Comput Optim Appl (2010) 47: 431–453
DOI 10.1007/s10589-008-9228-z

Using an iterative linear solver in an interior-point method for generating support vector machines

E. Michael Gertz · Joshua D. Griffin

Received: 21 February 2006 / Revised: 13 August 2008 / Published online: 6 January 2009

Abstract This paper concerns the generation of support vector machine classifiers for solving the pattern recognition problem in machine learning. A method is proposed based on interior-point methods for convex quadratic programming. This interior-point method uses a linear preconditioned conjugate gradient method with a novel preconditioner to compute each iteration from the previous. An implementation is developed by adapting the object-oriented package OOQP to the problem structure. Numerical results are provided, and computational experience is discussed.

Keywords Machine learning · Support vector machines · Quadratic programming · Interior-point methods · Krylov-space methods · Matrix-free preconditioning

1 Introduction

Researchers have expressed considerable interest in the use of support vector machine (SVM) classifiers in pattern recognition problems (see Burges [6]; Cristianini and Shawe-Taylor [8]; and Vapnik [35].) The problem of generating an SVM classifier can be reduced to one of solving a highly structured convex quadratic program. This quadratic program can be very large, and one must exploit the structure of the problem to solve it efficiently.

Several methods for solving the SVM quadratic subproblem have been developed. Active-set methods, such as those studied by Osuna et al. [29], Joachims [24],

E.M. Gertz (✉)
University of Wisconsin, Madison, USA
e-mail: emgertz@mac.com

J.D. Griffin
Computational Sciences and Mathematical Research Division, Sandia National Laboratories, Livermore, CA 94551, USA
e-mail: Joshua.Griffin@sas.com

Platt [31, 32], and Keerthi et al. [25] are popular, along with the more recent approaches Fan et al. [11], Glasmachers et al. [20], Dong et al. [37], and Scheinberg [33]. Other methods have also been proposed, such as the Lagrangian methods of Mangasarian and Musicant [28] and the semismooth methods of Ferris and Munson [13]. Jung et al. [23] employ an approach that uses adaptive constraint reduction to reduce the number of constraints considered at each iteration.

Recently, Ferris and Munson [12] have shown how to efficiently solve large problems, with millions of observations, using a primal-dual interior-point algorithm and specialized linear algebra. They vastly reduce the size of the linear systems to be solved by using the Sherman-Morrison-Woodbury (SMW) formula (see [21]), which is equivalent to using the Schur complement arising in a block row-reduction of the linear system. In this paper, we take a similar approach in formulating the SVM problem.

Interior-point methods based on the SMW formulation are relatively insensitive to the number of observations [12]. However, the computational cost of using these methods grows at least quadratically in the number of observed features. We present a new technique for reducing the run time of interior-point methods that use the SMW formulation when the number of observed features grows moderately large. In doing so, we increase the range of problems to which the SMW formulation may profitably be employed.

In Sect. 2, we briefly review the theory of support vector machines. The use of interior-point methods to generate SVM classifiers is considered in Sect. 3. The proposed matrix-free iterative method is outlined in Sect. 5. Numerical experiments with a specific implementation using the object-oriented QP solver OOQP [17, 18] are described in Sect. 6.

2 Support vector machines

A learning machine finds a mapping, known as a classifier, between a population of objects and a set of labels. For the pattern recognition problem, the labels are "yes" and "no," which we represent here as 1 and -1. A support vector machine is a specific type of learning machine for the pattern recognition problem. The simplest SVM generates a linear classifier—an affine function $x \mapsto w^T x - \beta$ that is used to define the classifier

$$x \mapsto \begin{cases} 1 & \text{if } w^T x - \beta \geq 0; \\ -1 & \text{otherwise.} \end{cases}$$

Classifiers are created by determining appropriate values of w and β by observing the features of a training set, a subset of the population that has a known classification. Let n denote the number of observations in the training set. Let m be the number of features in each observation vector x_i, and let $d_i \in \{-1, 1\}$ indicate its classification. Let X denote the $n \times m$ matrix whose rows are the observations x_i; in other words, $X^T = (x_i \cdots x_n)$. Similarly, let D denote the $n \times n$ diagonal matrix $\text{diag}(d)$. Then, a linear SVM classifier may be created by finding w and β that solve the minimization problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|_2^2 + \tau e^T z \\ \text{subject to} & D(Xw - \beta e) \geq e - z, \\ & z \geq 0, \end{array} \tag{1}$$

where e is a vector of all ones, z is a vector of appropriate size, and τ is a positive constant that is a parameter of the problem.

This formulation may be motivated by regarding $e^T z$ as a measure of the misclassification of the training set by the generated classifier. The term $\tau e^T z$ is known as an ℓ_1 penalty function in the theory of constrained optimization. A well-known property of the ℓ_1 penalty functions is that if there are values of w and β that separate the training data correctly, then these values will be the solution of the optimization problem for all sufficiently large τ (see, e.g., Fletcher [15]). It is easily shown that the distance between the two hyperplanes $x_i^T w - \beta = 1$ and $x_i^T w - \beta = -1$ is given by $2/\|w\|_2$ (see e.g. Burges [6]). Thus the objective in the optimization problem (1) can be seen as a balance between trying to minimize the empirical misclassification error and trying to maximize the separation margin. (See Vapnik [35] for a discussion of the composite objective and of why a larger separation margin may improve the generalization capability of the classifier.)

The dual of the problem (1) is

$$\begin{array}{ll} \text{minimize} & -e^T v + \frac{1}{2} v^T D X X^T D v \\ \text{subject to} & e^T D v = 0, \;\; 0 \leq v \leq \tau e. \end{array} \tag{2}$$

For the primal-dual methods described in Sect. 3, there is little difference between the primal (1) and dual (2) formulations. It is not hard to see that by making r_w identically zero in (5) and eliminating Δw from the system, one may obtain a primal-dual iteration on the dual problem (2). However, the constraints of the dual problem are mainly simple bounds, a fact that has been used to great effect in a number of algorithms, notably the chunking algorithms introduced by Osuna et al. [29].

The dual formulation has also been used to generalize the classical linear problem (1). This generalization involves replacing the product $DXX^T D$ in (2) by a matrix Q such that $q_{ij} = d_i \mathcal{K}(x_i, x_j) d_j$, where $\mathcal{K}$ is a given kernel function $\mathcal{K} : \Re^n \times \Re^n \mapsto \Re$. This yields a problem of the form

$$\begin{array}{ll} \text{minimize} & -e^T v + \frac{1}{2} v^T Q v \\ \text{subject to} & e^T D v = 0, \;\; 0 \leq v \leq \tau e. \end{array} \tag{3}$$

The $n \times n$ matrix Q is large and typically dense, making it inefficient to apply a primal-dual iteration naively to (3). Under suitable conditions, however, the use of a kernel function is equivalent to defining a transformation $\Phi(x)$ that maps the data into a larger, possibly infinite-dimensional feature space and finding a separating hyperplane in this space (see Burges [6] or Cristianini and Shawe-Taylor [8] for details). For some kernels, particularly polynomial kernels, the mapping $\Phi(x)$ is not hard to define. For other kernels, it may be possible to find a low-rank approximation to $\Phi(x)$ for use in a primal-dual method. Low-rank approximations are an active field of research; see for example A. J. Smola and B. Schölkopf [34], Fine and Scheinberg [14], Bach and Jordan [3], Louradour et. al. [27], and Drineas and Mahoney [10].

3 Interior-point methods

The problem (1) has a convex quadratic objective and only linear constraints. A general class of methods that have proven effective in solving such a problem is the class of interior-point methods. For a discussion of such methods, see Wright [36]. The method we derive here, previously described in Gertz and Wright [18], is the formulation used by the SVM module of OOQP. The method is similar to that derived in [12]. The preconditioned conjugate gradient method described below may be applied to either formulation.

As a general rule, primal-dual interior-point methods such as MPC operate by repeatedly solving Newton-like systems based on perturbations of the optimality conditions of the problem. For the SVM problem (1), these optimality conditions are

$$w - Y^T v = r_w = 0, \tag{4a}$$

$$d^T v = \rho_\beta = 0, \tag{4b}$$

$$\tau e - v - u = r_z = 0, \tag{4c}$$

$$Yw - \beta d + z - e - s = r_s = 0, \tag{4d}$$

$$ZUe = 0, \tag{4e}$$

$$SVe = 0, \tag{4f}$$

$$s,\ u,\ v,\ z \geq 0, \tag{4g}$$

where $Y = DX$ and S, U, V, and Z are diagonal matrices whose diagonals are the elements of the correspondingly named vector. These conditions are mathematically equivalent to those given in [12]. In (4b) we use ρ_β to denote the residual, rather than r_β, to emphasize that this quantity is a scalar.

The Newton system for (4a)–(4f) is

$$\Delta w - Y^T \Delta v = -r_w, \tag{5a}$$

$$d^T \Delta v = -\rho_\beta, \tag{5b}$$

$$-\Delta v - \Delta u = -r_z, \tag{5c}$$

$$Y\Delta w - d\Delta\beta + \Delta z - \Delta s = -r_s, \tag{5d}$$

$$Z\Delta u + U\Delta z = -r_u, \tag{5e}$$

$$S\Delta v + V\Delta s = -r_v, \tag{5f}$$

where $r_u = Zu$ and $r_v = Sv$. The MPC method solves systems with the same matrices, but for which the residuals r_u and r_v have been perturbed from their values in the Newton system.

The matrix of this system is large but sparse and highly structured. We use reductions similar to those described in Ferris and Munson [12] to reduce the system to a smaller dense system that may be efficiently solved by means of a Cholesky factorization. First, we eliminate the slack variables u and s from the system. Combining (5c) with (5e), and (5d) with (5f), we obtain the system

$$-\Delta v + Z^{-1}U\Delta z = -\hat{r}_z, \tag{6a}$$

$$Y\Delta w - d\Delta\beta + \Delta z + V^{-1}S\Delta v = -\hat{r}_s, \tag{6b}$$

where the residuals are defined to be $\hat{r}_z = r_z + Z^{-1}r_u$ and $\hat{r}_s = r_s + V^{-1}r_v$. We may eliminate Δz from this system to obtain

$$Y\Delta w - d\Delta\beta + \Omega\Delta v = -r_\Omega,$$

where we define

$$\Omega = V^{-1}S + U^{-1}Z \tag{7}$$

and $r_\Omega = \hat{r}_s - U^{-1}Z\hat{r}_z$.

In matrix form, the remaining equations are

$$\begin{pmatrix} I & 0 & -Y^T \\ 0 & 0 & d^T \\ Y & -d & \Omega \end{pmatrix} \begin{pmatrix} \Delta w \\ \Delta\beta \\ \Delta v \end{pmatrix} = -\begin{pmatrix} r_w \\ \rho_\beta \\ r_\Omega \end{pmatrix}.$$

Simple row eliminations yield the block-triangular system

$$\begin{pmatrix} I + Y^T\Omega^{-1}Y & -Y^T\Omega^{-1}d & 0 \\ -d^T\Omega^{-1}Y & d^T\Omega^{-1}d & 0 \\ Y & -d & \Omega \end{pmatrix} \begin{pmatrix} \Delta w \\ \Delta\beta \\ \Delta v \end{pmatrix} = -\begin{pmatrix} r_w + Y^T\Omega^{-1}r_\Omega \\ \rho_\beta - d^T\Omega^{-1}r_\Omega \\ r_\Omega \end{pmatrix}.$$

A final row-reduction may be used to solve for Δw and $\Delta\beta$. Let us introduce the notation

$$\hat{r}_w = r_w + Y^T\Omega^{-1}r_\Omega, \tag{8a}$$

$$\hat{\rho}_\beta = \rho_\beta - d^T\Omega^{-1}r_\Omega, \tag{8b}$$

$$y_d = Y^T\Omega^{-1}d, \tag{8c}$$

$$\sigma = d^T\Omega^{-1}d. \tag{8d}$$

The scalar σ is nonzero because Ω is positive definite and d is nonzero. Then Δw and $\Delta\beta$ may be found from the reduced system

$$\left(I + Y^T\Omega^{-1}Y - \frac{1}{\sigma}y_d y_d^T\right)\Delta w = -\left(\hat{r}_w + \frac{1}{\sigma}\hat{\rho}_\beta y_d\right) \tag{9a}$$

$$\Delta\beta = \frac{1}{\sigma}(-\hat{\rho}_\beta + y_d^T\Delta w). \tag{9b}$$

The value of Δv may be obtained by the forward-substitution

$$\Delta v = -\Omega^{-1}(r_\Omega + Y\Delta w - d\Delta\beta). \tag{10}$$

Equation (6a) may then be used to compute $\Delta z = -U^{-1}Z(\hat{r}_z - \Delta v)$. The values of Δu and Δs may be obtained through the equations $\Delta u = -Z^{-1}(r_u + U\Delta z)$ and $\Delta s = -V^{-1}(r_v + S\Delta v)$, which are derived from (5e) and (5f), respectively.

Numerical experience has shown that the cost of solving this system is typically dominated by the cost of forming the matrix

$$M = I + Y^T\Omega^{-1}Y - \frac{1}{\sigma}y_d y_d^T, \tag{11}$$

which is the coefficient matrix for (9a). In particular, if Y is dense, then $O(nm^2/2)$ multiplications are needed to compute M explicitly. The next most significant operations, in terms of multiplications, are the $O(nm)$ multiplications needed to compute products with Y^T and Y when computing the residuals of the optimality conditions (4), in computing $\hat{r}_w$ and y_d in the row-reduction (8), and in computing Δv through (10). If M is computed explicitly, it requires $O(m^3/6)$ multiplications to perform a Cholesky factorization of M. If m is much smaller than n, as is typical in many applications, the cost of this factorization is minor.

4 Active-set methods

A broad class of methods that is commonly used to generate SVMs is the class of active-set methods. The algorithms in this class differ considerably, but we call a method for generating an SVM an active-set method if it solves the dual problem (3) by performing iterations that explicitly fix a subset of the variables v at their current value. A variable that is on one of its bounds at the solution is called *active*. Active-set methods are based on the principle that if one knows which elements of v are active at the solution, then finding the value of the other elements becomes a linear algebra problem (see Fletcher [15] and Gill et al. [19]).

The set of variables explicitly fixed at an iteration of an active-set method is known as the *working set*. A classical active-set method searches through the space of all working sets to find the set of variables active at the solution. As such, the worst-case running time of an active-set method is inherently exponential in n, the number of observations. Modern variants of the active set method, sometimes known as *domain decomposition methods,* may add variables that are not on their bounds to the working set. Such methods do not necessarily terminate in a finite number of steps, because unlike active set methods they do not have a finite number of possible configurations. They do, however, limit the number of variables that change at each iteration, and may therefore significantly reduce the cost of each iteration. The algorithm used by the SVM$^{\text{light}}$ package is an example of such a method.

In practice, active-set methods are often able to find solutions quickly, while considering only a small subset of the search space of all possible working sets. They are also often able to use only large working sets at every iteration, leaving only a small subset of the variables off their bounds. Moreover, there are frequently iterates that provide a good solution, if not an optimal solution, and so active-set methods for SVMs may terminate early with an effective classifier. However, the number of possible working sets, even for those methods that only use large working sets, grows quickly in the number of observations, and an active-set method may require a large number of iterations.

By contrast, interior-point methods can work in provably polynomial time, though the method described here lacks the globalization technique necessary for this proof. In practice, however, neither active-set methods nor interior-point methods tend to behave according to their worst case bounds. The numerical experiments we present in Sect. 6 are consistent with the observation that the number of iterations of an interior-point method are only weakly dependent on problem size [12, 36]. Since the cost of an iteration of an interior-point method based on the SMW [12] or a low-rank update to a matrix inverse [14] is linear in the number of observations, interior-point methods are a promising alternative to active-set methods when the number of observations is large.

An advantage to active-set methods, however, is that the cost of a typical iteration is relatively small. Since active-set methods differ greatly in how they define an iteration, one can only speak in general terms about the cost per iteration. However, one may observe that variables that are fixed on their bounds may temporarily be eliminated from the problem (3), yielding a lower-dimensional subproblem. The number of features effects the cost of computing elements of Q, but the cost of computing an element of Q grows only linear in the number of features, whereas the cost of computing M grows quadratically. The method we propose here avoids part of the cost of using the SMW formulation by skipping, when possible, part of the computation for the dual variables that approach their bounds. How this is done is the topic of the next section.

It is worth repeating that active-set methods may be used directly with nonlinear kernels, whereas the interior-point method presented here cannot. This is further discussed in Sect. 7.

5 Preconditioned Krylov-space methods

As discussed in Sect. 3, the cost of computing the matrix M tends to dominate the overall computing time. Recall that the matrix M, introduced in (11), is the coefficient matrix of the linear system (9a). An alternative to computing M is to apply a matrix-free iterative method to solving the system (9a). Iterative techniques for solving linear systems, also known as Krylov-space methods, do not require that M be computed explicitly; they require only a mechanism for computing matrix-vector products of the form Mx.

A total of $O(2nm + n)$ multiplications is needed to compute the matrix-vector product Mx if the computation is done in the straightforward manner. On the other hand, $O(nm^2/2)$ multiplications are required to compute M explicitly. Thus, in order to gain any advantage from using an iterative method, the system must be solved in fewer than $m/4$ iterations. For small values of m, there is little hope that this will be the case, but if m is moderately large, then an iterative strategy can be effective. However, the effectiveness and efficiency of an iterative method strongly depend on the availability of an adequate preconditioner.

5.1 The SVM preconditioner

The Krylov-space method that we use is the preconditioned linear conjugate-gradient (PCG) method. We do not discuss the PCG method in detail, but rather refer the reader to Golub and Van Loan [21] and the references therein. A technique for defining a suitable preconditioner is to find a matrix that in some sense approximates the matrix of the linear system to be solved yet is inexpensive to factor using a direct method. In this section we describe a suitable preconditioner for the linear system (11). In Sect. 5.2, we investigate properties of this preconditioner that make it a good choice for this system.

A predictor-corrector interior method solves two linear systems per iteration with the same matrix but different right-hand sides. Thus, it might be supposed that a direct factorization of M has some advantage over a Krylov-space method because the factorization need be computed only once. To some degree, this supposition is true, but the apparent advantage of direct methods is partially negated by three facts. First, the preconditioning matrix used by the iterative method need be computed only once per iteration; thus the iterative method derives some benefit from solving multiple systems with the same matrix. Second, it is not necessary to solve both the predictor and corrector steps to the same accuracy. Third, the right-hand sides of the systems differ but are related. Therefore the computed predictor step can be profitably used as an initial guess for the combined predictor-corrector step.

The product $Y^T\Omega^{-1}Y$ may be written as the sum of outer products

$$Y^T\Omega^{-1}Y = \frac{1}{\omega_1}y_1y_1^T + \cdots + \frac{1}{\omega_n}y_ny_n^T. \tag{12}$$

Recall from (7) that

$$\omega_i = s_i/v_i + z_i/u_i.$$

The complementarity conditions (4e) and (4f) require that at the solution $v_is_i = 0$ and $u_iz_i = 0$ for all $i \in 1, \ldots, n$. A common regularity assumption, known as strict complementarity, is that $v_i + s_i > 0$ and $u_i + z_i > 0$ at the solution. For any i for which strict complementarity holds, the limit of ω_i as the iterations progress is necessarily zero or infinity.

It follows that in the later iterations of a primal-dual algorithm, the terms in the sum (12) have widely differing scales. A natural approximation to $Y^T\Omega^{-1}Y$ is obtained by either omitting terms in the sum (12) that are small or by replacing these small terms by a matrix containing only their diagonal elements. We have found that the strategy of retaining the diagonal elements is more effective.

Let $\mathcal{A}$ be the set of indices for which the terms in the sum (12) are large in a sense that we make precise below. An appropriate preconditioner for M, which we henceforth refer to as the SVM preconditioner, is then

$$P_{\mathcal{A}} = I - \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T + \sum_{i \in \mathcal{A}} \frac{1}{\omega_i} y_i y_i^T + \sum_{i \notin \mathcal{A}} \operatorname{diag}\left(\frac{1}{\omega_i} y_i y_i^T \right), \tag{13}$$

where $\tilde{y}_d = \sum_{i \in \mathcal{A}} (d_i / \omega_i) y_i$ and

$$\tilde{\sigma} = \begin{cases} \sum_{i \in \mathcal{A}} \omega_i^{-1} & \text{if } \mathcal{A} \text{ is nonempty;} \\ 1 & \text{otherwise.} \end{cases}$$

If the preconditioner is chosen wisely, the size of $\mathcal{A}$ may be significantly smaller than the number of observations, allowing $P_{\mathcal{A}}$ to be formed at significantly less cost than it would take to form M.

The PCG method is well defined and convergent if both M and preconditioning matrix $P_{\mathcal{A}}$ are positive definite. As the following propositions show, this is always the case.

Proposition 1 *Let $\mathcal{I}$ be a nonempty subset of the integers from 1 to n. Let $\widetilde{\Omega}$ be the diagonal matrix whose diagonal elements are ω_i for $i \in \mathcal{I}$. Let $\tilde{d}$ and $\widetilde{Y}$ be defined similarly, with the $\tilde{d}$ defined to be the vector whose elements are d_i for $i \in \mathcal{I}$ and $\widetilde{Y}$ the matrix whose rows are y_i for $i \in \mathcal{I}$. Then the matrix*

$$Q = \widetilde{Y}^T \widetilde{\Omega}^{-1} \widetilde{Y} - \left(\tilde{d}^T \widetilde{\Omega}^{-1} \tilde{d} \right)^{-1} \left(\widetilde{Y}^T \widetilde{\Omega}^{-1} \tilde{d} \right) \left(\widetilde{Y}^T \widetilde{\Omega}^{-1} \tilde{d} \right)^T$$

is positive semidefinite.

Proof Because $\widetilde{\Omega}^{-1}$ is positive definite, one may define the inner product

$$\langle x, y \rangle_\omega = x^T \widetilde{\Omega}^{-1} y$$

and associated inner-product norm $\|x\|_\omega = \sqrt{\langle x, x \rangle_\omega}$. For a vector v,

$$v^T Q v = \|\widetilde{Y} v\|_\omega^2 - \frac{\langle \widetilde{Y} v, \tilde{d} \rangle_\omega^2}{\|\tilde{d}\|_\omega^2}.$$

But by the Cauchy-Schwartz inequality $|\langle \widetilde{Y} v, \tilde{d} \rangle_\omega| \leq \|\widetilde{Y} v\|_\omega \|\tilde{d}\|_\omega$. It immediately follows that Q is positive semidefinite. □

Proposition 2 *For any index set $\mathcal{A}$, the matrix $P_{\mathcal{A}}$ defined by (13) is positive definite. Furthermore, the matrix M of (11) is positive definite.*

Proof The preconditioner, $P_{\mathcal{A}}$, has the general form

$$P_{\mathcal{A}} = I + \sum_{i \notin \mathcal{A}} \operatorname{diag}\left(\frac{1}{\omega_i} y_i y_i^T \right) + Q.$$

If $\mathcal{A}$ is empty, then $Q = 0$. If $\mathcal{A}$ is nonempty, then Q satisfies the conditions of Proposition 1. In either case Q is positive semidefinite. For any vector v, the product

vv^T is positive semidefinite and therefore has nonnegative diagonal elements. Thus $P_{\mathcal{A}}$ is the sum of I with several positive semidefinite matrices. It follows that $P_{\mathcal{A}}$ is positive definite.

If $\mathcal{A}$ contains all integers from 1 to n, then $P_{\mathcal{A}} = M$. Hence, it immediately follows that M is positive definite. □

Next we develop a rule for choosing the set $\mathcal{A}$ at each iteration. Consider the quantity

$$\mu = (v^T s + u^T z)/(2n),$$

and observe that by definition $v_i s_i \le \mu$ and $u_i z_i \le \mu$ for all $i = 1, \ldots, n$. For a typical convergent primal-dual iteration, we have

$$v_i s_i \ge \rho\mu \quad \text{and} \quad u_i z_i \ge \rho\mu \tag{14}$$

for some positive value ρ that is constant for all iterations. Thus those w_i that converge to zero typically converge at a rate proportional to μ. Similarly, those w_i that grow without bound typically grow at a rate proportional to $1/\mu$.

Based on the observations above, we choose $i \in \mathcal{A}$ if and only if

$$\frac{1}{\omega_i} y_i^T y_i \ge \gamma \min\left(1, \mu^{1/2}\right), \tag{15}$$

where the value γ is a parameter of the algorithm. Eventually the test will exclude all ω_i that are converging to zero at a rate proportional to μ.

Although one does not know a priori the most effective value for the parameter γ for a particular problem, we have found that an initial choice of $\gamma = 100$ works well in practice. Sometimes, however, typically at an early primal-dual iteration, the PCG method will converge slowly or will diverge because of numerical error. In such cases, we use a heuristic to decrease the value of γ and include more indices in $\mathcal{A}$. We describe this heuristic in Sect. 5.3. We emphasize that γ never increases during the solution of a particular SVM problem.

5.2 Analysis of the SVM preconditioner

For any positive-definite A, the worst-case convergence rate of the conjugate-gradient method is described by the inequality

$$\|x - x_k\|_2 \le 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k, \tag{16}$$

where $\kappa = \kappa_2(A) \triangleq \|A\|_2 \|A^{-1}\|_2$, is the condition-number of A and $\|v\|_A \triangleq \sqrt{v^T A v}$ for any vector v. (For a derivation of this bound, see, e.g., Golub and Van Loan [21].)

For the PCG method with the SVM preconditioner, the relevant condition number is $\kappa_2(P_{\mathcal{A}}^{-1} M)$. In this section, we show that the definition (13) of $P_{\mathcal{A}}$, together with the rule (15) for choosing indices to include in $\mathcal{A}$, implies that $\kappa_2(P_{\mathcal{A}}^{-1} M)$ converges to one as μ converges to zero. Thus, as the optimization progresses, $P_{\mathcal{A}}$ becomes an increasingly accurate preconditioner for M.

Note that both M and $P_{\mathcal{A}}$ have the form $I + G$, where G denotes a positive-semidefinite matrix. This motivates the following proposition.

Proposition 3 *If $A = I + G$, where G is positive-semidefinite, then A is invertible, and $\|A^{-1}\|_2 \le 1$.*

Proof By definition of A,

$$\|v\|_2 \|Av\|_2 = v^T A v = v^T v + v^T G v \ge \|v\|_2^2,$$

holds for any vector v. But by Cauchy-Schwartz inequality, $\|v\|_2\|Av\|_2 \ge v^T A v$, and so a factor of $\|v\|_2$ may be canceled from both sides of the preceding inequality to conclude that $\|Av\|_2 \ge \|v\|_2$. This inequality establishes the nonsingularity of A because $Av = 0$ implies that $v = 0$.

If u is any vector such that $\|u\|_2 = 1$, then

$$1 = \|AA^{-1}u\|_2 \ge \|A^{-1}u\|_2.$$

But $\|A^{-1}\|_2 = \max_{\|u\|_2=1} \|A^{-1}u\|_2$, and so $\|A^{-1}\|_2 \le 1$. □

The preceding proposition yields a bound on $\kappa_2(P_{\mathcal{A}}^{-1}M)$.

Proposition 4 *Let M and $P_{\mathcal{A}}$ be defined as in* (11) *and* (13), *respectively. Then*

$$\kappa_2(P_{\mathcal{A}}^{-1}M) \le (1 + \|M - P_{\mathcal{A}}\|_2)^2.$$

Proof Note that $P_{\mathcal{A}}^{-1}M = I + P_{\mathcal{A}}^{-1}(M - P_{\mathcal{A}})$. Therefore, from Proposition 3 it follows that

$$\|P_{\mathcal{A}}^{-1}M\|_2 \le 1 + \|P_{\mathcal{A}}^{-1}\|_2\|M - P_{\mathcal{A}}\|_2 \le 1 + \|M - P_{\mathcal{A}}\|_2.$$

It follows from a similar argument that

$$\|(P_{\mathcal{A}}^{-1}M)^{-1}\|_2 = \|M^{-1}P_{\mathcal{A}}\|_2 \le 1 + \|M - P_{\mathcal{A}}\|_2.$$

The result follows. □

The SVM preconditioner is specifically designed to bound $M - P_{\mathcal{A}}$.

Proposition 5 *Let $P_{\mathcal{A}}$ be defined by* (13), *and let $\mathcal{A}$ be chosen by the rule* (15) *for some $\gamma \le \hat{\gamma}$, where the upper bound $\hat{\gamma}$ is independent of the iteration number. Then there is a positive constant ξ such that, for any positive choice of the variables v, s, u, and z, it holds that $\|M - P_{\mathcal{A}}\|_2 \le \xi\mu^{1/2}$.*

Proof In the case in which all the observation vectors are zero, the proposition is obviously true because $M = P_{\mathcal{A}}$. We assume for the remainder of the proof that at least one of the observations is nonzero. With this assumption in mind, we define the two positive constants

$$c_1 = \max\{\|y_i\|_2 \mid i = 1, \ldots, n\},$$
$$c_2 = \min\{\|y_i\|_2 \mid y_i \neq 0,\ i = 1, \ldots, n\}$$

that we use below.

Consider the matrix difference

$$M - P_{\mathcal{A}} = \sum_{i \notin \mathcal{A}} \left(\frac{1}{\omega_i} y_i y_i^T - \operatorname{diag}\Big(\frac{1}{\omega_i} y_i y_i^T\Big) \right) + \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T - \frac{1}{\sigma} y_d y_d^T.$$

By rule (15), $y_i^T y_i / \omega_i < \hat{\gamma} \mu^{1/2}$ for $i \notin \mathcal{A}$. But then for $i \notin \mathcal{A}$,

$$\|y_i y_i^T\|_2 / \omega_i = y_i^T y_i / \omega_i < \hat{\gamma} \mu^{1/2}.$$

Because $y_i y_i^T$ is a rank-one matrix, its two norm is equal to its Frobenius norm. Subtracting the diagonal elements from a matrix can only decrease the Frobenius norm, and so

$$\sum_{i \notin \mathcal{A}} \left\| \frac{1}{\omega_i} y_i y_i^T - \operatorname{diag}\Big(\frac{1}{\omega_i} y_i y_i^T\Big) \right\|_2 \leq n \hat{\gamma} \mu^{1/2}. \tag{17}$$

We now seek to establish an upper bound on

$$\left\| \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T - \frac{1}{\sigma} y_d y_d^T \right\|_2 .$$

Consider first the case in which $\mathcal{A}$ is nonempty. If $\mathcal{A}$ is nonempty, we define the necessarily nonnegative value $\Theta = (\sigma - \tilde{\sigma})/\tilde{\sigma}$. We may expand the product

$$\frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T = \frac{1}{\sigma} (1 + \Theta) \big(y_d - (y_d - \tilde{y}_d)\big) \big(y_d - (y_d - \tilde{y}_d)\big)^T,$$

gather terms, and apply norm inequalities to determine that

$$\begin{aligned} \left\| \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T - \frac{1}{\sigma} y_d y_d^T \right\|_2 \leq \frac{\Theta}{\sigma} \|y_d\|_2^2 &+ 2\Big(\frac{1+\Theta}{\sigma}\Big) \|y_d - \tilde{y}_d\|_2 \|y_d\|_2 \\ &+ \Big(\frac{1+\Theta}{\sigma}\Big) \|y_d - \tilde{y}_d\|_2^2. \end{aligned} \tag{18}$$

Suppose $\mathcal{A}$ is nonempty, in other words that there is at least one i that satisfies rule (15). Then,

$$\tilde{\sigma} = \sum_{i \in \mathcal{A}} \omega_i^{-1} \geq \frac{\gamma}{c_1^2} \min\big(1, \mu^{1/2}\big).$$

Similarly,

$$\sigma - \tilde{\sigma} = \sum_{i \notin \mathcal{A}} \omega_i^{-1} \le \frac{n-1}{c_2^2} \gamma \min\left(1, \mu^{1/2}\right).$$

Thus, whenever $\mathcal{A}$ is nonempty, it follows that

$$\Theta = \frac{\sigma - \tilde{\sigma}}{\sigma} \left(1 + \frac{\sigma - \tilde{\sigma}}{\tilde{\sigma}}\right) \le \frac{\sigma - \tilde{\sigma}}{\sigma} \left(1 + (n-1)\left(\frac{c_1}{c_2}\right)^2\right). \tag{19}$$

We emphasize that this bound is independent of the choice of $\gamma > 0$ so long as $\mathcal{A}$ is nonempty for that value of γ.

By definition, $y_d = \sum_{i=1}^n (d_i/\omega_i) y_i$ and $y_d - \tilde{y}_d = \sum_{i \notin \mathcal{A}} (d_i/\omega_i) y_i$. Therefore,

$$\|y_d\|_2 \le \sum_{i=1}^n \frac{\|y_i\|_2}{\omega_i} \le c_1 \sum_{i=1}^n \frac{1}{\omega_i} = c_1 \sigma, \tag{20}$$

and for the same reason

$$\|y_d - \tilde{y}_d\|_2 \le c_1 \sigma. \tag{21}$$

Furthermore,

$$\|y_d - \tilde{y}_d\|_2 \le \frac{1}{c_2} \sum_{i \notin \mathcal{A}} \frac{c_2 \|y_i\|_2}{\omega_i} \le \frac{1}{c_2} \sum_{i \notin \mathcal{A}} \frac{y_i^T y_i}{\omega_i} \le \frac{n}{c_2} \hat{\gamma} \mu^{1/2}. \tag{22}$$

Here again we have made use of rule (15). If $\mathcal{A}$ is nonempty, we may combine inequalities (17)–(22) to conclude that there is an $\xi_1 > 0$ so that $\|M - P_{\mathcal{A}}\|_2 \le \xi_1 \mu^{1/2}$. If $\mathcal{A}$ is empty, then $\tilde{y}_d = 0$ and $\tilde{y}_d \tilde{y}_d^T / \tilde{\sigma} = 0$. Therefore, if $\mathcal{A}$ is empty, it follows from (20) and (22) that

$$\left\| \frac{1}{\tilde{\sigma}} \tilde{y}_d \tilde{y}_d^T - \frac{1}{\sigma} y_d y_d^T \right\|_2 = \left\| \frac{1}{\sigma} y_d y_d^T \right\|_2 \le \frac{c_1 n \hat{\gamma}}{c_2} \mu^{1/2}.$$

Thus, there is an ξ_2 for which $\|M - P_{\mathcal{A}}\|_2 \le \xi_2 \mu^{1/2}$ whenever $\mathcal{A}$ is empty. The result follows by setting $\xi = \max(\xi_1, \xi_2)$. □

5.3 Termination criteria for the PCG method

The PCG method does not solve linear systems exactly but only to some relative tolerance that is typically much greater than machine precision. We have found it advantageous to solve the early primal-dual systems to low accuracy and to increase the accuracy as iterations proceed.

Consider a single primal-dual iteration, and let $Mx = b_P$ denote the system (9a) with the right-hand side associated with the predictor step. Similarly, let $Mx = b_C$ denote the system associated with the corrector step. Let

$$\texttt{rtol} = \min(10^{-1}, 10^{-1}\mu).$$

The PCG algorithm is terminated when the predictor step, x_P, satisfies the condition

$$\|b_P - Mx_P\|_2 \le \max\bigl(\texttt{rtol} \times \|b_P\|_2, 10^{-12}\bigr).$$

For the corrector step, x_C, we tighten the tolerance and impose the condition

$$\|b_C - Mx_C\|_2 \le \max\left(10^{-2} \times \texttt{rtol} \times \|b_C\|_2, 10^{-12}\right).$$

We maintain a count of the cumulative number of PCG iterations used in solving both the predictor and corrector equations. While solving either system, if the cumulative number of iterations exceeds

$$i_{\max} = \max\left(m/8,\ 20\right),$$

then the value of γ is decreased, thereby increasing the size of the active-set. The preconditioner is then updated before proceeding to the next PCG iteration.

The following rule is used to adjust the value of γ. For $j \ge 1$, let

$$k_j = \min\left(|\mathcal{A}| + j\lfloor m/2\rfloor,\ n\right).$$

Then, if d_j is the k_j^{th} largest value of $y_i^T y_i/\omega_i$ for $i = 1, \ldots, n$, define

$$\gamma_j = (1 - 10^{-8})\frac{d_j}{\min\left(1, \mu^{1/2}\right)},$$

and let $\mathcal{A}_j$ be the set of indices chosen if $\gamma = \gamma_j$ in the rule (15). The intent of these definitions is to ensure that the size of $\mathcal{A}_j$ is at least k_j. For each $j \ge 1$ in turn, we use the indices in $\mathcal{A}_j$ to form the preconditioner, reset the cumulative iteration count to zero, and continue the PCG iteration for at most $i_{\max}$ additional iterations. Note that $\mathcal{A} \subset \mathcal{A}_1$ and that $\mathcal{A}_j \subset \mathcal{A}_{j+1}$ for $j \ge 1$. Therefore for each j, the preconditioner may be updated rather than being recalculated. Because the preconditioner is exact for sufficiently large j, there must be some γ_j for which both predictor and corrector systems converge. We choose this final γ_j to be γ for subsequent iterations of the optimization.

6 Numerical results

We present two sets of numerical tests of our new method, which in this paper we call the SVM-PCG method. First we test the performance of the SVM-PCG on several publicly available test sets. We compare its performance on these sets to a related method that uses a direct linear solver, SVM-Direct, and to the active-set package SVM$^{\text{light}}$. Second, we compare the new method against itself on artificially generated datasets of varying size to test the method's sensitivity to the number of features and to the number of observations.

6.1 Implementation details

SVM-PCG solver was implemented using the object-oriented quadratic programming code OOQP. The code implements Mehrotra's Predictor-Corrector algorithm (MPC) entirely in terms of an abstract representation of a convex quadratic program. We implemented the SVM solver by providing concrete implementations of the necessary abstract routines and data structures, tailoring these implementations to handle the SVM-PCG subproblem efficiently. We use an algorithm, supplied by OOQP, that follows Mehrotra's algorithm with minimal modifications necessary to handle quadratic, rather than linear, programs. This algorithm is described in Gertz and Wright [18].

The implementation described in all tests below uses a sparse representation for the matrix Y. To keep the results simple, we use the same implementation for all tests, even those with mostly dense data. The observation data is stored internally as 4-byte single precision floating point numbers, but these data are promoted to double precision as they are used. Linear algebra operations are usually performed using BLAS [4, 9] and LAPACK [2] routines, but operations not directly available in BLAS were coded in C++. We implemented the linear conjugate gradient solver in C++ using OOQP's internal data structures.

Some of our tests compare SVM-PCG to SVM-Direct, an algorithm that uses a direct linear solver to solve (9a). SVM-PCG and SVM-Direct are implemented using the same code. To cause the implementation to solve (9a) directly, we set $\gamma = 0$ at each iteration. With this setting, $M = P_{\mathcal{A}}$. Thus, M is factored at each step, and the linear conjugate gradient method converges in one iteration. When compared to the cost of factoring M, the cost of performing a single iteration of the conjugate gradient method is negligible.

For all problems described below, we use OOQP's default starting point strategy, described in [18]. The QP solvers in OOQP are said to converge when

$$\mu = (v^T s + u^T z)/(2n) < 10^{-6} \quad \text{and} \quad \|r\|_\infty < 10^{-6} \times \max_{ij} |x_{ij}|,$$

where

$$r = \left(r_w, \rho_\beta, r_z, r_s\right)^T,$$

with r_w, ρ_β, r_z, and r_s defined in (4).

OOQP was compiled using GCC version 3.4.6 with the optimization option-O3. All tests were performed on one processor of a quad-processor Intel Xeon 1.8 GHz Intel processor CPU with 8 GB of RAM.

6.2 SVM$^{\text{light}}$

We compare SVM-PCG with the active-set package SVM$^{\text{light}}$. The test sets we are using should be of reasonable size for this package. In our tests, we use version 6.01, compiled using GCC version 3.4.6 with the optimization option-O3. We run SVM$^{\text{light}}$'s solver, svm_learn, with its default settings, except that we provide the penalty parameter τ as described below. The default error tolerance of SVM$^{\text{light}}$ is 10^{-3}.

Table 1 Comparison of SVM-PCG, SVM-Direct, and SVM$^{\text{light}}$ for five data sets

Name	Obs.	Feat.	Avg. Features	SVM-PCG seconds	SVM-Direct seconds	SVM$^{\text{light}}$ seconds
Adult	32561	123	13.9	6.84	5.21	27.4
Website	49749	300	11.7	30.9	28.5	34.1
Faces	6979	361	360	18.4	113	93
Isolet	7779	617	615	36.4	330	5.11
MNIST	60000	780	150	217	888	52 min

6.3 Comparison of PCG on standard problems

We use the following six problem sets to test the new solver:

Adult The documentation accompanying this data set states, "The benchmark task is to predict whether a household has >\$50K income based on 14 other fields in a census form. Eight of those fields are categorical, while six are continuous. The six fields are quantized into quintiles, which yields a total of 123 sparse binary features." We obtained the test set directly from the SMO web site [30], but they in turn obtained it from the repository of machine learning composed by Hettich, Blake, and Merz [22] at the University of California, Irvine.

Website According to the accompanying documentation, "The benchmark task is to predict whether web page belongs to a category based on the presence of 300 selected keywords on the page." This benchmark is described in [32]. We obtained this test set, specifically the set web-a.dist, from the SMO web site [30].

Face Recognition The faces dataset is a collection of 19×19 images, some of which represent faces and some of which do not. There are two sets: a training set of 6,977 images and a testing set of 24,045 images. Each observation is "a single image, histogram equalized and normalized so that all pixel values are between 0 and 1." We obtained from data from the MIT CBCL web site [7]. It is the exact data used in Alvira and Rifkin [1].

Isolet Spoken Letter Recognition Database This database was created by Ron Cole and Mark Fanty [22]. Data was generated from 150 subjects who spoke each letter of the alphabet twice. Two data sets were provided, which we concatenated to create a single data set with 7,779 data points and 617 features.

MNIST Handwriting Recognition The MNIST database, consisting of 60,000 handwritten digits, originates from AT&T Research Labs [26]. We obtained this data from the LASVM website [5], where it is available in the correct input format for OOQP or SVM$^{\text{light}}$(or LASVM). We used training set 8, which has 60,000 observations with 780 sparse, continuous features.

The Adult problem was solved with $\tau = .05$ as was done in [32]. The Faces problem was solved using $\tau = 10$, as was done in [38]. All others were solved with $\tau = 1$.

Table 1 compares the run times for SVM-PCG, SVM-Direct, and SVM$^{\text{light}}$. Times shown for the SVM-PCG and SVM-Direct methods only include the time spent in the QP solver, not the time taken to read the data and write the results. The times

Table 2 Times and iteration counts for SVM-PCG and SVM-Direct on five data sets

Name	SVM-Direct iterations	SVM-Direct seconds	SVM-PCG iterations	SVM-PCG seconds	Avg. PCG iterations
Adult	35	5.21	36	6.84	12.3
Website	93	28.5	93	30.9	12.9
Faces	26	113	26	18.4	16.5
Isolet	24	330	25	36.4	18.8
MNIST	48	888	47	217	35.6

shown for SVM$^{\text{light}}$ were reported by the svm_learn program and also appear to only include time spent in the solver. The one exception to this is the time reported for SVM$^{\text{light}}$ on MNIST. On this problem SVM$^{\text{light}}$ reports a negative time, presumably because the numerical type that it uses to measure time overflows. In this case, we used the "time" built-in command of the BASH shell to obtain an estimate of elapsed time.

Table 1 shows the number of observations for each data set, the number of features, and the average number of nonzero features in each observation. The problem sets are sorted by the number of nonzero data points in each set. The Adult and Website data sets, in particular, are quite sparse, and have too few nonzero features for the PCG method to show an advantage. Table 1 does demonstrate, however, that the sparse implementation is efficient, and the extra overhead of applying the PCG method is small. For the datasets with a moderately large number of nonzero features, the PCG method shows a clear improvement.

Table 1 shows that on these datasets, the PCG method is competitive with SVM$^{\text{light}}$. The time required by SVM$^{\text{light}}$ is not entirely predictable, though it generally needs more time on sets with larger numbers of observations. For the Isolet problem, SVM$^{\text{light}}$ is much faster.

Table 2 shows a more detailed comparison of the SVM-PCG and SVM-Direct methods. Columns two and three compare the number of MPC iterations required by each method to solve each the test problems. The iteration counts are nearly identical. The SVM-PCG method does not require more MPC iterations, even though it solves the linear systems used in each MPC iteration to a more relaxed tolerance.

The final column of Table 2 shows the average number of SVM iterations per MPC iteration. The average number of PCG iterations is modest for each of these problems. Note that we report the average per MPC iteration, not per linear solve. The average per MPC iteration is the more relevant number because SVM-PCG uses the solution from the predictor step as a starting point for the corrector step. OOQP's default start strategy [18] also performs a linear solve. To be conservative, we include the iterations from the start strategy in the total number of PCG iterations when computing the average.

6.4 Tests on sampled data

We sample data from the Faces database to test the effectiveness of the PCG method for different values of m and n, where m is the number of features and n the number

Table 3 Times and iterations counts for data sets formed by sampling observations from Faces

Observations	MPC iterations	SVM-Direct seconds	SVM-PCG iterations	SVM-PCG seconds	Avg. PCG iterations
10000	32	197	32	33.9	20.8
15000	39	363	39	70.2	25.8
20000	43	514	44	112	27.9
25000	47	696	47	148	29.5
30000	53	981	54	209	30.6

Table 4 Times and iterations counts for data sets formed by sampling features from Faces

Features	MPC iterations	SVM-Direct seconds	SVM-PCG iterations	SVM-PCG seconds	Avg. PCG iterations
50	54	22.1	55	17.7	11.8
150	27	57.3	29	28.3	15.3
250	60	354	62	128	23.1
350	34	591	33	180	17.5

of observations. To test different values of m we must sample features, and therefore create artificial problems. We sample features, rather than resizing images, to avoid overly smoothing the data.

We prefer sampling to using purely randomly generated data. For iterative linear solvers, uniformly distributed random data tend to create simple problems. In general, a randomly generated square matrix tends to be well-conditioned. PCG linear solvers perform well on such matrices. In the SVM-PCG algorithm, there is a complex interaction between the MPC and PCG methods that makes it difficult to analyze exactly why the PCG does well on random data, but we have found it to generally be the case.

We sample without replacement from the data set formed by concatenating both the testing and training sets of Faces. The combined data sets has 361 features and 31022 observations. Table 3 shows the results of running both SVM-PCG and SVM-Direct on five data sets that each have 361 features but that have either 10000, 15000, 20000, 25000, or 30000 observations. Similarly, Table 4 shows the results of running the same algorithms on four data sets with 31022 observations, but having either 50, 150, 250, or 350 features. Tables 3 and 4 have the same format at Table 2.

Figure 1 plots CPU time against number of observations for both the SVM-Direct and SVM-PCG methods. Figure 2 plots CPU time against number of features for the same methods. These figures are another view of the timing data in Tables 3 and 4 respectively.

For each of the sampled data sets, the SVM-PCG method and the SVM-Direct method required a similar number of MPC iterations. Furthermore, the average number of PCG iterations per MPC iteration remains modest; for all these sets it was less than 33. For the data shown in Table 3 and Fig. 1, growth in CPU time is faster than

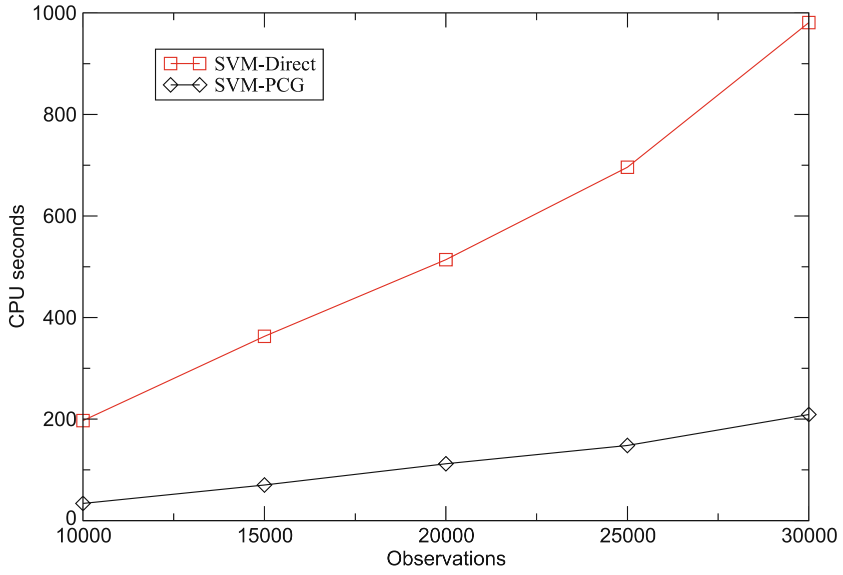

Fig. 1 The CPU time needed by SVM-PCG to generate an SVM for subsets of Faces created by sampling observations

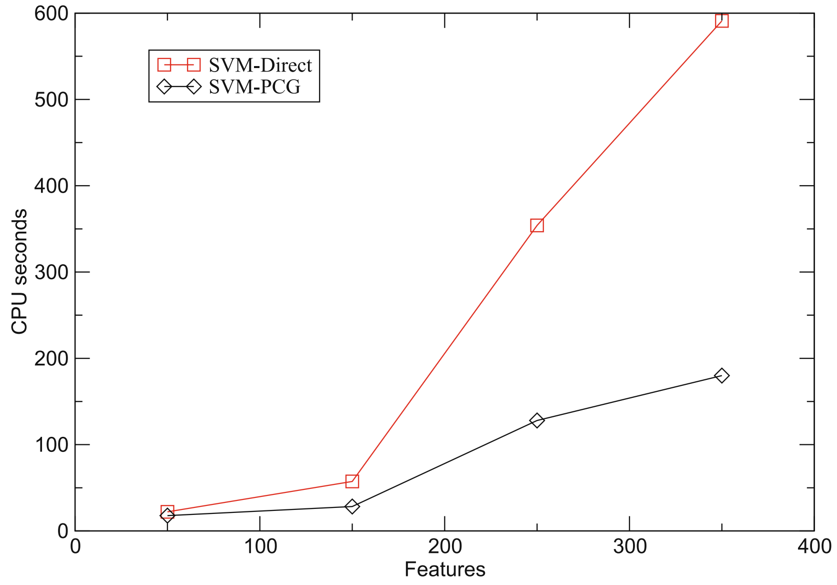

Fig. 2 The CPU time needed by SVM-PCG to generate an SVM for subsets of Faces created by sampling features

linear in the number of observations, which is expected. The number of multiplications needed to perform the forward multiplication in a PCG iteration is linear in the number of observations, and the number of MPC iterations generally grows weakly with size. Table 4 and Fig. 2 show a complex relationship between the number of features and the CPU time. This reflects the fact that the iteration counts of the MPC and PCG iterations are strongly influenced by the data itself, rather than just by the size of the data. While it is not possible to predict CPU time based on data size, Tables 3 and 4 show acceptable performance and do not show excessive growth as the problem size increases.

7 Discussion

We have described a new technique for reducing the computational cost of generating a SVM using an interior-point method based on the SMW formulation. The most time-consuming step in using the SMW formulation is obtaining a solution to (9a). We have implemented a solver for (9a) based on the preconditioned conjugate gradient method. This method is applicable when the number of features in the problem is moderately large. Thus, we have extended the number of problems to which the interior-point method may practically be applied. Because the cost-per-iteration of the interior-point method is linear in the number of observations, and because the number of iterations of the interior-point method is only weakly dependent on size [12, 36], the new method is promising for generating SVMs from a relatively large number of observations.

We have adapted the object-oriented interior-point package OOQP to solve the SVM subproblem using the PCG method with a new preconditioner that is based on the structure of the SVM subproblem itself. We denote this implementation SVM-PCG. We have presented numerical experiments that compare SVM-PCG to a related algorithm, SVM-Direct, that solves the linear system (9a) using a direct linear solver.

In our tests, when the average number of nonzero features per observation was moderately large, the SVM-PCG method was superior to the SVM-Direct methods. As our tests show, the time taken by the SVM-PCG method is not entirely predictable because it depends not only on the data set size, but also on the content of the data itself. Thus it is not possible to give a precise lower bound on the number of features for which the SVM-PCG method is superior to SVM-Direct. However, the SVM-PCG method is efficient even for those data sets in our tests, Adult and Web, that are too sparse to benefit from the use of the PCG linear solver. This suggests that the cost of incorrectly choosing to use the SVM-PCG, rather than the SVM-Direct, is not large.

We have also compared SVM-PCG with the active-set method SVM$^{\text{light}}$ and have shown that on our problem set, the new method is competitive. However, one must be careful when interpreting these data. SVM$^{\text{light}}$, and active-set methods in general, can solve the SVM subproblem quickly. Moreover, SVM$^{\text{light}}$ has capabilities that our implementation does not, including support for nonlinear kernels. Active-set methods, however, are inherently exponential algorithms and can require a large number of iterations.

Active-set methods have an important advantage over interior-point methods in that they can use nonlinear kernels directly. The radial basis function (RBF) kernel is a popular nonlinear kernel, and problems that use a RBF kernel cannot be mapped to a finite dimensional linear SVM. Ultimately, the value of the method presented here when used with a nonlinear kernel will depend on the effectiveness of these low-rank approximations. The efficiency of producing such approximations is also relevant, though the approximation need only be computed once. For instance, the cost of producing a low-rank approximation using a QR decomposition is proportional to the cost of an interior-point iteration of the resulting low-rank SVM using a dense factorization. The use of low-rank approximations with this method is a topic for further research. The PCG method, however, may allow for the use of higher-rank approximations than were previously possible.

Some SVM problems use a very large number of sparse features. Clearly there is a limit to the number of features beyond which the PCG method will have difficulty. At the very least, it would not make sense to do a dense factorization of the preconditioner if the number of features exceeds the number of observations. One must currently use active-set methods to solve such problems. Further research would be needed to determine how an interior point method can be applied in this case.

A natural extension of this PCG method would be to implement a version that may run in parallel on multiple processors, as the linear algebra operations of the MPC method may be performed using data-parallel algorithms. The primal-dual interior-point algorithm must solve several perturbed Newton-systems. Because the number of Newton-like systems that must be solved is typically modest, the cost of forming and solving each system tends to dominates the computational cost of the algorithm. Thus a strategy that reduces the solve time for (9a), can reduce the time required to compute a classifier.

In this paper we applied serial preconditioned iterative methods to solve (9a). A similarly motivated parallel direct-factorization approach, resulting in substantial time reductions, was also explored in [16]. In the parallel approach, $Y^T\Omega^{-1}Y$ from (9a) was formed using multiple processors and the equation was solved using a direct-factorization of M (defined in (11)).

For a PCG based method, one would form $P_{\mathcal{A}}$, which has the same structure as M, in parallel. The matrix-vector operations of the conjugate-gradient method itself could be performed in parallel on distributed data. However, in order to maximize the effectiveness of a parallel iterative method, necessary precautions would need be taken to ensure a proper distribution of data. The worst case scenario when forming $P_{\mathcal{A}}$ in parallel would occur if all vectors corresponding to $\mathcal{A}$ were to lie on a single processor. In this cases, little gain would be seen over a serial approach. Robust and effective methods of balancing the cost of forming $P_{\mathcal{A}}$ across available processors are the subject of further research.

Acknowledgements This work was supported by National Science Foundation grants ACI-0082100 and the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38 and the Mathematical, Information, and Computational Sciences Program of the U.S. Department of Energy, under contract DE-AC04-94AL85000 with Sandia Corporation. The second author was supported in part by the National Science Foundation grants DMS-0208449 and DMS-0511766 while a graduate student at the University of San Diego California.

We thank Todd Munson for informative discussions about support vector machine solvers. We thank Philip Gill, Steven J. Wright, and the anonymous referees for their careful reading of the paper and helpful suggestions.

References

1. Alvira, M., Rifkin, R.: An empirical comparison of snow and svms for face detection. A.I. memo 2001-004, Center for Biological and Computational Learning, MIT, Cambridge, MA (2001)
2. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK User's Guide. SIAM, Philadelphia (1992)
3. Bach, F.R., Jordan, M.I.: Predictive low-rank decomposition for kernel methods. In: ICML '05: Proceedings of the 22nd International Conference on Machine Learning, pp. 33–40. ACM Press, New York (2005)
4. Blackford, L.S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., Whaley, R.C.: An updated set of basic linear algebra subprograms (BLAS). ACM Trans. Math. Soft. **28**, 135–151 (2002)
5. Bottou, L.: LaSVM. http://leon.bottou.org/projects/lasvm/
6. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Min. Knowl. Discov. **2**, 121–167 (1998)
7. CBCL center for biological & computational learning. http://cbcl.mit.edu/projects/cbcl/
8. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press, Cambridge (2000)
9. Dongarra, J.: Basic linear algebra subprograms technical forum standard. Int. J. High Perform. Appl. Supercomput. **16**, 1–111 (2002), 115–199
10. Drineas, P., Mahoney, M.W.: On the Nyström method for approximating a gram matrix for improved kernel-based learning. J. Mach. Learn. Res. **6**, 2153–2175 (2005)
11. Fan, R.-E., Chen, P.-H., Lin, C.-J.: Working set selection using second order information for training support vector machines. J. Mach. Learn. Res. **6**, 1889–1918 (2005)
12. Ferris, M.C., Munson, T.S.: Interior point methods for massive support vector machines. SIAM J. Optim. **13**, 783–804 (2003)
13. Ferris, M.C., Munson, T.S.: Semismooth support vector machines. Math. Program. **101**, 185–204 (2004)
14. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. J. Mach. Learn. Res. **2**, 243–264 (2001)
15. Fletcher, R.: Practical Methods of Optimization. Constrained Optimization, vol. 2. Wiley, New York (1981)
16. Gertz, E.M., Griffin, J.D.: Support vector machine classifiers for large data sets, Technical memo ANL/MCS-TM-289, Argonne National Lab, October 2005
17. Gertz, E.M., Wright, S.J.: OOQP user guide. Technical Memorandum ANL/MCS-TM-252, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL (2001)
18. Gertz, E.M., Wright, S.J.: Object oriented software for quadratic programming. ACM Trans. Math. Softw. (TOMS) **29**, 49–94 (2003)
19. Gill, P.E., Murray, W., Wright, M.H.: Practical Optimization. Academic, London (1981)
20. Glasmachers, T., Igel, C.: Maximum-gain working set selection for SVMs. J. Mach. Learn. Res. **7**, 1437–1466 (2006)
21. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore (1996)
22. Hettich, C.B.S., Merz, C.: UCI repository of machine learning databases (1998). http://www.ics.uci.edu/~mlearn/MLRepository.html
23. In Hyuk Jung, A.L.T., O'Leary, D.P.: A constraint reduced IPM for convex quadratic programming with application to SVM training. In: INFORMS Annual Meeting (2006)
24. Joachims, T.: Making large-scale support vector machine learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods—Support Vector Learning, pp. 169–184. MIT Press, Cambridge (1998)
25. Keerthi, S., Shevade, S., Bhattacharyya, C., Murthy, K.: Improvements to Platt's SMO algorithm for SVM classifier design. Neural Comput. **13**, 637–649 (2001)

26. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
27. Louradour, J., Daoudi, K., Bach, F.: SVM speaker verification using an incomplete Cholesky decomposition sequence kernel. In: IEEE Odyssey 2006: The Speaker and Language Recognition Workshop, IEEE, June 2006
28. Mangasarian, O.L., Musicant, D.R.: Lagrangian support vector machines. J. Mach. Learn. Res. **1**, 161–177 (2001)
29. Osuna, E., Freund, R., Girosi, F.: Improved training algorithm for support vector machines. In: Proceedings of the IEEE Workshop on Neural Networks for Signal Processing, pp. 276–285. (1997)
30. Platt, J.: Sequential minimal optimization. http://research.microsoft.com/en-us/projects/svm/default.aspx
31. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods—Support Vector Learning, pp. 41–65. MIT Press, Cambridge (1998)
32. Platt, J.: Sequential minimal optimization: A fast algorithm for training support vector machine. Technical Report TR-98-14, Microsoft Research, (1998)
33. Scheinberg, K.: An efficient implementation of an active set method for SVMs. J. Mach. Learn. Res. **7**, 2237–2257 (2006)
34. Smola, A.J., Schölkopf, B.: Sparse greedy matrix approximation in machine learning. In: Proceedings of the 17th International Conference on Machine Learning, Stanford University, CA, pp. 911–918. (2000)
35. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, Heidelberg (1995)
36. Wright, S.J.: Primal–Dual Interior–Point Methods. SIAM Publications. SIAM, Philadelphia (1996)
37. Xiong Dong, J., Krzyzak, A., Suen, C.Y.: Fast SVM training algorithm with decomposition on very large data sets. IEEE Trans. Pattern Anal. Mach. Intell. **27**, 603–618 (2005)
38. Yang, M.-H.: Resources for face detection. http://vision.ai.uiuc.edu/mhyang/face-detection-survey.html