Support Vector Machine Solvers

Léon Bottou leon@bottou.org
NEC Labs America, Princeton, NJ 08540, USA
Chih-Jen Lin cjlin@csie.ntu.edu.tw
Department of Computer Science
National Taiwan University, Taipei, Taiwan

Abstract

Considerable efforts have been devoted to the implementation of efficient optimization method for solving the Support Vector Machine dual problem. This document proposes an historical perspective and and in depth review of the algorithmic and computational issues associated with this problem.

1. Introduction

The Support Vector Machine (SVM) algorithm (Cortes and Vapnik, 1995) is probably the most widely used kernel learning algorithm. It achieves relatively robust pattern recognition performance using well established concepts in optimization theory.

Despite this mathematical classicism, the implementation of efficient SVM solvers has diverged from the classical methods of numerical optimization. This divergence is common to virtually all learning algorithms. The numerical optimization literature focuses on the asymptotical performance: how quickly the accuracy of the solution increases with computing time. In the case of learning algorithms, two other factors mitigate the impact of optimization accuracy.

1.1 Three Components of the Generalization Error

The generalization performance of a learning algorithm is indeed limited by three sources of error:

- The *approximation error* measures how well the exact solution can be approximated by a function implementable by our learning system,
- The *estimation error* measures how accurately we can determine the best function implementable by our learning system using a finite training set instead of the unseen testing examples.
- The *optimization error* measures how closely we compute the function that best satisfies whatever information can be exploited in our finite training set.

The estimation error is determined by the number of training example and by the capacity of the family of functions (e.g. Vapnik, 1982). Large family of functions have *smaller approximation errors* but lead to *higher estimation errors*. This compromise has been extensively discussed in the literature. Well designed compromises lead to estimation and approximation errors that scale between the inverse and the inverse square root of the number of examples (Steinwart and Scovel, 2004).

In contrast, the optimization literature discusses algorithms whose error decreases exponentially or faster with the number of iterations. The computing time for each iteration usually grows linearly or quadratically with the number of examples.

It is easy to see that exponentially decreasing optimization error are irrelevant in comparison to the other sources of errors. Therefore it is often desirable to use poorly regarded optimization algorithms that trade asymptotical accuracy for lower iteration complexity. This document describes in depth how SVM solvers have evolved towards this objective.

1.2 Small Scale Learning vs. Large Scale Learning

There is a budget for any given problem. In the case of a learning algorithm, the budget can be a limit on the number of training examples or a limit on the computation time. Which constraint applies can be used to distinguish small-scale learning problems from large-scale learning problems.

- *Small-scale learning problems* are constrained by the number of training examples. The generalization error is dominated by the approximation and estimation errors. The optimization error can be reduced to insignificant levels since the computation time is not limited.

- *Large-scale learning problems* are constrained by the total computation time. Besides adjusting the approximation capacity of the family of function, one can also adjust the number of training examples that a particular optimization algorithm can process within the allowed computing resources. Approximate optimization algorithm can then achieve better generalization error because they process more training examples (Bottou and Le Cun, 2004).

The computation time is always limited in practice. This is why some aspects of large scale learning problems always percolate into small-scale learning problems. One simply looks for algorithms that can quickly reduce the optimization error comfortably below the expected approximation and estimation errors. This has been the main driver for the evolution of SVM solvers.

1.3 Contents

The present document contains an in-depth discussion of optimization algorithms for solving the dual formulation on a single processor. The objective is simply to give the reader a precise understanding of the various computational aspects of sparse kernel machines.

Section 2 reviews the mathematical formulation of SVMs. Section 3 presents the generic optimization problem and performs a first discussion of its algorithmic consequences. Section 5 and 6 discuss various approaches used for solving the SVM dual problem. Section 7 presents the state-of-the-art LIBSVM solver and explains the essential algorithmic choices.

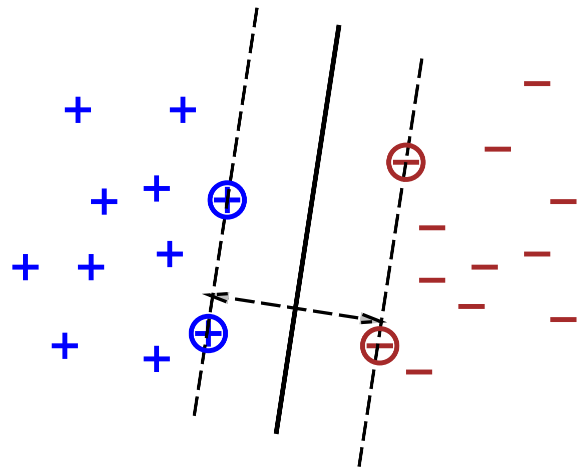

Figure 1: The optimal hyperplane separates positive and negative examples with the maximal margin. The position of the optimal hyperplane is solely determined by the few examples that are closest to the hyperplane (the support vectors.)

2. Support Vector Machines

The earliest pattern regognition systems were linear classifiers (Nilsson, 1965). A pattern $\mathbf{x}$ is given a class $y = \pm 1$ by first transforming the pattern into a feature vector $\Phi(\mathbf{x})$ and taking the sign of a linear discriminant function $\hat{y}(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$.

The hyperplane $\hat{y}(\mathbf{x}) = 0$ defines a decision boundary in the feature space. The problem specific features $\Phi(\mathbf{x})$ is usually chosen by hand. The parameters $\mathbf{w}$ and b are determined by running a learning procedure on a training set $(\mathbf{x}_1, y_1) \cdots (\mathbf{x}_n, y_n)$.

Three additional ideas define the modern SVMs.

2.1 Optimal Hyperplane

The training set is said to be linearly separable when there exists a linear discriminant function whose sign matches the class of all training examples. When a training set is linearly separable there usually is an infinity of separating hyperplanes.

Vapnik and Lerner (1963) propose to choose the separating hyperplane that maximizes the margin, that is to say the hyperplane that leaves as much room as possible between the hyperplane and the closest example. This optimum hyperplane is illustrated in figure 1.

The following optimization problem expresses this choice:

$$\begin{aligned} &\min \ \mathcal{P}(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^2 \\ &\text{subject to } \forall i \quad y_i(\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) \geq 1 \end{aligned} \tag{1}$$

Directly solving this problem is difficult because the constraints are quite complex. The mathematical tool of choice for simplifying this problem is the Lagrangian duality theory (e.g. Bertsekas, 1995). This approach leads to solving the following dual problem:

$$\begin{aligned} &\max\ \mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i\alpha_i\, y_j\alpha_j\ \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j) \\ &\text{subject to} \quad \begin{cases} \forall i \quad \alpha_i \geq 0, \\ \sum_i y_i\alpha_i = 0. \end{cases} \end{aligned} \tag{2}$$

Problem (2) is computationally easier because its constraints are much simpler. The direction $\mathbf{w}^*$ of the optimal hyperplane is then recovered from a solution $\boldsymbol{\alpha}^*$ of the dual optimization problem (2).

$$\mathbf{w}^* = \sum_i \alpha_i^* y_i \Phi(\mathbf{x}_i).$$

Determining the bias b^* becomes a simple one-dimensional problem. The linear discriminant function can then be written as:

$$\hat{y}(\mathbf{x}) = \mathbf{w}^{*\top}\mathbf{x} + b^* = \sum_{i=1}^{n} y_i\alpha_i\ \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b^* \tag{3}$$

Further details are discussed in section 3.

2.2 Kernels

The optimization problem (2) and the linear discriminant function (3) only involve the patterns $\mathbf{x}$ through the computation of dot products in feature space. There is no need to compute the features $\Phi(\mathbf{x})$ when one knows how to compute the dot products directly.

Instead of hand-choosing a feature function $\Phi(\mathbf{x})$, Boser, Guyon, and Vapnik (1992) propose to directly choose a kernel function $K(\mathbf{x}, \mathbf{x}')$ that represents a dot product $\Phi(\mathbf{x})^\top \Phi(\mathbf{x}')$ in some unspecified high dimensional space

The *Reproducing Kernel Hilbert Spaces* theory (Aronszajn, 1944) precisely states which kernel functions correspond to a dot product and which linear spaces are implicitly induced by these kernel functions. For instance, any continuous decision boundary can be implemented using the RBF kernel $K_\gamma(x, y) = e^{-\gamma\|x-y\|^2}$. Although the corresponding feature space has infinite dimension, all computations can be performed without ever computing a feature vector. Complex non-linear classifiers are computed using the linear mathematics of the optimal hyperplanes.

2.3 Soft-Margins

Optimal hyperplanes (section 2.1) are useless when the training set is not linearly separable. Kernels machines (section 2.2) can represent complicated decision boundaries that accomodate any training set. But this is not very wise when the problem is very noisy.

Cortes and Vapnik (1995) show that noisy problems are best addressed by allowing some examples to violate the margin constraints in the primal problem (1). These potential violations are represented using positive slack variables $\boldsymbol{\xi} = (\xi_i \ldots \xi_n)$. An additional parameter

C controls the compromise between large margins and small margin violations.

$$\begin{array}{l} \displaystyle\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \mathcal{P}(\mathbf{w},b,\boldsymbol{\xi}) = \frac{1}{2}\mathbf{w}^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \ \left\{ \begin{array}{ll} \forall i & y_i(\mathbf{w}^\top\Phi(\mathbf{x}_i)+b) \geq 1-\xi_i \\ \forall i & \xi_i \geq 0 \end{array}\right. \end{array} \tag{4}$$

The dual formulation of this soft-margin problem is strikingly similar to the dual forumlation (2) of the optimal hyperplane algorithm. The only change is the appearance of the upper bound C for the coefficients $\boldsymbol{\alpha}$.

$$\begin{array}{l} \max \ \mathcal{D}(\boldsymbol{\alpha}) = \displaystyle\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i\alpha_i\, y_j\alpha_j\, K(\mathbf{x}_i,\mathbf{x}_j) \\ \text{subject to} \ \left\{ \begin{array}{l} \forall i \quad 0 \leq \alpha_i \leq C \\ \sum_i y_i\alpha_i = 0 \end{array}\right. \end{array} \tag{5}$$

2.4 Other SVM variants

A multitude of alternate forms of SVMs have been introduced over the years(see Schölkopf and Smola, 2002, for a review). Typical examples include SVMs for computing regressions (Vapnik, 1995), for solving integral equations (Vapnik, 1995), for estimating the support of a density (Schölkopf et al., 2001), SVMs that use different soft-margin costs (Cortes and Vapnik, 1995), and parameters (Schölkopf et al., 2000; Chang and Lin, 2001b). There are also alternate formulations of the dual problem (Keerthi et al., 1999; Bennett and Bredensteiner, 2000). All these examples reduce to solving quadratic programing problems similar to (5).

3. Duality

This section discusses the properties of the SVM quadratic programming problem. The SVM literature usually establishes basic results using the powerful Karush-Kuhn-Tucker theorem (e.g. Bertsekas, 1995). We prefer instead to give a more detailled account in order to review mathematical facts of great importance for the implementation of SVM solvers.

The rest of this document focuses on solving the Soft-Margin SVM problem (4) using the standard dual formulation (5),

$$\begin{array}{l} \max \ \mathcal{D}(\boldsymbol{\alpha}) = \displaystyle\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i\alpha_i\, y_j\alpha_j\, K_{ij} \\ \text{subject to} \ \left\{ \begin{array}{l} \forall i \quad 0 \leq \alpha_i \leq C, \\ \sum_i y_i\alpha_i = 0, \end{array}\right. \end{array}$$

where $K_{ij} = K(\mathbf{x}_i,\mathbf{x}_j)$ is the matrix of kernel values.

After computing the solution $\boldsymbol{\alpha}^*$, the SVM discriminant function is

$$\hat{y}(\mathbf{x}) = \mathbf{w}^{*\top}\mathbf{x} + b^* = \sum_{i=1}^{n} y_i\alpha_i^* K(\mathbf{x}_i,\mathbf{x}) + b^*. \tag{6}$$

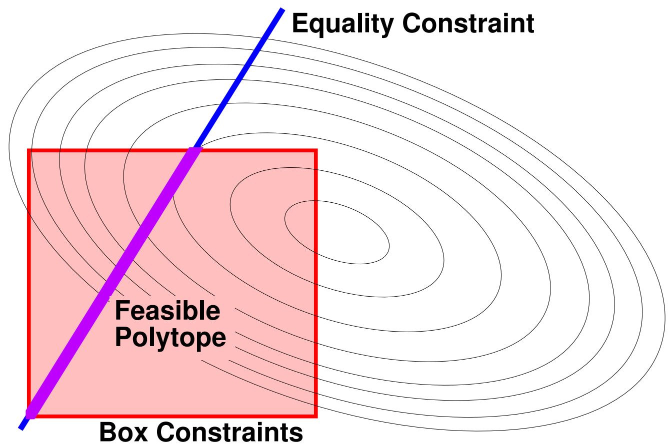

Figure 2: Geometry of the dual SVM problem (5). The box constraints $A_i \leq \alpha_i \leq B_i$ and the equality constraint $\sum \alpha_i = 0$ define the feasible polytope, that is, the domain of the $\boldsymbol{\alpha}$ values that satisfy the constraints.

The optimal bias b^* can be determined by returning to the primal problem, or, more efficiently, using the optimality criterion (11) discussed below.

It is sometimes convenient to rewrite the box constraint $0 \leq \alpha_i \leq C$ as box constraint on the quantity $y_i \alpha_i$:

$$y_i \alpha_i \in [A_i, B_i] = \begin{cases} [\,0, C\,] & \text{if } y_i = +1, \\ [-C, 0\,] & \text{if } y_i = -1. \end{cases} \tag{7}$$

In fact, some SVM solvers, such as SVQP2, optimize variables $\beta_i = y_i \alpha_i$ that are positive or negative depending on y_i. Other SVM solvers, such as LIBSVM (see section 7), optimize the standard dual variables α_i. This document follows the standard convention but uses the constants A_i and B_i defined in (7) when they allow simpler expressions.

3.1 Construction of the Dual Problem

The difficulty of the primal problem (4) lies with the complicated inequality constraints that represent the margin condition. We can represent these constraints using positive Lagrange coefficients $\alpha_i \geq 0$.

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^\top \Phi(\mathbf{x}_i) + b) - 1 + \xi_i\right).$$

The formal dual objective function $\underline{\mathcal{D}}(\boldsymbol{\alpha})$ is defined as

$$\underline{\mathcal{D}}(\boldsymbol{\alpha}) \quad = \quad \min_{\mathbf{w},b,\boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) \quad \text{subject to} \quad \forall i \;\; \xi_i \geq 0. \tag{8}$$

This minimization no longer features the complicated constraints expressed by the Lagrange coefficients. The $\xi_i \geq 0$ constraints have been kept because they are easy enough to handle directly. Standard differential arguments[1] yield the analytical expression of the dual objective function.

$$\underline{\mathcal{D}}(\boldsymbol{\alpha}) = \begin{cases} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i\alpha_i\, y_j\alpha_j\, K_{ij} & \text{if } \sum_i y_i\alpha_i = 0 \text{ and } \forall i \ \alpha_i \leq C, \\ -\infty & \text{otherwise.} \end{cases}$$

The dual problem (5) is the maximization of this expression subject to positivity constraints $\alpha_i \geq 0$. The conditions $\sum_i y_i\alpha_i = 0$ and $\forall i \ \alpha_i \leq C$ appear as constraints in the dual problem because the cases where $\underline{\mathcal{D}}(\boldsymbol{\alpha}) = -\infty$ are not useful for a maximization.

The differentiable function

$$\mathcal{D}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i\alpha_i\, y_j\alpha_j\, K_{ij}$$

coincides with the formal dual function $\underline{\mathcal{D}}(\boldsymbol{\alpha})$ when $\boldsymbol{\alpha}$ satisfies the constraints of the dual problem. By a slight abuse of language, $\mathcal{D}(\boldsymbol{\alpha})$ is also referred to as the dual objective function.

The formal definition (8) of the dual function ensures that the following inequality holds for any $(\mathbf{w}, b, \boldsymbol{\xi})$ satisfying the primal constraints (4) and for any $\boldsymbol{\alpha}$ satisfying the dual constraints (5).

$$\mathcal{D}(\boldsymbol{\alpha}) = \underline{\mathcal{D}}(\boldsymbol{\alpha}) \leq \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) \leq \mathcal{P}(\mathbf{w}, b, \boldsymbol{\xi}) \tag{9}$$

This property is called *weak duality*: the set of values taken by the primal is located above the set of values taken by the dual.

Suppose we can find $\boldsymbol{\alpha}^*$ and $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ such that $\mathcal{D}(\boldsymbol{\alpha}^*) = \mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$. Inequality (9) then implies that both $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ and $\boldsymbol{\alpha}^*$ are solutions of the primal and dual problems. Convex optimization problems with linear constraints are known to have such solutions. This is called *strong duality*.

Our goal is now to find such a solution for the SVM problem.

3.2 Optimality Criteria

Let $\boldsymbol{\alpha}^* = (\alpha_1^* \dots \alpha_n^*)$ be solution of the dual problem (5). Obviously $\boldsymbol{\alpha}^*$ satisfies the dual constraints. Let $\mathbf{g}^* = (g_1^* \dots g_n^*)$ be the derivatives of the dual objective function in $\boldsymbol{\alpha}^*$.

$$g_i^* = \frac{\partial \mathcal{D}(\boldsymbol{\alpha}^*)}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{n} y_j \alpha_j^*\, K_{ij} \tag{10}$$

Consider a pair of subscripts (i, j) such that $y_i\alpha_i^* < B_i$ and $A_j < y_j\alpha_j^*$. The constants A_i and B_j were defined in (7).

Define $\boldsymbol{\alpha}^\varepsilon = (\alpha_1^\varepsilon \dots \alpha_n^\varepsilon) \in \mathbb{R}^n$ as

$$\alpha_k^\varepsilon = \alpha_k^* \ + \begin{cases} +\varepsilon\, y_k & \text{if } k = i, \\ -\varepsilon\, y_k & \text{if } k = j, \\ 0 & \text{otherwise.} \end{cases}$$

1. This simple derivation is relatively lenghty because many cases must be considered.

The point $\boldsymbol{\alpha}^\varepsilon$ clearly satisfies the constraints if ε is positive and sufficiently small. Therefore $\mathcal{D}(\boldsymbol{\alpha}^\varepsilon) \leq \mathcal{D}(\boldsymbol{\alpha}^*)$ because $\boldsymbol{\alpha}^*$ is solution of the dual problem (5). On the other hand, we can write the first order expansion

$$\mathcal{D}(\boldsymbol{\alpha}^\varepsilon) - \mathcal{D}(\boldsymbol{\alpha}^*) = \varepsilon\,(y_i g_i^* - y_j g_j^*) + o(\varepsilon).$$

Therefore the difference $y_i g_i^* - y_j g_j^*$ is necessarily negative. Since this holds for all pairs (i, j) such that $y_i\alpha_i^* < B_i$ and $A_j < y_j\alpha_j^*$, we can write the following *necessary optimality criterion*.

$$\exists \rho \in \mathbb{R} \text{ such that } \max_{i \in I_{\text{up}}} y_i g_i^* \;\leq\; \rho \;\leq\; \min_{j \in I_{\text{down}}} y_j g_j^* \tag{11}$$

where $I_{\text{up}} = \{\, i \mid y_i\alpha_i < B_i \,\}$ and $I_{\text{down}} = \{\, j \mid y_j\alpha_j > A_j \,\}$.

Usually, there are some coefficients α_k^* strictly between their lower and upper bounds. Since the corresponding $y_k g_k^*$ appear on both sides of the inequality, this common situation leaves only one possible value for ρ.

We can rewrite (11) as

$$\exists \rho \in \mathbb{R} \text{ such that } \forall k, \quad \begin{cases} \text{if } \; y_k g_k^* > \rho \quad \text{then } \; y_k\alpha_k^* = B_k, \\ \text{if } \; y_k g_k^* < \rho \quad \text{then } \; y_k\alpha_k^* = A_k, \end{cases} \tag{12}$$

or, equivalently, as

$$\exists \rho \in \mathbb{R} \text{ such that } \forall k, \quad \begin{cases} \text{if } \; g_k^* > y_k\rho \quad \text{then } \; \alpha_k^* = C, \\ \text{if } \; g_k^* < y_k\rho \quad \text{then } \; \alpha_k^* = 0. \end{cases} \tag{13}$$

Let us now pick

$$\mathbf{w}^* = \sum_k y_k\alpha_k^*\Phi(\mathbf{x}_k), \quad b^* = \rho, \quad \text{and} \quad \xi_k^* = \max\{0, g_k^* - y_k\rho\}. \tag{14}$$

These values satisfy the constraints of the primal problem (4). A short derivation using (10) then gives

$$\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) - \mathcal{D}(\boldsymbol{\alpha}^*) = C\sum_{k=1}^{n}\xi_k^* - \sum_{k=1}^{n}\alpha_k^* g_k^* = \sum_{k=1}^{n}\left(C\xi_k^* - \alpha_k^* g_k^*\right).$$

We can compute the quantity $C\xi_k^* - \alpha_k^* g_k^*$ using (14) and (13). The relation $C\xi_k^* - \alpha_k^* g_k^* = -y_k\alpha_k^*\rho$ holds regardless of whether g_k^* is less than, equal to, or greater than $y_k\rho$. Therefore

$$\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) - \mathcal{D}(\boldsymbol{\alpha}^*) = -\rho\sum_{i=1}^{n} y_i\alpha_i^* = 0. \tag{15}$$

This strong duality has two implications:

- Our choice for $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ minimizes the primal problem because inequality (9) states that the primal function $\mathcal{P}(\mathbf{w}, b, \boldsymbol{\xi})$ cannot take values smaller than $\mathcal{D}(\boldsymbol{\alpha}^*) = \mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$.

- The optimality criteria (11—13) are in fact *necessary and sufficient*. Assume that one of these criteria holds. Choosing $(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ as shown above yields the strong duality property $\mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*) = \mathcal{D}(\boldsymbol{\alpha}^*)$. Therefore $\boldsymbol{\alpha}^*$ maximizes the dual because inequality (9) states that the dual function $\mathcal{D}(\boldsymbol{\alpha})$ cannot take values greater than $\mathcal{D}(\boldsymbol{\alpha}^*) = \mathcal{P}(\mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$.

The necessary and sufficient criterion (11) is particularly useful for SVM solvers (Keerthi et al., 2001).

4. Sparsity

The vector $\boldsymbol{\alpha}^*$ solution of the dual problem (5) contains many zeroes. The SVM discriminant function $\hat{y}(\mathbf{x}) = \sum_i y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$ is expressed using a sparse subset of the training examples called the Support Vectors (SVs).

From the point of view of the algorithm designer, sparsity provides the opportunity to considerably reduce the memory and time requirements of SVM solvers. On the other hand, it is difficult to combine these gains with those brought by algorithms that handle large segments of the kernel matrix at once. Algorithms for sparse kernel machines and algorithms for other kernel machines have evolved differently.

4.1 Support Vectors

The optimality criterion (13) characterizes which training examples become support vectors. Recalling (10) and (14)

$$g_k - y_k\rho = 1 - y_k \sum_{i=1}^{n} y_i \alpha_i K_{ik} - y_k b^* = 1 - y_k\, \hat{y}(\mathbf{x}_k). \tag{16}$$

Replacing in (13) gives

$$\begin{cases} \text{if} \;\; y_k\, \hat{y}(\mathbf{x}_k) < 1 \;\; \text{then} \;\; \alpha_k = C, \\ \text{if} \;\; y_k\, \hat{y}(\mathbf{x}_k) > 1 \;\; \text{then} \;\; \alpha_k = 0. \end{cases} \tag{17}$$

This result splits the training examples in three categories[2]:

- Examples $(\mathbf{x}_k, y_k)$ such that $y_k\, \hat{y}(\mathbf{x}_k) > 1$ are not support vectors. They do not appear in the discriminant function because $\alpha_k = 0$.

- Examples $(\mathbf{x}_k, y_k)$ such that $y_k\, \hat{y}(\mathbf{x}_k) < 1$ are called *bounded support vectors* because they activate the inequality constraint $\alpha_k \le C$. They appear in the discriminant function with coefficient $\alpha_k = C$.

- Examples $(\mathbf{x}_k, y_k)$ such that $y_k\, \hat{y}(\mathbf{x}_k) = 1$ are called *free support vectors*. They appear in the discriminant function with a coefficient in range $[0, C]$.

2. Some texts call an example k a *free support vector* when $0 < \alpha_k < C$ and a *bounded support vector* when $\alpha_k = C$. This is almost the same thing as the above definition.

Let $\mathcal{B}$ represent the best error achievable by a linear decision boundary in the chosen feature space for the problem at hand. When the training set size n becomes large, one can expect about $\mathcal{B}n$ misclassified training examples, that is to say $y_k \, \hat{y} \leq 0$. All these misclassified examples[3] are bounded support vectors. Therefore the number of bounded support vectors scales at least linearly with the number of examples.

When the hyperparameter C follows the right scaling laws, Steinwart (2004) has shown that the total number of support vectors is asymptotically equivalent to $2\mathcal{B}n$. Noisy problems do not lead to very sparse SVMs. Collobert et al. (2006) explore ways to improve sparsity in such cases.

4.2 Complexity

There are two intuitive lower bounds on the computational cost of any algorithm that solves the SVM problem for arbitrary kernel matrices K_{ij}.

- Suppose that an oracle reveals which examples are not support vectors ($\alpha_i = 0$), and which examples are bounded support vectors ($\alpha_i = C$). The coefficients of the R remaining free support vectors are determined by a system of R linear equations representing the derivatives of the objective function. Their calculation amounts to solving such a system. This typically requires a number of operations proportional to R^3.

- Simply verifying that a vector $\boldsymbol{\alpha}$ is a solution of the SVM problem involves computing the gradient $\mathbf{g}$ of the dual and checking the optimality conditions (11). With n examples and S support vectors, this requires a number of operations proportional to $n\,S$.

Few support vectors reach the upper bound C when it gets large. The cost is then dominated by the $R^3 \approx S^3$. Otherwise the term $n\,S$ is usually larger. The final number of support vectors therefore is the critical component of the computational cost of solving the dual problem.

Since the asymptotical number of support vectors grows linearly with the number of examples, the computational cost of solving the SVM problem has both a quadratic and a cubic component. It grows at least like n^2 when C is small and n^3 when C gets large. Empirical evidence shows that modern SVM solvers come close to these scaling laws.

4.3 Computation of the kernel values

Although computing the n^2 components of the kernel matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ seems to be a simple quadratic affair, a more detailled analysis reveals a much more complicated picture.

- *Computing kernels is expensive* — Computing each kernel value usually involves the manipulation of sizeable chunks of data representing the patterns. Images have thousands of pixels (e.g. Boser et al., 1992). Documents have thousands of words (e.g.

3. A sharp eyed reader might notice that a discriminant function dominated by these $\mathcal{B}n$ misclassified examples would have the wrong polarity. About $\mathcal{B}n$ additional well classified support vectors are needed to correct the orientation of $\mathbf{w}$.

Joachims, 1999). In practice, computing kernel values often accounts for more than half the total computing time.

- *Computing the full kernel matrix is wasteful* — The expression of the gradient (10) only depend on kernel values K_{ij} that involve at least one support vector (the other kernel values are multiplied by zero). All three optimality criteria (11—13) can be verified with these kernel values only. The remaining kernel values have no impact on the solution. To determine which kernel values are actually needed, efficient SVM solvers compute no more than 15% to 50% additional kernel values (Graf, personal communication). The total training time is usually smaller than the time needed to compute the whole kernel matrix. SVM programs that precompute the full kernel matrix are not competitive.

- *The kernel matrix does not fit in memory* — When the number of examples grows, the kernel matrix K_{ij} becomes very large and cannot be stored in memory. Kernel values must be computed on the fly or retrieved from a cache of often accessed values. The kernel cache hit rate becomes a major factor of the training time.

These issues only appear as "constant factors" in the asymptotic complexity of solving the SVM problem. But practice is dominated by these constant factors.

5. Early SVM Algorithms

The SVM solution is the optimum of a well defined convex optimization problem. Since this optimum does not depend on the details of its calculation, the choice of a particular optimization algorithm can be made on the sole basis[4] of its computational requirements. High performance optimization packages, such as MINOS, and LOQO (Vanderbei, 1994) can efficiently handle very varied quadratic programming workloads. There are however differences between the SVM problem and the usual quadratic programming benchmarks.

- Quadratic optimization packages were often designed to take advantage of sparsity in the quadratic part of the objective function. Unfortunately, the SVM kernel matrix is rarely sparse: sparsity occurs in the *solution* of the SVM problem.

- The specification of a SVM problem rarely fits in memory. Kernel matrix coefficient must be cached or computed on the fly. As explained in section 4.3, vast speedups are achieved by accessing the kernel matrix coefficients carefully.

- Generic optimization packages sometimes make extra work to locate the optimum with high accuracy. As explained in section 1.1, the accuracy requirements of a learning problem are unusually low.

Using standard optimization packages for medium size SVM problems is not straightforward (section 6). In fact, all early SVM results were obtained using ad-hoc algorithms.

These algorithms borrow two ideas from the optimal hyperplane literature (Vapnik, 1982; Vapnik et al., 1984). The optimization is achieved by performing successive applications of a very simple *direction search*. Meanwhile, *iterative chunking* leverages the sparsity

4. Defining a learning algorithm for a multilayer network is a more difficult exercise.

of the solution. We first describe these two ideas and then present the *modified gradient projection* algorithm.

5.1 Iterative Chunking

The first challenge was of course to solve a quadratic programming problem whose specification does not fit in the memory of a computer. Fortunately, the quadratic programming problem (2) has often a sparse solution. If we knew in advance which examples are support vectors, we could solve the problem restricted to the support vectors and verify a posteriori that this solution satisfies the optimality criterion for all examples.

Let us solve problem (2) restricted to a small subset of training examples called the *working set*. Because learning algorithms are designed to generalize well, we can hope that the resulting classifier performs honorably on the remaining training examples. Many will readily fulfill the margin condition $y_i\,\hat{y}(\mathbf{x}_i) \geq 1$. Training examples that violate the margin condition are good support vector candidates. We can add some of them to the working set, solve the quadratic programming problem restricted to the new working set, and repeat the procedure until the margin constraints are satisfied for all examples.

This procedure is not a general purpose optimization technique. It works efficiently *because* the underlying problem is a learning problem. It reduces the large problem to a sequence of smaller optimization problems.

5.2 Direction Search

The optimization of the dual problem is then achieved by performing *direction searches* along well chosen successive directions.

Assume we are given a starting point $\boldsymbol{\alpha}$ that satisfies the constraints of the quadratic optimization problem (5). We say that a direction $\mathbf{u} = (u_1 \ldots u_n)$ is a *feasible direction* if we can slightly move the point $\boldsymbol{\alpha}$ along direction $\mathbf{u}$ without violating the constraints.

Formally, we consider the set Λ of all coefficients $\lambda \geq 0$ such that the point $\boldsymbol{\alpha} + \lambda\mathbf{u}$ satisfies the constraints. This set always contains 0. We say that u is a feasible direction if Λ is not the singleton $\{0\}$. Because the feasible polytope is convex and bounded (see figure 2), the set Λ is a bounded interval of the form $[0,\ \lambda^{\max}]$.

We seek to maximizes the dual optimization problem (5) restricted to the half line $\{\boldsymbol{\alpha} + \lambda\mathbf{u},\ \lambda \in \Lambda\}$. This is expressed by the simple optimization problem

$$\lambda^* = \arg\max_{\lambda \in \Lambda} \mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u}).$$

Figure 3 represent values of the $\mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u})$ as a function of λ. The set Λ is materialized by the shaded areas. Since the dual objective function is quadratic, $\mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u})$ is shaped like a parabola. The location of its maximum λ^+ is easily computed using Newton's formula

$$\lambda^+ \;=\; \frac{\left.\frac{\partial \mathcal{D}(\boldsymbol{\alpha}+\lambda\mathbf{u})}{\partial \lambda}\right|_{\lambda=0}}{\left.\frac{\partial^2 \mathcal{D}(\boldsymbol{\alpha}+\lambda\mathbf{u})}{\partial \lambda^2}\right|_{\lambda=0}} \;=\; \frac{\mathbf{g}^\top \mathbf{u}}{\mathbf{u}^\top \mathbf{H}\,\mathbf{u}}$$

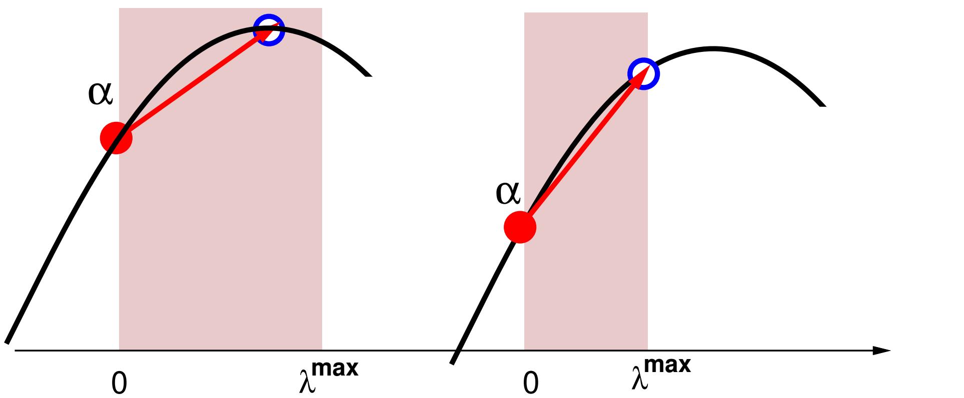

Figure 3: Given a starting point $\boldsymbol{\alpha}$ and a feasible direction $\mathbf{u}$, the direction search maximizes the function $f(\lambda) = \mathcal{D}(\boldsymbol{\alpha} + \lambda\mathbf{u})$ for $\lambda \geq 0$ and $\boldsymbol{\alpha} + \lambda\mathbf{u}$ satisfying the constraints of (5).

where vector $\mathbf{g}$ and matrix $\mathbf{H}$ are the gradient and the Hessian of the dual objective function $\mathcal{D}(\boldsymbol{\alpha})$,

$$g_i = 1 - y_i \sum_j y_j \alpha_j \, K_{ij} \quad \text{and} \quad H_{ij} = y_i y_j K_{ij}.$$

The solution of our problem is then the projection of the maximum λ^+ into the interval $\Lambda = [0, \lambda^{\max}]$,

$$\lambda^* = \max\left(0,\ \min\left(\lambda^{\max},\ \frac{\mathbf{g}^\top\mathbf{u}}{\mathbf{u}^\top\mathbf{H}\,\mathbf{u}}\right)\right). \tag{18}$$

This formula is the basis for a family of optimization algorithms. Starting from an initial feasible point, each iteration selects a suitable feasible direction and applies the direction search formula (18) until reaching the maximum.

5.3 Modified Gradient Projection

Several conditions restrict the choice of the successive search directions $\mathbf{u}$.

- The equality constraint (5) restricts $\mathbf{u}$ to the linear subspace $\sum_i y_i u_i = 0$.
- The box constraints (5) become sign constraints on certain coefficients of $\mathbf{u}$. Coefficient u_i must be non-negative if $\alpha_i = 0$ and non-positive if $\alpha_i = C$.
- The search direction must be an ascent direction to ensure that the search makes progress towards the solution. Direction $\mathbf{u}$ must belong to the half space $\mathbf{g}^\top\mathbf{u} > 0$ where $\mathbf{g}$ is the gradient of the dual objective function.
- Faster convergence rates are achieved when successive search directions are conjugate, that is, $\mathbf{u}^\top\mathbf{H}\mathbf{u}' = 0$, where $\mathbf{H}$ is the Hessian and $\mathbf{u}'$ is the last search direction. Similar

to the conjugate gradient algorithm (Golub and Van Loan, 1996), this condition is more conveniently expressed as $\mathbf{u}^\top(\mathbf{g} - \mathbf{g}') = 0$ where $\mathbf{g}'$ represents the gradient of the dual before the previous search (along direction $\mathbf{u}'$).

Finding a search direction that simultaneously satisfies all these restrictions is far from simple. To work around this difficulty, the optimal hyperplane algorithms of the seventies (see Vapnik, 1982, Addendum I, section 4) exploit a reparametrization of the dual problem that squares the number of variables to optimize. This algorithm (Vapnik et al., 1984) was in fact used by Boser et al. (1992) to obtain the first experimental results for SVMs.

The modified gradient projection technique addresses the direction selection problem more directly. This technique was employed at AT&T Bell Laboratories to obtain most early SVM results (Bottou et al., 1994; Cortes and Vapnik, 1995).

Algorithm 5.1 Modified Gradient Projection

1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
2: $\mathcal{B} \leftarrow \emptyset$
3: **while** there are examples violating the optimality condition, **do**
4: Add some violating examples to working set $\mathcal{B}$.
5: **loop**
6: $\forall k \in \mathcal{B} \quad g_k \leftarrow \partial \mathcal{D}(\boldsymbol{\alpha})/\partial \alpha_k$ using (10)
7: **repeat** ▷ Projection–eviction loop
8: $\forall k \notin \mathcal{B} \quad u_k \leftarrow 0$
9: $\rho \leftarrow \text{mean}\{\, y_k g_k \mid k \in \mathcal{B} \,\}$
10: $\forall k \in \mathcal{B} \quad u_k \leftarrow g_k - y_k \rho$ ▷ Ensure $\sum_i y_i u_i = 0$
11: $\mathcal{B} \leftarrow \mathcal{B} \setminus \{\, k \in \mathcal{B} \mid (u_k > 0 \ \textbf{and} \ \alpha_k = C) \ \textbf{or} \ (u_k < 0 \ \textbf{and} \ \alpha_k = 0) \,\}$
12: **until** $\mathcal{B}$ stops changing
13: **if** $\mathbf{u} = \mathbf{0}$ **exit loop**
14: Compute λ^* using (18) ▷ Direction search
15: $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \lambda^* \mathbf{u}$
16: **end loop**
17: **end while**

The four conditions on the search direction would be much simpler without the box constraints. It would then be sufficient to project the gradient $\mathbf{g}$ on the linear subspace described by the equality constraint and the conjugation condition. Modified gradient projection recovers this situation by leveraging the chunking procedure. Examples that activate the box constraints are simply evicted from the working set!

Algorithm 5.1 illustrates this procedure. For simplicity we omit the conjugation condition. The gradient $\mathbf{g}$ is simply projected (lines 8–10) on the linear subspace corresponding to the inequality constraint. The resulting search direction $\mathbf{u}$ might drive some coefficients outside the box constraints. When this is the case, we remove these coefficients from the working set (line 11) and return to the projection stage.

Modified gradient projection spends most of the computing time searching for training examples violating the optimality conditions. Modern solvers simplify this step by keeping the gradient vector $\mathbf{g}$ up-to-date and leveraging the optimality condition (11).

6. The Decomposition Method

Quadratic programming optimization methods achieved considerable progress between the invention of the optimal hyperplanes in the 60s and the definition of the contemportary SVM in the 90s. It was widely believed that superior performance could be achieved using state-of-the-art generic quadratic programming solvers such as MINOS or LOQO.

Unfortunately the design of these solvers assume that the full kernel matrix is readily available. As explained in section 4.3, computing the full kernel matrix is costly and unneeded. Decomposition methods (Osuna et al., 1997; Saunders et al., 1998; Joachims, 1999) were designed to overcome this difficulty. They address the full scale dual problem (5) by solving a sequence of smaller quadratic programming subproblems.

Iterative chunking (section 5.1) is a particular case of the decomposition method. Modified gradient projection (section 5.3) and shrinking (section 7.3) are slightly different because the working set is dynamically modified during the subproblem optimization.

6.1 General Decomposition

Instead of updating all the coefficients of vector $\boldsymbol{\alpha}$, each iteration of the decomposition method optimizes a subset of coefficients α_i, $i \in \mathcal{B}$ and leaves the remaining coefficients α_j, $j \notin \mathcal{B}$ unchanged.

Starting from a coefficient vector $\boldsymbol{\alpha}$ we can compute a new coefficient vector $\boldsymbol{\alpha}'$ by adding an additional constraint to the dual problem (5) that represents the frozen coefficients:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}'} \ \mathcal{D}(\boldsymbol{\alpha}') = \sum_{i=1}^{n} \alpha'_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i \alpha'_i \, y_j \alpha'_j \, K(\mathbf{x}_i, \mathbf{x}_j) \\ &\text{subject to} \ \begin{cases} \forall i \notin \mathcal{B} \quad \alpha'_i = \alpha_i, \\ \forall i \in \mathcal{B} \quad 0 \leq \alpha'_i \leq C, \\ \sum_i y_i \alpha'_i = 0. \end{cases} \end{aligned} \tag{19}$$

We can rewrite (19) as a quadratic programming problem in variables α_i, $i \in \mathcal{B}$ and remove the additive terms that do not involve the optimization variables $\boldsymbol{\alpha}'$:

$$\begin{gathered} \max_{\boldsymbol{\alpha}'} \ \sum_{i \in \mathcal{B}} \alpha'_i \left(1 - y_i \sum_{j \notin \mathcal{B}} y_j \alpha_j \, K_{ij} \right) - \frac{1}{2} \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}} y_i \alpha'_i \, y_j \alpha'_j \, K_{ij} \\ \text{subject to} \ \ \forall i \in \mathcal{B} \ \ 0 \leq \alpha'_i \leq C \ \ \text{and} \ \ \sum_{i \in \mathcal{B}} y_i \alpha'_i = - \sum_{j \notin \mathcal{B}} y_j \alpha_j. \end{gathered} \tag{20}$$

Algorithm 6.1 illustrates a typical application of the decomposition method. It stores a coefficient vector $\boldsymbol{\alpha} = (\alpha_1 \dots \alpha_n)$ and the corresponding gradient vector $\mathbf{g} = (g_1 \dots g_n)$. This algorithm stops when it achieves[5] the optimality criterion (11). Each iteration selects a working set and solves the corresponding subproblem using any suitable optimization algorithm. Then it efficiently updates the gradient by evaluating the difference between old gradient and new gradient and updates the coefficients. All these operations can be achieved using only the kernel matrix rows whose indices are in $\mathcal{B}$. This careful use of the kernel values saves both time and memory as explained in section 4.3

5. With some predefined accuracy, in practice. See section 7.1.

Algorithm 6.1 Decomposition method

$\forall k \in \{1 \dots n\} \quad \alpha_k \leftarrow 0$ ▷ Initial coefficients
$\forall k \in \{1 \dots n\} \quad g_k \leftarrow 1$ ▷ Initial gradient
loop
  $G^{\max} \leftarrow \max_i y_i g_i$ subject to $y_i \alpha_i < B_i$
  $G^{\min} \leftarrow \min_j y_j g_j$ subject to $A_j < y_j \alpha_j$
  if $G^{\max} \leq G^{\min}$ **stop.** ▷ Optimality criterion (11)
  Select a working set $\mathcal{B} \subset \{1 \dots n\}$ ▷ See text

$$\boldsymbol{\alpha}' \leftarrow \arg\max_{\boldsymbol{\alpha}'} \sum_{i \in \mathcal{B}} \alpha'_i \left(1 - y_i \sum_{j \notin \mathcal{B}} y_j \alpha_j \, K_{ij} \right) - \frac{1}{2} \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}} y_i \alpha'_i \, y_j \alpha'_j \, K_{ij}$$
$$\text{subject to } \forall i \in \mathcal{B} \quad 0 \leq \alpha'_i \leq C \quad \text{and} \quad \sum_{i \in \mathcal{B}} y_i \alpha'_i = -\sum_{j \notin \mathcal{B}} y_j \alpha_j$$

  $\forall k \in \{1 \dots n\} \quad g_k \leftarrow g_k - y_k \sum_{i \in \mathcal{B}} y_i (\alpha'_i - \alpha_i) K_{ik}$ ▷ Update gradient
  $\forall i \in \mathcal{B} \quad \alpha_i \leftarrow \alpha'_i$ ▷ Update coefficients
end loop

The definition of (19) ensures that $\mathcal{D}(\boldsymbol{\alpha}') \geq \mathcal{D}(\boldsymbol{\alpha})$. The question is then to define a *working set selection scheme* that ensures that the increasing values of the dual reach the maximum.

- Like the modified gradient projection algorithm (section 5.3) we can construct the working sets by eliminating coefficients α_i when they hit their lower or upper bound with sufficient strength.
- Joachims (1999) proposes a systematic approach. Working sets are constructed by selecting a predefined number of coefficients responsible for the most severe violation of the optimality criterion (11). Section 7.2.2 illustrates this idea in the case of working sets of size two.

Decomposition methods are related to *block coordinate descent* in bound-constrained optimization (Bertsekas, 1995). However the equality constraint $\sum_i y_i \alpha_i = 0$ makes the convergence proofs more delicate. With suitable working set selection schemes, asymptotic convergence results state that any limit point of the infinite sequence generated by the algorithm is an optimal solution (e.g. Chang et al., 2000; Lin, 2001; Hush and Scovel, 2003; List and Simon, 2004; Palagi and Sciandrone, 2005). Finite termination results state that the algorithm stops with a predefined accuracy after a finite time (e.g. Lin, 2002).

6.2 Decomposition in Practice

Different optimization algorithms can be used for solving the quadratic programming subproblems (20). The Royal Holloway SVM package was designed to compare various subproblem solvers (Saunders et al., 1998). Advanced solvers such as MINOS or LOQO have relatively long setup times. Their higher asymptotic convergence rate is not very useful because a coarse solution is often sufficient for learning applications. Faster results were often achieved using SVQP, a simplified version of the modified gradient projection algorithm (section 5.3): the outer loop of algorithm 5.1 was simply replaced by the main loop of algorithm 6.1.

Experiments with various working set sizes also gave surprising results. The best learning times were often achieved using working sets containing very few examples (Joachims, 1999; Platt, 1999; Collobert, 2004).

6.3 Sequential Minimal Optimization

Platt (1999) proposes to always use the smallest possible working set size, that is, two elements. This choice dramatically simplifies the decomposition method.

- Each successive quadratic programming subproblem has two variables. The equality constraint makes this a one dimensional optimization problem. A single direction search (section 5.2) is sufficient to compute the solution.
- The computation of the Newton step (18) is very fast because the direction $\mathbf{u}$ contains only two non zero coefficients.
- The asymptotic convergence and finite termination properties of this particular case of the decomposition method are very well understood (e.g. Keerthi and Gilbert, 2002; Takahashi and Nishi, 2005; Chen et al., 2006; Hush et al., 2006).

The Sequential Minimal Optimization algorithm 6.2 selects working sets using the maximum violating pair scheme. Working set selection schemes are discussed more thoroughly in section 7.2. Each subproblem is solved by performing a search along a direction $\mathbf{u}$ containing only two non zero coefficients: $u_i = y_i$ and $u_j = -y_j$. The algorithm is otherwise similar to algorithm 6.1. Implementation issues are discussed more thoroughly in section 7.

Algorithm 6.2 SMO with Maximum Violating Pair working set selection

$\forall k \in \{1 \dots n\} \quad \alpha_k \leftarrow 0$ ▷ Initial coefficients
$\forall k \in \{1 \dots n\} \quad g_k \leftarrow 1$ ▷ Initial gradient
loop
  $i \leftarrow \arg\max_i y_i g_i$ subject to $y_i \alpha_i < B_i$
  $j \leftarrow \arg\min_j y_j g_j$ subject to $A_j < y_j \alpha_j$ ▷ Maximal violating pair
  if $y_i g_i \leq y_j g_j$ **stop.** ▷ Optimality criterion (11)
  $\lambda \leftarrow \min \left\{ B_i - y_i \alpha_i,\ y_j \alpha_j - A_j,\ \dfrac{y_i g_i - y_j g_j}{K_{ii} + K_{jj} - 2K_{ij}} \right\}$ ▷ Direction search
  $\forall k \in \{1 \dots n\} \quad g_k \leftarrow g_k - \lambda y_k K_{ik} + \lambda y_k K_{jk}$ ▷ Update gradient
  $\alpha_i \leftarrow \alpha_i + y_i \lambda \qquad \alpha_j \leftarrow \alpha_j - y_j \lambda$ ▷ Update coefficients
end loop

The practical efficiency of the SMO algorithm is very compelling. It is easier to program and often runs as fast as careful implementations of the full fledged decomposition method.

SMO prefers sparse search directions over conjugate search directions (section 5.3). This choice sometimes penalizes SMO when the soft-margin parameter C is large. Reducing C

often corrects the problem with little change in generalization performance. Second order working set selection (section 7.2.3) also helps.

The most intensive part of each iteration of algorithm 6.2 is the update of the gradient on line 8. These operations require the computation of two full rows of the kernel matrix. The *shrinking* technique (Joachims, 1999) reduces this calculation. This is very similar to the decomposition method (algorithm 6.1) where the working sets are updated on the fly, and where successive subproblems are solved using SMO (algorithm 6.2). See section 7.3 for details.

7. A Case Study: LIBSVM

This section explains the algorithmic choices and discusses the implementation details of a modern SVM solver. The LIBSVM solver (version 2.82) (Chang and Lin, 2001a) is based on the SMO algorithm, but relies on a more advanced working set selection scheme. After discussing the stopping criteria and the working set selection, we present the shrinking heuristics and their impact on the design of the cache of kernel values.

This section discusses the dual maximization problem

$$\max\ \mathcal{D}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i\alpha_i\, y_j\alpha_j\, K(\mathbf{x}_i, \mathbf{x}_j)$$
$$\text{subject to } \begin{cases} \forall i \quad 0 \leq \alpha_i \leq C, \\ \sum_i y_i\alpha_i = 0, \end{cases}$$

Let $\mathbf{g} = (g_1 \dots g_n)$ be the gradient of the dual objective function,

$$g_i = \frac{\partial \mathcal{D}(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_{k=1}^{n} y_k\alpha_k\, K_{ik}.$$

Readers studying the source code should be aware that LIBSVM was in fact written as the minimization of $-\mathcal{D}(\boldsymbol{\alpha})$ instead of the maximization of $\mathcal{D}(\boldsymbol{\alpha})$. The variable `G[i]` in the source code contains $-g_i$ instead of g_i.

7.1 Stopping Criterion

The LIBSVM algorithm stops when the optimality criterion (11) is reached with a predefined accuracy ϵ,

$$\max_{i \in I_{\text{up}}} y_i g_i - \min_{j \in I_{\text{down}}} y_j g_j < \epsilon, \tag{21}$$

where $I_{\text{up}} = \{\, i \mid y_i\alpha_i < B_i \,\}$ and $I_{\text{down}} = \{\, j \mid y_j\alpha_j > A_j \,\}$ as in (11).

Theoretical results establish that SMO achieves the stopping criterion (21) after a finite time. These finite termination results depend on the chosen working set selection scheme. See (Keerthi and Gilbert, 2002; Takahashi and Nishi, 2005; Chen et al., 2006; Bordes et al., 2005, appendix).

Schölkopf and Smola (2002, section 10.1.1) propose an alternate stopping criterion based on the duality gap (15). This criterion is more sensitive to the value of C and requires additional computation. On the other hand, Hush et al. (2006) propose a setup with theoretical guarantees on the termination time.

7.2 Working set selection

There are many ways to select the pair of indices (i, j) representing the working set for each iteration of the SMO algorithm.

Assume a given iteration starts with coefficient vector $\boldsymbol{\alpha}$. Only two coefficients of the solution $\boldsymbol{\alpha}'$ of the SMO subproblem differs from the coefficients of $\boldsymbol{\alpha}$. Therefore $\boldsymbol{\alpha}' = \boldsymbol{\alpha} + \lambda \mathbf{u}$ where the direction $\mathbf{u}$ has only two non zero coefficients. The equality constraint further implies that $\sum_k y_k u_k = 0$. Therefore it is sufficient to consider directions $\mathbf{u}^{ij} = (u_1^{ij} \ldots u_n^{ij})$ such that

$$u_k^{ij} = \left\{ \begin{array}{rl} y_i & \text{if } k = i, \\ -y_j & \text{if } k = j, \\ 0 & \text{otherwise.} \end{array} \right. \tag{22}$$

The subproblem optimization then requires a single direction search (18) along direction $\mathbf{u}^{ij}$ (for positive λ) or direction $-\mathbf{u}^{ij} = \mathbf{u}^{ji}$ (for negative λ). Working set selection for the SMO algorithm then reduces to the selection of a search direction of the form (22). Since we need a feasible direction, we can further require that $i \in I_{\text{up}}$ and $j \in I_{\text{down}}$.

7.2.1 Maximal Gain Working Set Selection

Let $\mathcal{U} = \{ \mathbf{u}^{ij} \mid i \in I_{\text{up}},\ j \in I_{\text{down}} \}$ be the set of the potential search directions. The most effective direction for each iteration should be the direction that maximizes the increase of the dual objective function:

$$\begin{array}{l} \mathbf{u}^* = \displaystyle\arg\max_{\mathbf{u}^{ij} \in \mathcal{U}} \ \max_{0 \leq \lambda} \ \mathcal{D}(\boldsymbol{\alpha} + \lambda \mathbf{u}^{ij}) - \mathcal{D}(\boldsymbol{\alpha}) \\ \qquad \text{subject to } \left\{ \begin{array}{l} y_i \alpha_i + \lambda \leq B_i, \\ y_j \alpha_j - \lambda \geq A_j. \end{array} \right. \end{array} \tag{23}$$

Unfortunately the search of the best direction $\mathbf{u}^{ij}$ requires iterating over the $n(n-1)$ possible pairs of indices. The maximization of λ then amounts to performing a direction search. These repeated direction searches would virtually access all the kernel matrix during each SMO iteration. This is not acceptable for a fast algorithm.

Although maximal gain working set selection may reduce the number of iterations, it makes each iteration very slow. Practical working set selection schemes need to simplify problem (23) in order to achieve a good compromise between the number of iterations and the speed of each iteration.

7.2.2 Maximal Violating Pair Working Set Selection

The most obvious simplification of (23) consist in performing a first order approximation of the objective function

$$\mathcal{D}(\boldsymbol{\alpha} + \lambda \mathbf{u}^{ij}) - \mathcal{D}(\boldsymbol{\alpha}) \approx \lambda\, \mathbf{g}^\top \mathbf{u}^{ij},$$

and, in order to make this approximation valid, to replace the constraints by a constraint that ensures that λ remains very small. This yields problem

$$\mathbf{u}^* \ = \ \arg\max_{\mathbf{u}^{ij} \in \mathcal{U}} \ \max_{0 \leq \lambda \leq \epsilon} \ \lambda\, \mathbf{g}^\top \mathbf{u}^{ij}.$$

We can assume that there is a direction $\mathbf{u} \in \mathcal{U}$ such that $\mathbf{g}^\top \mathbf{u} > 0$ because we would otherwise have reached the optimum (see section 3.2). Maximizing in λ then yields

$$\mathbf{u}^* \;=\; \arg\max_{\mathbf{u}^{ij} \in \mathcal{U}} \;\; \mathbf{g}^\top \mathbf{u}^{ij}. \tag{24}$$

This problem was first studied by Joachims (1999). A first look suggests that we may have to check the $n(n-1)$ possible pairs (i, j). However we can write

$$\max_{\mathbf{u}^{ij} \in \mathcal{U}} \mathbf{g}^\top \mathbf{u}^{ij} \;=\; \max_{i \in I_{\text{up}}\ j \in I_{\text{down}}} (y_i g_i - y_j g_j) \;=\; \max_{i \in I_{\text{up}}} y_i g_i \;-\; \min_{j \in I_{\text{down}}} y_j g_j .$$

We recognize here the usual optimality criterion (11). The solution of (24) is therefore the *maximal violating pair* (Keerthi et al., 2001)

$$\begin{aligned} i &= \arg\max_{k \in I_{\text{up}}} \; y_k g_k, \\ j &= \arg\min_{k \in I_{\text{down}}} \; y_k g_k. \end{aligned} \tag{25}$$

This computation require a time proportional to n. Its results can immediately be reused to check the stopping criterion (21). This is illustrated in algorithm 6.2.

7.2.3 Second Order Working Set Selection

Computing (25) does not require any additional kernel values. However some kernel values will eventually be needed to perform the SMO iteration (see algorithm 6.2, lines 7 and 8). Therefore we have the opportunity to do better with limited additional costs.

Instead of a linear approximation of (23), we can keep the quadratic gain

$$\mathcal{D}(\boldsymbol{\alpha} + \lambda \mathbf{u}^{ij}) - \mathcal{D}(\boldsymbol{\alpha}) = \lambda(y_i g_i - y_j g_j) - \frac{\lambda^2}{2}(K_{ii} + K_{jj} - 2K_{ij})$$

but eliminate the constraints. The optimal λ is then

$$\frac{y_i g_i - y_j g_j}{K_{ii} + K_{jj} - 2K_{ij}}$$

and the corresponding gain is

$$\frac{(y_i g_i - y_j g_j)^2}{2(K_{ii} + K_{jj} - 2K_{ij})}.$$

Unfortunately the maximization

$$\mathbf{u}^* = \arg\max_{\mathbf{u}^{ij} \in \mathcal{U}} \frac{(y_i g_i - y_j g_j)^2}{2(K_{ii} + K_{jj} - 2K_{ij})} \quad \text{subject to} \;\; y_i g_i > y_j g_j \tag{26}$$

still requires an exhaustive search through the the $n(n-1)$ possible pairs of indices. A viable implementation of second order working set selection must heuristically restrict this search.

The LIBSVM solver uses the following procedure (Fan et al., 2005).

$$\begin{aligned} i &= \arg\max_{k \in I_{\text{up}}} \; y_k g_k \\ j &= \arg\max_{k \in I_{\text{down}}} \; \frac{(y_i g_i - y_k g_k)^2}{2(K_{ii} + K_{kk} - 2K_{ik})} \quad \text{subject to} \;\; y_i g_i > y_k g_k \end{aligned} \tag{27}$$

The computation of i is exactly as for the maximal violating pair scheme. The computation of j can be achieved in time proportional to n. It only requires the diagonal of the kernel matrix, which is easily cached, and the i-th row of the kernel matrix, which is required anyway to update the gradients (algorithm 6.2, line 8).

We omit the discussion of $K_{ii} + K_{jj} - 2K_{ij} = 0$. See (Fan et al., 2005) for details and experimental results. The convergence of this working set selection scheme is proved in (Chen et al., 2006).

Note that the first order (24) and second order (26) problems are only used for selecting the working set. They do not have to maintain the feasibility constraint of the maximal gain problem (23). Of course the feasibility constraints must be taken into account during the computation of $\boldsymbol{\alpha}'$ that is performed after the determination of the working set. Some authors (Lai et al., 2003; Glasmachers and Igel, 2006) maintain the feasibility constraints during the working set selection. This leads to different heuristic compromises.

7.3 Shrinking

The *shrinking* technique reduces the size of the problem by temporarily eliminating variables α_i that are unlikely to be selected in the SMO working set because they have reached their lower or upper bound (Joachims, 1999). The SMO iterations then continues on the remaining variables. Shrinking reduces the number of kernel values needed to update the gradient vector (see algorithm 6.2, line 8). The hit rate of the kernel cache is therefore improved.

For many problems, the number of free support vectors (section 4.1) is relatively small. During the iterative process, the solver progressively identifies the partition of the training examples into non support vectors ($\alpha_i = 0$), bounded support vectors ($\alpha_i = C$), and free support vectors. Coefficients associated with non support vectors and bounded support vectors are known with high certainty and are good candidates for shrinking.

Consider the sets[6]

$$\begin{aligned} &J_{\text{up}}(\boldsymbol{\alpha}) = \{\, k \mid y_k g_k > m(\boldsymbol{\alpha}) \} \quad &&\text{with} \quad m(\boldsymbol{\alpha}) = \max_{i \in I_{\text{up}}} \; y_i g_i, \quad \text{and} \\ &J_{\text{down}}(\boldsymbol{\alpha}) = \{\, k \mid y_k g_k < M(\boldsymbol{\alpha}) \} \quad &&\text{with} \quad M(\boldsymbol{\alpha}) = \min_{j \in I_{\text{down}}} \; y_j g_j. \end{aligned}$$

With these definitions, we have

$$\begin{aligned} k \in J_{\text{up}}(\boldsymbol{\alpha}) \quad &\Longrightarrow \quad k \notin I_{\text{up}} \quad &&\Longrightarrow \quad \alpha_k = y_k B_k, \quad \text{and} \\ k \in J_{\text{down}}(\boldsymbol{\alpha}) \quad &\Longrightarrow \quad k \notin I_{\text{down}} \quad &&\Longrightarrow \quad \alpha_k = y_k A_k. \end{aligned}$$

In other words, these sets contain the indices of variables α_k that have reached their lower or upper bound with a sufficiently "strong" derivative, and therefore are likely to stay there.

6. In these definitions, J_{up}, J_{down} and g_k implicitely depend on $\boldsymbol{\alpha}$.

- Let $\boldsymbol{\alpha}^*$ be an arbitrary solution of the dual optimization problem. The quantities $m(\boldsymbol{\alpha}^*)$, $M(\boldsymbol{\alpha}^*)$, and $y_k\, g_k(\boldsymbol{\alpha}^*)$ do not depend on the chosen solution (Chen et al., 2006, theorem 4). Therefore the sets $J^*_{\text{up}} = J_{\text{up}}(\boldsymbol{\alpha}^*)$ and $J^*_{\text{down}} = J_{\text{down}}(\boldsymbol{\alpha}^*)$ are also independent of the chosen solution.

- There is a finite number of possible sets $J_{\text{up}}(\boldsymbol{\alpha})$ or $J_{\text{down}}(\boldsymbol{\alpha})$. Using the continuity argument of (Chen et al., 2006, theorem 6), both sets $J_{\text{up}}(\boldsymbol{\alpha})$ and $J_{\text{down}}(\boldsymbol{\alpha})$ reach and keep their final values J^*_{up} and J^*_{down} after a finite number of SMO iterations.

Therefore the variables α_i, $i \in J^*_{\text{up}} \cup J^*_{\text{down}}$ also reach their final values after a finite number of iterations and can then be safely eliminated from the optimization. In practice, we cannot be certain that $J_{\text{up}}(\boldsymbol{\alpha})$ and $J_{\text{down}}(\boldsymbol{\alpha})$ have reached their final values J^*_{up} and J^*_{down}. We can however tentatively eliminate these variables and check later whether we were correct.

- The LIBSVM *shrinking* routine is invoked every $\min(n, 1000)$ iterations. It dynamically eliminates the variables α_i whose indices belong to $J_{\text{up}}(\boldsymbol{\alpha}) \cup J_{\text{down}}(\boldsymbol{\alpha})$.

- Since this strategy may be too aggressive, *unshrinking* takes place whenever $m(\boldsymbol{\alpha}^k) - M(\boldsymbol{\alpha}^k) < 10\epsilon$, where ϵ is the stopping tolerance (21). The whole gradient vector $\mathbf{g}$ is first reconstructed. All variables that do not belong to $J_{\text{up}}(\boldsymbol{\alpha})$ or $J_{\text{down}}(\boldsymbol{\alpha})$ are then reactivated.

The reconstruction of the full gradient $\mathbf{g}$ can be quite expensive. To decrease this cost, LIBSVM maintains an additional vector $\bar{\mathbf{g}} = (\bar{g}_1 \dots \bar{g}_n)$

$$\bar{g}_i \;=\; y_i \sum_{\alpha_k = C} y_k \alpha_k\, K_{ik} \;=\; y_i C \sum_{\alpha_k = C} y_k\, K_{ik}.$$

during the SMO iterations. This vector needs to be updated whenever a SMO iteration causes a coefficient α_k to reach or leave the upper bound C. The full gradient vector $\mathbf{g}$ is then reconstructed using the relation

$$g_i = 1 - \bar{g}_i - y_i \sum_{0<\alpha_k<C} y_k \alpha_k\, K_{ik}.$$

The shrinking operation never eliminate free variables ($0 < \alpha_k < C$). Therefore this gradient reconstruction only needs rows of the kernel matrix that are likely to be present in the kernel cache.

7.4 Implementation Issues

Two aspects of the implementation of a robust solver demand a particular attention: numerical accuracy and caching.

7.4.1 Numerical Accuracy

Numerical accuracy matters because many parts of the algorithm distinguish the variables α_i that have reached their bounds from the other variables. Inexact computations could unexpectedly change this partition of the optimization variables with catastrophic consequences.

Algorithm 6.2 updates the variables α_i without precautions (line 9). The LIBSVM code makes sure, when a direction search hits the box constraints, that at least one of the coefficient α_i or α_j is exactly equal to its bound. No particular attention is then necessary to determine which coefficients of $\boldsymbol{\alpha}$ have reached a bound.

Other solvers use the opposite approach (e.g. SVQP2) and always use a small tolerance to decide whether a variable has reached its lower or upper bound.

7.4.2 Data Structure for the Kernel Cache

Each entry i of the LIBSVM kernel cache represents the i-th row of the kernel matrix, but stores only the first $l_i \leq n$ row coefficients $K_{i1} \dots K_{i\,l_i}$. The variables l_i are dynamically adjusted to reflect the known kernel coefficients.

Shrinking is performed by permuting the examples in order to give the lower indices to the active set. Kernel entries for the active set are then grouped at the beginning of each kernel matrix row. To swap the positions of two examples, one has to swap the corresponding coefficients in vectors $\boldsymbol{\alpha}$, $\mathbf{g}$, $\bar{\mathbf{g}}$ and the corresponding entries in the kernel cache.

When the SMO algorithm needs the first l elements of a particular row i, the LIBSVM caching code retrieves the corresponding cache entry i and the number l_i of cached coefficients. Missing kernel values are recomputed and stored. Finally a pointer to the stored row is returned.

The LIBSVM caching code also maintains a circular list of recently used rows. Whenever the memory allocated for the cached rows exceeds the predefined maximum, the cached coefficients for the least recently used rows are deallocated.

8. Conclusion

This chapter has presented the state-of-the-art technique to solve the SVM dual optimization problem with an accuracy that comfortably exceeds the needs of most machine learning applications. Like early SVM solvers, these techniques perform repeated searches along well chosen feasible directions (section 5.2). The choice of search directions must balance several objectives:

- Search directions must be *sparse*: the SMO search directions have only two non zero coefficients (section 6.3).

- Search directions must leverage the *cached kernel values*. The decomposition method (section 6) and the shrinking heuristics (section 7.3) are means to achieve this objective.

- Search directions must offer good chances to increase the dual objective function (section 7.2).

This simple structure provides opportunities for large scale problems. Search directions are usually selected on the basis of the gradient vector which can be costly to compute. Bordes et al. (2005) use an *approximate* optimization algorithm that partly relies on chance to select the successive search directions.

Approximate optimization can yield considerable speedups because there is no point in achieving a small optimization error when the estimation and approximation errors are relatively large. However, the determination of stopping criteria in dual optimization can be very challenging (Tsang et al., 2005; Loosli and Canu, 2006).

Global approaches have been proposed for the approximate representation (Fine and Scheinberg, 2001) or computation (Yang et al., 2005) of the kernel matrix. These methods can be very useful for non sparse kernel machines. In the case of Support Vector Machines, it remains difficult to achieve the benefits of these methods without partly losing the benefits of sparsity.

Acknowledgements

Part of this work was funded by NSF grant CCR-0325463. The second author thanks his students for proofreading the paper.

References

Nachman Aronszajn. La théorie générale des noyaux réproduisants et ses applications. *Proceedings of the Cambridge Philosophical Society*, 39:133–153, 1944.

Kristin P. Bennett and Erin J. Bredensteiner. Duality and geometry in SVM classifiers. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, pages 57–64, San Francisco, California, 2000. Morgan Kaufmann.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

Léon Bottou and Yann Le Cun. Large scale online learning. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

Léon Bottou, Corinna Cortes, John S. Denker, Harris Drucker, Isabelle Guyon, Lawrence D. Jackel, Yann LeCun, Urs A. Muller, Edward Sackinger, Patrice Simard, and Vladimir N. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Conference B: Computer Vision & Image Processing.*, volume 2, pages 77–82, Jerusalem, October 1994. IEEE.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001a. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chih-Chung Chang and Chih-Jen Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119–2147, 2001b.

Chih-Chung Chang, Chih-Wei Hsu, and Chih-Jen Lin. The analysis of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 11(4):1003–1008, 2000.

Pai-Hsuen Chen, Rong-En Fan, and Chih-Jen Lin. A study on SMO-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 17:893–908, July 2006.

Ronan Collobert. *Large Scale Machine Learning*. PhD thesis, Université Paris VI, 2004.

Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Large scale transductive svms. *Journal of Machine Learning Research*, 7:1687–1712, September 2006.

Corinna Cortes and Vladimir N. Vapnik. Support vector networks. *Machine Learning*, 20: pp 1–25, 1995.

Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using second order information for training svm. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

Tobias Glasmachers and Christian Igel. Maximum-gain working set selection for SVMs. *Journal of Machine Learning Research*, 7:1437–1466, July 2006.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

Don Hush and Clint Scovel. Polynomial-time decomposition algorithms for support vector machines. *Machine Learning*, 51:51–71, 2003.

Don Hush, Patrick Kelly, Clint Scovel, and Ingo Steinwart. QP algorithms with guaranteed accuracy and run time for support vector machines. *Journal of Machine Learning Research*, 7:733–769, 2006.

Thorsten Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, USA, 1999. MIT Press.

S. Sathiya Keerthi and Elmer G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46:351–360, 2002.

S. Sathiya Keerthi, Shirish K. Shevade, Chiranjib Bhattacharyya, and Krishna R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. Technical Report Technical Report TR-ISL-99-03, Indian Institute of Science, Bangalore, 1999.

S. Sathiya Keerthi, Shirish K. Shevade, Chiranjib Bhattacharyya, and Krishna R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–649, 2001.

Damiel Lai, Nallasamy Mani, and Marimuthu Palaniswami. A new method to select working sets for faster training for support vector machines. Technical Report MESCE-30-2003, Dept. Electrical and Computer Systems Engineering Monash University, Australia, 2003.

Chih-Jen Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 13(5):1045–1052, 2002.

Chih-Jen Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

Niko List and Hans-Ulrich Simon. A general convergence theorem for the decomposition method. Technical report, Fakultät für Mathematik, Ruhr Universität Bochum, 2004.

Gaëlle Loosli and Stéphane Canu. Comments on the "Core Vector Machines: Fast SVM training on very large data sets". Technical report, INSA Rouen, July 2006. Submitted to JMLR.

Nils J. Nilsson. *Learning machines: Foundations of Trainable Pattern Classifying Systems*. McGraw–Hill, 1965.

Edgar Osuna, Robert Freund, and Frederico Girosi. Support vector machines: Training and applications. Technical Report AIM-1602, MIT Artificial Intelligence Laboratory, 1997.

Laura Palagi and Marco Sciandrone. On the convergence of a modified version of SVM^{light} algorithm. *Optimization Methods and Software*, 20(2-3):315–332, 2005.

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

Craig J. Saunders, Mark O. Stitson, Jason Weston, Bottou Léon, Bernhard Schölkopf, and J. Smola, Alexander. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway University of London, 1998.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

Bernhard Schölkopf, Alexanxer J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

Bernhard Schölkopf, John Platt, John Shawe-Taylor, Alexander J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

Ingo Steinwart. Sparseness of support vector machines—some asymptotically sharp bounds. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Ingo Steinwart and Clint Scovel. Fast rates for support vector machines using gaussian kernels. Technical Report LA-UR-04-8796, Los Alamos National Laboratory, 2004. to appear in Annals of Statistics.

Norikazu Takahashi and Tetsuo Nishi. Rigorous proof of termination of SMO algorithm for support vector machines. *IEEE Transactions on Neural Networks*, 16(3):774–776, 2005.

Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Very large SVM training using Core Vector Machines. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTAT'05)*. Society for Artificial Intelligence and Statistics, 2005.

Robert J. Vanderbei. Interior-point methods: Algorithms and formulations. *ORSA Journal on Computing*, 6(1):32–34, 1994.

Vladimir N. Vapnik. *Estimation of dependences based on empirical data*. Springer Series in Statistics. Springer Verlag, Berlin, New York, 1982.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995. ISBN 0-387-94559-8.

Vladimir N. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

Vladimir N. Vapnik, T. G. Glaskova, V. A. Koscheev, A. I. Mikhailski, and A. Y. Chervonenkis. *Algorihms and Programs for Dependency Estimation*. Nauka, 1984. In Russian.

Changjaing. Yang, Ramani Duraiswami, and Larry Davis. Efficient kernel machines using the improved fast Gauss transform. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1561–1568. MIT Press, Cambridge, MA, 2005.