Parallel Support Vector Machines

Dominik Brugger

WSI-2006-01

ISSN 0946-3851

Dominik Brugger
Arbeitsbereich Technische Informatik
Sand 13, 72074 Tübingen
`brugger@informatik.uni-tuebingen.de`

Parallel Support Vector Machines

Dominik Brugger
Arbeitsbereich Technische Informatik
Eberhard-Karls Universität Tübingen
Sand 13, 72074 Tübingen
brugger@informatik.uni-tuebingen.de

Abstract

The Support Vector Machine (SVM) is a supervised algorithm for the solution of classification and regression problems. SVMs have gained widespread use in recent years because of successful applications like character recognition and the profound theoretical underpinnings concerning generalization performance. Yet, one of the remaining drawbacks of the SVM algorithm is its high computational demands during the training and testing phase. This article describes how to efficiently parallelize SVM training in order to cut down execution times. The parallelization technique employed is based on a decomposition approach, where the inner quadratic program (QP) is solved using Sequential Minimal Optimization (SMO). Thus all types of SVM formulations can be solved in parallel, including C-SVC and ν-SVC for classification as well as ε-SVR and ν-SVR for regression. Practical results show, that on most problems linear or even superlinear speedups can be attained.

1 Introduction

The underlying idea of supervised algorithms is learning by examples. Thus given a set $x_i \in \mathcal{X}$ of input data and associated labels $y_i \in \mathcal{Y}$ the algorithms learns a mapping $f : \mathcal{X} \mapsto \mathcal{Y}$ using the training data given. If the algorithm generalizes well, then the number of correctly classified inputs on unknown test data will be high in the classification case. Analogously for regression the mean squared error (MSE) will be low.

Support Vector Machines (SVM) are a supervised algorithm first introduced in [21]. One of its advantages over other supervised algorithms, is the possibility to derive bounds concerning the generalization performance on unseen test data after the training phase. Another nice property of SVMs concerns the incorporation of prior knowledge about a learning problem, which can be achieved by a kernel function [14]. The kernel function $k(x_i, x_j)$ computes the dot product between input patterns x_i and x_j that have been mapped into a higher dimensional, or even infinite dimensional feature space using a mapping Φ:

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle \; .$$

Since the kernel function just evaluates the dot product between the mapped patterns the mapping is not carried out explicitly. By substitution of dot products with a kernel function, the SVM constructs a separating hyperplane with maximum margin in the feature space and this separating hyperplane then corresponds to a nonlinear decision surface in input space.

Different kernel functions have been suggested for a wide range of applications, including string kernels for document classification, spike kernels for neuronal signal processing and graph kernels for bioinformatics [14],[16],[10]. Although this clean concept of separation between prior knowledge and learning algorithms has been adopted quickly by many practitioners, the use of complicated kernel functions slows down SVM training considerably. One technique for avoiding this problem is the caching of kernel function evaluations first proposed in [11]. But as kernel functions will get more complex in future this might not be sufficient to speed up SVM training. One possible remedy for this problem is the parallel evaluation and caching of kernel function values as shown in [23, 22].

Another motivation for parallel SVM training are growing dataset sizes in many application areas, which usually range from several hundred thousand to millions of input patterns. Recent studies have shown, that subsampling the dataset in order to cut down training time is not an option in many cases, as it leads to a decrease in classification performance on the test set [17].

According to Moore's law one might argue that many large scale problems which cannot be solved on single processor hardware today might be solvable tomorrow. But this statement is only true in part, if one takes a closer look at hardware developments in the last two years. Most of the acceleration is achieved at the moment by an emerging new architectural concept: the multicore architecture. Yet exploiting the performance of multicore processors requires new threaded or parallel software [2].

1.1 Related Work

Speeding up SVM training has been an issue that was addressed by many authors in the past. But most of the approaches are based on different formulations of the original SVM algorithm or they rely on approximation techniques.

The Core Vector Machine (CVM) can be applied to solve SVM classification and regression problems efficiently on large datasets [17, 18]. It relies on an approximation technique for computing minimum enclosing balls by a concept called coresets which has its originates from the field of computational geometry [3]. In contrast to the original SVM formulation the dual quadratic program (QP) to be solved is simplified, by penalizing margin errors using the L2 loss function and additionally penalizing the hyperplane offset b. This leads to a QP problem with one simple linear constraint and a positivity constraint for the dual variables α which can be solved by the minimum enclosing ball algorithm.

An earlier approach which exploits the same kind of QP problem simplification is the Lagrangian Support Vector Machine (LSVM) of [12]. The LSVM is very efficient for the linear kernel and large problems in low dimensions (< 22), since it uses the Sherman-Morrison-Woodbury identity [8] to invert the kernel matrix.

Parallelizing the original SVM formulation with L1 loss function for margin errors is done by the Cascade SVM [9]. It is based on the idea, that only a small number of the patterns in the training data set will end up as support vectors. Therefore the Cascade SVM splits the dataset into smaller problems and filters out support vectors in a cascade of SVMs which can work in parallel. Although there is a formal proof of convergence for the method, one remaining drawback is the size of the final problem to be solved which is dependent on the number of support vectors. Especially for noisy training data this final problem might be huge.

A different parallel technique for solving SVM problems is the parallelization of the decomposition approach first described in [11]. Recently is has been shown [23], that with appropriate working set selection and inner QP solver, this decomposition approach can gain impressive speedups in practice. However so far this approach has only be used for the training of C-SVC.

In the work described in this article the main focus is on solving the original SVM formulation of [21] in parallel. The adopted approach is the parallel decomposition technique introduced by [23]. This article studies several different inner solvers including SMO, a parallel version of an interior point code (LOQO) and the projected gradient method of [5]. It turns out, that in practice only SMO in combination with the decomposition technique is able to solve all of the SVM formulations including C-SVC, ν-SVC, ε-SVR and ν-SVR reliably.

1.2 Outline

This article is organized as follows: Section 2 gives a brief introduction to the SVM algorithm and subsequently derives the underlying general form of the QP problem to be solved for the different SVM formulations. How these QP problems can be solved in practice is described in section 3. The decomposition method for large scale SVM training as well as details on the working set selection strategy and stopping criteria are described in section 4. Some hints on implementation specific details are given in section 5. Finally section 6 gives performance results on several large scale datasets.

2 Support Vector Machines

In the case of Support Vector Classification (SVC) labeled training data $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, m$ is given and the goal of the SVC algorithm is to learn a function $f : \mathcal{X} \mapsto \mathcal{Y}$, which can be subsequently used for the prediction of class labels on unknown test data. Figure 1 shows a simple binary classification problem, where the two classes are represented by balls and crosses. The SVC algorithm constructs a hyperplane $\langle w, x\rangle + b = 0$ with normal vector w and offset b to separate these two classes. Since there are many possibilities for the location of this hyperplane, SVC searches for a hyperplane with the largest margin, where the margin is defined to be the distance of the closest point to the hyperplane. Intuitively this approach leads to a good solution with respect to the unknown test data, since classes are somewhat well separated. Indeed the choice of a large margin can be directly related to the generalization performance of the classifier in a formal way [14].

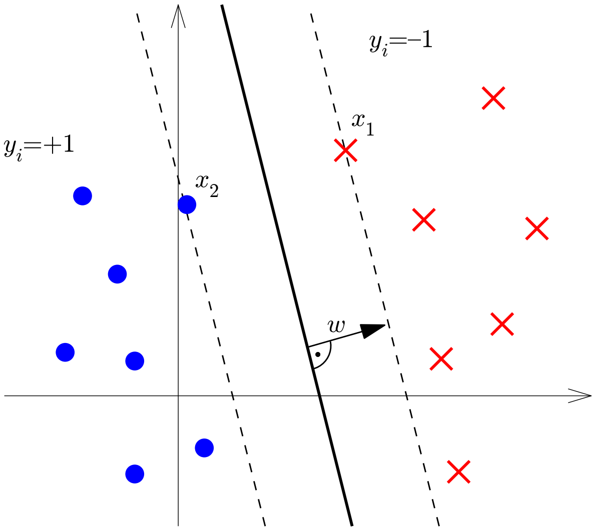

Figure 1: *Toy example of a binary classification problem where the points marked by balls and crosses represent the two classes. The SVC algorithm maximizes the margin between the two classes, which is the distance between the two points x_1 and x_2 closest to the separating hyperplane. This distance can be expressed in terms of the hyperplane normal vector w and is equal to $1/\|w\|$.*

The margin is exactly $1/\|w\|$, if the condition $|\langle w, x_i\rangle + b| = 1$ is satisfied by rescaling w and b appropriately, since:

$$\langle w, x_1\rangle + b = +1,\ \langle w, x_2\rangle + b = -1$$
$$\Rightarrow \langle w, x_1 - x_2\rangle = 2\ \Rightarrow \langle w/\|w\|, x_1 - x_2\rangle = 2/\|w\|.$$

Thus, to construct the optimal hyperplane, the SVC algorithm has to solve the following optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{1}$$

$$\text{subject to} \quad y_i\left(\langle w, x_i\rangle + b\right) \geq 1,\ \forall i = 1, \ldots, m\ . \tag{2}$$

If it is impossible to separate the data by a hyperplane, as often is the case in practice, a so called soft margin hyperplane [14] can be computed by introducing slack variables $\xi_i \geq 0$ and relaxing (2). As a consequence the margin may be violated by some of the input patterns x_i, for which $\xi > 0$. To nevertheless find a good classifier the number of violators is restricted by penalizing the margin error with an L1 loss in the objective function leading to the following optimization problem:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \tag{3}$$

$$\text{subject to} \quad y_i\left(\langle w, x_i\rangle + b\right) \geq 1 - \xi_i,\ \forall i = 1, \ldots, m\ . \tag{4}$$

The parameter C trades off between the number of margin errors and the size of the margin and thus the generalization performance of the classifier. The optimization problem above is usually solved in its dual form which is obtained by incorporating equation (4) into the objective function (3) using the Lagrange function:

$$L(w,b,\xi,\alpha,\beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y_i(\langle x_i, w\rangle + b) - 1) - \sum_{i=1}^{m}\beta_i\xi_i\ .$$

The variables $\alpha_i \geq 0$ and $\beta_i \geq 0$ are the dual variables of the optimization problem and L has to be maximized with respect to α, β and minimized with respect to the primal variables w, b, ξ. The goal therefore is to find a saddle point of L. In other words the derivatives with respect to the primal variables must be zero:

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial w} = w - \sum_{i=1}^{m}\alpha_i y_i x_i = 0 \Leftrightarrow w = \sum_{i=1}^{m} y_i\alpha_i x_i \tag{5}$$

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial b} = \sum_{i=1}^{m}\alpha_i y_i = 0 \tag{6}$$

$$\frac{\partial L(w,b,\xi,\alpha,\beta)}{\partial \xi} = C - \alpha_i - \beta_i = 0 \Leftrightarrow 0 \leq \alpha_i \leq C. \tag{7}$$

In equation (5) it can be seen that the hyperplane normal vector w can be expressed as a linear combination of input patterns x_i. Input patterns for which α_i is greater zero are called support vectors (SVs) and these patterns explain how the algorithm got its name Support Vector Machine. Furthermore these equations allow to eliminate the primal variables in the optimization problem (3) which leads to the dual optimization problem:

$$\begin{aligned} &\max_{\alpha} \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j\rangle + \sum_{i=1}^{m}\alpha_i \\ &\text{subject to} \quad \sum_{i=1}^{m}\alpha_i y_i = 0,\ 0 \leq \alpha_i \leq C,\ \forall i = 1, \ldots, m. \end{aligned} \tag{8}$$

So far the SVC algorithm can only compute a hyperplane to separate the classes and hence the resulting decision function $f(x) = \text{sgn}\left(\langle w, x\rangle + b\right)$ is linear. For patterns which cannot be separated by a linear decision function the already mentioned kernel trick is used to have SVC construct a hyperplane in a feature space, where the mapping to this space is done by a function Φ (Figure 2). Since only dot products between patterns are computed in (8) these dot products can be replaced by kernel function evaluations:

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j)\rangle\ .$$

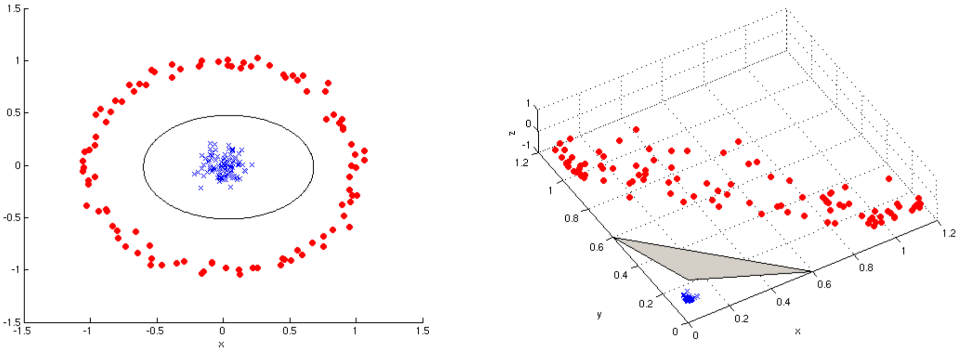

Figure 2: *Binary classification problem in input space (left) and the feature space induced by the mapping $\Phi(x_1 x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ (right). In input space the two classes can only be separated by a nonlinear decision function f, an ellipse in this case, whereas in feature space a plane is sufficient for separation of the classes.*

When dealing with regression rather than classification problems the labels y_i are real values, and the decision function f is used to predict y_i on unknown test data. Support Vector Regression (SVR) therefore computes a function $f(x) = \langle w, x\rangle + b$, where the loss is measured using Vapnik's ε-insensitive loss function (Figure 3):

$$|y - f(x)|_\varepsilon = \max\{0, |y - f(x)| - \varepsilon\}.$$

Thus the goal is to find a function f such that most of the points will lie inside an ε-tube, which is equivalent to minimizing the loss function. This can be expressed by the constraints $f(x_i) - y_i \leq \varepsilon$ and $y_i - f(x_i) \leq \varepsilon$. Again it will not be possible to find such a function for all values of ε making it necessary to relax the constraints analogous to the soft margin classification case. The resulting constrained optimization problem can hence be stated as follows:

$$\begin{aligned} \min_{w,b,\xi,\xi^*} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}(\xi + \xi^*) \\ \text{subject to} \quad & f(x_i) - y_i \leq \varepsilon + \xi_i \\ & y_i - f(x_i) \leq \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0,\ \forall i = 1, \ldots, m\ . \end{aligned} \tag{9}$$

Like in SVC the parameter C is used here to trade off between the capacity of the regression function and the number of violators of the ε-tube. Not surprisingly there is an interesting connection between the margin of SVC and the ε-tube of the SVR algorithm [14]. Finally application of the kernel trick and the introduction of Lagrange multipliers leads to the derivation of the dual optimization problem, which needs to be solved during SVR training:

$$\begin{aligned} \min_{\alpha,\alpha^*} \quad & \frac{1}{2}\sum_{i,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(x_i, x_j) + \varepsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i^*) \\ \text{subject to} \quad & \sum_{i=1}^{m}(\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i, \alpha_i^* \leq C,\ \forall i = 1, \ldots, m\ . \end{aligned} \tag{10}$$

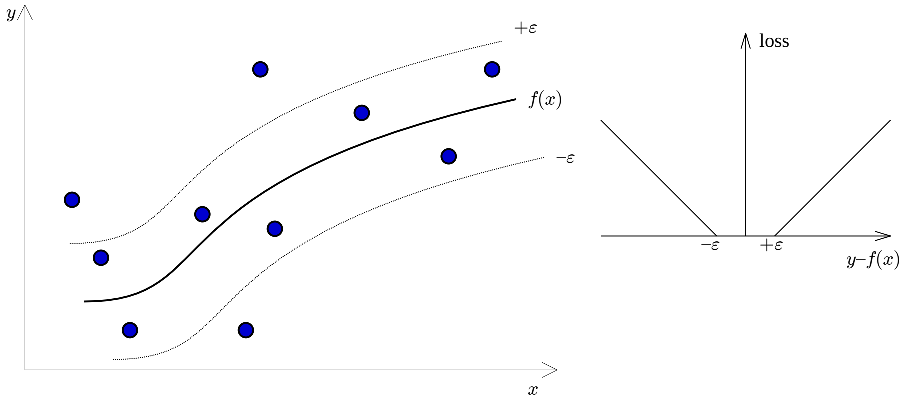

Figure 3: *The use of the ε-insensitive loss function in SVR corresponds to fitting a tube of width ε around the regression function $f(x)$ to be estimated. Points lying inside of this tube do not contribute to the loss as shown in the inset on the right.*

2.1 General formulation for C-SVC and ε-SVR

For C-SVC and ε-SVR the dual optimization problem can be stated in the following general form:

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha + p^T\alpha \\ \text{subject to} \quad & y^T\alpha = \delta, \\ & 0 \le \alpha_i \le C, \ \forall i = 1, \dots, m \ . \end{aligned} \tag{11}$$

With the kernel matrix $Q = y_i y_j k(x_i, x_j)$ for C-SVC it can be clearly seen that the problem (8) can be restated in the general form above. Following [4] the problem (10) for ε-SVR can be reformulated as:

$$\begin{aligned} &\min_{\alpha,\alpha^*} \frac{1}{2}\left[\alpha^T (\alpha^*)^T\right] \begin{bmatrix} Q & -Q \\ -Q & Q \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha^* \end{bmatrix} + \left[\varepsilon e^T + y^T, \varepsilon e^T - y^T\right] \begin{bmatrix} \alpha \\ \alpha^* \end{bmatrix} \\ \text{subject to} \quad & z^T \begin{bmatrix} \alpha \\ \alpha^* \end{bmatrix} = 0, \ 0 \le \alpha_i, \ \alpha_i^* \le C, \ \forall i = 1, \dots, m \ . \end{aligned} \tag{12}$$

where z is a $2m$ by 1 vector with $y_i = 1, i = 1, \dots, m$ and $y_i = -1, i = m+1, \dots, 2m$. The kernel matrix for ε-SVR is $Q = k(x_i, x_j)$.

2.2 General formulation for ν-SVC and ν-SVR

In ν-SVC and ν-SVR a new parameter ν is used to replace the parameter C in C-SVC and ε in ε-SVR [14]. The parameter $\nu \in (0, 1]$ allows the direct control of the number of support vectors and errors. It is an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. For ν-SVC the primal problem to be considered is

$$\begin{aligned} &\min_{w,b,\xi,\rho} \frac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & y_i(\langle w, x_i\rangle + b) \ge \rho - \xi_i \\ & \xi \ge 0, \ \forall i = 1, \dots, m, \ \rho \ge 0 \end{aligned} \tag{13}$$

and the corresponding dual

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha \\ \text{subject to} \quad & e^T\alpha \ge \nu, \\ & y^T\alpha = 0 \\ & 0 \le \alpha_i \le 1/m, \ \forall i = 1, \dots, m \ . \end{aligned} \tag{14}$$

It has been shown [4] that the inequality constraint (14) can be replaced by the equality $e^T\alpha = \nu$. Therefore one can solve the following scaled version of the problem:

$$\begin{aligned} &\min_{\alpha} \ \frac{1}{2}\alpha^T Q\alpha \\ &\text{subject to} \quad e^T\alpha = \nu, \\ &\qquad y^T\alpha = 0 \\ &\qquad 0 \le \alpha_i \le 1, \forall i = 1, \dots, m \ . \end{aligned} \tag{15}$$

The solution to the original problem is obtained by rescaling $\alpha \leftarrow \alpha/\rho$ afterwards.

For ν-SVR the primal problem is

$$\begin{aligned} &\min_{w,b,\xi,\xi^*,\varepsilon} \ \frac{1}{2}\|w\|^2 + C(\nu\varepsilon + \frac{1}{m}\sum_{i=1}^{m}(\xi_i + \xi_i^*)) \\ &\text{subject to} \quad (\langle w, x_i\rangle + b) - y_i \le \varepsilon + \xi_i, \\ &\qquad (y_i - \langle w, x_i\rangle + b) \le \varepsilon + \xi_i, \\ &\qquad \xi_i, \xi_i^* \ge 0, \ \forall i = 1, \dots, m \ \varepsilon \ge 0 \end{aligned} \tag{16}$$

and the dual

$$\begin{aligned} &\min_{\alpha,\alpha^*} \ \frac{1}{2}(\alpha - \alpha^*)^T Q(\alpha - \alpha^*) + y^T(\alpha - \alpha^*) \\ &\text{subject to} \quad e^T(\alpha - \alpha^*) = 0, \ e^T(\alpha + \alpha^*) \le C\nu, \\ &\qquad 0 \le \alpha, \alpha^* \le C/m, \ \forall i = 1, \dots, m \ . \end{aligned} \tag{17}$$

Similar to the classification case the inequality can be replaced by an equality. With rescaling the actual dual problem to be solved is:

$$\begin{aligned} &\min_{\alpha,\alpha^*} \ \frac{1}{2}(\alpha - \alpha^*)^T Q(\alpha - \alpha^*) + y^T(\alpha - \alpha^*) \\ &\text{subject to} \quad e^T(\alpha - \alpha^*) = 0, \ e^T(\alpha + \alpha^*) \le Cm\nu, \\ &\qquad 0 \le \alpha, \alpha^* \le C, \ \forall i = 1, \dots, m. \end{aligned} \tag{18}$$

As a result both ν-SVC and ν-SVR can be stated in the following general form:

$$\begin{aligned} &\min_{\alpha} \ \frac{1}{2}\alpha^T Q\alpha + p^T\alpha \\ &\text{subject to} \quad y^T\alpha = \delta_1, \\ &\qquad e^T\alpha = \delta_2, \\ &\qquad 0 \le \alpha_i \le C, \ \forall i = 1, \dots, m \ . \end{aligned} \tag{19}$$

Before discussing different methods for solving these problems in section 3 it is important to realize that the general problems to be solved for C-SVC/ε-SVR and ν-SVC/ν-SVR just differ in the number of linear constraints. The ν formulation seems to be harder to solve since it has two linear constraints. But the structure of the linear constraints is quite simple and using the fact that $y_i \in \{+1, -1\}$ they can be rewritten as:

$$y^T\alpha = \delta_1 \Leftrightarrow \textstyle\sum_{y_i=+1}\alpha_i = \delta_1 - \sum_{y_i=-1}\alpha_i \tag{20}$$

$$e^T\alpha = \delta_2 \Leftrightarrow \textstyle\sum_{y_i=+1}\alpha_i + \sum_{y_i=-1}\alpha_i = \delta_2 \ . \tag{21}$$

An initial feasible solution for the problem (19) can thus be easily found by setting $0 \le \alpha_i \le C$ such that $\sum_{y_i=+1}\alpha_i = (\delta_1 + \delta_2)/2$ and $\sum_{y_i=-1}\alpha_i = (\delta_2 - \delta_1)/2$ are satisfied. If an optimization algorithm now only changes variables α_i with either $y_i = +1$ or $y_i = -1$, but not both at the same time, it is clear that actively maintaining constraint (20) suffices to ensure that constraint (21) is always satisfied. The reason for this is that constraint (20) ensures that $\sum_{y_i=+1}\alpha_i = 0$ during a change of variables α_i with $y_i = +1$ and vice versa $\sum_{y_i=-1} \alpha_i = 0$ for a change of α_i with $y_i = -1$. Consequently with careful initialization and change of the optimization variables α_i it is possible to reduce the optimization problem (19) with two linear constraints to the general form given in section 2.1. Unfortunately this reduction thus not work well in practice with some of the QP solvers introduced in section 3 because of numerical problems and the restriction imposed on the variable selection method described above.

3 QP Solvers

There are different approaches to solve the general QP problems given in the last two subsections. The interior point algorithm in section 3.1 and gradient projection algorithm in section 3.2 find a numerical solution for the problem, whereas sequential minimal optimization (SMO) in section 3.3 finds a solution by sequentially solving two-variable subproblems, that can be solved analytically themselves.

For all optimization algorithms it is important to decide, when to stop the optimization process. To decide about the optimality of the current solution the so called Karush-Kuhn-Tucker (KKT) conditions are checked [14].

Theorem 1 (KKT conditions)**.** *: Let $f : \mathbb{R}^m \mapsto \mathbb{R}$ and $c_i : \mathbb{R}^m \mapsto \mathbb{R}$ be functions and $L(x, \alpha) = f(x) + \sum_{i=1}^n \alpha_i c_i(x), \alpha_i \geq 0$ the corresponding Lagrange function. If there exists $(\bar{x}, \bar{\alpha})$, so that:*

$$L(\bar{x}, \alpha) \leq L(\bar{x}, \bar{\alpha}) \leq L(x, \bar{\alpha})$$

then $\bar{x}$ is a solution to the constrained optimization problem:

$$\min f(x),\ s.t.\ c_i(x) \leq 0, \forall i = 1, \ldots, n\ .$$

The relation given in the theorem concerning $L(\bar{x}, \bar{\alpha})$ just states that the Lagrangian L is minimal w.r.t. $\bar{x}$ and maximal w.r.t. $\bar{\alpha}$ at a saddle point. For convex and differentiable objective function f and constraints c_i the above theorem can be restated.

Theorem 2 (KKT conditions for convex differentiable problems)**.** *: Let $f : \mathbb{R}^m \mapsto \mathbb{R}$ and $c_i : \mathbb{R}^m \mapsto \mathbb{R}$ be convex differentiable functions. Then $\bar{x}$ is a solution to the optimization problem*

$$\min f(x), \qquad \text{subject to} \quad c_i(x) \leq 0, \forall i = 1, \ldots, n\ ,$$

if there exists some $\bar{\alpha} \geq 0$, such that the following conditions are fulfilled:

$$\frac{\partial L(\bar{x},\bar{\alpha})}{\partial x} = \frac{\partial f(\bar{x})}{\partial x} + \sum_{i=1}^n \alpha_i \frac{\partial c_i(\bar{x})}{\partial x} = 0 \qquad \text{(saddlepoint in } \bar{x}) \tag{22}$$

$$\frac{\partial L(\bar{x},\bar{\alpha})}{\partial \alpha} = c_i(\bar{x}) =\leq 0 \qquad \text{(saddlepoint in } \bar{\alpha}) \tag{23}$$

$$\sum_{i=1}^n \bar{\alpha}_i c_i(\bar{x}) = 0 \qquad \text{(KKT gap)} \tag{24}$$

This is the form of the KKT conditions that will be used in the following subsections to derive stopping conditions for the optimization algorithms. But first it might be helpful to look at a simple example in one dimension to understand the implications of the theorem. Example:

$$\begin{aligned} \min_x\ f(x) &= \frac{1}{2}x^2,\ x \in \mathbb{R} \\ \text{subject to} \quad 3x + 2 \leq 0 &\Leftrightarrow x \leq -\frac{2}{3} \end{aligned} \tag{25}$$

A solution to the problem can be found by looking at the constraint, which gives $x \leq -2/3$. Since $f(x)$ is a monotonically decreasing function for $x < 0$ the function should reach its minimum value at point $x = -2/3$. To verify that this is an optimal solution the KKT conditions (22) have to be checked:

$$\begin{aligned} &L(x, \alpha) = 1/2x^2 + \alpha(3x + 2) \\ &\frac{\partial L(\bar{x}, \bar{\alpha})}{\partial x} = \bar{x} + 3\bar{\alpha} = 0 \\ &\frac{\partial L(\bar{x}, \bar{\alpha})}{\partial \alpha} = 3\bar{x} + 2 \leq 0 \\ &\bar{\alpha}(3\bar{x} + 2) = 0\ . \end{aligned} \tag{26}$$

Substituting $\bar{x} = -2/3$ results in $\bar{\alpha} = 2/9$ which satisfy all of the KKT conditions. Thus $\bar{x}$ in an optimal solution for the example problem (25). A contour plot of $L(x, \alpha)$ is given in figure 4, where the point $(-2/3, 2/9)$ obviously is a saddle point of $L(x, \alpha)$.

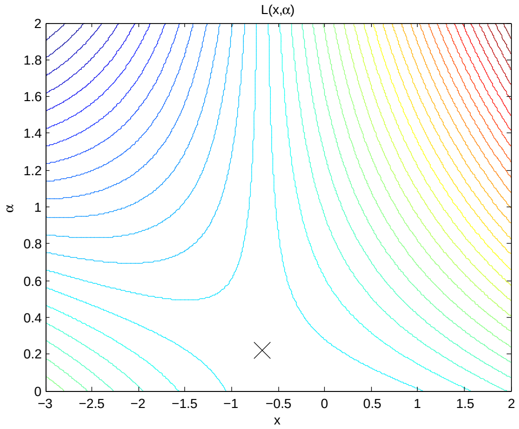

Figure 4: *Contour plot of the Lagrangian function $L(x, \alpha)$. A saddle point of the function is $(-2/3, 2/9)$, thus $\bar{x} = -2/3$ is an optimal solution to the example optimization problem given in the text.*

3.1 Interior Point Algorithm

The idea of interior point algorithms is based on solving the primal and dual QP problem simultaneously by searching for a pair of primal and dual variables which satisfy both the constraints and the KKT conditions (22). A pair of variables which satisfies primal and dual constraints only is called an interior point. The following exposition of the popular LOQO interior point algorithm for solving QPs follows [15]. With $A = [y^T; e^T], b = [\delta_1; \delta_2], l = 0$, $u = C$ and e the vector of all ones the general problem in equation (19) can be stated as:

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^T Q \alpha + p^T \alpha \\ \text{subject to} \quad & A\alpha = b \\ & l \leq \alpha \leq u \end{aligned} \tag{27}$$

By introducing slack variables g, t the inequalities can be reformulated as equality constraints. The resulting primal and dual problem are,

$$\begin{aligned} &\min_{\alpha, g, t} \frac{1}{2}\alpha^T Q \alpha + p^T \alpha \\ \text{subject to} \quad & A\alpha = b \\ & \alpha - g = l \\ & \alpha + t = u \\ & g, t \geq 0 \\ &\max_{y, s, z} \; -\frac{1}{2}\alpha^T Q + b^T y + l^T z + u^T s \\ \text{subject to} \quad & Q\alpha + p - (Ay)^T + s = z \\ & s, z \geq 0 \end{aligned} \tag{28}$$

and the KKT conditions are given by:

$$g_i z_i = 0, \quad s_i t_i = 0, \; \forall i = 1, \dots, m \tag{29}$$

The examination of the primal and dual constraints reveals, that an interior point can be found by solving a system of linear equations. Unfortunately the optimal solution cannot be found directly since the KKT conditions are unsolvable given one of the variables, e.g. g, s or z, t. As a consequence the KKT conditions are relaxed using a variable $\mu > 0$ which is decreased during the iterative solution process, leading to the two equations $g_i z_i = \mu,\ s_i t_i = \mu$. Since for a given μ there is no point in solving (28) exactly one solves the linearized system which results after expanding variables α into $\alpha + \Delta\alpha$ etc.:

$$\begin{aligned} A(\alpha + \Delta\alpha) &= b \\ \alpha + \Delta\alpha - g - \Delta g &= l \\ \alpha + \Delta\alpha + t + \Delta t &= u \\ p + Q\alpha + Q\Delta\alpha - (A(y + \Delta y))^T + s + \Delta s &= z + \Delta z \\ (g_i + \Delta g_i)(z_i + \Delta z_i) &= \mu \\ (s_i + \Delta s_i)(t_i + \Delta t_i) &= \mu \end{aligned} \tag{30}$$

Reformulation of this system yields,

$$A\Delta\alpha = b - A\alpha \qquad =: \rho \tag{31}$$
$$\Delta\alpha - \Delta g = l - \alpha + g \qquad =: \nu \tag{32}$$
$$\Delta\alpha + \Delta t = u - \alpha - t \qquad =: \tau \tag{33}$$
$$(A\Delta y)^T + \Delta z - \Delta s - Q\Delta\alpha = p - (Ay)^T + Q\alpha + s - z \qquad =: \sigma \tag{34}$$
$$g^{-1}z\Delta g + \Delta z = \mu g^{-1} - z - g^{-1}\Delta g\Delta z \qquad =: \gamma_z \tag{35}$$
$$t^{-1}s\Delta t + \Delta s = \mu t^{-1} - s - t^{-1}\Delta t\Delta s \qquad =: \gamma_s \tag{36}$$

where the notation g^{-1}, t^{-1} represents component wise inversion, that is $g^{-1} = (1/g_1, \ldots, 1/g_n)$, and $g^{-1}z, t^{-1}s$ represents component wise multiplication. Solving equation (31) for $\Delta g, \Delta t, \Delta z, \Delta s$ leads to:

$$\begin{aligned} \Delta g &= z^{-1}g(\gamma_z - \Delta z) \\ \Delta t &= s^{-1}t(\gamma_s - \Delta s) \\ \hat{\nu} &= \nu + z^{-1}g\gamma_z \\ \hat{\tau} &= \tau - s^{-1}t\gamma_s \\ \Delta z &= g^{-1}z(\hat{\nu} - \Delta\alpha) \\ \Delta s &= t^{-1}s(\Delta\alpha - \hat{\tau}) \end{aligned} \tag{37}$$

Finally $\Delta\alpha$ and Δy are the solution of the reduced KKT-system [15],

$$\begin{bmatrix} -(Q + g^{-1}z + t^{-1}s) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta\alpha \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma - g^{-1}z\hat{\nu} - t^{-1}s\hat{\tau} \\ \rho \end{bmatrix} \tag{38}$$

which is best solved by Cholesky decomposition [8] and explicit pivoting.

To see how to solve the reduced KKT system let $Q_1 = Q + g^{-1}z + t^{-1}s$, $Q_2 = 0$, $c_1 = \sigma - g^{-1}z\hat{\nu} - t^{-1}s\hat{\tau}$ and $c_2 = \rho$ which in conjunction with equation 38 leads to:

$$-Q_1\Delta\alpha + A^T\Delta y = c_1 \tag{39}$$
$$A\Delta\alpha + Q_2\Delta y = c_2 \tag{40}$$

Now solving (39) for $\Delta\alpha = Q_1^{-1}(A^T\Delta y - c_1)$ and substituting into (40) Δy can be expressed as:

$$\Delta y = (AQ_1^{-1}A^T + Q_2)^{-1}(c_2 + AQ_1^{-1}c_1)\ . \tag{41}$$

Using the Cholesky decomposition $Q_1 = L_1L_1^T$ and solution of the system $L_1Y_1 = A^T$ the first term in (41) can be computed by:

$$AQ_1^{-1}A^T + Q_2 = Y_1^TL_1^TL_1^{-T}L_1^{-1}L_1Y_1 + Q_2 = Y_1^TY_1 + Q_2\ . \tag{42}$$

With the solution Y_2 of the triangular system $L_1Y_2 = c_1$ the second term in (41) can be simplified:

$$c_2 + AQ_1^{-1}c_1 = c_2 + (L_1Y_1)^T L_1^{-T}L_1^{-1}c_1 = c_2 + Y_1^T Y_2 \ . \tag{43}$$

As a result Δy can be determined using the Cholesky decomposition $L_2L_2^T = Y_1^T Y_1 + Q_2$ as well as the factors Y_1 and Y_2. In the last step when Δy is known $\Delta\alpha$ can be computed by back-substitution:

$$\begin{aligned} L_2x &= c_2 + Y_1^T Y_2 \\ L_2^T \Delta y &= x \\ L_1^T \Delta\alpha &= Y_1\Delta y - Y_2 \ . \end{aligned} \tag{44}$$

During the iterative solution of the QP problem the reduced KKT system is usually solved by a predictor-corrector method. The predictor step involves solving (37) and (38) setting $\mu = 0$ and $\Delta z = \Delta s = \Delta\alpha = 0$ on the right hand side, e.g. $\gamma_z = -z$ and $\gamma_s = -s$. For the corrector step the resulting Δ-terms are substituted into the definitions of γ_z and γ_s and the equations (37) and (38) are solved again. At the end of each iteration the Δ-terms thus determined are used to update the values α, s, t, z,etc.. The step length ξ for these updates is chosen such that the new values do not violate the positivity constraints. A heuristic for decreasing μ is given by [20]:

$$\mu = \frac{\langle g, z\rangle + \langle s, t\rangle}{2n} \left(\frac{\xi - 1}{\xi + 10}\right)^2 \quad . \tag{45}$$

Thus μ is decreased rapidly if the average of the feasibility gap given by the first term is large and if the variables are far away from the boundaries of the positivity constraints as indicated by a large ξ in the second term. Such a decrease hence results in a stronger enforcement of the KKT conditions.

Starting points for the iterative procedure are found by solving a modified reduced KKT system (38) by setting auxiliary variables to 0:

$$\begin{bmatrix} -(Q + \mathbf{1}) & A^T \\ A & \mathbf{1} \end{bmatrix} \begin{bmatrix} \alpha \\ y \end{bmatrix} = \begin{bmatrix} p \\ b \end{bmatrix} \tag{46}$$

The positivity of these starting points can be ensured with:

$$\begin{aligned} y &= \max(x, u/100) \\ g &= \min(\alpha - l, u) \\ t &= \min(u - \alpha, u) \\ z &= \min(\max(Q + p - (Ay)^T, 0) + u/100, u) \\ s &= \min(\max(-Q - p + (Ay)^T, 0) + u/100, u) \end{aligned} \tag{47}$$

The runtime of the interior point algorithm is dominated by the Cholesky factorization which is the most expensive step during the iterative solution process. As a result LOQO has a runtime complexity of $\mathcal{O}(m^3)$.

3.2 Gradient Projection Algorithm

The gradient projection algorithm uses simple gradient descent for minimizing the objective function $f(\alpha)$ w.r.t. the optimization variable α. Feasibility of α is maintained by projection on the constraints after each update of the variable α. The two main steps repeated by the algorithm are [1]

1. Compute descent direction $d^t = \mathcal{P}_\Omega(\alpha^t - \delta_t \nabla f(\alpha^t)) - \alpha^k$
2. Determine step size λ_k and update $\alpha^{k+1} \leftarrow \alpha^k + \lambda_k d^k$,

where $\mathcal{P}_\Omega$ is the projection operator and δ_t is the step size found by doing a line-search.

The progress of this algorithm for a simple example is shown in figure 5. Crucial for the practical application of this algorithm are selection of a suitable step size and an efficient projection operation $\mathcal{P}$ on the constraint set Ω.

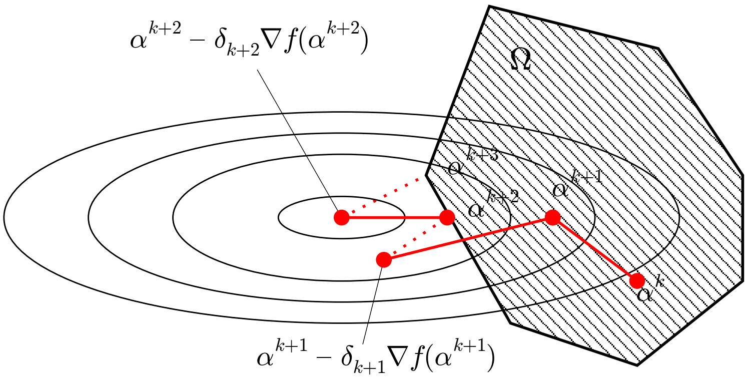
$\alpha^{k+2} - \delta_{k+2}\nabla f(\alpha^{k+2})$
Ω
α^{k+3}
α^{k+2}
α^{k+1}
α^{k}
$\alpha^{k+1} - \delta_{k+1}\nabla f(\alpha^{k+1})$

Figure 5: *This simple example shows the progress made by the gradient projection algorithm during the minimization of function $f(\alpha)$, which is indicated by the contour lines, on the constraint set Ω.*

For C-SVC and ε-SVR training the QP problem to be solved has just a single linear constraint in equation (11). In [5] they propose suitable step size selection rules and an efficient projection operation which are used in [23] to solve the C-SVC QP problem by a gradient projection algorithm.

By exploiting the simple constraint structure of QP problem (19) it is possible to use a gradient projection algorithm for training ν-SVC and ν-SVR. The idea is to reduce the projection operation for the two linear constraints to projecting on problems with a single linear constraint twice. Given the optimization variable α the projected variable β is obtained by solving the problem

$$\begin{aligned} &\min_{\beta} \frac{1}{2}\|\alpha - \beta\|^2 \\ \text{subject to}\quad & y^T\beta = 0 \\ & e^T\beta = \nu m \\ & 0 \le \beta \le 1 \end{aligned} \tag{48}$$

With the substitution of $\beta_i = \alpha_i - \Delta_i, \Delta_i \in \mathbb{R}$ and using $y_i \in \{\pm 1\}$ problem (48) can be reformulated as:

$$\begin{aligned} \min_{\Delta} \frac{1}{2}\|\Delta\|^2 &= \min_{\Delta} \frac{1}{2}(\sum_{y_i=+1} \Delta_i^2 + \sum_{y_i=-1} \Delta_i^2) \\ s.t. \sum_{y_i=+1} \Delta_i - \sum_{y_i=-1} \Delta_i &= y^T\alpha = c_1 \\ \sum_{y_i=+1} \Delta_i + \sum_{y_i=-1} \Delta_i &= e^T\alpha = c_2 \\ \alpha_i - 1 \le \Delta_i \le \alpha_i \; &\forall i = 1, \ldots, m \end{aligned} \tag{49}$$

Now close examination of objective function and constraints reveals that this optimization problem can be split into two smaller optimization problems which are independent of each other:

$$\min \frac{1}{2}\sum_{y_i=+1}\Delta_i^2 \qquad\qquad \min \frac{1}{2}\sum_{y_i=-1}\Delta_i^2 \tag{50}$$

$$\text{subject to} \quad \sum_{y_i=+1}\Delta_i^2 = \frac{1}{2}(c_1+c_2) \qquad \text{subject to} \quad \sum_{y_i=-1}\Delta_i^2 = \frac{1}{2}(c_2-c_1) \tag{51}$$

$$\alpha_i - 1 \leq \Delta_i \leq \alpha_i \ \forall y_i = +1 \qquad\qquad \alpha_i - 1 \leq \Delta_i \leq \alpha_i \ \forall y_i = -1 \tag{52}$$

With this reduction the QP problem (19) can be solved by the algorithm proposed in [5] the only difference being the number of simple projection operations required.

The gradient projection algorithm exhibits good scaling behavior since the main cost in each iteration is a matrix-vector product which has a runtime complexity of $\mathcal{O}(m^2)$ [23]. Unfortunately the gradient projection algorithm is not suited for solving QP problem (19) in practice due to slow convergence and numerical problems.

3.3 Sequential Minimal Optimization

The SMO algorithm proposed by [13] solves QP problem (11) by sequential optimization of only two variables while the values of all other variables are fixed. A solution for the QP problem in two variables can be found analytically and the choice of variables selected in each iteration is guided by the violation of the KKT conditions (22). The optimization problem (11) in two variables can be stated as follows:

$$\min \frac{1}{2}(\alpha_i\alpha_j)\begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj}\end{bmatrix}\begin{bmatrix}\alpha_i \\ \alpha_j\end{bmatrix} + (p_B + Q_{BN}\alpha_N)\begin{bmatrix}\alpha_i \\ \alpha_j\end{bmatrix} \tag{53}$$

$$\text{subject to} \quad y_i\alpha_i + y_j\alpha_j = \delta - y_N^T\alpha_N \tag{54}$$

$$0 \leq \alpha_i, \alpha_j \leq C \; . \tag{55}$$

If $I = \{1, \ldots, m\}$ denotes the index set of all variables then $B = \{i, j\}$ is the index set of those variables currently optimized and $N = I \setminus B$ is the index set of fixed variables. To analytically solve this problem the first step consists of expressing the objective function (53) in dependence of only one optimization variable α_i. Thus the starting point is the objective function which can be rewritten as

$$f(\alpha_i, \alpha_j) = \frac{1}{2}(\alpha_i^2 Q_{ii} + 2\alpha_i\alpha_j Q_{ij} + \alpha_j^2 Q_{jj}) + c_i\alpha_i + c_j\alpha_j \; , \tag{56}$$

where the constants c_i, c_j are given by:

$$\begin{aligned} c_i &= ((p_B + Q_{BN})\alpha_N)_i = \nabla f(\alpha)_i - Q_{ii}\alpha_i^{old} - Q_{ij}\alpha_j^{old} \\ c_j &= ((p_B + Q_{BN})\alpha_N)_j = \nabla f(\alpha)_j - Q_{ij}\alpha_i^{old} - Q_{jj}\alpha_j^{old} \; , \end{aligned} \tag{57}$$

and α^{old} is the value of the optimization variables at the previous optimization step. Because of constraint (54) variable α_j can be expressed by $\alpha_j = y_j(\gamma - y_i\alpha_i), \gamma = (y_j\alpha_i^{old} + y_j\alpha_j^{old})$ and α_j can be eliminated in (56) yielding:

$$\begin{aligned} f(\alpha_i) &= \frac{1}{2}\alpha_i^2 Q_{ii} + \alpha_i(y_j\gamma - y_i y_j\alpha_i)Q_{ij} + \frac{1}{2}(y_j\gamma y_i y_j\alpha_i)^2 Q_{jj} + c_i\alpha_i + c_j(y_j\gamma - y_j y_i\alpha_i) \\ &= \frac{1}{2}\alpha_i^2(Q_{ii} - 2y_i y_j Q_{ij} + Q_{jj}) + \alpha_i(y_j\gamma Q_{ij} - y_i\gamma Q_{jj} + c_i - c_j y_i y_j) \\ &+ \frac{1}{2}\gamma^2 Q_{jj} + c_j y_j\gamma \; . \end{aligned}$$

Now the location of the minimum for $f(\alpha_i)$ is determined by computing the derivative, setting it to zero and solving for α_i:

$$f'(\alpha_i) = \alpha_i(Q_{ii} - 2y_i y_j Q_{ij} + Q_{jj}) + (y_j\gamma Q_{ij} - y_i\gamma Q_{jj} + c_i - c_j y_i y_j) \overset{!}{=} 0 \tag{58}$$

$$\Rightarrow \alpha_i = \frac{y_i\gamma Q_{jj} - y_j\gamma Q_{ij} - c_i + c_j y_i y_j}{Q_{ii} - 2y_i y_j Q_{ij} + Q_{jj}} \tag{59}$$

A similar expression can be derived for α_j via elimination of α_i in the objective function. To get update equations for α_i and α_j it is beneficial to distinguish between two cases, namely $y_i = y_j$ and $y_i \neq y_j$:

$$\begin{aligned}
&y_i = y_j : \\
&\qquad \alpha_i = \frac{y_i \gamma Q_{jj} - y_j \gamma Q_{ij} - c_i + c_j}{Q_{ii} - 2Q_{ij} + Q_{jj}} \\
&\qquad\quad = \frac{\alpha_i^{old}(Q_{ii} - 2Q_{ij} + Q_{jj}) + \nabla f(\alpha)_j - \nabla f(\alpha)_i}{Q_{ii} - 2Q_{ij} + Q_{jj}} \\
&\qquad\quad = \alpha_i^{old} + \frac{\nabla f(\alpha)_j - \nabla f(\alpha)_i}{Q_{ii} - 2Q_{ij} + Q_{jj}} \\
&\qquad \alpha_j = \alpha_j^{old} + \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{Q_{ii} - 2Q_{ij} + Q_{jj}} \\
&y_i \neq y_j : \\
&\qquad \alpha_i = \frac{y_i \gamma Q_{jj} - y_j \gamma Q_{ij} - c_i - c_j}{Q_{ii} + 2Q_{ij} + Q_{jj}} \\
&\qquad\quad = \frac{\alpha_i^{old}(Q_{ii} + 2Q_{ij} + Q_{jj}) - \nabla f(\alpha)_i - \nabla f(\alpha)_j}{Q_{ii} + 2Q_{ij} + Q_{jj}} \\
&\qquad\quad = \alpha_i^{old} + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{Q_{ii} + 2Q_{ij} + Q_{jj}} \\
&\qquad \alpha_j = \alpha_j^{old} + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{Q_{ii} + 2Q_{ij} + Q_{jj}}
\end{aligned}$$

In the next step after updating the optimization variables one has to ensure that constraints (54) and (55) are satisfied. With $\alpha_j = y_j(\gamma - \alpha_i y_i)$ and $0 \leq \alpha_j \leq C$ the following constraints for α_i are derived:

$$y_i y_j \alpha_i^{old} + \alpha_j^{old} - C \leq y_i y_j \alpha_i \leq y_i y_j \alpha_i^{old} + \alpha_j^{old} \tag{60}$$

$$0 \leq \alpha_i \leq C \; . \tag{61}$$

Again the discussion is simplified by considering cases $y_i = y_j$ and $y_i \neq y_j$ separately. For $y_i = y_j$ the constraints (60) can be combined into:

$$\max(0, \sigma - C) \leq \alpha_i \leq \min(C, \sigma), \quad \text{with } \sigma = \alpha_i^{old} + \alpha_j^{old} \; . \tag{62}$$

Since by construction of the optimal solution $\alpha_j = \alpha_i^{old} + \alpha_j^{old} - \alpha_i = \sigma - \alpha_i$ for $y_i = y_j$ the decision on how to change α_i, α_j to satisfy the constraints can be solely based on the value of σ and α_i. The result of this reasoning are the following update rules for α_i and α_j:

$$\begin{aligned}
&\sigma > C : \\
&\qquad \alpha_i < 0 : \alpha_i \leftarrow \sigma - C, \alpha_j \leftarrow C \\
&\qquad \alpha_i > C : \alpha_i \leftarrow C, \alpha_j \leftarrow \sigma - C \\
&\sigma < C : \\
&\qquad \alpha_i < 0 : \alpha_i \leftarrow 0, \alpha_j \leftarrow \sigma \\
&\qquad \alpha_i > C : \alpha_i \leftarrow \sigma, \alpha_j \leftarrow 0 \; .
\end{aligned}$$

For $y_i \neq y_j$ similar update rules can be derived from on the combined constraint

$$\max(0, \rho) \leq \alpha_i \leq \min(C + \rho, C), \quad \text{with } \rho = \alpha_i^{old} - \alpha_j^{old} \tag{63}$$

leading to:

$$
\begin{aligned}
&\rho > 0 : \\
&\qquad \alpha_i < 0 : \alpha_i \leftarrow \rho, \alpha_j \leftarrow 0 \\
&\qquad \alpha_i > C : \alpha_i \leftarrow C, \alpha_j \leftarrow C - \rho \\
&\rho < 0 : \\
&\qquad \alpha_i < 0 : \alpha_i \leftarrow 0, \alpha_j \leftarrow \rho \\
&\qquad \alpha_i > C : \alpha_i \leftarrow C + \rho, \alpha_j \leftarrow C \ .
\end{aligned}
$$

With these update rules the only missing pieces to complete SMO are a suitable stopping condition for the optimization loop and a selection criterion for α_i, α_j. The stopping condition is derived for (53) from the general KKT conditions in theorem 2. The Lagrangian in this case is:

$$L(\alpha, b, \lambda, \mu) = \frac{1}{2}\alpha^T Q\alpha + p^T\alpha - b(\delta - y^T\alpha) - \lambda\alpha - \mu(C - \alpha) \ . \tag{64}$$

Application of theorem 2 results in the following KKT conditions:

$$
\begin{aligned}
&\frac{\partial L}{\partial \alpha} = \nabla f(\alpha) + by - \lambda + \mu = 0 \Leftrightarrow \nabla f(\alpha) + by = \lambda - \mu && (65)\\
&\frac{\partial L}{\partial b} = y^T\alpha - \delta = 0 && (66)\\
&\frac{\partial L}{\partial \lambda} = -\alpha \leq 0 \Leftrightarrow \alpha \geq 0 && (67)\\
&\frac{\partial L}{\partial \mu} = \alpha - C \leq 0 \Leftrightarrow \alpha \leq C && (68)\\
&\sum_{i=1}^{m} \mu_i(C - \alpha_i) + \sum_{i=1}^{m} \lambda_i\alpha_i + \sum_{i=1}^{m} y_i\alpha_i - \delta = 0 && (69)
\end{aligned}
$$

Combining conditions (66) and (69) leads to $\mu_i(C-\alpha_i) = 0, \mu_i \geq 0$ and $\lambda_i\alpha_i = 0, \lambda_i \geq 0$ for all i. Closer analysis of these conditions reveals the following identities:

$$
\begin{aligned}
\lambda_i\alpha_i = 0 &\Leftrightarrow (\alpha_i = 0 \wedge \lambda_i > 0) \vee (\alpha_i > 0 \wedge \lambda_i = 0)\\
\mu_i(C - \alpha_i) = 0 &\Leftrightarrow (\alpha_i = C \wedge \mu_i > 0) \vee (\alpha_i < C \wedge \mu_i = 0)\\
&\Rightarrow \lambda_i - \mu_i \geq 0 \Leftrightarrow \alpha_i < C\\
&\Rightarrow \lambda_i - \mu_i \leq 0 \Leftrightarrow \alpha_i > 0
\end{aligned}
$$

Since $\nabla f(\alpha_i) + by_i \geq 0 \Leftrightarrow \lambda_i - \mu_i \geq 0$ and $\nabla f(\alpha_i) + by_i \leq 0 \Leftrightarrow \lambda_i - \mu_i \leq 0$ by condition (65) with the aid of the identities above this KKT condition can be reformulated as

$$\nabla f(\alpha)_i + b \geq 0 \ \forall i \in I_{up} = \{i|(\alpha_i < C \wedge y_i = +1) \vee (\alpha_i > 0 \wedge y_i = -1\} \tag{70}$$

$$\nabla f(\alpha)_i + b \leq 0 \ \forall i \in I_{low} = \{i|(\alpha_i > 0 \wedge y_i = +1) \vee (\alpha_i < C \wedge y_i = -1\} \tag{71}$$

exploiting the fact that $y_i \in \{\pm 1\}$. With these definitions a suitable stopping condition for the optimization procedure is given by:

$$\max_{i \in I_{up}}(-y_i\nabla f(\alpha)_i) - \min_{i \in I_{low}}(-y_i\nabla f(\alpha)_i) \leq \epsilon \tag{72}$$

where ϵ is a small positive constant which controls to what extent the KKT conditions have to be fulfilled before stopping the optimization procedure. The variables α_i, α_j to be optimized in each step are those that maximize the progress w.r.t. the KKT gap. Therefore these variables are often called the 'maximal violating pair' given by:

$$
\begin{aligned}
i &= \arg\max_{i \in I_{up}}(-y_i\nabla f(\alpha)_i)\\
j &= \arg\min_{j \in I_{low}}(-y_j\nabla f(\alpha)_j)
\end{aligned}
$$

In comparison to the interior point algorithm in section 3.1 and the gradient projection algorithm in section 3.2 SMO has the big advantage of not running into numerical problems due to its analytical nature. Empirical experiments [13] with datasets of different sizes have shown, that SMO scales roughly with $\mathcal{O}(m^2)$ in practice. Unfortunately its sequential solution procedure does not lead to straightforward parallelization strategy for this algorithm. Despite this disadvantage the algorithm can be nonetheless successfully employed as inner solver for a decomposition based parallelization strategy.

4 Decomposition for large scale SVM training

In principle all the methods described in section 3 can be used to train SVMs for classification and regression tasks. Yet the interior point algorithm in section 3.1 and the gradient projection algorithm in section 3.2 are not suited to solve large scale problems with 10^5 - 10^6 patterns in practice due to their runtime complexities of $\mathcal{O}(m^3)$ and $\mathcal{O}(m^2)$. Another issue is the storage requirement of the kernel matrix. A dataset with 60000 patterns requires more than 13GB of memory for example if each entry is assumed to be a single precision floating point value.

To deal with these issues [11] introduced a decomposition method for breaking down the original QP problem into several smaller problems

$$\min \frac{1}{2}\alpha_B^T \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix} \alpha_B + (p_B + Q_{BN}\alpha_N)\alpha_B \tag{73}$$

$$\text{subject to} \quad y_B^T\alpha_B = \delta - y_N^T\alpha_N \tag{74}$$

$$0 \leq \alpha_B \leq C \; . \tag{75}$$

where B is the index set of the currently optimized variables and N the index set of the fixed variables. The SMO algorithm explained in section 3.3 essentially uses this decomposition idea in the extreme case where each subproblem has size two. To avoid storing the complete kernel matrix in main memory [11] proposes a caching scheme where only the most recently used kernel matrix rows are stored in memory.

Solving each of the subproblems arising in the decomposition approach can be done with all of the algorithms described in section 3. The only remaining question to be answered concerns the selection of an appropriate working set B and a stopping condition for the decomposition approach.

4.1 Working set selection

For the selection of the working set B the reasoning given in section 3.3 for the 'maximal violating pair' can be generalized for selecting more than two variables. Sorting the index set I of all optimization variables into a list in decreasing order w.r.t. $-y_i\nabla f(\alpha)_i$ and selecting pairs of variables, where the first variable is from the top of the list with $i \in I_{up}$ and the second variable is from the bottom of the list with $i \in I_{low}$, results in a working set B where each pair is in some sense a 'maximal violating pair'. With this selection strategy it is possible to fill the working set with more than two variables, but the number of selected variables might be less than the required working set size. Consequently, if necessary, the working set is filled up with the most recent indices[1] in the previous working set that are not yet in B, where preference is usually given to free variables [23].

Another important point of the working set selection strategy is the number of new variables n that enter the working set at each step. If n is chosen to be equal to the size of the working set ($n = |B| = q$) a so called 'zigzagging' of variables might occur, that is, some variables might enter and leave the working set for many times which in turn can slow down the optimization algorithm considerably. A suitable initial value for n that works in practice is $n = q/2$. To get faster convergence n is decreased during the optimization process as

[1] Indices that are in B for the lowest number of consecutive iterations.

described in [23]. With these considerations in mind the strategy for selecting B can be summarized as follows:

1. Let q be the required working set size and n the number of new variables to enter the working set.
2. Sort the index set I into decreasing order w.r.t. $-y_i \nabla f(\alpha)_i$ and let $(i_1, \ldots, i_n)$ be the sorted index sequence.
3. Select pairs (i_u, i_l) of indices with $l < u$ from the sequence where $i_u \in I_{up}$ and $i_l \in I_{low}$ until n indices are selected or no pair satisfying the above conditions can be found.
4. Let B' be the working set selected so far.
5. If $|B'| < q$ fill up B' with the most recent indices $i \in B \setminus B'$ with $0 < \alpha_i < C$ (free variables).
6. If $|B'| < n$ fill up B' with the most recent indices $i \in B \setminus B'$ with $\alpha_i = 0$ (variables at lower bound).
7. If $|B'| < n$ fill up B' with the most recent indices $i \in B \setminus B'$ with $\alpha_i = C$ (variables at upper bound).
8. Adapt n by setting $n = \min(n, \max(10, q', n'))$, where q' is the largest even integer with $q' < q/10$ and n' is the largest even integer with $n' < |\{i, i \in B' \setminus B\}|$. Set $B = B'$.

4.2 Stopping condition

As stopping condition for the decomposition approach the stopping condition (72) for the SMO algorithm in section 3.3 can be used. In this case the index sets I_{up} and I_{low} are subsets of the whole index set $I = B \cup N$.

5 Implementation

It was already pointed out in the last section that the decomposed QP problem (73) can be solved by all of the QP solvers described in section 3. After implementing these algorithms it becomes apparent that not all are suited to solve large scale SVM problems in practice. This section gives some hints on why the interior point algorithm and the gradient projection algorithm are not the first choice in practice and explain why a parallel SMO implementation should be preferred.

In [23] the gradient projection algorithm exhibits a very good parallel behavior for large scale C-SVC training. With the reformulation for ν-SVC given in section 3.2 it is possible to apply the gradient projection algorithm to the problem of large scale ν-SVC training. For the sequential implementation of this approach the code [2] of [23] is modified accordingly. Unfortunately the two step projection for solving the ν-SVC problem does not work well in practice. At least it only works for some datasets and settings of the parameter ν while on most datasets this approach failed to converge due to numerical problems. Therefore the gradient projection algorithm is not considered any further in this study.

The LOQO interior point algorithm described in section 3.1 is implemented using the parallel linear algebra library PLAPACK [19]. This library contains a very good parallel Cholesky solver for dense matrices which is essential for solving the reduced KKT system. Testing this approach on several large scale datasets with different sizes of the working set reveals that there is a linear relationship [3] between the working set size and the number of iterations required by the decomposition approach to converge (Table 1). Therefore one could expect a linear speedup when increasing the working set size which is not observed in practice as the runtime on a fixed number of processors is almost constant. This is caused by the runtime complexity of the interior point algorithm which scales with $\mathcal{O}(m^3)$, where

[2] Available at: `http://dm.unife.it/gpdt/`

[3] For a limited range.

Number of	Working Set	Number of	CPU Time
Processors	Size	Iterations	in min
1	256/128	103	64.85
2	256/128	103	51.23
4	256/128	103	29.53
1	512/256	51	65.48
2	512/256	51	51.93
4	512/256	51	30.05
1	1024/512	24	68.25
2	1024/512	24	53.46
4	1024/512	24	30.65

Table 1: *Performance of the decomposition approach using LOQO as inner solver for the dataset mnist-576-rbf-8vr.*

in this case m is the number of variables in the working set. When increasing the number of processors the runtime complexity of LOQO also explains the bad parallel performance. Figure 6 shows the results of LOQO in comparison to parallel SMO (described next) on one of the MNIST datasets (cf. section A). In addition to this LOQO has convergence problems on a subset of the ν-SVC tasks and for certain values of the parameter ν.

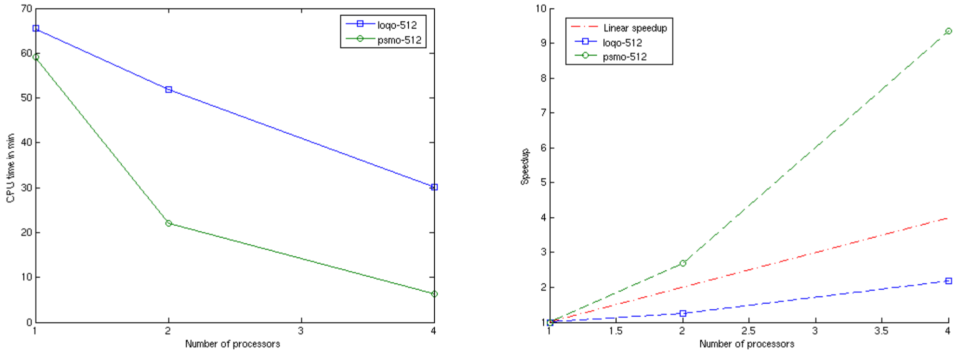

Figure 6: *Comparison of LOQO and parallel SMO (PSMO) with respect to runtime (left) and speedup (right) on the mnist-576-rbf-8vr dataset. Due to the $\mathcal{O}(m^3)$ runtime complexity of LOQO the parallel decomposition approach using LOQO as inner solver does not scale well. On the other hand PSMO is able to achieve a superlinear speedup for this dataset. The size of the working set in both cases is $q = 512$ and the number of new variables entering the working set is $n = 256$.*

To avoid the problems just mentioned a parallel implementation of the SMO algorithm described in section 3.3 can be used. This approach will be termed PSMO in the following discussion. It is based on the observation that in practice the main computational burden is not the solution of the inner QP problem, as long as the working set size is small and contains about $256 - 2048$ variables. Profiling information gathered for the parallel implementations on different datasets indicates that the computational bottleneck are the kernel evaluations which are needed to update the gradient $\nabla f(\alpha)$. Note that updating the gradient is essential for the working set selection and the evaluation of the stopping condition (72). The profiling information reveals that between $90 - 98\%$ of the runtime is spend on updating the gradient. This is the motivation for PSMO which uses the sequential SMO algorithm for solving the subproblems arising in the decomposition approach while performing problem setup, kernel evaluations, caching of kernel rows and gradient updates in parallel. Important for achieving good speedups is a good load balancing between the processors which in PSMO achieves by a distributed caching strategy.

The distributed caching strategy basically assigns the computational tasks with a round-robin strategy. Before the gradient is updated in each iteration the following steps are executed:

1. Each processor determines the kernel rows with indices in the current working set B, which are cached/not cached locally.
2. Next the local cache information is synchronized across all processors.
3. Using the global cache information the index set B is split into cached B_c and non-cached B_{nc} indices.
4. B_c is distributed among the processors with a round-robin strategy[4] Let the resulting sets be B_c^{local} and B_{nc}^{local}.
5. Then each processor updates all components of $\nabla f(\alpha)_i$, with index $i \in B_c^{local} \cup B_{nc}^{local}$. Cached entries are updated before the non-cached ones.
6. Finally the gradient $\nabla f(\alpha)$ is synchronized between all processors.

PSMO was implemented using a modified SMO from the LibSVM [4] software version 2.8. One change of the LibSVM library involves the the sparse data representation, which is replaced by a new sparse data structure recommended by the BLAS Technical Forum. [5] All communication that is necessary between the processors for the distributed caching strategy and data synchronization is implemented using the Message Passing Interface (MPI) [7].

6 Results

The results presented in the following subsections for C-SVC, ν-SVC , ε-SVR and ν-SVR are all based on the PSMO implementation of section 5. For classification four datasets mnist-576-rbf-8vr, mnist-784-poly-8vr, covtype-2vr and kddcup99-nvr are used. The two regression datasets are kddcup98 and mv (for details cf. section A). All performance tests are run on the Kepler [6] cluster, which has 32 Dual AMD Athlon MP 2000+ nodes with 1666 MHz , 256 L2 Cache and 1-2GB RAM running Linux (2.4.21 kernel). Communication between nodes occurs over a Myrinet interconnect with MPI Peak performance of 115MB per second and node. Because of technical reasons it is not possible to run programs on more than 8 processors. Therefore in the following all performance results are given for up to 8 processors. The size of the distributed cache is set to 256MB for all tests.

[4] First processor gets first index, second processor gets second index etc., wrapping around if necessary.

[5] http://www.netlib.org/blas/blast-forum/

[6] `http://kepler.sfb382-zdv.uni-tuebingen.de/kepler/index.shtml`

6.1 C-SVC

Parallel solution of the C-SVC problem for the classification datasets mnist-576-rbf-8vr, mnist-784-poly-8vr and covtype-2vr with PSMO yields a superlinear speedup on up to 8 processors as shown in figure 7 and figure 8. For the kddcup99-nvr an almost linear speedup is achieved on up to 4 processors (figure 8). The parameters used for training PSMO and LibSVM on the four datasets are given in table 2. To put these speedups in relation to LibSVM performance on a single processor, the runtime of LibSVM on one CPU is also measured and listed in table 3. On both MNIST datasets the single processor runtime of PSMO is better than that of LibSVM, whereas on the other two datasets LibSVM outperforms PSMO in the sequential case. But it is important to note that the single processor runtime of PSMO on these datasets could be potentially improved by choosing a different working set size. Nonetheless PSMO on four processors is still twice as fast as LibSVM on a single processor for the covtype-2vr dataset and a constant two hours faster for the kddcup99-nvr dataset.

Dataset	C-SVC Parameters	Working Set Size
mnist-576-rbf-8vr	$C = 10$, $\gamma = 1.667$	$512/256$
mnist-784-rbf-8vr	$C = 10$, $d = 7$	$512/256$
covtype-2vr	$C = 10$, $\gamma = 2e - 5$	$1024/512$
kddcup99-nvr	$C = 2$, $\gamma = 0.6$	$512/256$

Table 2: *PSMO and LibSVM training parameters and working set size for the C-SVC problems*

Dataset	CPU Time		Test Error	
	PSMO	LibSVM v2.8	PSMO	LibSVM v2.8
mnist-576-rbf-8vr	59.13 [m]	103.5 [m]	99.82%	99.82%
mnist-784-rbf-8vr	126.24 [m]	153.90 [m]	99.51%	99.51%
covtype-2vr	31.12 [h]	11.71 [h]	96.35%	96.36%
kddcup99-nvr	20.89 [h]	7.369 [h]	92.71%	92.71%

Table 3: *Single processor CPU Time and test error for PSMO in comparison with LibSVM v2.8 for the C-SVC problems.*

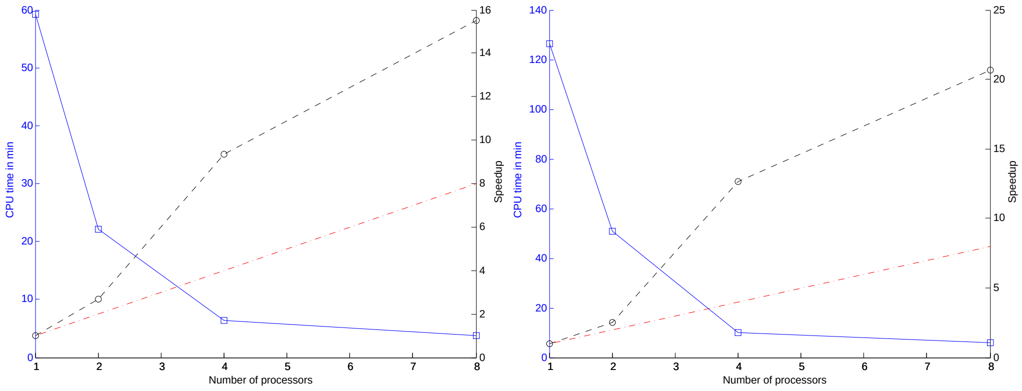

Figure 7: *C-SVC Speedup and CPU time for dataset mnist-576-rbf-8vr (left) and dataset mnist-784-poly-8vr (right).*

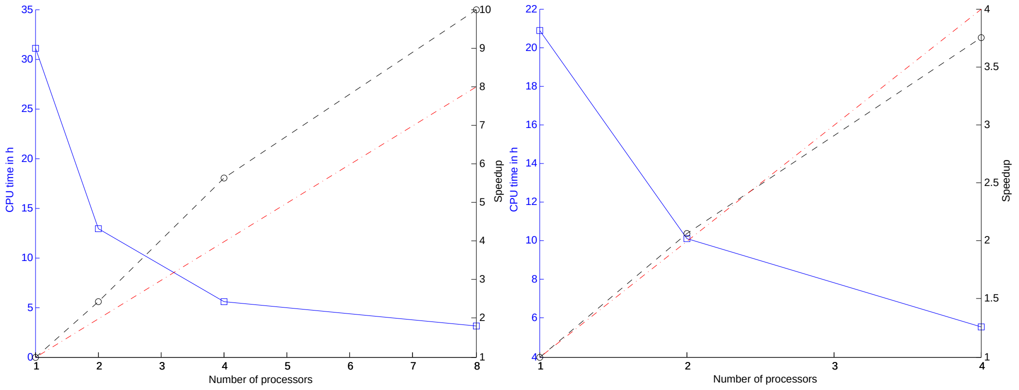

Figure 8: *C-SVC Speedup and CPU time for dataset covtype-tr-2vr (left) and dataset kddcup99-nvr (right).*

6.2 ν-SVC

Table 4 summarizes the parameters used for ν-SVC training on the four classification datasets. As shown in figure 9 and figure 10 superlinear speedups are achieved again on up to 8 processors. The only exception being the runtime for the mnist-576-rbf-8vr dataset where a superlinear speedup is observable for up to 4 processors. Comparison of single processor runtime with LibSVM shows the same situation as for C-SVC where PSMO runtime for mnist-576-rbf-8vr and mnist-784-poly-8vr is better than LibSVM.

Dataset	ν-SVC Parameters	Working Set Size
mnist-576-rbf-8vr	$\nu = 0.002356, \gamma = 1.667$	$512/256$
mnist-784-rbf-8vr	$\nu = 0.006753, d = 7$	$512/256$
covtype-2vr	$\nu = 0.131544, \gamma = 2e - 5$	$1024/512$
kddcup99-nvr	$\nu = 0.001164, \gamma = 0.6$	$512/256$

Table 4: *PSMO and LibSVM training parameters and working set size for the ν-SVC problems*

Dataset	CPU Time		Test Error	
	PSMO	LibSVM v2.8	PSMO	LibSVM v2.8
mnist-576-rbf-8vr	40.76 [m]	98.55 [m]	99.82%	99.82%
mnist-784-rbf-8vr	87.05 [m]	153.60 [m]	99.51%	99.51%
covtype-2vr	25.06 [h]	23.72 [h]	96.34%	96.33%
kddcup99-nvr	43.89 [h]	23.35[h]	92.71%	92.71%

Table 5: *Single processor CPU Time and test error for PSMO in comparison with LibSVM v2.8 for the ν-SVC problems.*

The difference in runtime for covtype-2vr is about two hours whereas LibSVM is twice as fast on kddcup99-nvr. When PSMO is run in parallel on 4 processors the runtime for kddcup99-nvr is cut down to 10 hours, which is twice as fast as the runtime of LibSVM. The test error reported in table 2 indicates that the parallelization technique employed in PSMO does not influence the classification performance.

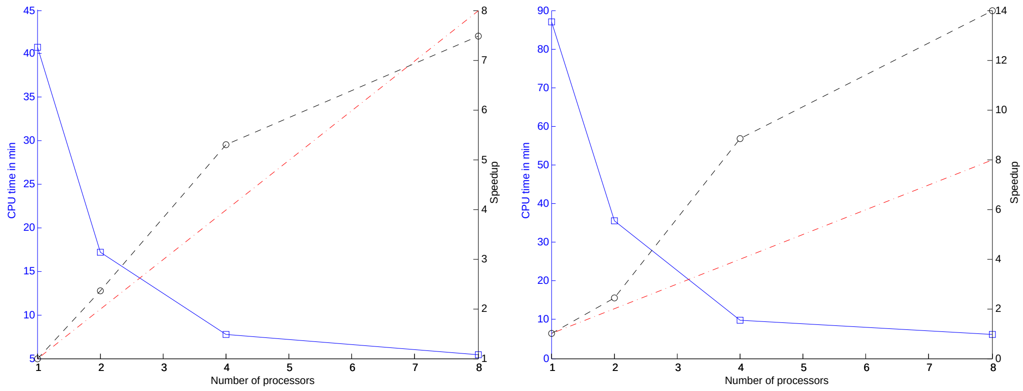

Figure 9: *ν-SVC Speedup and CPU time for dataset mnist-576-rbf-8vr (left) and dataset mnist-784-poly-8vr (right).*

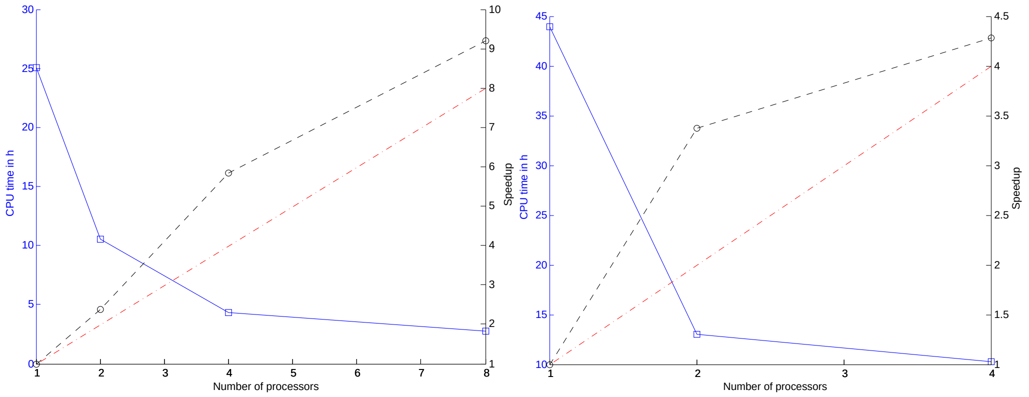

Figure 10: *ν-SVC Speedup and CPU time for dataset covtype-2vr (left) and dataset kddcup-nvr (right).*

6.3 ε-SVR

Training parameters used for ε-SVR on the datasets kddcup98 and mv are listed in table 6. The performance comparison of PSMO with LibSVM on a single processor shows that there is no difference in training time for the mv dataset. For the kddcup98 dataset PSMO is approximately three times as fast as LibSVM while the quality difference of the results, in terms of mean squared error (MSE) on the test set, is negligible (table 7).

Dataset	ε-SVR Parameters	Working Set Size
kddcup98	$C = 0.0078$, $\varepsilon = 0.01$, $\gamma = 13.6436$	$512/256$
mv	$C = 32$, $\varepsilon = 0.01$, $\gamma = 0.1084$	$512/256$

Table 6: *PSMO and LibSVM training parameters and working set size for the ε-SVR problems*

It can be seen in figure 11 that the speedup of PSMO for the ε-SVR is not linear for both datasets. For the mv dataset this can be attributed to the low speedup potential of this dataset that manifests itself in the small number of input patterns and low dimensionality of the data on the one hand and the short single processor runtime of about 20 minutes on the other hand. But this argumentation cannot be used to explain the behavior of PSMO on the kddcup98 dataset. Here one could speculate that the distributed caching strategy does not

Dataset	CPU Time		Test MSE	
	PSMO	LibSVM v2.8	PSMO	LibSVM v2.8
kddcup98	8.671 [h]	29.51[h]	6.06e-04	6.07e-04
mv	19.86 [m]	20.0 [m]	3.15e-05	3.24e-05

Table 7: *Single processor CPU Time and test error for PSMO in comparison with LibSVM v2.8 for the ε-SVR problems.*

work well, when the fraction of support vectors is low. Since for kddcup98 approximately half of the input patterns end up as support vector this cannot explain the lower speedup achieved by PSMO on this datasets and further investigations are necessary to elucidate the relationship between speedup and dataset properties.

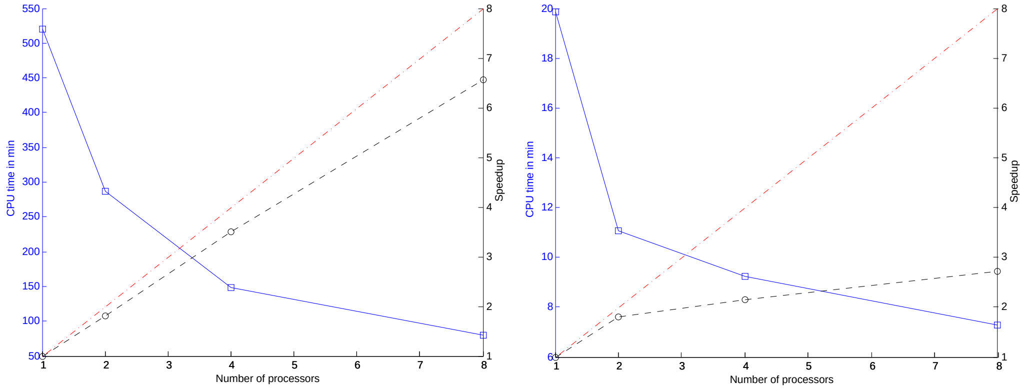

Figure 11: *ε-SVR Speedup and CPU time for dataset kddcup98 (left) and dataset mv (right).*

6.4 ν-SVR

The training of ν-SVR on the datasets mv and kddcup98 yields results with quality similar to C-SVR when the parameters given in table 8 are used. When viewed with respect to runtime and speedup the results give the same picture as for C-SVR the only exception being the runtime of LibSVM for the mv dataset which is about four times as high as in the C-SVR case. The statements made about the speedup potential of the datasets in section 6.3 also hold for the parallel ν-SVR training.

Dataset	ν-SVR Parameters	Working Set Size
kddcup98	$C = 0.0078$, $\nu = 0.092862$, $\gamma = 13.6436$	$512/256$
mv	$C = 32$, $\nu = 0.020947$, $\gamma = 0.1084$	$512/256$

Table 8: *PSMO and LibSVM training parameters and working set size for the ν-SVR problems*

Dataset	CPU Time		Test MSE	
	PSMO	LibSVM v2.8	PSMO	LibSVM v2.8
kddcup98	8.983 [h]	29.85[h]	6.05e-04	6.06e-04
mv	24.43 [m]	82.60 [m]	3.24e-05	3.21e-05

Table 9: *Single processor CPU Time and test error for PSMO in comparison with LibSVM v2.8 for the ν-SVR problems.*

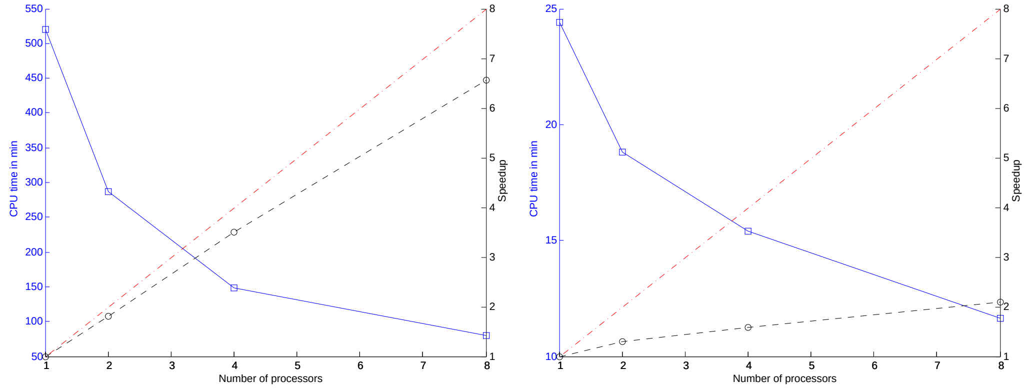

Figure 12: *ν-SVR Speedup and CPU time for dataset kddcup98 (right) and dataset mv (left).*

7 Conclusion

This article described various ways how to parallelize SVM training for the original non-simplified SVM formulations including C-SVC ν-SVC, ε-SVR and ν-SVR. Three different parallelization strategies arise from the use of the interior point algorithm, the gradient projection algorithm or SMO in combination with the decomposition approach for SVM training. While the gradient projection algorithm has already been successfully used for parallel C-SVC section 3.2 described how to extend the algorithm to solve QP problems with two linear constraints that need to be solved when training ν-SVC and ν-SVR. Although this extension is theoretically possible it does not work in practice due to slow convergence. Similar practical experience with the parallel LOQO implementation of the interior point algorithm and the careful analysis of profiling information have led to the implementation of PSMO. Despite the fact that PSMO uses a sequential inner QP solver it is possible to achieve superlinear speedups for C-SVC and ν-SVR. In the regression setting PSMO showed close to linear speedup on the examined kddcup98 dataset while on the mv dataset it is still unclear why only moderate speedups are obtained. Further work is needed to elucidate the relationship between speedup and properties of the dataset. Another important point to investigate in the future concerns an optimal parallelization strategy in terms of speedup or runtime for multi-class problems.

A Description of datasets

An overview of all the datasets used in this study is given in table 10. Datasets were selected to ease comparison with similar studies like [23, 17]. The preprocessing of each dataset and the selection of SVM parameters are described in detail in the following subsections. All datasets are available for download at `http://pisvm.sourceforge.net`. Pointers to the original sources of the datasets are provided at the same location. Kernels used for these datasets include the RBF kernel $k(x_i, x_j) = e^{-\gamma\|x_i - x_j\|^2}$ and the polynomial kernel $k(x_i, x_j) = \langle x_i, x_j\rangle^d$.

Dataset	Number of Patterns		Number of Dimensions
	Train	Test	
mnist-576-rbf-8vr	60000	10000	576
mnist-784-poly-8vr	60000	10000	784
covtype-2vr	435759	145253	54
kddcup99-nvr	4898430	311029	122
kddcup98	95412	96367	403
mv	36768	4000	10

Table 10: *Overview of dataset size and dimension.*

A.1 mnist-576-rbf-8vr and mnist-784-poly-8vr

Both datasets originate from the MNIST dataset for handwritten digit recognition and only differ in the type of preprocessing that is done. For mnist-576-rbf-8vr which is used in conjunction with the RBF kernel a 576-dimensional discriminative feature vector is extracted from the original data [6]. The dataset mnist-784-poly-8vr is prepared by centering each digit image in a 28×28 box, smoothing with a 3×3 mask (center element 1/2, rest 1/16) and normalizing each pattern, such that its dot product is always within $[0, 1]$ [6]. SVC parameters are $C = 10$ for both datasets, $\gamma = 1.667$ for mnist-576-rbf-8vr and $d = 7$ for mnist-784-poly-8vr and are determined using cross-validation on a subset of the training data [6]. Finally the 10-class problem of the MNIST dataset is reduced to a 2-class problem by separating digit 8 from the rest [23].

A.2 covtype-2vr

The task of distinguishing between 8 different classes of forest covertype is represented by the covtype dataset. For the conversion to a binary problem class 2 is to be separated from the other classes. Preparation of the dataset and choice of SVM parameters is done as described in [23]. The RBF kernel is used with parameter $\gamma = 2e - 5$, the regularization parameter of the SVC is set to $C = 10$ and the stopping condition is $\epsilon = 0.01$.

A.3 kddcup99-nvr

The kddcup99-nvr dataset is based on an intrusion detection problem. During preprocessing of the dataset it became apparent that pattern 4817100 obviously contained data formatting errors and was removed from the training dataset. Furthermore symbolic features in the original dataset are converted to unary coded features and all features are scaled to lie in the interval $[0, 1]$ following [18]. Parameters are set as in [23] with $\gamma = 0.6$, $C = 2$ and stopping condition $\epsilon = 0.01$.

A.4 kddcup98

This regression dataset was originally provided by the Paralyzed Veterans of America (PVA) a non-profit organization that provides programs and services for US veterans with spinal cord injuries or diseases. Since most of the funding of PVA is raised by mailing donors the goal is to maximize the donated money in dependence of behavioral and social features of the donors. By including only numerical features the original dataset is reduced to contain 403 features. Missing values are imputed by replacing them by the mean of the given values. Then the features and target values are scaled to lie in the interval $[0, 1]$. To estimate the RBF kernel parameter γ the method proposed in [18] is used, that is $\gamma = 1/(1/m^2) \sum_{i,j=1}^{m} \|x_i - x_j\|^2$, leading to $\gamma = 13.6436$ for this dataset. Finally SVR parameters are selected by 5-fold cross-validation with $C \in \{2^{-7}, 2^{-5}, \dots, 2^{7}\}$ and $\varepsilon \in \{0.01\}$ resulting in $C = 2^{-7}$ and $\varepsilon = 0.01$. All parameters are selected on a 1000 element subset of the training data.

A.5 mv

Estimation of SVR parameters follows the description given in section A.4 for dataset kddcup98 and results in $\gamma = 0.1084$, $C = 32$ and $\varepsilon = 0.01$. The dataset is an artificial regression task with dependencies among the features.

References

[1] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 2003.

[2] A. Bode. Multicore-Architekturen. *Informatik Spektrum*, 29(5):349–352, October 2006.

[3] Mihai Bădoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *SODA '03: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a Library for Support Vector Machines, 2006. Software available at
`http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[5] Yu-Hong Dai and Roger Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Math. Progm. Ser. A*, (106):403–421, October 2005.

[6] Jian-Xiong Dong, Adam Krzyzak, and Ching Y. Suen. Fast SVM Training Algorithm with Decomposition on Very Large Data Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):603–618, April 2005.

[7] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

[8] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. The John Hopkins University Press, third edition, 1996.

[9] Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel Support Vector Machines: The Cascade SVM. *Advances in Neural Information Processing Systems*, 17, 2005.

[10] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In B. Schölkopf and M.K. Warmuth, editors, *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop*, pages 129–143, Stanford University, CA, USA, 2003. Springer Verlag.

[11] T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[12] O. L. Mangasarian and David R. Musicant. Active support vector machine classification. In *NIPS*, pages 577–583, 2000.

[13] J. Platt. Fast training of SVMs using sequential minimal optimization. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1999.

[14] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, first edition edition, 2002.

[15] A. J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, 1998.

[16] John Shawe Taylor and Nello Cristiaini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[17] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *Journal of Machine Learning Research*, (6):363–392, 2005.

[18] Ivor W. Tsang, James T. Kwok, and Kimo T. Lai. Core Vector Regression for Very Large Regression Problems. In *Proceedings of the 22^nd International Conference on Machine Learning*, pages 913–920, 2005.

[19] Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.

[20] R. J. Vanderbei and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. Technical Report SOR-97-21, Princeton University, 1997.

[21] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1999.

[22] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, (29):535–551, 2002.

[23] 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.