7

Implementation Techniques

In the previous chapter we showed how the training of a Support Vector Machine can be reduced to maximising a convex quadratic form subject to linear constraints. Such convex quadratic programmes have no local maxima and their solution can always be found efficiently. Furthermore this dual representation of the problem showed how the training could be successfully effected even in very high dimensional feature spaces. The problem of minimising differentiable functions of many variables has been widely studied, especially in the convex case, and most of the standard approaches can be directly applied to SVM training. However, in many cases specific techniques have been developed to exploit particular features of this problem. For example, the large size of the training sets typically used in applications is a formidable obstacle to a direct use of standard techniques, since just storing the kernel matrix requires a memory space that grows quadratically with the sample size, and hence exceeds hundreds of megabytes even when the sample size is just a few thousand points.

Such considerations have driven the design of specific algorithms for Support Vector Machines that can exploit the sparseness of the solution, the convexity of the optimisation problem, and the implicit mapping into feature space. All of these features help to create remarkable computational efficiency. The elegant mathematical characterisation of the solutions can be further exploited to provide stopping criteria and decomposition procedures for very large datasets.

In this chapter we will briefly review some of the most common approaches before describing in detail one particular algorithm, Sequential Minimal Optimisation (SMO), that has the additional advantage of not only being one of the most competitive but also being simple to implement. As an exhaustive discussion of optimisation algorithms is not possible here, a number of pointers to relevant literature and on-line software is provided in Section 7.8.

7.1 General Issues

The optimisation problem associated to classification Support Vector Machines can be written as follows:

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

$$\begin{array}{ll} \text{maximise} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} & \sum_{i=1}^{\ell} \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C,\ i = 1, \ldots, \ell, \end{array} \tag{7.1}$$

where $C = \infty$ gives the hard margin case and with an adaptation of the kernel function the 2-norm soft margin optimisation, while $C < \infty$ gives the 1-norm soft margin case. In the regression case, the problem for the linear ε-insensitive loss is

$$\begin{array}{ll} \text{maximise} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} y_i \alpha_i - \varepsilon \sum_{i=1}^{\ell} |\alpha_i| - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} & \sum_{i=1}^{\ell} \alpha_i = 0,\ -C \leq \alpha_i \leq C,\ i = 1, \ldots, \ell, \end{array} \tag{7.2}$$

where setting $C = \infty$ and adding a constant to the diagonal of the kernel matrix delivers the quadratic ε-insensitive loss optimisation.

The convexity of the functional $W(\boldsymbol{\alpha})$ and of the feasible region ensure that the solution can always be found efficiently. The solution satisfies the Karush–Kuhn–Tucker complementarity conditions. For the classification maximal margin case they are

$$\alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 \right] = 0, \quad i = 1, \ldots, \ell.$$

For the 2-norm soft margin optimisation they are

$$\alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right] = 0, \quad i = 1, \ldots, \ell,$$

while for the 1-norm soft margin optimisation

$$\begin{array}{ll} \alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right] = 0, & i = 1, \ldots, \ell, \\ (\alpha_i - C) \xi_i = 0, & i = 1, \ldots, \ell. \end{array}$$

In the regression case for the quadratic ε-insensitive loss function they are

$$\begin{array}{ll} \alpha_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i - \varepsilon - \xi_i \right) = 0, & i = 1, \ldots, \ell, \\ \hat{\alpha}_i \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b - \varepsilon - \hat{\xi}_i \right) = 0, & i = 1, \ldots, \ell, \\ \xi_i \hat{\xi}_i = 0,\ \alpha_i \hat{\alpha}_i = 0, & i = 1, \ldots, \ell, \end{array}$$

and for the linear ε-insensitive loss function they are

$$\begin{array}{ll} \alpha_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i - \varepsilon - \xi_i \right) = 0, & i = 1, \ldots, \ell, \\ \hat{\alpha}_i \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b - \varepsilon - \hat{\xi}_i \right) = 0, & i = 1, \ldots, \ell, \\ \xi_i \hat{\xi}_i = 0,\ \alpha_i \hat{\alpha}_i = 0, & i = 1, \ldots, \ell, \\ (\alpha_i - C) \xi_i = 0,\ (\hat{\alpha}_i - C) \hat{\xi}_i = 0, & i = 1, \ldots, \ell, \end{array}$$

As discussed below, in practice such conditions will only be approximated to a certain tolerance level. Most numerical strategies follow the approach of starting from an arbitrary feasible point and iteratively increasing the value of the dual objective function without leaving the feasible region, until a stopping criterion is satisfied. Additionally, some approaches implement heuristics that

achieve this by acting only on a small subset of the α_i at each time, in order to improve the computational efficiency. Such techniques, which somehow exploit the sparseness of the problem, make it possible to scale to very large datasets (tens or hundreds of thousands of examples).

Stopping criteria can be obtained by exploiting the properties of convex optimisation problems in different ways. Based on the fact that the feasibility gap vanishes at the solution, one can monitor convergence by checking this quantity. Alternatively, one can stop when the increase in the dual objective function is less than a certain pre-fixed threshold. Finally, the Karush–Kuhn–Tucker conditions can be explicitly calculated and monitored in order to determine whether the solution has been found. We describe these three stopping criteria in more detail.

1. Monitoring the growth of the dual objective function. The quadratic dual objective function for the particular SVM optimisation problem achieves its maximum at the solution. Monitoring the value of the function, and especially the increase in that value at each step, provides the simplest stopping criterion. The training can be stopped when the fractional rate of increase of the objective function $W(\boldsymbol{\alpha})$ falls below a given tolerance (e.g. 10^{-9}). Unfortunately, this criterion has been shown to be unreliable and in some cases can deliver poor results.

2. Monitoring the Karush–Kuhn–Tucker conditions for the primal problem. They are necessary and sufficient conditions for convergence, so they provide the natural criterion. For example in the classification case the following criteria must be checked for the 1-norm soft margin optimisation:

$$0 \leq \alpha_i \leq C,$$

$$y_i f(\mathbf{x}_i) \begin{cases} \geq 1 & \text{for points with } \alpha_i = 0, \\ = 1 & \text{for points with } 0 < \alpha_i < C, \\ \leq 1 & \text{for points with } \alpha_i = C. \end{cases}$$

Notice that one does not need to compute the slack variables ξ_i for this case. For the 2-norm soft margin optimisation the criteria are:

$$\alpha_i \geq 0,$$

$$y_i f(\mathbf{x}_i) \begin{cases} \geq 1 & \text{for points with } \alpha_i = 0, \\ = 1 - \alpha_i/C & \text{for points with } \alpha_i > 0. \end{cases}$$

In this case the slack variable is implicitly defined by $\xi_i = \alpha_i/C$. Naturally these criteria must again be verified to within some chosen tolerance, for example a good choice in this case is to within 10^{-2}.

3. Another way to characterise the solution is by means of the gap between the primal and dual objective functions, since this vanishes only at the optimal point. We call this difference the feasibility gap to distinguish it from the duality gap of an optimisation problem, which is the difference between the values of the primal and dual solutions. For convex quadratic

optimisation problems this difference is zero. In the case of the 1-norm soft margin optimisation the feasibility gap can be computed as described in Subsection 6.1.2. We first set

$$\xi_i = \max\left(0, 1 - y_i\left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b\right)\right),$$

where $\boldsymbol{\alpha}$ is the current estimate for the dual problem and b has been chosen so that $y_i f(\mathbf{x}_i) = 1$ for some i with $C > \alpha_i > 0$. The difference between primal and dual objectives is then given by

$$\sum_{i=1}^{\ell} \alpha_i \left[y_i \left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) \right) - 1 \right] + C \sum_{i=1}^{\ell} \xi_i = \\ = \sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C \sum_{i=1}^{\ell} \xi_i,$$

where $W(\boldsymbol{\alpha})$ is the dual objective. The ratio

$$\begin{aligned} \frac{\text{primal obj.} - \text{dual obj.}}{\text{primal objective} + 1} &= \frac{\sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C \sum_{i=1}^{\ell} \xi_i}{W(\boldsymbol{\alpha}) + \sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C \sum_{i=1}^{\ell} \xi_i + 1} \\ &= \frac{\sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C \sum_{i=1}^{\ell} \xi_i}{\sum_{i=1}^{\ell} \alpha_i - W(\boldsymbol{\alpha}) + C \sum_{i=1}^{\ell} \xi_i + 1} \end{aligned} \tag{7.3}$$

provides a useful measure of progress, and checking if this is less than say 10^{-3} can be used as a stopping criterion.

For the maximal margin and 2-norm margin slack optimisation (implemented by adding a constant to the diagonal of the kernel matrix) we can imagine introducing a box constraint equal to the largest α_i at each iteration t, $C_t = \max_i(\alpha_i^t) + 1$. This will mean that the actual running of the algorithm will not be affected, but at any iteration the current $\boldsymbol{\alpha}$ can be regarded as a feasible solution of the corresponding box constraint optimisation. For this problem we can compute the feasibility gap using the above equations and for large enough $C > \max_i(\alpha_i^*)$, where $\boldsymbol{\alpha}^*$ is the optimal solution of the unconstrained problem, the two solutions coincide.

Remark 7.1 Note that the tolerance level used to verify the stopping criteria is very important. Achieving high levels of accuracy can be very time consuming but may not give any significant advantage in prediction accuracy. So in practice one needs to set tolerance levels to ensure approximately optimal conditions.

Remark 7.2 An important consequence of the stopping criteria is that they can also motivate heuristics to speed convergence: for example one can expect to approach the solution faster by acting on points that contribute more to the

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

feasibility gap, for example those for which

$$\alpha_i \left[y_i \left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) \right) - 1 \right] + C\xi_i$$

is large, or by performing operations that lead to a larger increase in the dual objective function. These considerations will be exploited in the following sections to design more effective training algorithms.

Remark 7.3 It should be noted that even if the solution $\mathbf{w}$ is unique, its expansion in terms of $\boldsymbol{\alpha}$ may not be if the kernel matrix is only positive semi-definite.

In the rest of the chapter, we will first discuss a simple gradient ascent algorithm that does not optimise the bias. This will provide a vehicle for introducing a series of important concepts that will be needed for the more sophisticated approaches in the later sections.

7.2 The Naive Solution: Gradient Ascent

The simplest numerical solution of a convex optimisation problem is obtained by gradient ascent, sometimes known as the steepest ascent algorithm. The algorithm starts with an initial estimate for the solution, denoted by $\boldsymbol{\alpha}^0$, and then iteratively updates the vector following the steepest ascent path, that is moving in the direction of the gradient of $W(\boldsymbol{\alpha})$ evaluated at the position $\boldsymbol{\alpha}^t$ for update $t+1$. At each iteration the direction of the update is determined by the steepest ascent strategy but the length of the step still has to be fixed. The length of the update is known as the *learning rate*.

In the sequential or *stochastic* version, this strategy is approximated by evaluating the gradient for just one pattern at a time, and hence updating a single component α_i^t by the increment

$$\delta\alpha_i^t = \eta \frac{\partial W(\boldsymbol{\alpha}^t)}{\partial \alpha_i}$$

where the parameter η is the learning rate. If η is chosen carefully, the objective function will increase monotonically, and the average direction approximates the local gradient. One can modify η as a function of time, or as a function of the input pattern being learned, in order to improve the convergence. The choice of η is a delicate one; too large values can cause the system to oscillate without converging to the solution, while too small values result in very slow rates of convergence. Following this strategy means that the direction of each step is parallel to one of the ℓ axes and so is known a priori. The algorithm determines the step length and more importantly its sign. One further freedom available is the choice of the ordering in which the points are updated, as some points may cause a greater movement along the path towards a solution. This will be discussed further below.

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

Another way of looking at the same algorithm is that the quantity $W(\boldsymbol{\alpha})$ is iteratively increased by freezing all variables but one. Hence a multi-dimensional problem is reduced to a sequence of one dimensional ones. The uniqueness of the global maximum guarantees that for suitable choices of η the algorithm will always find the solution. Such a strategy is usually not optimal from the point of view of speed, but is surprisingly good for datasets of up to a couple of thousand points and has the advantage that its implementation is very straightforward. As we will see later, another major advantage is that it only acts on one training point at a time, and so does not require that we store all the data simultaneously in fast memory.

Using this approach, one source of problems is the linear constraint

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{7.4}$$

derived from optimising the bias b in the decision function. This constraint defines a hyperplane in the ℓ dimensional space of parameters, and restricts the feasible region to the intersection between this hyperplane and the hypercube of side C for the 1-norm soft margin optimisation or the positive orthant in the case of the maximal margin and 2-norm soft margin optimisation. A natural strategy for enforcing such a constraint is to make sure that the current solution never leaves the feasible region. Unfortunately, updating one component at a time makes this impossible: if at the time t the constraint of equation (7.4) is satisfied, then after performing a non-trivial update on one α_i it will cease to hold. The minimum number of multipliers that can be simultaneously updated without leaving the feasible region is hence 2, and this consideration will form the basis of the SMO algorithm, described later, in Section 7.5. Other strategies also exist, such as for example enforcing the constraint

$$-\zeta_t \leq \sum_{i=1}^{\ell} \alpha_i^t y_i \leq \zeta_t,$$

where ζ_t is decreased at each iteration. In this section we will discuss the algorithm for the case where the bias b is fixed a priori and hence the equality constraint does not need to be enforced.

Remark 7.4 Fixing the bias beforehand may seem restrictive, but if we consider the embedding of the input space X into a space $\hat{X}$ of one extra dimension, in which we denote the new vector by $\hat{\mathbf{x}} = (\mathbf{x}, \tau)$, for some fixed value τ, then the linear function on X represented by a unit weight vector $\mathbf{w}$ and bias b is equivalent to the function with weight vector $\hat{\mathbf{w}} = (\mathbf{w}, b/\tau)$ and zero bias in the space $\hat{X}$, since

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \langle \hat{\mathbf{w}} \cdot \hat{\mathbf{x}} \rangle .$$

Note that for a non-trivial classification of a training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

we must have $b \leq R = \max_{1 \leq i \leq \ell} \left(\|\mathbf{x}_i\| \right)$. Adding this extra dimension to a feature space induced by the kernel K is equivalent to adding τ^2 to create the adapted kernel

$$\hat{K}(\mathbf{x}, \mathbf{z}) = K(\mathbf{x}, \mathbf{z}) + \tau^2.$$

The only drawback of taking this route is that the geometric margin of the separating hyperplane in the augmented space will typically be less than that in the original space. For poor choices of τ this difference can be very large, affecting both the convergence of the algorithm and generalisation performance of the resulting classifier. Assuming that $\mathbf{w}$ is normalised, the functional margin of the new weight vector $\hat{\mathbf{w}}$ on the set $\hat{S}$ is equal to the geometric margin γ of $\mathbf{w}$ on S. Hence the geometric margin of $\hat{\mathbf{w}}$ is

$$\begin{aligned} \gamma \|\hat{\mathbf{w}}\|^{-1} &= \gamma \left(1 + b^2/\tau^2\right)^{-1/2} \\ &\leq \gamma \left(1 + R^2/\tau^2\right)^{-1/2}. \end{aligned}$$

The quantity that measures the fat-shattering dimension in the space X (see Theorem 4.16 in Chapter 4) is the ratio

$$\frac{\max_{1 \leq i \leq \ell} \left(\|\mathbf{x}_i\|^2 \right)}{\gamma^2} = \frac{R^2}{\gamma^2},$$

while in $\hat{X}$ this ratio becomes

$$\frac{\max_{1 \leq i \leq \ell} \left(\|\mathbf{x}_i\|^2 \right) + \tau^2}{\gamma^2 \left(1 + R^2/\tau^2\right)^{-1}} = \frac{\left(R^2 + \tau^2\right)\left(1 + R^2/\tau^2\right)}{\gamma^2}.$$

The right hand side of the inequality is minimised by taking $\tau = R$, when it becomes $4R^2/\gamma^2$. Hence, a safe choice of τ is R, in that it will only increase the bound on the fat-shattering dimension by a factor of 4.

If we set the bias to a fixed value then the dual of the optimisation problem becomes

$$\begin{array}{ll} \text{maximise} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} & 0 \leq \alpha_i \leq C, \; i = 1, \ldots, \ell. \end{array}$$

The ith component of the gradient of $W(\boldsymbol{\alpha})$ is

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

so one can maximise $W(\boldsymbol{\alpha})$ simply by iterating the update rule

$$\alpha_i \leftarrow \alpha_i + \eta \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}$$

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

Given training set S and learning rates $\boldsymbol{\eta} \in (\mathbb{R}^+)^\ell$
$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
repeat
  for $i = 1$ to ℓ
    $\alpha_i \leftarrow \alpha_i + \eta_i \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$
    if $\alpha_i < 0$ then $\alpha_i \leftarrow 0$
    else
    if $\alpha_i > C$ then $\alpha_i \leftarrow C$
  end for
until stopping criterion satisfied
return $\boldsymbol{\alpha}$

Table 7.1: **Simple on-line algorithm for the 1-norm soft margin**

with a suitable learning rate η, and maintaining simultaneously all the α_i in the positive orthant. This can be enforced by resetting $\alpha_i \leftarrow 0$ if it becomes negative, that is performing the update

$$\alpha_i \leftarrow \max\left(0, \alpha_i + \eta \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}\right).$$

Similarly, the upper bound on the α_i due to the soft margin technique can be enforced by setting $\alpha_i \leftarrow C$ each time a multiplier becomes larger than C, that is performing the update

$$\alpha_i \leftarrow \min\left(C, \max\left(0, \alpha_i + \eta \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}\right)\right),$$

which is also known as the 'projection method'.

The resulting simple algorithm for training SVMs in the non-bias case is shown in Table 7.1. Note that each training example has been given its own learning rate η_i.

The algorithm fails to implement strict gradient ascent in two ways. Firstly, using a different learning rate for each training example biases the gradient direction. Secondly, if we wish to implement the gradient strictly, we should create a new vector $\boldsymbol{\alpha}^{new}$ inside the *for* loop and then set $\boldsymbol{\alpha} = \boldsymbol{\alpha}^{new}$ after completion of the loop. In practice it is convenient and efficient to use the new values of α_i as soon as they have been obtained. This approach is known as stochastic gradient ascent and is related to successive overrelaxation techniques for solving matrix equations. As a stopping criterion one can use any of the methods given above, namely monitoring the Karush–Kuhn–Tucker conditions, or the feasibility gap, or simply the rate of increase of the objective function $W(\boldsymbol{\alpha})$. Indeed, the stationarity conditions for the algorithm correspond to the KKT conditions of the problem. It is possible to show that for suitable choices of η_i the algorithm converges. More precisely, since we are optimising a quadratic function of α_i the

update that causes the derivative to go to zero is

$$\hat{\alpha}_i \leftarrow \alpha_i + \frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)} \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$$

since

$$\begin{aligned} \frac{\partial W(\hat{\boldsymbol{\alpha}})}{\partial \alpha_i} &= 1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &\quad - y_i y_i K(\mathbf{x}_i, \mathbf{x}_i) \frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)} \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right) \\ &= 0. \end{aligned}$$

Hence, provided $0 \leq \hat{\alpha}_i \leq C$ the maximal gain is made by choosing

$$\eta_i = \frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)},$$

while a sufficient condition for convergence is $0 < \eta_i K(\mathbf{x}_i, \mathbf{x}_i) < 2$. When the update is constrained by the boundary of the feasible region, the step length is shorter and the gain correspondingly smaller, but still positive. Despite this apparent unreliability it can be shown that using the above value for η_i there exist constants μ, $\delta \in (0, 1)$, and $\tau \in (0, 1)$, such that

$$\left\|\boldsymbol{\alpha}^t - \boldsymbol{\alpha}^*\right\| \leq \mu \delta^t,$$

and

$$W\left(\boldsymbol{\alpha}^*\right) - W\left(\boldsymbol{\alpha}^{t+1}\right) \leq \tau \left(W\left(\boldsymbol{\alpha}^*\right) - W\left(\boldsymbol{\alpha}^t\right)\right).$$

Such rates of convergence are reassuring, though in practice the performance of the algorithm can vary very significantly. One important way in which the convergence rate can be improved is to exploit the freedom to vary the order in which the parameters are updated. Table 7.1 shows the parameters being updated sequentially but clearly the order can be varied from one cycle to the next or indeed the update can be made iteratively on any point provided points are not overlooked. Clearly, if we can identify points that will give significant increases to the dual objective function, it will be preferable to choose them first. This suggests the heuristic of choosing from among those points that violate the Karush–Kuhn–Tucker conditions. The outer loop of the algorithm goes through the training set looking for points that violate KKT conditions and selects any it finds for update. In order to increase the chances of finding KKT violations, the outer loop first considers the points for which the corresponding parameter α_i satisfies $0 < \alpha_i < C$ implying that its value is not on the boundary of the feasible region, and only when all such points satisfy the KKT conditions to the specified tolerance level is a complete loop through all the training set again undertaken. A simpler variant of this approach is described in Remark 7.8.

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

Remark 7.5 The algorithm which optimises one dual variable at a time is known in the optimisation literature as Hildreth's method, while in machine learning it is often referred to as kernel-Adatron, because ignoring the use of kernels it is equivalent to the Adatron algorithm for training single neurons.

Remark 7.6 A recently developed variant of the on-line gradient algorithm uses an additional parameter $\omega \in (0,2)$ to implement successive over-relaxation, giving the update

$$\alpha_i \leftarrow \alpha_i + \frac{\omega}{K(\mathbf{x}_i, \mathbf{x}_i)} \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$$

before clipping, though in the experiments with this approach ω was chosen equal to 1, equivalent to choosing $\eta_i = (K(\mathbf{x}_i, \mathbf{x}_i))^{-1}$, in the algorithm of Table 7.1.

Remark 7.7 One can reduce the training time by caching the values of the kernel matrix $K(\mathbf{x}_i, \mathbf{x}_j)$ during initialisation, since these values are used repeatedly and the kernel can be a costly function to evaluate. This approach to a certain extent sacrifices the inherently on-line character of the algorithm, but is practical if the training set size is moderate. It is adopted in most of the standard techniques described in the following sections, sometimes in conjunction with the subset-selection methods discussed in Section 7.4. For larger datasets one can re-evaluate the kernels at every iteration, reducing the space complexity but increasing the running time. Stochastic gradient ascent is typically used in practice for moderate sample sizes (see Chapter 8 for an example) though, combined with sample selection heuristics, it can also prove effective for massive datasets where other algorithms would fail anyway. We describe one heuristic that proved successful for datasets of one million points in the next remark.

Remark 7.8 Heuristics for choosing the order in which the data points are processed can have a very significant impact on the rate of convergence. A particularly simple ranking is to use the size of the parameters α_i. This is combined with the two-stage strategy described above of optimising on the current support vectors and, only when this optimum is reached, again considering points with $\alpha_i = 0$. This suggests only holding those parameters with non-zero α_i in memory. These are the current estimates of the support vectors. They are repeatedly processed in order of ascending size of α_i until no further progress is observed in the objective. At this point an iteration through the rest of the dataset is undertaken, which produces a new set of support vectors to begin the cycle over again. We give references in Section 7.8 to papers reporting the successful application of this method to solving the Support Vector optimisation of one million points.

Remark 7.9 Notice that the perceptron algorithm described in Chapter 2 can be regarded as a gradient descent algorithm. The cost function is $\sum_{i=1}^{\ell} \xi_i$ where ξ_i

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

is defined as $\max\left(0, -y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b)\right)$. This function is sometimes referred to as the hinge loss and is equal to the linear ε-insensitive loss with $\varepsilon = 0$. An inclusive theory of gradient descent algorithms has been developed in recent years, see Section 7.8 for more information.

Remark 7.10 The algorithm described above solves the 1-norm soft margin optimisation problem. Clearly the maximal margin algorithm is recovered if we ignore the $\alpha_i \leq C$ constraint. If we wish to solve the 2-norm soft margin optimisation, we simply add a diagonal shift $\mathbf{K} \leftarrow \mathbf{K} + \frac{1}{C}\mathbf{I}$ to the kernel matrix.

Of course the simple procedures outlined in this section suffer from many of the problems associated with naive gradient ascent: they can be extremely slow on some datasets; they can oscillate before converging; and so on. However, their conceptual and computational simplicity makes them ideal candidates for a first implementation of Support Vector Machines, and also for small size applications. More advanced techniques are discussed in the following sections.

7.3 General Techniques and Packages

A number of optimisation techniques have been devised over the years, and many of them can be directly applied to quadratic programmes. The Newton method, conjugate gradient, primal dual interior-point methods, not only can be straightforwardly applied to the case of Support Vector Machines, but given the specific structure of the objective function also can be considerably simplified. Conceptually they are not very different from the simple gradient ascent strategy described above, as they all iteratively climb the objective function to its maximum, but they are all likely to be more efficient, essentially because the direction and the length of each step are chosen in a more sophisticated way. Some care should however be paid to the computational issues arising from the size of the problem. Many of them require that the kernel matrix is stored in memory, implying that the space complexity is quadratic in the sample size. For large size problems, these approaches can be inefficient, and should therefore be used in conjunction with the decomposition techniques described in Section 7.4. As it is impossible to describe the plethora of methods in this chapter, pointers to surveys will be given in Section 7.8.

One of the main advantages of such techniques is that they are well understood, and widely available in a number of commercial and freeware packages, some also accessible through the Internet. It is these packages that were used for Support Vector Machines before specially tailored algorithms were developed. One of the most common choices is the package MINOS, from the Stanford Optimization Laboratory, which uses a hybrid strategy; another standard choice is LOQO, which uses a primal dual interior-point method. In contrast, the quadratic programme subroutine provided in the MATLAB optimisation toolbox is very general but the routine *quadprog* is significantly better than *qp*. Pointers to these and other packages are provided in Section 7.8.

Finally, a very convenient solution is to use one of the existing Support Vector packages, like SVM^{light} by Joachims, the package of Royal Holloway, University of London, and the others freely available through the Internet. Similarly, packages for Gaussian processes are freely available on the Internet. Pointers to literature and on-line software are provided in Section 7.8.

7.4 Chunking and Decomposition

Techniques like those described in the previous section do not have the on-line flavour of stochastic gradient ascent as they require that the data are held in memory in the form of the kernel matrix. The complexity of the training problem grows with the size of this matrix, limiting the approaches to datasets of a few thousand points.

For larger problems we wish to take advantage of an approach that forms the basis of the so-called 'active set' or 'working set' methods in optimisation: if one knew in advance which constraints were active, it would be possible to discard all of the inactive constraints and simplify the problem. This leads to several strategies, all based on somehow guessing the active set, and restricting training to this guess. Iterative heuristics that gradually build the active set are the most common. Although these techniques are very often heuristics, the fact that at each step they reduce the feasibility gap or increase the dual objective function can be used to guarantee the eventual convergence of the algorithm.

The simplest heuristic is known as *chunking*. It starts with an arbitrary subset or 'chunk' of the data, and trains an SVM using a generic optimiser on that portion of the data. The algorithm then retains the support vectors from the chunk while discarding the other points and then it uses the hypothesis found to test the points in the remaining part of the data. The M points that most violate the KKT conditions (where M is a parameter of the system) are added to the support vectors of the previous problem, to form a new chunk. This procedure is iterated, initialising $\boldsymbol{\alpha}$ for each new sub-problem with the values output from the previous stage, finally halting when some stopping criterion is satisfied. The chunk of data being optimised at a particular stage is sometimes referred to as the working set. Typically the working set grows, though it can also decrease, until in the last iteration the machine is trained on the set of support vectors representing the active constraints. Pseudocode for this algorithm is given in Table 7.2.

This heuristic assumes that the kernel matrix for the set of support vectors fits in memory and can be fed to the optimisation package being used. In general, it can happen that the problem is not sparse, or simply that the size is so large that the set of support vectors is still too large to be dealt with by the optimisation routine. One can still deal with such problems by using the more advanced *decomposition* algorithm, which was inspired by the use of chunking in working set methods. The decomposition algorithm only updates a fixed size subset of multipliers α_i, while the others are kept constant. So every time a new point is added to the working set, another point has to be removed. In this algorithm, the

```
Given training set S
α ← 0
select an arbitary working set Ŝ ⊂ S
repeat
        solve optimisation problem on Ŝ
        select new working set from data not satisfying
                Karush–Kuhn–Tucker conditions
until stopping criterion satisfied
return α
```

Table 7.2: **Pseudocode for the general working set method**

goal is not to identify all of the active constraints in order to run the optimiser on all of them, but is rather to optimise the global problem by only acting on a small subset of data at a time. Also in this system, as in the previous one, the 'nucleus' of the algorithm is provided by some generic quadratic programme optimiser, possibly one of the many available packages.

Although no theoretical proof has been given of the convergence of these methods, in practice they work very well and make it possible to deal with datasets of several tens of thousands of points.

The important point is to select the working set in such a way that the optimisation of the corresponding quadratic programme sub-problem leads to an improvement in the overall objective function. Several heuristics for this can be used. An efficient heuristic for choosing the working set at each step is to use the stopping criteria: for example one could include the points that contribute most to the feasibility gap or equivalently that most violate the Karush–Kuhn–Tucker conditions.

7.5 Sequential Minimal Optimisation (SMO)

The Sequential Minimal Optimisation (SMO) algorithm is derived by taking the idea of the decomposition method to its extreme and optimising a minimal subset of just two points at each iteration. The power of this technique resides in the fact that the optimisation problem for two data points admits an analytical solution, eliminating the need to use an iterative quadratic programme optimiser as part of the algorithm.

The requirement that the condition $\sum_{i=1}^{\ell} \alpha_i y_i = 0$ is enforced throughout the iterations implies that the smallest number of multipliers that can be optimised at each step is 2: whenever one multiplier is updated, at least one other multiplier needs to be adjusted in order to keep the condition true.

At each step SMO chooses two elements α_i and α_j to jointly optimise, finds the optimal values for those two parameters given that all the others are fixed, and updates the $\boldsymbol{\alpha}$ vector accordingly. The choice of the two points is determined by a heuristic, while the optimisation of the two multipliers is performed analytically.

Despite needing more iterations to converge, each iteration uses so few operations that the algorithm exhibits an overall speed-up of some orders of magnitude. Besides convergence time, other important features of the algorithm are that it does not need to store the kernel matrix in memory, since no matrix operations are involved, that it does not use other packages, and that it is fairly easy to implement. Notice that since standard SMO does not use a cached kernel matrix, its introduction could be used to obtain a further speed-up, at the expense of increased space complexity.

7.5.1 Analytical Solution for Two Points

Without loss of generality we will assume that the two elements that have been chosen are α_1 and α_2. In order to compute the new values for these two parameters, one can observe that in order not to violate the linear constraint $\sum_{i=1}^{\ell} \alpha_i y_i = 0$, the new values of the multipliers must lie on a line,

$$\alpha_1 y_1 + \alpha_2 y_2 = \text{ constant } = \alpha_1^{old} y_1 + \alpha_2^{old} y_2,$$

in (α_1, α_2) space, and in the box defined by $0 \leq \alpha_1, \alpha_2 \leq C$. The one dimensional problem resulting from the restriction of the objective function to such a line can be solved analytically.

Without loss of generality, the algorithm first computes α_2^{new} and successively uses it to obtain α_1^{new}. The box constraint $0 \leq \alpha_1, \alpha_2 \leq C$, together with the linear equality constraint, provides a more restrictive constraint on the feasible values for α_2^{new}:

$$U \leq \alpha_2^{new} \leq V$$

where

$$\begin{aligned} U &= \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right), \\ V &= \min\left(C, C - \alpha_1^{old} + \alpha_2^{old}\right), \end{aligned} \tag{7.5}$$

if $y_1 \neq y_2$, and

$$\begin{aligned} U &= \max\left(0, \alpha_1^{old} + \alpha_2^{old} - C\right), \\ V &= \min\left(C, \alpha_1^{old} + \alpha_2^{old}\right), \end{aligned} \tag{7.6}$$

if $y_1 = y_2$.

Remark 7.11 In the following theorem we will use the above definitions of U and V. We also introduce some more notation that will simplify the statement and proof of the theorem. We use $f(\mathbf{x})$ to denote the current hypothesis determined by the values of $\boldsymbol{\alpha}$ and b at a particular stage of learning. Let

$$E_i = f(\mathbf{x}_i) - y_i = \left(\sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b\right) - y_i, \; i = 1, 2, \tag{7.7}$$

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

be the difference between function output and target classification on the training points $\mathbf{x}_1$ or $\mathbf{x}_2$. Note that this may be large even if a point is correctly classified. For example if $y_1 = 1$, and the function output is $f(\mathbf{x}_1) = 5$, the classification is correct, but $E_1 = 4$. A further quantity that we will require is the second derivative of the objective function along the diagonal line, which can be expressed as $-\kappa$, where

$$\kappa = K(\mathbf{x}_1, \mathbf{x}_1) + K(\mathbf{x}_2, \mathbf{x}_2) - 2K(\mathbf{x}_1, \mathbf{x}_2) = \|\boldsymbol{\phi}(\mathbf{x}_1) - \boldsymbol{\phi}(\mathbf{x}_2)\|^2, \tag{7.8}$$

where $\boldsymbol{\phi}(\cdot)$ is the mapping into the feature space.

It is now possible to prove the following theorem.

Theorem 7.12 *The maximum of the objective function for the optimisation problem (7.1), when only α_1 and α_2 are allowed to change, is achieved by first computing the quantity*

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\kappa}$$

and clipping it to enforce the constraint $U \leq \alpha_2^{new} \leq V$:

$$\alpha_2^{new} = \begin{cases} V, & \text{if } \alpha_2^{new,unc} > V, \\ \alpha_2^{new,unc}, & \text{if } U \leq \alpha_2^{new,unc} \leq V, \\ U, & \text{if } \alpha_2^{new,unc} < U, \end{cases}$$

where E_i is given in equation (7.7), κ is given by equation (7.8), and U and V are given by equation (7.5) or (7.6). The value of α_1^{new} is obtained from α_2^{new} as follows:

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2(\alpha_2^{old} - \alpha_2^{new}).$$

Proof Define

$$v_i = \sum_{j=3}^{\ell} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i) - \sum_{j=1}^{2} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - b, \quad i = 1, 2,$$

and consider the objective as a function of α_1 and α_2:

$$\begin{aligned} W(\alpha_1, \alpha_2) \quad = \quad & \alpha_1 + \alpha_2 - \frac{1}{2} K_{11} \alpha_1^2 - \frac{1}{2} K_{22} \alpha_2^2 \\ & - y_1 y_2 K_{12} \alpha_1 \alpha_2 - y_1 \alpha_1 v_1 - y_2 \alpha_2 v_2 + \text{constant}, \end{aligned}$$

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, 2$. Note also that the constraint $\sum_{i=1}^{\ell} \alpha_i^{old} y_i = \sum_{i=1}^{\ell} \alpha_i y_i = 0$ implies the condition

$$\alpha_1 + s\alpha_2 = \text{constant} = \alpha_1^{old} + s\alpha_2^{old} = \gamma,$$

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

where $s = y_1 y_2$. This equation demonstrates how α_1^{new} is computed from α_2^{new}. The objective function along the constraint becomes

$$\begin{aligned} W(\alpha_2) &= \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}K_{11}(\gamma - s\alpha_2)^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ &\quad -sK_{12}(\gamma - s\alpha_2)\alpha_2 - y_1(\gamma - s\alpha_2)v_1 - y_2\alpha_2 v_2 + \text{constant} \end{aligned}$$

and the stationary point satisfies

$$\begin{aligned} \frac{\partial W(\alpha_2)}{\partial \alpha_2} &= 1 - s + sK_{11}(\gamma - s\alpha_2) - K_{22}\alpha_2 \\ &\quad + K_{12}\alpha_2 - sK_{12}(\gamma - s\alpha_2) + y_2 v_1 - y_2 v_2 \\ &= 0. \end{aligned}$$

This yields

$$\begin{aligned} \alpha_2^{new,unc}(K_{11} + K_{22} - 2K_{12}) &= 1 - s + \gamma s(K_{11} - K_{12}) + y_2(v_1 - v_2) \\ &= y_2(y_2 - y_1 + \gamma y_1(K_{11} - K_{12}) + v_1 - v_2). \end{aligned}$$

Hence,

$$\begin{aligned} \alpha_2^{new,unc}\kappa y_2 &= y_2 - y_1 + f(\mathbf{x}_1) - \sum_{j=1}^{2} y_j\alpha_j K_{1j} + \gamma y_1 K_{11} \\ &\quad -f(\mathbf{x}_2) + \sum_{j=1}^{2} y_j\alpha_j K_{2j} - \gamma y_1 K_{12} \\ &= y_2 - y_1 + f(\mathbf{x}_1) - f(\mathbf{x}_2) \\ &\quad +y_2\alpha_2 K_{11} - y_2\alpha_2 K_{12} + y_2\alpha_2 K_{22} - y_2\alpha_2 K_{12} \\ &= y_2\alpha_2\kappa + (f(\mathbf{x}_1) - y_1) - (f(\mathbf{x}_2) - y_2), \end{aligned}$$

giving

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2))}{\kappa}.$$

Finally, we must clip $\alpha_2^{new,unc}$ if necessary to ensure it remains in the interval $[U, V]$. □

7.5.2 Selection Heuristics

In order to speed up the convergence, it is possible to choose the set of two points to optimise, based on their contribution to the general progress towards the solution. If the amount of computation required to implement the selection strategy is smaller than that saved by the reduction in the number of iterations, one obtains a gain in the rate of convergence.

The chosen stopping criterion can give a good indication of which points are more likely to make a larger contribution towards convergence. For example,

if one monitors the feasibility gap, a natural choice is to optimise the points that most violate the KKT conditions, as they contribute most to that gap (see stopping criterion 3 in Section 7.1). Computing the KKT conditions of each point at each iteration is, however, computationally expensive, and cheaper heuristics can deliver a better overall performance.

SMO uses two criteria for selecting the two active points to ensure that the objective function enjoys a large increase from their optimisation. There are two separate heuristics for choosing the first and the second point.

First Choice Heuristic The first point $\mathbf{x}_1$ is chosen from among those points that violate the Karush–Kuhn–Tucker conditions. The outer loop of the algorithm goes through the training set looking for points that violate KKT conditions and selects any it finds for update. When one such point is found, the second heuristic is used to select the second point, and the values of the respective multipliers are updated. Then the outer loop is resumed looking for new KKT violations. In order to increase the chances of finding KKT violations, the outer loop goes through the points for which the corresponding parameter α_i satisfies $0 < \alpha_i < C$ implying that its value is not on the boundary of the feasible region, and only when all such points satisfy the KKT conditions to the specified tolerance level is a complete loop through all the training set again undertaken.

Second Choice Heuristic The second point $\mathbf{x}_2$ must be chosen in such a way that updating on the pair α_1, α_2 causes a large change, which should result in a large increase of the dual objective. In order to find a good point without performing too much computation, a quick heuristic is to choose $\mathbf{x}_2$ to maximise the quantity $|E_1 - E_2|$, where E_i is defined in Theorem 7.12. If E_1 is positive, SMO chooses an example $\mathbf{x}_2$ with minimum error E_2, while if E_1 is negative, SMO maximises the error E_2. A cached list of errors for every non-bound point in the training set is kept to further reduce the computation. If this choice fails to deliver a significant increase in dual objective, SMO tries each non-bound point in turn. If there is still no significant progress SMO looks through the entire training set for a suitable point. The loops through the non-bound points, and the whole training set, start from random locations in the respective lists, so that no bias is introduced towards the examples occurring at the beginning of either of them.

We have included pseudocode for the SMO classification algorithm due to John Platt in Appendix A. Note also that in Section 7.8, pointers are provided to on-line software implementing SMO, as well as to references with a complete and detailed description of SMO.

Remark 7.13 Note that the SMO algorithm makes apparent use of the parameter C, and so at first appears only applicable to the 1-norm soft margin optimisation problem. We can, however, run SMO treating C as infinite, and hence reducing

the constraints on the interval $[U, V]$, which subsequently only provides a lower bound on α_2^{new} of

$$U = \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right)$$

when $y_1 \neq y_2$, and is given by

$$\begin{aligned} U &= 0, \\ V &= \alpha_1^{old} + \alpha_2^{old} \end{aligned}$$

if $y_1 = y_2$.

Remark 7.14 The SMO algorithm makes no provision for choosing the bias, and yet uses it in calculating the values E_i, $i = 1, 2$. This apparent indeterminacy does not change the algorithm since whatever value of b is used both E_is are equally affected and so the resulting update, being determined by their difference, is identical. Hence, b can be taken to be zero throughout the computation and set after convergence using the Karush–Kuhn–Tucker conditions as described at the beginning of this chapter for the particular optimisation being performed. Note, however, that we may need to compute b in order to evaluate the stopping criterion.

Remark 7.15 The stopping criterion 3 in Section 7.1 can be used to assess convergence. Note that this will require setting the bias. As indicated as part of that calculation the b should be chosen by reference to some α_i satisfying $0 < \alpha_i < C$. The original SMO algorithm calculated the bias based on an estimate derived from the updated points. If neither satisfies the required inequalities, this could lead to an over-estimate of the feasibility gap.

Remark 7.16 Notice that SMO is not directly applicable in the fixed bias case, since the choice of α_2^{new} was made using the constraint resulting from the variable bias. For fixed bias the SMO algorithm reduces to the algorithm described in Table 7.1 with

$$\eta_i = (K(\mathbf{x}_i, \mathbf{x}_i))^{-1}.$$

For regression the SMO algorithm can again update α_1 and α_2 from the optimisation problem given in problem (7.2). The equations encode four separate problems depending on the signs of the two parameters. Here, the constraint on the $\boldsymbol{\alpha}$ vector does not involve the classifications and so the interval for α_2 is given by

$$\begin{aligned} U &= \max\left(C_U^2, \alpha_1^{old} + \alpha_2^{old} - C_V^1\right), \\ V &= \min\left(C_V^2, \alpha_1^{old} + \alpha_2^{old} - C_U^1\right), \end{aligned} \tag{7.9}$$

where the four problems each have different settings of the parameters C_U^i and C_V^i that are specified in the following table:

	$\alpha_i \geq 0$	$\alpha_i \leq 0$
C_U^i	0	$-C$
C_V^i	C	0

(7.10)

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

Remark 7.17 In the following theorem we will use the above definitions of U and V. We also introduce some more notation that will simplify the statement and proof of the theorem. We use $f(\mathbf{x})$ to denote the current hypothesis determined by the values of $\boldsymbol{\alpha}$ and b at a particular stage of learning. Let

$$E_i = f(\mathbf{x}_i) - y_i = \left(\sum_{j=1}^{\ell} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b \right) - y_i, \; i = 1, 2, \tag{7.11}$$

be the difference between function output and target value on the training points $\mathbf{x}_1$ or $\mathbf{x}_2$.

It is now possible to prove the following theorem showing how to apply SMO for the regression case. Note that this theorem is effectively giving four update rules, one for each quadrant of the space of current values of α_1 and α_2. It is important that the optimisation is performed and hence the new values are sought within a chosen quadrant containing the current values. If the current values fall in more than one quadrant then the corresponding optimisations can be performed for each quadrant and the one giving greater increase in the objective function chosen.

Theorem 7.18 *The maximum of the objective function for the optimisation problem (7.1), when only α_1 and α_2 are allowed to change in a specified quadrant containing them both, is achieved by first computing the quantity*

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{(E_1 - E_2) - \varepsilon(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1))}{\kappa}$$

and clipping it to enforce the constraint $U \leq \alpha_2^{new} \leq V$:

$$\alpha_2^{new} = \begin{cases} V, & \text{if } \alpha_2^{new,unc} > V, \\ \alpha_2^{new,unc}, & \text{if } U \leq \alpha_2^{new,unc} \leq V, \\ U, & \text{if } \alpha_2^{new,unc} < U, \end{cases}$$

where the values of the sgn *function are determined by the chosen quadrant, E_i is given in equation (7.11), κ is given by equation (7.8), and U and V are given by equation (7.9) or (7.10). The value of α_1^{new} is obtained from α_2^{new} as follows:*

$$\alpha_1^{new} = \alpha_1^{old} + \alpha_2^{old} - \alpha_2^{new}.$$

Proof We define

$$v_i = \sum_{j=3}^{\ell} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i) - \sum_{j=1}^{2} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - b, \; i = 1, 2,$$

and consider the objective as a function of α_1 and α_2:

$$\begin{aligned} W(\alpha_1, \alpha_2) \;=\; & y_1\alpha_1 + y_2\alpha_2 - \varepsilon(|\alpha_1| + |\alpha_2|) - \frac{1}{2}K_{11}\alpha_1^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ & - K_{12}\alpha_1\alpha_2 - \alpha_1 v_1 - \alpha_2 v_2 + \text{constant}, \end{aligned}$$

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, 2$. Note also that the constraint $\sum_{i=1}^{\ell} \alpha_i^{old} = \sum_{i=1}^{\ell} \alpha_i = 0$ implies the condition

$$\alpha_1 + \alpha_2 = \text{constant} = \alpha_1^{old} + \alpha_2^{old} = \gamma.$$

This equation demonstrates how α_1^{new} is computed from α_2^{new}. The objective function along the constraint becomes

$$\begin{aligned} W(\alpha_2) &= y_1\gamma - y_1\alpha_2 + y_2\alpha_2 - \varepsilon\left(|\alpha_1| + |\alpha_2|\right) - \frac{1}{2}K_{11}(\gamma - \alpha_2)^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ &\quad -K_{12}(\gamma - \alpha_2)\alpha_2 - (\gamma - \alpha_2)v_1 - \alpha_2 v_2 + \text{constant} \end{aligned}$$

and the stationary point satisfies

$$\begin{aligned} \frac{\partial W(\alpha_2)}{\partial \alpha_2} &= y_2 - y_1 - \varepsilon\left(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1)\right) + K_{11}(\gamma - \alpha_2) - K_{22}\alpha_2 \\ &\quad +K_{12}\alpha_2 - K_{12}(\gamma - \alpha_2) + v_1 - v_2 \\ &= 0. \end{aligned}$$

This yields

$$\begin{aligned} \alpha_2^{new,unc}\left(K_{11} + K_{22} - 2K_{12}\right) &= y_2 - y_1 - \varepsilon\left(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1)\right) \\ &\quad + \gamma\left(K_{11} - K_{12}\right) + v_1 - v_2 \\ &= y_2 - y_1 + \gamma\left(K_{11} - K_{12}\right) + v_1 - v_2. \end{aligned}$$

Hence,

$$\begin{aligned} \alpha_2^{new,unc}\kappa &= y_2 - y_1 + f(\mathbf{x}_1) - \sum_{j=1}^{2}\alpha_j K_{1j} + \gamma K_{11} \\ &\quad -f(\mathbf{x}_2) + \sum_{j=1}^{2}\alpha_j K_{2j} - \gamma K_{12} - \varepsilon\left(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1)\right) \\ &= y_2 - y_1 + f(\mathbf{x}_1) - f(\mathbf{x}_2) \\ &\quad +\alpha_2 K_{11} - \alpha_2 K_{12} + \alpha_2 K_{22} - \alpha_2 K_{12} - \varepsilon\left(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1)\right) \\ &= \alpha_2\kappa + \left(f(\mathbf{x}_1) - y_1\right) - \left(f(\mathbf{x}_2) - y_2\right) - \varepsilon\left(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1)\right), \end{aligned}$$

giving

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{E_1 - E_2 - \varepsilon\left(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1)\right)}{\kappa}.$$

Finally, we must clip $\alpha_2^{new,unc}$ if necessary to ensure it remains in the interval $[U, V]$. □

7.6 Techniques for Gaussian Processes

The solution of the Bayesian learning problem for the training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

using Gaussian processes was described in Subsection 6.2.3 where it was shown to be equivalent to ridge regression using the covariance matrix as a kernel. Solving this problem involves computing (see equation (6.10)) the parameter vector $\boldsymbol{\alpha}$ satisfying

$$f(\mathbf{x}) = \mathbf{y}'(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k} = \boldsymbol{\alpha}'\mathbf{k} = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}),$$

where

$$\begin{aligned} \boldsymbol{\alpha} &= (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \text{ or} \\ \mathbf{y} &= (\mathbf{K} + \sigma^2\mathbf{I})\boldsymbol{\alpha}, \end{aligned}$$

where $\mathbf{k}$ is a vector whose ith entry is $K(\mathbf{x}_i, \mathbf{x})$ and $\mathbf{K}$ the kernel matrix with entries $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. Hence, the solution of the problem is obtained by solving a system of linear equations of size $\ell \times \ell$, while the evaluation of the solution on a novel input requires the computation of an inner product involving the values of the kernel between the new point and each of the training examples. Note that in the case of a Gaussian process there is no reason to expect the parameter vector $\boldsymbol{\alpha}$ to be sparse.

Several methods exist for solving systems of linear equations, for example LU decomposition, Gauss Jordan elimination and in our case of a symmetric matrix Cholesky decomposition. Most off-the-shelf numerical packages will offer a choice of methods. Unfortunately the complexity of solving a system of linear equations scales with ℓ^3, making the approach impractical for very large datasets.

The problem can be tackled by a naive gradient descent method. Indeed Exercise 1 of Chapter 2 asked the reader to rewrite the Widrow–Hoff algorithm in dual form. If this is implemented without a bias and using the augmented kernel $\mathbf{K} + \sigma^2\mathbf{I}$, it will converge to the solution of the Gaussian process. A more sophisticated way of tackling the computational complexity problems is provided by conjugate gradient techniques. Table 7.3 shows the sequence of calculations that perform the conjugate gradient calculation.

The process is guaranteed to converge to the solution in $n = \ell$ iterations, but an early stopping will deliver an approximation of $\boldsymbol{\alpha}_{\max}$ with a complexity that scales only as $n \times \ell^2$. The quality of the approximation can be estimated using so-called Skilling methods, which hence give a good stopping criterion for the iterative procedure (for more details, follow the links in Section 7.8).

7.7 Exercises

1. Implement the gradient descent algorithm for the maximum margin optimisation problem. Test it on an artificially created dataset. Print out the Karush–Kuhn–Tucker conditions. Plot the margin as a function of the number of iterations. Introduce some sample selection heuristics and redo the plot.

https://doi.org/10.1017/CBO9780511801389.009 Published online by Cambridge University Press

$\boldsymbol{\alpha}^1 \leftarrow \mathbf{0}$, $\mathbf{h}^1 \leftarrow \mathbf{g}^1 \leftarrow \mathbf{y}$
for $k = 1, \ldots, n$
$\lambda \leftarrow \frac{(\mathbf{g}^k)'\mathbf{g}^k}{(\mathbf{g}^k)'\mathbf{C}\mathbf{h}^k}$
$\boldsymbol{\alpha}^{k+1} \leftarrow \boldsymbol{\alpha}^k + \lambda \mathbf{h}^k$
$\mathbf{g}^{k+1} \leftarrow \mathbf{g}^k - \lambda \mathbf{C}\mathbf{h}^k$
$\gamma \leftarrow \frac{(\mathbf{g}^{k+1})'\mathbf{g}^{k+1}}{(\mathbf{g}^k)'\mathbf{g}^k}$
$\mathbf{h}^{k+1} \leftarrow \mathbf{g}^{k+1} + \gamma \mathbf{h}^k$
end for
return $\boldsymbol{\alpha}$

Table 7.3: **Pseudocode for the conjugate gradient method**

2. Try the same algorithm on non-separable data. Add the soft margin features to the algorithm. Experiment with the tunable parameters. Run the above algorithms on a small real-world dataset, taken for example from the UCI repository [96].

3. Implement SMO for classification and run it on a range of datasets.

7.8 Further Reading and Advanced Topics

The problem of convex optimisation has been extensively studied since the 1950s, and a number of techniques exist to solve it. This chapter did not attempt to be exhaustive, but simply to provide the reader with some techniques that are both easy to implement and exemplify some standard approaches. Discussions of optimisation algorithms can be found in the books [41], [80], [11], and [86].

For the particular case of Support Vector Machines, see the excellent surveys by Smola and Schölkopf [145] and Burges [23]. Implementation problems and techniques are discussed by Kauffman [70], Joachims [68], Platt [114][112], Osuna and Girosi [111], and Keerthy et al. [73][74].

The gradient ascent techniques were introduced into the SVM literature by papers on the kernel-Adatron procedure, [44][24]; similar ideas have been used independently by Haussler and Jaakkola [65]. They are also related to SMO with fixed b [112]; and to Hildreth's QP method [62]. Mangasarian and his co-workers recently introduced algorithms that can deal with massive datasets ([90], [91], [20]). Albeit with different motivations, the algorithm SOR (Successive Over Relaxation) of Mangasarian and Musicant [89] is equivalent to the stochastic gradient ascent algorithm described in Section 7.2, combined with sample selection heuristics motivated by Platt's approach [112]. Mangasarian and Musicant [89] also give the proof of linear convergence of the algorithm using the link with SOR.

In Remark 7.4, we discussed the case when it is possible to use fixed bias; note that also Jaakkola and Haussler [65] use this assumption; and Mangasarian and

Musicant [89] discuss the cases in which such an assumption is not too restrictive. Similar approaches have also been used in [170] and [45].

An on-line theory of generalisation for gradient descent algorithms has been developed by Littlestone, Warmuth and others [75]. A discussion of square loss algorithms can be found in [75], while the theory of hinge loss is developed in [48]. This theory provides tight bounds on the maximum number of mistakes an on-line algorithm can make in the worst case, and requires surprisingly few assumptions. The bounds can also be translated for the case where the data have been generated by a distribution. One of its best-known consequences is the motivation of multiplicative updating algorithms, and the characterisation of the cases in which they are expected to perform better than standard gradient descent. A multiplicative algorithm for Support Vector Machines has been studied by Cristianini et al. [29].

The elegant SMO algorithm was devised by Platt [112], and applied to text categorisation problems. Pseudocode (kindly supplied by John Platt) for SMO is available in Appendix A of this book. An extension of SMO, differing in the way it calculates the bias, has been proposed in [74] and shown to be faster. Alex Smola has generalised SMO for the case of regression [148], [145], and the code can be found on the GMD website [53].

Keerthy et al. [73] proposed a very elegant algorithm for SVMs that does not maximise the margin by minimising the norm of the weight vector. Rather, they note that the distance vector between the nearest points of the convex hulls of the positive and negative data uniquely determines the maximal margin hyperplane. This more geometrical view of the problem is discussed for example in [14] and in [129]. Exercise 2 of Chapter 5 asks the reader to devise an optimisation problem for this criterion, hence motivating this alternative solution strategy for the maximum margin problem. Based on the same approach, Kowalczyk [76] has proved a convergence rate for a new iterative algorithm that can be applied to the hard and soft margin problems, as well as extensive experimental comparisons with other iterative algorithms Guyon and Stork [56] give another variant of an iterative algorithm for soft margin optimisation.

Chunking techniques in Support Vector Machines were already used by Vapnik and Chervonenkis, and were improved, generalised and discussed in a number of papers, among others Osuna and Girosi [111], [110], [109], Joachims [68], Platt [112], Smola and Schölkopf [145], and Kauffman [70]. The work of Osuna and Girosi inspired the subsequent work on data selection, which ultimately led to systems like SMO.

Techniques exist for tuning the parameters automatically. For example [31] adjusts the kernel parameters while the ν-SVMs allow the user to set an upper bound on the number of support vectors for classification [135], and regression [130]. The stopping criterion based on the feasibility gap is discussed in [148], who proposes a criterion similar to that of equation (7.3). Other criteria are discussed in [73] and [74].

The implementation of Gaussian processes is discussed in [50]. An experimental study of Widrow–Hoff with kernels can be found in [46].

The book [100] discusses general techniques for convex optimisation and gives

pointers to commercial optimisation packages. Early implementations of SVMs have been based on optimisation packages such as MINOS [101], LOQO [156], MATLAB optimisation package [92], and others mentioned above. Chapter 1 of the book [149] contains a useful survey of different implementations.

Packages specifically for SVM implementation are available on-line. The package jointly prepared by the groups at Royal Holloway, ATT and GMD FIRST as available at the Royal Holloway, University of London, website [126]. The package SVM^{light} of Joachims [68] is also available on the web via the website [30]. The package prepared by Alex Smola for SVM classification is available at the GMD-FIRST website [53]. Web links to this and other software are available via the website [30].

The original reference for the conjugate gradient algorithm is [61], while the discussion of Skilling methods for estimating the quality of the approximation obtained is given in [49]. Software is also available on-line for Gaussian processes: Radford Neal's code [103]; Gibbs and MacKay [49].

These references are also given on the website **www.support-vector.net**, which will be kept up to date with new work, pointers to software and papers that are available on-line.