Parallel support vector machine training with nonlinear kernels

Kristian Woodsend * Jacek Gondzio †

School of Mathematics, University of Edinburgh
The King's Buildings, Edinburgh, EH9 3JZ, UK

November 30, 2007

Abstract

A parallel implementation of Support Vector Machine training for problems involving nonlinear kernels has been developed. The kernel matrix is approximated by a partial Cholesky decomposition. Theoretical issues associated with constructing the best possible (and yet computationally efficient) approximation are discussed in detail and implemented in OOPS. The structure of the augmented system matrix is exploited to partition data and computations amongst parallel processors efficiently. The new implementation has been applied to solve problems which involve very large data sets. Excellent parallel efficiency was observed on such problems.

1 Introduction

Support Vector Machines (SVMs) are powerful machine learning techniques for classification and regression, and they offer state-of-the-art performance. The training of an SVM is computationally expensive and relies on optimization. The core of the approach is a dense convex quadratic optimization problem (QP), which for a general-purpose QP solver scales cubically with the number of data points ($\mathcal{O}(n^3)$). 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.

*Email: k.j.woodsend@sms.ed.ac.uk
†Email: j.gondzio@ed.ac.uk

The standard approach to handle this problem is to build a solution by solving a sequence of small scale problems, e.g. Decomposition (Osuna et al., 1997) or Sequential Minimal Optimization (Platt, 1999). State-of-the-art software such as `SVMlight` (Joachims, 1999) and `SVMTorch` (Collobert and Bengio, 2001) 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. Empirical computation time measurements on `SVMTorch` show that training time grows much closer to $\mathcal{O}(n^2)$ than $\mathcal{O}(n^3)$ of a general purpose QP solver, but with noisy data, the set of support vectors is not so clear, and the performance of these algorithms deteriorates (Woodsend and Gondzio, 2007).

The standard active set technique is essentially sequential, choosing a small subset of variables to form the active set at each iteration, based upon the results of the previous selection. Improving the computation time through parallelization of the algorithm is difficult due to dependencies between subsequent iterations, and it is not clear how to implement this efficiently.

Parallelization schemes so far proposed have involved splitting the training data to give smaller, separable optimization sub-problems which can be distributed amongst the processors. Dong et al. (2003) used a block-diagonal approximation of the kernel matrix to derive independent optimization problems. The resulting SVMs were used to filter out samples that were likely not to be support vectors. A SVM was then trained on the remaining samples, using the standard serial algorithm. Collobert et al. (2002) proposed a mixture of multiple SVMs where single SVMs are trained on subsets of the training set and a neural network is used to assign samples to different subsets.

Another approach is to use a variation of the standard SVM algorithm that is better suited to a parallel architecture. Tveit and Engum (2003) developed an exact parallel implementation of the Proximal SVM, which modifies the standard SVM formulation to remove the single constraint in the dual and give an unconstrained QP. It is not clear how applicable this formulation is to real-world data sets.

There have only been a few parallel methods in the literature which train a standard SVM on the whole of the data set. We briefly survey the methods of Zanghirati and Zanni (2003), Graf et al. (2005) and Chang et al. (2007).

In an approach similar to `SVMlight`, the algorithm of Zanghirati and Zanni (2003) decomposes the SVM training problem into a sequence of smaller, though still dense, QP sub-problems. Zanghirati and Zanni imple-

ment the inner solver using a technique called variable projection method, which is able to work efficiently on relatively large dense inner problems, and is suitable for implementing in parallel. The performance of the inner QP solver was improved in Zanni et al. (2006). Scalar performance was competitive with `SVMlight`, and parallel efficiency was reasonable.

In the cascade algorithm introduced by Graf et al. (2005), the SVMs are layered. The support vectors given by the SVMs of one layer are combined to form the training sets of the next layer. The support vectors of the final layer are re-inserted into the training sets of the first layer at the next iteration, until the global KKT conditions are met. The authors show that this feedback loop corresponds to standard SVM training.

Another family of approaches to QP optimization are based on Interior Point Method (IPM) technology, which works by delaying the split between active and inactive variables for as long as possible. IPMs generally work well on large-scale problems. A straight-forward implementation of the standard SVM dual formulation would have complexity $\mathcal{O}(n^3)$, and be unusable for anything but the smallest problems. Chang et al. (2007) use IPM technology for the optimizer, and avoid the problem of inverting the dense Hessian matrix by generating a low-rank approximation of the kernel matrix using partial Cholesky decomposition with pivoting. The dense Hessian matrix can then be efficiently inverted implicitly using the low-rank approximation and the Sherman-Morrison-Woodbury (SMW) formula. Moreover, a large part of the calculations at each iteration can be distributed amongst the processors effectively. The SMW formula has been widely used in interior point methods; however, sometimes it runs into numerical difficulties. Goldfarb and Scheinberg (2005) constructed data sets where an SMW-based algorithm required many more iterations to terminate, and in some cases stalled before achieving an accurate solution. They also showed that this situation arises in real-world data sets.

This paper has **two** essential contributions:

- For non-linear SVMs, the data is preprocessed to give a partial Cholesky decomposition. We propose to use the approximation $LL^T + D$ rather than the standard LL^T, allowing outliers to be more efficiently approximated.

- We propose a parallel SVM algorithm that trains the SVM using the full data set, using an interior point method to give efficient optimization, and Cholesky decomposition to give good numerical stability.

For the partial Cholesky decomposition of non-linear kernel matrices,

error bounds resulting from the approximation are analyzed. From our observation that choosing outliers to form columns of L does not reduce the error bounds substantially, we construct a greedy heuristic for choosing pivots. We provide evidence that approximations using explicit diagonal terms ($LL^T + D$) require significantly fewer vectors to be included in matrix L than approximations which do not add diagonal terms (LL^T). Since the complexity of IPM SVMs depends on the square of the number of columns included in L (but does not depend on the number of nonzero elements included in the D part), we observe spectacular improvements resulting from the use of $LL^T + D$ approximations. In addition, we describe an algorithm for performing this decomposition, without requiring explicit storage of the kernel matrix.

Our parallel SVM algorithm distributes the data evenly amongst the processors. We use an interior point method to give efficient optimization of the QP that is the core of SVM training, but unlike previous approaches we use Cholesky decomposition to give good numerical stability and efficient memory caching. Our approach directly tackles the most computationally expensive part of the optimization, namely the inversion of the dense Hessian matrix, through providing an efficient implicit inverse representation of an approximation to the Hessian. By exploiting the structure of the problem, we show how this can be parallelized with excellent parallel efficiency. The resulting implementation is significantly faster at SVM training than can be achieved using state-of-the-art active set methods implemented serially.

The structure of the rest of this paper is as follows. Section 2 provides a short description of support vector machines. Section 3 gives an outline of interior point method for optimizing quadratic programs. Then in Section 4 we describe our approach to training linear and non-linear SVMs efficiently. Our algorithm for choosing pivots for the partial Cholesky decomposition is described in Section 5. The parallel algorithm that exploits the structure of the QP is given in Section 6, along with numerical performance results.

We now briefly describe the notation used in this paper. x_i is the attribute vector for the i^{th} data point, and it consists of the observation values directly. There are n observations in the training set, and k attributes in each vector x_i. X is the $n \times k$ matrix whose rows are the attribute row vectors x_i^T associated with each point. The classification label for each data point is denoted by $y_i \in \{-1, 1\}$. The variables $w \in \mathbb{R}^k$ 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 Support vector machines

In this section we briefly outline the formulations for Support Vector Machines used for classification problems. In particular, we draw attention to the relationship between the primal weight variables $w \in \mathbb{R}^k$ and the dual variables $z \in \mathbb{R}^n$, as we will use it later to develop new formulations for each of the problems described here.

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) (Vapnik, 1998, 1999). A fuller description is also given in Cristianini and Shawe-Taylor (2000).

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:

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

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

Due to the convex nature of the problem, a Lagrangian function associ-

ated with (1) can be formulated,

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

where $z \in \mathbb{R}^n$ is the vector of Lagrange multiplers associated with the inequality constraint, and $\mu \in \mathbb{R}^n$ is the vector of Lagrange multipliers associated with the non-negativity constraint on ξ. The solution to (1) 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:

$$\begin{aligned} w &= (YX)^T z \\ y^T z &= 0 \\ 0 &\leq z \leq \tau e. \end{aligned} \tag{2}$$

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

$$\begin{aligned} \min_z \quad & \frac{1}{2} z^T Y X X^T Y z - e^T z \\ \text{s.t.} \quad & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{3}$$

Non-linear kernels are a powerful extension to the Support Vector Machine technique. Attribute vectors x are transformed into some feature space through a non-linear mapping $x \to \Phi(x)$. This allows SVMs to handle data sets that are not linearly separable, but which can be separated by a polynomial curve or by clustering. One of the main advantages of the dual formulation is that the mapping can be represented by a kernel matrix K, where each element is given by $K_{ij} = \Phi(x_i)^T \Phi(x_j)$, resulting in the QP

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

Kernel functions allow the matrix K to be calculated without knowing $\Phi(x)$ explicitly. The original attribute vectors appear only in terms of inner products. As an example, a commonly used kernel function is the radial basis kernel (RBF)

$$K_{ij} = e^{-\gamma \|x_i - x_j\|^2}.$$

3 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; for more details, see Wright (1997).

We are interested in solving the general convex quadratic problem

$$\begin{aligned} \min_{z} \quad & \frac{1}{2} z^T Q z - e^T z \\ \text{s.t.} \quad & Az = b \\ & 0 \leq z \leq u, \end{aligned} \tag{5}$$

where u is a vector of upper bounds, and the constraint matrix A is assumed to have full row rank. Dual feasibility requires that $A^T\lambda + s - v - Qz = -e$, where λ is the Lagrange multiplier associated with the linear constraint $Az = b$ and $s, v > 0$ are the Lagrange multipliers associated with the lower and upper bounds of z respectively. An interior point method progresses towards satisfying the KKT conditions over a series of iterations, by reducing primal and dual infeasibilities 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, 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.

The Newton system to be solved at each iteration can be transformed into the *augmented system equations*:

$$\begin{bmatrix} -(Q + \Theta^{-1}) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_c \\ r_b \end{bmatrix}, \tag{6}$$

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. If the block $(Q + \Theta^{-1})$ is diagonal, an efficient method to solve such a system is to form the Schur complement $C = A\,(Q + \Theta^{-1})^{-1}\,A^T$, solve the smaller system $C\Delta\lambda = \hat{r}_b$ for $\Delta\lambda$, and back-substitute into (6) to calculate Δz. Unfortunately in the case of SVM training, the kernel matrix, and therefore Q, is a completely dense matrix.

4 Efficient training of SVMs using IPM

In this section we describe a formulation for training a SVM that allows the efficient use of IPM technology to solve the QP. We use Cholesky decomposition to give good numerical stability, applied to all features at once resulting in a more efficient implementation in terms of memory caching. For linear SVMs, the feature matrix is used directly. Our approach can be applied to the case of non-linear SVMs once the data has been preprocessed using partial Cholesky decomposition with pivoting.

4.1 Linear SVMs

Using the form $Q = (YX)(YX)^T$ enabled by the linear kernel, we can rewrite the quadratic objective in terms of w, and ensure the relationship (2) between w and z to hold at optimality by introducing it into the constraints. Consequently, we can state the classification problem (3) as the following separable QP:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2} w^T w - e^T z \\ \text{s.t.} \quad & w - (YX)^T z = 0 \\ & y^T z = 0 \\ & 0 \leq z \leq \tau e. \end{aligned} \tag{7}$$

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

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

while the constraint matrix is in the form:

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

Determining the Newton step requires calculating the matrix product:

$$\begin{aligned} C &\equiv A(Q + \Theta^{-1})^{-1} A^T \\ &= \begin{bmatrix} (I_k + \Theta_w^{-1})^{-1} + X^T Y \Theta_z Y X & -X^T Y \Theta_z y \\ -y^t \Theta_z Y X & y^T \Theta_z y \end{bmatrix} \in \mathbb{R}^{(k+1)\times(k+1)}. \end{aligned} \tag{8}$$

We need to solve $A(Q + \Theta^{-1})^{-1} A^T \Delta\lambda = r$ for $\Delta\lambda$. Building the matrix (8) is the most expensive operation, of order $\mathcal{O}(n(k+1)^2)$, while inverting the resulting matrix is of order $\mathcal{O}((k+1)^3)$.

4.2 Non-linear SVMs

The formulation in the above section can be extended to the non-linear SVM training problem (4). The matrix resulting from a non-linear kernel is normally dense, which results in a dense QP. It has been noted by several researchers (see Fine and Scheinberg, 2002) that it is possible to make a good low-rank approximation of the kernel matrix. Fine and Scheinberg (2002) use the approximation $K \approx LL^T$ based on partial Cholesky decomposition with pivoting. In Woodsend and Gondzio (2007) we showed that the approximation $K \approx LL^T + \mathrm{diag}(d)$ can be determined at no extra computational expense. By applying a similar reformulation we can derive the QP:

$$\begin{aligned} \min_{w,z} \quad & \frac{1}{2}(w^T w + z^T D z) - e^T z \\ \text{s.t.} \quad & w - (YL)^T z = 0 \\ & y^T z = 0 \\ & 0 \le z \le \tau e. \end{aligned} \tag{9}$$

Note that we can still exploit the separability of the objective as the Hessian is again diagonal, so the computational complexity using the $LL^T + D$ approximationg with L of rank r is $\mathcal{O}(n(r+1)^2 + nkr + (r+1)^3)$.

5 Choosing pivots for partial Cholesky decomposition of the kernel matrix

To form the approximation $K \approx LL^T$ using partial Cholesky decomposition, Fine and Scheinberg (2002) chose pivots by applying the greedy heuristic algorithm of selecting the largest diagonal element each time, and this approach has been used subsequently (Chang et al., 2007; Woodsend and Gondzio, 2007).

In this section we show that this algorithm is not the best for approximating clustered data using a low-rank matrix, particularly if the residual diagonal forms part of the approximation.

5.1 Approximating clusters and outliers

Consider a data-set that consists of two clusters A and B, both with a large number (n_A and n_B respectively, and $n_A \approx n_B$) of very closely situated points, and n_C outlying points C which are far from either cluster (see Figure

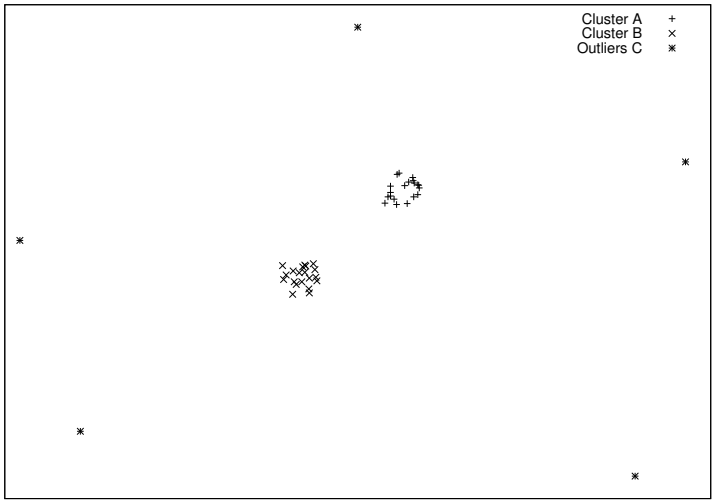

Figure 1: Data set consisting of two clusters plus outliers.

1). If the Radial Basis Kernel is used for this data set, it will generate a kernel matrix with a large block of elements equal to or very close to 1 corresponding to cluster A, a similar block corresponding to cluster B, and a diagonal block very close to the identity matrix corresponding to points C. Using the notation $e_{n_A} = (1, 1, \ldots, 1)^T \in \mathbb{R}^{n_A}$ and $e_{n_B} = (1, 1, \ldots, 1)^T \in \mathbb{R}^{n_B}$, the kernel matrix can be closely approximated by

$$K^{(0)} = \begin{bmatrix} e_{n_A} e_{n_A}^T & \alpha e_{n_A} e_{n_B}^T & \\ \alpha e_{n_B} e_{n_A}^T & e_{n_B} e_{n_B}^T & \\ & & I_{n_C} \end{bmatrix} \begin{matrix} (A) \\ (B) \\ (C) \end{matrix}$$

if A and B are reasonably close, while the points of C are far from both the A and B clusters. In other words, the large blocks corresponding to clusters A and B can be very well approximated by rank-one matrices. Note that all the pivots have the same value at this stage.

Next we show by example that such a kernel matrix, resulting from natural clusters in the data set, cannot be well approximated by a partial Cholesky decomposition LL^T with pivoting based on the largest diagonal element unless the rank of L is at least $n_c + 2$. Suppose a small rank, say $r = 2$ is used. Proposition 5.1 gives the error in the approximation of $K^{(0)}$ using such an approximation, when rank $r = 2$. Alternatively, Proposition 5.2 shows that an approximation using $LL^T + D$ with L build of merely two columns, where pivots are chosen on some measure based on clustering,

has minimal error. Let the notation $\mathcal{O}(\epsilon)$ indicate a number very close to zero, and blocks in the form $\mathcal{O}(\epsilon)_{n_A \times n_A}$ indicate sub-matrices where diagonal elements are zero and non-diagonal elements are very close to zero.

Proposition 5.1. *If a low-rank approximation LL^T of matrix $K^{(0)}$ is constructed, with the rank $r = 2$ and at each iteration the largest pivot is chosen, the lowest Frobenius norm of the residual matrix $\|K - LL^T\|_F^2 \approx (1-\alpha^2)n_B^2 + n_C - 1$.*

Proof. If we assume that the first pivot chosen at random is taken from cluster A (if $n_A \approx n_B$ then clusters A and B are interchangable), the first approximation of the kernel matrix is

$$K^{(0)} = \begin{bmatrix} e_{n_A} \\ \alpha e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} e_{n_A}^T & \alpha e_{n_B}^T & 0_{n_C}^T \end{bmatrix} + K^{(1)},$$

$$\text{where} \qquad K^{(1)} = \begin{bmatrix} \mathcal{O}(\epsilon)_{n_A \times n_A} & \mathcal{O}(\epsilon)_{n_A \times n_B} & \\ \mathcal{O}(\epsilon)_{n_B \times n_A} & (1-\alpha^2) e_{n_B} e_{n_B}^T & \\ & & I_{n_C} \end{bmatrix}.$$

At the next iteration, the largest pivot will be found among diagonal elements corresponding to points in C, giving

$$K^{(0)} = \begin{bmatrix} e_{n_A} \\ \alpha e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} e_{n_A}^T & \alpha e_{n_B}^T & 0_{n_C}^T \end{bmatrix}$$
$$+ \begin{bmatrix} 0_{n_A+n_B} \\ 1 \\ 0_{n_C-1} \end{bmatrix} \begin{bmatrix} 0_{n_A+n_B}^T & 1 & 0_{n_C-1}^T \end{bmatrix} + K^{(2)}$$

$$\text{where} \quad K^{(2)} = \begin{bmatrix} \mathcal{O}(\epsilon)_{n_A \times n_A} & \mathcal{O}(\epsilon)_{n_A \times n_B} & & \\ \mathcal{O}(\epsilon)_{n_B \times n_A} & (1-\alpha^2) e_{n_B} e_{n_B}^T & & \\ & & 0 & \\ & & & I_{n_C-1} \end{bmatrix}.$$

The Frobenius norm of this residual matrix $\|K - LL^T\|_F^2 \approx (1-\alpha^2)n_B^2 + n_C - 1$. □

By using a similar argument we can prove that unless all pivots corresponding to points in C are eliminated, no pivot from the cluster B and be chosen. Hence we need a rank of at least $n_C + 2$ to reduce the error of the approximation to an $\mathcal{O}(\epsilon)$ level.

Proposition 5.2. *If an approximation $LL^T + D$ of matrix $K^{(0)}$ is constructed, where a point in cluster A and a point in cluster B are chosen to form the two columns of L, and D is the diagonal of the residual matrix, then the Frobenius norm of the residual matrix $\|K - (LL^T + D)\|_F = \|\mathcal{O}(\epsilon)_{(n_A+n_B)\times(n_A+n_B)}\|_F$.*

Proof. If, instead of choosing the pivots based on the largest elements in the diagonal, we choose a pivot from cluster A and a pivot from cluster B, then we get

$$K^{(0)} = \begin{bmatrix} e_{n_A} \\ \alpha e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} e_{n_A}^T & \alpha e_{n_B}^T & 0_{n_C}^T \end{bmatrix} + \begin{bmatrix} 0_{n_A} \\ \sqrt{1-\alpha^2} e_{n_B} \\ 0_{n_C} \end{bmatrix} \begin{bmatrix} 0_{n_A}^T & \sqrt{1-\alpha^2} e_{n_B}^T & 0_{n_C}^T \end{bmatrix} + K^{(2)}$$

$$\text{where} \quad K^{(2)} = \begin{bmatrix} \mathcal{O}(\epsilon)_{(n_A+n_B)\times(n_A+n_B)} & \\ & I_{n_C} \end{bmatrix},$$

with a Frobenius norm $\|K - LL^T\|_F^2 \approx n_C$. If cluster B is large and C small, then this is obviously preferable. Additionally, if we include a residual diagonal into our approximation $(LL^T + D)$, the norm of the residual error matrix can be reduced further to $\|K - (LL^T + D)\|_F = \|\mathcal{O}(\epsilon)_{(n_A+n_B)\times(n_A+n_B)}\|_F$. □

We can see from this example that the algorithm that chooses the largest pivot will have to choose all the outliers C before it will select any point from cluster B. The approximation will need to be of higher rank than the number of outliers. We can remedy this by using the form $K \approx LL^T + D$ and packing all elements corresponding to outliers C into matrix D, without increasing the rank of L.

5.2 Bounding the error in the approximation

If we approximate the original kernel matrix using the matrix $LL^T + D$, and use it to form the Hessian of the SVM training QP (9), how close is the perturbed problem to the original one? One method of judging is to measure the error in the objective value. To do this, we consider the original nonlinear SVM dual formulation (4), with the kernel matrix forming part of the Hessian, and the approximate QP in which the new (approximate) Hessian $\tilde{K}$ is used. Let $f(z) = \frac{1}{2} z^T Y K Y z^T - e^T z$ and $\tilde{f}(z) = \frac{1}{2} z^T Y \tilde{K} Y z^T - e^T z$

denote the objective functions in the original problem (4) and in its approximation respectively. Bounds for the error in the approximated objective function are given in Theorem 5.3.

Theorem 5.3. *Let z^* be the optimal solution of (4) and $\tilde{z}^*$ be the optimal solution of the approximate QP. Let $K = \tilde{K} + \Delta K$. Then the error in the objective value $|f(z^*) - \tilde{f}(\tilde{z}^*)|$ is bounded by*

$$\frac{1}{2} z^{*T} Y \Delta K Y z^* \leq f(z^*) - \tilde{f}(\tilde{z}^*) \leq \frac{1}{2} \tilde{z}^{*T} Y \Delta K Y \tilde{z}^*.$$

Proof. First we observe that the two QP problems differ only in the objective functions; the constraints are identical. Hence any feasible solution of (4) is also a feasible solution of the approximate problem. From the definitions of $f(z)$ and $\tilde{f}(z)$, for any point z,

$$\begin{aligned} f(z) - \tilde{f}(z) &= \frac{1}{2} z^T Y K Y z - \frac{1}{2} z^T Y \tilde{K} Y z \\ &= \frac{1}{2} z^T Y \Delta K Y z. \end{aligned} \tag{10}$$

For the upper bound, consider the point z^*. It is the minimizer of $f(\cdot)$, therefore $f(z^*) \leq f(\tilde{z}^*)$. Applying (10) to the point $\tilde{z}^*$ and substituting in this inequality gives

$$f(z^*) - \tilde{f}(\tilde{z}^*) \leq \frac{1}{2} \tilde{z}^{*T} Y \Delta K Y \tilde{z}^*.$$

Similarly, the point $\tilde{z}^*$ minimizes $\tilde{f}(\cdot)$, so $\tilde{f}(\tilde{z}^*) \leq \tilde{f}(z^*)$. Substituting this relationship for $f(z^*)$ into (10) at z^* gives

$$f(z^*) - \tilde{f}(\tilde{z}^*) \geq \frac{1}{2} z^{*T} Y \Delta K Y z^*. \qquad \square$$

A good approximation will keep these bounds as tight as possible. Below we attempt to quantify this. Let us define the matrix norm $\|\Delta K\|_e := \sum_i \sum_j |\Delta K_{ij}|$, the sum of all absolute values in ΔK.

Corollary 5.4. *If the matrix K in QP (4) is approximated by the matrix $\tilde{K}$, where $K = \tilde{K} + \Delta K$, the error in the objective due to the approximation is bounded by*

$$|f(z^*) - \tilde{f}(\tilde{z}^*)| \leq \frac{1}{2} \tau^2 \|\Delta K\|_e.$$

Proof. In (4) z is bounded by $0 \leq z \leq \tau e$, and only the support vectors themselves will have non-zero values, but obviously we can only know the values of z^* after the approximation is made. Following Theorem 5.3, a worst case for the bounds of the error, $|f(z^*) - \tilde{f}(\tilde{z}^*)|$, can be constructed by taking all elements of z to their bounds:

$$\begin{aligned} |f(z^*) - \tilde{f}(\tilde{z}^*)| &\leq \frac{1}{2} \sum_i \sum_j \max(\tau^2 y_i y_j \Delta K_{ij}, 0) \\ &\leq \frac{1}{2} \tau^2 \sum_i \sum_j |y_i y_j \Delta K_{ij}| \\ &= \frac{1}{2} \tau^2 \sum_i \sum_j |\Delta K_{ij}| \end{aligned}$$

as $y_i, y_j \in \{-1, +1\}$.

Using the definition of $\|\Delta K\|_e$, the maximum error in the objective is then

$$|f(z^*) - \tilde{f}(\tilde{z}^*)| \leq \frac{1}{2} \tau^2 \|\Delta K\|_e,$$

and clearly minimizing $\|\Delta K\|_e$ will reduce the error. □

5.3 A cost measure for selecting pivots

In section 5.1 we observed that the diagonal elements of K corresponding to outlying data points will remain large during partial Cholesky decomposition, yet other elements in the columns will be small. An algorithm that selects columns by choosing the largest diagonal element will construct columns of L based on such points, yet they can be adequately represented by a diagonal matrix D. In that example, the decision whether a pivot should contribute to a diagonal term of D or a column of L was straightforward. In real-life examples this decision is less obvious. Moreover, we wish to keep the rank of L as small as possible, hence we would like to encourage an early choice of essential columns which indeed should contribute to L. We start our discussion from an analysis of a simplified example in which two attractive pivot candidates are available.

Let the kernel submatrix at an iteration be

$$K = \begin{bmatrix} u_1 & 0 & \bar{u}^T \\ 0 & v_1 & \bar{v}^T \\ \bar{u} & \bar{v} & \bar{K} \end{bmatrix}. \tag{11}$$

The columns containing u and v are the candidates for pivoting. For simplicity, we consider them after being permuted to the top of the matrix. The elements u_1 and v_1 are pivot elements on the diagonal, while $\bar{u}$ and $\bar{v}$ are vectors containing the rest of the column. $\bar{K}$ contains the rest of the matrix. To describe the effect on the residual matrix after a candidate is chosen, we make the following assumption.

Assumption 5.5. *The kernel matrix $K \geq 0$, and for all pivot column candidates $K - \frac{1}{u_1} u u^T \geq 0$.*

Assumption 5.5 corresponds to assuming that all elements of the Schur complement will be reduced in absolute value as the result of the pivoting. First let us observe that, by construction, kernel matrices are positive semi-definite. For some types of kernel, it is possible to say more: for instance $K \geq 0$ for a polynomial kernel with an even power, and $K > 0$ for the commonly used RBF kernel. Unfortunately for later iterations this is only an approximation: $K \geq 0$ cannot be guaranteed even for an initial kernel matrix using the radial basis kernel. However, we have noted empirically that the vast majority of elements of the Schur complement remain positive during at least the early iterations of Cholesky decomposition. We exploit these observations for the purposes of developing a heuristic, by taking that Assumption 5.5 will hold at least for the early iterations where this heuristic will be used.

Remark 5.6. *If Assumption 5.5 holds, the residual matrix after choosing column u will have the norm*

$$\|K - ll^T\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2.$$

Proof. If u is chosen for the pivot, the corresponding column of the Cholesky decomposition l will be $\begin{bmatrix} l_1 \\ 0 \\ \bar{l} \end{bmatrix}$, where $l_1 = \sqrt{u_1}$ and $\bar{l} = \frac{1}{l_1}\bar{u}$. The residual matrix $K - ll^T$ resulting when column u is chosen will be

$$K - ll^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & v_1 & \bar{v}^T \\ 0 & \bar{v} & \bar{K} - \bar{l}\bar{l}^T \end{bmatrix}.$$

The entry-wise norm defined before Corollary 5.4 will be $\|K - ll^T\|_e = |v_1| + 2\|\bar{v}\|_1 + \|\bar{K} - \bar{l}\bar{l}^T\|_e$.

Under the (restrictive) Assumption 5.5,

$$\|\bar{K} - \overline{l}\overline{l}^T\|_e = \|\bar{K}\|_e - \|\overline{l}\overline{l}^T\|_e.$$

Furthermore, using the relationships $\bar{l} = \frac{1}{l_1}\bar{u}$ and $l_1 = \sqrt{u_1}$, the norm of the sub-matrix

$$\|\bar{K} - \overline{l}\overline{l}^T\|_e = \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\bar{u}^T\|_e.$$

Using the following: $\|\bar{u}\bar{u}^T\|_e = \sum_i \sum_j (|\bar{u}_i||\bar{u}_j|) = \sum_i |\bar{u}_i| \sum_j |\bar{u}_j| = \|\bar{u}\|_1^2$, we conclude

$$\|\bar{K} - \overline{l}\overline{l}^T\|_e = \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2, \quad \text{and}$$
$$\|K - ll^T\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2. \qquad \square$$

There will be an analogous result if column v was used. It is now possible to say which column should be chosen as the pivot: we will prefer column u to v if it gives a smaller error in the residual matrix when compared to the error resulting from choosing column v; in other words, if $\|K - (ll^T)^{(u)}\|_e < \| K - (ll^T)^{(v)}\|_e$.

Remark 5.7. *If Assumption 5.5 holds, an approximation $LL^T \approx K$ with minimum residual is constructed by choosing column u as the pivot in preference to column v if*

$$\frac{1}{v_1}\|v\|_1^2 < \frac{1}{u_1}\|u\|_1^2.$$

Proof. Using Remark 5.6 for $\|K - ll^T\|_e$, we get

$$\|K - (ll^T)^{(u)}\|_e < \| K - (ll^T)^{(v)}\|$$
$$v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2 < u_1 + 2\|\bar{u}\|_1 + \|\bar{K}\|_e - \frac{1}{v_1}\|\bar{v}\|_1^2$$
$$v_1 + 2\|\bar{v}\|_1 + \frac{1}{v_1}\|\bar{v}\|_1^2 < u_1 + 2\|\bar{u}\|_1 + \frac{1}{u_1}\|\bar{u}\|_1^2$$
$$\frac{1}{v_1}\|v\|_1^2 < \frac{1}{u_1}\|u\|_1^2. \qquad \square$$

We can see from this inequality that whether column u is the best choice of pivot depends in a non-linear way on both the pivot value u_1 and the norm of the column $\|u\|_1$. It is not always best to choose the largest pivot.

The measure changes if we use the residual diagonal to improve our approximation, as shown in Remark 5.8. Let the matrix K be approximated

using $LL^T + \bar{D} \approx K$, where $\bar{D}$ is the diagonal of $\bar{K}$. The residual error matrix is therefore redefined as $K - (LL^T + \bar{D})$. We consider again matrix (11) and the pivot selection that minimizes the error in $\Delta K = K - (LL^T + \bar{D})$.

Proposition 5.8. *The pivot corresponding to column u is preferable to that of column v if*

$$2\|\bar{v}\|_1 + \frac{1}{v_1}\|\bar{v}\|_1^2 < 2\|\bar{u}\|_1 + \frac{1}{u_1}\|\bar{u}\|_1^2.$$

Proof. Using the redefined residual error matrix, then when column u is chosen as the pivot

$$\|\Delta_u\|_e = v_1 + 2\|\bar{v}\|_1 + \|\bar{K}\|_e - \frac{1}{u_1}\|\bar{u}\|_1^2 - v_1 - \|\bar{D}\|_e.$$

Column u is the better pivot if

$$2\|\bar{v}\|_1 + \frac{1}{v_1}\|\bar{v}\|_1^2 < 2\|\bar{u}\|_1 + \frac{1}{u_1}\|\bar{u}\|_1^2. \qquad \square$$

We can use the measure

$$m_u = 2\|\bar{u}\|_1 + \frac{1}{u_1}\|\bar{u}\|_1^2 \tag{12}$$

as the basis for selecting a pivot column. m_u can be thought of as a cost measure for approximating the column with a diagonal rather than providing an exact representation through a column in L.

5.4 An algorithm for Cholesky decomposition using column norms

We use (12) as the basis of greedy heuristic for selecting columns to form a Cholesky decomposition of the kernel matrix. At each iteration, we choose the column with the highest measure, while rejecting columns with very small pivots that would increase numerical errors during the decomposition.

A simplified version is given as Algorithm 1. In this algorithm, n is the number of samples, r the maximum allowed number of columns of the Cholesky factor L, and d is the diagonal of the residual matrix $K - LL^T$. The full set of columns $\mathcal{J}$ of the matrix K are partitioned into four disjoint sets: $\mathcal{L}$ (columns allocated to L), $\mathcal{D}$ (allocated to D), $\mathcal{P}$ (columns postponed from consideration in the current iteration) and $\mathcal{U}$ (columns available for consideration, and not yet allocated to either L or D), where $\mathcal{J} = \mathcal{L} \cup \mathcal{D} \cup \mathcal{P} \cup \mathcal{U}$.

Algorithm 1 A simplified serial algorithm to construct a Cholesky decomposition with partial pivoting, $\tilde{K} = LL^T + \text{diag}(d)$, using the cost measure m_u to minimize the residual $\bar{K} - diag(d)$.

$\mathcal{J} := \{1 \dots n\}$,
$\mathcal{L} := \emptyset, \quad \mathcal{D} := \emptyset, \quad \mathcal{P} := \emptyset, \quad \mathcal{U} := \mathcal{J}$
$d_j := K_{jj} \quad \forall j \in \mathcal{J}$ // Initialise the diagonal
// Calculate a maximum of r columns
for $i = 1 : r$ **do**
  Choose $j^* : d_{j^*} = \max_{j \in \mathcal{U}} d_j$
  Compute column j^* of the Schur complement:
  $S_{kj^*} := K_{kj^*} - \sum_{l=1}^{i} (L_{kl} \cdot L_{j^*l}) \quad \forall k = i+1, \dots, n$
  Compute the "cost" of approximating this column using just the diagonal, and not as a full column of L:
  $\bar{s}_{j^*} := \sum_{l=i+1}^{n} |S_{lj^*}|$
  $m_{j^*} := \frac{1}{d_{j^*}} \bar{s}_{j^*}^2 + 2\bar{s}_{j^*}$
  if $d_{j^*} < \epsilon_{\text{small}}$ or $m_{j^*} < \epsilon_{\mathcal{D}}$ **then**
    $\mathcal{D} := \mathcal{D} \cup \{j^*\}, \quad \mathcal{U} := \mathcal{U} \backslash \{j^*\}$
  else if $m_{j^*} > \epsilon_{\mathcal{L}}$ **then**
    $\mathcal{L} := \mathcal{L} \cup \{j^*\}, \quad \mathcal{U} := \mathcal{U} \backslash \{j^*\}$, form new column i of L using $L_{\cdot i} := S_{\cdot j^*}$, update the diagonal: $d_j := d_j - (L_{ji})^2 \quad \forall j \in \mathcal{U} \cup \mathcal{P}$
  else
    For a column where the allocation is not clear, temporarily remove it from consideration:
    $\mathcal{P} := \mathcal{P} \cup \{j^*\}, \quad \mathcal{U} := \mathcal{U} \backslash \{j^*\}$
  end if
  Return columns j to $\mathcal{U}$ that have been in $\mathcal{P}$ for the required number of iterations:
  $\mathcal{U} := \mathcal{U} \cup \{j\}, \quad \mathcal{P} := \mathcal{P} \backslash \{j\}$
  if $\mathcal{U} = \emptyset$ **then**
    $r := i$
    return
  end if
end for

In addition, we use a number of algorithm parameters as the basis of decisions: $\epsilon_{\mathcal{L}}$ is a threshold value for including a column in $\mathcal{L}$, while $\epsilon_{\mathcal{D}}$ is a similar threshold value for inclusion in $\mathcal{D}$. We set $\epsilon_{\mathcal{D}} \ll \epsilon_{\mathcal{L}}$. If $\epsilon_{\mathcal{D}} \leq m_u \leq \epsilon_{\mathcal{L}}$ then we postpone the decision regarding column u. To aid numerical stability, columns are not chosen for pivoting if their diagonal element is below a minimum acceptable value ϵ_{small}.

To aid readability, we present the algorithm as a serial procedure. Initially all columns are in $\mathcal{U}$. At each iteration, the column with the largest diagonal element is selected, and the corresponding column of the Schur complement is calculated. From this we calculate the cost measure (12). Using this information, a number of options are available: we allocate the column to $\mathcal{D}$ if either the cost measure m_u is low or pivoting using this column would be numerically unstable; if the cost measure is high we allocate the column to $\mathcal{L}$; or we can postpone making a decision on this column by allocating it to $\mathcal{P}$. Columns remain in this set for a number of iterations, before being returned to $\mathcal{U}$. The algorithm terminates when r columns of L have been so constructed, or there are no more columns in $\mathcal{U}$.

In this presentation we assume here that the columns are already permuted into the correct order; in the actual implementation, column permutations are handled implicitly. Kernel functions are expensive to evaluate, so individual elements of the matrix K are calculated as required in steps 3 and 6. Computing column j^* of the Schur complement can be performed in parallel, with each processor calculating elements corresponding to samples held locally; this requires the broadcast of the attribute vector x_{j^*} and the first i rows of L.

In practice it is hard to define a suitable value for $\epsilon_{\mathcal{L}}$, as it depends on the matrix K. We used a cache of best columns, rather than the single column j^* shown in Algorithm 1, and chose the column with the highest cost measure to enter $\mathcal{L}$. When a column was added to $\mathcal{L}$, the Schur complement for candidate columns held in the cache were updated with the newly added column $L_{\cdot i}$, rather than fully recalculated as shown in Step 6 of Algorithm 1.

5.5 Numerical examples

To investigate how this greedy algorithm performed, we created a data set containing 4 clusters of 25 points each, arranged in an XOR pattern. 10 outliers were added. This is shown in Figure 2. The radial basis function was used to create the kernel matrix, with parameter $\gamma = (2\sum_{i=1}^{k} \sigma_i^2)^{-1}$ where σ_i is the standard deviation of attribute i of the original data X.

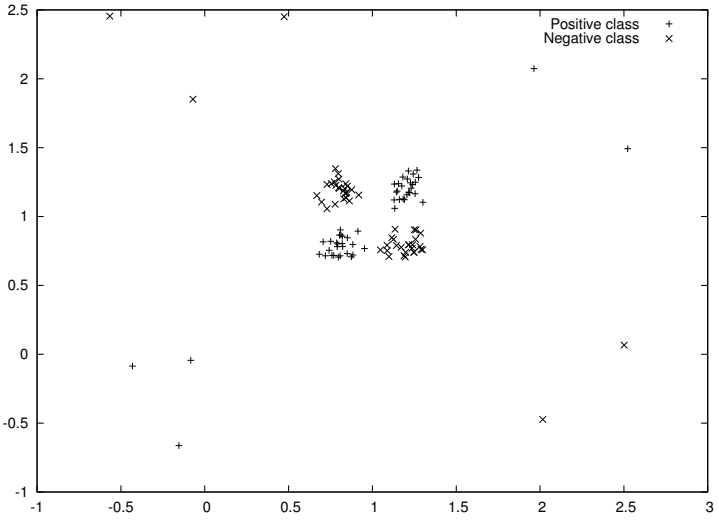

Figure 2: Data set containing 4 clusters plus outliers. No points are misclassified within the clusters.

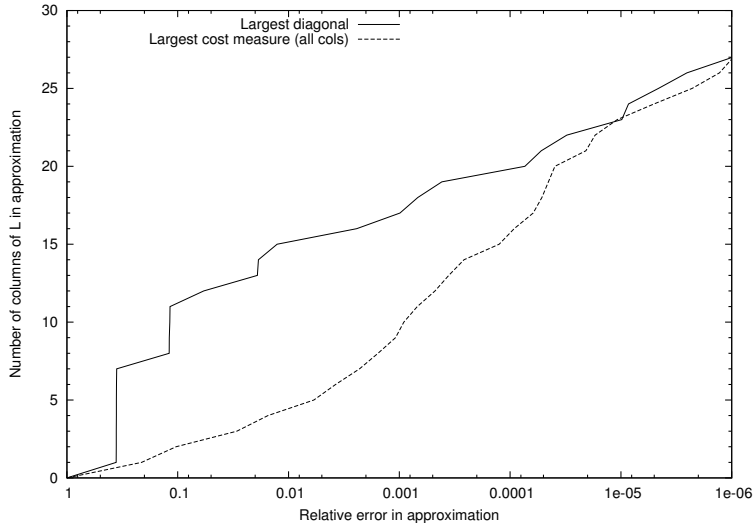

Figure 3: Rank of L in the approximation $LL^T + D$ against the norm of the residual error matrix $\|K - (LL^T + D)\|_e$ achieved, using the data set shown in Figure 2. Choosing each pivot column based on the cost measure m_u (12) is compared against the standard heuristic of choosing the largest diagonal element.

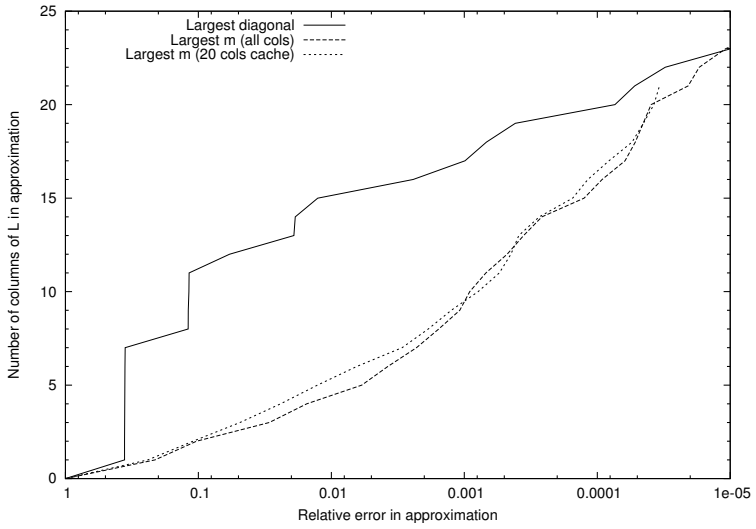

Figure 4: Performance of using a cache of 20 columns gives an approximation that is comparable to knowing the whole matrix.

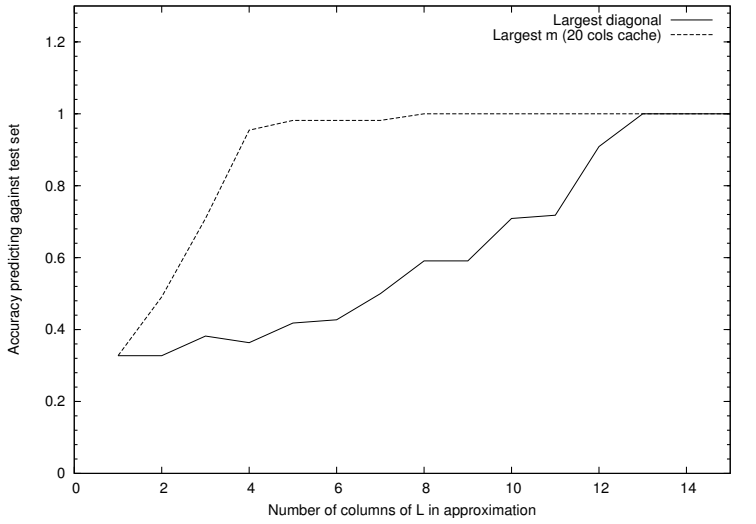

Figure 5: Accuracy of SVM training based on the two approximation techniques in classifying an unseen test data set.

We compared the greedy heuristic for constructing the partial Cholesky decomposition, by choosing pivots based on the largest diagonal element, and on the largest cost measure m_u (12). The performance of the pivot selection methods are shown in Figure 3. The experiments show that the algorithm does not make fast progress in reducing the error in the approximation until most of the outliers have been chosen. In contrast, using the measure m_u gives the best approximation. For example, for an error of 1% of the original kernel matrix, 5 columns are required choosing based on m_u, compared to 16 columns using the largest diagonal element. As the optimization problem associated with SVM training scales to the square of the number of columns, this represents an order of magnitude difference in the time required for the optimization. For L with more than 20 columns, there is little difference between the methods, and for many columns the algorithm using m_u suffers a little from numerical errors.

As described above, it is computationally too expensive to assess all columns. To avoid this, we maintain a cache Schur complements of attractive columns. Figure 4 shows that, in this particular example, the algorithm using a cache of 20 columns performs almost as well in terms of the approximation matrices it generates as an algorithm that knows the full Schur complement matrix.

We also assessed the accuracy of the resulting SVMs in classifying a previously unseen test set with a similar distribution of data points. The results (Figure 5) show that an error of less than approximately 1% of the original kernel matrix is enough to obtain a very high classification accuracy with this particular data set, and the algorithm choosing pivots based on the cost measure achieves this with far fewer columns in L than the one using largest diagonal elements.

We also applied the algorithm to the United States Postal Service (USPS) [1] data set of hand-written digits, and compared the classification accuracy of the cost measure m_u to using the largest diagonal elements as pivots. This data set comprises 7291 training and 2007 test patterns, represented as dense vectors of dimension 257 with entries between 0 and 255. The digits have been classified by hand and labelling is highly accurate. The classification task set was to discriminate the digit 4 from the others. Results are shown in Figure 5.5. For up to 200 columns of L, the cost measure considerably reduces the number of columns required to achieve a certain level of accuracy. After this point, there is little difference in the approximations. It is also apparent from the figure that a very small number of columns (10-20)

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

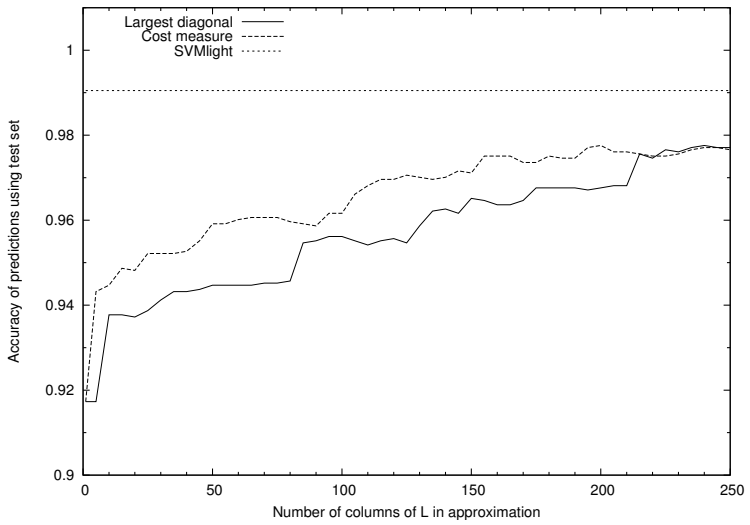

Figure 6: Accuracy of predicting the USPS test set, for approximations $K \approx LL^T + D$. Columns were chosen to enter L either based on largest diagonal element or our cost measure.

gives good results, but subsequently adding columns increases the accuracy only very slowly.

6 Implementing the QP for parallel computation

To apply formulations (7) and (9) to truly large-scale data sets, it is necessary to employ linear algebra operations that exploit the block structure of the formulations (Gondzio and Grothey, 2004).

6.1 Linear algebra operations

We use the augmented system matrix $\Phi = \begin{bmatrix} -Q - \Theta^{-1} & A^T \\ A & 0 \end{bmatrix}$ from (6), where Q, Θ and A were described in Section 3.

For the formulations (7) and (9), this results in Φ having a symmetric

bordered block diagonal structure. We can break Φ into blocks:

$$\Phi = \begin{bmatrix} \Phi_1 & & & & A_1^T \\ & \Phi_2 & & & A_2^T \\ & & \ddots & & \vdots \\ & & & \Phi_p & A_p^T \\ A_1 & A_2 & \dots & A_p & 0 \end{bmatrix},$$

where Φ_i and A_i result from partitioning the data set evenly across the p processors.

A block-based Cholesky decomposition of the matrix $\Phi = LDL^T$ results in the structure:

$$\Phi = \begin{bmatrix} L_1 & & & \\ & \ddots & & \\ & & L_p & \\ L_{A1} & \dots & L_{Ap} & L_C \end{bmatrix} \begin{bmatrix} D_1 & & & \\ & \ddots & & \\ & & D_p & \\ & & & D_C \end{bmatrix} \begin{bmatrix} L_1^T & & & L_{A1}^T \\ & \ddots & & \vdots \\ & & L_p^T & L_{Ap}^T \\ & & & L_C^T \end{bmatrix}$$

We can employ the Schur-complement mechanism for the blocks L_C and D_C. The Cholesky decomposition can be computed by performing the series of computations outlined below:

$$\Phi_i = L_i D_i L_i^T \quad \Rightarrow D_i = \Phi_i, L_i = I \tag{13}$$

$$L_{Ai} = A_i L_i^{-T} D_i^{-1} = A_i \Phi_i^{-1} \tag{14}$$

$$C = -\sum_{i=1}^{p} A_i \Phi_i^{-1} A_i^T \tag{15}$$

$$= L_C D_C L_C^T \tag{16}$$

Matrix C is a dense matrix of relatively small size $(r+1) \times (r+1)$, and the Cholesky decomposition $C = L_C D_C L_C^T$ is performed in the normal way on a single processor.

Once the representation $\Phi = LDL^T$ above is known, we can use it to compute the solution of the system $\Phi \begin{bmatrix} \Delta z \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_c \\ r_b \end{bmatrix}$ (6) through back-substitution. $\Delta z'$, $\Delta\lambda'$ and $\Delta\lambda''$ are vectors used for intermediate calcula-

tions, with the same dimensions as Δz and $\Delta\lambda$.

$$\Delta\lambda'' = L_C^{-1}(r_b - \sum_i^p L_{Ai} r_{ci}) \tag{17}$$

$$\Delta\lambda' = D_C^{-1}\Delta\lambda'' \tag{18}$$

$$\Delta\lambda = L_C^{-T}\Delta\lambda' \tag{19}$$

$$\Delta z_i' = D_i^{-1} r_{ci} \tag{20}$$

$$\Delta z_i = \Delta z_i' - L_{Ai}^T \Delta\lambda \tag{21}$$

For the formation of LDL^T, equations (13) and (14) can be calculated on each processor individually. Outer products (15) can be calculated, and the results gathered onto a single master processor to form C; this requires each processor to transfer approximately $\frac{1}{2}(r+1)^2$ elements. The master processor performs the Cholesky decomposition of C (16). Each processor needs to calculate $L_{Ai}r_{ci}$, which again can be performed without any inter-processor communication, and the results gathered onto the master processor. The master processor then performs the calculations in equations (17), (18) and (19) of the back-substitution. Vector $\Delta\lambda$ is broadcast to all processors for them to calculate equations (20) and (21) locally.

6.2 Performance

We used the SensIT data set[2], collected on types of moving vehicles by using a wireless distributed sensor networks. It consists of 100 dense attributes, combining acoustic and seismic data. There were 78,823 samples in the training set and 19,705 in the test set. The classification task set was to discriminate class 3 from the other two. This is a relatively noisy data set — benchmark accuracy is around 85%. By partitioning the data evenly across the processors, and exploiting the structure as outlined above, we get very good parallel efficiency results, as shown in Figure 6.2 . Training times are competitive: our implementation on 8 processors was 7.5 times faster than LIBSVM running serially ($\tau = 100$).

7 Conclusions

In this paper, we have shown how to develop a parallel implementation of Support Vector Machine training for problems involving nonlinear kernels,

[2]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

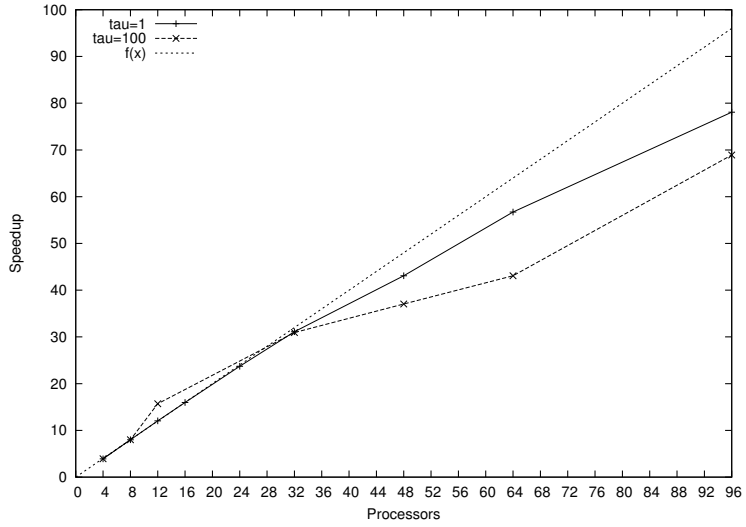

Figure 7: Parallel efficiency, training on the SensIT data set.

and that allowed the entire data set to be used. It consisted of following the steps:

1. Forming a partial Cholesky decomposition of the kernel matrix, taking into account that some columns are adequately approximated using diagonal elements alone.

2. Reformulating the problem to give an implicit inverse of the kernel matrix.

3. Using interior point method to solve the optimization problem in a predictable time, and Cholesky decomposition to give good numerical stability of implicit inverses.

4. Exploiting the block structure of the augmented system matrix, to partition the data and linear algebra computations amongst parallel processors efficiently.

The above steps were implemented in OOPS, and the implementation was applied to solve problems involving very large data sets. Excellent parallel efficiency was observed on such problems.

References

Edward Chang, Kaihua Zhu, Hao Wang, Jongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. Parallelizing support vector machines on distributed computers. In *NIPS*, 2007. To be published.

Ronan Collobert and Samy Bengio. SVMTorch: support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of svms for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

Jian-xiong Dong, Adam Krzyzak, and Ching Suen. A fast parallel optimization for training support vector machine. In P. Perner and A. Rosenfeld, editors, *Proceedings of 3rd International Conference on Machine Learning and Data Mining*, pages 96–105, Leipzig, Germany, 2003. Springer Lecture Notes in Artificial Intelligence.

Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.

Donald Goldfarb and Katya Scheinberg. Solving structured convex quadratic programs by interior point methods with application to support vector machines and portfolio optimization. *Submitted for publication*, 2005.

J. Gondzio and A. Grothey. Exploiting structure in parallel implementation of interior point method for optimization. Technical report, MS-04-004, School of Mathematics, University of Edinburgh, 2004.

Hans Peter Graf, Eric Cosatto, Léon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: the Cascade SVM. In Lawrence Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2005. volume 17.

Thorsten Joachims. Making large-scale support vector machine learning practical. In Bernhard Scholkopf, Christopher. J. C. Burges, and Alexander. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184. MIT Press, 1999.

Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, 1997.

John Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Scholkopf, Christopher. J. C. Burges, and Alexander. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208. MIT Press, 1999.

Amund Tveit and Havard Engum. Parallelization of the incremental proximal support vector machine classifier using a heap-based tree topology. Technical report, IDI, NTNU, Trondheim, 2003.

Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 1999.

Kristian Woodsend and Jacek Gondzio. Exploiting separability in large-scale support vector machine training. Technical Report MS-07-002, School of Mathematics, University of Edinburgh, August 2007. Submitted for publication. Available at *http://www.maths.ed.ac.uk/~gondzio/reports/wgSVM.html*.

Stephen J. Wright. *Primal-dual interior-point methods*. S.I.A.M., 1997.

Gaetano Zanghirati and Luca Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29(4): 535–551, 2003.

Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 2006. ISSN 1533-7928.