Chapter 2

Support Vector Machine Training Methods

An overview of current methods of Support Vector Machine training is given in this chapter. The method of decomposing the training problem into subproblems is discussed in detail, and includes conditions for optimality of the training problem, methods for selecting good subproblems, and different optimisations to the decomposition algorithm itself. The chapter concludes with a complete Support Vector Machine training algorithm.

2.1 Introduction to Support Vector Machine training methods

Training a Support Vector Machine (SVM) involves solving a linearly constrained quadratic optimisation problem. The SVM fits a decision function to a labelled set of l training patterns, which correspond to the total of l free parameters in the optimisation problem. The training data set consists of a total of l N-dimensional patterns $\mathbf{x}_i$ and their respective class labels y_i. The quadratic programming (QP) problem, from chapter one, is to find

$$\begin{aligned} \max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) &= \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ \text{subject to} \quad \boldsymbol{\alpha}^T\mathbf{y} &= 0 \\ \boldsymbol{\alpha} &\geq \mathbf{0} \\ C\mathbf{1} - \boldsymbol{\alpha} &\geq \mathbf{0} \end{aligned} \tag{2.1}$$

In the QP problem, the objective function – the function to be maximised – depends on the α_i quadratically, while the parameters α_i only appear linearly in the constraints. Q is

an l by l matrix that depends on both a kernel function of the training inputs, and their respective labels: $(Q)_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$.

The QP problem is equivalent to finding the maximum of a bowl-shaped objective function. The search for the maximum occurs in l dimensions, and is constrained to lie inside a hypercube and on a hyperplane. Due to the definition of the kernel function, the matrix Q always gives a convex QP problem. The convexity of the optimisation problem implies that every local maximum is also a global maximum [20]. A global maximum means that there is no other point inside the feasible region at which the objective function takes a higher value. When Q is positive definite, the objective function will be bowl-shaped; when Q is positive semi-definite, the objective function will have flat-bottomed troughs. The objective function will never be saddle-shaped. Thus there exists a unique maximum or a connected set of maximums. Certain optimality conditions – the Karush-Kuhn-Tucker (KKT) conditions [20] – give conditions determining whether the constrained maximum has been found.

The SVM QP problem is simple and well understood; yet solving the QP problem for real-world cases can prove to be very difficult. Analytic solutions are possible when the number of training patterns is very small, or when the data is separable and it is known beforehand which vectors will be support vectors. In most real-world cases, numeric solutions are called for. Small problems can be solved with general-purpose optimisation packages that solve linearly constrained convex QPs. Larger problems, however, bring about difficulties in both the size and density of Q.

The matrix Q has a dimension equal to the number of training examples. A training set of 60,000 vectors gives rise to a matrix Q with 3.6 billion elements, which does not fit into the memory of a standard computer. For large learning tasks, off-the-shelf optimisation packages and techniques for general quadratic programming quickly become intractable in their memory and time requirements.

In general $(Q)_{ij}$ is nonzero, which makes Q completely dense. Most mathematical approaches either assume that Q is sparse (i.e. most $(Q)_{ij}$ are zero), or are only suitable for small problems.

Since standard QP techniques cannot easily be used to train SVMs with several thousands of examples, a number of other approaches have been invented. These algorithms allow for fast convergence and small memory requirements, even on large problems.

2.1.1 Chunking

The chunking algorithm is based on the fact that the non-support vectors play no role in the SVM decision boundary. If they are removed from the training set of examples, the SVM solution will be exactly the same.

Chunking was first suggested by V. Vapnik in [57]. The large QP problem is broken

down into a number of smaller problems:

A QP routine is used to optimise the Lagrangian on an arbitrary subset of data. After this optimisation, the set of nonzero α_i (the current support vectors) are retained, and all other data points ($\alpha_i = 0$) are discarded. At every subsequent step, chunking solves the QP problem that consists of all nonzero α_i, plus some of the α_i that violates the KKT conditions. These are in general the worst M violations, for some value of M. After optimising the subproblem, data points with $\alpha_i = 0$ are again discarded. This procedure is iterated until the KKT conditions are met, and the margin is maximised. Solving each subproblem still requires a numeric quadratic optimiser.

The size of the subproblem varies, but tends to grow with time. At the last step, chunking has identified and optimised all the nonzero α_i, which correspond to the set of all the support vectors. Thus the overall QP problem is solved.

Although this technique of reducing the Q matrix's dimension from the number of training examples to approximately the number of support vectors makes it suitable to large problems, a limitation still exists. The number of support vectors may exceed the maximal number of parameters α_i that the quadratic optimiser can handle, and even the reduced matrix may not fit into memory.

2.1.2 Decomposition

Decomposition methods solve a sequence of smaller QP problems, and are similar in spirit to chunking. The difference from chunking is in the size of the subproblems: the size remains fixed.

Decomposition methods were introduced in 1997 by E. Osuna *et al.* [41]. The large QP problem is broken down into a series of smaller subproblems, and a numeric QP optimiser solves each of these problems. It was suggested that one vector be added and one removed from the subproblem at each iteration, and that the size of the subproblems should be kept fixed. The motivation behind this method is based on the observation that as long as at least one α_i violating the KKT conditions is added to the previous subproblem, each step reduces the objective function and maintains all of the constraints. In this fashion the sequence of QP subproblems will asymptotically converge. For faster practical convergence, researchers use different unpublished heuristics to add and delete multiple examples.

While the strategy used in chunking takes advantage of the fact that the expected number of support vectors is small (< 3000), decomposition allows for training arbitrarily large data sets.

Another decomposition method was introduced by T. Joachims in [25]. Joachim's method is based on the gradient of the objective function. The idea is to pick α_i for the QP subproblem such that the α_i form the steepest possible direction of ascent on the objective

function, where the number of nonzero elements in the direction is equal to the size of the QP subproblem. As in Osuna's method, the size of the subproblem remains fixed.

2.1.3 Sequential Minimal Optimisation

The most extreme case of decomposition is Sequential Minimal Optimisation (SMO) – where the smallest possible optimisation problem is solved at each step [42]. Due to the fact that the α_i must obey the linear equality constraint, the smallest set of α_i that can be optimised at each step is two. At every step, SMO chooses two α_i to jointly optimise, finds the optimal values for these α_i, and updates the SVM to reflect these changes.

SMO avoids numerical QP optimisation and large matrix storage entirely: if the two chosen α_i are optimised and the rest of the parameters α_i kept fixed, it derives an analytic solution which is executed in a few numerical operations. The method therefore consists of a heuristic step for finding the best pair of parameters to optimise, and the use of an analytic expression to ensure the objective function increases monotonically. Because the smallest possible subproblem is optimised at each iteration of the algorithm, SMO solves more subproblems than other methods of decomposition. Optimising each subproblem, however, is so fast that the overall QP problem can be solved quickly. Due to the decomposition of the QP problem and its speed, SMO is probably the method of choice for training SVMs [11].

In this chapter a decomposition algorithm based on the ideas of T. Joachims [25] is discussed. Joachims' method is presented in Section 2.3.2. This algorithm makes no assumption on the expected number of support vectors, and allows training arbitrary large data sets. In constructing the algorithm, conditions for optimality, decomposition and optimality conditions on the working set are discussed. Finally, a complete training algorithm is presented.

2.2 Conditions for optimality

In this section, conditions for optimality of a solution $\boldsymbol{\alpha}$ to problem (2.1) are introduced. Since Q is a positive semi-definite matrix (the kernel function used is positive definite), and the constraints are linear, the Karush-Kuhn-Tucker (KKT) conditions [20] are necessary and sufficient for optimality.

The KKT multipliers are introduced by letting μ be the associated multiplier of $\boldsymbol{\alpha}^T\mathbf{y} = 0$, $\boldsymbol{\pi}^T = (\pi_1, \ldots, \pi_l)$ be the associated multiplier of $-\boldsymbol{\alpha} \leq \mathbf{0}$, and $\boldsymbol{\upsilon}^T = (\upsilon_1, \cdots, \upsilon_l)$ be the associated multiplier of $\boldsymbol{\alpha} - C\mathbf{1} \leq \mathbf{0}$. The following KKT conditions must then hold for optimality:

$$\nabla W(\boldsymbol{\alpha}) - \nabla \boldsymbol{\upsilon}^T(\boldsymbol{\alpha} - C\mathbf{1}) - \nabla \boldsymbol{\pi}^T(-\boldsymbol{\alpha}) - \nabla \mu(\boldsymbol{\alpha}^T\mathbf{y}) \quad = \quad \mathbf{0}$$

$$\Rightarrow \nabla W(\boldsymbol{\alpha}) - \upsilon + \pi - \mu \mathbf{y} = \mathbf{0} \tag{2.2}$$

$$\upsilon^T(\boldsymbol{\alpha} - C\mathbf{1}) = 0 \tag{2.3}$$

$$\pi^T \boldsymbol{\alpha} = 0 \tag{2.4}$$

$$\upsilon \geq \mathbf{0} \tag{2.5}$$

$$\pi \geq \mathbf{0} \tag{2.6}$$

The Lagrange multipliers α_i can have three possible values: The value of α_i can be at zero, at the upper bound C, or somewhere in the interval $(0, C)$. By defining the classifier function

$$f^\star(\mathbf{x}) = \sum_{i=1}^{l} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b \tag{2.7}$$

similar to (1.27), each of these cases are now considered and expanded separately.

Case 1: $0 < \alpha_i < C$

Consider a single value of α_i, i.e. the Lagrange multiplier associated with some input vector i. Then, from equation (2.2),

$$1 - (Q\boldsymbol{\alpha})_i - \upsilon_i + \pi_i - \mu y_i = 0$$

Since this case examines α_i from the interval $(0, C)$, the term $(\boldsymbol{\alpha} - C\mathbf{1})_i$ from (2.3) must be non-zero and negative. For equations (2.3) and (2.5) to hold, υ_i must be equal to zero.

By using a similar argument, conditions (2.4) and (2.6) imply that π_i can only be zero. This gives

$$1 - (Q\boldsymbol{\alpha})_i - \mu y_i = 0 \tag{2.8}$$

Because the equation

$$y_i f^\star(\mathbf{x}_i) = y_i \Big(\sum_{j=1}^{l} y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + b \Big) = 1 \tag{2.9}$$

holds when $0 < \alpha_i < C$, and given that

$$\begin{aligned} (Q\boldsymbol{\alpha})_i &= \sum_{j=1}^{l} y_i y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \\ &= y_i \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \\ &= y_i \big(f^\star(\mathbf{x}_i) - b \big) \end{aligned}$$

equation (2.8) can be rewritten, and simpifies as

$$\begin{aligned} 1 - (Q\boldsymbol{\alpha})_i - \mu y_i &= 1 - y_i\big(f^\star(\mathbf{x}_i) - b\big) - \mu y_i \\ &= 1 - 1 + y_i b - \mu y_i = 0 \end{aligned}$$

From this the value of b is equal to the KKT multiplier μ, i.e.

$$\mu = b \tag{2.10}$$

Case 2: $\alpha_i = C$

As in the previous case, consider equation (2.2) for a single Lagrange multiplier α_i at the upper bound C:

$$1 - (Q\boldsymbol{\alpha})_i - \upsilon_i + \pi_i - \mu y_i = 0$$

Because $\alpha_i = C$, conditions (2.4) and (2.6) imply that π_i must be equal to zero. Then,

$$1 - (Q\boldsymbol{\alpha})_i - \upsilon_i - \mu y_i = 0 \tag{2.11}$$

Equation (2.5) specifies that $\upsilon_i \geq 0$, and thus

$$\begin{aligned} 1 - (Q\boldsymbol{\alpha})_i - \mu y_i &\geq 0 \\ 1 - y_i\big(f^\star(\mathbf{x}_i) - b\big) - b y_i = 1 - y_i f^\star(\mathbf{x}_i) &\geq 0 \end{aligned}$$

Thus for a value of $\alpha_i = C$ to meet the KKT conditions, it must be true that

$$y_i f^\star(\mathbf{x}_i) \leq 1 \tag{2.12}$$

Case 3: $\alpha_i = 0$

In the case of $\alpha_i = 0$, equation (2.2) becomes

$$1 - (Q\boldsymbol{\alpha})_i - \upsilon_i + \pi_i - \mu y_i = 0$$

Conditions (2.3) and (2.5), with $\alpha_i = 0$, imply that $\upsilon_i = 0$. Therefore,

$$1 - (Q\boldsymbol{\alpha})_i + \pi - \mu y_i = 0 \tag{2.13}$$

Using similar reasoning as the above case of $\alpha_i = C$, it can be shown that a value of $\alpha_i = 0$ meets the KKT conditions if

$$y_i f^\star(\mathbf{x}_i) \geq 1 \tag{2.14}$$

Concluding on the KKT conditions

From the three cases presented above, a solution $\boldsymbol{\alpha}$ of problem (2.1) is an optimal solution if the following relations hold for each α_i:

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i f^\star(\mathbf{x}_i) \geq 1 \\ 0 < \alpha_i < C &\Rightarrow y_i f^\star(\mathbf{x}_i) = 1 \\ \alpha_i = C &\Rightarrow y_i f^\star(\mathbf{x}_i) \leq 1 \end{aligned} \tag{2.15}$$

If, for some given stage in the process of training a SVM, all Lagrange multipliers meet the KKT conditions, an optimal solution to (2.1) is found and SVM training can stop.

Computing the value of the threshold b

A value for the threshold b is needed for (2.7), and can be computed for each of the support vectors. From (2.9),

$$b_i = y_i - \sum_{j=1}^{l} y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.16}$$

The average of these values is taken as the value for b.

2.3 A decomposition method

Decomposition methods break the large QP problem down to a series of smaller subproblems, and these subproblems are optimised to improve the objective function.

In the process of decomposition, a subset of variables is chosen for optimisation. The original set of Lagrange multiplier variables is divided into two sets, called B and N. Set B is called the "working set," and is created by picking q sub-optimal variables from all l α_i. The working set of variables is optimised while keeping the remaining variables (set N) constant. After subset B is optimised, it is "put back" into the original set and a new working set is selected for optimisation.

Since it is known when a solution $\boldsymbol{\alpha}$ is an optimal solution (the solution satisfies all KKT conditions), the problem can be decomposed and optimised until these conditions are met with an adequate tolerance. The general decomposition algorithm is summarized as follows:

Algorithm 2.1 - General decomposition algorithm

1. While the optimality conditions (2.15) are violated
   (a) Select q variables for the working set B. The remaining $l - q$ variables are fixed at their current values.

(b) Decompose the problem and solve the quadratic program subproblem, i.e. optimise $W(\boldsymbol{\alpha})$ on B.

2. Terminate and return $\boldsymbol{\alpha}$.

Concerns of the above algorithm are the creation of KKT criteria for knowing when the working set B is optimised, and methods of picking the optimal working set.

Firstly, however, it is necessary to rewrite equation (2.1) as a function that is only dependent on the working set. Let $\boldsymbol{\alpha}$ be split into two sets $\boldsymbol{\alpha}_B$ and $\boldsymbol{\alpha}_N$. If $\boldsymbol{\alpha}$, $\mathbf{y}$ and Q are appropriately rearranged, one has

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} \mathbf{y}_B \\ \mathbf{y}_N \end{bmatrix}, \qquad Q = \begin{bmatrix} Q_{BB} & Q_{BN} \\ Q_{NB} & Q_{NN} \end{bmatrix}$$

Since only $\boldsymbol{\alpha}_B$ is being optimised for the subproblem, W is rewritten from equation (2.1) in terms of $\boldsymbol{\alpha}_B$ to give

$$\begin{aligned} W(\boldsymbol{\alpha}_B) = \left(\boldsymbol{\alpha}_B^T\mathbf{1} + \boldsymbol{\alpha}_N^T\mathbf{1}\right) - \frac{1}{2}\Big(&\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B + \boldsymbol{\alpha}_B^T Q_{BN}\boldsymbol{\alpha}_N \\ &+\boldsymbol{\alpha}_N^T Q_{NB}\boldsymbol{\alpha}_B + \boldsymbol{\alpha}_N^T Q_{NN}\boldsymbol{\alpha}_N\Big) \end{aligned} \tag{2.17}$$

If terms that do not contain $\boldsymbol{\alpha}_B$ are dropped, the optimisation problem remains essentially the same. Also, since Q is a symmetric matrix, with $Q_{BN} = Q_{NB}^T$, the problem reduces to finding

$$\begin{aligned} \max_{\boldsymbol{\alpha}_B} \; & W(\boldsymbol{\alpha}_B) = \boldsymbol{\alpha}_B^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B - \boldsymbol{\alpha}_B^T Q_{BN}\boldsymbol{\alpha}_N \\ \text{subject to} \quad & \boldsymbol{\alpha}_B^T\mathbf{y}_B + \boldsymbol{\alpha}_N^T\mathbf{y}_N = 0 \\ & \boldsymbol{\alpha}_B \geq \mathbf{0} \\ & C\mathbf{1} - \boldsymbol{\alpha}_B \geq \mathbf{0} \end{aligned} \tag{2.18}$$

With $|B| \ll |N|$, the term $\boldsymbol{\alpha}_B^T Q_{BN}\boldsymbol{\alpha}_N$ consumes the majority of computing time when determining $W(\boldsymbol{\alpha}_B)$. As a performance optimisation, define a vector $\mathbf{q}_{BN} = Q_{BN}\boldsymbol{\alpha}_N$ in the following way:

$$(q_{BN})_i = y_i \sum_{j \in N} \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.19}$$

The vector $\mathbf{q}_{BN}$ is computed once at the start of every subset optimisation. The complexity of the optimisation problem then becomes proportional to the size of the working set, independent of l. Given that l can be very large and that $q = |B|$ will be relatively small, it is a vast improvement. The optimisation problem becomes equivalent to finding

$$\max_{\boldsymbol{\alpha}_B} W(\boldsymbol{\alpha}_B) = \boldsymbol{\alpha}_B^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B - \boldsymbol{\alpha}_B^T\mathbf{q}_{BN}$$

$$\begin{aligned} \text{subject to} \quad \boldsymbol{\alpha}_B^T \mathbf{y}_B + \boldsymbol{\alpha}_N^T \mathbf{y}_N &= 0 \\ \boldsymbol{\alpha}_B &\geq \mathbf{0} \\ C\mathbf{1} - \boldsymbol{\alpha}_B &\geq \mathbf{0} \end{aligned} \tag{2.20}$$

2.3.1 Optimality of the working set

The optimisation problem in (2.20) has one particularly useful property: one can computationally determine if a solution is an optimal solution. This gives a stopping criterion for optimising the working set B.

The decomposed problem (2.20) consists of a convex objective function (since matrix Q_{BB} is positive semi-define), and linear constraints. The KKT conditions are thus necessary and sufficient for optimality.

The KKT conditions must hold for each element in $\boldsymbol{\alpha}_B$, and by again considering the possible values of $(\alpha_B)_i$, as in Section 2.2, the conditions are:

$$\begin{aligned} (\alpha_B)_i = 0 &\Rightarrow (Q_{BB}\boldsymbol{\alpha}_B)_i + (q_{BN})_i + \mu(y_B)_i \geq 1 \\ 0 < (\alpha_B)_i < C &\Rightarrow (Q_{BB}\boldsymbol{\alpha}_B)_i + (q_{BN})_i + \mu(y_B)_i = 1 \\ (\alpha_B)_i = C &\Rightarrow (Q_{BB}\boldsymbol{\alpha}_B)_i + (q_{BN})_i + \mu(y_B)_i \leq 1 \end{aligned} \tag{2.21}$$

When the Lagrange multiplier α_i lies between zero and C, the value of μ can be computed with

$$\mu = (y_B)_i\big(1 - (Q_{BB}\boldsymbol{\alpha}_B)_i - (q_{BN})_i\big)$$

The value of μ, as it appears in the above KKT conditions (2.21), can be taken as the average of μ computed for each i where $0 < (\alpha_B)_i < C$.

Apart from the optimality conditions described here, a method for selecting good or optimal working sets – a decomposition algorithm – is needed. Such a method will choose the working set B, while the KKT conditions presented here determines the termination criteria on optimising B.

2.3.2 Selecting the working set

One of the most important issues in a decomposition algorithm is the selection of the working set. The working set selected plays a major role in the speed of the SVM training algorithm. Selecting working sets at random causes the training algorithm (Algorithm 2.1) to converge very slowly, while continually selecting optimal variables causes the training algorithm to cycle. A method for selecting approximately optimal working sets is presented below.

The decomposition method presented in this section is due to [25, 39]. It works on the classical method of feasible directions, proposed in the optimisation theory by [63]. If Ω is

a feasible region of a general constrained problem, then a vector $\mathbf{d}$ is a feasible direction at the point $\boldsymbol{\alpha}$ in Ω, if there exists a $\tilde{\lambda}$ such that $\boldsymbol{\alpha} + \lambda\mathbf{d}$ lies in Ω for all $0 \leq \lambda \leq \tilde{\lambda}$.

The main idea of the method of feasible directions is to start with an initial feasible solution, and to find the optimal solution by making steps along feasible directions. At each iteration of a feasible directions algorithm, the optimal feasible direction (the direction giving the largest rate of increase of the objective function) is found. The algorithm then aims to maximise the objective function along this direction, by making a line search to determine a step length along the feasible direction. The solution is moved by "stepping" along the feasible direction to the better solution found. The algorithm terminates when no feasible directions can be found which improve the objective function.

The optimal feasible direction of a general constrained optimisation problem of the form

$$\text{Maximise } f(\boldsymbol{\alpha}) \qquad \text{subject to } A\boldsymbol{\alpha} \leq \mathbf{b}$$

is found by solving the direction finding linear program

$$\text{Maximise } \nabla f^T\mathbf{d} \qquad \text{subject to } A\mathbf{d} \leq \mathbf{0},\ \ ||\mathbf{d}||_2 \leq 1$$

SVM training solves a constrained quadratic optimisation problem, therefore the method of feasible directions is directly applicable to training a SVM. Finding the optimal feasible direction when solving the SVM problem (2.1) can be stated as

$$\begin{aligned} \text{Maximise} \quad & \nabla W(\boldsymbol{\alpha})^T\mathbf{d} \\ \text{subject to} \quad & \mathbf{y}^T\mathbf{d} = 0 \\ & d_i \geq 0 \quad \text{if } \alpha_i = 0 \\ & d_i \leq 0 \quad \text{if } \alpha_i = C \\ & ||\mathbf{d}||_2 \leq 1 \end{aligned} \tag{2.22}$$

Optimisation problem (2.22) is a full-scale linear program of dimension l, which is computationally expensive to solve at every iteration of the decomposition method of SVM training. An approximate solution to this problem, which can be obtained in linear time, was proposed by T. Joachims [25].

A requirement is added to (2.22), specifying that only q components of $\mathbf{d}$ be non-zero. The variables corresponding to these q non-zero components are included in the working set. Since this only gives an approximation to (2.22), $\mathbf{d}$ is only used to identify B, and not as a search direction. Instead of doing a line search on $\mathbf{d}$, the optimum solution is found in the *entire* subspace spanned by the non-zero components of $\mathbf{d}$.

By specifying that only q components of $\mathbf{d}$ be non-zero, the problem becomes intractable. This problem of intractability is overcome by letting d_i be equal to either -1, 0 or $+1$,

$\mathbf{g}^T$ =	-1	-2	+3	+4	-4	+5	+1	0	-2	+5
$\mathbf{y}^T$ =	+1	+1	-1	-1	+1	-1	+1	+1	-1	+1
$\mathbf{d}^T$ =	0	0	0	+1	-1	+1	0	0	0	+1
$y_i d_i$ =	0	0	0	-1	-1	-1	0	0	0	+1

(a) Selecting the four largest values of $|g_i|$, and setting each corresponding d_i to the sign of g_i, maximises $\mathbf{g}^T\mathbf{d}$, but the equality constraint $\mathbf{y}^T\mathbf{d} = 0$ is not met.

$\mathbf{g}^T$ =	-1	-2	+3	+4	-4	+5	+1	0	-2	+5
$\mathbf{y}^T$ =	+1	+1	-1	-1	+1	-1	+1	+1	-1	+1
$y_i g_i$ =	-1	-2	-3	-4	-4	-5	+1	0	+2	+5
$\mathbf{d}^T$ =	0	0	0	+1	0	+1	0	0	-1	+1
$y_i d_i$ =	0	0	0	-1	0	-1	0	0	+1	+1

(b) Selecting the two smallest and largest $y_i g_i$, and respectively letting d_i be of opposite and similar sign to y_i, $\mathbf{g}^T\mathbf{d}$ is maximised such that the equality constraint $\mathbf{y}^T\mathbf{d} = 0$ is also met.

FIGURE 2.1: Selecting a working set of size four.

such that the Lagrange multipliers α_i corresponding to $d_i = \pm 1$ are included in B. An approximation of (2.22) is thus found by

$$\begin{aligned} \text{Maximise} \quad & \nabla W(\boldsymbol{\alpha})^T \mathbf{d} \\ \text{subject to} \quad & \mathbf{y}^T \mathbf{d} = 0 \\ & d_i \geq 0 \qquad \text{if } \alpha_i = 0 \\ & d_i \leq 0 \qquad \text{if } \alpha_i = C \\ & d_i \in \{-1, 0, 1\} \\ & |\{d_i : d_i \neq 0\}| = q \end{aligned} \tag{2.23}$$

From this approximation the question arises: how is the direction $\mathbf{d}$ determined? Firstly, assume that the constraints $\mathbf{y}^T\mathbf{d} = 0$, $d_i \geq 0$ if $\alpha_i = 0$, and $d_i \leq 0$ if $\alpha_i = C$, are all absent. Also, to simplify the notation used, let the shorthand $\mathbf{g} = \nabla W(\boldsymbol{\alpha})$ denote the directional derivative of W. With the equality and inequality constraints absent, the maximum of the objective function is achieved by selecting q points with the highest values of $|g_i|$. Then d_i will take the value of $\text{sign}(g_i)$.

As an example, consider Figure 2.1(a), with q equal to four. The four largest values of $|g_i|$ are chosen ($|g_4| = 4$, $|g_5| = 4$, $|g_6| = 5$ and $|g_{10}| = 5$), and each corresponding d_i is set to the sign of g_i. In this way $\mathbf{g}^T\mathbf{d}$ is maximised.

The first remark that can be made about the example in Figure 2.1(a), is that the equality constraint $\mathbf{y}^T\mathbf{d} = 0$ is being violated. For $\mathbf{y}^T\mathbf{d}$ to be equal to zero, the number of elements with sign matches between d_i and y_i must be equal to the number of elements with sign mismatches between d_i and y_i. This means that if a working set of size q is selected,

with q being even, each number must be equal to $\frac{q}{2}$. The working set can thus be selected by making two passes over the data. A "forward pass" will select $\frac{q}{2}$ sign mismatches, while a "backward pass" will select $\frac{q}{2}$ sign matches. To implement selection of the working set, let γ_k denote the largest contribution to the objective function $\mathbf{g}^T\mathbf{d}$ by some point k, subject to the equality constraint $\mathbf{y}^T\mathbf{d} = 0$. The two passes over the data, each selecting $\frac{q}{2}$ variables, are expanded in the following way:

"Forward pass"

The forward pass attempts to select $\frac{q}{2}$ variables such that $y_k d_k$ is negative. This implies that the signs of y_k and d_k must be different in maximising $\mathbf{g}^T\mathbf{d}$. To maximise $\mathbf{g}^T\mathbf{d}$, the minimum g_i is chosen when d_i is negative, while the maximum g_i is selected when d_i is positive, i.e.

$$\begin{aligned} y_k = 1 &\Rightarrow d_k = -1 \Rightarrow \gamma_k = \min_{i:y_i=1}(g_i) &&\Rightarrow \gamma_k = \min_{i:y_i=1}(y_i g_i) \\ y_k = -1 &\Rightarrow d_k = 1 &&\Rightarrow \gamma_k = \max_{i:y_i=-1}(g_i) \Rightarrow \gamma_k = \min_{i:y_i=-1}(y_i g_i) \end{aligned}$$

If the subscripts are combined, the largest contribution to the objective function (with y_k and d_k having different signs), subject to the equality constraint, is

$$\gamma_k = \min_i(y_i g_i) \tag{2.24}$$

"Backward pass"

The backward pass over the data selects a total of $\frac{q}{2}$ variables, such that $y_k d_k$ is positive. Thus the signs of y_k and d_k must be the same in maximising $\mathbf{g}^T\mathbf{d}$, i.e.

$$\begin{aligned} y_k = 1 &\Rightarrow d_k = 1 &&\Rightarrow \gamma_k = \max_{i:y_i=1}(g_i) &&\Rightarrow \gamma_k = \max_{i:y_i=1}(y_i g_i) \\ y_k = -1 &\Rightarrow d_k = -1 &&\Rightarrow \gamma_k = \min_{i:y_i=-1}(g_i) &&\Rightarrow \gamma_k = \max_{i:y_i=-1}(y_i g_i) \end{aligned}$$

If the subscripts are combined, the largest contribution to the objective function (with y_k and d_k having the same signs), subject to the equality constraint, is

$$\gamma_k = \max_i(y_i g_i) \tag{2.25}$$

The working set is thus selected based on the equations (2.24, 2.25) defined above. The example of Figure 2.1(a) selected an optimal but useless working set, since it does not include the equality constraint.

In Figure 2.1(b) the two smallest and largest $y_i g_i$ ($y_4 g_4 = -4$, $y_6 g_6 = -5$, $y_9 g_9 = +2$ and $y_{10} g_{10} = +5$) are selected, such that the example correctly meets the equality constraint $\mathbf{y}^T\mathbf{d} = 0$.

It is clear that the quantity $y_i g_i$ gives an indication of an element's contribution to the objective function subject to the equality constraint. This quantity is used to select the working set, by *sorting* the data elements according to $y_i g_i$ and selecting the top and bottom $\frac{q}{2}$.

Accounting for the inequality constraints in (2.23) then becomes a trivial task – when selecting the top and bottom Lagrange multiplier variables α_i from the sorted list, a variable is skipped if the inequality constraints are violated. Thus variables are skipped if $d_i = -y_i$ (or in the case of the backward pass, if $d_i = y_i$) violates $d_i \geq 0$ if $\alpha_i = 0$, and $d_i \leq 0$ if $\alpha_i = C$. Consider the forward pass: if $d_i = -y_i$, then variables should be chosen when $-y_i \geq 0$ if $\alpha_i = 0$, and $-y_i \leq 0$ if $\alpha_i = C$. These conditions hold when $y_i = -1$ and $\alpha_i = 0$, or when $y_i = 1$ and $\alpha_i = C$. A similar argument on the backward pass states that variables should be chosen when $y_i = 1$ and $\alpha_i = 0$, or when $y_i = -1$ and $\alpha_i = C$.

The decomposition algorithm, which selects variables with a forward and backward pass over the data, is implemented below:

Algorithm 2.2 - Decomposition algorithm

1. Let L be a list of all Lagrange multipliers.
2. While the optimality conditions (2.15) are violated
   (a) sort L by $y_i g_i$ in increasing order
   (b) select $\frac{q}{2}$ samples from the front of L such that
      - $0 < \alpha_i < C$ or
      - $(y_i = -1 \text{ and } \alpha_i = 0)$ or $(y_i = 1 \text{ and } \alpha_i = C)$
   (c) select $\frac{q}{2}$ samples from the back of L such that
      - $0 < \alpha_i < C$ or
      - $(y_i = 1 \text{ and } \alpha_i = 0)$ or $(y_i = -1 \text{ and } \alpha_i = C)$
   (d) optimise the newly selected working set
3. Terminate and return $\boldsymbol{\alpha}$.

2.3.3 Shortcuts and optimisations to the decomposition algorithm

The speed of the decomposition algorithm is hampered by many redundant computations. This section discusses some of these performance bottlenecks, and ways minimise additional computations.

Let t define a certain iteration in Algorithm 2.2. At time t, a number of factors consume the algorithm's execution time: Its efficiency greatly depends on the amount of time taken to

compute the vector $\mathbf{g} = \nabla W(\boldsymbol{\alpha}^{(t)})$ and matrices Q_{BB} and Q_{BN}. Its speed is also influenced by the time taken to compute the KKT conditions at each iteration, since it too requires the kernel matrix.

Due to the approach taken by the decomposition method, the quantities $\mathbf{g} = \nabla W(\boldsymbol{\alpha}^{(t)})$ (needed for selecting the working set) and $y_i f^*(\mathbf{x}_i)$ (needed for KKT conditions), can be defined using knowledge of only q rows of the Hessian Q. These q rows correspond to the q elements in the current working set.

For this purpose, define a vector $\mathbf{s}^{(t)}$, that is computed directly after working set selection, and is stored throughout the training iteration:

$$s_i^{(t)} = \sum_{j=1}^{l} \alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{j \in B} \alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_{j \in N} \alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.26}$$

As $\boldsymbol{\alpha}^{(t)}$ is refined, the objective function $W(\boldsymbol{\alpha}^{(t)})$ is increased by each iteration of the decomposition method. The best vector $\boldsymbol{\alpha}^{(t)}$ found in iteration t is therefore used as the vector $\boldsymbol{\alpha}^{(t+1)}$, which the decomposition method uses to select a working set for iteration $t+1$. The vector $\boldsymbol{\alpha}^{(t+1)}$ is therefore the vector that maximises $W(\boldsymbol{\alpha}^{(t)})$ over the working set B from iteration t, i.e.

$$W(\boldsymbol{\alpha}^{(t+1)}) = \max_{B} W(\boldsymbol{\alpha}^{(t)}) \tag{2.27}$$

and

$$\begin{aligned} W(\boldsymbol{\alpha}^{(t)}) &= (\boldsymbol{\alpha}^{(t)})^T \mathbf{1} - \frac{1}{2}(\boldsymbol{\alpha}^{(t)})^T Q \boldsymbol{\alpha}^{(t)} \\ &= \sum_{i=1}^{l} \alpha_i^{(t)} - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i^{(t)} \alpha_j^{(t)} y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{i=1}^{l} \alpha_i^{(t)} - \frac{1}{2} \sum_{i=1}^{l} \alpha_i^{(t)} y_i \sum_{j=1}^{l} \alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{i=1}^{l} \alpha_i^{(t)} - \frac{1}{2} \sum_{i=1}^{l} \alpha_i y_i s_i^{(t)} \end{aligned} \tag{2.28}$$

When a vector $\boldsymbol{\alpha}^{(t)}$ has been found that maximises $W(\boldsymbol{\alpha}^{(t)})$ over the working set B, the starting vector for the next iteration – which is also the best solution $\boldsymbol{\alpha}$ found thus far – is updated with $\boldsymbol{\alpha}^{(t+1)} \leftarrow \boldsymbol{\alpha}^{(t)}$. Because $\boldsymbol{\alpha}$ is updated, the value of $\mathbf{s}$ must also be updated. Since only the value of $\boldsymbol{\alpha}_B$, or the working set of variables, has changed from time t to time $t+1$, $\mathbf{s}$ is updated with

$$s_i^{(t+1)} = s_i^{(t)} + \sum_{j \in B} (\alpha_j^{(t+1)} - \alpha_j^{(t)}) y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.29}$$

Many optimisations can be implemented using definition (2.26) and simple update (2.29) of vector $\mathbf{s}$. At the start of training of a new working set, the value of $\mathbf{q}_{BN}$ from (2.19) is computed with

$$(q_{BN})_{i\in B}^{(t)} = y_i\Big(s_i^{(t)} - \sum_{j\in B}\alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j)\Big) \tag{2.30}$$

The derivative of W at time t (needed for selecting an optimal working set) is easily determined from $\mathbf{s}$, i.e.

$$\begin{aligned}\nabla W(\boldsymbol{\alpha}^{(t)})_i &= 1 - \frac{1}{2}\cdot 2y_i\sum_{j=1}^{l}\alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j)\\ &= 1 - y_i s_i^{(t)}\end{aligned} \tag{2.31}$$

By using $\mathbf{s}$, the value of the threshold b (2.16) is rewritten for each support vector as

$$\begin{aligned}b_i^{(t)} &= y_i - \sum_{j=1}^{l} y_j\alpha_j^{(t)} k(\mathbf{x}_i, \mathbf{x}_j)\\ &= y_i - s_i^{(t)}\end{aligned} \tag{2.32}$$

The value of $b^{(t)}$ is taken as the average over all the $b_i^{(t)}$ of all support vectors i.

Finally, the KKT optimality conditions specified in (2.15) are also rewritten in terms of $\mathbf{s}$, and are computed in linear time. A solution $\boldsymbol{\alpha}^{(t)}$ of (2.1) is an optimal solution if the following relations hold for each $\alpha_i^{(t)}$:

$$\begin{aligned}\alpha_i^{(t)} = 0 &\Rightarrow y_i(s_i^{(t)} + b^{(t)}) \geq 1\\ 0 < \alpha_i^{(t)} < C &\Rightarrow y_i(s_i^{(t)} + b^{(t)}) = 1\\ \alpha_i^{(t)} = C &\Rightarrow y_i(s_i^{(t)} + b^{(t)}) \leq 1\end{aligned} \tag{2.33}$$

2.4 The training algorithm

Almost all necessary tools are now gathered to create a SVM training algorithm.

In this chapter the Karush-Kuhn-Tucker conditions have been used to specify whether an optimal solution has been found and the training algorithm can terminate. A method was developed to decompose the SVM problem into more workable subproblems. Optimisations to reduce the number of computations were also introduced.

Finally, the detailed training algorithm is presented:

Algorithm 2.3 - SVM training algorithm

1. Pick an initial vector $\boldsymbol{\alpha}^{(0)}$

2. Compute the initial value of $\mathbf{s}^{(0)}$:

$$s_i^{(0)} = \sum_{j=1}^{l} \alpha_j^{(0)} y_j k(\mathbf{x}_i, \mathbf{x}_j).$$

3. Compute the initial value of b with

$$b^{(0)} = \frac{1}{SVs} \sum_{i \in SVs} (y_i - s_i^{(0)}),$$

where SVs is the total number of current support vectors.

4. Let L be a list of all l Lagrange multipliers α_i.

5. While the Karush-Kuhn-Tucker conditions in (2.33) are not met

   (a) Let $\mathbf{g} \in \mathbb{R}^l$ be defined by

$$g_i = \nabla W(\boldsymbol{\alpha}^{(t)})_i = 1 - y_i s_i^{(t)}.$$

   (b) Sort L by $y_i g_i$ in increasing order.

   (c) Select $\frac{q}{2}$ samples from the front of L such that

   - $0 < \alpha_i^{(t)} < C$ or
   - $\left(y_i = -1 \text{ and } \alpha_i^{(t)} = 0\right)$ or $\left(y_i = 1 \text{ and } \alpha_i^{(t)} = C\right)$

   (d) Select $\frac{q}{2}$ samples from the back of L such that

   - $0 < \alpha_i^{(t)} < C$ or
   - $\left(y_i = 1 \text{ and } \alpha_i^{(t)} = 0\right)$ or $\left(y_i = -1 \text{ and } \alpha_i^{(t)} = C\right)$

   (e) After selection of the elements $\boldsymbol{\alpha}_B$ in the working set B, compute the Hessian matrix Q_{BB}.

   (f) Determine the vector $\mathbf{q}_{BN}$ with

$$(q_{BN})_{i \in B}^{(t)} = y_i \Big(s_i^{(t)} - \sum_{j \in B} \alpha_j^{(t)} y_j k(\mathbf{x}_i, \mathbf{x}_j) \Big).$$

   (g) Re-optimise the working set, using

$$W(\boldsymbol{\alpha}_B) = \boldsymbol{\alpha}_B^T \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}_B^T Q_{BB} \boldsymbol{\alpha}_B - \boldsymbol{\alpha}_B^T \mathbf{q}_{BN},$$

   and constraints defined in (2.20). Replace the optimised $\boldsymbol{\alpha}_B$ into $\boldsymbol{\alpha}^{(t)}$ to get $\boldsymbol{\alpha}^{(t+1)}$.

(h) Update the vector $\mathbf{s}^{(t+1)}$ with

$$s_i^{(t+1)} = s_i^{(t)} + \sum_{j \in B} (\alpha_j^{(t+1)} - \alpha_j^{(t)}) y_j k(\mathbf{x}_i, \mathbf{x}_j).$$

(i) Recompute the value of b with

$$b^{(t+1)} = \frac{1}{SVs} \sum_{i \in SVs} (y_i - s_i^{(t+1)}).$$

(j) Increase time t with $t := t + 1$.

6. Terminate and return $\boldsymbol{\alpha}$.

There is one tool needed to complete the SVM training algorithm, and that is a routine to optimise the working set, i.e. a routine that can solve (2.20). The following chapter introduces Particle Swarm Optimisation (PSO) as a general optimisation method. Since (2.20) is a problem with linear and boxed constraints, PSO is adapted to handle linear equality and inequality constraints, and the working set can be optimised using PSO, and the SVM trained.