SIAM J. OPTIM.
Vol. 13, No. 3, pp. 783–804


# INTERIOR-POINT METHODS FOR MASSIVE SUPPORT VECTOR MACHINES*

MICHAEL C. FERRIS† AND TODD S. MUNSON‡

**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 semidefinite 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, if written explicitly, a dense $Q$ matrix. Our code is based on OOQP, an object-oriented interior-point code, with the linear algebra specialized for the support vector machine application. For the targeted massive problems, all of the data is stored out of core and we overlap computation and input/output to reduce overhead. Results are reported for several linear support vector machine formulations demonstrating that the method is reliable and scalable.

**Key words.** support vector machine, interior-point method, linear algebra

**AMS subject classifications.** 90C51, 90C20, 62H30

**PII.** S1052623400374379

## 1. Introduction.

Interior-point methods [30] are frequently used to solve large convex quadratic and linear programs for two reasons. First, the number of iterations taken is typically either constant or grows very slowly with the problem dimension. Second, the major computation involves solving (one or) two systems of linear equations per iteration, for which many efficient, large-scale algorithms exist. Thus, interior-point methods become more attractive as the size of the problem increases. General-purpose implementations of these methods can be complex, relying upon sophisticated sparse techniques to factor the relevant matrix at each iteration. However, the basic algorithm is straightforward and can be used in a wide variety of problems by simply tailoring the linear algebra to the 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 (dense) low-rank update to a positive semidefinite matrix. The key to solving these problems is to exploit structure using block eliminations. One source of massive problems of this type is the data mining community, where several linear support vector machine (SVM) formulations [28, 1, 2, 19] fit into the framework. A related example is the Huber regression problem [17, 21, 31], which can also be posed as a quadratic program of the type considered.

The linear SVM attempts to construct a hyperplane partitioning two sets of observations, where each observation is an element of a low-dimensional space. An

*Received by the editors May 24, 2000; accepted for publication (in revised form) March 21, 2002; published electronically January 3, 2003. This material is based on research supported by National Science Foundation grants CCR-9972372 and CDA-9726385; Air Force Office of Scientific Research grant F49620-01-1-0417; Microsoft Corporation; and the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under contract W-31-109-Eng-38.

http://www.siam.org/journals/siopt/13-3/37437.html

†Computer Sciences Department, University of Wisconsin, Madison, WI 53706 (ferris@cs.wisc.edu).

‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 (tmunson@mcs.anl.gov).

interesting characteristic of these models is the volume of data, which can lead to quadratic programs with between 10 and 100 million variables and, if written explicitly, a dense $Q$ matrix. The large number of practical applications of the SVM [6, 26] is indicative of the importance of robust, scalable algorithms to the data mining and machine learning communities.

Sampling techniques [3] can be used to decrease the number of observations needed to construct a good separating surface. However, if we considered a "global" application and randomly sampled only 1% of the current world population, we would generate a problem with around 60 million observations. Recent work [10] has shown that although a random sampling of 20–30% is sufficient for many applications, sampling even as high as 70–80% can produce statistically significant differences in the models. Furthermore, for comparative purposes, a researcher might wish to solve the nonsampled problem to validate the choice of sampling technique.

Solving realistic, large-scale models of this form raises important research issues. In particular, codes targeting massive problems need to handle the required data volume effectively. For example, one dense vector with 50 million double-precision elements requires 400 megabytes of storage. If all data were to be kept in core, we would rapidly exhaust the memory resources of most machines available today. Therefore, we store *all* data out of core and overlap computation and input/output (I/O) to reduce the overhead inherent in such a scheme.

As mentioned above, the crucial implementation details are in the linear algebra calculation. Rather than reimplement a standard predictor-corrector interior-point code [23], we use OOQP [11, 12] as the basis for our work. A key property of OOQP is the object-oriented design, which enables us to 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 [25] to the underlying algorithm is also available to improve robustness on some of the SVM formulations considered.

We begin in section 2 by formally stating the general optimization problem we are interested in solving, and we show specializations of the framework for linear SVMs and Huber regression. In section 3, we describe the interior-point method and linear algebra requirements. The basic proximal-point idea is discussed, and we demonstrate the use of block eliminations to exploit problem structure. The implementation of the linear algebra using out of core computations is presented in section 4, along with some numerical considerations for massive problems. In section 5, we present experimental results for several linear SVM formulations on two large, randomly generated data sets. These results indicate that the method is reliable and scalable to massive problems. In section 6, we summarize our work and briefly outline future efforts.

**2. Quadratic programming framework.** The general optimization problem we consider has a quadratic term consisting of a low-rank update to a positive semidefinite matrix and a small number of linear constraints. In particular, the problems discussed have $m$ 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 semidefinite, $H \in \Re^{k\times k}$ is symmetric positive definite, and $R \in \Re^{m\times k}$. Typically, $S$ is a very large matrix while $H$ is small. We are

concerned with solving the convex problem

$$\begin{array}{ll} \min_x & \frac{1}{2}x^T Q x + c^T x \\ \text{s.t.} & Bx = b, \\ & \ell \le x \le u \end{array} \tag{2.1}$$

for given $B \in \Re^{n\times m}$ with full row rank, $b \in \Re^n$, $c \in \Re^m$, and general bounds, $\ell \in \Re^m \cup \{-\infty\}^m$ and $u \in \Re^m \cup \{+\infty\}^m$ with $\ell < u$. We assume that $k + n \ll m$. That is, the rank of the update and the number of constraints must be small in relation to the overall size of the problem.

To solve instances of this problem, we exploit structure in the matrices generated by an interior-point algorithm using block eliminations. The underlying operations are 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 $S + T$ can be inverted easily for any positive diagonal matrix $T$. These assumptions are satisfied in our applications because $H$ and $S$ are diagonal matrices. However, general cases satisfying these criteria clearly exist.

**2.1. Linear SVMs.** The linear SVM 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 [28, 1, 2, 19] for these problems, which are becoming increasingly important because of the large number of practical applications [6, 26]. The common variation among the optimization models is in the choice of the subset of the variables ($w$ and $\gamma$) 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 $k \ll m$. Typically, the observation matrix $A$ is scaled so that $\|A\|_\infty \approx k$. Let $D \in \Re^{m\times m}$ be 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 SVM [28, 6] is the following optimization problem:

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w\|_2^2 + \nu e^T y \\ \text{subject to} & D(Aw - e\gamma) + y \ge e, \\ & y \ge 0. \end{array} \tag{2.2}$$

The essential idea is to minimize a weighted sum of the one-norm of the misclassification error, $e^T y$, and the two-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 [22]. Here, $\nu$ is a parameter weighting the two compet-

ing goals related to misclassification error and margin of separation. The inequality constraints implement the misclassification error.

Various modifications of (2.2) are developed in the literature. The motivation for many of them is typically to improve the tractability of the problem and to allow novel reformulations in the solution phase. For example, one formulation incorporates $\gamma$ into the objective function:

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w,\gamma\|_2^2 + \nu e^T y \\ \text{subject to} & D(Aw - e\gamma) + y \geq e, \\ & y \geq 0. \end{array} \tag{2.3}$$

This formulation is described in [20] to allow successive overrelaxation to be applied to the (dual) problem.

A different permutation replaces the one-norm of $y$ in (2.3) with the two-norm, such that the nonnegativity constraint on $y$ becomes redundant. The resulting problem, first introduced in [22], is then

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w,\gamma\|_2^2 + \frac{\nu}{2}\|y\|_2^2 \\ \text{subject to} & D(Aw - e\gamma) + y \geq e. \end{array} \tag{2.4}$$

An active set method on the Wolfe dual of (2.4) is proposed in [22] to calculate a solution. Concurrent with work described here, Mangasarian and Musicant advocated the use of the Sherman–Morrison–Woodbury update formula in their active set algorithm.

Another variant considered [5] is a slight modification of (2.4):

$$\begin{array}{ll} \min_{w,\gamma,y} & \frac{1}{2}\|w\|_2^2 + \frac{\nu}{2}\|y\|_2^2 \\ \text{subject to} & D(Aw - e\gamma) + y \geq e. \end{array} \tag{2.5}$$

We can also use a one-sided Huber M-estimator [16] for the misclassification error within the linear SVM. This function is a convex quadratic for small values of its argument and is linear for large values. The resulting quadratic program is a combination of (2.2) and (2.5),

$$\begin{array}{ll} \min_{w,\gamma,y,t} & \frac{1}{2}\|w\|_2^2 + \frac{\nu_1}{2}\|t\|_2^2 + \nu_2 e^T y \\ \text{subject to} & D(Aw - e\gamma) + t + y \geq e, \\ & y \geq 0, \end{array} \tag{2.6}$$

where $\nu_1$ and $\nu_2$ are two parameters. We note that $\frac{\nu_2}{\nu_1}$ is the switching point between the quadratic and linear error terms. When $\nu_1 \to \infty$ or $\nu_2 \to \infty$, we recover (2.2) or (2.5), respectively. A similar unification of (2.3) and (2.4) can be made by incorporating $\gamma$ into the objective function of (2.6). For completeness, this problem is

$$\begin{array}{ll} \min_{w,\gamma,y,t} & \frac{1}{2}\|w,\gamma\|_2^2 + \frac{\nu_1}{2}\|t\|_2^2 + \nu_2 e^T y \\ \text{subject to} & D(Aw - e\gamma) + t + y \geq e, \\ & y \geq 0. \end{array} \tag{2.7}$$

As stated, these problems are not in a form matching (2.1). However, the Wolfe duals [18] of (2.2)–(2.7) are, respectively,

$$\begin{array}{ll} \min_x & \frac{1}{2}x^T DAA^T Dx - e^T x \\ \text{subject to} & e^T Dx = 0, \\ & 0 \le x \le \nu e, \end{array} \tag{2.8}$$

$$\begin{array}{ll} \min_x & \frac{1}{2}x^T DAA^T Dx + \frac{1}{2}x^T Dee^T Dx - e^T x \\ \text{subject to} & 0 \le x \le \nu e, \end{array} \tag{2.9}$$

$$\begin{array}{ll} \min_x & \frac{1}{2\nu}x^T x + \frac{1}{2}x^T DAA^T Dx + \frac{1}{2}x^T Dee^T Dx - e^T x \\ \text{subject to} & x \ge 0, \end{array} \tag{2.10}$$

$$\begin{array}{ll} \min_x & \frac{1}{2\nu}x^T x + \frac{1}{2}x^T DAA^T Dx - e^T x \\ \text{subject to} & e^T Dx = 0, \\ & x \ge 0, \end{array} \tag{2.11}$$

$$\begin{array}{ll} \min_x & \frac{1}{2\nu_1}x^T x + \frac{1}{2}x^T DAA^T Dx - e^T x \\ \text{subject to} & e^T Dx = 0, \\ & 0 \le x \le \nu_2 e, \end{array} \tag{2.12}$$

$$\begin{array}{ll} \min_x & \frac{1}{2\nu_1}x^T x + \frac{1}{2}x^T DAA^T Dx + \frac{1}{2}x^T Dee^T Dx - e^T x \\ \text{subject to} & 0 \le x \le \nu_2 e, \end{array} \tag{2.13}$$

which are of the desired form. In addition to the papers cited above, several specialized codes have been applied to solve (2.8); for example, see [24]. Once the dual problems above are solved, the hyperplane in the primal problems can be recovered as follows:

- $w = A^T Dx$, and $\gamma$ is the multiplier on $e^T Dx = 0$ for (2.2), (2.5), and (2.6).
- $w = A^T Dx$, and $\gamma = -e^T Dx$ for (2.3), (2.4), and (2.7).

Clearly, (2.8)–(2.13) 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 we leave the relative merits of each to be discussed by application experts in the machine learning field.

**2.2. Huber regression.** A problem related to the SVM is to determine a Huber M-estimator, as discussed in [17, 21, 28, 31]. 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 deemphasize outliers and avoid nondifferentiability when $\rho(\cdot) = |\cdot|$, the Huber M-estimator [16] has been used.

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

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

The 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 (2.1) is a convex quadratic program, the Karush–Kuhn–Tucker first-order optimality conditions [18] are both necessary and sufficient. These optimality conditions can be written as the mixed complementarity problem

$$\begin{bmatrix} S+RHR^T & -B^T & -I & I \\ B & 0 & 0 & 0 \\ I & 0 & 0 & 0 \\ -I & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \\ w \\ v \end{bmatrix} + \begin{bmatrix} c \\ -b \\ -\ell \\ u \end{bmatrix} \perp \begin{array}{l} x \text{ free}, \\ \lambda \text{ free}, \\ w \geq 0, \\ v \geq 0, \end{array} \tag{3.1}$$

where we have augmented the system with slack variables to handle general lower and upper bounds. The $\perp$ notation is defined componentwise by using

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

If the lower or upper bounds are infinite, then the corresponding $w$ and $v$ variables are removed from the problem. The reason for adding slack variables is to eliminate the bounds on $x$ and make the initial starting-point calculation easy.

The basic idea of an interior-point method for (3.1) is to solve the equivalent nonlinear system of equations

$$\begin{aligned} (S+RHR^T)x - B^T\lambda - w + v &= -c, \\ Bx &= b, \\ x - \ell &= y, \\ u - x &= z, \\ Wy &= 0, \\ Vz &= 0, \end{aligned} \tag{3.2}$$

with $w \geq 0$, $v \geq 0$, $y \geq 0$, and $z \geq 0$, where $W$ and $V$ are the diagonal matrices formed from $w$ and $v$. Furthermore, $y$ and $z$ represent the variables complementary to $w$ and $v$. Convergence results for these methods can be found in [30] and are not discussed here. Specializations of the interior-point method to the SVM case can be found in [27].

The Mehrotra predictor-corrector method [23] is a specific type of interior-point method. The iterates for the algorithm are guaranteed to remain interior to the simple bounds; that is, $w_i > 0$, $v_i > 0$, $y_i > 0$, and $z_i > 0$ for each iteration $i$. During the predictor phase, we calculate the Newton direction for (3.2), while the corrector moves the iterate closer to the central path. The direction $(\Delta x, \Delta\lambda, \Delta w, \Delta v, \Delta y, \Delta z)$ is calculated by solving the linearization

$$\begin{bmatrix} S+RHR^T & -B^T & -I & I & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 \\ I & 0 & 0 & 0 & -I & 0 \\ -I & 0 & 0 & 0 & 0 & -I \\ 0 & 0 & Y_i & 0 & W_i & 0 \\ 0 & 0 & 0 & Z_i & 0 & V_i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta w \\ \Delta v \\ \Delta y \\ \Delta z \end{bmatrix}$$

$$= \begin{bmatrix} -c - (S+RHR^T)x_i + B^T\lambda_i + w_i - v_i \\ b - Bx_i \\ \ell - x_i + y_i \\ -u + x_i + z_i \\ -W_iy_i + \sigma\frac{(w_i)^Ty_i + (v_i)^Tz_i}{2m}e \\ -V_iz_i + \sigma\frac{(w_i)^Ty_i + (v_i)^Tz_i}{2m}e \end{bmatrix},$$

where $\sigma \in [0,1]$ is a chosen parameter. Different choices for $\sigma$ give rise to the predictor and the corrector steps, respectively. Since these two systems just differ in the right-hand sides, in the following we simplify our presentation by omitting the details of these vectors and replacing them with generic terms $r$.

The particular structure of the above linearization allows us to eliminate the variables $\Delta w$, $\Delta v$, $\Delta y$, and $\Delta z$ from the system. Thus, at each iteration of the algorithm, we solve two systems of linear equations of the form

$$\begin{bmatrix} C + RHR^T & -B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}, \tag{3.3}$$

where

$$C := S + Y_i^{-1} W_i + Z_i^{-1} V_i \tag{3.4}$$

is iteration dependent and $r_1$ and $r_2$ are the appropriate right-hand sides. The right-hand sides are the only values that change between the predictor and corrector step. See [11] for further information on the calculation of the right-hand sides and diagonal modification.

For the remainder of this section, we 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 have only 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 SVM formulation in (2.10). In this case, $S = \frac{1}{\nu} I$ is positive definite, $R = D \begin{bmatrix} A & -e \end{bmatrix}$, $H = I$, and $B$ is not present. The linear system (3.3) reduces to

$$(C + RHR^T)\Delta x = r_1. \tag{3.5}$$

While the matrices $R$ and $H$ are constant over iterations, the matrix $C$ in (3.4) and $r_1$ are iteration dependent.

The matrix in (3.5) is a rank-$k$ update to an easily invertible matrix. Therefore, we can use the Sherman–Morrison–Woodbury [13] formula

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

to solve for $\Delta x$. It is trivial to form $C^{-1}$ and $H^{-1}$ because they are both positive definite diagonal matrices. The matrix $H^{-1} + R^T C^{-1} R$ is a (small) symmetric $k \times k$ matrix that, once formed, can be handled by standard dense linear algebra subroutines. Since this matrix is independent of $r_1$, we have to form and factor this small dense matrix only once per iteration. That is, we can use the same factors in both the predictor and the corrector steps. To summarize, to solve (3.5), we carry out the following steps.

Algorithm SMW.

1. Calculate $t_1 = R^T C^{-1} r_1$.
2. Solve $(H^{-1} + R^T C^{-1} R) t_2 = t_1$.
3. Determine $\Delta x = C^{-1}(r_1 - R t_2)$.

Note that $t_1$ and $t_2$ are small $k$-vectors. Furthermore, the calculation in step 1 can be carried out at the same time the matrix required in step 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 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, problem (2.11) falls into this class, where $S = \frac{1}{\nu}I$ is positive definite, $R = DA$, $H = I$, and $B = e^T D$. Note that $B$ is a nonzero, $1 \times m$ matrix with full row rank.

The predictor-corrector method requires the solution of (3.3) at each iteration. We have already shown how to apply $(C + RHR^T)^{-1}$ using Algorithm SMW. We use this observation to eliminate $\Delta x = (C + RHR^T)^{-1}(r_1 + B^T\Delta\lambda)$ from (3.3) and generate the following system in $\Delta\lambda$:

$$B(C + RHR^T)^{-1}B^T\Delta\lambda = r_2 - B(C + RHR^T)^{-1}r_1. \tag{3.6}$$

Since $B$ has full row rank and $C + RHR^T$ is symmetric positive definite, we conclude that $B(C+RHR^T)^{-1}B^T$ is symmetric and positive definite. Hence, it is nonsingular, and the linear system (3.6) is solvable for any $r_1$ and $r_2$.

To use (3.6), we must solve the system

$$(C + RHR^T)\begin{bmatrix} T_1 & t_2 \end{bmatrix} = \begin{bmatrix} B^T & r_1 \end{bmatrix}$$

with multiple right-hand sides corresponding to the columns of $B^T$ and $r_1$. However, we never need to form or factor $(C + RHR^T)$ explicitly, since we can solve for all the right-hand sides simultaneously using Algorithm SMW, incurring the cost only of storing $T_1$, an $m \times n$ matrix, and $t_2$. Note that in our SVM examples, $n = 1$.

Let us review the steps needed to solve (3.3).

1. Form $T_1 = (C + RHR^T)^{-1}B^T$ and $t_2 = (C + RHR^T)^{-1}r_1$ 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 (3.6).
5. Calculate $\Delta x = t_2 + T_1\Delta\lambda$.

Steps 2 and 3 can be done concurrently 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. Per iteration, this scheme requires only 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 (3.3) per iteration with different $r_1$ and $r_2$, the extra storage used for $T_1$ means that we need to calculate $T_1$ only once per iteration. For efficiency, we reuse the factors of $C + RHR^T$ 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 because formulations (2.8) and (2.9) 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 $C = Y^{-1}W + Z^{-1}V$ (for appropriately defined $W$, $V$, $Y$, and $Z$) 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 the optimization problem to have a positive definite $S$ matrix. When $S$ is already positive definite, no modifications are needed in (2.1). For example, (2.10) and (2.11) have positive definite $Q$ matrices and are strongly convex quadratic programs.

However, when $S$ is only positive semidefinite (or zero), we can use a proximal-point modification [25]. 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

$$\begin{array}{ll} \min_x & \frac{1}{2}x^TQx + c^Tx + \frac{\eta}{2}\|x - x_i\|_2^2 \\ \text{subject to} & Bx = b, \\ & x \geq 0 \end{array} \tag{3.7}$$

for some $\eta > 0$, possibly iteration dependent, to find a new $x^{i+1}$. The algorithm repeatedly solves subproblems of the form (3.7) until convergence occurs. Properties of such algorithms are developed in [25, 7], where it is 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 may be used to solve (2.8) and (2.9), for example. However, rather than solving each subproblem (3.7) exactly, we instead solve the subproblems inexactly by applying just one step of the interior-point method before updating the subproblem. Thus, in effect, we are solving at each iteration the system of equations

$$\begin{bmatrix} C + RHR^T + \eta I & -B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}.$$

Therefore, when using the proximal-point perturbation algorithm, we use the same interior-point implementation and simply modify the $C$ matrix.

Since a proximal-point perturbation can cause the algorithm to take many iterations, our code initiates the proximal-point perturbation algorithm only when numerical difficulties are encountered. We identify numerical difficulties with an increase in the error for satisfying the equations. This technique switches to the proximal-point algorithm when necessary.

The linear algebra issues are now the same as the issues already covered above except for 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 an out of core computation to reduce memory requirements.

**4. Implementation.** An interesting feature of the SVM problem is the volume of data, which can lead to quadratic programs with between 10 and 100 million variables and a $Q$ matrix that would be dense if formed explicitly. Quadratic programming codes explicitly using the $Q$ matrix will not work well for these problems. We need a method for which we can utilize specialized linear algebra. Therefore, we use the Mehrotra predictor-corrector algorithm [23] as implemented in OOQP [11] as the basis for our interior-point method. The OOQP code is written in such a way that we can easily tailor the linear algebra to the application. This feature can be exploited to enable the solution of large data mining problems.

The linear algebra outlined in section 3 is used in our implementation. As mentioned in section 3, we simultaneously solve systems of equations involving different right-hand side vectors and also 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 subsections along with further information on selecting a starting point. Clearly, the fact that interior-point algorithms typically require only a small number of iterations is crucial for performance.

The scaling of the problem can affect the behavior of the numerical linear algebra used to calculate the solution. For example, we experimented with the substitution $\tilde{x} = \frac{x}{\nu}$ in (2.8) when $\nu > 1$. This helped to some extent when the standard OOQP starting point was used, but was not necessary with the special starting point described next.

**4.1. Starting point.** The starting point chosen for the method can significantly impact both the theoretical and practical performance of the algorithm. To achieve flexibility in the starting-point choice, we use the augmented system in (3.2) that has removed the bounds on $x$.

We know that the majority of the variables are zero at a solution to the SVM problem because the zero variables correspond to those observations correctly classified. Therefore, the starting point uses $x^0 = 0$ and $\lambda^0 = 0$. We choose $w^0$ and $v^0$ so that $w^0 - v^0 = c$. That is, the residual in the first equation of (3.2) at the starting point is zero. Since $c = -e$ for the SVM, we set $w^0 = (\nu + 1)e$ and $v^0 = (\nu + 2)e$. We are then left with a choice for the slack variables, $y^0$ and $z^0$, added to the augmented system for $w$ and $v$. To retain parity with our choice for $w^0$ and $v^0$, we set $y^0 = (\nu + 2)e$ and $z^0 = (\nu + 1)e$. We use the same starting point for the formulations without upper bounds but note that $v$ and $z$ are removed from the problem.

Better numerical performance might be achieved by the algorithm with an alternative starting point. For example, we expect that at a solution, most elements of $y$ will be zero and most elements of $z$ will be $\nu$. This fact is not reflected in the current choice of $y^0$. We did not perform any further investigation of this topic.

**4.2. Data issues.** Consider a model with 50 million observations and suppose there are 35 features, each represented by a 1-byte quantity. Then, the observation matrix $R$ is $50,000,000 \times 35$ and 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. Thus, 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 on today's machines.

We must attempt to perform most, if not all, of the operations using data kept out of core, while still achieving adequate performance. 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 $M = H^{-1} + R^T C^{-1} R$ (see step 2 of Algorithm SMW). We begin by splitting $R$ and $C^{-1}$ into $p$ buffers of data and calculate

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

Note that $C$ 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 $(C^{-1})_1$ from disk, and set $M = H^{-1}$.
2. For $j = 1$ to $p - 1$ do
   (a) Wait for $R_j$ and $C_j^{-1}$ to finish loading.
   (b) Request $R_{j+1}$ and $C_{j+1}^{-1}$ from disk.
   (c) Accumulate $M = M + R_j^T (C^{-1})_j R_j$.
3. Wait for $R_p$ and $C_p^{-1}$ to finish loading.
4. Accumulate $M = M + R_p^T (C^{-1})_p R_p$.

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 as possible 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 is not as fast as an in core implementation. In section 5, we quantify the impact of the out of core calculation.

**4.3. Numerical considerations.** Because of 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. A naive implementation of Algorithm SMW that does nothing to address these problems results in divergence of the interior-point method for a moderately sized problem with one million observations. In an attempt to limit the effect of these numerical errors, we use a combination of aggregation to identify a block and bucketing within the block for the computations.

**4.3.1. Aggregation.** Consider the construction of the matrix $H^{-1} + R^T C^{-1} R$ using the above technique. The aggregation technique accumulates the $R_j^T (C^{-1})_j R_j$ components in temporary matrices, $M_l$ for $l = 1, \ldots, L$, and then merges these as $M = \sum_{l=1}^{L} M_l$. Specifically, the initialization and accumulation steps are updated from the algorithm above into the following final form.

1. Request $R_1$ and $(C^{-1})_1$ 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 $C_j^{-1}$ to finish loading.
   (b) Request $R_{j+1}$ and $C_{j+1}^{-1}$ from disk.
   (c) Accumulate $M_{(j \mod L)+1} = M_{(j \mod L)+1} + R_j^T (C^{-1})_j R_j$.
3. Wait for $R_p$ and $C_p^{-1}$ to finish loading.
4. Accumulate $M_{(p \mod L)+1} = M_{(p \mod L)+1} + R_p^T (C^{-1})_p R_p$.
5. Merge $M = \sum_{l=1}^{L} M_l$.

Our merge is implemented by repeatedly adding the $\frac{L}{2}$ neighbors as depicted in Figure 4.1 (termed pairwise summation in [15]). 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 on the buffer size read from disk. This dependency is removed in the code by further partitioning $R_j$ and $C_j^{-1}$ into smaller buffers with 50,000 elements. This is a heuristic to limit the size of the intermediate summation values without having to perform an expensive sorting operation.

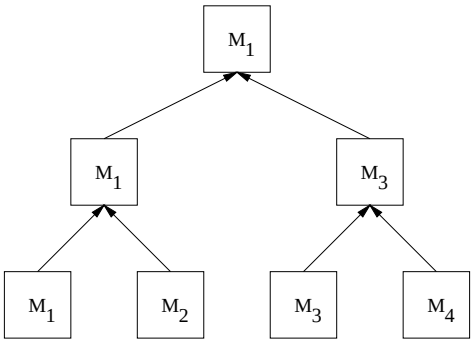


FIG. 4.1. *Accumulation diagram.*

**4.3.2. Bucketing.** While aggregation accumulates small batches of results, the bucketing strategy accumulates results of the same order of magnitude. The code uses 11 buckets with the ranges listed in Table 4.1. Whenever a result needs to be accumulated, it is assigned to the appropriate bucket. At the end of the computation, the buckets are merged. We decided to add first the positive and negative buckets of the same magnitude, and then accumulate the buckets starting with the smallest in magnitude. Again, this is a heuristic that does not require a sort of the data being accumulated. For summations involving numbers of the same sign, the accumulation from smallest to largest is as recommended in [29]. The addition of the positive and negative buckets of the same magnitude is designed to alleviate cancellation effects.

An example is given in [15] to test the effects of ordering on summations. The example has a large value $M$ such that in floating-point arithmetic $1 + M \equiv M$, with the requirement that the values $1, 2, 3, 4, M, -M$ should be summed. Our bucketing and summation scheme results in the following summation:

$$(((1 + 2) + 3) + 4) + (M - M).$$

Furthermore, the correct result is calculated independent of the initial ordering of the values, and no sort is required. Further details on other orderings and examples can be found in [15].

Since some of our calculations have mixed signs and some involve just positive numbers, the combination of heuristics, aggregation to identify a block and bucketing within each block, was found to be very effective.

TABLE 4.1
*Bucket ranges.*

| | Range | |
|---|---|---|
| Bucket | Lower bound | Upper bound |
| 1 | $-\infty$ | $-10^8$ |
| 2 | $-10^8$ | $-10^4$ |
| 3 | $-10^4$ | $-1$ |
| 4 | $-1$ | $-10^{-4}$ |
| 5 | $-10^{-4}$ | $-10^{-8}$ |
| 6 | $-10^{-8}$ | $10^{-8}$ |
| 7 | $10^{-8}$ | $10^{-4}$ |
| 8 | $10^{-4}$ | 1 |
| 9 | 1 | $10^4$ |
| 10 | $10^4$ | $10^8$ |
| 11 | $10^8$ | $\infty$ |

**4.4. Termination criteria.** The termination criterion is based on the inf-norm of the Fischer–Burmeister function [9] for the complementarity problem (3.1), with an appropriate modification for the presence of equations [8]. If we denote all variables in (3.1) by $x$ and the affine function on the left of (3.1) by $F(x)$, then each component of the Fischer–Burmeister function is defined by

$$\phi(x_i, F_i(x)) := \sqrt{x_i^2 + F_i(x)^2} - x_i - F_i(x)$$

for those variables with lower bounds of zero and by $\phi(x_i, F_i(x)) = -F_i(x)$ for variables without bounds. We can see from this definition that $\phi(x_i, F_i(x)) = 0$ if and only if the complementarity relationship is satisfied between $x_i$ and $F_i(x)$. 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 (3.1). We further note that the function $F$ can be evaluated during the calculation of the right-hand side in the predictor step. Therefore, the function calculation does not cost an additional pass through the data. We use a termination criterion of $10^{-6}$ for the Fischer–Burmeister function within the code, which is much more stringent than the default criterion for OOQP. In Figure 4.2 we plot the (log) residual as a function of the iteration for problem (2.8) with 10 million observations.

We terminate unsuccessfully whenever the iteration limit is reached or we fail to achieve a decrease in the residual for satisfying the equations in the interior-point method for six consecutive iterations and the complementarity residual $(w^T y + v^T z)$ is less than $\frac{10^{-15}}{2m}$.

The machine learning community sometimes terminates an algorithm based upon conditions other than optimality, such as tuning set accuracy [20]. Similar criteria could be used within our code, but we prefer to terminate at an optimal solution to the quadratic program.

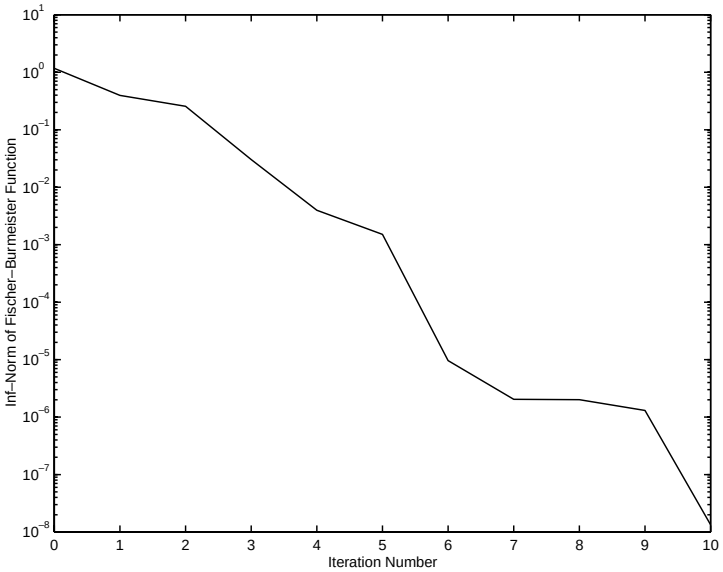


FIG. 4.2. *Log residual as a function of iterations for problem* (2.8) *with* 10 *million observations.*

**5. Computational results.** All of the tests were run on a 296 MHz Sun Ultrasparc with two processors and 768 megabytes of RAM. We stored all data on a locally mounted disk with 18 gigabytes of storage space available. This disk is not backed up. This setup prevents all overhead due to network communication and disk contention with nightly backups. Since the disk is not dedicated, our results reflect some effects due to contention with other users.

The asynchronous I/O routines are implemented using threads. Thus, both of the processors can be used for the tests. However, the workstation is shared by many individuals. During our tests the second processor was typically running a different user's jobs. Further results on a uniprocessor machine indicate that the impact of the second processor is minimal.

**5.1. Data sets.** For experimentation, we generated a separable, random data set with 34 features. 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 the data was generated, with $De$ being encoded as an additional column to the observation set. Each of the feature measurements is a 1-byte quantity. A nonseparable dataset was constructed by randomly changing the classification of the observations with a 1% probability. The nonseparable dataset has exactly 600,108 misclassified observations.

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.

**5.2. Out of core impact.** The impact on performance of using an out of core implementation was tested by using the formulation in (2.10) with $\nu = 1$ on the separable dataset. Since $S$ is positive definite in this case, no proximal-point modification was added.

The first property investigated was the effect of out of core computations on performance using asynchronous I/O. To test the performance, we ran problems for sizes varying between 200,000 and 1 million observations. A data 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 five times and used the minimum values in the figures. The average time per iteration is reported in Figure 5.1 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 note only an 8.2–9.9% increase in time over the in core implementation for the chosen buffer and problem sizes. Synchronous I/O results in a 9.4–13.1% increase. For both of these tests the maximum percentage increase in time occurred with a problem size of 800,000 elements. We conclude that an out of core implementation of the algorithm uses limited memory but results in increased time. We believe that the enormous decrease in the amount of RAM used for a less than 10% increase in time is a reasonable tradeoff to make. A case can also be made for using the easier to implement synchronous I/O.

The next set of experiments was designed to determine the impact of modifying the file buffer size. For these tests, we fixed the problem size to 1 million observations and varied the file buffer size from 50,000 to 500,000 elements. The average time per iteration is plotted in Figure 5.2. The results indicate that a file buffer size of around 250,000 elements is optimal with a 9.0% increase in time over the in core solution. The total amount of data buffered in main memory is between 110 and 152 megabytes

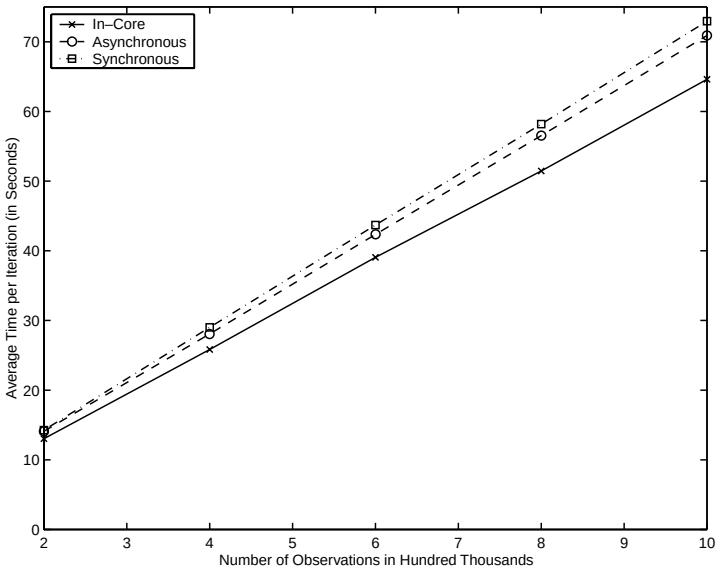


FIG. 5.1. *Average time per iteration for various problem sizes with a fixed file buffer size of* 100,000 *elements.*

depending on the problem formulation used. Based on these results, we decided to use asynchronous I/O and a buffer size of 250,000 elements for the remainder of the numerical experiments.

**5.3. Baseline comparison.** We next investigated the performance of our interior-point algorithm compared with other methods from the machine learning community.

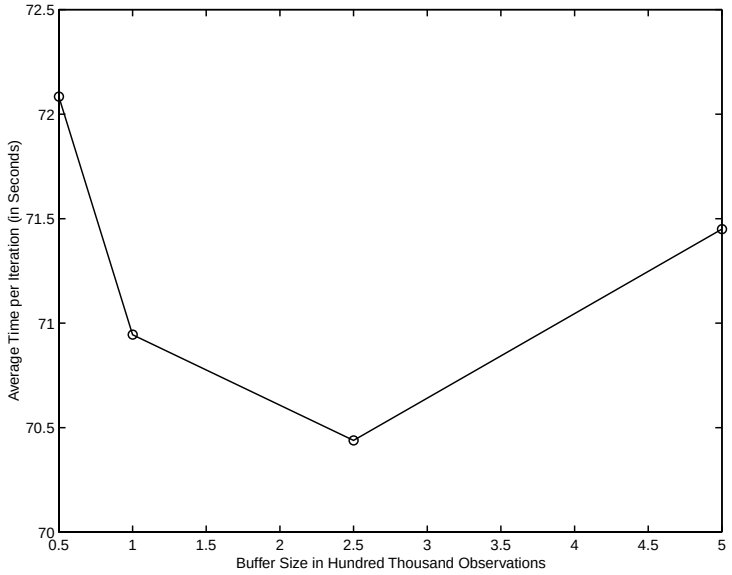


FIG. 5.2. *Average time per iteration for various file buffer sizes with a fixed problem size of* 1,000,000 *observations.*

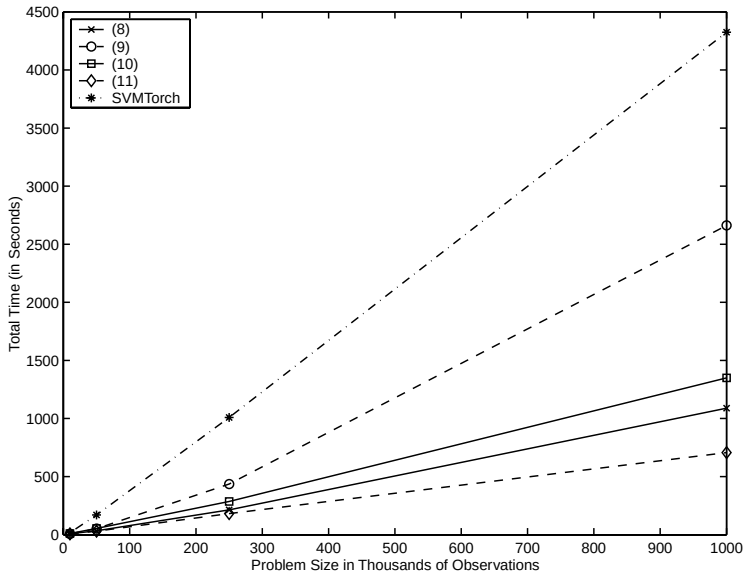


FIG. 5.3. *Total time comparison of the different formulations and SVMTorch with varying problem sizes on the separable dataset.*

We used SVMTorch [4] for this series of tests because the code is freely available and, according to the documentation, is specifically tailored for large-scale problems. We compiled both codes with the same compiler and options and converted the datasets into the binary format requested by SVMTorch. We ran SVMTorch using a linear kernel with $\nu = 1$. All other options, including the termination tolerance, were set to their default values.

Figure 5.3 reports the total running time for SVMTorch and each of (2.8)–(2.11) on the separable dataset with various numbers of observations. From these results, SVMTorch is 1.6–6.1 times slower than our codes on the separable dataset depending on the size and formulation chosen.

Results on the nonseparable dataset are more dramatic. SVMTorch took 1156.3 seconds to find a solution with 10,000 observations. Our interior-point codes took 5.8, 5.8, 8.6, and 9.8 seconds with formulations (2.8)–(2.11), respectively. These numbers indicate that SVMTorch is between 116 and 196 times slower on the nonseparable dataset. The magnitude becomes even larger when the number of observations is increased. With 50,000 observations, we let SVMTorch spend over 15 hours of CPU time in 380,000 iterations before terminating the SVMTorch code with a "current error" of 2.15. These numbers indicate that the SVMTorch code is at best more than 1,060 times slower than our interior-point code on this particular dataset. We did not perform any further tests with this code.

**5.4. Sensitivity to $\nu$.** The next set of experiments was to determine the sensitivity of the method to increases in $\nu$. In all of these tests, a proximal-perturbation of $\eta = 10^{-5}$ was added for the models in (2.8) and (2.9) when the error in solving the equations increased. We report in Figures 5.4 and 5.5 the number of iterations taken by the interior-point methods for various values of $\nu$ between 1 and 10,000 on the separable and nonseparable datasets with 1 million observations, respectively. On the separable dataset we notice increases in the number of iterations taken to find a

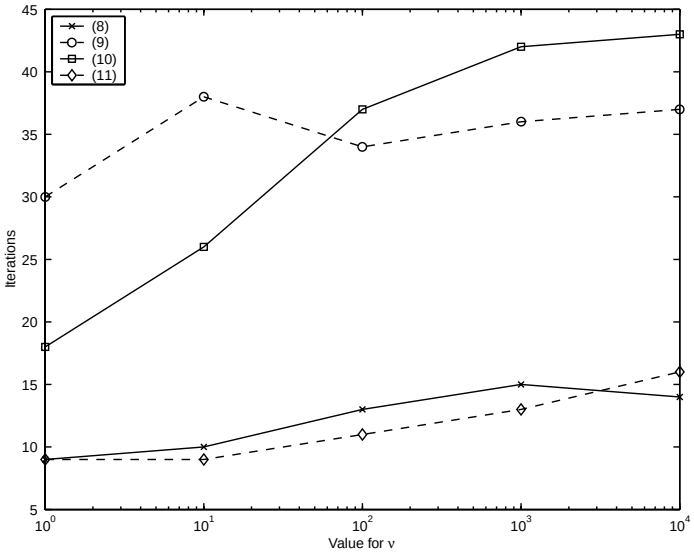


FIG. 5.4. *Iteration comparison of the different formulations with varying $\nu$ on the separable dataset.*

solutions. The iteration counts taken on the nonseparable dataset also increase. For (2.8) and (2.9), a small percentage of the variables at the solution were at the upper bound for all values of $\nu$ in both the separable and nonseparable tests. We note that on the nonseparable dataset for $\nu = 1,000$ and $\nu = 10,000$, all of the formulations failed to achieve termination tolerances; they stopped with final and best residuals reported in Table 5.1. Further improvements to the linear algebra implementation

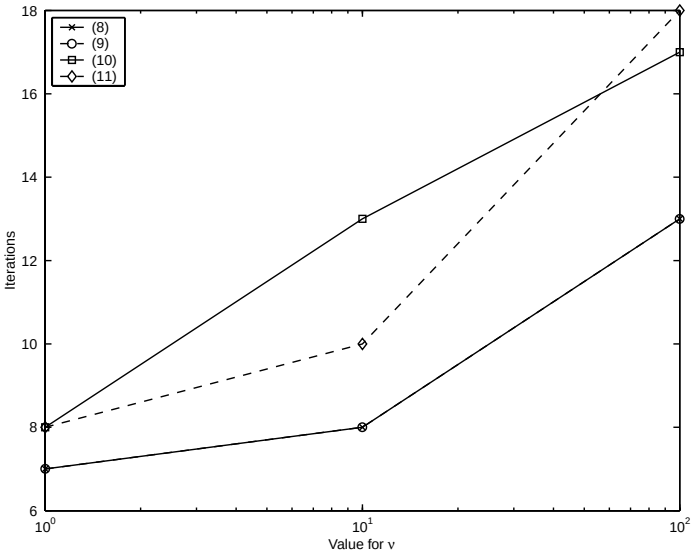


FIG. 5.5. *Iteration comparison of the different formulations with varying $\nu$ on the nonseparable dataset.*

TABLE 5.1
*Final and best residuals reported for the different formulations with $\nu = 1,000$ and $\nu = 10,000$ on the nonseparable dataset.*

| | $\nu = 1,000$ | | $\nu = 10,000$ | |
|---|---|---|---|---|
| Formulation | Final | Best | Final | Best |
| (2.8) | 2.12e-6 | 1.38e-6 | 2.78e-5 | 1.71e-5 |
| (2.9) | 2.65e-6 | 1.36e-6 | 1.84e-5 | 1.04e-5 |
| (2.10) | 5.73e-6 | 3.76e-6 | 6.87e-5 | 3.50e-5 |
| (2.11) | 8.57e-6 | 3.93e-6 | 3.73e-5 | 2.60e-5 |

would need to be investigated in order to produce reasonable results for larger values of $\nu$.

**5.5. Massive problems.** The final set of experiments was designed to determine the reliability of the algorithm on the various formulations and the scalability of the implementation to massive problems. Specifically, we varied the problem size between 1 and 60 million observations. In all of these tests $\nu = 1$ was used, and for the models in (2.8) and (2.9) a proximal-perturbation of $\eta = 10^{-5}$ was added when the error in solving (3.3) increased.

Each model was run one time with problem sizes of 1, 5, 10, 20, and 60 million observations. We plot average time per iteration in Figure 5.6 and number of iterations as functions of problem size in Figures 5.7 and 5.8, respectively. The similarity in the average time per iteration between formulations (2.10) and (2.11) (and also between (2.8) and (2.9)) is indistinguishable. To avoid clutter, we plot the results only for (2.8) and (2.11) in Figure 5.6. The total times are reported in Figures 5.9 and 5.10.

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 $H^{-1} + R^T C^{-1} R$. The number of floating-point operations necessary

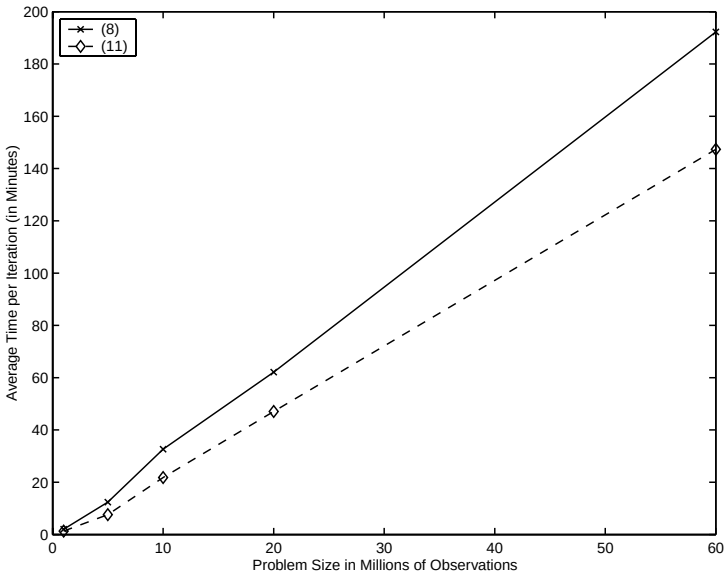


FIG. 5.6. *Average time per iteration comparison of the different formulations with varying problem size.*

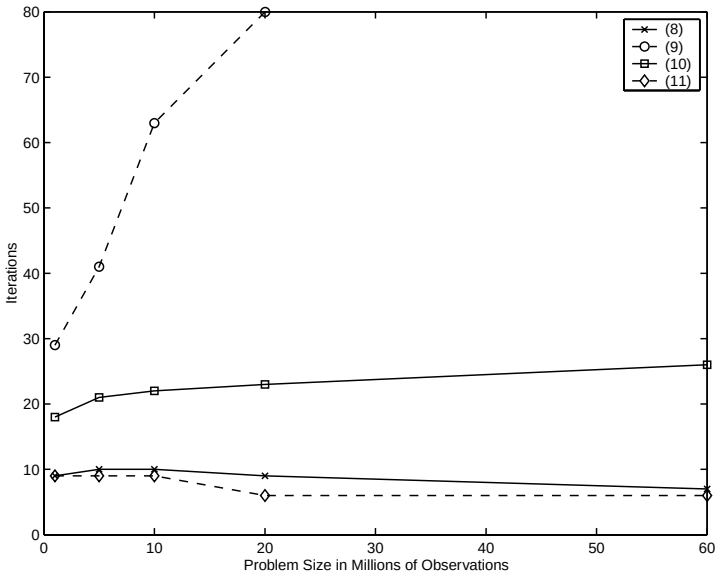


FIG. 5.7. *Iteration comparison of the different formulations with varying problem size on the separable dataset.*

to calculate this quantity grows linearly with problem size $m$ (but quadratically with the number of features $k$). The extra time needed for (2.8) is due to the treatment of upper bounds.

A surprising result for the constrained formulations, (2.8) and (2.11), on the separable dataset is that the number of iterations remains fairly flat as the problem

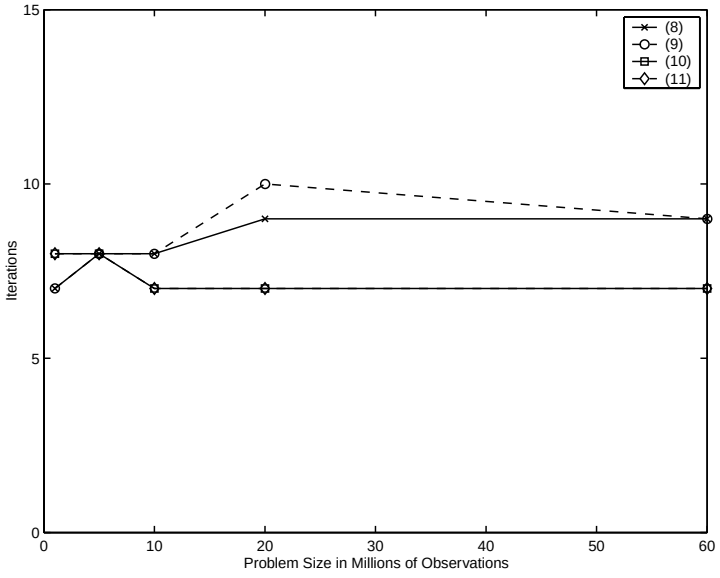


FIG. 5.8. *Iteration comparison of the different formulations with varying problem size on the nonseparable dataset.*

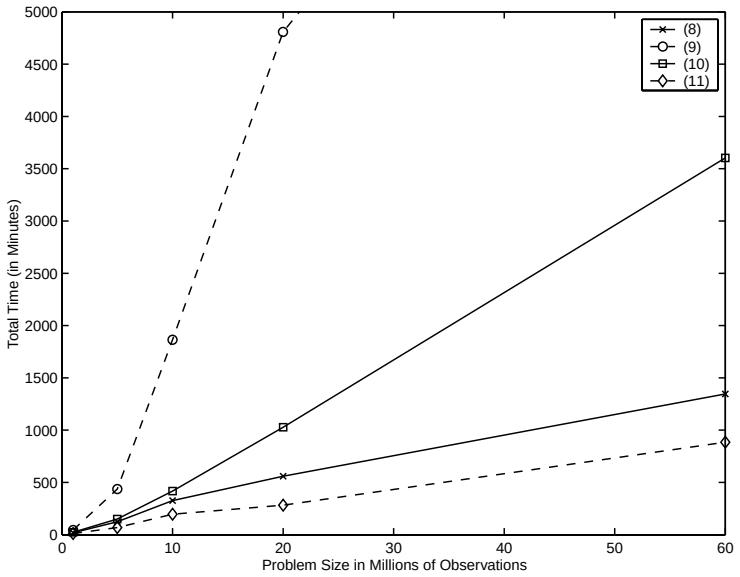


FIG. 5.9. *Total time comparison of the different formulations with varying problem size on the separable dataset.*

size increases and even decreases for some of the larger problems. As expected, the number of iterations taken for (2.9) and (2.10) increases with the dimension of the problem. We note a large difference in the number of iterations taken to converge for formulations (2.8) and (2.9) even though the problems are similar. The main reason for this difference is that many small steps are taken when solving (2.9).

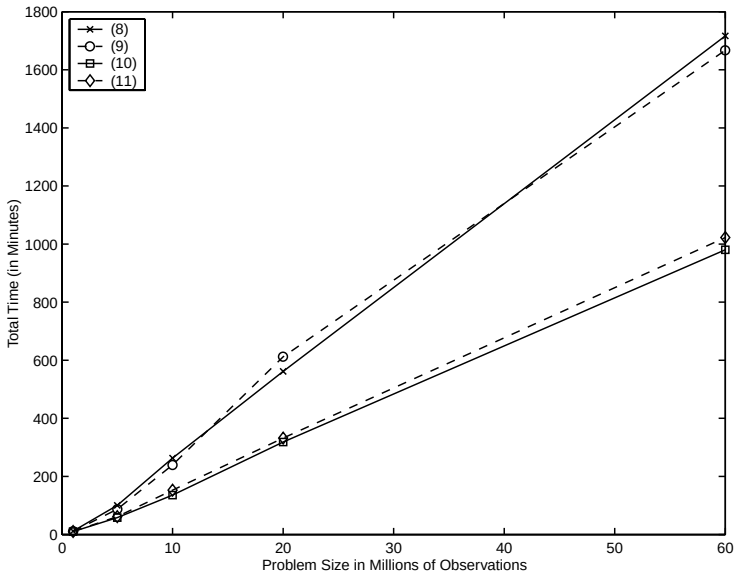


FIG. 5.10. *Total time comparison of the different formulations with varying problem size on the nonseparable dataset.*

All of the formulations performed extremely well on the nonseparable dataset, with very little variation in the number of iterations. These facts are counterintuitive and are probably related to the random nature of the model. However, more tests on "real" datasets need to be performed before drawing any firm conclusions.

The constrained formulations (2.8) and (2.11) appear to be the most tractable for interior-point methods. Both of these formulations solved the 60-million observation problem in 15–22.5 hours on a standard workstation. Formulations (2.9) and (2.10) also work for the 60-million observation problem but take longer times.

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 elected to use the same defaults on all problems and have not encountered any numerical difficulties beyond those documented in section 5.4.

**6. Conclusions.** We have developed an interior-point code for solving several quadratic programming formulations of the linear SVM. 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.

The linear algebra can be parallelized easily, and further speedups can be realized through storage of the data across multiple disks. More sophisticated corrector implementations [14] of the interior-point code can be used to further reduce the iteration count. These are topics for future work, along with extensions to nonlinear SVM, and techniques to further reduce the number of data scans.

**Acknowledgments.** We thank Olvi Mangasarian for bringing this problem to our attention; Yuh-Jye Lee and David Musicant for numerous insightful discussions on machine learning problems; Mike Gertz, Jeff Linderoth, and Stephen Wright for making a preliminary version of OOQP available to us; and Michael Saunders and two anonymous referees for encouraging us to further explore the numerical properties of the code.

REFERENCES

[1] P. S. Bradley and O. L. Mangasarian, *Massive data discrimination via linear support vector machines*, Optim. Methods Softw., 13 (2000), pp. 1–10.
[2] C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, 2 (1998), pp. 121–167.
[3] W. G. Cochran, *Sampling Techniques*, 3rd ed., John Wiley and Sons, New York, 1977.
[4] R. Collobert and S. Bengio, *SVMTorch: Support vector machines for large-scale regression problems*, J. Mach. Learn. Res., 1 (2001), pp. 143–160.
[5] C. Cortes and V. Vapnik, *Support vector networks*, Machine Learning, 20 (1995), pp. 273–297.
[6] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, Cambridge, UK, 2000.
[7] M. C. Ferris, *Finite termination of the proximal point algorithm*, Math. Program., 50 (1991), pp. 359–366.
[8] M. C. Ferris, C. Kanzow, and T. S. Munson, *Feasible descent algorithms for mixed complementarity problems*, Math. Program., 86 (1999), pp. 475–497.
[9] A. Fischer, *A special Newton-type optimization method*, Optimization, 24 (1992), pp. 269–284.
[10] 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, ACM Press, New York, 1999, pp. 126–137.
[11] M. Gertz and S. Wright, *Object-Oriented Software for Quadratic Programming*, Preprint ANL/MCS-P891-1000, Argonne National Laboratory, Argonne, IL, 2001.

[12] M. Gertz and S. Wright, *OOQP User Guide*, Technical Memorandum ANL/MCS-TM-252, Argonne National Laboratory, Argonne, IL, 2001.
[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[14] J. Gondzio, *Multiple centrality corrections in a primal-dual method for linear programming*, Comput. Optim. Appl., 6 (1996), pp. 137–156.
[15] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.
[16] P. J. Huber, *Robust Statistics*, John Wiley and Sons, New York, 1981.
[17] W. Li and J. J. Swetits, *The linear $\ell_1$ estimator and the Huber M-estimator*, SIAM J. Optim., 8 (1998), pp. 457–475.
[18] O. L. Mangasarian, *Nonlinear Programming*, McGraw–Hill, New York, 1969; Classics Appl. Math. 10, SIAM, Philadelphia, 1994.
[19] O. L. Mangasarian, *Generalized support vector machines*, in Advances in Large Margin Classifiers, A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, eds., MIT Press, Cambridge, MA, 2000, pp. 135–146.
[20] O. L. Mangasarian and D. R. Musicant, *Successive overrelaxation for support vector machines*, IEEE Transactions on Neural Networks, 10 (1999), pp. 1032–1037.
[21] O. L. Mangasarian and D. R. Musicant, *Robust linear and support vector regression*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 950–955.
[22] O. L. Mangasarian and D. R. Musicant, *Active set support vector machine classification*, in Advances in Neural Information Processing Systems 13, T. K. Leen, T. G. Dietterich, and V. Tresp, eds., MIT Press, Cambridge, MA, 2001, pp. 577–583.
[23] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM J. Optim., 2 (1992), pp. 575–601.
[24] J. Platt, *Sequential minimal optimization: A fast algorithm for training support vector machines*, in Advances in Kernel Methods—Support Vector Learning, B. Schölkopf, C. J. C. Burges, and A. J. Smola, eds., MIT Press, Cambridge, MA, 1999, pp. 185–208.
[25] R. T. Rockafellar, *Monotone operators and the proximal point algorithm*, SIAM J. Control Optim., 14 (1976), pp. 877–898.
[26] B. Schölkopf, C. Burges, and A. Smola, eds., *Advances in Kernel Methods: Support Vector Machines*, MIT Press, Cambridge, MA, 1998.
[27] A. J. Smola, *Learning with Kernels*, Ph.D. thesis, GMD Research Series 25, Technische Universität Berlin, Germany, 1998.
[28] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed., Springer-Verlag, New York, 2000.
[29] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.
[30] S. J. Wright, *Primal–Dual Interior–Point Methods*, SIAM, Philadelphia, 1997.
[31] S. J. Wright, *On reduced convex QP formulations of monotone LCP problems*, Math. Program., 90 (2001), pp. 459–474.