Applied Intelligence 23, 219–239, 2005

An Efficient Support Vector Machine Learning Method with Second-Order Cone Programming for Large-Scale Problems

RAMESWAR DEBNATH
Department of Information and Communication Engineering, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585, Japan
rdebnath@ice.uec.ac.jp

MASAKAZU MURAMATSU
Department of Computer Science, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585, Japan

HARUHISA TAKAHASHI
Department of Information and Communication Engineering, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585, Japan

Abstract. In this paper we propose a new fast learning algorithm for the support vector machine (SVM). The proposed method is based on the technique of second-order cone programming. We reformulate the SVM's quadratic programming problem into the second-order cone programming problem. The proposed method needs to decompose the kernel matrix of SVM's optimization problem, and the decomposed matrix is used in the new optimization problem. Since the kernel matrix is positive semidefinite, the dimension of the decomposed matrix can be reduced by decomposition (factorization) methods. The performance of the proposed method depends on the dimension of the decomposed matrix. Experimental results show that the proposed method is much faster than the quadratic programming solver LOQO if the dimension of the decomposed matrix is small enough compared to that of the kernel matrix. The proposed method is also faster than the method proposed in (S. Fine and K. Scheinberg, 2001) for both low-rank and full-rank kernel matrices. The working set selection is an important issue in the SVM decomposition (chunking) method. We also modify Hsu and Lin's working set selection approach to deal with large working set. The proposed approach leads to faster convergence.

Keywords: second-order cone programming, quadratic programming, Cholesky factorization, eigenvalue decomposition, support vector machine

1. Introduction

Recently, the support vector machine (SVM) becomes an interesting research issue in the machine learning fields for its excellent generalization performance on a wide variety of real-world problems such as hand written character recognition, face detection, text categorization and object detection in machine vision, etc [2]. Although it has a good generalization ability, learning methods suffer a large computational effort for large optimization problem. Given training data $\boldsymbol{x}_i \in \Re^n, i = 1, \ldots, l$, derived from two classes where the class labels are $y_i \in \{-1, 1\}$, the optimization problem can be written as [3]:

$$\min \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \tag{1}$$

$$\text{subject to} \quad y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i)+b) \geq 1-\xi_i, \quad i=1,\ldots,l, \quad (2)$$

$$\xi_i \geq 0, \quad i=1,\ldots,l, \quad (3)$$

where C is a penalty parameter, and $\phi(\boldsymbol{x}_i)$ is a nonlinear mapping of $\boldsymbol{x}_i$ in a high-dimensional space. This problem is solved using its dual which is a convex quadratic programming problem:

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha}-\boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha}=0, \\ & 0\leq\alpha_i\leq C, \quad i=1,\ldots,l, \end{aligned} \quad (4)$$

where α_i's are Lagrange multipliers, $\boldsymbol{e}$ is a unit vector of all ones, $\boldsymbol{Q}$ is a positive semidefinite matrix, $Q_{ij}=y_iy_jK(\boldsymbol{x}_i,\boldsymbol{x}_j)$, and $K(\boldsymbol{x}_i,\boldsymbol{x}_j)=\langle\phi(\boldsymbol{x}_i)\cdot\phi(\boldsymbol{x}_j)\rangle$ is the inner product kernel. By using kernel, we can implicitly compute the inner product of the mapped data. The input data $\boldsymbol{x}_i$ corresponding to non-zero α_i are called support vectors.

The computational complexity of the optimization problem (4), which is a quadratic programming (QP) problem, depends on the dimension of the kernel matrix. The kernel matrix has a number of elements equal to the square of the number of training data. Usually, interior point method (IPM) is applied to solve QP problems. The most expensive step at every iteration of an IPM is the factorization cost, which is $O(l^3)$ and this operation requires $O(l^2)$ memory space. Although the SVM problem is well understood, for large learning tasks with many training data the general quadratic programs quickly become intractable in their memory and time requirements. Several researchers have proposed decomposition (chunking) methods to conquer this difficulty. In the decomposition method, the indices $\{1,2,\ldots,l\}$ of the training set are separated into two sets B and N, where B is the working set and $N=\{1,2,\ldots,l\}\backslash B$ is the fixed set. The working set has a constant size q much smaller than l. If we denote $\boldsymbol{\alpha}_B$ and $\boldsymbol{\alpha}_N$ as vectors containing the elements from working set B and fixed set N respectively, the objective function of the optimization problem (4) becomes $\frac{1}{2}\boldsymbol{\alpha}_B^T\boldsymbol{Q}_{BB}\boldsymbol{\alpha}_B-(\boldsymbol{e}_B-\boldsymbol{Q}_{BN}\boldsymbol{\alpha}_N)^T\boldsymbol{\alpha}_B+\frac{1}{2}\boldsymbol{\alpha}_N^T\boldsymbol{Q}_{NN}\boldsymbol{\alpha}_N-\boldsymbol{e}_N^T\boldsymbol{\alpha}_N$. Then the decomposition algorithm works as follows:

1. Select q variables for the working set B. The remaining $l-q$ variables are fixed at their current values.
2. Solve the following problem defined by the variables in set B

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{\alpha}_B^T\boldsymbol{Q}_{BB}\boldsymbol{\alpha}_B-(\boldsymbol{e}_B-\boldsymbol{Q}_{BN}\boldsymbol{\alpha}_N)^T\boldsymbol{\alpha}_B \\ \text{subject to} \quad & \boldsymbol{y}_B^T\boldsymbol{\alpha}_B=-\boldsymbol{y}_N^T\boldsymbol{\alpha}_N, \\ & 0\leq(\alpha_B)_i\leq C, \quad i=1,\ldots,q, \end{aligned} \quad (5)$$

   where

$$\mathrm{Q}=\begin{bmatrix}\mathrm{Q}_{BB} & \mathrm{Q}_{BN}\\ \mathrm{Q}_{NB} & \mathrm{Q}_{NN}\end{bmatrix}$$

   and q is the size of B.
3. If $\boldsymbol{\alpha}$ is the optimal solution of (4), stop. Otherwise, goto Step 1.

The subproblem (5) is solved by using standard convex quadratic programming solvers (e.g., LOQO [4]). Note that B is updated in each iteration. As the decomposition method finds an optimal solution of the subproblem (5), the strict decrease of the objective function holds and under some conditions this method converges to an optimal solution of (4) [5, 6]. Any large size data set can be tackled by the decomposition method because the memory requirements is linear in the number of training examples and linear in the number of support vectors.

In this paper we reformulate the SVM's quadratic programming (QP) problem into the second-order cone programming (SOCP) problem. To transform the QP problem into the SOCP problem, we have to decompose the kernel matrix. We apply two well-known factorization methods: (1) Cholesky factorization method, and (2) eigenvalue factorization method. We formulate two SOCP problems for each factorization method. Note that all these proposed methods have the same solution as the QP solution, and SOCPs also follow SVM's *Karush-Kuhn-Tucker* (KKT) conditions. As the kernel matrix is positive semidefinite, some columns of the decomposed matrix will diminish or some eigenvalues will be zero. It reduces the dimension of the matrix which is used in the new optimization method. Therefore, it requires less computational effort for low-rank kernel matrix. All these proposed methods are much faster than the quadratic programming solver LOQO[1] when the dimension of decomposed matrix (or the number of non-zero eigenvalues) is small enough.

We apply an interior-point method (IPM) to solve SOCP problems. We compute search directions (in each iteration of IPM) based on a reduced augmented equation that is derived by applying the block matrix splitting technique. Then, we apply a simple technique on factorization method which is applied to solve reduced augmented equations. The factorization cost of the proposed method is $O(lr^2)$ where r the rank of the kernel matrix and l is the size of the training set. This operation requires $O(lr)$ memory space. Exploiting the structure of SVM problem, we calculate many matrix-computations analytically. The proposed SOCP solvers for SVM problems are much faster than general-purpose SOCP solvers.

The SVM decomposition (chunking) method is necessary for solving large-scale problems. In this paper we also apply the SOCP method to solve optimization problems of the SVM decomposition method as described above to check the numerical stability and performance. Experimental results on various problems show that the decomposition (chunking) method including proposed SOCP solver is better or competitive with that using LOQO. The weakest part of decomposition method is that it can not consider all variables together. If the working set selection is not appropriate, though the strict decrease of the objective value holds, the decomposition method may converge very slowly. However, good sets of working data may lead the decomposition method to converge fast. Thus, working set selection is an important issue in the SVM decomposition method. Several existing working set selection approaches have been proposed in [2, 5–14]. Recently, Hsu and Lin [8] have proposed a working set selection approach which leads to faster convergence than the other existing approaches. In this paper we simply modify Hsu and Lin's approach. The modified approach is better than Hsu and Lin's approach for most of the problems.

Platt's sequential minimal optimization (SMO) algorithm is one of the fastest methods. The SMO is a decomposition method which restricts the size of working set to be two. The advantage of this method is that subproblem (5) becomes so small problem that no optimization software is required in practice. On the other hand, this method has a disadvantage that it selects active data randomly from the KKT violated variables; speeds for some problems are fast while (very) slow for other problems. Experimental results show that the decomposition method including proposed SOCP solver is better or competitive with Platt's SMO.

In [1], Fine and Scheinberg have proposed a method which uses a product form Cholesky factorization to solve the system of equations of an IPM applied in the QP problem. The factorization cost of their method is $O(lr^2)$ where r the rank of the kernel matrix. Since the product form Cholesky factorization method is applied instead of the straight Cholesky factorization method to solve the system of equations of an IPM, their method is (very) slower than the general QP solver LOQO if the kernel matrix has full rank or if the rank of the kernel matrix is not below to a prescribed bound. Experimental results show that the proposed method is faster than the method in [1] for both low-rank and full-rank kernel matrices. The 'incomplete' Cholesky factorization technique is applied in both our method and the method in [1] for low-rank kernel representation. If, however, the eigenvalues of the kernel matrix have a more complicated structure (varying from very small to relatively large), then it is crucial to find a best approximated matrix of decomposed matrix while keeping the rank of the approximating matrix below a prescribed bound. Therefore, for a large-scale problem (e.g., training set with 10,000 data or more), the SVM problem becomes infeasible without decomposition (chunking) method if the rank of the kernel matrix is not below to a prescribed bound. As the method in [1] is not applied in the chunking method, it is not clear about the numerical stability and performance especially for very large-scale problems (we will discuss in this point more details in Section 8).

After we submitted the first version of this paper, we found that Lanckriet et al. [15] have proposed a learning method that defines the kernel matrix as a combination of multiple kernels. To find the coefficients in the combination, a semidefinite programming (SDP) problem is needed to be solved (software 'SeDuMi' is used [15] to solve SDP problem). Then, they solve a quadratically constrained quadratic program (QCQP) problem (QCQP is a special instance of SOCP) to find the classifier. The commercial software 'Mosek' is used to solve the QCQP problem. Both 'SeDuMi' and 'Mosek' are general-purpose softwares. Their works focus on only improving the performance of classifier by combining several kernels over the classifier with the best individual kernel. However, our goal is to develop a fast SVM learning software for large-scale problems. In this paper, we reformulate SVM's QP problem into SOCP problem. More than that, we propose some techniques to enhance the efficiency of the interior-point methods by exploiting the structure of the problem appeared

in the SVM. In particular, the block matrix splitting is the key technique (see Section 4 for details). We also develop a SVM learning software for large-scale problems applying the SOCP into the decomposition method.

In the next section we briefly explain the SOCP problem. In Section 3, we reformulate the SVM's QP problem into the SOCP problems. Two types of SOCP formulation for each kernel matrix factorization method are shown in this section. We also discuss the computational complexity of each method. In Section 4, we introduce the implementation technique for SOCP problems applying the block matrix splitting technique. In Section 5, we briefly discuss the kernel matrix characteristics based on kernel parameter, input data dimension and size of the training data set. In Section 6, we compare the performance of our developed SOCP solver, the method in [1] and the QP solver LOQO. In Section 7, we describe a modification of Hsu and Lin's working set selection approach. Computational experiments are shown in Section 8. Section 9 concludes the paper.

2. The Second-Order Cone Programming (SOCP)

The second-order cone programming problem is to minimize or maximize a linear function over the intersection of an affine space with the Cartesian product of a finite number of second-order cones [16]. Recently, this problem has received considerable attention for its wide range of applications and for being easily solvable via interior-point methods (IPMs). In this section, we briefly introduce the SOCP. See [16, 17] for more about SOCP.

In this paper we consider the following SOCP problem:

$$\begin{aligned} (P) \qquad \min \quad & \sum_{i=1}^{n} \boldsymbol{c}_i^T \boldsymbol{x}_i \\ \text{subject to} \quad & \sum_{i=1}^{n} \boldsymbol{A}_i \boldsymbol{x}_i = \boldsymbol{b}, \\ & \boldsymbol{x}_i \in \mathcal{K}_i, i = 1, \ldots, n, \end{aligned} \tag{6}$$

where $\boldsymbol{x}_i \in \Re^{k_i}, i = 1, \ldots n$, are variables, $\boldsymbol{b} \in \Re^m$, $\boldsymbol{A}_i \in \Re^{m \times k_i}$ and $\boldsymbol{c}_i \in \Re^{k_i}, i = 1, \ldots n$, are data, and the set $\mathcal{K}_i, i = 1, \ldots n$, is the second-order cone of dimension k_i defined by

$$\mathcal{K}_i = \{\boldsymbol{x}_i = [x_{i0}; \boldsymbol{x}_{i1}] \in \Re \times \Re^{k_i - 1} : x_{i0} - \|\boldsymbol{x}_{i1}\| \geq 0\},$$

where x_{i0} is the first component of $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i1}$ is the vector consisting of the remaining components, and $\|\boldsymbol{x}_{i1}\|$ is the standard Euclidean norm. In particular, if the cone dimension k_i is 1, then the constraint $x_i \in \mathcal{K}_i$ is simply the standard non-negativity constraint $x_i \geq 0$, and such a variable is called a linear variable.

The dual of (P) is given by

$$\begin{aligned} (D) \qquad \max \quad & \boldsymbol{b}^T \boldsymbol{t} && (7) \\ \text{subject to} \quad & \boldsymbol{s}_i + \boldsymbol{A}_i^T \boldsymbol{t} = \boldsymbol{c}_i, \quad i = 1, \ldots, n, \\ & \boldsymbol{s}_i \in \mathcal{K}_i, \quad i = 1, \ldots, n, && (8) \end{aligned}$$

where $\boldsymbol{t} \in \Re^m$. Defining

$$\begin{aligned} K &= k_1 + \cdots + k_n, \\ \mathcal{K} &= \mathcal{K}_1 \times \cdots \times \mathcal{K}_n, \\ \boldsymbol{A} &= [\boldsymbol{A}_1 \ \boldsymbol{A}_2 \ \ldots \ \boldsymbol{A}_n] \in \Re^{m \times K}, \\ \boldsymbol{c} &= [\boldsymbol{c}_1; \boldsymbol{c}_2; \ldots; \boldsymbol{c}_n] \in \Re^K, \\ \boldsymbol{x} &= [\boldsymbol{x}_1; \boldsymbol{x}_2; \ldots; \boldsymbol{x}_n] \in \Re^K, \\ \boldsymbol{s} &= [\boldsymbol{s}_1; \boldsymbol{s}_2; \ldots; \boldsymbol{s}_n] \in \Re^K, \end{aligned}$$

problems (P) and (D) can be simply written as:

$$\begin{aligned} (P) \qquad \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, && (9) \\ & \boldsymbol{x} \in \mathcal{K}, \\ (D) \qquad \max \quad & \boldsymbol{b}^T \boldsymbol{t} \\ \text{subject to} \quad & \boldsymbol{s} + \boldsymbol{A}^T \boldsymbol{t} = \boldsymbol{c}, && (10) \\ & \boldsymbol{s} \in \mathcal{K}. \end{aligned}$$

The perturbed KKT conditions of the primal-dual systems (9) and (10) are:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b}, && \text{(primal feasibility)} \\ \boldsymbol{s} + \boldsymbol{A}^T \boldsymbol{t} &= \boldsymbol{c}, && \text{(dual feasibility)} \\ \boldsymbol{x}_i \circ \boldsymbol{s}_i &= \mu \boldsymbol{e}_i, \quad i = 1, \ldots, n, && \text{(complementary)} \\ \boldsymbol{x}, \boldsymbol{s} &\in \mathcal{K}^0, && (11) \end{aligned}$$

where $\mathcal{K}^0$ denotes the interior of the cone $\mathcal{K}$, $\boldsymbol{e}_i = [1; 0; \cdots; 0]$ is a vector of length k_i with the first element is one and the rest of the elements are zero, and

μ is a positive parameter that is to be driven to zero explicitly. Here

$$\boldsymbol{x}_i \circ \boldsymbol{s}_i = \left[\boldsymbol{x}_i^T \boldsymbol{s}_i; x_{i0}\boldsymbol{s}_{i1} + s_{i0}\boldsymbol{x}_{i1}\right]$$

is the multiplication inducing second-order cone. Indeed, with this multiplication, the space $\Re^n$ can be regarded as a Jordan algebra, and the SOCP can be considered as a special case of symmetric cone programming. For more details about Jordan algebra and IPM for symmetric cone programming, see [18]. Suppose that the interior of the primal and dual feasible region is nonempty. Then as μ varies, the solution to the perturbed KKT conditions (11) form a path (known as the central path) in the interior of the primal-dual feasible region, and as μ gradually reduces to zero, the path converges to an optimal solution of the primal and dual SOCP problems. The IPM introduced in Section 4 follows the central path numerically to get an optimal solution.

3. Reformulation of the SVM's Optimization Problem

In this section we reformulate the SVM's QP problem into the SOCP problem (see also [19] for a first approach). The SVM problem can be written as:

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \\ & \boldsymbol{\alpha} + \boldsymbol{\beta} = C\boldsymbol{e}, \\ & \boldsymbol{\alpha}, \boldsymbol{\beta} \geq \boldsymbol{0}, \quad \boldsymbol{\alpha}, \boldsymbol{\beta} \in \Re^l, \end{aligned}$$

where $\boldsymbol{\beta}$ is a slack variable vector. The square matrix $\boldsymbol{Q}$ is a symmetric positive semidefinite matrix. The $\boldsymbol{Q}$ can be represented as $\boldsymbol{Q} = \boldsymbol{G}\boldsymbol{G}^T$. If the rank of $\boldsymbol{Q}$ is r, $\boldsymbol{G}$ is a matrix with r columns and l rows.

3.1. Formulation 1

Assume that $\boldsymbol{Q}$ is decomposed as $\boldsymbol{Q} = \boldsymbol{G}\boldsymbol{G}^T$ where $\boldsymbol{G} \in \Re^{l\times r}$ and r is the rank of $\boldsymbol{Q}$. Then,

$$\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha} = \boldsymbol{\alpha}^T\boldsymbol{G}\boldsymbol{G}^T\boldsymbol{\alpha} = \|\boldsymbol{G}^T\boldsymbol{\alpha}\|^2.$$

Minimizing $\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha}$ is equivalent to minimizing θ under the constraint $\|\boldsymbol{G}^T\boldsymbol{\alpha}\|^2 \leq \theta$. This constraint is rewritten as

$$\left(\frac{\theta-1}{2}\right)^2 + \|\boldsymbol{G}^T\boldsymbol{\alpha}\|^2 \leq \left(\frac{\theta+1}{2}\right)^2.$$

Let

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{G}^T\boldsymbol{\alpha} \\ z_1 &= \frac{\theta+1}{2} \\ z_2 &= \frac{\theta-1}{2}. \end{aligned}$$

Then, the SVM problem can be written as:

$$\begin{aligned} \min \quad & \frac{1}{2}(z_1+z_2) - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0 \\ & \boldsymbol{\alpha} + \boldsymbol{\beta} = C\boldsymbol{e}, \\ & \boldsymbol{G}^T\boldsymbol{\alpha} - \boldsymbol{u} = \boldsymbol{0}, \quad \boldsymbol{u} \in \Re^r, \\ & z_1 - z_2 = 1, \\ & \boldsymbol{\alpha}, \boldsymbol{\beta} \geq \boldsymbol{0}, \\ & z_1^2 \geq z_2^2 + \|\boldsymbol{u}\|^2, \end{aligned} \tag{12}$$

which is an SOCP problem of the form (9) as follows.

$$\begin{aligned} \min \quad & \begin{bmatrix} -\boldsymbol{e}^T & \boldsymbol{0} & \frac{1}{2} & \frac{1}{2} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \\ z_1 \\ z_2 \\ \boldsymbol{u} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} \boldsymbol{y}^T & \boldsymbol{0} & 0 & 0 & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}^T & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & 1 & -1 & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \\ z_1 \\ z_2 \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} 0 \\ C\boldsymbol{e} \\ \boldsymbol{0} \\ 1 \end{bmatrix} \\ & \boldsymbol{\alpha}, \boldsymbol{\beta} \geq \boldsymbol{0}, \\ & z_1^2 \geq z_2^2 + \|\boldsymbol{u}\|^2. \end{aligned} \tag{13}$$

The dual of (13) is

$$\max \quad \begin{bmatrix} 0 & C\boldsymbol{e}^T & \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} t_0 \\ \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ t_3 \end{bmatrix} \tag{14}$$

$$\text{subject to} \quad \begin{bmatrix} \boldsymbol{s}_0 \\ \boldsymbol{s}_1 \\ s_2 \\ s_3 \\ \boldsymbol{s}_4 \end{bmatrix} + \begin{bmatrix} \boldsymbol{y} & \boldsymbol{I} & \boldsymbol{G} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ 0 & \boldsymbol{0} & \boldsymbol{0} & 1 \\ 0 & \boldsymbol{0} & \boldsymbol{0} & -1 \\ \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} t_0 \\ \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ t_3 \end{bmatrix}$$
$$= \begin{bmatrix} -\boldsymbol{e} \\ \boldsymbol{0} \\ 1/2 \\ 1/2 \\ \boldsymbol{0} \end{bmatrix}$$
$$\boldsymbol{s}_0, \boldsymbol{s}_1 \geq \boldsymbol{0},$$
$$s_2^2 \geq s_3^2 + \|\boldsymbol{s}_4\|^2,$$

or in a similar form:

$$\begin{aligned} \max \quad & C\boldsymbol{e}^T\boldsymbol{t}_1 + t_3 && (15) \\ \text{subject to} \quad & \boldsymbol{s}_0 + t_0\boldsymbol{y} + \boldsymbol{t}_1 + \boldsymbol{G}\boldsymbol{t}_2 = -\boldsymbol{e} && (16) \\ & \boldsymbol{s}_1 + \boldsymbol{t}_1 = \boldsymbol{0}, && (17) \\ & s_2 + t_3 = 1/2, && (18) \\ & s_3 - t_3 = 1/2, && (19) \\ & \boldsymbol{s}_4 - \boldsymbol{t}_2 = \boldsymbol{0}, && (20) \\ & \boldsymbol{s}_0, \boldsymbol{s}_1 \geq \boldsymbol{0}, && (21) \\ & s_2^2 \geq s_3^2 + \|\boldsymbol{s}_4\|^2. && (22) \end{aligned}$$

Lemma 1. *The problem* (15)–(22) *and* (1)–(3) *are equivalent, and* $b = -t_0$.

The proof is in Appendix 9. Thus, if the QP problem in (4) has an optimal solution of (1)–(3) then the SOCP in (12) has the same optimal solution of (1)–(3).

3.2. Formulation 2

We can write $\boldsymbol{G} = [\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_r]$ where $\boldsymbol{g}_i \in \Re^l$ is the ith column vector of matrix $\boldsymbol{G} \in \Re^{l \times r}$. Then

$$\boldsymbol{Q} = \boldsymbol{G}\boldsymbol{G}^T = \sum_{i=1}^{r} \boldsymbol{g}_i \boldsymbol{g}_i^T,$$

and

$$\boldsymbol{\alpha}^T \boldsymbol{Q} \boldsymbol{\alpha} = \boldsymbol{\alpha}^T \sum_{i=1}^{r} \boldsymbol{g}_i \boldsymbol{g}_i^T \boldsymbol{\alpha} = \sum_{i=1}^{r} \boldsymbol{\alpha}^T \boldsymbol{g}_i \boldsymbol{g}_i^T \boldsymbol{\alpha}$$
$$= \sum_{i=1}^{r} \left\| \boldsymbol{g}_i^T \boldsymbol{\alpha} \right\|^2.$$

Again, minimizing $\boldsymbol{\alpha}^T \boldsymbol{Q} \boldsymbol{\alpha}$ is equivalent to minimizing $\sum_{i=1}^{r} \theta_i$ under the constraints $\|\boldsymbol{g}_i^T \boldsymbol{\alpha}\|^2 \leq \theta_i, i = 1, \ldots, r$. These constraints are rewritten as

$$\left(\frac{\theta_i - 1}{2} \right)^2 + \left\| \boldsymbol{g}_i^T \boldsymbol{\alpha} \right\|^2 \leq \left(\frac{\theta_i + 1}{2} \right)^2, \quad i = 1, \ldots, r.$$

Let for $i = 1, \ldots, r$,

$$u_i = \boldsymbol{g}_i^T \boldsymbol{\alpha}$$
$$z_{1i} = \frac{\theta_i + 1}{2}$$
$$z_{2i} = \frac{\theta_i - 1}{2}.$$

Then, the SVM's QP problem is transformed into the following SOCP problem as follows:

$$\begin{aligned} \min \quad & \frac{1}{2} \sum_{i=1}^{r} (z_{1i} + z_{2i}) - \boldsymbol{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T \boldsymbol{\alpha} = 0 \\ & \boldsymbol{\alpha} + \boldsymbol{\beta} = C\boldsymbol{e}, && (23) \\ & \boldsymbol{g}_i^T \boldsymbol{\alpha} - u_i = 0, \quad i = 1, \ldots, r, \\ & z_{1i} - z_{2i} = 1, \quad i = 1, \ldots, r, \\ & \boldsymbol{\alpha}, \boldsymbol{\beta} \geq \boldsymbol{0}, \\ & z_{1i}^2 \geq z_{2i}^2 + u_i^2, \quad i = 1, \ldots, r. \end{aligned}$$

The dual of (23) is

$$\begin{aligned} \max \quad & C\boldsymbol{e}^T\boldsymbol{t}_1 + \sum_{i=1}^{r} t_{3i} \\ \text{subject to} \quad & \boldsymbol{s}_0 + t_0\boldsymbol{y} + \boldsymbol{t}_1 + \sum_{i=1}^{r} t_{2i}\boldsymbol{g}_i = -\boldsymbol{e} \\ & \boldsymbol{s}_1 + \boldsymbol{t}_1 = \boldsymbol{0}, \\ & s_{2i} + t_{3i} = 1/2, \quad i = 1, \ldots, r, \\ & s_{3i} - t_{3i} = 1/2, \quad i = 1, \ldots, r, && (24) \\ & s_{4i} - t_{2i} = 0, \quad i = 1, \ldots, r, \\ & \boldsymbol{s}_0, \boldsymbol{s}_1 \geq \boldsymbol{0}, \\ & s_{2i}^2 \geq s_{3i}^2 + s_{4i}^2, \quad i = 1, \ldots, r. \end{aligned}$$

The IPM for SOCP needs matrix computation in each iteration, and the computational cost of the SOCP solver depends on both the dimension of second-order cones and the number of second-order cones. In the SOCP-Formulation 1, there is only one $(r + 2)$-dimensional second-order cone, while there are

r 3-dimensional second-order cones in the SOCP-Formulation 2. As a result, SOCP-Formulation 1 needs to compute matrix computation of $(r+2) \times (r+2)$ matrices, while SOCP-Formulation 2 computes a block diagonal matrix where the number of blocks is r and each block is 3×3 matrix. The sparse structure of SOCP-Formulation 2 is more desirable in IPMs. Furthermore, as each matrix size is 3×3, we compute some matrices calculations analytically as shown in Section 4. Thus, the SOCP-Formulation 2 is faster than the SOCP-Formulation 1.

3.3. Matrix Factorization Methods

In this paper we apply two approaches to decompose the matrix $\boldsymbol{Q}$. The first approach is Cholesky factorization method. Applying Cholesky factorization method to $\boldsymbol{Q}$, it is decomposed into the product of two symmetric matrix as $\boldsymbol{Q} = \boldsymbol{G}\boldsymbol{G}^T$ where $\boldsymbol{G}$ is a lower triangular matrix, and each element of $\boldsymbol{G}$ is given by

$$G_{ii} = \left(Q_{ii} - \sum_{k=1}^{i-1} G_{ik}^2 \right)^{1/2}$$

and

$$G_{ji} = \frac{1}{G_{ii}} \left(Q_{ij} - \sum_{k=1}^{i-1} G_{ik} G_{jk} \right), \quad j = i+1, i+2, \ldots, n.$$

If $\boldsymbol{Q}$ is positive semidefinite and singular, then it is still possible to compute an 'incomplete' Cholesky factorization $\boldsymbol{G}\boldsymbol{G}^T$, where some columns of $\boldsymbol{G}$ are zero. In this case, if a zero G_{ii} is encountered then G_{ji}'s for $j = i+1, i+2, \ldots, n$ are zero. Even if G_{ii} is not precisely zero, but very small, Cholesky factorization may be unstable. We set $G_{ii} = 0$ if $G_{ii} < \eta$, where $\eta > 0$ is a threshold value, and G_{ji}'s for $j = i+1, i+2, \ldots, n$ at zero. This setting gives numerical stability, and reduces the columns of $\boldsymbol{G}$ as well. In this paper, we set η at 10^{-3}. Thus, if there are r positive G_{ii}, $\boldsymbol{G} \in \Re^{n \times n}$ is represented as $\boldsymbol{G} \in \Re^{n \times r}$, and $\boldsymbol{G}\boldsymbol{G}^T \approx \boldsymbol{Q}$. According to this representation of $\boldsymbol{Q}$, we can derive SOCP-Formulation 1 for SVM problem. When we represent $\boldsymbol{G}$ as $\boldsymbol{G} = [\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_r]$ where $\boldsymbol{g}_i \in \Re^l$ is the ith column vector of matrix $\boldsymbol{G} \in \Re^{l \times r}$, SOCP-Formulation 2 is driven.

The second approach is the eigenvalue factorization method. Let $\lambda_1, \lambda_2, \ldots, \lambda_l$ are the eigenvalues and $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_l$ are the corresponding eigenvectors of $\boldsymbol{Q}$. As $\boldsymbol{Q}$ is a symmetric matrix, we can write,

$$\boldsymbol{Q} = \sum_{i=1}^{l} \lambda_i \boldsymbol{q}_i \boldsymbol{q}_i^T$$

where all eigenvalues are positive. As the kernel matrix is positive semidefinite we may get some eigenvalues are very small or zero. If $\lambda_i < \eta'$, where $\eta' > 0$ is a threshold value, we set $\lambda_i = 0$. In this paper we set η' at 10^{-6}. If the number of positive eigenvalues are $r' < l$, then,

$$\boldsymbol{Q} \approx \sum_{i=1}^{r'} \lambda_i \boldsymbol{q}_i \boldsymbol{q}_i^T \tag{25}$$

When $\boldsymbol{Q}$ is decomposed according to (25), SOCP-Formulation 2 is driven. The representation of $\boldsymbol{Q}$ by using eigenvalues can also be written in the following way:

$$\begin{aligned} \boldsymbol{Q} &\approx \sum_{i=1}^{r'} \lambda_i \boldsymbol{q}_i \boldsymbol{q}_i^T \\ &= \sum_{i=1}^{r'} \left(\sqrt{\lambda_i} \boldsymbol{q}_i \right) \left(\sqrt{\lambda_i} \boldsymbol{q}_i^T \right) \\ &= \sum_{i=1}^{r'} \widehat{\boldsymbol{q}}_i \widehat{\boldsymbol{q}}_i^T \\ &= \hat{\boldsymbol{G}} \hat{\boldsymbol{G}}^T \end{aligned}$$

According to this representation of $\boldsymbol{Q}$, SOCP-Formulation 1 is driven. We discussed earlier that SOCP-Formulation 2 is faster than SOCP-Formulation 1. Furthermore, SOCP-Formulation 2 with Cholesky factorization is faster than that with the eigenvalue factorization. This is because: (1) Cholesky factorization method is faster than the eigenvalue factorization method, (2) Cholesky factorization produces lower triangular matrix (on an average $(r \times r)/2$ elements of matrix $\boldsymbol{G}$ are zero) while the decomposed matrix from the eigenvalue factorization method is a full dense matrix.

4. Implementation Technique

In this section, we describe how to calculate the search direction of an IPM for the SOCP in (23) efficiently.

We use the so-called Mehrotra predictor-corrector algorithm with the HKM search direction which is considered to be one of the most efficient methods for SOCP problems [16, 18, 20–25]. At every iteration of interior-point method there are two basic steps: (1) predictor step and (2) corrector step. In the predictor step, the algorithm predict the best reduction in the duality gap, by evaluating a step directly towards optimality. The corrector step enforces the centrality and also takes into account the approximate curvature of the central path estimated by the predictor step. For the predictor step we compute $(\Delta \boldsymbol{x}, \Delta \boldsymbol{s}, \Delta \boldsymbol{t}) \in \Re^K \times \Re^K \times \Re^m$ of the linear system of equations satisfying,

$$\boldsymbol{A}\Delta \boldsymbol{x} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \tag{26}$$

$$\Delta \boldsymbol{s} + \boldsymbol{A}^T \Delta \boldsymbol{t} = \boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T \boldsymbol{t} \tag{27}$$

$$(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} + \boldsymbol{P}^{-1}\boldsymbol{s} \circ \boldsymbol{P}\Delta \boldsymbol{x} = -(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\boldsymbol{s}, \tag{28}$$

where $\boldsymbol{P}$ is a linear operator defined below. Consider $\boldsymbol{s}_i$ is a 3-dimensional second-order cone as $\boldsymbol{s}_i = \left[\begin{smallmatrix} s_0 \\ \boldsymbol{s}_1 \end{smallmatrix}\right]$ with $s_0 > \|\boldsymbol{s}_1\|$, then

$$\boldsymbol{s}_i^{-1} = \frac{1}{\gamma^2}\begin{bmatrix} s_0 \\ -\boldsymbol{s}_1 \end{bmatrix},$$

where $\gamma = \sqrt{s_0^2 - \|\boldsymbol{s}_1\|^2}$ and

$$\boldsymbol{P}_i = \begin{bmatrix} s_0 & \boldsymbol{s}_1 & s_2 \\ \boldsymbol{s}_1 & \frac{\boldsymbol{s}_1^2}{s_0+\gamma} + \gamma & \frac{\boldsymbol{s}_1 s_2}{s_0+\gamma} \\ s_2 & \frac{\boldsymbol{s}_1 s_2}{s_0+\gamma} & \frac{s_2^2}{s_0+\gamma} + \gamma \end{bmatrix},$$

$$\boldsymbol{P}_i^{-1} = \frac{1}{\gamma^2}\begin{bmatrix} s_0 & -\boldsymbol{s}_1 & -s_2 \\ -\boldsymbol{s}_1 & \frac{\boldsymbol{s}_1^2}{s_0+\gamma} + \gamma & \frac{\boldsymbol{s}_1 s_2}{s_0+\gamma} \\ -s_2 & \frac{\boldsymbol{s}_1 s_2}{s_0+\gamma} & \frac{s_2^2}{s_0+\gamma} + \gamma \end{bmatrix}.$$

Note that the SOCP formulation for SVM has 1-dimensional cones also. The formulation of $\boldsymbol{s}_i^{-1}$ and $\boldsymbol{P}_i$ for the 1-dimensional cones is the same as that of 3-dimensional cone. From the above definition of $\boldsymbol{s}_i^{-1}$ and $\boldsymbol{P}_i$, we get

$$\boldsymbol{s}^{-1} \circ \boldsymbol{s} = \boldsymbol{s} \circ \boldsymbol{s}^{-1} = \boldsymbol{e}, \quad \boldsymbol{e} \circ \boldsymbol{x} = \boldsymbol{x} \circ \boldsymbol{e} = \boldsymbol{x},$$
$$\boldsymbol{P}^{-1}\boldsymbol{e} = \boldsymbol{s}^{-1}, \quad \boldsymbol{P}^{-1}\boldsymbol{s} = \boldsymbol{e},$$

where $\boldsymbol{P} = \text{diag}(\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n)$ is a block diagonal matrix with $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n$ as its diagonal blocks, $\boldsymbol{e} = [\boldsymbol{e}_1; \boldsymbol{e}_2; \ldots; \boldsymbol{e}_n]$ and $\boldsymbol{e}_i = [1; 0; 0]$ (for 3-dimensional cones). Putting the value $\boldsymbol{P}^{-1}\boldsymbol{s} = \boldsymbol{e}$ in (28), we get

$$(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} + \boldsymbol{P}\Delta \boldsymbol{x} = -\boldsymbol{P}\boldsymbol{x}$$
$$\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} + \Delta \boldsymbol{x} = -\boldsymbol{x}$$
(premultiplied by $\boldsymbol{P}^{-1}$)
$$\boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} + \boldsymbol{A}\Delta \boldsymbol{x} = -\boldsymbol{A}\boldsymbol{x}$$
(premultiplied by $\boldsymbol{A}$)
$$\boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} = -\boldsymbol{b}$$

Multiplied by $\boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}$ into (27), we obtain

$$\boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} + \boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\boldsymbol{A}^T \Delta \boldsymbol{t} = \boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}(\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T \boldsymbol{t}) \tag{29}$$

$$\boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\boldsymbol{A}^T \Delta \boldsymbol{t} = \boldsymbol{A}\boldsymbol{P}^{-1}(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}(\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T \boldsymbol{t}) + \boldsymbol{b} \tag{30}$$

Define linear operator $\boldsymbol{L}(\boldsymbol{u})$, $\boldsymbol{u} \in \Re^n$

$$\boldsymbol{L}(\boldsymbol{u})\boldsymbol{x} = \boldsymbol{u} \circ \boldsymbol{x} \qquad \text{where}$$
$$\boldsymbol{L}(\boldsymbol{u}) = \begin{bmatrix} u_0 & \boldsymbol{u}_1^T \\ \boldsymbol{u}_1 & u_0 \boldsymbol{I} \end{bmatrix}.$$

Now (30) becomes

$$\boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1}\boldsymbol{A}^T \Delta \boldsymbol{t} = \boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1}(\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T \boldsymbol{t}) + \boldsymbol{b} \tag{31}$$

Note that $\boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1}\boldsymbol{A}^T$ is positive definite. We can apply Cholesky factorization method to solve the set of equations. From the value of $\Delta \boldsymbol{t}$, we get

$$\Delta \boldsymbol{s} = \boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T \boldsymbol{y} - \boldsymbol{A}^T \Delta \boldsymbol{t} \tag{32}$$

and

$$\Delta \boldsymbol{x} = -\boldsymbol{x} - \boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-T}\Delta \boldsymbol{s} \tag{33}$$

To compute the corrector step, we set $\Delta \boldsymbol{x}_p = \Delta \boldsymbol{x}$ and $\Delta \boldsymbol{s}_p = \Delta \boldsymbol{s}$ and compute new $(\Delta \boldsymbol{x}, \Delta \boldsymbol{s}, \Delta \boldsymbol{t})$ satisfying,

$$\boldsymbol{A}\Delta \boldsymbol{x} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} \tag{34}$$

$$\Delta \boldsymbol{s} + \boldsymbol{A}^T \Delta \boldsymbol{t} = \boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T \boldsymbol{t} \tag{35}$$

$$(\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s} + \boldsymbol{P}^{-1}\boldsymbol{s} \circ \boldsymbol{P}\Delta \boldsymbol{x} = \mu \boldsymbol{e} - (\boldsymbol{P}\boldsymbol{x}) \circ \boldsymbol{P}^{-1}\boldsymbol{s} - (\boldsymbol{P}\Delta \boldsymbol{x}_p) \circ \boldsymbol{P}^{-1}\Delta \boldsymbol{s}_p \tag{36}$$

where $\mu = \mu_a(\frac{\mu_a}{\mu_b})^2$, and where $\mu_a = (\boldsymbol{x} + \Delta \boldsymbol{x}_p)^T(\boldsymbol{s} + \Delta \boldsymbol{s}_p)$ and $\mu_b = \boldsymbol{x}^T \boldsymbol{s}$. Solving these equations as previous, we get

$$\begin{aligned} &\boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1}\boldsymbol{A}^T\Delta\boldsymbol{t} \\ &\quad = -\mu\boldsymbol{A}\boldsymbol{s}^{-1} + \boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1}\left(\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T\boldsymbol{t}\right) \\ &\qquad + \boldsymbol{b} + \boldsymbol{A}\boldsymbol{P}^{-1}\boldsymbol{L}\left(\boldsymbol{P}\Delta\boldsymbol{x}_p\right)\boldsymbol{P}^{-1}\Delta\boldsymbol{s}_p \end{aligned} \tag{37}$$

From the value of $\Delta \boldsymbol{t}$, we get

$$\Delta\boldsymbol{s} = \boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T\boldsymbol{t} - \boldsymbol{A}^T\Delta\boldsymbol{t} \tag{38}$$

and

$$\begin{aligned} \Delta\boldsymbol{x} = \mu\boldsymbol{s}^{-1} - \boldsymbol{x} - \boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1}\Delta\boldsymbol{s} \\ - \boldsymbol{P}^{-1}\boldsymbol{L}\left(\boldsymbol{P}\Delta\boldsymbol{x}_p\right)\boldsymbol{P}^{-1}\Delta\boldsymbol{s}_p \end{aligned} \tag{39}$$

Solving the search directions $(\Delta \boldsymbol{x}^k, \Delta \boldsymbol{s}^k, \Delta \boldsymbol{t}^k)$ are computationally most expensive in each iteration of an IPM. We compute the search directions based on a reduced augmented equation that is derived by applying the block matrix splitting technique. The reduced augmented equation has generally much smaller size compared to the original augmented equation. Exploiting the structure of SVM problem, we compute many matrix calculations analytically. Numerical results show that reduced augmented equation based IPM, together with analytic calculation of many matrices computation, computationally very faster and more stable than the typical IPM. The method is described below. According to the SOCP-Formulation 2,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{y}^T & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}^T & \boldsymbol{0} & \boldsymbol{\epsilon}_1 \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\epsilon}_2 \end{bmatrix}, \tag{40}$$

where identity matrix $\boldsymbol{I}$ is $l \times l$ matrix, $\boldsymbol{y}$ is a vector of length l (target output), $\boldsymbol{G} \in \Re^{l \times r}$ is decomposed lower triangular matrix of the matrix $\boldsymbol{Q}$, and

$$\boldsymbol{\epsilon}_1 = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & \cdots & 0 & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 \end{bmatrix}, \tag{41}$$

and

$$\boldsymbol{\epsilon}_2 = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -1 & -1 & 0 \end{bmatrix}, \tag{42}$$

are $r \times 3r$ matrix.

$$\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{P}\boldsymbol{x})\boldsymbol{P}^{-1} = \begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}_2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{H} \end{bmatrix} \tag{43}$$

where $\boldsymbol{D}_i$'s are $l \times l$ diagonal matrix and

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}_2 & \cdots & \boldsymbol{0} \\ & & \ddots & \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{H}_r \end{bmatrix} \tag{44}$$

$$= \begin{bmatrix} a_1 & a_2 & a_3 & & & & & & & \\ a_2 & b_2 & b_3 & & \boldsymbol{0} & & & & \boldsymbol{0} & \\ a_3 & b_3 & c_3 & & & & & & & \\ & & & d_1 & d_2 & d_3 & & & & \\ & \boldsymbol{0} & & d_2 & e_2 & e_3 & & & \boldsymbol{0} & \\ & & & d_3 & e_3 & f_3 & & & & \\ & & & & & & \ddots & & & \\ & & & & & & & g_1 & g_2 & g_3 \\ & \boldsymbol{0} & & & \boldsymbol{0} & & & g_2 & h_2 & h_3 \\ & & & & & & & g_3 & h_3 & i_3 \end{bmatrix}, \tag{45}$$

where

$$\boldsymbol{H}_i = \boldsymbol{P}_i^{-1}\boldsymbol{L}(\boldsymbol{P}_i\boldsymbol{x}_i)\boldsymbol{P}_i^{-1} = \frac{1}{\gamma^2}\begin{bmatrix} s_0^i x_0^i - s_1^i x_1^i - s_2^i x_2^i & s_0^i x_1^i - s_1^i x_0^i & s_0^i x_2^i - s_2^i x_0^i \\ s_0^i x_1^i - s_1^i x_0^i & s_0^i x_0^i - s_1^i x_1^i - s_2^i x_2^i & s_1^i x_2^i - s_2^i x_1^i \\ s_0^i x_2^i - s_2^i x_0^i & s_1^i x_2^i - s_2^i x_1^i & s_0^i x_0^i - s_1^i x_1^i - s_2^i x_2^i \end{bmatrix} \tag{46}$$

are computed analytically for each 3-dimensional cone. The diagonal elements for $\boldsymbol{D}_i$ are calculated in a similar way on one-dimension cones. In a typical IPM iteration $\boldsymbol{A} \in \Re^{(l+1+2r)\times(2l+3r)}$, $\boldsymbol{P}^{-1}\boldsymbol{L}(\boldsymbol{Px})\boldsymbol{P}^{-T} \in \Re^{(2l+2r)\times(2l+3r)}$, and

$$\begin{aligned}
&\boldsymbol{AP}^{-1}\boldsymbol{L}(\boldsymbol{Px})\boldsymbol{P}^{-1}\boldsymbol{A}^T \\
&= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{y}^T & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}^T & \boldsymbol{0} & \boldsymbol{\epsilon}_1 \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\epsilon}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}_2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{H} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{y} & \boldsymbol{G} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\epsilon}_1^T & \boldsymbol{\epsilon}_2^T \end{bmatrix} \qquad (47) \\
&= \begin{bmatrix} \boldsymbol{D}_1 + \boldsymbol{D}_2 & \boldsymbol{D}_1\boldsymbol{y} & \boldsymbol{D}_1\boldsymbol{G} & \boldsymbol{0} \\ \boldsymbol{y}^T\boldsymbol{D}_1 & \boldsymbol{y}^T\boldsymbol{D}_1\boldsymbol{y} & \boldsymbol{y}^T\boldsymbol{D}_1\boldsymbol{G} & \boldsymbol{0} \\ \boldsymbol{G}^T\boldsymbol{D}_1 & \boldsymbol{G}^T\boldsymbol{D}_1\boldsymbol{y} & \boldsymbol{G}^T\boldsymbol{D}_1\boldsymbol{G} + \boldsymbol{\epsilon}_1\boldsymbol{H}\boldsymbol{\epsilon}_1^T & \boldsymbol{\epsilon}_1\boldsymbol{H}\boldsymbol{\epsilon}_2^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\epsilon}_2\boldsymbol{H}\boldsymbol{\epsilon}_1^T & \boldsymbol{\epsilon}_2\boldsymbol{H}\boldsymbol{\epsilon}_2^T \end{bmatrix} \qquad (48)
\end{aligned}$$

Let,

$$\begin{aligned}
\tilde{C} &= \boldsymbol{D}_1 + \boldsymbol{D}_2 \\
\tilde{\boldsymbol{y}}^T &= \boldsymbol{y}^T\boldsymbol{D}_1 \Leftrightarrow \tilde{\boldsymbol{y}} = \boldsymbol{D}_1\boldsymbol{y} \\
\tilde{c} &= \boldsymbol{y}^T\boldsymbol{D}_1\boldsymbol{y} \\
\tilde{\boldsymbol{G}}^T &= \boldsymbol{G}^T\boldsymbol{D}_1 \Leftrightarrow \tilde{\boldsymbol{G}} = \boldsymbol{D}_1\boldsymbol{G} \\
\tilde{\boldsymbol{r}}^T &= \boldsymbol{y}^T\boldsymbol{D}_1\boldsymbol{G} = \boldsymbol{y}^T\tilde{\boldsymbol{G}} \Leftrightarrow \tilde{\boldsymbol{r}} = \tilde{\boldsymbol{G}}^T\boldsymbol{y} \\
\tilde{\boldsymbol{R}} &= \boldsymbol{G}^T\boldsymbol{D}_1\boldsymbol{G} + \boldsymbol{\epsilon}_1\boldsymbol{H}\boldsymbol{\epsilon}_1^T = \tilde{\boldsymbol{G}}^T\boldsymbol{G} + \tilde{\boldsymbol{P}}_1 \\
\tilde{\boldsymbol{P}}_1 &= \boldsymbol{\epsilon}_1\boldsymbol{H}\boldsymbol{\epsilon}_1^T \\
\tilde{\boldsymbol{P}}_{12} &= \boldsymbol{\epsilon}_1\boldsymbol{H}\boldsymbol{\epsilon}_2^T \Leftrightarrow \tilde{\boldsymbol{P}}_{12}^T = \boldsymbol{\epsilon}_2\boldsymbol{H}\boldsymbol{\epsilon}_1^T \\
\tilde{\boldsymbol{P}}_2 &= \boldsymbol{\epsilon}_2\boldsymbol{H}\boldsymbol{\epsilon}_2^T \\
\tilde{\boldsymbol{b}} &= [\tilde{\boldsymbol{b}}_1; \tilde{b}_2; \tilde{\boldsymbol{b}}_3; \tilde{\boldsymbol{b}}_4] = \boldsymbol{AP}^{-1}\boldsymbol{L}(\boldsymbol{Px})\boldsymbol{P}^{-1} \\
&\quad \times (\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T\boldsymbol{t}) + \boldsymbol{b}, \quad \text{or} \\
[\tilde{\boldsymbol{b}}_1; \tilde{b}_2; \tilde{\boldsymbol{b}}_3; \tilde{\boldsymbol{b}}_4] &= -\mu\boldsymbol{A}\boldsymbol{s}^{-1} + \boldsymbol{AP}^{-1}\boldsymbol{L}(\boldsymbol{Px})\boldsymbol{P}^{-1} \\
&\quad \times (\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T\boldsymbol{t}) + \boldsymbol{b} + \boldsymbol{AP}^{-1}\boldsymbol{L} \\
&\quad \times (\boldsymbol{P}\Delta\boldsymbol{x}_p)\boldsymbol{P}^{-1}\Delta\boldsymbol{s}_p \\
\boldsymbol{b} &= [\boldsymbol{Ce}; 0; \boldsymbol{0}; \boldsymbol{1}] \\
\boldsymbol{c} &= [-\boldsymbol{e}^T; \boldsymbol{0}; \boldsymbol{k}; \ldots; \boldsymbol{k}], \\
\boldsymbol{k} &= \left[\frac{1}{2}; \quad \frac{1}{2}; \quad 0\right].
\end{aligned}$$

Again, we define $\boldsymbol{x} = [\boldsymbol{x}_1; \boldsymbol{x}_2; \boldsymbol{x}_3] \in \Re^{(l+l+3r)}$, $\boldsymbol{s} = [\boldsymbol{s}_1; \boldsymbol{s}_2; \boldsymbol{s}_3] \in \Re^{(l+l+3r)}$ and $\boldsymbol{t} = [\boldsymbol{t}_1; t_2; \boldsymbol{t}_3; \boldsymbol{t}_4] \in \Re^{(l+1+r+r)}$. Now, from (31) or (37), we get

$$\begin{bmatrix} \tilde{\boldsymbol{C}} & \tilde{\boldsymbol{y}} & \tilde{\boldsymbol{G}} & \boldsymbol{0} \\ \tilde{\boldsymbol{y}}^T & \tilde{c} & \tilde{\boldsymbol{r}}^T & \boldsymbol{0} \\ \tilde{\boldsymbol{G}}^T & \tilde{\boldsymbol{r}} & \tilde{\boldsymbol{R}} & \tilde{\boldsymbol{P}}_{12} \\ \boldsymbol{0} & \boldsymbol{0} & \tilde{\boldsymbol{P}}_{12}^T & \tilde{\boldsymbol{P}}_2 \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{t}_1 \\ \Delta t_2 \\ \Delta\boldsymbol{t}_3 \\ \Delta\boldsymbol{t}_4 \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{b}}_1 \\ \tilde{b}_2 \\ \tilde{\boldsymbol{b}}_3 \\ \tilde{\boldsymbol{b}}_4 \end{bmatrix} \qquad (49)$$

$$\tilde{\boldsymbol{C}}\Delta\boldsymbol{t}_1 + \tilde{\boldsymbol{y}}\Delta t_2 + \tilde{\boldsymbol{G}}\Delta\boldsymbol{t}_3 = \tilde{\boldsymbol{b}}_1 \qquad (50)$$

$$\tilde{\boldsymbol{y}}^T\Delta\boldsymbol{t}_1 + \tilde{c}\Delta t_2 + \tilde{\boldsymbol{r}}^T\Delta\boldsymbol{t}_3 = \tilde{b}_2 \qquad (51)$$

$$\tilde{\boldsymbol{G}}^T\Delta\boldsymbol{t}_1 + \tilde{\boldsymbol{r}}\Delta t_2 + \tilde{\boldsymbol{R}}\Delta\boldsymbol{t}_3 + \tilde{\boldsymbol{P}}_{12}\Delta\boldsymbol{t}_4 = \tilde{\boldsymbol{b}}_3 \qquad (52)$$

$$\tilde{\boldsymbol{P}}_{12}^T\Delta\boldsymbol{t}_3 + \tilde{\boldsymbol{P}}_2\Delta\boldsymbol{t}_4 = \tilde{\boldsymbol{b}}_4 \qquad (53)$$

From (53)

$$\Delta\boldsymbol{t}_4 = \tilde{\boldsymbol{P}}_2^{-1}\big(\tilde{\boldsymbol{b}}_4 - \tilde{\boldsymbol{P}}_{12}^T\Delta\boldsymbol{t}_3\big). \qquad (54)$$

From (52) and (54)

$$\begin{aligned}
&\tilde{\boldsymbol{G}}^T\Delta\boldsymbol{t}_1 + \tilde{\boldsymbol{r}}\Delta t_2 + \big(\tilde{\boldsymbol{R}} - \tilde{\boldsymbol{P}}_{12}\tilde{\boldsymbol{P}}_2^{-1}\tilde{\boldsymbol{P}}_{12}^T\big)\Delta\boldsymbol{t}_3 \\
&\quad = \tilde{\boldsymbol{b}}_3 - \tilde{\boldsymbol{P}}_{12}\tilde{\boldsymbol{P}}_2^{-1}\tilde{\boldsymbol{b}}_4. \qquad (55)
\end{aligned}$$

The coefficient matrix $(\tilde{\boldsymbol{R}} - \tilde{\boldsymbol{P}}_{12}\tilde{\boldsymbol{P}}_2^{-1}\tilde{\boldsymbol{P}}_{12}^T)$ is known as Schur complement matrix. It is symmetric and positive definite. We solve the following set of equations applying Cholesky factorization:

$$\begin{aligned}
&\begin{bmatrix} \tilde{\boldsymbol{C}} & \tilde{\boldsymbol{y}} & \tilde{\boldsymbol{G}} \\ \tilde{\boldsymbol{y}}^T & \tilde{c} & \tilde{\boldsymbol{r}}^T \\ \tilde{\boldsymbol{G}}^T & \tilde{\boldsymbol{r}} & \tilde{\boldsymbol{R}} - \tilde{\boldsymbol{P}}_{12}\tilde{\boldsymbol{P}}_2^{-1}\tilde{\boldsymbol{P}}_{12}^T \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{t}_1 \\ \Delta t_2 \\ \Delta\boldsymbol{t}_3 \end{bmatrix} \\
&\quad = \begin{bmatrix} \tilde{\boldsymbol{b}}_1 \\ \tilde{b}_2 \\ \tilde{\boldsymbol{b}}_3 - \tilde{\boldsymbol{P}}_{12}\tilde{\boldsymbol{P}}_2^{-1}\tilde{\boldsymbol{b}}_4 \end{bmatrix}, \qquad (56)
\end{aligned}$$

where

$$\begin{aligned}
&\begin{bmatrix} \tilde{\boldsymbol{C}} & \tilde{\boldsymbol{y}} & \tilde{\boldsymbol{G}} \\ \tilde{\boldsymbol{y}}^T & \tilde{c} & \tilde{\boldsymbol{r}}^T \\ \tilde{\boldsymbol{G}}^T & \tilde{\boldsymbol{r}} & \tilde{\boldsymbol{R}} - \tilde{\boldsymbol{P}}_{12}\tilde{\boldsymbol{P}}_2^{-1}\tilde{\boldsymbol{P}}_{12}^T \end{bmatrix} \\
&\quad = \boldsymbol{V} \in \Re^{(l+1+r)\times(l+1+r)}.
\end{aligned}$$

Note that $\tilde{\boldsymbol{C}} \in \Re^{l\times l}$ is a diagonal matrix. We apply the following technique in Cholesky factorization method to decompose the matrix $\boldsymbol{V}$.

$$G'_{ii} = V_{ii}^{1/2}, \quad i = 1, \ldots, l, \qquad (57)$$

$$G'_{ji} = 0, \quad i = 1, \ldots, l, \quad j = i+1, \ldots, l, \qquad (58)$$

$$G'_{ji} = \frac{V_{ij}}{G'_{ii}}, \quad i = 1, \ldots, l, \quad j = l+1, \ldots, l+1+r, \tag{59}$$

$$G'_{ii} = \left(V_{ii} - \sum_{k=idx(i,1)}^{i-1} G'^{2}_{ik} \right)^{1/2}, \quad i = l+1, \ldots, l+1+r, \tag{60}$$

$$G'_{ji} = \frac{1}{G'_{ii}} \left(V_{ij} - \sum_{k=idx(i,1)}^{i-1} G'_{ik} G'_{jk} \right), \quad i = l+1, \ldots, l+1+r, \; j = i+1, \ldots, l+1+r, \tag{61}$$

where $idx(i, 1)$ is the first nonzero element of the i-th row. This procedure requires $O(lr^2)$ computations to compute Cholesky factorization for $\boldsymbol{V}$, which is very small compared to that in (49) (in a typical IPM iteration). From the above solution, we get $\Delta \boldsymbol{t}_4$, and consequently $\Delta \boldsymbol{s}$ and $\Delta \boldsymbol{x}$. Here, we will show some matrices calculations which are done analytically. For example,

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{y}^T & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{G}^T & \boldsymbol{0} & \boldsymbol{\epsilon}_1 \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\epsilon}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1 + \boldsymbol{x}_2 \\ \boldsymbol{y}^T \boldsymbol{x}_1 \\ \boldsymbol{G}^T \boldsymbol{x}_1 + \boldsymbol{\epsilon}_1 \boldsymbol{x}_3 \\ \boldsymbol{\epsilon}_2 \boldsymbol{x}_3 \end{bmatrix}$$

$$\boldsymbol{A}^T \boldsymbol{t} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{y} & \boldsymbol{G} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{\epsilon}_1^T & \boldsymbol{\epsilon}_2^T \end{bmatrix} \begin{bmatrix} \boldsymbol{t}_1 \\ t_2 \\ \boldsymbol{t}_3 \\ \boldsymbol{t}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{t}_1 + t_2 \boldsymbol{y} + \boldsymbol{G}\boldsymbol{t}_3 \\ \boldsymbol{t}_1 \\ \boldsymbol{\epsilon}_1^T \boldsymbol{t}_3 + \boldsymbol{\epsilon}_2^T \boldsymbol{t}_4 \end{bmatrix}$$

$$\begin{aligned} \boldsymbol{\epsilon}_1 \boldsymbol{H} \boldsymbol{\epsilon}_1^T &= diag(c_3, f_3, \ldots, i_3) \\ \boldsymbol{\epsilon}_2 \boldsymbol{H} \boldsymbol{\epsilon}_1^T = \boldsymbol{\epsilon}_1 \boldsymbol{H} \boldsymbol{\epsilon}_2^T &= diag((-a_3 + b_3), \\ &\quad (-d_3 + e_3), \ldots, (-g_3 + h_3)) \\ \boldsymbol{\epsilon}_2 \boldsymbol{H} \boldsymbol{\epsilon}_2^T &= diag((a_1 - 2a_2 + b_2), \\ &\quad (d_1 - 2d_2 + e_2), \ldots, \\ &\quad (g_1 - 2g_2 + h_2)) \\ \boldsymbol{\epsilon}_1 \boldsymbol{x}_3 &= \left[-x_{3(2)}; -x_{3(5)}; \ldots; -x_{3(3r-1)} \right] \\ \boldsymbol{\epsilon}_2 \boldsymbol{x}_3 &= \left[x_{3(0)} - x_{3(1)}; x_{3(3)} - x_{3(4)}; \ldots; \right. \\ &\quad \left. x_{3(3r-3)} - x_{3(3r-2)} \right] \end{aligned}$$

$$\begin{aligned} \boldsymbol{\epsilon}_1^T \boldsymbol{t}_3 &= \left[0; 0; -t_{3(0)}; 0; 0; -t_{3(1)}; \cdots; 0; 0; \right. \\ &\quad \left. -t_{3(r-1)} \right] \\ \boldsymbol{\epsilon}_2^T \boldsymbol{t}_4 &= \left[t_{4(0)}; -t_{4(0)}; 0; t_{4(1)}; -t_{4(1)}; 0; \cdots; \right. \\ &\quad \left. t_{4(r-1)}; -t_{4(r-1)}; 0 \right] \end{aligned}$$

Here, we see that only $O(lr)$ computations need for calculating $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{A}^T \boldsymbol{t}$. All computations including block matrix splitting technique and analytic matrix multiplications make the proposed solver much faster than general-purpose solvers.

5. Kernel Matrices Characteristics

In this section we discuss two kernel matrices: the radial basis function (RBF) kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / \gamma} \tag{62}$$

and the linear kernel, $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$.

5.1. *Radial Basis Function (RBF) Kernel Matrix*

When $\gamma \to 0$, the RBF kernel matrix tends to the identity matrix. Then it becomes a full rank matrix. When $\gamma \to \infty$, the kernel matrix tends to a matrix with all elements to be one. Then, it becomes a rank-one matrix. From these simple statements we can say that when the γ is large enough, many eigenvalues of the kernel matrix become smaller than the predefined threshold value, i.e., the dimension of the decomposed matrix is reduced. In the existing methods (e.g., LOQO, SMO), learning time is controlled by C because γ has almost no effect to the learning time. The learning time can be controlled by both parameters γ and C applying the proposed methods.

5.2. *Linear Kernel Matrix*

We can write the kernel matrix, $\boldsymbol{K}$, as follows:

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T$$

where $\boldsymbol{X}^T = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_l]$ and $\boldsymbol{x}_i \in \Re^n$ are input data. Since the linear kernel matrix is positive semidefinite, for all non-zero $\boldsymbol{v} \in \Re^l$

$$\begin{aligned} \boldsymbol{v}^T \boldsymbol{K} \boldsymbol{v} &= \boldsymbol{v}^T \boldsymbol{X} \boldsymbol{X}^T \boldsymbol{v} \\ &= \sum_{i=1}^{l} v_i \boldsymbol{x}_i \sum_{i=1}^{l} v_i \boldsymbol{x}_i \geq 0 \end{aligned}$$

Table 1. Features of benchmark data sets.

Problems	#training data	#testing data	#attributes	(γ, C)
titanic	150	2051	3	(2.00, 10.0)
banana	400	4900	2	(1.00, 316.2)
diabetes	468	300	8	(20.00, 10.0)
iris	100	0	4	(−, 10.0)
wine	130	0	13	(−, 10.0)

Usually the number of training data is larger than the dimension of the input vector. Without loss of generality we assume that $\boldsymbol{x}_i \neq \boldsymbol{x}_j$. As the training data size is larger than the dimension of the input vector, even though $\boldsymbol{x}_i \neq \boldsymbol{x}_j$, many $\boldsymbol{x}_i$'s will be linearly depended on other $\boldsymbol{x}_j$'s. Thus, the dimension of decomposed matrix is greatly reduced when the linear kernel is applied for the SVM learning.

6. Performance of SOCP, Method in [1] and QP Solver LOQO for SVM Problems

In this section, we compare the performance of SOCP methods against the QP solver LOQO and the method in [1]. Data sets[2] *titanic*, *banana*, *diabetes*, *iris*, *wine*, *adult5*, and *web4* are used for experiments. The first three problems are tested using the RBF kernel (62), and the other problems are tested using the linear kernel. Features of data sets, and the kernel and optimal parameters[3] are given in Tables 1 and 5. All experiments are done on a 750 MHz Pentium-III with 256 MB RAM computer, using the gcc compiler for the proposed method and LOQO, and g77 fortran compiler for the method in [1].

Experimental results show that objective values, numbers of support vectors, and generalization error rates of all SOCP methods, the OP solver LOQO and the method in [1] are same for all problems. The SOCP methods also follow SVM's KKT conditions. As the SOCP-Formulation 2 with Cholesky factorization (for matrix $\boldsymbol{Q}$) is faster than the other three SOCP methods, we show only the results of SOCP-Formulation 2 with Cholesky factorization throughout the paper. Table 2 shows the experimental results of SOCP-Formulation 2 with Cholesky factorization, method in [1], and LOQO for the first three problems. Learning methods use the training sets without chunking. Our resulting times include the kernel matrix factorization time with the actual learning time of the SOCP solver. According to the experimental results, the SOCP-Formulation 2 is 9 times faster than the LOQO for *titanic* problem where the dimension of the decomposed matrix is smaller than that of the kernel matrix, and LOQO is about twice as fast as SOCP-Formulation 2 for *diabetes* problem where the kernel matrix is a full rank matrix. The number of columns of the decomposed matrix (the rank of the kernel matrix) is the same in the proposed method and in the method in [1] for each problem. Experimental results show that the proposed method is faster than the method in [1] for both low-rank and full-rank kernel matrices. The method in [1] is better than LOQO only for *titanic* problem which has very low-rank kernel matrix. The performance of the method in [1] cannot be better than the QP solver LOQO if the rank of the kernel matrix is not very low. The method in [1] is very slower than LOQO for *diabetes* problem which has full-rank kernel matrix.

For *banana* problem, we see that the number of columns of the decomposed matrix is 122 by using the

Table 2. Comparison of SOCP-Formulation 2 with Cholesky factorization, QP solver LOQO and the method in [1].

		SOCP-Formulation 2	Method [1]	LOQO	SOCP-Formulation 2, Method [1] & LOQO		
Problems	# cols. (% cols. reduction)	Learning time (in seconds)	Learning time (in seconds)	Learning time (in seconds)	Obj.	SV (BSV)	%err.
titanic	12 (92%)	**0.01**	0.02	0.09	−537.26	143 (28)	21.75
banana	140 (65%)	**3.31**	9.47	8.18	−19587.67	87 (52)	11.88
diabetes	468 (0%)	29.18	101.39	**12.53**	−1833.14	239 (174)	23.33

"# cols." represents the number of columns of the decomposed matrix, "Obj." represents the objective value, "SV" represents the number of support vectors while "BSV" represents the bounded support vectors, and "%err." represents the percentage of generalization error respectively. The kernel and optimal parameters are given in Table 1.

Table 3. Comparison of the SOCP-Formulation 2 with Cholesky factorization, LOQO and the method in [1] using the new set of kernel and optimal parameters.

		SOCP-Formulation 2	Method [1]	LOQO	SOCP-Formulation 2, Method [1] & LOQO		
Problems	# cols. (% cols. reduction)	Learning time (in seconds)	Learning time (in seconds)	Learning time (in seconds)	SV (BSV)	% err.	(C, γ)
banana	97 (75%)	**1.24**	4.28	8.41	89 (63)	11.31	(350, 2.0)
diabetes	205 (56%)	**7.87**	26.25	14.49	244 (226)	23.33	(100, 320)

eigenvalue factorization method while 140 by using Cholesky factorization method. Although the number of columns is a little larger by Cholesky factorization method than that of eigenvalue factorization method, the SOCP-Formulation 2 with Cholesky factorization gives faster results than that with eigenvalue factorization (results are not shown here).

Next, we apply cross-validation to get optimal values of γ and C for *banana* and *diabetes* problems where the optimization problems are solved by the SOCP-Formulation 2. Results are shown in Table 3. The proposed method is faster than the LOQO for *diabetes* problem, and much faster than the previous result for *banana* problem with the new parameters. Usually, the proposed method is also faster than the method in [1]. From the result, we see that kernel parameter can also help to improve the learning time of the proposed method. Since the kernel parameter has no effect in the LOQO's learning time, it's time is almost the same as the previous parameter setting gives (learning time is a little bit larger for *diabetes* problem as the optimal parameter C is larger than the previous value). Results for linear kernel on *iris*, *wine*, *adult5*, and *web4* problems are shown in Table 4. The optimal parameter C is equal to 1 for *adult5* problem, and 5 for *web4* problem. In general, the ranks of linear kernel matrices are very small. Thus, the SOCP-Formulation 2 is much faster than the LOQO for linear kernel. The proposed method is also much faster than the method in [1]. In [1], Fine and Scheinberg have shown (by experiment) that their method (without chunking) performs better than either Joachims's SVM$^{\text{light}}$ or Platt's SMO if the kernel matrix has a low rank compared to the size of the training data or if it can be approximated by a low-rank positive semidefinite matrix. Since the proposed method is faster than the method in [1], the proposed method will be more efficient.

7. Selection of Good Working Set for Decomposition (Chunking) Method

The optimization problem of the SVM is challenging when the size of the training data set is large enough, which is often the case of practical applications. To tackle this problem, the SVM optimization problem is decomposed into a series of smaller optimization problems, and to solve the small optimization problems iteratively until the global solution is found, which is briefly described in Section 1. The convergence of decomposition method depends on the selection of training data for smaller problems, which is so-called

Table 4. Comparison of SOCP-Formulation 2 with Cholesky factorization, LOQO and the method in [1] for linear kernel.

		SOCP-Formulation 2	Method [1]	LOQO
Problems	# cols. (% cols. reduction)	Learning time (in seconds)	Learning time (in seconds)	Learning time (in seconds)
iris	4 (96%)	**0.00**	0.01	0.03
wine	13 (90%)	**0.01**	0.04	0.07
adult5	107 (98%)	**60.43**	198.55	N/A
web4	256 (96.5%)	**520.17**	1515.01	N/A

Iris and *wine* are three-class problems. We use two-class out of three-class for these problems. Here "N/A" represents that the corresponding method is infeasible or too slow.

'working set'. Thus, working set selection has the same importance as improving the performance of optimizing solver for the SVM problem. Recently, Hsu and Lin [8] have proposed a working set selection approach for SVM formulation which needs a smaller number of iterations among the existing approaches, and the approach can be applied with a large working set. The algorithm [8] is given below:

- Let r be the number of free variables at kth iteration.
- Let $q = q_1 + q_2$, where $q_1 \leq r$ and q_2 is an even number.
- For those $0 < \alpha_i^{(k)} < C$, select q_1 elements with the smallest $|\nabla f(\alpha_i^{(k)}) + by_i|$, $i = 1 \ldots l$.
- Select q_2 elements from the rest of the index set by SVM^{light}'s working set selection approach.

In this paper we modify Hsu and Lin's approach. A simple modification to Hsu and Lin's approach leads to converge faster with a large working set than Hsu and Lin's approach. As SVM^{light}'s working set is included in their approach, we first briefly describe SVM^{light}'s working set selection strategy. Joachims [5] has proposed a working set selection strategy used in his software SVM^{light} based on Zoutendijk's method. The following optimization problem [5] is solved to select the working set:

$$\begin{aligned} \min \quad & \nabla f(\boldsymbol{\alpha}^{(k)})^T \boldsymbol{d} \qquad (63)\\ \text{subject to} \quad & \boldsymbol{y}^T \boldsymbol{d} = 0, -\mathbf{1} \leq \boldsymbol{d} \leq \mathbf{1},\\ & d_i \geq 0, \quad \text{if } \alpha_i^{(k)} = 0,\\ & d_i \leq 0, \quad \text{if } \alpha_i^{(k)} = C,\\ & |\{d_i | d_i \neq 0\}| \leq q, \quad i = 1 \ldots l, \end{aligned}$$

where $f(\boldsymbol{\alpha}) \equiv \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}$, $\boldsymbol{\alpha}^{(k)}$ is the solution at the kth iteration, $\nabla f(\boldsymbol{\alpha}^{(k)})$ is the gradient of $f(\boldsymbol{\alpha})$ at $\boldsymbol{\alpha}^{(k)}$. The components of $\boldsymbol{\alpha}^{(k)}$ with non-zero d_i's are included in the working set B. This optimization problem is solved using a simple strategy: (1) sort all $\alpha_i^{(k)}$ according to $y_i \nabla f(\alpha_i^{(k)})$ in descending order for $i = 1 \ldots l$; 2) successively pick the $q/2$ elements from the top of the list for which $0 < \alpha_i^{(k)} < C$, or $\alpha_i^{(k)} = 0$ if $y_i = -1$ or $\alpha_i^{(k)} = C$ if $y_i = 1$, and set $d_i = -y_i$; and 3) similarly pick the $q/2$ elements from the bottom of the list for which $0 < \alpha_i^{(k)} < C$, or $\alpha_i^{(k)} = 0$ if $y_i = 1$ or $\alpha_i^{(k)} = C$ if $y_i = -1$, and set $d_i = y_i$. Here $\boldsymbol{d}$ is used only for selecting the working set. The value of q must be an even number. This algorithm takes at most $O(lq)$ operations that is acceptable for practical application. If we consider that $\bar{\boldsymbol{\alpha}}$ is the optimal solution of (4), then the following KKT conditions hold:

$$\nabla f(\bar{\alpha}_i) + by_i > 0, \quad \text{if } \bar{\alpha}_i = 0, \qquad (64)$$

$$\nabla f(\bar{\alpha}_i) + by_i < 0, \quad \text{if } \bar{\alpha}_i = C, \qquad (65)$$

$$\nabla f(\bar{\alpha}_i) + by_i = 0, \quad \text{if } 0 < \bar{\alpha}_i < C. \qquad (66)$$

Now, we will show that Joachims's approach selects the most violated bounded variables from each class. We first consider for the case of $\alpha_i^{(k)} = 0$ which violates the KKT condition at the kth iteration. We can write $\nabla f(\alpha_i^{(k)}) + by_i \leq 0$ if $\alpha_i^{(k)} = 0$ violates the KKT condition. Thus, $y_i \nabla f(\alpha_i^{(k)}) + b \geq 0$ if $y_i = -1$ or $y_i \nabla f(\alpha_i^{(k)}) + b \leq 0$ if $y_i = 1$. Joachims's approach selects $\alpha_i^{(k)} = 0$ from the top of the sorted list of $\boldsymbol{\alpha}^{(k)}$ if $y_i = -1$ or bottom of the list if $y_i = 1$ where $\alpha_i^{(k)}$ is sorted according to $y_i \nabla f(\alpha_i^{(k)})$ in descending order. Note that the value of b is the same for all variables at any iteration, however b is not a constant and must be recalculated in each iteration. When $\alpha_i^{(k)} = 0$ is selected from the top of the list, $\alpha_i^{(k)}$ is the most violated variable from the class label -1. When $\alpha_i^{(k)} = 0$ is selected from the bottom of the list, it is the most violated variable from the class label 1. Similarly we can show that if $\alpha_i^{(k)} = C$ are selected, they are also the most violated variables from either class. The free variables $0 < \alpha_i^{(k)} < C$ are selected without any constraint, i.e., the selection of free variables $0 < \alpha_i^{(k)} < C$ depends only on ordered $y_i \nabla f(\alpha_i^{(k)})$ values in the list. Thus, all selected $0 < \alpha_i^{(k)} < C$ may not be the most KKT violated free variables from each class. This approach performs well for small-sized working set selection [5, 8].

Hsu and Lin's approach adds some free variables whose corresponding $|\nabla f(\alpha_i^{(k)}) + by_i|$ are the smallest with Joachims's working set. It can be seen from (64)–(66) that the free variables which slightly violated KKT condition are added with Joachims's working set. For any free variable $\alpha_i^{(k)}$ if $\nabla f(\alpha_i^{(k)}) + by_i < 0$, the free variable $\alpha_i^{(k)}$ shows the characteristics as upper bounded variable, $\alpha_i^{(k)} = C$. Similarly, for free variable $\alpha_i^{(k)}$ if $\nabla f(\alpha_i^{(k)}) + by_i > 0$, the free variable $\alpha_i^{(k)}$ shows the characteristics as lower bounded variable, $\alpha_i^{(k)} = 0$. Note that variables either upper or lower bounded corresponding to the smallest $|\nabla f(\alpha_i^{(k)}) + by_i|$ are near to the hyperplane than the other (either upper or lower bounded) variables. When the free variables with the smallest $|\nabla f(\alpha_i^{(k)}) + by_i|$ are selected for the next iteration, some of them will be either upper bounded or lower bounded variables, and it keeps the number of

free variables to be small. This is because some variables near to the (current) hyperplane from both classes have possibility to select for the next iteration (and more possibility to be upper bounded variable than to be 0 in the next iteration because input vectors corresponding those free variables are closer to each other). If a component is correctly identified at C or 0, there is no problem of numerical accuracy. Since the decomposition method cannot consider all variables together in each iteration, it is difficult to decide the value of a free variable. Thus, a larger number of free variables in the working set causes more difficulty. Hsu and Lin's approach leads decomposition method to converge fast as it always tries to keep the number of free variables small. In addition, after the subproblem (5) is solved, the free variables at the current working set satisfy $\nabla f(\alpha_i^{(k)}) + by_i = 0$. Thus some free variables in the current working set may again include in the next set. It reduces zigzagging of the series of working set elements, and hyperplanes may not oscillate (jump) more through the whole training set. For these reasons, Hsu and Lin's approach may lead to converse faster than Joachims's approach. Again, an approach with a relatively large sized working set might reduce the overall computation cost. Thus, overall performance of Hsu and Lin's approach is better than Joachims's approach.

From the above description it is clear to understand that if the successive hyperplanes do not oscillate more, some free variables which are far from the hyperplanes, i.e., free variables corresponding $(\nabla f(\alpha_i^{(k)}) + by_i) \gg 0$ may be correctly identified at $\alpha_i^{(k+1)} = 0$. With the working set proposed by Hsu and Lin, we add some free variables with the largest $\nabla f(\alpha_i^{(k)}) + by_i$, and the proposed algorithm is as follows:

- Let r be the number of free variables at kth iteration.
- Let $q = q_1 + q_2 + q_3$, where $(q_1 + q_2) \leq r$ and q_3 is an even number.
- For those $0 < \alpha_i^{(k)} < C$, select q_1 elements with the smallest $|\nabla f(\alpha_i^{(k)}) + by_i|$, $i = 1 \cdots l$.
- Select q_2 elements with the largest $\nabla f(\alpha_i^{(k)}) + by_i$, $i = 1 \cdots l$ from $r - q_1$ variables.
- Select q_3 elements from the rest of the index set by the SVM^{light}'s working set selection approach.

To speed up the convergence, we should carefully choose q_1, q_2 and q_3. In the previous explanation, we have seen that Joachims's approach has a nice property to select (q_3) variables from both classes. However, the other selection strategies (for q_1 and q_2) have no such criteria. Hsu and Lin's approach adds q_1 elements with q_3 elements where q_1 elements are selected with the smallest $|\nabla f(\alpha_i^{(k)}) + by_i|$. Although q_1 variables cannot decrease the objective function much, Hsu and Lin's approach tries to push free variables to be bounded variables, and keeps the number of free variables to be small. If many free variables in the iterative process are still free in the final solution, the learning method with the large value of q_1 and small value of q_3 converses slowly because it may wrongly put many free variables as bounded variables. There may arise another problem when elements from one class are much larger than those of the other class in the working set. In this case, many variables may be wrongly put as bounded variables, or subproblem (5) may be a problem with separable data. If subproblem (5) becomes a problem with separable data, i.e., $\xi_i = 0, i \in B, (\alpha_B)_i$'s are in general not equal to C. Then, it is difficult to identify correct bounded variables and the convergence becomes slow. However, SVM^{light}'s working set selection approach has very good criteria to increase much progress towards the minimum of $f(\alpha)$ with all types of variables (free and bounded variables). Considering all above conditions, a good large working set should follow a sufficiently large value of q_3, and then a large value of q_1 ($q_1 < q_3$). The free variables corresponding to $(\nabla f(\alpha_i^{(k)}) + by_i) \gg 0$ are the most KKT violated free variables on the right side of the current hyperplane. Joachims's approach selects q_3 variables which are the most violated KKT free ($0 < \alpha_i < C$) and bounded ($\alpha_i = 0$ and $\alpha_i = C$) variables from the top and bottom of the sorted list of $y_i \nabla f(\alpha_i^{(k)}), i = 1, \ldots, l$. If $y_i \nabla f(\alpha_i^{(k)})$ of free variables are smaller than those of q_3 bounded variables ($\alpha_i = 0$ and $\alpha_i = C$), then no free variable will be selected in the next iteration. The proposed method guarantees to select the most KKT violated free variables. The value of q_2 will be small enough.

8. Experimental Results

In this section, we show the comparison among the decomposition method including SOCP-Formulation 2 and LOQO, and Platt's SMO. Cholesky factorization method is applied to decompose kernel matrices for the SOCP-Formulation 2. In the first experiment we use Hsu and Lin's working set selection approach for the SOCP-Formulation 2 and LOQO. In Hsu and Lin's working set selection approach, we set $q = 10$ where $q_1 = 6$ and $q_2 = 4$. The size of working set of

Table 5. Features of new benchmark data sets.

Problems	#training data	#testing data	#attributes	(γ, C)
thyroid	140	75	5	(3.00, 10.0)
heart	170	100	13	(120.00, 3.162)
splice	1000	2175	60	(70.00, 1000.0)
adult1	1605	29589	123	(200.00, 1.0)
adult2	2265			
adult3	3185			
adult4	4781			
adult5	6414			
web1	2477	38994	300	(200.00, 5.0)
web2	3470			
web3	4912			
web4	7366			

the SMO is restricted to be two. The gcc compiler is used for computing experimental results of the SOCP-Formulation 2 and LOQO, and matlab with mex interface is applied to implement the SMO algorithm. For this experiment, we add two more problems, *adult* and *web*, with *titanic*, *banana* and *diabetes*. The *adult* and *web* problems have various sizes training sets. Features of these problems are shown in Table 5. Table 6 presents the results for the RBF kernel (62). The (γ, C) for *adult* and *web* are shown in Table 5 (best parameter sets for these problems given in [5]). The kernel and optimal parameters for the other problems are given in Table 1. From Table 6, we see that the learning times of SOCP-Formulation 2 are less than or equal to that of LOQO. This is because, for a very small size problem (which is usually used in the chunking method), although the rank of the kernel matrix is full (or almost so), the SOCP-Formulation 2 is competitive with the LOQO, and the decomposition method including the proposed method needs smaller number of iterations.

When perturbed KKT conditions are very tight, any optimizer (either QP or SOCP solver) finds the optimal solution with high accuracy. Using a higher accuracy leads to considerably longer training time, however, it does not show improved generalization performance. Thus, loose perturbed KKT conditions are used in practice. Note that QP solver (LOQO) and reformulating SOCP give the same optimal solution when perturbed KKT conditions are almost tight for both methods. In the decomposition (chunking) process, perturbed KKT condition parameters are heuristics, and are changed according to the complexity of working data set in the iterative process (sometimes robust or sometimes loose setting). When very loose parameters are being set in some working sets, LOQO (also SOCP solver) gives (feasible) solutions for those working data

Table 6. Comparison of SOCP-Formulation 2, LOQO and Platt's SMO. The SOCP-Formulation 2 and LOQO are applied in the SVM decomposition method, and Hsu and Lin's approach is used for working set selection where $q_1 = 6$ and $q_2 = 4$. Here 'iter.' represents the number of iterations.

	SOCP-Formulation 2				LOQO				Platt's SMO		
Problem	iter.	Time (in sec.)	SV (BSV)	%err.	iter.	Time (in sec.)	SV (BSV)	%err.	Time (in sec.)	SV (BSV)	%err.
titanic	36	0.05	71 (46)	21.75	46	0.07	72 (50)	21.75	**0.02**	79 (48)	21.75
banana	1126	**5.19**	88 (52)	11.88	2845	10.39	87 (52)	11.88	111.25	87 (52)	11.88
diabetes	515	**2.81**	239 (174)	23.33	554	**2.81**	239 (175)	23.33	3.28	239 (174)	23.33
adult1	231	**24.71**	786 (759)	17.61	232	25.04	785 (759)	17.64	32.03	786 (756)	17.61
adult2	324	**48.50**	1104 (1081)	16.9	325	49.14	1105 (1082)	16.93	70.92	1104 (1077)	16.93
adult3	450	**100.85**	1454 (1412)	16.72	450	101.45	1454 (1412)	16.74	222.71	1554 (2050)	16.72
adult4	638	**210.13**	2090 (2051)	16.54	645	216.14	2092 (2050)	16.52	231.95	2093 (2050)	16.52
adult5	841	**370.93**	2712 (2658)	16.27	843	375.07	2714 (2656)	16.27	2217.83	2715 (2658)	16.26
web1	300	**236.83**	232 (100)	2.56	305	242.02	234 (99)	2.56	304.03	250 (103)	2.56
web2	397	**441.09**	301 (157)	2.54	442	493.21	297 (158)	2.54	643.10	375 (226)	2.53
web3	506	783.14	363 (215)	2.41	557	867.68	366 (215)	2.41	**611.14**	375 (226)	2.41
web4	584	**1361.4**	511 (345)	2.13	650	1521.9	514 (347)	2.13	1946.4	513 (349)	2.13

sets but resulting variables $(\alpha_B)_i$'s do not satisfy KKT complementary conditions. In this case, iteration may increase. Thus, a good heuristic may reduce the number of iterations as well as overall computational cost. For implementing the LOQO in the decomposition method, we use the same default options of SVM^{light}. The perturbed KKT parameters of the proposed method is different from the LOQO, that's why the number of iterations varies between the proposed method and LOQO.

Platt's SMO selects the active data randomly from the KKT condition violations. Therefore, the convergence is very slow for some problems and fast for other problems, and training time varies in each simulator run (sometimes support vectors and/or error rates also vary). This is one of the disadvantages of the SMO method. In this paper we show the results for SMO on the average of 4 simulator runs. The proposed method and LOQO are faster than the SMO for most of the problems.

Next, we apply the modified Hsu and Lin's approach with the LOQO and SOCP-Formulation 2. We will show the experimental results on previous problems with some other problems such as *heart*, *thyroid*, and *splice*. They are collected from [27]. Table 7 presents results for the RBF kernel (62) using LOQO with deferent permutations of q_1, q_2, and q_3 where q is fixed for each problem. The kernel and optimal parameters for these new problems are given in [27] as shown in Table 5. We do not show the learning time in Table 7 because the learning time is proportional to the number of iterations (as the same optimizer, LOQO, is used). Although free variables with the largest $\nabla f(\alpha_i^{(k)}) + by_i$ are most KKT violated free variables which can be selected by Joachims's approach, the working set elements are different for $q = q_1 + q_2 + q_3$ and $q = q_1 + 0 + \hat{q}_3$ where $\hat{q}_3 = q_2 + q_3$, and numerical experiments show different results. From Table 7, we see that a simple modification to Hsu and Lin's approach can improve the performance of it.

Table 8 presents results of the modified Hsu and Lin's approach with SOCP-Formulation 2 and LOQO for the RBF kernel (62). Learning times as well as iterations in Table 8 are much less than that in Table 6 for all problems. It shows the importance of applying a sufficient large working set. From the results in Table 8 we see that the SOCP-Formulation 2 is better or very competitive with the LOQO. The ranks of kernel matrices are full (or almost so) in many problems. Problems which have very low-rank kernel

Table 7. Results of the modified Hsu and Lin's approach with LOQO on different permutations of q_1, q_2 and q_3 for fixed q. When $q_2 = 0$, it becomes Hsu and Lin's approach.

Problems	q_1-q_2-q_3	iter.	SV (BSV)	% err.
titanic	**10-4-14**	**9**	85 (49)	21.75
	14-0-14	25	99 (32)	21.75
	16-0-12	11	80 (45)	21.75
	10-0-18	21	94 (33)	21.75
	20-0-8	15	81 (45)	21.75
banana	**20-4-50**	**12**	87 (52)	11.88
	24-0-50	14	87 (52)	11.88
	20-0-54	15	87 (52)	11.88
	34-0-40	16	87 (52)	11.88
diabetes	**21-3-40**	**28**	239 (174)	23.33
	24-0-40	30	239 (175)	23.33
	20-0-44	29	239 (175)	23.33
	14-0-50	30	239 (174)	23.33
thyroid	**6-2-12**	**6**	19 (0)	8.00
	8-0-12	6	19 (0)	8.00
	6-0-14	8	19 (0)	8.00
	16-0-4	9	19 (0)	8.00
heart	**4-2-12**	**12**	80 (63)	19.00
	6-0-12	13	80 (63)	19.00
	4-0-14	15	80 (63)	19.00
	14-0-4	28	80 (63)	19.00
	10-0-8	16	80 (63)	19.00
splice	**6-2-24**	**166**	677 (0)	10.99
	8-0-24	178	677 (0)	11.13
	12-0-20	197	678 (0)	10.71
	16-0-16	233	676 (0)	19.77
adult2	**8-2-24**	**58**	1105 (1081)	16.93
	10-0-24	59	1105 (1081)	16.93
	18-0-16	82	1105 (1082)	16.94
adult3	**8-2-22**	**81**	1455 (1412)	16.78
	10-0-22	82	1454 (1412)	16.75
	20-0-12	142	1454 (1412)	16.78
adult4	**10-2-20**	**135**	2092 (2049)	16.56
	12-0-20	135	2092 (2049)	16.54
	20-0-12	210	2093 (2050)	16.54
adult5	**8-2-20**	**173**	2716 (2656)	16.27
	10-0-20	178	2717 (2654)	16.26
	20-0-10	328	2717 (2656)	16.27
web1	**8-2-12**	**110**	234 (100)	2.56
	10-0-12	124	240 (100)	2.56
	8-0-14	114	243 (100)	2.56
	16-0-6	176	230 (100)	2.56
web2	**8-2-10**	**185**	308 (156)	2.54
	10-0-10	199	303 (156)	2.54
	8-0-12	205	307 (155)	2.54
	16-0-4	345	298 (160)	2.54
web3	**9-1-12**	**168**	369 (214)	2.41
	10-0-12	177	368 (216)	2.41
	16-0-6	289	366 (218)	2.41
web4	**9-1-12**	**210**	518 (342)	2.13
	10-0-12	248	518 (345)	2.13
	16-0-6	335	512 (346)	2.13

Table 8. Results of the modified Hsu and Lin's approach with SOCP-Formulation 2 and LOQO.

	SOCP-Formulation 2					LOQO				
Problems	q_1-q_2-q_3	iter.	time	SV (BSV)	% err.	q_1-q_2-q_3	iter.	time	SV (BSV)	%err.
titanic	10-4-14	9	**0.03**	85 (49)	21.75	10-4-14	9	0.04	85 (49)	21.75
diabetes	20-4-40	28	1.62	239 (174)	23.33	21-3-40	28	**1.02**	239 (174)	23.33
thyroid	5-3-12	6	**0.02**	19 (0)	8.00	5-3-12	6	0.02	19 (0)	8.00
heart	4-2-12	13	0.06	80 (63)	19.00	4-2-12	12	**0.04**	80 (63)	19.00
splice	6-2-24	168	22.59	675 (0)	11.67	6-2-24	166	**21.30**	675 (0)	11.67
adult2	8-2-24	59	45.98	1105 (1081)	16.93	8-2-24	58	**45.39**	1105 (1081)	16.93
adult3	8-2-22	81	**83.23**	1455 (1413)	16.78	8-2-22	81	83.71	1455 (1412)	16.78
adult4	10-2-20	133	**192.56**	2093 (2049)	16.56	10-2-20	135	197.69	2092 (2049)	16.56
adult5	8-2-22	161	**335.64**	2717 (2656)	16.28	8-2-22	162	340.73	2717 (2654)	16.27
web1	8-2-10	118	**182.47**	236 (100)	2.56	8-2-10	124	193.74	236 (100)	2.56
web2	8-2-12	157	**344.70**	294 (155)	2.53	8-2-12	177	430.61	303 (156)	2.53
web3	9-1-12	160	**540.94**	370 (216)	2.41	9-1-12	168	574.14	369 (214)	2.41
web4	9-1-12	193	**974.84**	500 (338)	2.13	9-1-12	210	1064.9	518 (342)	2.13

matrices, the proposed SOCP method gives better results than that of the LOQO in the decomposition method.

We now discuss the method proposed in [1] for large-scale problems whose ranks of the kernel matrices are not small enough compared to the size of the training set. In *adult5* problem, the training set size is 6414 and the rank is 5710 with RBF kernel where $\sigma = 200$ (applying 'incomplete' Cholesky factorization). This problem is infeasible on a 256 MB RAM computer by the method proposed in [1] (without chunking). The problem *web4* is also infeasible because the problem has 7366 training data, and the rank of the kernel matrix is 6046 with the same kernel parameter. Moreover, as the method in [1] is slower than LOQO if the rank of the kernel matrix is not small enough compared to the size of the training set, the chunking method including the method in [1] may not show better performance than that including the LOQO for, e.g., the working set size is 5 and the rank of the kernel matrix is 4. Now we show the experimental results of our proposed method in the decomposition method for some problems whose smaller QP subproblems have full rank (or almost so) kernel matrices. Table 9 shows the comparison of the proposed method and LOQO in the decomposition method for *adult1* and *adult2* problems where the working set sizes are equal to 2, 4, 8, 12 and 20. Joachims's working set selection

Table 9. Comparison of the SOCP-Formulation 2 and LOQO in the decomposition method when the ranks of the kernel matrices of smaller problems are full or almost so. Joachims's approach is applied for the working set selection. The working set sizes are equal to 2, 4, 8, 12 and 20.

	adult1 problem				adult2 problem			
	SOCP-Formulation 2		LOQO		SOCP-Formulation 2		LOQO	
Working set size	iter.	Time (in sec.)	iter.	Time (in sec.)	iter.	Time (in sec.)	iter.	Time (in sec.)
2	505	26.65	505	26.74	694	51.61	693	51.76
4	252	24.28	252	24.36	350	47.53	350	47.72
8	139	25.53	139	25.66	192	49.77	194	50.68
12	106	28.61	106	28.79	127	48.27	131	50.04
20	60	25.87	61	26.54	79	48.51	79	48.81

approach is applied for these experiments. The number of iterations of the decomposition method is the same (or almost so) for these problems in applying both the proposed SOCP method and LOQO, thus the number of iterations does not affect the overall learning time. There is no dimension reduction at any iteration (i.e., ranks of kernel matrices are full) for the working set size which is equal to 2. The ranks are almost full for other sizes (i.e., the ranks are 3, 7 and 11 for the working set sizes which are equal to 4, 8 and 12 respectively in a very few number of iterations, and the rank is 19 for the working set size which is equal to 20 in some iterations). However, the experimental results show that the decomposition method with the proposed method is faster or very competitive than that with LOQO. Thus, the proposed method can be replaced to LOQO for a small sized working set though the ranks of kernel matrices are full but the method in [1] may not be competitive with LOQO for the full rank kernel matrix in the decomposition (chunking) method.

9. Conclusion

In this paper we propose a fast learning method for support vector machines. The computational cost of the proposed method depends on the rank of the kernel matrix. The proposed method is much faster than the QP solver LOQO when the rank of the kernel matrix is small enough. The proposed method is also faster than the method in [1] for both low-rank and full-rank kernel matrices. To solve large-scale problems, we apply the proposed method into the SVM decomposition (chunking) method. In the decomposition method, the proposed method is also better or competitive with the LOQO. The proposed method is faster than the SMO for most of the problems where the SMO results are the average of 4 simulator runs. We also modify Hsu and Lin's working set selection approach which leads to converge fast. As we need to solve many difficult problems, more optimization knowledge and techniques should be considered. Future work will be improving the SOCP method using some advanced techniques, and incorporating Joachims's 'shrinking' technique in the chunking method.

Appendix A: Proof of Lemma 3.1

Since $\boldsymbol{s}_1 \geq \boldsymbol{0}$, thus $\boldsymbol{t}_1 \leq \boldsymbol{0}$ from (17). As $s_2^2 \geq s_3^2 + \|\boldsymbol{s}_4\|^2$ and $\boldsymbol{s}_4 = \boldsymbol{t}_2$ from (20) and (22) respectively, we can write,

$$\left(\frac{1}{2} - t_3\right)^2 \geq \left(\frac{1}{2} + t_3\right)^2 + \|\boldsymbol{t}_2\|^2,$$
$$-2t_3 \geq \|\boldsymbol{t}_2\|^2.$$

Since $\boldsymbol{s}_0 \geq \boldsymbol{0}$, it follows from (16) that

$$-t_0\boldsymbol{y} - \boldsymbol{G}\boldsymbol{t}_2 \geq \boldsymbol{e} + \boldsymbol{t}_1.$$

Let

$$-t_0 = f_0, \quad -\boldsymbol{t}_1 = \boldsymbol{f}_1, \quad -\boldsymbol{t}_2 = \boldsymbol{f}_2, \quad \text{and} \quad -2t_3 = f_3.$$

Then, Eqs. (15)–(22) can be written as

$$\begin{aligned} \min \quad & C\boldsymbol{e}^T\boldsymbol{f}_1 + \frac{1}{2}f_3 \\ \text{subject to} \quad & f_0\boldsymbol{y} + \boldsymbol{G}\boldsymbol{f}_2 \geq \boldsymbol{e} - \boldsymbol{f}_1 \\ & \boldsymbol{f}_1 \geq \boldsymbol{0}, \\ & f_3 \geq \|\boldsymbol{f}_2\|^2. \end{aligned} \tag{67}$$

If we let

$$\boldsymbol{G}^T = [y_1\boldsymbol{\phi}(\boldsymbol{x}_1), y_2\boldsymbol{\phi}(\boldsymbol{x}_2), \ldots, y_l\boldsymbol{\phi}(\boldsymbol{x}_l)],$$

then $\boldsymbol{G}\boldsymbol{G}^T = \boldsymbol{Q}$ holds, and

$$(\boldsymbol{G}\boldsymbol{w})^T = [y_1\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_1), y_2\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_2), \ldots, y_l\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_l)].$$

Letting

$$f_0 = b, \quad \boldsymbol{f}_1 = \boldsymbol{\xi}, \quad \boldsymbol{f}_2 = \boldsymbol{w} \quad \text{and } f_3 = \theta,$$

problem (67) becomes

$$\min \quad \frac{1}{2}\theta + C\sum_{i=1}^{l}\xi_i \tag{68}$$

$$\text{subject to} \quad y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \ \ i = 1, \ldots, l, \tag{69}$$

$$\xi_i \geq 0, \ \ i = 1, \ldots, l, \tag{70}$$

$$\theta \geq \|\boldsymbol{w}\|^2. \tag{71}$$

The problem (68–71) is the same as the problem (1–3) and $b = -t_0$.

Acknowledgments

The authors would like to thanks Katya Scheinberg, IBM T. J. Watson Research Center, NY, for supplying their source code and helpful comments.

Notes

1. LOQO solver is used in Joachims's software SVM^{light} [5], implemented by A. J. Smola.
2. The first three data sets are collected from [27]. The remaining two data sets are collected from UCI Repository of machine learning databases [26].
3. For the first three data sets, we use the same parameter sets (γ, C) given in [27]. For *diabetes* problem the optimal value C is not given in [27]. We have chosen this value by preliminary experiments. For *iris* and *wine* problems, we have chosen optimal parameters by cross-validation.

References

1. S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representations," *Journal of Machine Learning Research*, vol. 2, pp. 243–264, 2001.
2. C. Campbell and N. Cristianini, "Simple learning algorithms for training support vector machine," Technical report, University of Bristol, 1998.
3. V.N. Vapnik, *Stasistical Learning Theory*, Wiley: New York, 1998.
4. R.J. Vanderbei, "Loqo: An interior point code for quadratic programming," Tecnical report SOR 94-15, Princeton University, 1994.
5. T. Joachims, "Making large-scale support vector machine learning practical," in *Advanvced in Kernel Methods: Support Vector Machine*, edited by B. Schölkopf, C. Burges, and A. Smola, MIT Press: Cambridge, MA, 1998, pp. 169–184.
6. E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Proc. of IEEE'97*, FL, 1997.
7. J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advanced in Kernel Methods: Support Vector Machine*, edited by B. Schölkopf, C. Burges, and A. Smola, MIT Press: Cambridge, MA, 1998, pp. 185–208.
8. C.-W. Hsu and C.-J. Lin, "A simple decomposition method for support vector machines," *Machine Learning*, vol. 46, pp. 291–314, 2002.
9. P. Laskov, "An improved decomposition algorithm for regression support vector machines," *Machine Learning*, vol. 46, pp. 315–350, 2002.
10. S.S. Kertee, S. Shevade, C. Bhattacharyya, and K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.
11. C.-C. Chang and C.-J. Lin, "Training ν-support vector cclassifiers: Theory and algorithm," *Neural Computation*, vol. 13, no. 9, pp. 2119–2147, 2001.
12. R. Collobert and S. Bengio, "SVMTorch: A support vector machine for large-scale regression and classification problems," *Journal of Machine Learning Research*, vol. 1, pp. 143–160, 2001. Available at http://www.idiap.ch/learning/SVMTorch.html
13. C.-C. Chang and C.-J. Lin, "LIMSVM: A library for support vector machines," 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
14. C.-J. Lin, "On the convergence of the decomposition method for support vector machines," *IEEE Trans. Neural Network*, vol. 12, pp. 1288–1298, 2001.
15. G.R.G. Lanckriet, N. Cristianini, P.L. Bartlett, L El Ghaoui, and M.I. Jordan, "Learning the kernel matrix with semidefinite programming," *Journal of Machine Learning Research*, vol. 5, pp. 27–72, 2004.
16. R.D.C. Monterio and T. Tsuchiya, "Polynomial convergence of primal-dual algorithms for the second-order cone programming based on the MZ-family of directions," *Math. Program.*, vol. 88, pp. 61–83, 2000.
17. A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* MPS-SIAM Series on Optimization: Philadelphia, 2001.
18. M. Muramatsu, "On a commutative class of search directions for linear programming over symmetric cones," *Journal of Optimization Theory and Applications*, vol. 112, no. 3, pp. 595–625, 2002.
19. R. Debnath, M. Muramatsu, and H. Takahashi, "The support vector machine learning using second order cone programming," in *Proc. IEEE Int. Joint Conference on Neural Networks*, Budapest, Hungary, 25–29 July, 2004, pp. 2991–2996.
20. R.D.C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," *SIAM Journal on Optimization*, vol. 7, pp. 663–678, 1997.
21. C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization*, vol. 6, pp. 342–361, 1996.
22. M. Kojima, S. Shindoh, and S. Hara, "Interior-point methods for the monotone linear complementary problem in symmetric matrices," *SIAM Journal on Optimization*, vol. 7, pp. 86–125, 1997.
23. E.D. Andersen, C. Roos, and T. Terlaky, "On implementing a primal-dual interior-point method for conic quadratic optimization," *Math. Programming Ser. B*, vol. 95, pp. 249–277, 2003.
24. Z. Cai, K.-C. Toh, "Solving second order cone programming via the augmented systems". [online] Available: http://www.optimization-online.org/DB_HTML/2002/08/517.html
25. S. Mehrotra, "On implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.
26. C.L. Blake and C.J. Merz, "UCI repository of machine learning databases," Univ. California, Dept. Inform. Comp. Sc., Irvine, CA 1998. [online] Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.
27. G. Rätsch, Benchmark data sets. Available at http://www.first.gmd.de/~raetsch/data/benchmarks.htm.
28. R. Debnath and H. Takahashi, "An improved working set selection method for SVM decomposition method," in *Proc. IEEE Int. Conference Intelligence Systems*, Varna, Bulgaria, 21–24, 2004, pp. 520–523.

29. C. Saunders, M.O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola, "Support vector machine reference manual," Technical Report CSD-TR-98-03, Royal Holloway, University of London, Egham, UK, 1998.
30. T. Joachims, Department of Computer Science, Cornell University, personal communication, 2003.
31. W. Bress, W. Vetterling, S. Teukolsky, and B. Slannery, *Numerical Receipes in C (The Art of Scientific Computing)*, 2nd ed. Cambridge University Press, 1992.
32. G.H. Golub, C.F.V. Loan, *Matrix Computations*, 2nd ed. Johns Hopkins University Press, 1989.
33. M.S. Bazaraa, C.M. Shetty, *Nonlinear Programming: Theory and Algorithms*, Wiley: New York, 1979.
34. J. Werner, *Optimization-Theory and Applications*, Vieweg, 1984.

Rameswar Debnath is a Ph.D candidate at the University of Electro-Communications, Tokyo, Japan and also a lecturer of the Computer Science & Engineering Discipline at Khulna University, Bangladesh. He received the bachelor's degree in computer science and engineering from Khulna University in 1997 and masters of engineering degree in communication and systems from the University of Electro-Communications in 2002. His research interests include support vector machines, artificial neural networks, pattern recognition, and image processing.

Masakazu Muramatsu is an associate professor of the Department of Computer Science at the University of Electro-Communications, Japan. He received a bachelor's degree from the University of Tokyo in 1989, master's degree in engineering from University of Tokyo in 1991, and Ph.D from the Graduate University for Advanced Studies in 1994. He was an assistant professor of the Department of Mechanical Engineering at Sophia University from 1994 to 2000, when he moved to the current university. His research interests include mathematical programming, second-order cone programming and its application to machine learning.

Haruhisa Takahashi was born in Shizuoka Prefecture Japan, on March 31, 1952. He graduated from the University of Electro-Communications. He received the Dr Eng. degree from Osaka University. He was a faculty member of the Department of Computer Science and Engineering at Toyohashi University of Technology from 1980 to 1986. Since 1986, he has been with the University of Electro-Communications where he is currently professor of the Department of Information and Communication Engineering. He was previously engaged in the fields of nonlinear network theory, queueing theory and performance evaluation of communication systems. His current research includes learning machines, artificial neural networks, and cognitive science.