Interior Point Methods for Massive Support Vector Machines*

Michael C. Ferris and Todd S. Munson†

May 25, 2000

Abstract

We investigate the use of interior point methods for solving quadratic programming problems with a small number of linear constraints where the quadratic term consists of a low-rank update to a positive semi-definite matrix. Several formulations of the support vector machine fit into this category. An interesting feature of these particular problems is the volume of data, which can lead to quadratic programs with between 10 and 100 million variables and a dense Q matrix. We use OOQP, an object-oriented interior point code, to solve these problem because it allows us to easily tailor the required linear algebra to the application. Our linear algebra implementation uses a proximal point modification to the underlying algorithm, and exploits the Sherman-Morrison-Woodbury formula and the Schur complement to facilitate efficient linear system solution. Since we target massive problems, the data is stored out-of-core and we overlap computation and I/O to reduce overhead. Results are reported for several linear support vector machine formulations demonstrating the reliability and scalability of the method.

Keywords: support vector machine, interior-point method, linear algebra

1 Introduction

Interior point methods [25] are frequently used to solve large convex quadratic and linear programs for two reasons. Firstly, the number of iterations taken is typically either constant or grows very slowly with the problem dimension. Secondly, the major computational component is in solving (one or) two systems of linear equations per iteration, for which many efficient, large scale algorithms exist. Thus interior point methods become more and more attractive as the scale of the problem increases. General-purpose implementations of these methods

*This material is based on research supported by National Science Foundation Grants CCR-9972372, CDA-9726385 Air Force Office of Scientific Research Grant F49620-98-1-0417 and Microsoft Corporation.

†Computer Sciences Department, University of Wisconsin, Madison, WI 53706

can be complex, relying upon sophisticated sparse techniques to factor the matrix at each iteration. However, the basic algorithm is straightforward and can be used in a wide variety of applications by simply tailoring the linear algebra to the specific application.

We are particularly interested in applying an interior point method to a class of quadratic programs with two properties: each model contains a small number of linear constraints and the quadratic term consists of a low-rank update to a positive semi-definite matrix. The key to solving these problems is to exploit structure using the Sherman-Morrison-Woodbury formula and the Schur complement. One source of massive problems of this type is the data mining community where several formulations of the linear support vector machine [24, 1, 2, 16] fit into the framework. A related example is the Huber regression problem [14, 17, 26] which can also be posed as a quadratic program of the type considered.

Essentially, the linear support vector machine attempts to construct a hyperplane partitioning two sets of observations where an observation is a point in a low dimensional space consisting of feature measurements. An interesting characteristic of these models is the volume of data, which can lead to quadratic programs with between 10 and 100 million variables and a dense Q matrix. These problems are becoming increasingly important in data mining and machine learning because of the large number of practical applications [4, 23].

Sampling techniques [3] can be used to decrease the number of observations needed to construct a good separating surface. However, if we consider a "global" application and randomly sample only 1% of the current world population, we would generate a problem with around 60 million observations. Recent work [9] has shown that a random sampling of 20% - 30% is sufficient for many applications, but that even sampling 70% - 80% leads to statistically significant differences in the models. Furthermore, for comparative purposes, a researcher might want to solve the non-sampled problem to validate the choice of sampling technique. Therefore, solving realistic, large scale models of this form is an important research area.

Codes targeting massive problems need to effectively deal with the required data volume. For example, one dense vector with 50 million double precision elements requires 400 MB of storage. If all of the data were to be kept in-core, we would rapidly exhaust the available memory resources of most machines. We therefore propose storing *all* of the data out-of-core and overlapping computation and I/O to reduce overhead.

As mentioned above, the crucial implementation details are in the linear algebra calculations. Rather than re-implement a standard predictor-corrector code [20], we have opted to use OOQP [10] as the basis for our work. A key property of OOQP is the object-oriented design which enables us to easily tailor the required linear algebra to the application. Our linear algebra implementation exploits problem structure while keeping all of the data out-of-core. A proximal point modification [22] to the underlying algorithm is also implemented within the linear algebra to improve robustness on the particular formulations considered.

We begin in Section 2 by formally stating the general optimization problem we are interested in solving and show specializations of the framework for linear support vector machines and Huber regression. We then describe the interior-point method and linear algebra requirements in Section 3. The basic proximal point idea is discussed and we demonstrate the use of the Sherman-Morrison-Woodbury formula and the Schur complement to exploit problem structure. The implementation of the linear algebra using out-of-computations is then given in Section 4 along with some numerical considerations made for massive problems. Finally, in Section 5, we present experimental results for several linear support vector machine formulations on a large, randomly generated data set. These results indicate that the method is reliable and scalable to massive problems.

2 Quadratic Programming Framework

The general optimization problem considered has a quadratic term consisting of a low-rank update to a positive semi-definite matrix and a small number of linear constraints. To fix notation, we will consider problems with m variables, n constraints, and a rank k update. Let $Q \in \Re^{m \times m}$ be of the form

$$Q = S + RHR^T$$

where $S \in \Re^{m \times m}$ is symmetric, positive semi-definite, $H \in \Re^{k \times k}$ is symmetric, positive definite and $R \in \Re^{m \times k}$. Note that S is typically a very large matrix while H is small. We are concerned with solving the convex problem:

$$\begin{array}{ll} \min_x & \frac{1}{2}x^T Qx + c^T x \\ \text{s.t.} & Bx = b \\ & x \geq 0 \end{array} \tag{1}$$

for given $B \in \Re^{n \times m}$ with full row rank, $b \in \Re^n$, and $c \in \Re^m$ with $k + n \ll m$. That is, the rank of the update and the number of constraints must be small in relation to the size of the problem. While general bounds, $l \in \Re^m \cup \{-\infty\}^m$ and $u \in \Re^m \cup \{+\infty\}^m$ with $l < u$, can be used in the formulation, we only discuss the model with a simple non-negativity constraint in the sequel for ease of exposition.

To solve instances of this problem, we will exploit structure in the matrices generated by an interior-point algorithm using the Sherman-Morrison-Woodbury formula and the Schur complement [11]. The underlying operations will be carried out in the context of the machine learning applications outlined below. As will become evident, in addition to the assumptions made concerning the form of the quadratic program, we also require that the matrices H and $S + T$ be easily invertible for any positive diagonal matrix T. These assumptions are satisfied in our applications because H and S will be diagonal matrices. However, general cases satisfying the criteria clearly exist.

2.1 Linear Support Vector Machines

The linear support vector machine attempts to construct a hyperplane $\{x \mid w^T x = \gamma\}$ correctly separating two point sets with a maximal separation margin. Several quadratic programming formulations exist in the data mining literature [24, 1, 2, 16] for these problems, which are becoming increasingly important due to the large number of practical applications [4, 23]. The common variation among the optimization models is in the choice of the subset of the variables (w and γ) selected to measure the separation margin and the norm used for the misclassification error.

We first introduce some notation chosen to be consistent with that typically used in the data mining literature. We let $A \in \Re^{m \times k}$ be a (typically dense) matrix representing a set of observations drawn from two sample populations where m is the total number of observations and k the number of features measured for each observation with $k \ll m$. D is a diagonal matrix defined as

$$D_{i,i} = \begin{cases} +1 & \text{if } i \in P_+ \\ -1 & \text{if } i \in P_- \end{cases}$$

where P_+ and P_- are the indices of the elements in the two populations. We use the notation e to represent a vector of all ones of the appropriate dimension.

The standard support vector machine [24, 4] is the following optimization problem:

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w\|_2^2 + \nu e^T y \\ \text{subject to} & D(Aw - e\gamma) + y \geq e \\ & y \geq 0 \end{array} \tag{2}$$

The essential idea is to minimize a weighted sum of the 1-norm of the misclassification error, $e^T y$, and the 2-norm of w, the normal to the hyperplane being derived. The relationship between minimizing $\|w\|_2$ and maximizing the margin of separation is described, for example, in [19]. Here, ν is a parameter weighting the two competing goals related to misclassification error and margin of separation. The constraints just implement the misclassification error.

Various modifications of (2) are developed in the literature. The motivation for many of them is typically to improve the tractability of the problem and to allow novel reformulation in the solution phase. For example, one alternative incorporates γ into the objective function:

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w,\gamma\|_2^2 + \nu e^T y \\ \text{subject to} & D(Aw - e\gamma) + y \geq e \\ & y \geq 0 \end{array} \tag{3}$$

This formulation is described in [18] to allow successive over-relaxation to be applied to the (dual) problem.

A different permutation replaces the 1-norm of y in (3) with the 2-norm and noticing that the non-negativity constraint on y becomes redundant. The resulting problem, first introduced in [19], is then:

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\left\|w,\gamma\right\|_2^2 + \frac{\nu}{2}\left\|y\right\|_2^2 \\ \text{subject to} & D(Aw - e\gamma) + y \geq e \end{array} \tag{4}$$

Again using the Wolfe dual (see (8)), an active set method has been proposed for solution. Concurrent to work described herein, Mangasarian and Musicant have advocated the use of the Sherman-Morrison-Woodbury formula in their active set algorithm.

The final variant considered appears to be new, but is a trivial modification of (4).

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\left\|w\right\|_2^2 + \frac{\nu}{2}\left\|y\right\|_2^2 \\ \text{subject to} & D(Aw - e\gamma) + y \geq e \end{array} \tag{5}$$

As stated, these problems are not in a form matching (1). However, the Wolfe duals [15] of (2) - (5) are respectively:

$$\begin{array}{ll} \min_{x} & \frac{1}{2}x^T DAA^T D^T x - e^T x \\ \text{subject to} & e^T D^T x = 0 \\ & 0 \leq x \leq \nu e \end{array} \tag{6}$$

$$\begin{array}{ll} \min_{x} & \frac{1}{2}x^T DAA^T D^T x + \frac{1}{2}x^T Dee^T D^T x - e^T x \\ \text{subject to} & 0 \leq x \leq \nu e \end{array} \tag{7}$$

$$\begin{array}{ll} \min_{x} & \frac{1}{2\nu}x^T x + \frac{1}{2}x^T DAA^T D^T x + \frac{1}{2}x^T Dee^T D^T x - e^T x \\ \text{subject to} & x \geq 0 \end{array} \tag{8}$$

$$\begin{array}{ll} \min_{x} & \frac{1}{2\nu}x^T x + \frac{1}{2}x^T DAA^T D^T x - e^T x \\ \text{subject to} & e^T D^T x = 0 \\ & x \geq 0 \end{array} \tag{9}$$

which are of the desired form. In addition to the papers cited above, several specialized codes have been applied to solve (6), for example, see [21]. Once the dual problems above are solved, the hyperplane in the primal problems can be recovered with:

- $w = A^T D^T x$ and γ is the multiplier on $e^T D^T x = 0$ for (2) and (5).
- $w = A^T D^T x$ and $\gamma = -e^T D^T x$ for (3) and (4).

Clearly, (6) - (9) are in the class of problems considered. Rather than become embroiled in a debate over the various formulations, we show that our method can be successfully applied to any of them, and leave the relative merits of each to be discussed by application experts.

2.2 Huber Regression

A related problem to the support vector machine is to determine a Huber M-estimator as discussed in [14, 17, 26]. For an inconsistent system of equations $Aw = b$, an error residual is typically minimized, namely $\sum_{i=1}^{m} \rho((Aw - b)_i)$. In order to de-emphasize outliers and avoid non-differentiability when $\rho(\cdot) = |\cdot|$, the Huber M-estimator [13] has been used, which is a convex quadratic for small values of its argument, and is linear for large values.

The corresponding optimization problem is a convex quadratic program:

$$\begin{array}{ll} \min_{w,y,t} & \frac{1}{2}\|t\|_2^2 + \nu e^T y \\ \text{subject to} & -y \le Aw - b - t \le y \end{array}$$

whose dual has the form

$$\begin{array}{ll} \min_{x} & \frac{\nu}{2}\|x\|_2^2 + b^T x \\ \text{subject to} & A^T x = 0 \\ & -e \le x \le e \end{array}$$

Again, this dual has the structure considered whenever the number of observations m is enormous, and the number of features k is small. The aforementioned references indicate how to recover a primal solution from the dual.

3 Interior Point Method

Since (1) is a convex quadratic program, the Karush-Kuhn-Tucker first order optimality conditions [15] are both necessary and sufficient. These optimality conditions can be written as the mixed complementarity problem:

$$\begin{bmatrix} S + RHR^T & -B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} + \begin{bmatrix} c \\ -b \end{bmatrix} \perp \begin{array}{l} x \ge 0 \\ \lambda \text{ free} \end{array} \tag{10}$$

where we define the $\perp$ notation component-wise using

- $a \perp b \ge 0$ if and only if $a \ge 0$, $b \ge 0$, and $ab = 0$
- $a \perp b$ free if and only if $a = 0$.

General lower and upper bounds on x are also easily handled as detailed in [5].

The basic idea of an interior point method for (10) is to solve the nonlinear system of equations:

$$\begin{array}{rcl} (S + RHR^T)x - B^T\lambda + c & = & z \\ Bx & = & b \\ XZe & = & 0 \end{array} \tag{11}$$

with $x \ge 0$ and $z \ge 0$. X and Z are the diagonal matrices formed from x and z, and z represents the complementary variable to x. Letting $x^i > 0$, λ^i, and

$z^i > 0$ be the current iterate, assumed interior to the feasible region, a direction $(\Delta x, \Delta\lambda, \Delta z)$ is calculated by solving the linearization:

$$\begin{array}{rcl} (S + RHR^T)\Delta x - B^T\Delta\lambda - \Delta z & = & z^i - (S + RHR^T)x^i + B^T\lambda^i - c \\ B\Delta x & = & b - Bx^i \\ Z^i\Delta x + X^i\Delta z & = & -X^iZ^ie + \sigma\frac{(x^i)^Tz^i}{m} \end{array}$$

where $\sigma \in [0, 1]$ is a chosen parameter. The new iterate is $x^{i+1} = x^i + \alpha\Delta x$, $\lambda^{i+1} = \lambda^i + \alpha\Delta\lambda$, and $z^{i+1} = z^i + \alpha\Delta z$ where α is chosen so that $(x^{i+1}, \lambda^{i+1}, z^{i+1})$ is in the interior of the feasible region. Note that for $\sigma = 0$ we are calculating a Newton direction for the nonlinear system of equations (11). When using the Newton direction, α is typically small because the iterates rapidly approach the boundary of the feasible region. Therefore, the direction is biased towards the interior of the feasible region by choosing an alternate σ. Convergence results for these methods can be found in [25] and are not discussed here.

The Mehrotra predictor-corrector method is a specific implementation of this basic approach. For the remainder if this section, we will look at the linear algebra necessary to calculate the direction at each iteration. We initially develop the case where S is positive definite and we only have simple bounds. We then discuss the modification made for arbitrary linear constraints. We finish with the most general case where S is not assumed to be positive definite.

3.1 Simple Bound Constrained Case

We first describe the method in the simplest context, that of the support vector machine formulation in (8). In this case, $S = \frac{1}{\nu}I$ is positive definite, $R = D\left[\begin{array}{cc} A & -e \end{array}\right]$, $H = I$, and B is not present. For each iteration of the primal-dual method, we solve two related systems of equations. During the predictor phase, we calculate the Newton direction by solving the system:

$$\left[\begin{array}{cc} S + RHR^T & -I \\ Z^i & X^i \end{array}\right] \left[\begin{array}{c} \Delta\bar{x} \\ \Delta\bar{z} \end{array}\right] = \left[\begin{array}{c} z^i - (S + RHR^T)x^i - c \\ -X^iZ^ie \end{array}\right]$$

The corrector moves the iterate closer to the central path by solving the system:

$$\left[\begin{array}{cc} S + RHR^T & -I \\ Z^i & X^i \end{array}\right] \left[\begin{array}{c} \Delta x \\ \Delta z \end{array}\right] = \left[\begin{array}{c} 0 \\ \sigma\mu e - \Delta\bar{X}\Delta\bar{Z}e \end{array}\right]$$

for some chosen σ and μ. Note that these are the same system but with differing right hand sides. In explaining how to solve these systems, we ignore details concerning the right hand side, and just use Δx and Δz to label the solution variables in both cases above.

Since we are using an interior point code, X and Z are positive diagonal matrices and we can use the second equation to eliminate Δz. Writing

$$V = (Z^i)^{-1}X^i + S,$$

we are then faced with a system of equations:

$$(V + RHR^T)\Delta x = r \tag{12}$$

for appropriately defined r. While the matrices R and H are constant over iterations, the matrix V is iteration dependent.

(12) is a rank-k update to an easily invertible matrix. Therefore, we can use the Sherman-Morrison-Woodbury [11] update formula

$$(V + RHR^T)^{-1} = V^{-1} - V^{-1}R(H^{-1} + R^T V^{-1} R)^{-1} R^T V^{-1}$$

to solve for Δx and then recover Δz. Note that it is trivial to form V^{-1} and H^{-1} since they are both positive definite diagonal matrices and that the matrix $H^{-1} + R^T V^{-1} R$ is a (small) symmetric $k \times k$ matrix, that once formed can be inverted (factor/solved) using standard dense linear algebra subroutines. Since this matrix is independent of the right hand side vector, r, we only have to form and factor this small dense matrix once per iteration. That is, we can use the same factors in both the predictor and the corrector steps. To summarize, in applying the inverse to the vector r we carry out the following steps, which we term

Algorithm SMW:

1. Calculate $t^1 = R^T V^{-1} r$;
2. Solve $(H^{-1} + R^T V^{-1} R)t^2 = t^1$;
3. Determine Δx as the difference between $V^{-1}r$ and $V^{-1}Rt^2$.

Note that t^1 and t^2 are small k-vectors. Furthermore, the calculation in 1 can be carried out at the same time that the matrix required in 2 is being formed. Thus, a complete solve requires two passes through the data stored as R, namely one for steps 1 and 2, and one for step 3. This feature is important to note for the out-of-core implementation discussed in Section 4.

3.2 Constrained Case

We now turn to the case where the quadratic program under consideration still has a positive definite Q matrix but the problem has a small number of linear constraints. For example, the problem (9) falls into this class, where $S = \frac{1}{\nu}I$ is positive definite, $R = DA$, $H = I$ and $B = e^T D^T$. Note that B is a $1 \times m$ matrix with full row rank.

Using the same notation as above, the primal-dual interior point method requires the solution of the following two systems at each iteration:

$$\begin{bmatrix} S + RHR^T & -B^T & -I \\ B & 0 & 0 \\ Z^i & 0 & X^i \end{bmatrix} \begin{bmatrix} \Delta\bar{x} \\ \Delta\bar{\lambda} \\ \Delta\bar{z} \end{bmatrix} = \begin{bmatrix} z^i - (S + RHR^T)x^i - c \\ b - Bx^i \\ -X^i Z^i e \end{bmatrix}$$

and

$$\begin{bmatrix} S+RHR^T & -B^T & -I \\ B & 0 & 0 \\ Z^i & 0 & X^i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e - \Delta\bar{X}\Delta\bar{Z}e \end{bmatrix}$$

for some chosen σ and μ.

Eliminating Δz as before, we generate the following system:

$$\begin{bmatrix} V+RHR^T & -B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r^1 \\ r^2 \end{bmatrix} \tag{13}$$

for appropriately defined r^1 and r^2.

However, we have already shown how to apply $(V + RHR^T)^{-1}$ using the Sherman-Morrison-Woodbury formula in **Algorithm SMW**. We use this observation to eliminate $\Delta x = (V+RHR^T)^{-1}(r^1+B^T\Delta\lambda)$ from (13), and generate the following system in $\Delta\lambda$:

$$\Delta\lambda = (B(V + RHR^T)^{-1}B^T)^{-1}(r^2 + B(V + RHR^T)^{-1}r^1) \tag{14}$$

Note that, in effect, we have just applied the Schur complement to system (13). Since B has full row rank and $V+RHR^T$ is symmetric positive definite, we can conclude that $B(V + RHR^T)^{-1}B^T$ is symmetric and positive definite. Hence, it is invertible. Therefore, the linear system (14) is solvable for any r^1 and r^2.

To form (14) we must solve the system

$$(V + RHR^T) \begin{bmatrix} T^1 & t^2 \end{bmatrix} = \begin{bmatrix} B^T & r^1 \end{bmatrix}$$

essentially the same system, but with multiple right hand sides, corresponding to the columns of B^T and r^1. However, we never need to explicitly form or factor $(V+RHR^T)$, since we can solve for all the right hand sides simultaneously using **Algorithm SMW** of the previous section, only incurring the cost of storing the result $(V + RHR^T)^{-1}B^T$ an an $m \times n$ matrix. Note that in our support vector machine examples $n = 1$.

Let us review the steps needed to solve (13).

1. Form $T^1 = (V + RHR^T)^{-1}B^T$ and $t^2 = (V + RHR^T)^{-1}r^1$ using a simultaneous application of **Algorithm SMW**.

2. Calculate $t^3 = r^2 + Bt^2$ using the solution from step 1.

3. Form the $n \times n$ matrix $T^2 = BT^1$.

4. Solve $T^2\Delta\lambda = t^3$, for the solution of (14).

5. Calculate Δx as $t^2 + T^1\Delta\lambda$.

Steps 2 and 3 can be done concurrent with step 1. Specifically, we can accumulate T^2 and t^3 as the elements in T^1 and t^2 become available from step 3 of **Algorithm SMW**. Note that per iteration, this scheme only requires two passes through the data in R, all in step 1, and one pass through T^1 in step 5.

Furthermore, since the predictor-corrector method requires two solves of the form (13) per iteration with differing r^1 and r^2, the extra storage used for T^1 means that we only need to calculate T^1 once per iteration. In an efficient implementation, we reuse the factors of $V + RHR^T$ in step 2 of **Algorithm SMW** and T^2 in step 4 of the above algorithm in both the predictor and corrector steps of the interior point algorithm.

3.3 General Case

Unfortunately, this is not the end of the story since formulations (6) and (7) do not have a positive definite matrix S, but instead use $S = 0$. In fact, these problems also have lower and upper bounds. In this setting, while the matrix $V = Z^{-1}X$ (for appropriately defined Z and X) is positive definite on the interior of the box defined by the bound constraints, the interior point method typically runs into numerical difficulties when the solution approaches the boundary of the box constraints.

Algorithmically, we would like for the optimization problem to have a positive definite S matrix. In the case where S is already positive definite, no modification need be made to (1). For example, (8) and (9) have positive definite Q matrices and are strongly convex quadratic programs.

However, when S is only positive semi-definite, we use a proximal point modification [22]. Proximal point algorithms augment the objective function with a strongly convex quadratic term and repeatedly solve the resulting quadratic program until convergence is achieved. That is, given x^i, they solve the quadratic program:

$$\begin{array}{ll} \min_x & \frac{1}{2}x^TQx + c^Tx + \frac{\eta}{2}\left\|x - x^i\right\|_2^2 \\ \text{subject to} & Bx = b \\ & x \geq 0 \end{array} \tag{15}$$

for some $\eta > 0$, possibly iteration dependent, to find a new x^{i+1}. The algorithm repeatedly solves subproblems of the form (15) until convergence occurs. Properties of such algorithms are developed in [22, 6] where it can be shown that if the original problem has a solution, then the proximal point algorithm converges to a particular element in the solution set of the original problem. Furthermore, each of the quadratic subproblems is strongly convex.

This approach is used when solving (6) and (7) for example. However, rather than solving each subproblem (15) exactly, we instead solve the subproblems inexactly by just applying one step of the interior point method before updating the subproblem. Thus, in effect, we are solving the following two systems of equations at each iteration:

$$\begin{bmatrix} \eta I + S + RHR^T & -I \\ Z^i & X^i \end{bmatrix} \begin{bmatrix} \Delta\bar{x} \\ \Delta\bar{z} \end{bmatrix} = \begin{bmatrix} z^i - (S + RHR^T)x^i - c \\ -X^iZ^ie \end{bmatrix}$$

and

$$\begin{bmatrix} \eta I + S + RHR^T & -I \\ Z^i & X^i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta z \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma\mu e - \Delta\bar{X}\Delta\bar{Z}e \end{bmatrix}$$

for some chosen σ and μ. Therefore, when using the proximal perturbation algorithm, we use the same interior-point implementation and simply modify the S matrix.

Note that the linear algebra issues are now entirely the same as the issues already covered above. The only differences are in the particular values present in S. The remaining challenge is to solve massive problems. The implementation is discussed in the next section where we use out-of-core computation to reduce memory requirements.

4 Implementation

An interesting feature of these particular models is the volume of data, which can lead to quadratic programs with between 10 and 100 million variables and a dense Q matrix. Quadratic programming codes explicitly using the Q matrix will not work well for these problems. Therefore, we need a method for which we can utilize tailored linear algebra.

We use the Mehrotra predictor-corrector algorithm [20] as implemented in OOQP [10], as the basis for our interior-point method. The OOQP code is written in such a way that we can easily specialize the linear algebra to the application. This feature becomes key when we want to solve large data mining problems.

The linear algebra outlined in Section 3 is used in our linear algebra implementation. As mentioned in Section 3 we calculate as much of the data needed concurrently and reuse appropriate vectors and matrices for the predictor and corrector steps. However, because of the target size, we must effectively deal with the volume of data. Potentially, round-off or accumulation errors could become significant, so we want to minimize these as much as possible. Finally, we want to use a termination condition independent of the problem size. These topics are discussed in the following sections. Clearly, the fact that an interior point algorithms typically require only a small number of iterations is crucial for performance.

4.1 Data Issues

As mentioned, the target problem contains 10 - 100 million observations. Consider for example a model with 50 million observations and assume there are 35 features, each represented by a 1-byte quantity. Then, the observation matrix consumes 1.75 gigabytes of storage. If the features are measured as double precision values, the storage requirement balloons to 14 gigabytes. Furthermore, the quadratic program has 50 million variables. Therefore, each double precision vector requires 400 megabytes of space. If we assume 10 vectors are used, an additional 4 gigabytes of storage is necessary. Therefore, the total space requirement for the algorithm on a problem of this magnitude is between 5.75 and 18 gigabytes. Clearly, an in-core solution is not possible.

Therefore, we must attempt to perform most, if not all, of the operations using data kept out-of-core, while still achieving adequate performance. One observation is that all of the linear algebra discussed in Section 3 accesses the data sequentially. Therefore, while working on one buffer (block) of data, we can be reading the next from disk. The main computational component is constructing the matrix $M = H^{-1} + R^T V^{-1} R$ (see step 2 of **Algorithm SMW**). We begin by splitting R and V^{-1} into p buffers of data and calculate

$$M = H^{-1} + \sum_{j=1}^{p} R_j^T (V^{-1})_j R_j.$$

Note that V is a diagonal matrix in the examples considered, but that more general matrices can be handled with more sophisticated splitting techniques.

To summarize, we perform the following steps to calculate M.

1. Request R_1 and $(V^{-1})_1$ from disk and set $M = H^{-1}$.
2. For $j = 1$ to $p - 1$ do
   (a) Wait for R_j and V_j^{-1} to finish loading.
   (b) Request R_{j+1} and V_{j+1}^{-1} from disk.
   (c) Accumulate $M = M + R_j^T (V^{-1})_j R_j$.
3. Wait for R_p and V_p^{-1} to finish loading.
4. Accumulate $M = M + R_p^T (V^{-1})_p R_p$.

The code uses asynchronous I/O constructs to provide the request and wait functionality. The remainder of the linear algebra in Section 3 can be calculated similarly. The code performs as many of the required steps concurrently with the reading of the R_j buffers from disk.

The amount of data kept in-core is significantly reduced with such a scheme. The tradeoff is that the code will not be as fast as an in-core implementation. We quantify the impact of the out-of-core calculation in Section 5.

4.2 Numerical Considerations

Due to the number of variables in the problems solved, we can run into significant round-off errors while performing the linear algebra, particularly when accumulating the matrices. In an attempt to limit the effect of these numerical errors, we use a hierarchical scheme for the computations.

Consider the construction of the matrix $H^{-1} + R^T V^{-1} R$ using the above technique. Our implementation accumulates the $R_j^T (V^{-1})_j R_j$ components in temporary matrices, M_l for $l = 1, \ldots, L$ and then merges these together as

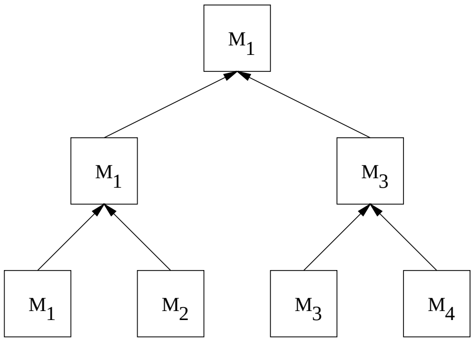

Figure 1: Accumulation Diagram

$M = \sum_{l=1}^{L} M_l$. To do this, the initialization and accumulation steps are updated from the algorithm above into the final form:

1. Request R_1 and $(V^{-1})_1$ from disk and set $M_1 = H^{-1}$ and $M_l = 0$ for $l = 2, \ldots, L$.
2. For $j = 1$ to $p - 1$ do
   (a) Wait for R_j and V_j^{-1} to finish loading.
   (b) Request R_{j+1} and V_{j+1}^{-1} from disk.
   (c) Accumulate $M_{(j \mod L)+1} = M_{(j \mod L)+1} + R_j^T (V^{-1})_j R_j$.
3. Wait for R_p and V_p^{-1} to finish loading.
4. Accumulate $M_{(p \mod L)+1} = M_{(p \mod L)+1} + R_p^T (V^{-1})_p R_p$.
5. Merge $M = \sum_{l=1}^{L} M_l$.

The merge is implemented by repeatedly adding the $\frac{L}{2}$ neighbors as depicted in Figure 1. A similar procedure is used for the vector computations. The code uses $L = 8$ for the calculations. We note that the above algorithm is dependent upon the buffer size read from disk. This dependency is removed in the code by further partitioning R_j and V_j^{-1} into smaller buffers with 50,000 elements.

4.3 Termination Criteria

Finally, we terminate based on the inf-norm of the Fischer-Burmeister function [8], with an appropriate modification for the presence of equations [7], for the complementarity problem (10). The inf-norm is independent of the number of variables in the problem and can be stably calculated given evaluations of the linear functions in (10). We further note that the function evaluation is calculated while determining the right-hand side, $z^i - (S + RHR^T)x^i - c$, for the

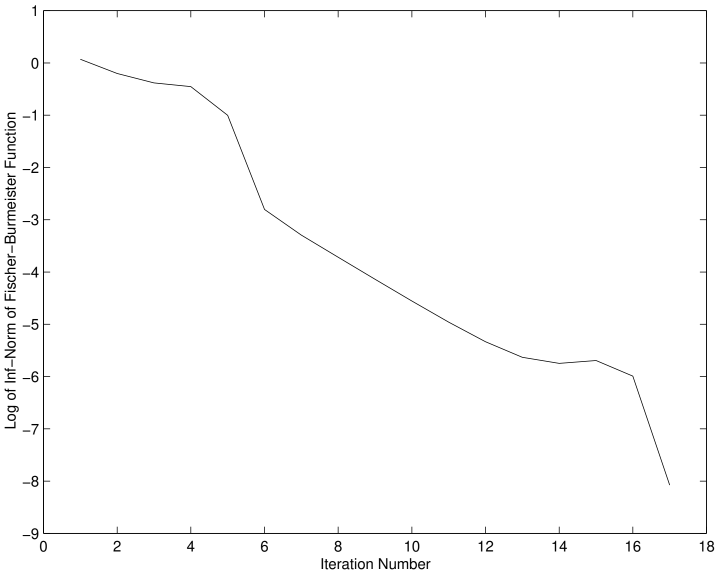

Figure 2: Log residual as a function of iterations for problem (6) with 10 million observations

predictor step. Therefore, the function calculation does not cost an additional pass through the data. We use a termination criteria of 10^{-6} for the Fischer-Burmeister function within the code, which is much more stringent than the default criteria for OOQP. In Figure 2 we plot the (log) residual as a function of the iteration for problem (6) with 10 million observations. Note the consistency of the decrease in the residual.

The machine learning community will sometimes terminate an algorithm based upon conditions other than optimality, such as tuning set accuracy [18]. As Figure 2 shows, using similar criteria would provide benefits to our code as well.

5 Results

For experimentation, we generated a random data set with 34 features that is separable. We did this by constructing a separating hyperplane and then creating data points and classifying them with the hyperplane. The data generated contains 60 million observations of 34 features where each feature has an integer value between 1 and 10. Multiplication by D was performed while generating the data, with De being encoded as an additional column to the observation set. Each of the feature measurements is a 1-byte quantity.

We limited the size to 60 million observations to avoid problems with the 2 gigabyte file size restriction imposed by various operating systems. To increase the size further without changing operating system, we could store the original data in multiple files.

All of the tests were run on a 330 Mhz SUN Ultrasparc with 2 processors and 768 MB of RAM. The asynchronous I/O routines are implemented using threads. Thus, both of the processors are used for the tests. Results on a uniprocessor machine indicate that the impact of the second processor is minimal.

We first quantify the effects of using an out-of-core implementation. We then look at the scalability of the algorithm.

5.1 Out-of-Core Impact

The impact on performance of using an out-of-core implementation was tested using the formulation in (8) with $\nu = 1$. Since S is positive definite in this case, no proximal point modification was necessary.

The first property investigated is the effect of out-of-core computations on performance using asynchronous I/O. To test the performance, we ran problems varying the size between 200,000 and 1 million observations. A buffer size of 100,000 observations (elements) for each matrix (vector) was used for the out-of-core computations. We ran each of the tests 5 times and use the median values in the figures. The average time per iteration is reported in Figure 3 for in-core, asynchronous I/O, and synchronous I/O implementations. While the asynchronous I/O is not as fast as keeping everything in core, we notice only a 14% increase in time over the in core implementation for the chosen buffer and problem sizes. Synchronous I/O results in a 16% increase. The conclusion to be drawn here is that an out-of-core implementation of the algorithm uses limited space but results in increased time. We believe that the enormous decrease in the amount of RAM used for a 14% increase in time is a reasonable tradeoff to make. A case can also be made for using the easier to implement synchronous I/O. However, we note from the graph that the margin between asynchronous and synchronous I/O appears to be growing, thus leading us to believe that for larger problems, synchronous I/O would have a more significant impact.

The next set of experiments was designed to determine the impact of modifying the buffer size. For these tests, we fixed the problem size to 1 million observations and varied the buffer size from 50,000 to 500,000 elements. The average time per iteration is plotted in Figure 4. The results indicate a buffer size of around 250,000 elements is optimal with an 11% increase in time over the in-core solution. The total amount of data buffered in main memory is between 114 and 160 megabytes depending upon the problem formulation used.

5.2 Massive Problems

Based on the results in the previous section, we decided to use asynchronous I/O and a buffer size of 250,000 elements. We are now interested in determining the reliability of the algorithm on the various formulations and the scalability of the implementation to massive problems. In order to do this, we varied the problem size between 1 and 60 million observations. In all of these tests $\nu = 1$

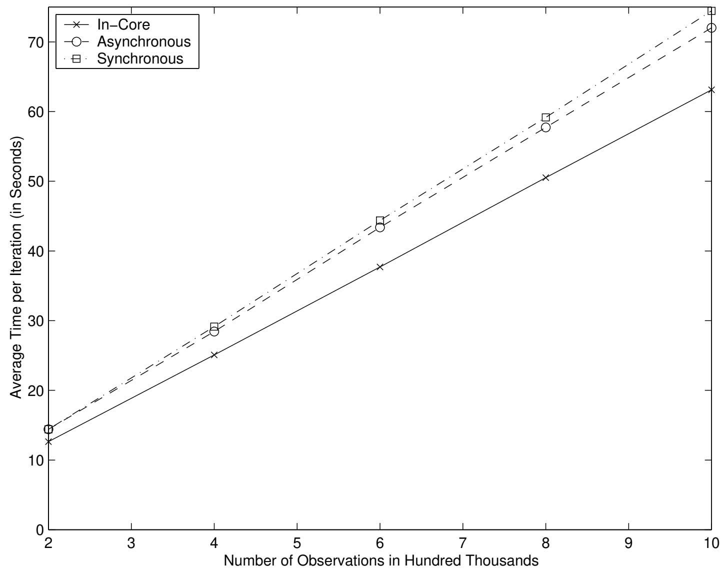

Figure 3: Average time per iteration for various problem sizes with a fixed buffer size of 100,000 elements

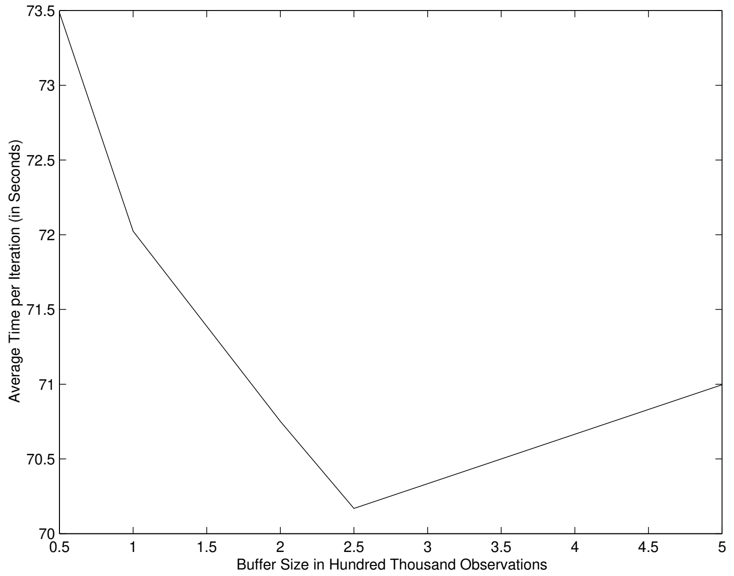

Figure 4: Average time per iteration for various buffer sizes with a fixed problem size of 1,000,000 observations

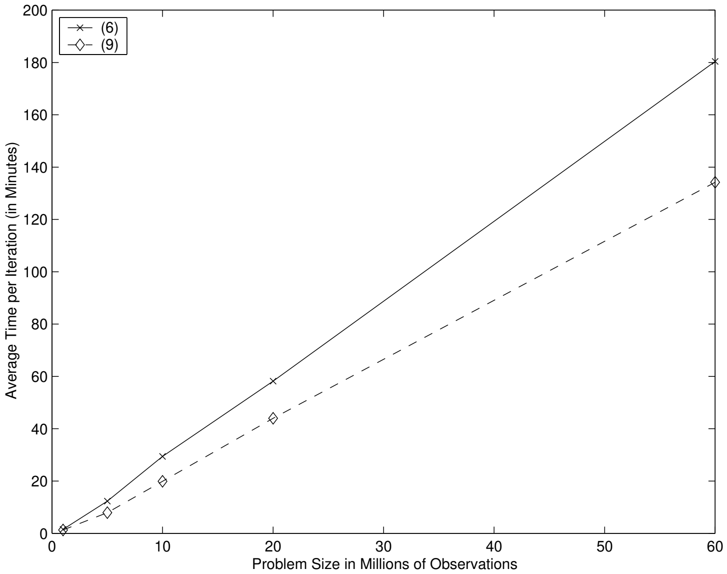

Figure 5: Average time per iteration comparison of the different formulations with varying problem size

was used and for the models in (6) and (7), a proximal perturbation of $\eta = 100$ was used.

Each model was run one time with problem sizes of 1, 5, 10, 20, and 60 million observations. We plot average time per iteration and number of iterations as functions of problem size in Figure 5 and Figure 6 respectively. The similarity in the average time per iteration between formulations (8) and (9) (and also between (6) and (7)) is indistinguishable. To avoid clutter, we only plot the results for (6) and (9) in Figure 5. The total times are reported in Figure 7.

The average time per iteration appears to grow almost linearly with the problem size. This result is to be expected, as the majority of the time taken per iteration is in constructing $H^{-1} + R^T V^{-1} R$. The floating-point operations necessary to calculate this quantity grows linearly with problem size m (but quadratically with the number of features k). The extra time needed for (6) is due to the treatment of upper bounds.

A surprising result for the constrained formulations, (6) and (9), is that the number of iterations remains fairly flat as the problem size increases and even decreases for some of the larger problems. This fact is counter-intuitive and likely related to the random nature of the model. However, more tests on "real" datasets need to be performed before drawing any conclusions. As expected, the number of iterations taken for (8) and (9) increases slowly with the dimension of the problem.

It would appear that the constrained formulations (6) and (9) are most

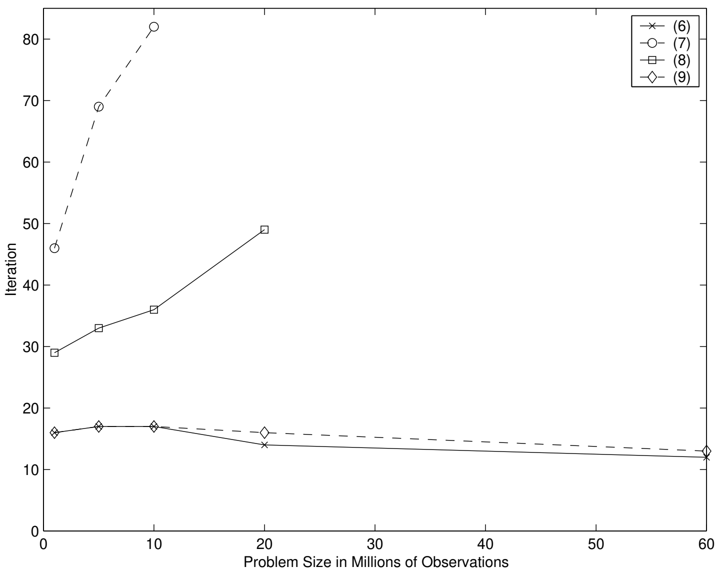

Figure 6: Iteration comparison of the different formulations with varying problem size

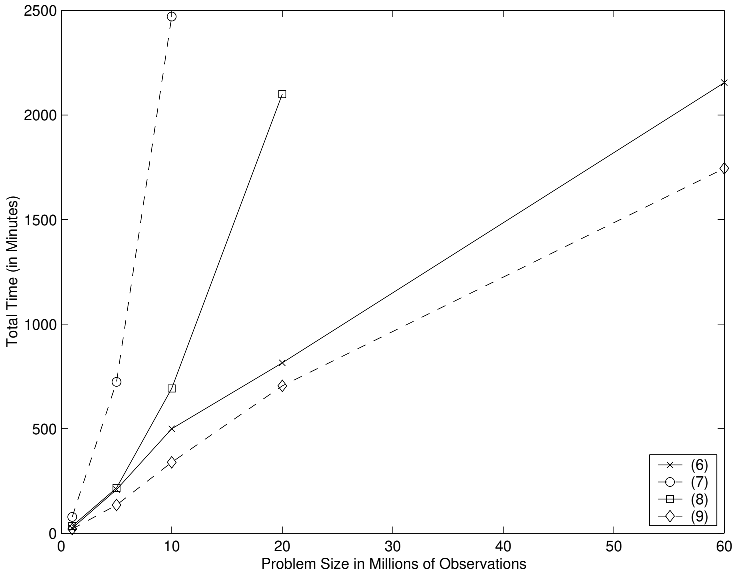

Figure 7: Total time comparison of the different formulations with varying problem size

tractable for interior point methods. Both of these formulations solved the 60 million observation problem in a 29 - 36 hours on a standard workstation. We have not run formulations (7) and (8) for 60 million observations since their timings are somewhat longer.

We believe the strength of this approach is its scalability and reliability. While it may be possible to adjust the parameters of the interior point method or the parameters of the proximal point iteration for improved performance, we have just elected to go with the same defaults on all problems and have not run into any numerical problems beyond those that we addressed in Section 4.2.

6 Conclusions

We have developed an interior-point code for solving several quadratic programming formulations of the linear support vector machine. We are able to solve large problems reasonably by exploiting the linear algebra and using out-of-core computations. Scalability of the approach has been demonstrated.

We also remark that our framework allows other formulations of the support vector machine to be explored. For example, using the inf-norm of the misclassification error we have the problems:

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w\|_2^2 + \nu y \\ \text{subject to} & D(Aw - e\gamma) + ey \geq e \\ & y \geq 0 \end{array}$$

with $y \in \Re$, and

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w,\gamma\|_2^2 + \nu y \\ \text{subject to} & D(Aw - e\gamma) + ey \geq e \\ & y \geq 0 \end{array}$$

whose Wolfe duals are

$$\begin{array}{ll} \min_{x} & \frac{1}{2}x^T DAA^T D^T x - e^T x \\ \text{subject to} & e^T D^T x = 0 \\ & e^T x + s = \nu \\ & x \geq 0, s \geq 0 \end{array}$$

and

$$\begin{array}{ll} \min_{x} & \frac{1}{2}x^T DAA^T D^T x + \frac{1}{2}x^T Dee^T D^T x - e^T x \\ \text{subject to} & e^T x + s = \nu \\ & x \geq 0, s \geq 0 \end{array}$$

respectively, where we have added a slack variable, s, to the general constraints. Note that the general constraints for these problems have full row rank. Therefore, they can be effectively solved using the interior-point method developed.

Finally, the linear algebra used can be parallelized, and by distributing the data across multiple disks further speedups can be realized. More sophisticated corrector implementations [12] of the interior-point code can be used to further reduce the iteration count. These are the topics of future work.

Acknowledgements

We would like to thank Olvi Mangasarian for bringing this problem to our attention, Yuh-Jye Lee and Dave Musicant for numerous insightful discussions on machine learning problems, and to Mike Gertz, Jeff Linderoth, and Steve Wright for making a preliminary version of OOQP available to us.

References

[1] P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13:1–10, 2000.

[2] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[3] W. Cochran. *Sampling Techniques*. Wiley, New York, third edition, 1977.

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

[5] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.

[6] M. C. Ferris. Finite termination of the proximal point algorithm. *Mathematical Programming*, 50:359–366, 1991.

[7] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86:475–497, 1999.

[8] A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.

[9] V. Ganti, J. Gehrke, and R. Ramakrishnan. A framework for measuring changes in data characteristics. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 126–137. ACM Press, 1999.

[10] M. Gertz, J. Linderoth, and S. Wright. Object-oriented software for linear complementarity and quadratic programming. Technical report, MCS Division, Argonne National Laboratory, in preparation.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[12] J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications*, 6:137–156, 1996.

[13] P. J. Huber. *Robust Statistics.* John Wiley, New York, 1981.

[14] W. Li and J. J. Swetits. The linear ℓ_1 estimator and the Huber M-estimator. *SIAM Journal on Optimization*, 8:457–475, 1998.

[15] O. L. Mangasarian. *Nonlinear Programming.* McGraw–Hill, New York, 1969. SIAM Classics in Applied Mathematics 10, SIAM, Philadelphia, 1994.

[16] O. L. Mangasarian. Generalized support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146. MIT Press, Cambridge, MA, 2000.

[17] O. L. Mangasarian and David R. Musicant. Robust linear and support vector regression. Technical Report 99-09, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999.

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

[19] O. L. Mangasarian and David R. Musicant. Active set support vector machine classification. Technical Report 00-04, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 2000.

[20] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.

[21] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, 1999.

[22] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14:877–898, 1976.

[23] B. Schölkopf, C. Burges, and A. Smola, editors. *Advances in Kernel Methods: Support Vector Machines.* MIT Press, Cambridge, MA, 1998.

[24] V. N. Vapnik. *The Nature of Statistical Learning Theory.* John Wiley & Sons, New York, 1996.

[25] S. J. Wright. *Primal–Dual Interior–Point Methods.* SIAM, Philadelphia, Pennsylvania, 1997.

[26] S. J. Wright. On reduced convex qp formulations of monotone lcp problems. Technical Report MCS–P808–0400, Argonne National Laboratory, Argonne, Illinois, 2000.