Noname manuscript No.
(will be inserted by the editor)

Exploiting separability in large-scale linear Support Vector Machine training

Kristian Woodsend · Jacek Gondzio

April 9, 2009

Abstract Linear support vector machine training can be represented as a large quadratic program. We present an efficient and numerically stable algorithm for this problem using interior point methods, which requires only $\mathcal{O}(n)$ operations per iteration. Through exploiting the separability of the Hessian, we provide a unified approach, from an optimization perspective, to 1-norm classification, 2-norm classification, universum classification, ordinal regression and ϵ-insensitive regression. Our approach has the added advantage of obtaining the hyperplane weights and bias directly from the solver. Numerical experiments indicate that, in contrast to existing methods, the algorithm is largely unaffected by noisy data, and they show training times for our implementation are consistent and highly competitive. We discuss the effect of using multiple correctors, and monitoring the angle of the normal to the hyperplane to determine termination.

Keywords Support Vector Machines · Interior Point Method · Separable Quadratic Program · Large Scale

1 Introduction

With the exponential increases in storage capacity and computing power, and fields such as text categorisation, image recognition, and bioinformatics generating huge real-world data sets, scalability and efficiency become important issues for machine learning approaches. Support Vector Machines (SVMs) are

Kristian Woodsend · Jacek Gondzio
School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, The King's Buildings, Edinburgh, EH9 3JZ, UK

Kristian Woodsend
E-mail: k.woodsend@ed.ac.uk

Jacek Gondzio
E-mail: j.gondzio@ed.ac.uk

a powerful machine learning technique, and they offer state-of-the-art performance, but the training of an SVM is computationally expensive and relies on optimization. The core of the approach is a convex quadratic optimization problem (QP) which scales with the number of data points rather than the feature space dimension. This complexity result makes applying SVMs to large scale data sets challenging, and in practise the optimization problem is intractable by general purpose optimization solvers as each iteration scales cubically with the size of the training set.

The standard approach is to build a solution by solving a sequence of small scale problems, e.g. Decomposition [26; 23] or Sequential Minimal Optimization [27]. State-of-the-art software such as SVM$^{\text{light}}$ [18] and SVMTorch [4] use these techniques. These are basically active-set techniques, which work well when the separation into active and non-active variables is clear, in other words when the data is separable by a hyperplane. With noisy data, the set of support vectors is not so clear, and the performance of these algorithms deteriorates. The Incremental Active Set (INCAS) method [9] is an approach where variables change set one at a time, and is better able to handle noisy data.

Other optimization techniques have also been tried. In SVM$^{\text{perf}}$ [19], an equivalent reformulation was developed with fewer variables and suited to cutting-plane algorithms, where time to converge is linear in the size of the training set. Coordinate descent methods update one variable at a time by minimizing a single-variable sub-problem, and this approach is implemented in the LibLinear software package [17].

Approaches have been proposed to adapt the SVM training problem to one that can be handled more efficiently. Mangasarian and co-workers have proposed several, and the Reduced SVM can be seen as representative [22]. A random subset of the data is chosen to characterize the space, and this is used to select a larger subset of the data for training. The approach of [20] is similar in that a set of feature basis vectors are defined by a relatively small number of points. To provide a continuously differentiable function suitable for unconstrained optimization techniques, both approaches rewrite the SVM training problem to use 2-norms for misclassification errors, and additionally the Reduced SVM maximises the margin with respect to both orientation and location relative to the origin. Both modifications were originally described in [24], and both are used in the LibLinear software package [17]. We make some comparisons with this software later, but in general we are interested in developing an efficient method without needing to resort to such modifications.

Another family of approaches are based on Interior Point Method (IPM) technology (Section 2), which works by delaying the split between active and inactive variables for as long as possible. IPMs generally work well on large-scale problems, as the number of outer iterations required grows very slowly with problem size [see 34]. A straight-forward implementation of the standard SVM dual formulation using IPM would have complexity $\mathcal{O}(n^3)$, where n is the number of data points, and be unusable for anything but the smallest problems. Within this family, several approaches based on different formula-

tions have been researched [7; 10; 13]. They are discussed further in Section 2.2. They have in common an aim to exploit the low-rank structure of the kernel matrix and reduce the problem to one where the algorithm works on a small dense matrix of size m, the number of features, giving a per-iteration computational complexity of $\mathcal{O}(nm^2)$. These approaches, however, inherently suffer from either numerical instability or memory caching inefficiencies.

In this paper, we present a set of efficient and numerically stable IPM-based formulations (Section 3), which unify, from an optimization perspective, linear 1-norm classification, 2-norm classification, universum classification, ordinal regression and ϵ-insensitive regression. We show that all these problems can be equivalently reformulated as very large, yet structured, separable QPs. Exploiting separability has been investigated for general sparse convex QPs [28; 25], but not for the SVM problem. Further, we show how IPM can be specialized to exploit separability in all these problems efficiently. We investigated performance for 1-norm classification. Our implementation matches the other IPM-based techniques mentioned above in terms of computational complexity. In practice it gives consistent and highly competitive training times, and for some problems it outperforms other implementations of the 1-norm classification problem (including the active-set and cutting plane algorithms mentioned above) by a large margin. Performance is confirmed through extensive numerical experiments (Section 4).

We now briefly describe the notation used in this paper. x_i is the attribute vector for the i^{th} data point, and are the observation values directly. There are n observations in the training set, and m attributes in each vector x_i. X is the $m \times n$ matrix whose columns are the attribute vectors x_i associated with each point. The classification label for each data point is denoted by $y_i \in \{-1, 1\}$. The variables $w \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ are used for the primal variables ("weights") and dual variables (α in SVM literature) respectively, and $w_0 \in \mathbb{R}$ for the bias of the hyperplane. Scalars and column vectors are denoted using lower case letters, while upper case letters denote matrices. D, S, U, V, Y and Z are the diagonal matrices of the corresponding lower case vectors.

2 Interior Point Methods

Interior point methods represent state-of-the-art techniques for solving linear, quadratic and non-linear optimization programmes. In this section the key issues of implementation for QPs are discussed very briefly to highlight areas of computational cost [for more details, see 34].

2.1 Outline of IPM

We are interested in solving the general convex quadratic problem

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Q z + c^T z \\ \text{s.t.} \quad & Az = b \\ & 0 \leq z \leq u, \end{aligned} \tag{1}$$

where u is a vector of upper bounds, and the constraint matrix A is assumed to have full rank. Dual feasibility requires that $A^T\lambda + s - v - Qz = c$, where λ is the dual variable associated with the constraints and $s, v > 0$ are the dual variables associated with the lower and upper bounds of z respectively. An interior point method (outlined in Algorithm 1) moves towards satisfying the KKT conditions over a series of iterations, by monitoring primal and dual feasibility and controlling the complementarity products

$$\begin{aligned} ZSe &= \mu e \\ (U - Z)Ve &= \mu e, \end{aligned}$$

where μ is a strictly positive parameter. At each iteration (steps 2–7), the method makes a damped Newton step towards satisfying the primal feasibility, dual feasibility and complementarity product conditions for a given μ. Then the algorithm decreases μ before making another iteration. The algorithm continues until both infeasibilities and the duality gap (which is proportional to μ) fall below required tolerances. An alternative termination criterion is discussed in Section 4.3.

Algorithm 1 Outline of interior point method

Require: Initial point $(z^0, s^0, v^0, \lambda^0)$
1: $(z, s, v, \lambda) := (z^0, s^0, v^0, \lambda^0)$
2: **while** stopping criteria is not fulfilled **do**
3: Calculate matrix M
4: Factorize matrix M
5: Calculate search direction $(\Delta z, \Delta s, \Delta v, \Delta\lambda)$ by solving $M\Delta\lambda = -\hat{r}_b$ and backsolving for other variables
6: Determine step size and calculate new iterates (z, s, v, λ)
7: Correct (z, s, v, λ) to obtain a more central point
8: **end while**
9: **return** (z, s, v, λ)

The Newton system to be solved at each iteration (steps 3–5) can be transformed into the *augmented system equations*:

$$\begin{bmatrix} -(Q + \Theta^{-1}) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_c \\ r_b \end{bmatrix}, \tag{2}$$

where $\Delta z, \Delta\lambda$ are components of the Newton direction in the primal and dual spaces respectively, $\Theta^{-1} \equiv Z^{-1}S + (U-Z)^{-1}V$, and r_c and r_b are appropriately

defined residuals. Furthermore, the *normal equations* are a set of equations found by eliminating Δz from the augmented system. Solving them requires calculating $M \equiv A\left(Q+\Theta^{-1}\right)^{-1} A^T$ (step 3), factorizing M (step 4), and then solving $M\Delta\lambda = -\hat{r}_b$ for $\Delta\lambda$ (step 5). Using multiple correctors to gain a more central iterate (step 7) is discussed in Section 4.4.

Calculating (step 3) and factorizing M (step 4) are the most computationally expensive operations of the algorithm. Interior point methods are efficient for solving quadratic programmes when the matrix Q is easily invertible; however, if the matrix is dense the time taken to invert M can become prohibitive. In such a case, it is advantageous to solve system (2).

2.2 Previous IPM approaches to SVM training

For large-scale data sets, we assume that the number of data points n greatly exceeds the number of features m, and we assume that the kernel matrix $Q \equiv YX^TXY$ is dense (see Section 3.1). Inverting this dense $n \times n$ matrix directly requires $\mathcal{O}(n^3)$ operations, and for large data-sets this approach is impractical.

By exploiting the low-rank representation of the linear kernel, it is possible to design IPM algorithms where the only matrix to be inverted has dimension $m \times m$, and the overall effort associated with computing its implicit inverse representation scales linearly with n and quadratically with m. This gives a significant improvement if $n \gg m$. A common approach is to use low-rank corrections in the representation of the Newton system, and exploit it through implicit inverse representation by applying the Sherman-Morrison-Woodbury (SMW) formula. An algorithm based on the dual formulation (4) and SMW formula has a computational complexity of $\mathcal{O}(nm^2)$ for the multiplication and $\mathcal{O}(m^3)$ for the inversion at each IPM iteration [7]; a similar approach working in primal space has the same complexity [10].

The SMW formula has been widely used in interior point methods, where it often runs into numerical difficulties. There are two main causes of the difficulties: if the matrix Θ^{-1} that is inverted is ill-conditioned; and if there is near-degeneracy in the data matrix (XY). Ill-conditioning of the scaling matrix Θ^{-1} is a feature of IPMs, especially in the later iterations. Near-degeneracy in (XY) will occur if there are multiple data points which lie along or close to the separating hyperplanes, and this is accentuated if the data is not well scaled. Neither of these problems can really be avoided by a SMW-based algorithm. In [13], data sets of this type were constructed where an SMW-based algorithm required many more iterations to terminate, and in some cases stalled before achieving an accurate solution. The authors also showed that this situation arises in real-world data sets.

Goldfarb and Scheinberg [13] proposed an alternative technique based on Product Form Cholesky Factorization. In this technique, a Cholesky factorization is computed for a very sparse matrix and then updated to take into account each of the $m+1$ dense columns of A. The approach has the same

complexity $\mathcal{O}(nm^2)$ as the previous approaches (although a small multiple of flops are required), but better numerical properties: LDL^T Cholesky factorization of the IPM normal equation matrix with variables following the central path has the property that L remains numerically stable despite D becoming increasingly ill-conditioned, as happens in the later iterations of IPM algorithms [12]. Although Goldfarb and Scheinberg exploit symmetries in their technique to reduce the computations required, their approach suffers from some memory caching inefficiencies because each feature is handled separately (this is investigated in Section 4.6). It is also intrinsically sequential, and so does not facilitate a parallel computing implementation.

3 Support Vector Machines

In this section we briefly outline the formulations for Support Vector Machines used for linear classification and regression problems, and by showing how optimality conditions between the primal weight variables $w \in \mathbb{R}^m$ and the dual variables $z \in \mathbb{R}^n$ can be used to derive equivalent formulations, we present a new, unified approach for SVM training that combines the separability of the primal formulation with the small number of constraints of the dual formulation. As will be seen, our separable formulations introduce m additional variables and constraints to the standard dual problems, but such an approach enables an IPM algorithm with a complexity that is linear in the dataset size. Although the decision variables w and $\bar{z}$ in the following discussion are free, we chose to give them bounds which brings all variables in line with (1). Efficient setting of these bounds is described in Section 4.2.

3.1 Classification

A Support Vector Machine (SVM) is a classification learning machine that learns a mapping between the features and the target label of a set of data points known as the *training set*, and then uses a hyperplane $w^T x + w_0 = 0$ to separate the data set and predict the class of further data points. The labels are the binary values "yes" or "no", which we represent using the values $+1$ and -1. The objective is based on the Structural Risk Minimization (SRM) principle, which aims to minimize the risk functional with respect to both the empirical risk (the quality of the approximation to the given data, by minimising the misclassification error) and maximize the confidence interval (the complexity of the approximating function, by maximising the separation margin) [29; 30]. A fuller description is also given in [5].

For a *linear kernel*, the attributes in the vector x_i for the i^{th} data point are the observation values directly, while for a *non-linear kernel* the observation values are transformed by means of a (possibly infinite dimensional) non-linear mapping Φ.

Training an SVM has at its core a convex quadratic optimization problem. For a linear SVM classifier using a 2-norm for the hyperplane weights w and a 1-norm for the misclassification errors $\xi \in \mathbb{R}^n$, this takes the following form:

$$\begin{aligned} \min_{w,w_0,\xi} \quad & \frac{1}{2} w^T w + \tau e^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \geq e - \xi \\ & \xi \geq 0 \end{aligned} \tag{3}$$

where e is the vector of all ones, and τ is a positive constant that parametrises the problem.

Due to the convex nature of the problem, a Lagrangian function associated with (3) can be formulated,

$$\mathcal{L}(w, w_0, \xi, z, \nu) = \quad \frac{1}{2} w^T w + \tau e^T \xi - \sum_{i=1}^{n} z_i [y_i (w^T x_i + w_0) - 1 + \xi_i] - \nu^T \xi$$

where $\nu \in \mathbb{R}^n$ is the vector of Lagrange multipliers associated with the non-negativity constraint on ξ. The solution to (3) will be at the saddle point of the Lagrangian. Partially differentiating the Lagrangian function gives relationships between the primal variables w, w_0 and ξ, and the dual variables z at optimality:

$$\begin{aligned} & w = XYz \\ & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned}$$

Substituting these relationships back into the Lagrangian function gives the dual problem formulation

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y X^T X Y z - e^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{4}$$

Measuring the misclassification error using $\|\xi\|_2$ rather than $\|\xi\|_1$ is also standard practice. The primal formulation is the QP

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2} w^T w + \frac{\tau}{2} \xi^T \xi \\ \text{s.t.} \quad & Y(X^T w + w_0 e) \geq e - \xi \\ & \xi \geq 0 \end{aligned}$$

and the dual formulation becomes

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T (YX^T XY + \frac{1}{\tau} I) z - e^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{5}$$

The relationship $w = XYz$ holds for the 2-norm classification problem. Using the form $Q = (XY)^T(XY)$ enabled by the linear kernel, we can rewrite the quadratic objective in terms of w, and ensure the relationship between w and z to hold at optimality by introducing it into the constraints. Consequently, we can state the classification problem (4) as the following separable QP:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2} w^T w - e^T z \\ \text{s.t.} \quad & w - XYz = 0 \\ & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{6}$$

The quadratic matrix in the objective is no longer dense, but simplified to the diagonal matrix

$$Q = \begin{bmatrix} I_m & 0 \\ 0 & 0_n \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$$

while the constraint matrix is in the form:

$$A = \begin{bmatrix} I_m & -XY \\ 0 & y^T \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+n)}.$$

Determining the Newton step requires calculating the matrix product:

$$\begin{aligned} M &\equiv A(Q + \Theta^{-1})^{-1} A^T \\ &= \begin{bmatrix} (I_m + \Theta_w^{-1})^{-1} + XY\Theta_z YX^T & -XY\Theta_z y \\ -y^T \Theta_z YX^T & y^T \Theta_z y \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}. \end{aligned} \tag{7}$$

We need to solve $A(Q+\Theta^{-1})^{-1}A^T \Delta\lambda = r$ for $\Delta\lambda$. Building the matrix (7) is the most expensive operation, of order $\mathcal{O}(n(m+1)^2)$, while inverting the resulting matrix is of order $\mathcal{O}((m+1)^3)$.

[11] developed a near-equivalent formulation for M, through successive block eliminations of the general form of the QP, as part of their OOQP software. Their software appears to have received little attention from the machine learning community.

To determine the hyperplane, we also require the value of the bias w_0, a variable in the primal problem (3). Note that the element of λ corresponding to the constraint $y^T z = 0$ is in fact the variable w_0. Using our approach and a primal-dual interior point method, we can obtain w_0 directly from the solver.

This is in contrast to active-set methods, where w_0 has to be estimated from a subset of z values.

We use the same technique to develop a formulation for the 2-norm SVM (5), leading to a separable QP with the following diagonal Hessian matrix Q:

$$Q = \begin{bmatrix} I_m & 0 \\ 0 & \frac{1}{\tau} I_n \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}.$$

3.2 Universum SVM

An approach to binary classification was proposed [32] where the problem is augmented with an additional data set belonging to the same domain (but not the same classes) as the classification data, called the Universum [31], as intuitively it captures a general backdrop. The SVM is trained to label points from the distributions of the binary classification sets, but make no strong statement for the Universum distribution.

Let $\mathcal{C}$ be the set of classification points, with data $X_{\mathcal{C}}$ and labels $y_{\mathcal{C}}$. Similarly, let $\mathcal{U}$ be the Universum set with data $X_{\mathcal{U}}$ and no labels. As with normal binary classification, data points are penalized for being on the wrong side of the hyperplane margin, measured by error $\xi_{\mathcal{C}} \in \mathbb{R}^{|\mathcal{C}|}$. Samples in the Universum set should lie close to the hyperplane; $\xi_{+\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}|}$ and $\xi_{-\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}|}$ are the errors if they are more than ϵ above or below the hyperplane. It is possible to use different parameters for misclassification errors in the two sets, here shown as $\tau_{\mathcal{C}}$ and $\tau_{\mathcal{U}}$. The primal formulation is then:

$$\begin{aligned} \min_{w,w_0,\xi_{\mathcal{C}},\xi_{+\mathcal{U}},\xi_{-\mathcal{U}}} \quad & \frac{1}{2} w^T w + \tau_{\mathcal{C}} e^T \xi_{\mathcal{C}} + \tau_{\mathcal{U}} \left(e^T \xi_{+\mathcal{U}} + e^T \xi_{-\mathcal{U}} \right) \\ \text{s.t.} \quad & X_{\mathcal{C}}^T w + w_0 e \geq e - \xi_{\mathcal{C}} \\ & X_{\mathcal{U}}^T w + w_0 e \geq \epsilon e - \xi_{+\mathcal{U}} \\ & X_{\mathcal{U}}^T w + w_0 e \leq -\epsilon e + \xi_{-\mathcal{U}} \\ & \xi_{\mathcal{C}}, \xi_{+\mathcal{U}}, \xi_{-\mathcal{U}} \geq 0. \end{aligned}$$

A dual formulation can be developed by following the procedure described in Section 3.1 of forming the Lagrangian and partially differentiating with respect to the primal variables. Let us define the block matrices and vectors $X = \begin{bmatrix} X_{\mathcal{C}} X_{\mathcal{U}} X_{\mathcal{U}} \end{bmatrix}$, $y = \begin{bmatrix} y_{\mathcal{C}} \\ e_{\mathcal{U}} \\ -e_{\mathcal{U}} \end{bmatrix}$, $Y = \text{diag}(y)$, and dual variables for the hyperplane constraints $z = \begin{bmatrix} z_{\mathcal{C}} \\ z_{+\mathcal{U}} \\ z_{-\mathcal{U}} \end{bmatrix}$. Using this notation, the dual formulation

becomes:

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Y X^T X Y z + \begin{bmatrix} -e_{\mathcal{C}} \\ \epsilon e_{+\mathcal{U}} \\ \epsilon e_{-\mathcal{U}} \end{bmatrix}^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \leq z_{\mathcal{C}} \leq \tau_{\mathcal{C}} e_{\mathcal{C}} \\ & 0 \leq z_{+\mathcal{U}}, z_{-\mathcal{U}} \leq \tau_{\mathcal{U}} e_{\mathcal{U}}. \end{aligned}$$

Using the relationship between w and z at optimality

$$w = X_{\mathcal{C}} Y_{\mathcal{C}} z_{\mathcal{C}} + X_{\mathcal{U}} z_{+\mathcal{U}} - X_{\mathcal{U}} z_{-\mathcal{U}}$$

the Universum classification problem can be transformed into an equivalent separable formulation

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2} w^T w + \begin{bmatrix} -e_{\mathcal{C}} \\ \epsilon e_{+\mathcal{U}} \\ \epsilon e_{-\mathcal{U}} \end{bmatrix}^T z \\ \text{s.t.} \quad & w = XYz \\ & y^T z = 0 \\ & 0 \leq z_{\mathcal{C}} \leq \tau_{\mathcal{C}} e_{\mathcal{C}} \\ & 0 \leq z_{+\mathcal{U}}, z_{-\mathcal{U}} \leq \tau_{\mathcal{U}} e_{\mathcal{U}}. \end{aligned}$$

3.3 Ordinal regression

Ordinal regression refers to a learning technique that bridges classification and metric regression. Training samples are labelled with an ordinal number; in other words the classification categories are ranked. The task of the supervised learning problem is to predict the position on the ordinal scale of new samples. Unlike metric regression problems, the label numbers are discrete and the metric distances between labels do not have any real significance, while unlike multiple classification problems, ordering information is present.

Several approaches have been proposed to move beyond using multiple classification techniques or naively transforming the ordinal scales into numerical values and solving as a standard regression problem. In the formulation of [16], the goal is to learn a function $f(x) = w^T x + w_0$ which correctly orders the samples, so that $f(x_i) > f(x_j) \Leftrightarrow y_i > y_j$ for any pair of examples (x_i, y_i) and (x_j, y_j). Using the set of pairings $\mathcal{P} = \{(i,j) : y_i > y_j\}$, the authors formulate the following ordinal regression SVM:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2} w^T w + \tau \sum_{(i,j)\in\mathcal{P}} \xi_{ij} & \\ \text{s.t.} \quad & w^T (x_i - x_j) \geq 1 - \xi_{ij} & \forall (i,j) \in \mathcal{P} \\ & \xi_{ij} \geq 0 & \forall (i,j) \in \mathcal{P}. \end{aligned}$$

The objective promotes a large-margin linear function $f(x)$ that also minimizes the number of pairs of training examples that are incorrectly ordered. The formulation has the same structure as the classification SVM, and so it is open to the reformulation in Section 3.1.

There are two main disadvantages with the above formulation. The first is that the hyperplane bias w_0 does not play a role in the optimization problem, and has to be estimated afterwards. The second disadvantage is the number of constraints and the number of variables ξ grow quadratically with the training data set size. Partly to address that, two new approaches were proposed [3] for support vector ordinal regression where the size of the training problem is linear in the number of samples. The first formulation ("explicit thresholds") takes only the ranks immediately adjacent to each separating hyperplane to determine each threshold w_0^j. They introduce the constraints $w_0^{(j-1)} \geq w_0^j \quad \forall j$ explicitly on the thresholds to enforce the correct ordering. The reverse ordering of w_0^j is due to us using $w^T x + w_0^j = 0$ to define the hyperplane. Assume that there are r classes, indexed with $j \in \mathcal{J} = \{1, 2, \ldots, r\}$, each with n^j data samples. $r-1$ parallel hyperplanes separate the classes; the hyperplane with bias w_0^j separates class j from class $j+1$. $X^j \in \mathbb{R}^{m \times n^j}$ is the data matrix for class j. We define the misclassification error vector $\xi_+^j \in \mathbb{R}^{n^j}$ and dual variables $z_+^j \in \mathbb{R}^{n^j}$ for points in class j which should lie above the hyperplane $j-1$, and similarly errors $\xi_-^j \in \mathbb{R}^{n^j}$ and dual variables $z_-^j \in \mathbb{R}^{n^j}$ for points in class j below hyperplane j. Variables ξ_+^j and z_+^j are defined for all classes $j = 1, \ldots, r-1$, while ξ_+^j and z_+^j are defined for all classes $j = 2, \ldots, r$, but we can write a simplified but equivalent formulation if we add auxiliary variables $\xi_+^1, z_+^1, \xi_-^r, z_-^r = 0$. We also introduce dual variables $\beta^j \in \mathbb{R}$ for each of the ordering constraints. Again it simplifies the formulation if we set $w_0^0 = +\infty$ and $w_0^r = -\infty$. Then, with $j = 1, \ldots, r$, the primal formulation is:

$$
\begin{aligned}
\min_{w, w_0^j, \xi_-^j, \xi_+^j} \quad & \frac{1}{2} w^T w + \tau \sum_{j=1}^{r} \left(e^T \xi_-^j + e^T \xi_+^j \right) & \\
\text{s.t.} \quad & (X^j)^T w + w_0^j e \leq -e + \xi_-^j & \forall j \\
& (X^j)^T w + w_0^{(j-1)} e \geq e - \xi_+^j & \forall j \\
& w_0^{j-1} \geq w_0^j & \forall j \\
& \xi_-^j, \xi_+^j \geq 0 & \forall j.
\end{aligned}
$$

By following the same Lagrange duality technique as Section 3.1, and using the relationship between w and $(z_+^j - z_-^j)$ at optimality, an equivalent separable

formulation is:

$$\begin{aligned}
\min_{w,z_-,z_+,\beta} \quad & \frac{1}{2}w^T w - \sum_j e^T(z_+^j + z_-^j) \\
\text{s.t.} \quad & w = \sum_j X^j(z_+^j - z_-^j) \\
& e^T z_-^j + \beta^j = e^T z_+^{j+1} + \beta^{j+1} \quad \forall j \\
& 0 \le z_-^j, z_+^j \le \tau e \quad \forall j \\
& \beta^j \ge 0 \quad \forall j.
\end{aligned}$$

In the second formulation ("implicit thresholds") of [3], there are no constraints to correctly order the hyperplane biases. Instead, samples from all of the classes are used to define each threshold, and this approach ensures the correct ordering. X^j is defined as before. $\xi_+^{jk} \in \mathbb{R}^{n^k}$ is the misclassification error vector for class k that should lie above the hyperplane j, and similarly $\xi_-^{jk} \in \mathbb{R}^{n^k}$ for classes lying below hyperplane j. $z_+^{jk} \in \mathbb{R}^{n^k}$ and $z_-^{jk} \in \mathbb{R}^{n^k}$ are the dual variables for the hyperplane constraints. With $j = 1, \ldots, r-1$, the primal formulation is then:

$$\begin{aligned}
\min_{w,w_0,\xi_-,\xi_+} \quad & \frac{1}{2}w^T w + \tau \sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} e^T \xi_-^{jk} \quad + \sum_{k=j+1}^{r} e^T \xi_+^{jk} \right) \\
\text{s.t.} \quad & (X^k)^T w + w_0^j e \le -e + \xi_-^{jk} \quad \forall j \text{ and } k = 1, \ldots, j \\
& (X^k)^T w + w_0^j e \ge e - \xi_+^{jk} \quad \forall j \text{ and } k = j+1, \ldots, r \\
& \xi_-^{jk} \ge 0 \quad \forall j \text{ and } k = 1, \ldots, j \\
& \xi_+^{jk} \ge 0 \quad \forall j \text{ and } k = j+1, \ldots, r.
\end{aligned}$$

Using the relationship $w = -\sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} X^k z_-^{jk} - \sum_{k=j+1}^{r} X^k z_+^{jk} \right)$, our equivalent separable formulation is:

$$\begin{aligned}
\min_{z_-,z_+} \quad & w^T w - \sum_k e^T \left(\sum_{j=1}^{k-1} z_+^{jk} + \sum_{j=k}^{r-1} z_-^{jk} \right) \\
\text{s.t.} \quad & w + \sum_{j=1}^{r-1} \left(\sum_{k=1}^{j} X^k z_-^{jk} - \sum_{k=j+1}^{r} X^k z_+^{jk} \right) = 0 \\
& \sum_{k=1}^{j} e^T z_-^{jk} = \sum_{k=j+1}^{r} e^T z_+^{jk} \quad \forall j \\
& 0 \le z_-^{jk} \le \tau e \quad \forall j \text{ and } k = 1, \ldots, j \\
& 0 \le z_+^{jk} \le \tau e \quad \forall j \text{ and } k = j+1, \ldots, r.
\end{aligned}$$

3.4 Regression

Support Vector Regression (SVR) uses similar techniques to learn a mapping between the input vector x and a real-valued target value y. In ϵ-insensitive SVR, the loss function is defined as $L^{\epsilon} \equiv max(0, |y - f(x)| - \epsilon)$. This loss is represented by the errors ξ_i and $\hat{\xi}_i$, which are the losses if the predicted value $f(x_i)$ of the point x_i is above or below the band around y of half-width ϵ. Fuller descriptions are again given in [5, 29, 30].

The dual variables $z, \hat{z}$ are the Lagrange multipliers relating to the two sets of constraints. The objective function minimizes risk using the SRM principle (balancing the complexity of the function against misclassifications in the training data), resulting in the primal optimization problem

$$\begin{aligned} \min_{w,\xi,\hat{\xi}} \quad & \frac{1}{2} w^T w + \tau e^T (\xi + \hat{\xi}) \\ \text{s.t.} \quad & y - (w^T X + w_0 e) \leq \epsilon e + \xi \\ & (w^T X + w_0 e) - y \leq \epsilon e + \hat{\xi} \\ & \xi, \hat{\xi} \geq 0 \end{aligned}$$

and its dual

$$\begin{aligned} \min_{z,\hat{z}} \quad & \frac{1}{2} (z - \hat{z})^T X^T X (z - \hat{z}) - y^T (z - \hat{z}) + \epsilon e^T (z + \hat{z}) \\ \text{s.t.} \quad & e^T (z - \hat{z}) = 0 \\ & 0 \leq z, \hat{z} \leq \tau e. \end{aligned} \tag{8}$$

The relationship between w and $(z, \hat{z})$ is now $w = X(z - \hat{z})$.

We exploit separability in a similar way for Support Vector Regression, by introducing into the standard dual formulation (8) the auxiliary variable $\bar{z} \equiv z - \hat{z}$ and the relationship $w = X(z - \hat{z})$:

$$\begin{aligned} \min_{w,z,\hat{z},\bar{z}} \quad & \frac{1}{2} w^T w + \epsilon e^T (z + \hat{z}) - y^T \bar{z} \\ \text{s.t.} \quad & -z + \hat{z} + \bar{z} = 0 \\ & w - X\bar{z} = 0 \\ & e^T \bar{z} = 0 \\ & 0 \leq z, \hat{z} \leq \tau e. \end{aligned}$$

We define decision variables $(w, z, \hat{z}, \bar{z})$ and the corresponding constraint matrix

$$A = \begin{bmatrix} 0 & -I & I & I \\ I & & & -X \\ & & & e^T \end{bmatrix},$$

while both Q and Θ are diagonal matrices. We need to set bounds on $\bar{z}$ (i.e., $-\tau e \leq \bar{z} \leq \tau e$) so that $\Theta_{\bar{z}}$ is defined. The matrix $M \equiv A(Q+\Theta^{-1})^{-1}A^T$ requiring factorization is therefore

$$M = \begin{bmatrix} \Theta_z + \Theta_{\hat{z}} + \Theta_{\bar{z}} & -\Theta_{\bar{z}}X^T & \Theta_{\bar{z}}e \\ -X\Theta_{\bar{z}} & (I_m + \Theta_w^{-1})^{-1} + X\Theta_{\bar{z}}X^T & -X\Theta_{\bar{z}}e \\ e^T\Theta_{\bar{z}} & -e^T\Theta_{\bar{z}}X & e^T\Theta_{\bar{z}}e \end{bmatrix}.$$

The Cholesky factorization LDL^T of matrix K can be computed efficiently using the Schur complement method.

$$LDL^T = M = \begin{bmatrix} I_n & \\ F & L_m \end{bmatrix} \begin{bmatrix} D_n & \\ & D_m \end{bmatrix} \begin{bmatrix} I_n & F^T \\ & L_m^T \end{bmatrix}$$

where

$$D_n = \Theta_z + \Theta_{\hat{z}} + \Theta_{\bar{z}}$$
$$F = \begin{bmatrix} -X \\ e^T \end{bmatrix} \Theta_{\bar{z}} D_n^{-1},$$

while L_m and D_m are found from the Cholesky factorization

$$L_m D_m L_m^T = \begin{bmatrix} (I_m + \Theta_w^{-1})^{-1} & \\ & 0 \end{bmatrix} + \begin{bmatrix} -X \\ e^T \end{bmatrix} (\Theta_{\bar{z}} - \Theta_{\bar{z}} D_n^{-1} \Theta_{\bar{z}}) \begin{bmatrix} -X^T & e \end{bmatrix}.$$

The formation of this smaller matrix is an $\mathcal{O}(n(m+1)^2)$ operation, while the factorization is of order $\mathcal{O}((m+1)^3)$. Calculation of the other variables require $\mathcal{O}(n)$ operations.

4 Numerical experiments and results

We implemented the 1-norm formulation (6) in `HOPDM` [14; 1] which was modified to perform matrix multiplications in dense mode using the BLAS library [21]. The experiments were performed using an Intel Pentium 4 PC running at 3GHz, with 1GB RAM and 1024KB cache. BLAS functions were provided by Intel's Maths Kernel Library [1].

A comparison of several techniques for calculating $A(Q+\Theta^{-1})A^T$ is shown in Table 1. Any technique that takes advantage of the structure of the problem gave better performance than multiplying the elements individually. Computing $A(Q+\Theta^{-1})A^T$ by outer products most directly exploits the structure (as $(Q+\Theta^{-1})$ is diagonal), but the computation using `DGEMM` on blocks of 32 data points at a time gave the best performance, probably due to better use of the CPU's cache.

The numerical results in this section were based on artificial data sets. The training data sets used were created by uniformly sampling points in the space $[-1,+1]^m$. A separating hyperplane was defined by choosing random integer values for w in the range [-1000,1000]. Points were labelled based on the

[1] http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/index.htm

Technique	Non-optimized BLAS library	Intel optimized BLAS library
Multiplication of individual elements	296.36	296.36
Outer products (`DSYRK`)	25.45	20.50
Matrix-vector multiplication (`DGEMV`)	27.18	22.00
Block-based matrix multiplication (`DGEMM`)	15.27	8.58

Table 1 Time taken (in seconds) to train an SVM, comparing different techniques to calculate $A(Q + \Theta^{-1})^{-1}A^T$. The data set given contained 5000 points of 320 attributes. 14 IPM iterations were required.

hyperplane. For the non-separable data sets, the required proportion of data points were randomly selected and misclassified. In contrast to the training data, the test data sets contain no misclassified points.

4.1 Confirmation of scalability

The complexity analysis above gave the computations required for each iteration as $\mathcal{O}(nm^2)$ if $n \gg m$. To verify this, the software was trained first using data sets with 255 features. Figure 1(a) shows that the length of time taken by an iteration varies linearly with the number of samples n. However, the total time taken by the algorithm increases super-linearly with the number of samples, as the number of iterations required also increases slowly.

The experiment was repeated for the number of features, using a data set of 20,000 samples, and the number of features varied. Figure 1(b) shows that, once there is a reasonably large number of features, approximately $m > 250$, the algorithm scales quadratically with m, while the number of iterations required remains roughly the same.

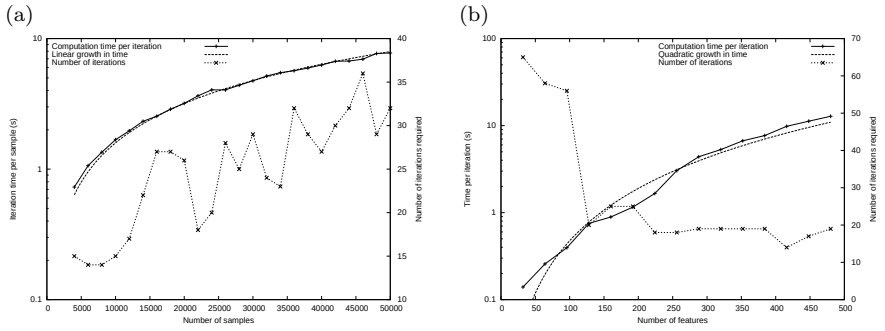

Fig. 1 The computational scalability of iteration time is predictable, and results confirm $\mathcal{O}(nm^2)$. The number of iterations required is less predictable, but grows slowly with n. (a) Computational complexity and iteration count with respect to the number of samples n, using fully separable data sets with 255 features. (b) Computational complexity and iteration count with respect to the number of features m. Data sets were fully separable with 20,000 samples.

4.2 Bounds on w

In the formulation (6) w is free, while the standard IPM formulation (1) requires all variables to be in the positive quadrant. While free variables can be implemented as $w \equiv w_+ - w_-$, where $w_+, w_- \geq 0$, this approach suffers from numerical difficulties: logically one of each pair (w_+, w_-) would be zero, but this is prevented by the logarithmic barrier used in IPM. The approach we adopted is to define bounds $l_w \leq w \leq u_w$, and the problem can then be adjusted appropriately to shift the bounds to $0 \leq w' \leq u_w - l_w$. From (6), $w = XYz$, so bounds can be safely set as $\tau \sum_i \min(y_i x_{ij}, 0) \leq w_j \leq \tau \sum_i \max(y_i x_{ij}, 0)$. For problems where the solution set of support vectors is sparse, the optimal values for w differ substantially from either bound. Since large bounds affect the numerical accuracy of the algorithm, it is useful to tighten the bounds to within say a couple of orders of magnitude of the true values of w once these are known (e.g. when searching for the best parameters through repeated training).

4.3 Accuracy due to termination criteria

Several termination criteria are possible. Normally the measure of most interest for an optimization problem is the value of the objective function, and the algorithm stops when this value is reached to within a set relative error, e.g. 10^{-8}, but for SVMs the objective value is not of interest so may not be a good basis for termination. Similarly, the errors associated with primal and dual feasibility can be monitored, and the algorithm terminated when these are within a small tolerance.

The approach normally used for SVM is to monitor the set of support vectors, and terminate the algorithm when this is stable. The KKT complementarity conditions are used to determine the support vectors.

With the formulation presented in this paper, we have access to the weights variables w directly. It is therefore possible to monitor these values, and measure the change in the angle ϕ of the normal to the hyperplane between iteration $i-1$ and i:

$$\cos\phi = \frac{(w^{(i-1)})^T w^{(i)}}{\|w^{(i-1)}\| \|w^{(i)}\|}$$

We conducted experiments to see how these measures relate to classification accuracy, using a training set of 20,000 samples and 255 features, with 5% misclassifications, and a separable test set of the same size.

Figure 2 shows how the duality gap and $\sin\phi$ decrease as the IPM algorithm progresses. Primal feasibility was reached quickly, while it took the algorithm longer to attain dual feasibility. All measures were sensitive to the scale of the bounds on w, which made it hard to define a set tolerance for any of the measures. In particular, the values of w decrease with each iteration, so it is not useful to monitor these to see if they are converging to their final values.

A noticeable feature of the figures is that a high classification accuracy is achieved at an early stage in the algorithm, long before the number of support vectors has stabilized, indicating that the hyperplane has been accurately identified at this point. Although at this stage the values of the weights change in scale, proportionally they are stable, as can be seen by measuring $\sin\phi$ (Figure 2). Once suitable bounds on w have been established, a tolerance of 10^{-4} on $\sin\phi$ could be used to give earlier termination.

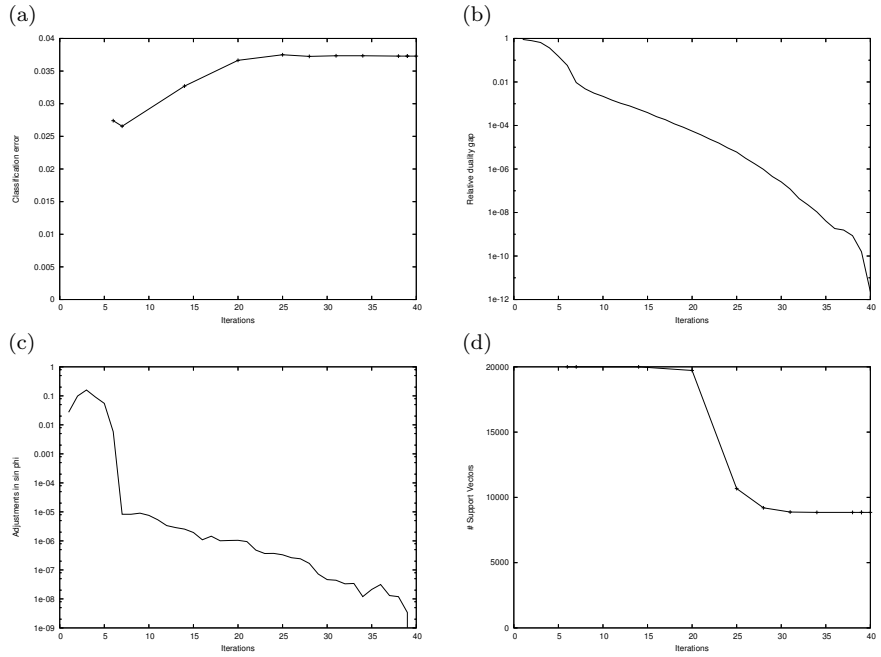

Fig. 2 Performance of the algorithm relative to the number of IPM iterations, using a training data set with 5% of points mis-classified, 20,000 samples and 255 attributes. (a) Classification error using an unseen test set. (b) Error in the value of the objective. (c) Change in angle of the normal to the separating hyperplane. (d) The number of support vectors.

4.4 Multiple correctors

The use of multiple correctors [15] can reduce the number of IPM iterations required, by improving the centrality of the current iterate. Several correctors can be calculated by repeatedly solving $M\Delta\lambda = -\hat{r}_b$ for $\Delta\lambda$. The same factorization of M is used for all the correctors in an iteration, so it is advantageous to perform multiple corrections when the effort involved the backsolves (here $\mathcal{O}(nm+m^2)$) is significantly less than that of factorizing M (here $\mathcal{O}(nm^2+m^3)$).

We conducted experiments to show the comparative performance of the algorithm using multiple correctors, against the algorithm using a single Mehrotra's corrector. Both the number of iterations and the overall time were improved by using multiple correctors. For example, using a data set of 20,000 samples and 255 features, an average of 2.8 correctors were used in each iteration, and the optimization required 2 fewer iterations.

4.5 Stability in case of near-linear dependency in X

We used the data set of [13] that caused an algorithm using the Sherman-Morrison-Woodbury update to fail. This data set (shown in Figure 3) causes degeneracy in the matrix (XY), as there are multiple data points which lie along the separating hyperplanes. Scaling one of the dimensions accentuates the numerical instability. With our algorithm, there was no penalty in performance, with the number of IPM iterations required always around 20 no matter the scaling imposed on the data set (this is similar performance to that reported by Goldfarb and Scheinberg for their Product Form Cholesky Factorization algorithm). Stability of our approach is a consequence of the use of primal-dual regularization [1]. Regularization improves any ill-conditioning related to the original data, and allows Cholesky factorization to be computed without severe loss of accuracy.

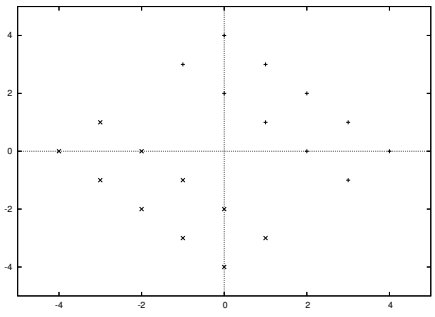

Fig. 3 The data set of [13] causing degeneracy in the constraint matrix.

4.6 Comparison against standard tools

To assess the performance of our algorithm `SVM-HOPDM`, we tested it against a range of state-of-the-art SVM tools: $\texttt{SVM}^{\texttt{light}}$ [18], $\texttt{SVM}^{\texttt{perf}}$ [19], `LibLinear` [17], `LibSVM` [2] and `SVMTorch` [4]. We also included the `SVM-QP` active set algorithm [9] and the IPM-based algorithm (`SVM-QP Presolve`) of Scheinberg that uses the Product Form Cholesky Factorization described earlier [13]. They were all used with their software performance options (such as cache size) set to their

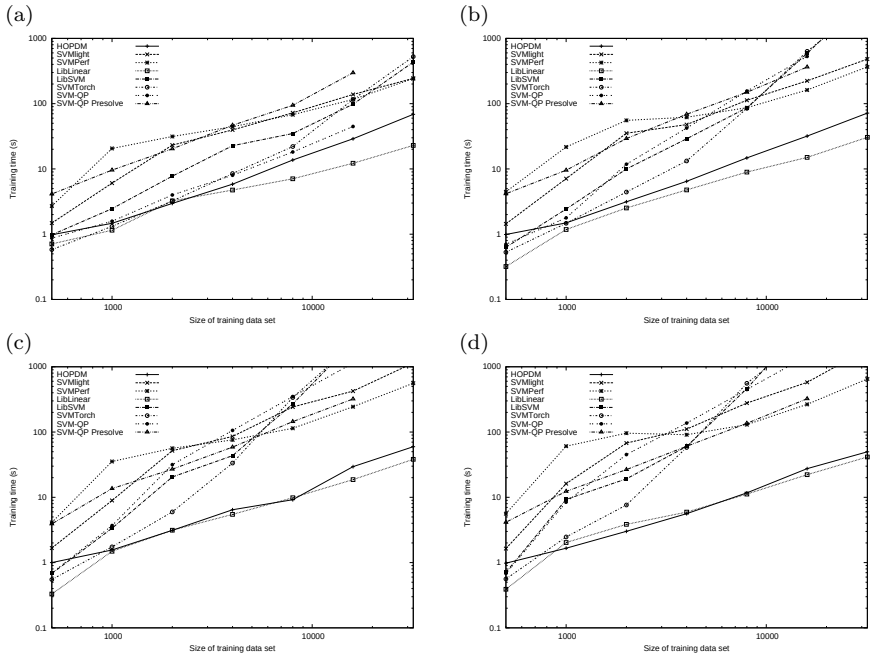

Fig. 4 Comparison of efficiency of `SVM-HOPDM` against other algorithms, with respect to data set size, as noise is increased. Artificial data sets used of 255 attributes. $\tau = 1$. (a) Fully separable. (b) 1% misclassified. (c) 5% misclassified. (d) 10% misclassified.

default values. We conducted SVM training experiments using synthetically-constructed data sets as described earlier, with 255 features. `SVMTorch` has been used as the comparison tool for other IPM-based techniques, e.g. [7, 10, 13].

Figure 4 shows the comparative efficiency of the algorithms as the size of the data set is increased, with a relatively low penalty for misclassifications ($\tau = 1$). For separable data sets (a) all the algorithms show linear or sublinear scaling. But the relative performance changes dramatically when the data set contains noise, as is typically the case with real-world data sets. We used synthetic data sets again, but this time introduced noise by choosing (b) 1%, (c) 5% and (d) 10% of the points randomly and swapping their target label. The computation time required by the active set methods is greatly increased by introducing noise, while other algorithms are less affected.

The experiments were repeated for a higher misclassification penalty of $\tau = 100$ (Figure 5). It can be clearly seen that all except the IPM algorithms are greatly affected by the level of noise, with training times increased by several orders of magnitude, or the algorithms fail to converge.

The training times of `SVM-HOPDM` and `SVM-QP Presolver`, both based on interior point methods, are similar in all eight cases, yet there was almost an order of magnitude difference between the two algorithms. This difference

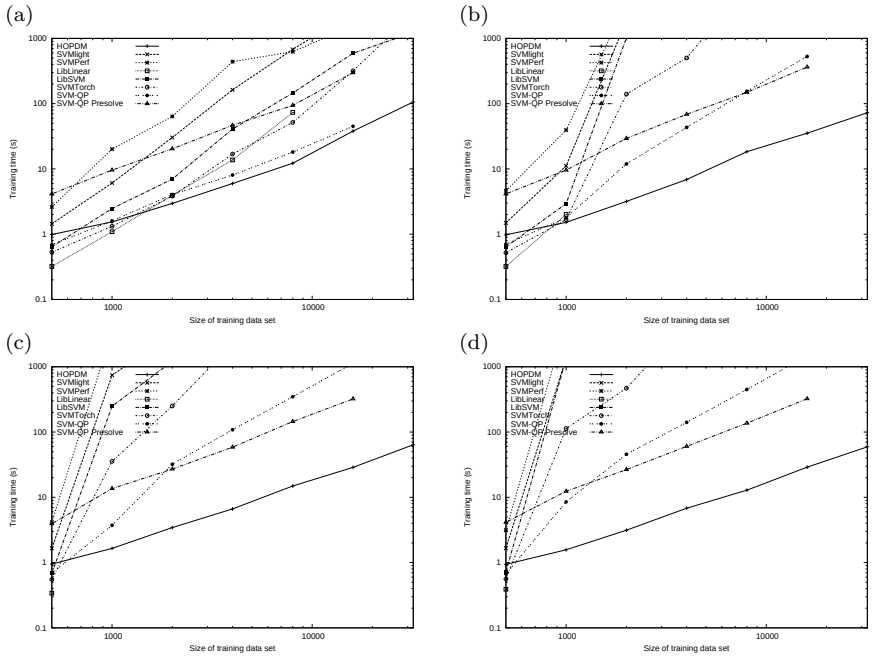

Fig. 5 Comparison of algorithm efficiency, similar to Figure 4 but with higher penalty for misclassifications ($\tau = 100$). (a) Fully separable. (b) 1% misclassified. (c) 5% misclassified. (d) 10% misclassified.

cannot be accounted for by a complexity analysis. We investigated this further using Valgrind's `Cachegrind` cache simulator[2] set as a Pentium 4 processor cache, and the results for four data sets are shown in Table 2. The algorithms required different numbers of iterations which complicated the comparison, so we considered only the functions associated with forming and solving the normal system matrix, and this accounted for some 70% to 80% of the instructions executed (the "coverage" in Table 2). The final two columns of the table show instruction count and runtime ratios for the two programs. It is clear that the number of executed instructions does not explain the whole increase in runtime. The number of data read cache misses (which is determined by how the algorithm accesses the data structure) is also an important factor in runtime performance, yet it is rarely discussed in comparisons of computational complexity of algorithms.

4.7 Real-world data sets

To investigate what performance results can be expected in real-world applications, we used the standard data sets Adult, Covtype, MNIST, SensIT and

[2] http://valgrind.org/

Data set		SVM-HOPDM			SVM-QP presolver			SVM-QP/ SVM-HOPDM	
n	m	Time	Cov.	D2mr	Time	Cov.	D2mr	Instructions	Time
10000	63	3.69	70%	2.68%	16.44	72%	6.99%	2.36	4.46
10000	127	10.66	83%	1.73%	55.46	80%	7.56%	1.94	5.21
10000	255	15.67	79%	1.34%	127.53	79%	7.72%	2.73	8.14
20000	63	9.13	76%	2.97%	40.74	76%	6.95%	2.15	4.46

Table 2 Comparison of SVM-HOPDM and SVM-QP Presolver in terms of instructions and cache misses. Synthetic data sets of dimension n samples and m features were used. Time is the total runtime of the program, running with hardware cache, in seconds. Coverage is the proportion of the program included in the instruction and cache miss count. D2mr is the proportion of Level 2 data read misses to total Level 2 data reads. The final two columns show the ratio between the two programs, for instructions and runtime. The increase in runtime of SVM-QP Presolver cannot be accounted for by instructions alone, and cache performance has a significant effect.

USPS.[3] Each problem was solved using a linear kernel with $\tau = 1, 10$ and 100. Table 3 shows the wall-clock times to perform the training (including time taken to read the data).

Dataset ($n \times m$)	τ	HOPDM	SVM$^{\text{light}}$	SVM$^{\text{perf}}$	LibLinear	LibSVM	SVMTorch	SVM-QP	SVM-QP presolve
Adult	1	16.5	87.7	280.7	1.6	192.4	621.8	164.5	188.8
32561 × 123	10	26.5	1043.3	3628.0	9.3	857.7	5046.0	284.1	206.8
	100	27.9	10447.4	29147.2	64.2	5572.1	44962.5	544.8	216.9
Covtype	1	47.7	992.4	795.6	8.5	2085.8	2187.9	731.8	405.6
150000 × 54	10	52.7	6021.2	12274.5	34.3	2516.7	10880.6	971.6	441.3
	100	55.4	66263.8	58699.8	235.2	6588.0	74418.1	1581.8	457.4
MNIST	1	79.6	262.9	754.1	9.3	197.1	660.1	233.0	1019.1
10000 × 780	10	83.4	3425.5	8286.8	65.4	1275.2	5748.1	349.4	1104.4
	100	86.2	NC	196789.0	NC	11456.4	54360.6	602.5	1267.1
SensIT	1	55.2	913.5	8418.3	53.6	2542.0	2814.4	535.2	456.7
78823 × 100	10	60.1	7797.4	> 125000	369.1	7867.8	21127.8	875.4	470.7
	100	63.6	NC	> 125000	NC	49293.7	204642.6	1650.1	489.3
USPS	1	13.2	15.0	40.9	4.4	10.4	7.7	51.2	117.4
7291 × 256	10	14.2	147.4	346.6	27.7	20.9	23.9	64.7	127.4
	100	14.3	1345.2	2079.5	NC	93.8	142.4	86.9	143.8

Table 3 Comparison of training times using real-world data sets. Each data set was trained using $\tau = 1, 10$ and 100. NC indicates that the method did not converge to a solution.

The same results are shown as a performance profile [6] in Figure 6. Here, the runtime $t_{s,p}$ for each solver $s \in \mathcal{S}$ on problem $p \in \mathcal{P}$ is transformed into a ratio to the fastest solver for problem p:

$$r_{s,p} = \frac{t_{s,p}}{\min_{s \in \mathcal{S}} t_{s,p}}.$$

[3] All datasets are available from the LibSVM collection at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. Due to memory restrictions, some data sets were reduced to the sizes given in Table 3. SVM-QP had tighter memory restrictions, so the datasets were further reduced and the times linearly scaled up; this is probably fair for the SVM-QP presolver but is rather favourable for the active set solver.

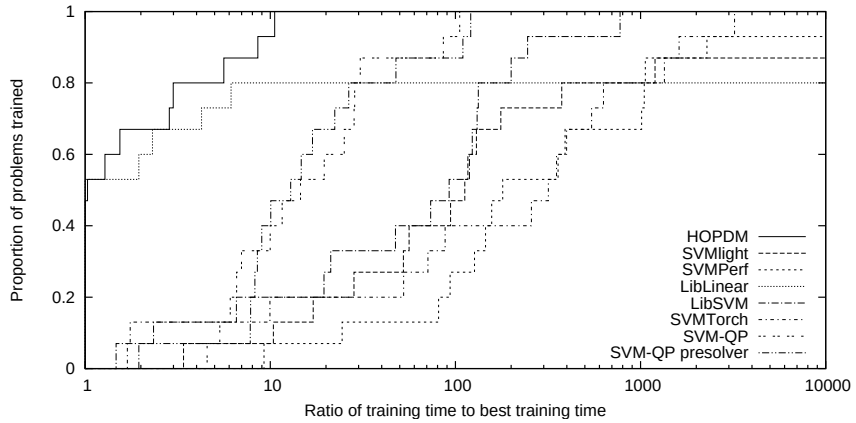

Fig. 6 Performance profile of the SVM tools on the problems in Table 3.

Dataset	τ	HOPDM	LIBLINEAR	SVMTorch
Adult	1	85.01%	84.98%	84.95%
32561 × 123	10	84.98%	84.96%	85.01%
	100	84.95%	84.95%	84.97%
Covtype	1	61.59%	61.59%	60.85%
150000 × 54	10	61.92%	61.92%	59.84%
	100	61.92%	61.92%	61.46%
MNIST	1	86.31%	86.31%	86.40%
10000 × 780	10	86.43%	86.40%	86.41%
	100	86.27%	—	???
SensIT	1	85.78%	85.42%	85.78%
78823 × 100	10	85.80%	85.43%	85.82%
	100	85.79%	85.45%	85.85%
USPS	1	96.41%	97.11%	97.11%
7291 × 256	10	97.21%	97.11%	97.21%
	100	97.01%	96.61%	96.66%

Table 4 Comparison of prediction accuracy on unseen test sets. For all except Covtype, we used the standard test sets. Covtype does not have a standard test set, so we used the first 150000 samples of the data set for training and the final 100000 samples as the test set (there was no overlap). The results show in terms of accuracy, our method is broadly equivalent to other methods.

The performance profile is the cumulative distribution function of these ratios for each solver

$$\rho_s(T) = \frac{\text{size}\{p \in \mathcal{P} : |r_{s,p} \leq T\}}{\text{size}\{\mathcal{P}\}},$$

that is the proportion of problems which can be solved with a runtime ratio $r_{s,p}$ less than T. The value of $\rho_s(1)$ is the proportion of problems that solver s wins over other solvers. The value of $\lim_{T\to\infty} \rho_s(T)$ is the proportion of

problems solved at all. A high proportion of problems solved with small ratios $r_{s,p}$ is shown by the profile being close to the upper left axes.

The results confirm that real-world data sets do tend to be noisy, as most methods take considerably longer with high τ misclassification penalty values. The performance profile highlights that LibLinear is the fastest for many of the problems, generally involving low values of τ. For higher values of τ, however, HOPDM is faster than the other solvers. This is due to the training time of our algorithm being roughly constant, relative to the value of τ. It was at most one order of magnitude slower than the fastest solver for any problem, which was the best performance of any of the solvers in this regard: other solvers were two or three orders of magnitude slower, or failed to converge at all. We consider this dependability, in terms of both predictable training times and ability to train with a wide range of τ values, to be a valuable property of our algorithm.

Table 4 confirms that there is no penalty to be paid in terms of prediction accuracy. Experiments using unseen test samples show that the prediction accuracy of our formulation is comparable with other methods.

5 Conclusions

Support Vector Machines are a powerful machine learning technique. However, it is not a trivial exercise to extend it to very large scale problems. Due to the Hessian in the standard dual formulation being completely dense, interior point methods have not traditionally been used. Instead, standard SVM tools have mainly been based around active-set methods. These work well for small and separable problems, but when the split between basic and non-basic variables becomes less clear (as is the case with noisy data sets), the performance of these algorithms starts to scale exponentially with the number of samples. Previous IPM-based approaches have exploited the structure of the linear kernel, to give algorithms with an overall complexity of $\mathcal{O}(nm^2)$. However, these algorithms have suffered from either numerical instability through use of the Sherman-Morrison-Woodbury formula, or memory caching inefficiencies.

In this paper we have presented a new, unified set of formulations for linear 1-norm and 2-norm classification, universum and ordinal classification, and ϵ-insensitive regression, which all exploit the separability of the Hessian in the objective of the standard SVM primal formulation, while keeping a small number of constraints as in the dual. Like the other IPM-based approaches, it has a per-iteration complexity of $\mathcal{O}(nm^2 + m^3)$. It relies upon Cholesky decomposition for its numerical stability, but the factorization is applied to all m features at once, allowing for a more efficient implementation in terms of memory caching.

Numerical experiments showed that the performance of the algorithm for large dense or noisy data sets is consistent and highly competitive, and in some cases can surpass all other approaches by a large margin. Unlike active set methods, performance is largely unaffected by noisy data. Using multiple

correctors, tightening the bounds on w, and monitoring the angle of the normal to the hyperplane all positively contributed to efficiency.

It is possible to extend these formulations to non-linear kernels, by approximating the positive semidefinite kernel matrix K with a low-rank outer product representation such as partial Cholesky factorization $LL^T \approx K$ [8]. This approach produces the first r columns of the matrix L (corresponding to the r largest pivots) and leaves the other columns as zero, giving an approximation of the matrix K of rank r. It is an attractive algorithm for partial decomposition, since its complexity is linear with the number of samples, it is faster than eigenvalue decomposition, and it exploits the symmetry of K. A separable formulation suitable for non-linear kernels is therefore:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2} w^T w - e^T z \\ \text{s.t.} \quad & w - L^T Y z = 0 \\ & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{9}$$

Computational complexity is $\mathcal{O}(nr^2 + nmr)$ for the initial Cholesky factorization, and $\mathcal{O}(nr^2 + r^3)$ for each IPM iteration.

It is also possible to develop this algorithm to handle very large scale problems in parallel. The key computation part is the calculation of the matrix M; as described earlier, this was handled on a block basis using the BLAS library. By dividing the sample points equally amongst the processors, the block-based matrix multiplications can be performed in parallel with no communication required between the processors. Then at the end of the multiplication, a single gather operation is required on the $(m+1) \times (m+1)$ matrix at each processor to form the matrix M and then factorize it. Implementation details are addressed in [33]. This could point the way forward for tackling large and complex data sets.

Acknowledgements

We would like to thank the anonymous reviewers for their careful reading of our manuscript and the insightful comments they provided.

References

1. Altman, A., Gondzio, J.: Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. Optimization Methods and Software **11**, 275–302 (1999)
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001). Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm
3. Chu, W., Keerthi, S.S.: New approaches to support vector ordinal regression. In: ICML '05: Proceedings of the 22nd international conference on machine learning, pp. 145–152. ACM, New York, NY, USA (2005)

4. Collobert, R., Bengio, S.: SVMTorch: support vector machines for large-scale regression problems. Journal of Machine Learning Research **1**, 143–160 (2001)
5. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press (2000)
6. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Mathematical Programming **91**(2), 201–13 (2002)
7. Ferris, M., Munson, T.: Interior point methods for massive support vector machines. SIAM Journal on Optimization **13**(3), 783–804 (2003)
8. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. Journal of Machine Learning Research **2**, 243–264 (2002)
9. Fine, S., Scheinberg, K.: INCAS: An incremental active set method for SVM. Tech. rep., IBM Research Labs, Haifa (2002)
10. Gertz, E.M., Griffin, J.D.: Support vector machine classifiers for large data sets. Technical memo, Argonne National Lab ANL/MCS-TM-289 (2005)
11. Gertz, E.M., Wright, S.J.: Object-oriented software for quadratic programming. ACM Transactions on Mathematical Software **29**(1), 58–81 (2003)
12. Goldfarb, D., Scheinberg, K.: A product-form Cholesky factorization method for handling dense columns in interior point methods for linear programming. Mathematical Programming **99**(1), 1–34 (2004)
13. Goldfarb, D., Scheinberg, K.: Solving structured convex quadratic programs by interior point methods with application to support vector machines and portfolio optimization. Submitted for publication (2005)
14. Gondzio, J.: HOPDM: a fast LP solver based on a primal-dual interior point method. European Journal of Operational Research **85**, 221–225 (1995)
15. Gondzio, J.: Multiple centrality corrections in a primal-dual method for linear programming. Computational Optimization and Applications **6**, 137–156 (1996)
16. Herbrich, R., Graepel, T., Obermayer, K.: Advances in Large Margin Classifiers, chap. Large margin rank boundaries for ordinal regression. MIT Press (2000)
17. Hsieh, C.J., Chang, K.W., Lin, C.J., Keerthi, S.S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: ICML '08: Proceedings of the 25th international conference on machine learning (2008)
18. Joachims, T.: Making large-scale support vector machine learning practical. In: B. Schölkopf, C.J.C. Burges, A.J. Smola (eds.) Advances in Kernel Methods: Support Vector Learning, pp. 169–184. MIT Press (1999)
19. Joachims, T.: Training linear SVMs in linear time. In: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 217–226. ACM, New York, NY, USA (2006)
20. Keerthi, S.S., Chapelle, O., DeCoste, D.: Building support vector machines with reduced classifier complexity. Journal of Machine Learning Research **7**, 1493–1515 (2006)
21. Lawson, C.L., Hanson, R.J., Kincaid, D.R., Krogh, F.T.: Algorithm 539: Basic Linear Algebra Subprograms for Fortran usage [F1]. ACM Transactions on Mathematical Software **5**(3), 324–325 (1979)
22. Lee, Y.J., Mangasarian, O.L.: RSVM: Reduced support vector machines. In: Proceedings of the SIAM International Conference on Data Mining. SIAM, Philadelphia (2001)
23. Lucidi, S., Palagi, L., Risi, A., Sciandrone, M.: A convergent decomposition algorithm for support vector machines. Computational Optimization and Applications **38**, 217–234 (2007)
24. Mangasarian, O.L., Musicant, D.R.: Successive overrelaxation for support vector machines. IEEE Transactions on Neural Networks **10**(5), 1032–1037 (1999)
25. Mészáros, C.: The separable and non-separable formulations of convex quadratic problems in interior point methods. Tech. Rep. WP 98-3, Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest (1998)
26. Osuna, E., Freund, R., Girosi, F.: An improved training algorithm for support vector machines. In: J. Principe, L. Gile, N. Morgan, E. Wilson (eds.) Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop, pp. 276–285. IEEE (1997)
27. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: B. Schölkopf, C.J.C. Burges, A.J. Smola (eds.) Advances in Kernel Methods:

Support Vector Learning, pp. 185–208. MIT Press (1999)
28. Vanderbei, R.J.: Linear Programming Foundations and Extensions. Kluwer, Boston (1997)
29. Vapnik, V.: Statistical Learning Theory. Wiley (1998)
30. Vapnik, V.: The Nature of Statistical Learning Theory, 2nd edn. Springer (1999)
31. Vapnik, V.: Transductive inference and semi-supervised learning. In: O. Chapelle, B. Schölkopf, A. Zien (eds.) Semi-supervised learning, chap. 24, pp. 454–472. MIT Press (2006)
32. Weston, J., Collobert, R., Sinz, F., Bottou, L., Vapnik, V.: Inference with the Universum. In: ICML '06: Proceedings of the 23rd international conference on machine learning, pp. 1009–1016. ACM (2006)
33. Woodsend, K., Gondzio, J.: Hybrid MPI/OpenMP parallel support vector machine training. Technical Report ERGO 09-001, School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh (2009). Submitted for publication.
34. Wright, S.J.: Primal-dual interior-point methods. S.I.A.M. (1997)