 & & & & \cdots & \\
 & p_{m1}x_1 & - & x_2 & \leq & p_{m2}
\end{array}
$$

where, without loss of generality, we assume that p_i, $1 \leq i \leq q$ are positive examples and the others are negative examples. The constraints in the LP specify the correct class labels for each point and the LP problem itself means that we want to find a feasible solution to satisfy all the constraints. Figure 3 gives a geometric interpretation of an LP problem. If there is a feasible solution in the dual plane, it means that we can find a hyperplane through p_1 in the primal plane such that all the points of the same class fall on the same side of the hyperplane (i.e., have the same class label). In other words, the intersection of the half planes specified by all the constraints is not empty as shown in Figure 3 (a). Since we do not have prior knowledge about whether the points of the same class are above or below the hyperplane, we need to change the inequality sign to check the other possibility. A point is a guard vector if there exists one feasible solution to the corresponding LP. If there is no feasible solution to the LP, it means that there is no existing hyperplane such that all the points are separated correctly. This means the intersection of all the half planes specified by the constraints is empty as shown in Figure 3 (b). Note that the exact form of the objective function is immaterial since we are interested in whether there is a solution to the constraints or not, which means the optimization problem can be solved more efficiently if we choose a simple form.

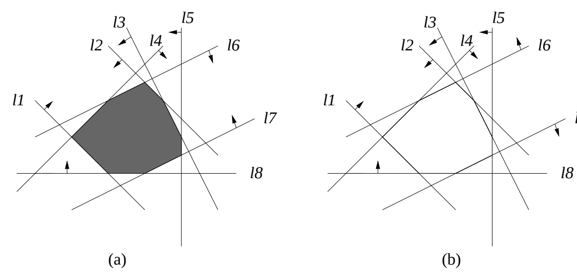


Figure 3: Geometric interpretation of a linear programming problem. (a) shows one case where a feasible solution exists. (b) shows one case where no feasible solution exists.

Figure 4 shows a set of 50 points belonging to two classes: the points labeled with '+' belong to class 1 and the others labeled with '*' belong to class 2. The guard vectors are labeled with circles and the optimal separating hyperplane found by a linear SVM is the dash-dot line. For point p_{17}, the corresponding LP does not have a feasible solution and therefore it is not a guard vector. In other words, it means that there exists on line such that all the points can be correctly separated. Similarly point p_{45} is not a guard vector. For each guard vector, there exists one line such that all the points can separated correctly. A close study shows that guard vector 1 and 4 are support vectors for the optimal separating hyperplane found by a linear SVM. Also note that guard vectors form a superset of support vectors.

Duality applies to high dimensional point sets as well. For a point in n-dimensional space, $p_i = (p_{i1}, p_{i2}, \ldots, p_{in})$, its dual p^* is the hyperplane $x_d = p_{i1}x_1 + p_{i2}x_2 + \ldots + p_{in-1}x_{n-1} - p_{in}$. Similar to the 2D case, we can use solve a linear programming problem to

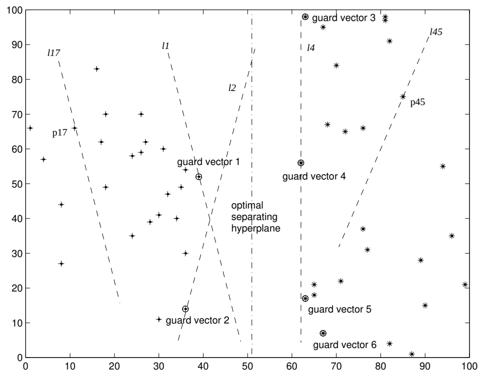


Figure 4: Geometric interpretations of guard vectors and support vectors (guard vectors form a superset of support vectors): guard vector 1 and 4 are the support vectors for the optimal separating hyperplane found by the SVM for this set of points.

determine whether a point is a guard vector or not:

$$Ax \leq b$$

In matrix form, we have

$$\begin{bmatrix} -p_{11} & -p_{12} & \cdots & +1 \\ -p_{21} & -p_{22} & \cdots & +1 \\ \vdots & \vdots & \ddots & \\ p_{m1} & p_{m2} & \cdots & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} -p_{1n} \\ -p_{2n} \\ \vdots \\ p_{mn} \end{bmatrix}$$

where m is the number of points and the sign of each element depends on its class label and the class label of the point that the LP corresponds to.

We summarize the abovementioned algorithm as follows:

Algorithm: Finding the guard vectors

1. Transform each point to its dual hyperplane representation with inequality '$\geq$'. Initialize i to 1.

2. Consider point p_i with class label l_i, first set the equality sign for dual hyperplane for point p_i. Then set the inequality signs of all dual hyperplanes of the points having the same class label to '$\geq$', and set the inequality signs of all the other hyperplanes of the points having the opposite class label to '$\leq$'.

3. Solve the current LP problem. If there is a feasible solution then it is a guard vector, else reverse all the inequality signs. If there is still no feasible solution to the new LP, then point p_i is not a guard vector

4. Increase i by 1 and go to step 2 until all the points have been checked.

3.2 Linear SVM

Given a set of samples $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, $\ldots$, $(\mathbf{x}_l, y_m)$ where $\mathbf{x}_i$ ($\mathbf{x}_i \in R^n$) is the input vector of n dimension and y_i is its label ($y_i \in \{-1, 1\}$) for a recognition problem, SVM aims to find the optimal hyperplane that leaves the largest possible fraction of data points of the same class on the same side while maximizing the distance of either class from the hyperplane (margin). Vapnik [15] shows that maximizing the margin distance is equivalent to minimizing the VC dimension in constructing an optimal hyperplane. The optimal hyperplane is in the form

$$f(\mathbf{x}) = \sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{x} + b = \sum_{i=1}^{m} y_i \alpha_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b$$

where $k(\mathbf{x}, \mathbf{x}_i) = \mathbf{x} \cdot \mathbf{x}_i$ for linear SVMs. The sign of $f(\mathbf{x})$ determines the class label of $\mathbf{x}$.

Since the optimal hyperplane can be constructed *exactly* using only the support vectors, and the guard vectors form a superset of SVs. We can use the proposed algorithm to extract all the guard vectors and then find the support vectors and corresponding nonzero α_i by solving a small QP to construct the optimal hyperplane.

3.3 Nonlinear SVM

For the patterns that cannot be lineally separated, we use a nonlinear function similar to [4] to project training vectors from input space to a high dimensional space in which we construct a linear hyperplane. For better generalization performance, we may also want to project training vectors from input space to a higher dimensional space. One distinct feature of this projection function is that the projected vectors are guaranteed to be linearly separated in the projected space.

Let $\mathbf{X}$ be a Hilbert space and for any fixed nonnegative Δ, we define a projection of $\mathbf{X}$ onto another higher dimensional Hilbert space $\mathbf{X}'$ as

$$\tau_\Delta : \mathbf{x}_i \in \mathbf{X} \mapsto \mathbf{x}_i' = (\mathbf{x}_i, \Delta\delta_{\mathbf{x}_i}) \in \mathbf{X}'$$

where $\delta_{\mathbf{x}_i} \in R^m$ is defined by

$$\delta_{\mathbf{x}_i}(j) = \begin{cases} 1; & \text{if } j = i; 1 \leq j \leq m \\ 0; & \text{otherwise.} \end{cases}$$

Given a set of samples $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, $\ldots$, $(\mathbf{x}_m, y_m)$ where $\mathbf{x}_i$ ($\mathbf{x}_i \in R^n$) is the input vector of n dimension and y_i is its label ($y_i \in \{-1, 1\}$), the projection extends the input space R^n to R^{n+m} by adding m new dimensions, one for each example. Let $\mathbf{x}_i' \in R^{n+m}$ denote the extension of the point $\mathbf{x}_i$, we set the first n coordinates of $\mathbf{x}_i^{'}$ equal to $\mathbf{x}_i$ and the $(n+i)$-th coordinate to Δ. The rest of the coordinates of $\mathbf{x}_i^{'}$ are set to zero.

For a linear classifier $\mathbf{w}$ on $\mathbf{X}$ and threshold $b \in R^n$, we define deviation d_i for each example as

$$d_i((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma) = \max\{0, \gamma - y_i((\mathbf{w} \cdot \mathbf{x}_i) - b)\}$$

where d_i is the amount by which $\mathbf{w}$ fails to reach the margin at point $(\mathbf{x}_i, y_i)$ or 0 if its margin is larger than γ, i.e.

$$d_i((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma) \geq \gamma - y_i((\mathbf{w} \cdot \mathbf{x}_i) - b)$$

Only those points whose margins are less than γ have $d_i > 0$ cause *errors* in classification.

Similar to the projection for $\mathbf{x}_i$, we project the normal vector $\mathbf{w}$ of a hyperplane $\in R^n$ to $\mathbf{w}^{'} \in R^{n+m}$ by the following function:

$$\mathbf{w}^{'} = (\mathbf{w}, \frac{1}{\Delta} \sum_{i=1}^{m} d((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma) y_i \delta_{\mathbf{x}_i})$$

Claim: All the points $\mathbf{x}^{'}$ in the projected space $\mathbf{X}^{'}$ are linearly separable.

Proof: For a point $(\mathbf{x}_i, y_i)$ and a hyperplane $(\mathbf{w}, b)$, we have the following in the projected space $X^{'}$,

$$\begin{aligned}
&y_i^{'}((\mathbf{w}^{'} \cdot \mathbf{x}_i^{'}) - b) \\
&= y_i^{'}((\mathbf{w}, \mathbf{x}^{'}) - b) + y_i^{'}(\textstyle\sum_{i=1}^{m} d((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma) y_i \delta_{\mathbf{x}_i} \cdot \delta_{\mathbf{x}_i^{'}}) \\
&\geq \gamma - d_i((\mathbf{x}_i^{'}, y_i^{'}), \mathbf{w}, \gamma) + d((\mathbf{x}_i^{'}, y_i^{'}, \mathbf{w}, \gamma) = \gamma
\end{aligned}$$

In other words, all the points in Hilbert space $\mathbf{X}^{'}$ have margins that are large than γ. Therefore, all the points can be linearly separated.

4 Experiments

In this section, the proposed algorithm is tested against SVMs with QP numerical library on synthetic and benchmark data. All implementations are written in MATLAB and the CPU times of all the experiments are measured on an unloaded Sun Ultra 10. The objective of these experiments is to benchmark the CPU time and memory usage of the proposed method and the conventional QP approach in SVMs, rather than the recognition performance. In other words, we do not tune the parameters in SVMs to get the optimal performance (e.g., the parameter C is set to 1000 for all experiments). However, both methods do produce the same optimal separating hyperplane in each experiment.

4.1 Synthetic Data

Each synthetic data set is randomly generated from a n-dimensional hypercube. For each data set, $m/2$ positive examples are generated from a hypercube of $[0, 40]^n$ and $m/2$ negative examples are generated from a hypercube of $[60, 100]^n$. Figure 5 shows the CPU time and memory requirements on a series of synthetic data setsin 2 dimensions. The experimental results show that the proposed method outperforms the QP implementation for SVM since it requires much less CPU time and memory. A close examination also shows that the number of guard vectors found by the proposed method is at most twice the number of support vectors.

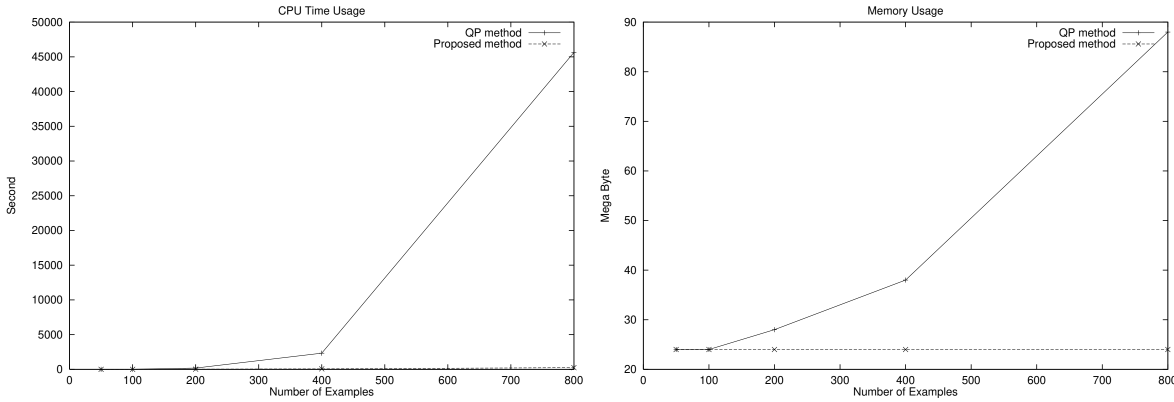
Table 1 shows more experimental results in which points are generated from hypercubes of higher dimensions. Note the reported CPU time of the proposed method in each entry includes the time to extract guard vectors and the time to train a SVM using the conventional QP method. The results show that the proposed algorithm performs well when the ratio between the number of points and the dimensionality (i.e., m/n) is high. Meanwhile, the ratio between the number guard vectors and the number of support vectors depends both on the ratios of m/n and the structure of the data set itself.

4.2 Real-World Data

The first data set we use is from the Wisconsin Breast Cancer data set [9]. Each of the 699 data elements has 9 attributes, with class label indicating whether it represents a benign or malignant example. The second data set consists of 500 20×20 face images and 1000 20×20 nonface face images, to be used for face detection testssimilar tothose reported in [10]. Both data sets are trained using a linear SVM for benchmarking. Figure 6 shows some of the face images in the data set.

Figure 6: Some 20×20 face images used in the experiment. Each image is converted to a 400-dimensional vector.

Table 2 shows the CPU time and memory usage of



(a) (b)

Figure 5: Benchmark results of the proposed algorithm against a linear SVM in terms of CPU time and memory requirements using the first 2-dimensional data set shown in Table 1.

Table 1: Experimental results on synthetic data: Each data set has m points that are randomly scattered in an n-dimensional hypercube.

Dimension (n): 2								
	Proposed method			QP method			Ratio	
# pt (m)	cpu (sec)	mem (MB)	# gv (g)	cpu (sec)	mem (MB)	# sv(s)	g/s	m/n
50	1.96	24	3	1.63	24	2	1.5	25.0
100	5.61	24	4	8.50	24	3	1.3	50.0
200	18.4	24	4	187.2	28	3	1.3	100.0
400	64.57	24	8	2326.69	38	5	1.6	200.0
800	243.19	24	11	45633.23	88	6	1.9	400.0
Dimension (n): 3								
	Proposed method			QP method			Ratio	
# pt (m)	cpu (sec)	mem (MB)	# gv (g)	cpu (sec)	mem (MB)	# sv(s)	g/s	m/n
50	2.23	24	16	1.91	24	3	5.3	16.6
100	6.31	24	29	11.9	25	6	4.8	33.3
200	20.73	24	35	307.85	28	14	2.5	66.6
400	70.64	24	46	3717.81	39	23	2.0	133.3
Dimension (n): 5								
	Proposed method			QP method			Ratio	
# pt (m)	cpu (sec)	mem (MB)	# gv (g)	cpu (sec)	mem (MB)	# sv(s)	g/s	m/n
50	2.6	24	40	1.84	24	2	12.0	10.0
100	7.2	24	61	19.18	24	4	15.3	20.0
200	23.41	24	130	178.51	28	12	10.8	40.0
400	78.10	24	181	1885.75	45	10	18.1	80.0

Table 2: Experimental results on benchmark data sets: The first one is the Wisconsin Breast Cancer data set in which each of the 699 points has 9 features. The second data set consists of 500 20×20 face images and 1000 20×20 nonface images for face detection.

	Proposed method			QP method			Ratio	
# pt (m)	cpu (sec)	mem (MB)	# gv (g)	cpu (sec)	mem (MB)	# sv(s)	g/s	m/n
699	99.02	28	317	3572.91	68	52	6.1	77.7
1500	39086.21	38	712	36506400.81	168	347	4.2	3.8

both methods. For both cases, the proposed algorithm is more than 30 times faster than the conventional QP approach. Furthermore, the memory requirement of the proposed method is one fourth that of the QP method in a large scale experiment (i.e. the data set for face detection). The results also show that the set of guard vectors is a small superset (i.e., roughly 20 times) of support vectors. Note that both the proposed and conventional methods generate the same optimal hyperplane in all the experiments.

5 Discussion and Conclusion

The optimal decision surface of a SVM is constructed from its support vectors which are conventionally determined by solving a quadratic programming problem. However, solving a large optimization problem is challenging since it is computationally intensive and the memory requirement grows with the square of the number of training vectors. In this paper, we have proposed a geometric method to extract guard vectors, a small superset of support vectors, to construct the optimal decision surface. Specifically, the superset of support vectorsis found by solving a set of linear programming problems. Experimental results on synthetic and real data sets show that the proposed method is more efficient than conventional methods using QPs and requires much less memory.

Future work will focus on theoretical analysis of the number of guard vector versus the number of support vectors. Although the numbers of guard vectors and support vectors depend on how the data points scatter in the input and feature space, we believe the relationship between the number of guard and support vectors can be characterized with some assumptions such as general position of points. More experiments will also be conducted to benchmark the proposed algorithm against other training methods for SVMs.

Acknowledgements

The support of the Office of Naval Research under grant N00014-00-1-0091 is gratefully acknowledged.

References

[1] J. K. Anlauf and M. Biehl. The adatron: An adaptive perceptron algorithm. *Europhysics Letters*, 10(7):687–692, 1989.

[2] B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 1992.

[3] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20, 1995.

[4] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 1999. To appear.

[5] T. Friess, N. Cristianini, and C. Campbell. The kernel-adatron: a fast and simple learning procedure for support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 188–196, 1998.

[6] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, chapter 11, pages 169–184. MIT Press, 1998.

[7] L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*, chapter 10, pages 147–167. MIT Press, 1998.

[8] O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10(5):1032–1037, 1999.

[9] O. L. Mangasarian, R. Setiono, and W. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. In T. F. Coleman and Y. Li, editors, *Large-scale numerical optimization*, pages 22–30. SIAM Publications, 1990. data avilable at ftp://128.195.1.46/ pub/ machine- learning-databases/.

[10] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 130–136, 1997.

[11] 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*, chapter 12, pages 185–208. MIT Press, 1998.

[12] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[13] P. Ruján. Playing billiards in version space. *Neural Computation*, 9(1):99–122, 1997.

[14] P. Ruján. Computing the bayes support vector machine. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999. To appear.

[15] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.