XXX

2 Training a Support Vector Machine in the Primal

Olivier Chapelle olivier.chapelle@tuebingen.mpg.de
MPI for Biological Cybernetics
72076 Tübingen, Germany

Most literature on Support Vector Machines (SVMs) concentrate on the dual optimization problem. In this paper, we would like to point out that the primal problem can also be solved efficiently, both for linear and non-linear SVMs, and that there is no reason to ignore this possibility. On the contrary, from the primal point of view new families of algorithms for large scale SVM training can be investigated.

2.1 Introduction

The vast majority of text books and articles introducing Support Vector Machines (SVMs) first state the primal optimization problem, and then go directly to the dual formulation (Vapnik, 1998; Burges, 1998; Cristianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2002). A reader could easily obtain the impression that this is the only possible way to train an SVM.

In this paper, we would like to reveal this as being a misconception, and show that someone unaware of duality theory could train an SVM. Primal optimizations of linear SVMs have already been studied by Keerthi and DeCoste (2005); Mangasarian (2002). One of the main contributions of this paper is to complement those studies to include the non-linear case.[1] Our goal is not to claim that the primal optimization is better than dual, but merely to show that they are *two equivalent ways of reaching the same result*. Also, we will show that when the goal is to find an *approximate* solution, primal optimization is superior.

1. Primal optimization of non-linear SVMs have also been proposed in (Lee and Mangasarian, 2001b, Section 4), but with a different regularizer.

Given a training set $\{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq n}, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{+1, -1\}$, recall that the primal SVM optimization problem is usually written as:

$$\min_{\mathbf{w},b} \; ||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i^p \quad \text{under constraints} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \;\; \xi_i \geq 0. \tag{2.1}$$

where p is either 1 (hinge loss) or 2 (quadratic loss). At this point, in the literature there are usually two main reasons mentioned for solving this problem in the dual:

1. The duality theory provides a convenient way to deal with the constraints.
2. The dual optimization problem can be written in terms of dot products, thereby making it possible to use kernel functions.

We will demonstrate in section 3 that those two reasons are not a limitation for solving the problem in the primal, mainly by writing the optimization problem as an unconstrained one and by using the representer theorem. In section 4, we will see that performing a Newton optimization in the primal yields exactly the same computational complexity as optimizing the dual; that will be validated experimentally in section 5. Finally, possible advantages of a primal optimization are presented in section 6. But we will start now with some general discussion about primal and dual optimization.

2.2 Links between primal and dual optimization

As mentioned in the introduction, primal and dual optimization have strong connections and we illustrate some of them through the example of regularized least-squares (RLS).

Given a matrix $X \in \mathbb{R}^{n \times d}$ representing the coordinates of n points in d dimensions and a target vector $\mathbf{y} \in \mathbb{R}^n$, the primal RLS problem can be written as

$$\min_{\mathbf{w} \in \mathbb{R}^d} \quad \lambda \mathbf{w}^\top \mathbf{w} + \|X\mathbf{w} - \mathbf{y}\|^2, \tag{2.2}$$

where λ is the regularization parameter. This objective function is minimized for $\mathbf{w} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$ and its minimum is

$$\mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top X (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}. \tag{2.3}$$

Introducing slack variables $\xi = X\mathbf{w} - \mathbf{y}$, the dual optimization problem is

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad 2\boldsymbol{\alpha}^\top \mathbf{y} - \frac{1}{\lambda} \boldsymbol{\alpha}^\top (XX^\top + \lambda I) \boldsymbol{\alpha}. \tag{2.4}$$

The dual is maximized for $\boldsymbol{\alpha} = \lambda (XX^\top + \lambda I)^{-1} \mathbf{y}$ and its maximum is

$$\lambda \mathbf{y}^\top (XX^\top + \lambda I)^{-1} \mathbf{y}. \tag{2.5}$$

The primal solution is then given by the KKT condition,

$$\mathbf{w} = \frac{1}{\lambda} X^\top \boldsymbol{\alpha}. \tag{2.6}$$

Now we relate the inverses of $XX^\top + \lambda I$ and $X^\top X + \lambda I$ thanks to the *Woodbury formula* (Golub and Loan, 1996, page 51),

$$\lambda(XX^\top + \lambda I)^{-1} = I - X(\lambda I + X^\top X)^{-1} X^\top \tag{2.7}$$

With this equality, we recover that primal (2.3) and dual (2.5) optimal values are the same, i.e. that the duality gap is zero.

Let us now analyze the computational complexity of primal and dual optimization. The primal requires the computation and inversion of the matrix $(X^\top X + \lambda I)$, which is in $\mathcal{O}(nd^2 + d^3)$. On the other hand, the dual deals with the matrix $(XX^\top + \lambda I)$, which requires $\mathcal{O}(dn^2 + n^3)$ operations to compute and invert. It is often argued that one should solve either the primal or the dual optimization problem depending on whether n is larger or smaller than d, resulting in an $\mathcal{O}(\max(n,d)\min(n,d)^2)$ complexity. But this argument does not really hold because one can always use (2.7) in case the matrix to invert is too big.[2] So both for primal and dual optimization, the complexity is $\mathcal{O}(\max(n,d)\min(n,d)^2)$.

The difference between primal and dual optimization comes when computing approximate solutions. Let us optimize both the primal (2.2) and dual (2.4) objective functions by conjugate gradient and see how the primal objective function decreases as a function of the number of conjugate gradient steps. For the dual optimization, an approximate dual solution is converted to an approximate primal one by using the KKT condition (2.6).

Intuitively, the primal optimization should be superior because it directly minimizes the quantity we are interested in. Figure 2.1 confirms this intuition. In some cases, there is no difference between primal and dual optimization (left), but in some other cases, the dual optimization can be slower to converge (right).[3] In Appendix 2.A, we try to analyze this phenomenon by looking at the primal objective value after one conjugate gradient step. We show that the primal optimization always yields a lower value than the dual optimization, and we quantify the difference.

The conclusion from this analysis is that even though primal and dual optimization are equivalent, both in terms of the solution and time complexity, when it comes to *approximate solution*, *primal optimization is superior* because it is more focused on minimizing what we are interested in: the primal objective function. In general

2. Note that primal optimization with the Woodbury formula is in general not equivalent to dual optimization (even though they have the same complexity). Indeed, the dual problem involves the conjugate of the loss function. RLS is a special case because $t \to \frac{1}{2}t^2$ is self-conjugate.

3. As discussed above the time complexity is the same for finding the exact solution. But with approximate methods, this is not necessarily the case.

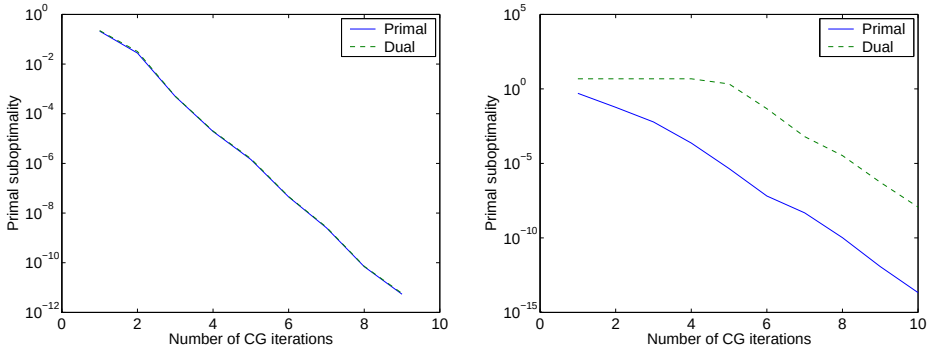

Figure 2.1: Plots of the primal suboptimality, (2.2)-(2.3), for primal and dual optimization by conjugate gradient (`pcg` in Matlab). n points are drawn randomly from a spherical Gaussian distribution in d dimensions. The targets are also randomly generated according a Gaussian distribution. λ is fixed to 1. Left, $n = 10$ and $d = 100$. Right, $n = 100$ and $d = 10$.

there is indeed no guarantee that an approximate dual solution yields a good approximate primal solution.

2.3 Primal objective function

Coming back to Support Vector Machines, let us rewrite (2.1) as an unconstrained optimization problem:

$$||\mathbf{w}||^2 + C \sum_{i=1}^{n} L(y_i, \mathbf{w} \cdot \mathbf{x}_i + b), \tag{2.8}$$

with $L(y, t) = \max(0, 1 - yt)^p$ (see figure 2.2). More generally, L could be any loss function.

Let us now consider non-linear SVMs with a kernel function k and an associated Reproducing Kernel Hilbert Space $\mathcal{H}$. The optimization problem (2.8) becomes

$$\min_{f \in \mathcal{H}} \lambda ||f||^2_{\mathcal{H}} + \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)), \tag{2.9}$$

where we have made a change of variable by introducing the regularization parameter $\lambda = 1/C$. We have also dropped the offset b for the sake of simplicity. However all the algebra presented below can be extended easily to take it into account (see Appendix 2.B).

Suppose now that the loss function L is differentiable with respect to its second argument. Using the reproducing property $f(\mathbf{x}_i) = \langle f, k(\mathbf{x}_i, \cdot) \rangle_{\mathcal{H}}$, we can differentiate (2.9) with respect to f and at the optimal solution f^*, the gradient

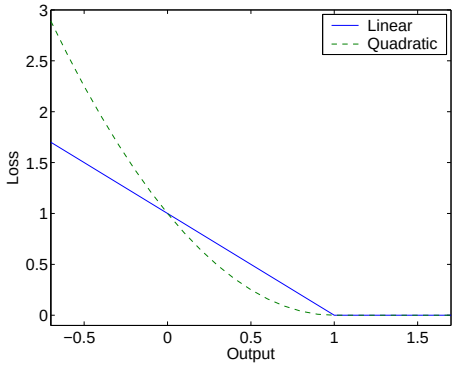

Figure 2.2: SVM loss function, $L(y,t) = \max(0, 1-yt)^p$ for $p = 1$ and 2.

vanishes, yielding

$$2\lambda f^* + \sum_{i=1}^{n} \frac{\partial L}{\partial t}(y_i, f^*(\mathbf{x}_i))k(\mathbf{x}_i, \cdot) = 0, \tag{2.10}$$

where $\partial L/\partial t$ is the partial derivative of $L(y,t)$ with respect to its second argument. This implies that the optimal function can be written as a linear combination of kernel functions evaluated at the training samples. This result is also known as the *representer theorem* (Kimeldorf and Wahba, 1970).

Thus, we seek a solution of the form:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \beta_i k(\mathbf{x}_i, \mathbf{x}). \tag{2.11}$$

We denote those coefficients β_i and not α_i as in the standard SVM literature to stress that they should *not* be interpreted as Lagrange multipliers.

Let us express (2.9) in term of β_i,

$$\lambda \sum_{i,j=1}^{n} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n} L\left(y_i, \sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j)\beta_j\right), \tag{2.12}$$

where we used the kernel reproducing property in $||f||_{\mathcal{H}}^2 = \sum_{i,j=1}^{n} \beta_i\beta_j < k(\mathbf{x}_i, \cdot), k(\mathbf{x}_j, \cdot) >_{\mathcal{H}} = \sum_{i,j=1}^{n} \beta_i\beta_j k(\mathbf{x}_i, \mathbf{x}_j)$.

Introducing the kernel matrix K with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and K_i the i^{th} column of K, (2.12) can be rewritten as

$$\Omega(\boldsymbol{\beta}) \;:=\; \lambda \boldsymbol{\beta}^\top K \boldsymbol{\beta} + \sum_{i=1}^{n} L(y_i, K_i^\top \boldsymbol{\beta}). \tag{2.13}$$

As long as L is differentiable, we can optimize (2.13) by gradient descent. Note

that this is an unconstrained optimization problem.

2.4 Newton optimization

The unconstrained objective function (2.13) can be minimized using a variety of optimization techniques such as conjugate gradient. Here we will only consider Newton optimization as the similarities with dual optimization will then appear clearly.

We will focus on two loss functions: the quadratic penalization of the training errors (figure 2.2) and a differentiable approximation to the linear penalization, the Huber loss.

2.4.1 Quadratic loss

Let us start with the easiest case, the L_2 penalization of the training errors,

$$L(y_i, f(\mathbf{x}_i)) = \max(0, 1 - y_i f(\mathbf{x}_i))^2.$$

For a given value of the vector $\boldsymbol{\beta}$, we say that a point $\mathbf{x}_i$ is a *support vector* if $y_i f(\mathbf{x}_i) < 1$, i.e. if the loss on this point is non zero. Note that this definition of support vector is different from $\beta_i \neq 0$ [4]. Let us reorder the training points such that the first n_{sv} points are support vectors. Finally, let I^0 be the $n \times n$ diagonal matrix with the first n_{sv} entries being 1 and the others 0,

$$I^0 \equiv \begin{pmatrix} 1 & & & & & \\ & \ddots & & & 0 & \\ & & 1 & & & \\ & & & 0 & & \\ & 0 & & & \ddots & \\ & & & & & 0 \end{pmatrix}$$

4. From (2.10), it turns out at the optimal solution that the sets $\{i,\ \beta_i \neq 0\}$ and $\{i,\ y_i f(\mathbf{x}_i) < 1\}$ will be the same. To avoid confusion, we could have defined this latter as the set of *error vectors*.

The gradient of (2.13) with respect to $\boldsymbol{\beta}$ is

$$\begin{aligned}\nabla &= 2\lambda K\boldsymbol{\beta} + \sum_{i=1}^{n_{\mathsf{sv}}} K_i \frac{\partial L}{\partial t}(y_i, K_i^\top \boldsymbol{\beta}) \\ &= 2\lambda K\boldsymbol{\beta} + 2\sum_{i=1}^{n_{\mathsf{sv}}} K_i y_i (y_i K_i^\top \boldsymbol{\beta} - 1) \\ &= 2(\lambda K\boldsymbol{\beta} + KI^0(K\boldsymbol{\beta} - Y)), \end{aligned} \tag{2.14}$$

and the Hessian,

$$H = 2(\lambda K + KI^0 K). \tag{2.15}$$

Each Newton step consists of the following update,

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma H^{-1}\nabla,$$

where γ is the step size found by line search or backtracking (Boyd and Vandenberghe, 2004, Section 9.5). In our experiments, we noticed that the default value of $\gamma = 1$ did not result in any convergence problem, and in the rest of this section we only consider this value. However, to enjoy the theoretical properties concerning the convergence of this algorithm, backtracking is necessary.

Combining (2.14) and (2.15) as $\nabla = H\boldsymbol{\beta} - 2KI^0 Y$, we find that after the update,

$$\begin{aligned}\boldsymbol{\beta} &= (\lambda K + KI^0 K)^{-1} KI^0 Y \\ &= (\lambda I_n + I^0 K)^{-1} I^0 Y \end{aligned} \tag{2.16}$$

Note that we have assumed that K (and thus the Hessian) is invertible. If K is not invertible, then the expansion is not unique (even though the solution is), and (2.16) will produce one of the possible expansions of the solution. To avoid these problems, let us simply assume that an infinitesimally small ridge has been added to K.

Let I_p denote the identity matrix of size $p \times p$ and K_{sv} the first n_{sv} columns and rows of K, i.e. the submatrix corresponding to the support vectors. Using the fact that the lower left block $\lambda I_n + I^0 K$ is 0, the inverse of this matrix can be easily computed, and finally, the update (2.16) turns out to be

$$\begin{aligned}\boldsymbol{\beta} &= \begin{pmatrix} (\lambda I_{n_{\mathsf{sv}}} + K_{\mathsf{sv}})^{-1} & 0 \\ 0 & 0 \end{pmatrix} Y, \\ &= \begin{pmatrix} (\lambda I_{n_{\mathsf{sv}}} + K_{\mathsf{sv}})^{-1} Y_{\mathsf{sv}} \\ 0 \end{pmatrix}. \end{aligned} \tag{2.17}$$

If the current solution is far from the optimal one, the set sv might be large and some computational ressources wasted on trying to invert (2.17). We will present later (algorithm 2.1) a way to avoid this problem.

2.4.1.1 Link with dual optimization

Update rule (2.17) is not surprising if one has a look at the SVM dual optimization problem:

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (K_{ij} + \lambda \delta_{ij}), \quad \text{under constraints } \alpha_i \geq 0.$$

Consider the optimal solution: the gradient with respect to all $\alpha_i > 0$ (the support vectors) must be 0,

$$\mathbf{1} - \text{diag}(Y_{\mathsf{sv}})(K_{\mathsf{sv}} + \lambda I_{n_{\mathsf{sv}}})\text{diag}(Y_{\mathsf{sv}})\boldsymbol{\alpha} = 0,$$

where $\text{diag}(Y)$ stands for the diagonal matrix with the diagonal being the vector Y. Thus, up to a sign difference, the solutions found by minimizing the primal and maximizing the dual are the same: $\beta_i = y_i \alpha_i$.

2.4.1.2 Complexity analysis

Only a couple of iterations are usually necessary to reach the solution (rarely more than 5), and this number seems independent of n. The overall complexity is thus the complexity of one Newton step, which is $O(nn_{\mathsf{sv}} + n_{\mathsf{sv}}^3)$. Indeed, the first term corresponds to finding the support vectors (i.e. the points for which $y_i f(\mathbf{x}_i) < 1$) and the second term is the cost of inverting the matrix $K_{\mathsf{sv}} + \lambda I_{n_{\mathsf{sv}}}$. It turns out that this is the same complexity as in standard SVM learning (dual maximization) since those 2 steps are also necessary. Of course n_{sv} is not known in advance and this complexity analysis is an *a posteriori* one. In the worse case, the complexity is $O(n^3)$.

It is important to note that in general this time complexity is also a lower bound (for the *exact* computation of the SVM solution). Chunking and decomposition methods, for instance (Joachims, 1999; Osuna et al., 1997), do not help since there is fundamentally a linear system of size n_{sv} to be be solved.[5] Chunking is only useful when the K_{sv} matrix cannot fit in memory. Keep in mind that we did not take into account the complexity of computing entries of the kernel matrix: in practice, training time of different SVM solvers can differ significantly based on the kernel cache strategy (see also chapter 1).

5. We considered here that solving a linear system (either in the primal or in the dual) takes cubic time. This time complexity can however be improved.

2.4.2 Huber / hinge loss

The hinge loss used in SVMs is not differentiable. We propose to use a differentiable approximation of it, inspired by the Huber loss (cf figure 2.3):

$$L(y,t) = \begin{cases} 0 & \text{if} \quad yt > 1+h \\ \frac{(1+h-yt)^2}{4h} & \text{if} \quad |1-yt| \leq h \\ 1-yt & \text{if} \quad yt < 1-h \end{cases} \tag{2.18}$$

where h is a parameter to choose, typically between 0.01 and 0.5.

Note that we are not minimizing the hinge loss, but *this does not matter*, since from a machine learning point of view there is no reason to prefer the hinge loss anyway. If really one wants to approach the hinge loss solution, one can make $h \to 0$ (similarly to (Lee and Mangasarian, 2001b)).

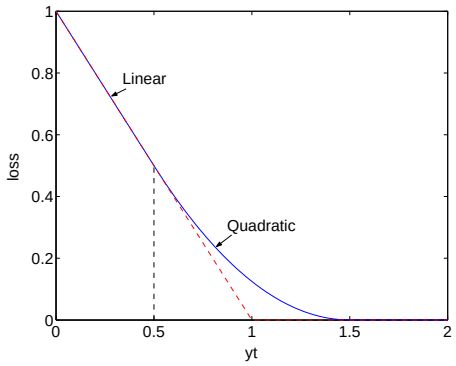

Figure 2.3: The Huber loss is a differentiable approximation of the hinge loss. The plot is (2.18) with $h = 0.5$.

The derivation of the Newton step follows the same line as for the L_2 loss and we will not go into the details. The algebra is just a bit more complicated because there are 3 different parts in the loss (and thus 3 different categories of points):

- n_{sv} of them are in the quadratic part of loss;
- $n_{\bar{\mathsf{sv}}}$ are in the linear part of the loss. We will call this category of points the support vectors "at bound", in reference to dual optimization where the Lagrange multipliers associated with those points are at the upper bound C.
- The rest of the points have zero loss.

We reorder the training set in such a way that the points are grouped in the 3 above categories. As in the previous section, I^0 corresponds to the points in the first category. For the points in the second category, let I^1 be diagonal matrix with

first n_{sv} 0 elements followed by $n_{\bar{\mathsf{sv}}}$ 1 elements (and 0 for the rest).

The gradient is

$$\nabla = 2\lambda K\boldsymbol{\beta} + \frac{KI^0(K\beta - (1+h)Y)}{2h} - KI^1Y$$

and the Hessian

$$H = 2\lambda K + \frac{KI^0K}{2h}.$$

Thus,

$$\nabla = H\boldsymbol{\beta} - K\left(\frac{1+h}{2h}I^0 + I^1\right)Y$$

and the new $\boldsymbol{\beta}$ is

$$\begin{aligned}\boldsymbol{\beta} &= \left(2\lambda I_n + \frac{I^0K}{2h}\right)^{-1}\left(\frac{1+h}{2h}I^0 + I^1\right)Y \\ &= \begin{pmatrix}(4h\lambda I_{n_{\mathsf{sv}}} + K_{\mathsf{sv}})^{-1}((1+h)Y_{\mathsf{sv}} - K_{\mathsf{sv},\bar{\mathsf{sv}}}Y_{\bar{\mathsf{sv}}}/(2\lambda)) \\ Y_{\bar{\mathsf{sv}}}/(2\lambda) \\ 0\end{pmatrix} \equiv \begin{pmatrix}\beta_{\mathsf{sv}} \\ \beta_{\bar{\mathsf{sv}}} \\ 0\end{pmatrix}. \quad (2.19)\end{aligned}$$

Again, one can see the link with the dual optimization: letting $h \to 0$, the primal and the dual solution are the same, $\beta_i = y_i\alpha_i$. This is obvious for the points in the linear part of the loss (with $C = 1/(2\lambda)$). For the points that are right on the margin, their output is equal to their label,

$$K_{\mathsf{sv}}\beta_{\mathsf{sv}} + K_{\mathsf{sv},\bar{\mathsf{sv}}}\beta_{\bar{\mathsf{sv}}} = Y_{\mathsf{sv}}.$$

But since $\beta_{\bar{\mathsf{sv}}} = Y_{\bar{\mathsf{sv}}}/(2\lambda)$,

$$\beta_{\mathsf{sv}} = K_{\mathsf{sv}}^{-1}(Y_{\mathsf{sv}} - K_{\mathsf{sv},\bar{\mathsf{sv}}}Y_{\bar{\mathsf{sv}}}/(2\lambda)),$$

which is the same equation as the first block of (2.19) when $h \to 0$.

2.4.2.1 Complexity analysis

Similar to the quadratic loss, the complexity is $O(n_{\mathsf{sv}}^3 + n(n_{\mathsf{sv}} + n_{\bar{\mathsf{sv}}}))$. The $n_{\mathsf{sv}} + n_{\bar{\mathsf{sv}}}$ factor is the complexity for computing the output of one training point (number of nonzero elements in the vector $\boldsymbol{\beta}$). Again, the complexity for dual optimization is the same since both steps (solving a linear system of size n_{sv} and computing the outputs of all the points) are required.

2.4.3 Other losses

Some other losses have been proposed to approximate the SVM hinge loss (Lee and Mangasarian, 2001b; Zhang et al., 2003; Zhu and Hastie, 2005). However, none of them has a linear part and the overall complexity is $O(n^3)$ which can be much

larger than the complexity of standard SVM training. More generally, the size of the linear system to solve is equal to n_{sv}, the number of training points for which $\frac{\partial^2 L}{\partial t^2}(y_i, f(\mathbf{x}_i)) \neq 0$. If there are some large linear parts in the loss function, this number might be much smaller than n, resulting in significant speed-up compared to the standard $O(n^3)$ cost.

2.5 Experiments

The experiments in this section can be considered as a sanity check to show that primal and dual optimization of a non-linear SVM have similar time complexities. However, for linear SVMs, the primal optimization is definitely superior (Keerthi and DeCoste, 2005) as illustrated below.[6]

Some Matlab code for the quadratic penalization of the errors and taking into account the bias b is available online at: `http://www.kyb.tuebingen.mpg.de/bs/people/chapelle/primal`.

2.5.1 Linear SVM

In the case of quadratic penalization of the training errors, the gradient of the objective function (2.8) is

$$\nabla = 2\mathbf{w} + 2C \sum_{i \in \mathsf{sv}} (\mathbf{w} \cdot \mathbf{x}_i - y_i)\mathbf{x}_i,$$

and the Hessian is

$$H = I_d + C \sum_{i \in \mathsf{sv}} \mathbf{x}_i \mathbf{x}_i^\top.$$

The computation of the Hessian is in $O(d^2 n_{\mathsf{sv}})$ and its inversion in $O(d^3)$. When the number of dimensions is relatively small compared to the number of training samples, it is advantageous to optimize directly on $\mathbf{w}$ rather than on the expansion coefficients. In the case where d is large, but the data is sparse, the Hessian should not be built explicitly. Instead, the linear system $H^{-1}\nabla$ can be solved efficiently by conjugate gradient (Keerthi and DeCoste, 2005).

Training time comparison on the Adult dataset (Platt, 1998) in presented in figure 2.4. As expected, the training time is linear for our primal implementation, but the scaling exponent is 2.2 for the dual implementation of LIBSVM (comparable to the 1.9 reported in (Platt, 1998)). This exponent can be explained as follows : n_{sv} is very small (Platt, 1998, Table 12.3) and $n_{\bar{\mathsf{sv}}}$ grows linearly with n (the misclassified

6. For a dual optimizer to be competitive, it needs to make use of the fact that the kernel matrix is low rank. For instance, the Lagrangian SVM (Mangasarian and Musicant, 2001) relies on the Woodbury formula (2.7) to train linear SVMs.

training points). So for this dataset the complexity of $O(n_{\text{sv}}^3 + n(n_{\text{sv}} + n_{\overline{\text{sv}}}))$ turns out to be about $O(n^2)$.

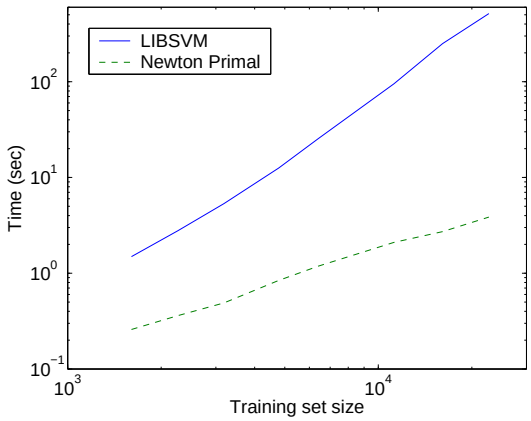

Figure 2.4: Time comparison of LIBSVM (an implementation of SMO (Platt, 1998)) and direct Newton optimization on the normal vector $\mathbf{w}$.

It is noteworthy that, for this experiment, the number of Newton steps required to reach the exact solution was 7. More generally, this algorithm is usually extremely fast for linear SVMs.

2.5.2 L_2 loss

We now compare primal and dual optimization for non-linear SVMs. To avoid problems of memory management and kernel caching and to make time comparison as straightforward as possible, we decided to precompute the entire kernel matrix. For this reason, the Adult dataset used in the previous section is not suitable because it would be difficult to fit the kernel matrix in memory (about 8G).

Instead, we used the USPS dataset consisting of 7291 training examples. The problem was made binary by classifying digits 0 to 4 versus 5 to 9. An RBF kernel with bandwidth $\sigma = 8$ was chosen,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right).$$

We consider in this section the hard margin SVM by fixing λ to a very small value, namely 10^{-8}.

The training for the primal optimization is performed as follows (see algorithm 2.1): we start from a small number of training samples, train, double the number of samples, retrain and so on. In this way, the set of support vectors is rather well identified (otherwise, we would have to invert an $n \times n$ matrix in the first Newton

step).

Algorithm 2.1 SVM primal training by Newton optimization

Function: $\boldsymbol{\beta} = \text{PRIMALSVM(K,Y,}\lambda\text{)}$
  $n \leftarrow \text{length(Y)}$ *% Number of training points*
  if $n > 1000$ **then**
    $n_2 \leftarrow n/2$ *% Train first on a subset to estimate the decision boundary*
    $\boldsymbol{\beta} \leftarrow \text{PRIMALSVM}(K_{1..n_2,1..n_2}, Y_{1..n_2}, \lambda)]$
    sv $\leftarrow$ non zero components of $\boldsymbol{\beta}$
  else
    sv $\leftarrow \{1, \ldots, n\}$.
  end if
  repeat
    $\boldsymbol{\beta}_{\text{sv}} \leftarrow (K_{\text{sv}} + \lambda I_{n_{\text{sv}}})^{-1} Y_{\text{sv}}$
    Other components of $\boldsymbol{\beta} \leftarrow 0$
    sv $\leftarrow$ indices i such that $y_i[K\boldsymbol{\beta}]_i < 1$
  until sv has not changed

The time comparison is plotted in figure 2.5: the running times for primal and dual training are almost the same. Moreover, they are directly proportional to n_{sv}^3, which turns out to be the dominating term in the $O(nn_{\text{sv}} + n_{\text{sv}}^3)$ time complexity. In this problem n_{sv} grows approximately like $\sqrt{n}$. This seems to be in contradiction with the result of Steinwart (2003), which states than the number of support vectors grows linearly with the training set size. However this result holds only for noisy problems, and the USPS dataset has a very small noise level.

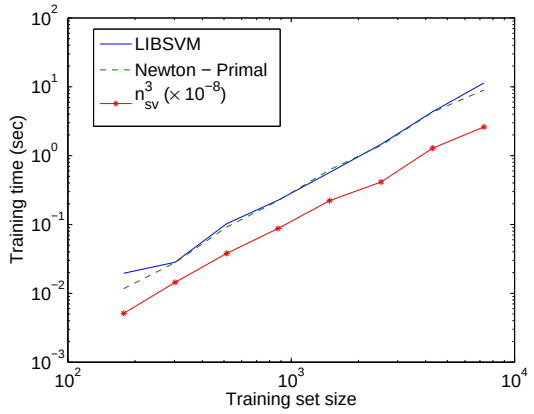

Figure 2.5: With the L_2 penalization of the slacks, the parallel between dual optimization and primal Newton optimization is striking: the training times are almost the same (and scale in $O(n_{\text{sv}}^3)$). Note that both solutions are *exactly the same.*

Even though we ignored the time spent on computing the kernel matrix, it is noteworthy that algorithm 2.1 only needs the compute submatrix K_{ij}, $1 \leq i \leq n$, $i \in \mathsf{sv}$. So the number of kernel evaluations would typically be of the order of nn_{sv}, as for dual methods (see chapter 1). If this matrix does not fit in memory, *shrinking* can be used. But if K_{sv} is also too big, then one has to resort to approximations (see section 2.6).

2.5.3 Huber loss

We perform the same experiments as in the previous section, but introduced noise in the labels: for randomly chosen 10% of the points, the labels are flipped. In this kind of situation the L_2 loss is not well suited, because it penalizes the noisy examples too much. However if the noise were, for instance, Gaussian in the inputs, then the L_2 loss would have been very appropriate.

We will study the time complexity and the test error when we vary the parameter h. Note that the solution will not usually be exactly the same as for a standard hinge loss SVM (it will only be the case in the limit $h \to 0$). The regularization parameter λ was set to $1/8$, which corresponds to the best test performance.

In the experiments described below, a line search was performed in order to make Newton converge more quickly. This means that instead of using (2.19) for updating $\boldsymbol{\beta}$, the following step was made,

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - tH^{-1}\nabla,$$

where $t \in [0, 1]$ is found by 1D minimization. This additional line search does not increase the complexity since it is just $O(n)$. Indeed, given the direction $\mathbf{u} = H^{-1}\nabla$, let us the write the objective function (2.13) along this line as

$$g(t) := \Omega(\boldsymbol{\beta} + t\mathbf{u}) = \lambda(\boldsymbol{\beta}^\top K\boldsymbol{\beta} + 2t\mathbf{u}^\top K\boldsymbol{\beta} + t^2\mathbf{u}^\top K\mathbf{u}) + \sum_{i=1}^{n} L(y_i, K_i^\top\boldsymbol{\beta} + tK_i^\top\mathbf{u}).$$

This function g takes $O(n)$ operations to evaluate once $K\boldsymbol{\beta}$ and $K\mathbf{u}$ have been precomputed. Its minimum is easily found after couple of 1D Newton steps. For the L_2 loss described in the previous section, this line search was not necessary and full Newton steps were taken ($t = 1$).

2.5.3.1 Influence of h

As expected the left hand side of figure 2.6 shows that the test error is relatively unaffected by the value of h, as long as it is not too large. For $h = 1$, the loss looks more like the L_2 loss, which is inappropriate for the kind of noise we generated.

Concerning the time complexity (right hand side of figure 2.6), there seems to be an optimal range for h. When h is too small, the problem is highly non-quadratic (because most of the loss function is linear), and a lot of Newton steps are necessary. On the other hand, when h is large, n_{sv} increases, and since the complexity is mainly

in $O(n_{\text{sv}}^3)$, the training time increases (cf figure 2.7).

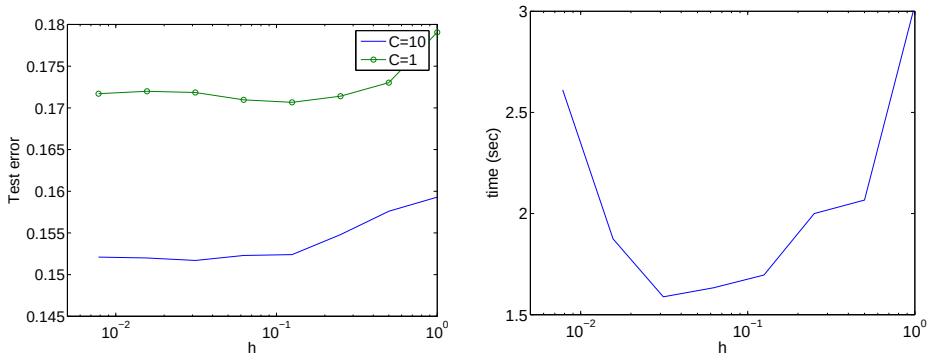

Figure 2.6: Influence of h on the test error (left, n=500) and the training time (right, $n = 1000$)

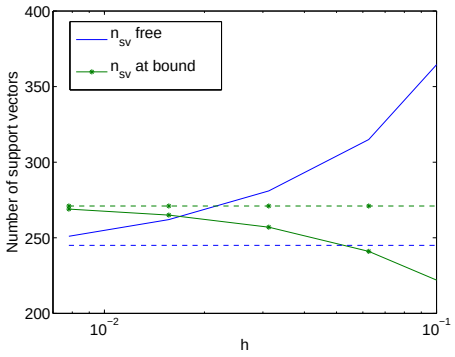

Figure 2.7: When h increases, more points are in the quadratic part of the loss (n_{sv} increases and $n_{\bar{\text{sv}}}$ decreases). The dashed lines are the LIBSVM solution. For this plot, $n = 1000$.

2.5.3.2 Time comparison with LIBSVM

Figure 2.8 presents a time comparison of both optimization methods for different training set sizes. As for the quadratic loss, the time complexity is $O(n_{\text{sv}}^3)$.

However, unlike figure 2.5, the constant for LIBSVM training time is better. This is probably the case because the loss function is far from quadratic and the Newton optimization requires more steps to converge (on the order of 30). But we believe that this factor can be improved on by not inverting the Hessian from scratch

in each iteration or by using a more direct optimizer such as conjugate gradient descent.

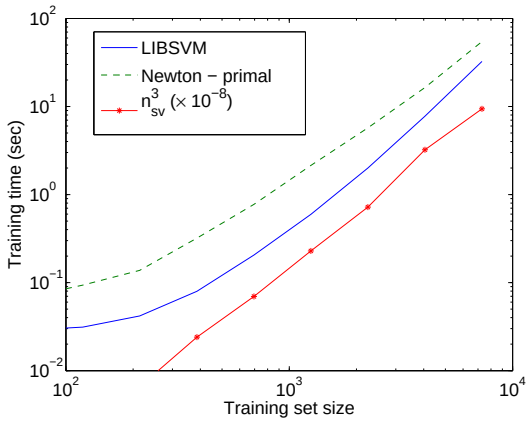

Figure 2.8: Time comparison between LIBSVM and Newton optimization. Here n_{sv} has been computed from LIBSVM (note that the solutions are not exactly the same). For this plot, $h = 2^{-5}$.

2.6 Advantages of primal optimization

As explained throughout this paper, primal and dual optimizations are very similar, and it is not surprising that they lead to same computational complexity, $O(nn_{\text{sv}} + n_{\text{sv}}^3)$. So is there a reason to use one rather than the other?

We believe that primal optimization might have advantages for *large scale optimization*. Indeed, when the number of training points is large, the number of support vectors is also typically large and it becomes intractable to compute the exact solution. For this reason, one has to resort to *approximations* (Bordes et al., 2005; Bakır et al., 2005; Collobert et al., 2002; Tsang et al., 2005); see also chapters 3 and 14 of this book. But introducing approximations in the dual may not be wise. There is indeed no guarantee that an approximate dual solution yields a good approximate primal solution. Since what we are eventually interested in is a good primal objective function value, it is more straightforward to directly minimize it (cf the discussion at the end of Section 2.2).

Below are some examples of approximation strategies for primal minimization. One can probably come up with many more, but our goal is just to give a flavor of what can be done in the primal.

2.6.1 Conjugate gradient

One could directly minimize (2.13) by conjugate gradient descent. For squared loss without regularizer, this approach has been investigated in (Ong, 2005). The hope is that on a lot of problems a reasonable solution can be obtained with only a couple of gradient steps.

In the dual, this strategy is hazardous: there is no guarantee that an approximate dual solution corresponds to a reasonable primal solution. We have indeed shown in Section 2 and Appendix 2.A that for a given number of conjugate gradient steps, the primal objective function was lower when optimizing the primal than when optimizing the dual.

However one needs to keep in mind that the performance of a conjugate gradient optimization strongly depends on the parametrization of the problem. In Section 2, we analyzed an optimization in terms of the primal variable $\mathbf{w}$, whereas in the rest of the paper, we used the reparametrization (2.11) with the vector $\boldsymbol{\beta}$. The convergence rate of conjugate gradient depends on the condition number of the Hessian (Shewchuk, 1994). For an optimization on $\boldsymbol{\beta}$, it is roughly equal to the condition number of K^2 (cf equation (2.15)), while for an optimization on $\mathbf{w}$, this is the condition number of K.

So optimizing on $\boldsymbol{\beta}$ could be much slower. There is fortunately an easy fix to this problem: preconditioning by K. In general, preconditioning by a matrix M requires to be able to compute efficiently $M^{-1}\nabla$, where ∇ is the gradient (Shewchuk, 1994, Section B5). But in our case, it turns out that one can factorize K in the expression of the gradient (2.14) and the computation of $K^{-1}\nabla$ becomes trivial. With this preconditioning (which comes at no extra computational cost), the convergence rate is now the same as for the optimization on $\mathbf{w}$. In fact, in the case of regularized least squares of section 2, one can show that the conjugate gradient steps for the optimization on $\mathbf{w}$ and for the optimization on $\boldsymbol{\beta}$ with preconditioning are identical.

Pseudocode for optimizing (2.13) using the Fletcher-Reeves update and this preconditioning is given in algorithm 2.2. Note that in this pseudocode $\mathbf{g}$ is exactly the gradient given in (2.14) but "divided" by K. Finally, we would like to point out that algorithm 2.2 has another interesting interpretation: it is indeed equivalent to performing a conjugate gradients minimization on $\mathbf{w}$ (cf (2.8)), while maintaining the solution in terms of $\boldsymbol{\beta}$, i.e. such that $\mathbf{w} = \sum \beta_i \mathbf{x}_i$. This is possible because at each step of the algorithm, the gradient (with respect to $\mathbf{w}$) is always in the span of the training points. More precisely, we have that the gradient of (2.8) with respect to $\mathbf{w}$ is $\sum (\mathbf{g}_{new})_i \mathbf{x}_i$.

Let us now have an empirical study of the conjugate gradient behavior. As in the previous section, we considered the 7291 training examples of the USPS dataset. We monitored the test error as a function of the number of conjugate gradient iterations. It can be seen in figure 2.9 that

- A relatively small number of iterations (between 10 and 100) are enough to reach a good solution. Note that the test errors at the right of the figure (corresponding

Algorithm 2.2 Optimization of (2.13) (with the L_2 loss) by preconditioned conjugate gradients.

Let $\boldsymbol{\beta} = 0$ and $\mathbf{d} = \mathbf{g}_{old} = -Y$
repeat
  Let t^* be the minimizer of (2.13) on the line $\boldsymbol{\beta} + t\mathbf{d}$.
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + t^*\mathbf{d}$.
  Let $\mathbf{o} = K\boldsymbol{\beta} - Y$ and $\mathsf{sv} = \{i, o_i y_i < 1\}$. Update I^0.
  $\mathbf{g}_{new} \leftarrow 2\lambda\boldsymbol{\beta} + I^0\mathbf{o}$.
  $\mathbf{d} \leftarrow -\mathbf{g}_{new} + \frac{\mathbf{g}_{new}^\top K \mathbf{g}_{new}}{\mathbf{g}_{old}^\top K \mathbf{g}_{old}}\mathbf{d}$.
  $\mathbf{g}_{old} \leftarrow \mathbf{g}_{new}$.
until $||\mathbf{g}_{new}|| \leq \varepsilon$

to 128 iterations) are the same as for a fully trained SVM: the objective values have almost converged at this point.

- The convergence rate depends, via the condition number, on the bandwidth σ. For $\sigma = 1$, K is very similar to the identity matrix and one step is enough to be close to the optimal solution. However, the test error is not so good for this value of σ and one should set it to 2 or 4.

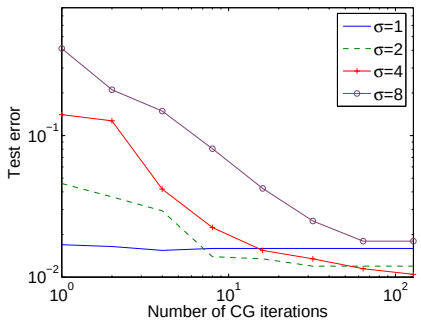

Figure 2.9: Optimization of the objective function (2.13) by conjugate gradient for different values of the kernel width.

It is noteworthy that the preconditioning discussed above is really helpful. For instance, without it, the test error is still 1.84% after 256 iterations with $\sigma = 4$.

Finally, note that each conjugate gradient step requires the computation of $K\beta$ (cf equation (2.14)) which takes $O(n^2)$ operations. If one wants to avoid recomputing the kernel matrix at each step, the memory requirement is also $O(n^2)$. Both time and memory requirement could probably be improved to $O(n_{\mathsf{sv}}^2)$ per conjugate gradient iteration by directly working on the linear system (2.19). But this complexity is probably still too high for large scale problems. Here are some cases where the

matrix vector multiplication can be done more efficiently.

Sparse kernel If a compactly supported RBF kernel (Schaback, 1995; Fasshauer, 2005) is used, the kernel matrix K is sparse. The time and memory complexities are then proportional to the number of non-zeros elements in K. Also, when the bandwidth of the Gaussian RBF kernel is small, the kernel matrix can be well approximated by a sparse matrix.

Low rank Whenever the kernel matrix is (approximately) low rank, one can write $K \approx AA^\top$ where $A \in \mathbb{R}^{n \times p}$ can be found through an incomplete Cholesky decomposition in $O(np^2)$ operations. The complexity of each conjugate iteration is then $O(np)$. This idea has been used in (Fine and Scheinberg, 2001) in the context of SVM training, but the authors considered only dual optimization. Note that the kernel matrix is usually low rank when the bandwidth of the Gaussian RBF kernel is large.

Fast Multipole Methods Generalizing both cases above, Fast Multipole methods and KD-Trees provide an efficient way of computing the multiplication of an RBF kernel matrix with a vector (Greengard and Rokhlin, 1987; Gray and Moore, 2000; Yang et al., 2004; de Freitas et al., 2005; Shen et al., 2006). These methods have been successfully applied to Kernel Ridge Regression and Gaussian Processes, but do not seem to be able to handle high dimensional data. See (Lang et al., 2005) for an empirical study of time and memory requirement of these methods.

2.6.2 Reduced expansion

Instead of optimizing on a vector $\boldsymbol{\beta}$ of length n, one can choose a small subset of the training points to expand the solution on and optimize only those weights. More precisely, the same objective function (2.9) is considered, but unlike (2.11), it is optimized on the subset of the functions expressed as

$$f(\mathbf{x}) = \sum_{i \in S} \beta_i k(\mathbf{x}_i, \mathbf{x}), \tag{2.20}$$

where S is a subset of the training set. This approach has been pursued in chapter 3 where the set S is greedily constructed and in (Lee and Mangasarian, 2001a) where S is selected randomly. If S contains k elements, these methods have a complexity of $O(nk^2)$ and a memory requirement of $O(nk)$.

2.6.3 Model selection

Another advantage of primal optimization is when some hyperparameters are optimized on the training cost function (Chapelle et al., 2002; Grandvalet and Canu, 2002). If $\boldsymbol{\theta}$ is a set of hyperparameters and $\boldsymbol{\alpha}$ the dual variables, the standard way of learning $\boldsymbol{\theta}$ is to solve a min max problem (remember that the maximum of the

dual is equal to the minimum of the primal):

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\alpha}} \quad \text{Dual}(\boldsymbol{\alpha}, \boldsymbol{\theta}),$$

by alternating between minimization on $\boldsymbol{\theta}$ and maximization on $\boldsymbol{\alpha}$ (see for instance (Grandvalet and Canu, 2002) for the special case of learning scaling factors). But if the primal is minimized, a *joint optimization* on $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ can be carried out, which is likely to be much faster.

Finally, to compute an approximate *leave-one-out* error, the matrix $K_{\mathsf{sv}} + \lambda I_{n_{\mathsf{sv}}}$ needs to be inverted (Chapelle et al., 2002); but after a Newton optimization, this inverse is already available in (2.17).

2.7 Conclusion

In this paper, we have studied the primal optimization of non-linear SVMs and derived the update rules for a Newton optimization. From these formulae, it appears clear that there are strong similarities between primal and dual optimization. Also, the corresponding *implementation is very simple* and does not require any optimization libraries.

The historical reasons for which most of the research in the last decade has been about dual optimization are unclear. We believe that it is because SVMs were first introduced in their hard margin formulation (Boser et al., 1992), for which a dual optimization (because of the constraints) seems more natural. In general, however, soft margin SVMs should be preferred, even if the training data are separable: the decision boundary is more robust because more training points are taken into account (Chapelle et al., 2000).

We do not pretend that primal optimization is better in general; our main motivation was to point out that primal and dual are two sides of the same coin and that there is no reason to look always at the same side. And by looking at the primal side, some new algorithms for finding approximate solutions emerge naturally. We believe that an approximate primal solution is in general superior to a dual one since an approximate dual solution can yield a primal one which is arbitrarily bad.

In addition to all the possibilities for approximate solutions mentioned in this paper, the primal optimization also offers the advantage of tuning the hyperparameters simultaneously by performing a conjoint optimization on parameters and hyperparameters.

Acknowledgments

We are grateful to Adam Kowalczyk and Sathiya Keerthi for helpful comments.

Appendix

2.A Primal suboptimality

Let us define the following quantities,

$$\begin{aligned} A &= \mathbf{y}^\top \mathbf{y} \\ B &= \mathbf{y}^\top X X^\top \mathbf{y} \\ C &= \mathbf{y}^\top X X^\top X X^\top \mathbf{y} \end{aligned}$$

After one gradient step with exact line search on the primal objective function, we have $\mathbf{w} = \frac{B}{C+\lambda B} X^\top \mathbf{y}$, and the primal value (2.2) is

$$\frac{1}{2} A - \frac{1}{2} \frac{B^2}{C + \lambda B}.$$

For the dual optimization, after one gradient step, $\boldsymbol{\alpha} = \frac{\lambda A}{B+\lambda A}\mathbf{y}$and by (2.6), $\mathbf{w} = \frac{A}{B+\lambda A} X^\top \mathbf{y}$. The primal value is then

$$\frac{1}{2} A + \frac{1}{2} \left(\frac{A}{B + \lambda A} \right)^2 (C + \lambda B) - \frac{AB}{B + \lambda A}.$$

The difference between these two quantities is

$$\frac{1}{2} \left(\frac{A}{B + \lambda A} \right)^2 (C + \lambda B) - \frac{AB}{B + \lambda A} + \frac{1}{2} \frac{B^2}{C + \lambda B} = \frac{1}{2} \frac{(B^2 - AC)^2}{(B + \lambda A)^2 (C + \lambda B)} \geq 0.$$

This proves that if ones does only one gradient step, one should do it on the primal instead of the dual, because one will get a lower primal value this way.

Now note that by the Cauchy-Schwarz inequality $B^2 \leq AC$, and there is equality only if $XX^\top \mathbf{y}$ and $\mathbf{y}$ are aligned. In that case the above expression is zero: the primal and dual steps are as efficient. That is what happens on the left side of Figure 2.1: when $n \ll d$, since X has been generated according to a Gaussian distribution, $XX^\top \approx dI$ and the vectors $XX^\top \mathbf{y}$ and $\mathbf{y}$ are almost aligned.

2.B Optimization with an offset

We now consider a joint optimization on $\begin{pmatrix} b \\ \boldsymbol{\beta} \end{pmatrix}$ of the function

$$f(\mathbf{x}) = \sum_{i=1}^{n} \beta_i k(\mathbf{x}_i, \mathbf{x}) + b.$$

The augmented Hessian (cf (2.15)) is

$$2 \begin{pmatrix} 1^\top I^0 1 & 1^\top I^0 K \\ K I^0 1 & \lambda K + K I^0 K \end{pmatrix},$$

where 1 should be understood as a vector of all 1.

This can be decomposed as

$$2\begin{pmatrix} -\lambda & 1^\top \\ 0 & K \end{pmatrix}\begin{pmatrix} 0 & 1^\top \\ I^0 1 & \lambda I + I^0 K \end{pmatrix}.$$

Now the gradient is

$$\nabla = H\beta - 2\begin{pmatrix} 1^\top \\ K \end{pmatrix} I^0 Y$$

and the equivalent of the update equation (2.16) is

$$\begin{pmatrix} 0 & 1^\top \\ I^0 1 & \lambda I + I^0 K \end{pmatrix}^{-1}\begin{pmatrix} -\lambda & 1^\top \\ 0 & K \end{pmatrix}^{-1}\begin{pmatrix} 1^\top \\ K \end{pmatrix} I^0 Y = \begin{pmatrix} 0 & 1^\top \\ I^0 1 & \lambda I + I^0 K \end{pmatrix}^{-1}\begin{pmatrix} 0 \\ I^0 Y \end{pmatrix}$$

So instead of solving (2.17), one solves

$$\begin{pmatrix} b \\ \boldsymbol{\beta}_{\mathsf{sv}} \end{pmatrix} = \begin{pmatrix} 0 & 1^\top \\ 1 & \lambda I_{n_{\mathsf{sv}}} + K_{\mathsf{sv}} \end{pmatrix}^{-1}\begin{pmatrix} 0 \\ Y_{\mathsf{sv}} \end{pmatrix}.$$

References

Gökhan Bakır, Léon Bottou, and Jason Weston. Breaking SVM complexity with cross training. In *Proceedings of the* 17^{th} *Neural Information Processing Systems Conference*, 2005.

Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. Technical report, NEC Research Institute, Princeton, 2005. `http://leon.bottou.com/publications/pdf/huller3.pdf`.

Bernhard Boser, Isabelle Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Computational Learing Theory*, pages 144–152, 1992.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir N. Vapnik. Vicinal risk minimization. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

Olivier Chapelle, Vladimir N. Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002.

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

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

Nando de Freitas, Yang Wang, Maryam Mahdaviani, and Dustin Lang. Fast krylov methods for n-body learning. In *NIPS*, 2005.

Greg Fasshauer. Meshfree methods. In M. Rieth and W. Schommers, editors, *Handbook of Theoretical and Computational Nanotechnology*. American Scientific Publishers, 2005.

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

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

Yves Grandvalet and Stéphane Canu. Adaptive scaling for feature selection in SVMs. In *Neural Information Processing Systems*, volume 15, 2002.

Alexander Gray and Andrew W. Moore. 'n-body' problems in statistical learning. In *NIPS*, 2000.

Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

Thorsten Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, Massachussetts, 1999.

S. Sathiya Keerthi and Dennis M. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

George S. Kimeldorf and Grace Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *Annals of Mathematical Statistics*, 41:495–502, 1970.

Dustin Lang, Mike Klaas, and Nando de Freitas. Empirical testing of fast kernel density estimation algorithms. Technical Report UBC TR-2005-03, University of British Columbia,, 2005.

Yuh-Jye Lee and Olvi L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the SIAM International Conference on Data Mining*. SIAM, Philadelphia, 2001a.

Yuh-Jye Lee and Olvi L. Mangasarian. SSVM: A smooth support vector machine for classification. *Computational Optimization and Applications*, 20(1):5–22, 2001b.

O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001.

Olvi L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17:913–929, 2002.

Cheng Soon Ong. *Kernels: Regularization and Optimization*. PhD thesis, The Australian National University, 2005.

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

John Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learninng*. MIT Press, 1998.

Robert Schaback. Creating surfaces from scattered data using radial basis functions. In M. Daehlen, T. Lyche, and L. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, pages 477–496. Vanderbilt University Press, 1995.

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

Yirong Shen, Andrew Ng, and Matthias Seeger. Fast gaussian process regression using kd-trees. In *Advances in Neural Information Processing Systems 18*, 2006.

Jonathan R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

Ingo Steinwart. Sparseness of support vector machines. *Journal of Machine Learning Research*, 4:1071–1105, 2003.

Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: fast SVM training on very large datasets. *Journal of Machine Learning Research*, 6: 363–392, 2005.

Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

Changjiang Yang, Ramani Duraiswami, and Larry Davis. Efficient kernel machines using the improved fast gauss transform. In *Advances in Neural Information Processing Systems 16*, 2004.

Jian Zhang, Rong Jin, Yiming Yang, and Alexander G. Hauptmann. Modified logistic regression: An approximation to SVM and its applications in large-scale text categorization. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics*, 14:185–205, 2005.