10 Implementation

This chapter gives an overview of methods for solving the optimization problems specific to Support Vector Machines. Algorithms specific to other settings, such as Kernel PCA and Kernel Feature Analysis (Chapter 14), Regularized Principal Manifolds (Chapter 17), estimation of the support of a distribution (Chapter 8), Kernel Discriminant Analysis (Chapter 15), or Relevance Vector Machines (Chapter 16) can be found in the corresponding chapters. The large amount of code and number of publications available, and the importance of the topic, warrants this separate chapter on Support Vector implementations. Moreover, many of the techniques presented here are prototypical of the solutions of optimization problems in other chapters of this book and can be easily adapted to particular settings.

Overview

Due to the sheer size of the optimization problems arising in the SV setting we must pay special attention to how these problems can be solved efficiently. In Section 10.1 we begin with a description of strategies which can benefit almost all currently available optimization methods, such as universal stopping criteria, caching strategies and restarting rules. Section 10.2 details low rank approximations of the kernel matrix, $K \in \mathbb{R}^{m \times m}$. These methods allow the replacement of K by the outer product $ZZ^\top$ of a "tall and skinny" matrix $Z \in \mathbb{R}^{m \times n}$ where $n \ll m$. The latter can be used directly in algorithms whose speed improves with linear Support Vector Machines (SMO, Interior Point codes, Lagrangian SVM, and Newton's method).

Subsequently we present four classes of algorithms; interior point codes, subset selection, sequential minimization, and iterative methods. Interior Point methods are explained in Section 10.3. They are some of the most reliable methods for moderate problem sizes, yet their implementation is not trivial. Subset selection methods, as in Section 10.4, act as meta-algorithms on top of a basic optimization algorithm by carving out sets of variables on which the actual optimization takes place. Sequential Minimal Optimization, presented in Section 10.5, is a special case thereof. Due to the choice of only two variables at a time the restricted optimization problem can be solved analytically which obviating the need for an underlying base optimizer. Finally, iterative methods such as online learning, gradient descent, and Lagrangian Support Vector Machines are described in Section 10.6. Figure 10.1 gives a rough overview describing under which conditions which optimization algorithm is recommended.

Prerequisites

This chapter is intended for readers interested in implementing an SVM themselves. Consequently we assume that the reader is familiar with the basic concepts of both optimization (Chapter 6) and SV estimation (Chapters 1, 7, and 9).

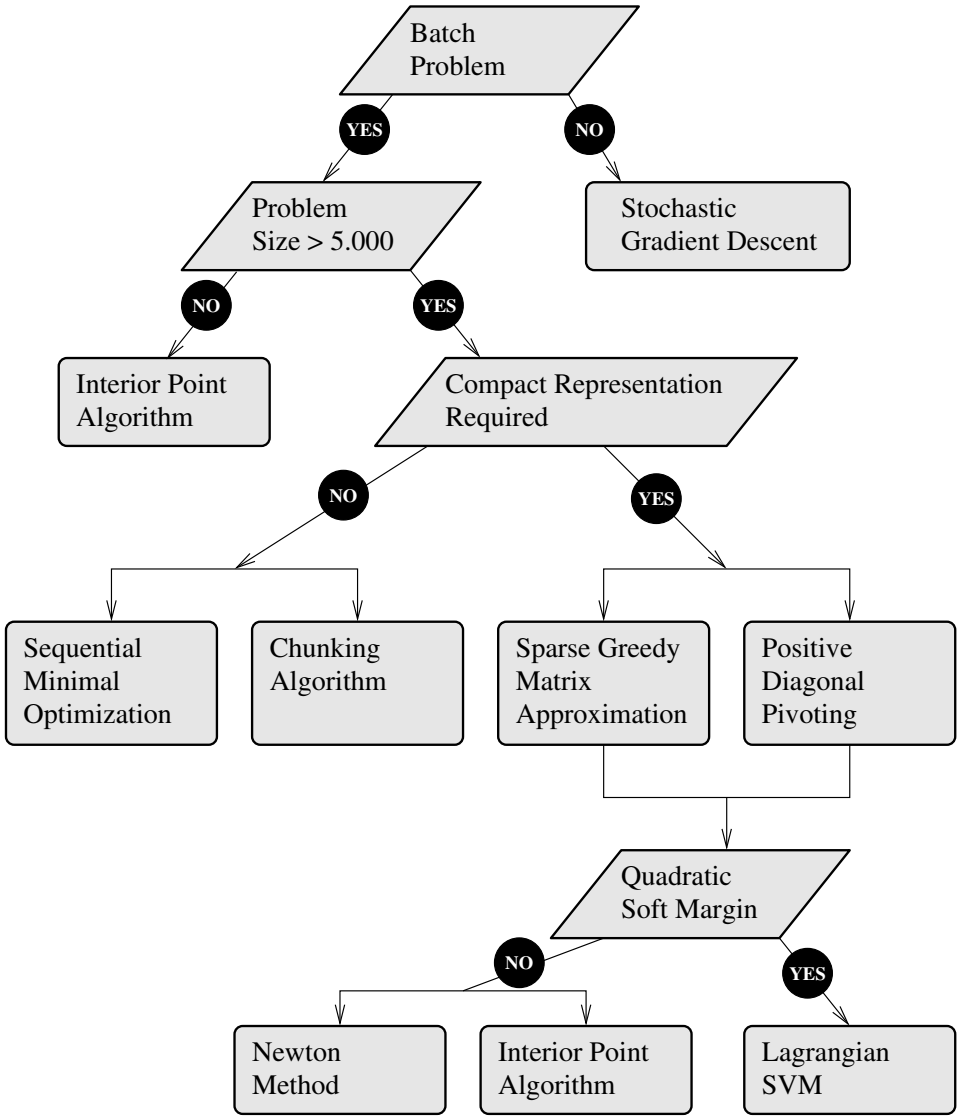

Figure 10.1 A decision tree for selecting suitable optimization algorithms. Most kernel learning problems will be batch ones (for the online setting see Section 10.6.3). For small and medium sized problems, that is as long as the kernel matrix fits into main memory, an interior point code is recommended, since it produces optima of very high quality. For larger problems approximations are required which leads to sparse greedy approximation schemes or other reduced set methods. An alternative strategy, which is particularly attractive if the size of the final kernel expansion is not important, can be found in subset selection methods such as chunking and SMO.

Knowledge of Lagrangians, duality, and optimality conditions (Section 6.3.1) is required to understand both the stopping rules in more detail and the section on interior point methods. Essentially, this chapter builds on the general exposition of Section 6.4. In addition, some of the methods from subset approximation rely on the randomized optimization concepts of Section 6.5. A basic understanding of fundamental concepts in numerical analysis, for example the notion of a Cholesky decomposition, will also prove useful, yet it is not an essential requirement. See textbooks [530, 247, 207, 46] for more on this topic.

Notation Issues

Note that in this chapter we will alternate between the ν and the C-formulation (see Section 7.5), for different algorithms. This is due to the fact that some algorithms are not capable of treating the ν-formulation efficiently. Furthermore, the C-formulation differs from the one of minimizing the regularized risk functional insofar as we minimize

$$C\sum_{i=1}^{m} c(x_i, y_i, f(x_i)) + \Omega[f] \tag{10.1}$$

rather than

$$\frac{1}{m}\sum_{i=1}^{m} c(x_i, y_i, f(x_i)) + \lambda\Omega[f]. \tag{10.2}$$

We can transform one setting into the other via $C = \frac{1}{\lambda m}$. The C notation is used in order to be consistent with the published literature on SV optimization algorithms.

10.1 Tricks of the Trade

We start with an overview of useful "tricks,"; modifications from which almost any algorithm will benefit. For instance, techniques which are useful for speeding up training significantly or which determine when the algorithm should be stopped.

This section is intended both for the practitioner who would like to improve an existing SV algorithm and also for readers new to SV optimization, since most of the tools developed prove useful in the optimization equations later. We present three tricks; a practical stopping criterion, a restart method, and an overview of caching strategies.

10.1.1 Stopping Criterion

It would be ideal if we always were able to obtain the solution by optimization methods (e.g., from Section 6.4). Unfortunately, due to the size of the problem, this is often not possible and we must limit ourselves to *approximating* the solution by an iterative strategy.

Several stopping criteria have been suggested regarding when to stop training a Support Vector Machine. Some of these focus mainly on the precision of the Lagrange multipliers α_i [266, 409, 494], whereas others [514, 459] use the proximity

of the values of the primal and dual objective functions. Yet others stop simply when no further improvement is made [398].

Before we develop a stopping criterion recall that ultimately we want to find a solution $f(x) = \langle \mathbf{w}, \Phi(x) \rangle + b$ that minimizes one of the regularized risk functionals described below. In the case of classification,

$$\begin{array}{lll} \underset{\mathbf{w},\xi}{\text{minimize}} & C\sum_{i=1}^{m} c(\xi_i) + \frac{1}{2}\|\mathbf{w}\|^2 & \sum_{i=1}^{m} c(\xi_i) + \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho \\ \text{subject to} & y_i f(x_i) \geq 1 - \xi_i \quad \text{or} & y_i f(x_i) \geq \rho - \xi_i \\ & \xi_i \geq 0 & \xi_i \geq 0, \rho \in \mathbb{R} \end{array} \tag{10.3}$$

(the right half of the equations describes the analogous setting with the ν-parameter), similarly, for regression,

$$\begin{array}{lll} \underset{\mathbf{w},\xi}{\text{minimize}} & C\sum_{i=1}^{m} c(\xi_i) + c(\xi_i^*) + \frac{1}{2}\|\mathbf{w}\|^2 & C\sum_{i=1}^{m} c(\xi_i) + c(\xi_i^*) + \frac{1}{2}\|\mathbf{w}\|^2 - \nu\epsilon \\ \text{subject to} & f(x_i) \geq y_i - \epsilon - \xi_i \quad \text{or} & f(x_i) \geq y_i - \epsilon - \xi_i \\ & f(x_i) \leq y_i + \epsilon + \xi_i^* & f(x_i) \leq y_i + \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 & \xi_i, \xi_i^* \geq 0, \epsilon \in \mathbb{R} \end{array} \tag{10.4}$$

This means that ultimately not the Lagrange multipliers α_i but rather $\mathbf{w}$, or only the value of the primal objective function, matters. Thus, algorithms [266, 290, 291, 398] which rely on the assumption that proximity to the optimal parameters will ensure a good solution may not be using an optimal stopping criterion. In particular, such a criterion may sometimes be overly conservative, especially if the influence of individual parameters on the final estimate is negligible. For instance, assume that we have a linear dependency in the dual objective function. Then there exists a linear subspace of parameters which would all be suitable solutions, leading to identical vectors $\mathbf{w}$. Therefore, convergence within this subspace may not occur and, even if it does, it would not be relevant to the quality of the solution.

Proximity in Parameters $\neq$ Proximity in Solution

What we would prefer to have is a way of bounding the distance between the objective function at the current solution f and at f_{opt}. Since (10.3) and (10.4) are both constrained optimization problems we may make use of Theorem 6.27 and lower bound the values of (10.3) and (10.4) via the KKT Gap. The advantage is that we do not have to know the optimal value in order to assess the quality of the approximate solution. The following Proposition formalizes this connection.

Proposition 10.1 (KKT-Gap for Support Vector Machines) *Denote by f the (possibly not optimal) estimate obtained during a minimizing procedure of the optimization problem (10.3) or (10.4) derived from the regularized risk functional $R_{\text{reg}}[f]$. Further, denote by f_{opt} the minimizer of $R_{\text{reg}}[f]$. Then under the condition of dual feasible variables (namely that the equality and box constraints are satisfied), the following inequality holds:*

$$R_{\text{reg}}[f] \geq R_{\text{reg}}[f^*] \geq R_{\text{reg}}[f] - \frac{1}{Cm}\text{Gap}[f] \tag{10.5}$$

where $\text{Gap}[f]$ *is defined as follows:*

1. *In the case of classification with C-parametrization*

$$\begin{aligned}\mathrm{Gap}[f] &= \sum_{j=1}^{m} \max(0, 1 - y_j f(x_j)) C \partial_{\xi_j} \tilde{c}(\max(0, 1 - y_j f(x_j))) + \alpha_j (y_j f(x_j) - 1) \\ &= \sum_{j=1}^{m} C \max(0, 1 - y_j f(x_j)) + \alpha_j (y_j f(x_j) - 1) \end{aligned} \tag{10.6}$$

2. *For classification in the ν-formulation*

$$\mathrm{Gap}[f] = \sum_{j=1}^{m} \max(0, \rho - y_j f(x_j)) + \alpha_j (y_j f(x_j) - \rho) \tag{10.7}$$

3. *For ε-regression, where $\xi_j = \max(0, y_i - f(x_i) - \varepsilon)$ and $\xi_j^* = \max(0, f(x_i) - y_i - \varepsilon)$,*

$$\begin{aligned}\mathrm{Gap}[f] &= \sum_{j=1}^{m} \xi_j C \partial_{\xi_j} \tilde{c}(\xi_j) + \xi_j^* C \partial_{\xi_j^*} \tilde{c}(\xi_j^*) + \alpha_j (\varepsilon + f(x_j) - y_j) + \alpha_j^* (\varepsilon - f(x_j) + y_j) \\ &= \sum_{j=1}^{m} C\xi_j + C\xi_j^* + \alpha_j (\varepsilon + f(x_j) - y_j) + \alpha_j^* (\varepsilon - f(x_j) + y_j) \end{aligned} \tag{10.8}$$

4. *In the ν-formulation the gap is identical to (10.8), with the only difference being that $C = 1$ and that ε is a variable of the optimization problem.*

Here $\rho = 1$ is a constant in the C-formulation and $C = 1$ is one in the ν-formulation. For regression we denote by $\tilde{c}(\xi)$ the nonzero branch of $c(x_i, y_i, f(x_i))$ which for the ε-insensitive regression setting becomes $\tilde{c}(\xi) = \xi$. Finally, note that in the ν-regression formulation, ε is a variable.

Stopping Criterion and Significant Number of Figures

Such a lower bound on the minimum of the objective function has the added benefit that it can be used to devise a stopping criterion. We simply use the same strategy as in interior point codes (Section 6.4.4) and stop when the relative size of the gap is below some threshold ϵ, that is, if

$$\mathrm{Gap}[f] \leq \epsilon \frac{|R_{\mathrm{reg}}[f]| + |R_{\mathrm{reg}}[f] - \mathrm{Gap}|}{2}. \tag{10.9}$$

Proof All we must do is apply Theorem 6.27 by rewriting (6.60) in terms of the currently used expressions and subsequently find good values for the variables that have not been specified explicitly. This will show that the size of the KKT-gap is given by (10.6), (10.7), and (10.8).

The first thing to note is that free variables do not contribute directly to the size of the KKT gap provided the corresponding equality constraints in the dual optimization problem are satisfied. Therefore it is sufficient to give the proof only for the *C*-parametrization — the ν-parametrization simply uses an additional equality constraint due to an extra free variable.

Rewriting (6.60) in terms of the SV optimization problem means that now $\mathbf{w}$ and ξ are the *variables* of the optimization problem and x_i, y_i are merely constants. We review the constraints,

$$0 \quad \geq \quad \rho - \xi_i - y_i f(x_i) \quad \text{and } 0 \quad \geq \quad -\xi_i \quad \text{(classification)} \tag{10.10}$$

$$\begin{array}{rcllcll} 0 & \geq & -f(x_i) + y_i - \epsilon - \xi_i & \text{and } 0 & \geq & -\xi_i & \text{(regression)} \\ 0 & \geq & +f(x_i) - y_i - \epsilon - \xi_i^* & \text{and } 0 & \geq & -\xi_i^* & \end{array} \tag{10.11}$$

Since the optimizations take place in dual space (the space of the corresponding Lagrange multipliers α_i, η_i), we can use the latter directly in our bound. What remains to be done is to find ξ_i, since f is determined by α_i. The constraint imposed on ξ_i is that f and the corresponding α_i satisfy the dual feasibility constraints (6.53). We obtain

$$\begin{aligned} \partial_{\xi_j} L(f, \xi, \alpha, \eta) &= \partial_{\xi_j} \left(C \sum_{i=1}^{m} \tilde{c}(\xi_i) + \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i \left(1 - \xi_i - y_i f(x_i)\right) - \eta_i \xi_i \right) \\ &= C \partial_{\xi_j} \tilde{c}(\xi_j) - \alpha_j - \eta_j = 0 \end{aligned} \tag{10.12}$$

for classification. Now we have to choose η_j, ξ_j such that $C\partial_{\xi_j}\tilde{c}(\xi_j) = \alpha_j + \eta_j$ is satisfied. At the same time we would like to obtain good lower bounds. Hence we must choose the parameters in such a way that the KKT gap (6.53) is minimal, that is, all the terms

$$\begin{aligned} \mathrm{KKT}_j &:= \eta_j \xi_j + \alpha_j \left(y_j f(x_j) - 1 + \xi_j\right) && (10.13) \\ &= \left(C\partial_{\xi_j}\tilde{c}\left(\xi_j\right) - \alpha_j\right)\xi_j + \alpha_j \left(y_j f(x_j) - 1 + \xi_j\right) && (10.14) \\ &= \xi_j C \partial_{\xi_j} \tilde{c}(\xi_j) + \alpha_j \left(y_j f(x_j) - 1\right) && (10.15) \end{aligned}$$

are minimized. The second term is independent of ξ_j and the first term is monotonically increasing with ξ_j (since $\tilde{c}$ is convex). The smallest value for ξ_j is given by (10.10). Combining of these two constraints gives

$$\xi_j = \max(0, 1 - y_j f(x_j)). \tag{10.16}$$

Together (10.16), (10.13), and (6.53) prove the bound for classification. Finally, substituting $\tilde{c}(\xi_j) = \xi_j$ (the soft-margin loss function [40, 111]) yields (10.6). For regression we proceed analogously. The optimality criteria for ξ_j, ξ_j^* and η_j, η_j^* are

$$C\partial_{\xi_j}\tilde{c}(\xi_j) - \alpha_j - \eta_j = 0 \text{ and } C\partial_{\xi_j^*}\tilde{c}(\xi_j^*) - \alpha_j^* - \eta_j^* = 0 \tag{10.17}$$

Explicit Optimization of Dual Variables

In addition, from (6.53), we obtain

$$\begin{aligned} \mathrm{KKT}_j &:= \eta_j \xi_j + \eta_j^* \xi_j^* + \alpha_j \left(\varepsilon + \xi_j + f(x_j) - y_j\right) + \alpha_j^* \left(\varepsilon + \xi_j^* - f(x_j) + y_j\right) && (10.18) \\ &= \xi_j C \partial_{\xi_j} c(\xi_j) + \xi_j^* C \partial_{\xi_j^*} c(\xi_j) + \alpha_j(\varepsilon + f(x_j) - y_j) + \alpha_j(\varepsilon + y_j - f(x_j)). && (10.19) \end{aligned}$$

By similar reasoning as before we can see that the optimal ξ_j is given by

$$\xi_j = \max(0, y_i - f(x_i) - \varepsilon) \text{ and } \xi_j^* = \max(0, f(x_i) - y_i - \varepsilon) \tag{10.20}$$

which completes the proof. ■

The advantage of (10.6) and (10.8) is that they can be computed in $O(m)$ time provided that the function values $f(x_i)$ are already known. This means that convergence checking can be done for almost no additional cost with respect to the overall cost of the training algorithm.

Non-monotonic Gap Size

An important thing to remember is that for algorithms which minimize only the dual or only the primal objective function the size of the gap may *grow* between optimization steps. This is since an improvement in the primal objective does not necessarily imply an improvement in the dual and vice versa. One way to overcome this problem (besides a redesign of the optimization algorithm which may be out of question in most cases) is to note that it immediately follows from (10.5) that

$$\min_i R_{\text{reg}}[f_i] \geq R_{\text{reg}}[f_{\text{opt}}] \geq \max_i \left[R_{\text{reg}}[f_i] - \text{Gap}[f_i]\right] \tag{10.21}$$

where f_i is the estimate at iteration i. In many algorithms such as SMO, where the dual gap can fluctuate considerably, this leads to much improved bounds on $R_{\text{reg}}[f_{\text{opt}}]$ compared to (10.5). In experiments a gap-optimal choice of b led to decreased, but still existing, fluctuations. See also [291, 494] for details how such an optimal value of b can be found.

10.1.2 Restarting with Different Parameters

Quite often we must train a Support Vector Machine for more than one specific parameter setting. In such cases it is beneficial to re-use the solution obtained for one specific parameter setting in finding the remaining ones. In particular, situations involving different choices of regularization parameter or different kernel widths benefit significantly from this parameter re-use as opposed to starting from $f = 0$. Let us analyze the situation in more detail.

Restarting for C: Denote by f_C the minimizer of the regularized risk functional (slightly modified to account for C rather than for λ)

$$R_{\text{reg}}[f, C] := C \sum_{i=1}^{m} c(x_i, y_i, f(x_i)) + \Omega[f]. \tag{10.22}$$

By construction

$$R_{\text{reg}}\left[f_C, C'\right] \geq R_{\text{reg}}\left[f_{C'}, C'\right] \geq R_{\text{reg}}\left[f_{C'}, C\right] \geq R_{\text{reg}}[f_C, C] \text{ for all } C' > C. \tag{10.23}$$

The first inequality follows from the fact that $f_{C'}$ is the minimizer of $R_{\text{reg}}\left[f, C'\right]$. The second inequality is a direct consequence of $C' > C$, and, finally, the third inequality is due to the optimality of f_C. Additionally, we conclude from (10.22) that

$$R_{\text{reg}}\left[f_{C'}, C'\right] \leq R_{\text{reg}}\left[f_C, C\right] + (C' - C) m R_{\text{emp}}\left[f_C\right] \leq \frac{C'}{C} R_{\text{reg}}\left[f_C, C\right] \tag{10.24}$$

and thus

$$\frac{C}{C'} R_{\text{reg}}\left[f_{C'}, C'\right] \leq R_{\text{reg}}\left[f_C, C\right] \leq R_{\text{reg}}\left[f_{C'}, C'\right]. \tag{10.25}$$

In other words, changes in the regularized risk functional $R_{\text{reg}}[f, C]$ are bounded by the changes in C. This has two implications; first, it does not make sense to use an overly fine grid in C when looking for minima of $R_{\text{reg}}[f]$. Second, (10.23) shows

that for changes of C into C' which are not too large it is beneficial to re-use f_C rather than to restart from $f = 0$.

Decreasing C

In practice it is often advantageous to start with a solution for large C and keep on increasing the regularization (decrease C). This has the effect that initially most of the α_i, α_i^* will be unconstrained. Subsequently, all the variables that have been found to be constrained will typically tend to stay that way. This can dramatically speed up the training phase by up to a factor of 20, since the algorithm can focus on unconstrained variables only. See [502], among others, for experimental details. In order to satisfy the dual constraints it is convenient to rescale the Lagrange multipliers in accordance with the change in C. This means we rescale each coefficient by $\alpha_i' = \frac{C'}{C}\alpha_i$, where α_i' are the start values when training with C' instead of C. Such a modification leaves the summation constraints intact. See also [288] for further details on how to adjust parameters for changed values of C.

Restarting for σ: Gaussian RBF kernels (2.68) are popular choices for Support Vector Machines. Here one problem is to adapt the width σ of the kernel suitably. In [123] it is shown that for the soft-margin loss function the minimizer (and its RKHS norm) of the regularized risk is a smooth function of σ.

More generally, if the regularization operator only changes smoothly, we can employ similar reasoning to that above. Note that in the current case not only the regularizer but also f itself changes (since we only have a representation of f via α_i). Yet, unless the change in σ is too large, the value of the regularized risk functional will be smaller than for the default guess $f = 0$, hence it is advantageous to re-use the old parameters α_i, b.

10.1.3 Caching

A simple and useful trick is to store parts of the kernel matrix K_{ij}, or also $f(x_i)$, for future use when storage of the whole kernel matrix K is impossible due to memory constraints. We have to distinguish between different techniques.

Row Cache: This is one of the easiest techniques to implement. Usually we allocate as much space for an $m_c \times m$ matrix as memory is available. Simple caching strategies as LRU (least recently used — keep only the most recently used rows of K in the cache and update the oldest rows first) can provide an 80% – 90% hit rate (= fraction of elements found in the cache) with a cache size of 10% of the original matrix. See, for example, [266, 309, 494, 134] for details. Row cache strategies work best for sequential update and subset selection methods such as SMO (see Section 10.5). Moreover, we can expect significant improvement via row cache strategies if the number of non-bound Lagrange multipliers $\alpha_i \in (0, C)$ is small, since these are the parameters revisited many times.

Hit Rate

Element Cache: A more fine-grained caching strategy would store individual elements of K rather than entire rows or columns. This has the additional advantage that for sparse solutions, where many $\alpha_i = 0$, possibly all relevant entries K_{ij} can be cached [172]. The downside is that the organization of the cache, as, for example,

a list, is considerably more complex and that this overhead[1] may easily outweigh the improvement in the hit rate in terms of kernel entries.

Function Cache: If only very few Lagrange Multipliers α_i change per iteration step, we may update $f(x_j)$ (which is useful for the KKT stopping criteria described in Section 10.1.1) cheaply.
Assume that a set $\alpha_1, \ldots, \alpha_n$ of Lagrange multipliers is changed (without loss of generality we pick the first n). Then $f(x_j)$ can be rewritten as

$$f^{\text{new}}(x_j) = \sum_{i=1}^{m} \alpha_i^{\text{new}} k(x_i, x_j) + b = f^{\text{old}}(x_j) + \left[\sum_{i=1}^{n} (\alpha_i^{\text{new}} - \alpha_i^{\text{old}}) k(x_i, x_j) \right]. \tag{10.26}$$

Numerical Precision

Note that in order to prevent numerical instabilities building up too quickly, it is advisable to update (10.26) in the way displayed rather than computing

$$f^{\text{old}}(x_j) + \sum_{i=1}^{n} \alpha_i^{\text{new}} k(x_i, x_j) - \sum_{i=1}^{n} \alpha_i^{\text{old}} k(x_i, x_j).$$

After several updates, depending on the machine precision, it is best to recalculate $f(x_j)$ from scratch.

10.1.4 Shrinking the Training Set

While it may not always be possible to carry out optimization on a subset of patterns right from the beginning, we may, as the optimization progresses, drop the patterns for which the corresponding Lagrange multipliers will end up being constrained to their upper or lower limits.

If we discard those patterns x_i with $\alpha_i = 0$ this amounts to effectively reducing the training set (see also Section 10.4.2 for details and equations). This is in agreement with the idea that only the Support Vectors will influence the decision functions. There exist several implementations which use such subset selection heuristics to improve training time [559, 111, 561, 463, 409, 172].

We describe another example of subset methods in Section 10.3 where we apply subset selection to interior point methods. In a nutshell the idea is that with decreasing KKT terms (10.13) either the constraint must be satisfied and the corresponding Lagrange multiplier vanishes or the constraint must be met exactly.

Sticky Patterns

Finally, assigning *sticky*-flags (cf. [85]) to variables at the boundaries also improves optimization. This means that once a variable is determined to be bound constrained it will remain fixed for the next few iterations. This heuristic avoids oscillatory behavior during the solution process.

1. Modern microprocessor architectures are largely limited by their memory bandwidth which means that an increased hardware cache miss rate due to non-contiguous storage of the matrix elements may affect performance quite dramatically. Furthermore such a strategy will also lead to paging operations of the operating system.

10.2 Sparse Greedy Matrix Approximation

In the following we describe what can be thought of as another useful trick. The practical significance warrants a more detailed description however. The reader only interested in the basic optimization algorithms may skip this section.

Low Dimensional Approximation is Sufficient

Sparse greedy approximation techniques are based on the observation that typically the matrix K has many small eigenvalues which could easily be removed without sacrificing too much precision.[2] This suggests that we could possibly find a subset of basis functions $k(x_i, x)$ which would minimize the regularized risk functional $R_{\text{reg}}[f]$ almost as well as the full expansion required by the Representer Theorem (Th. 4.2). This topic will be discussed in more detail in Chapter 18.

10.2.1 Sparse Approximations

In one way or another, most kernel algorithms have to deal with the kernel matrix K which is of size $m \times m$. Unfortunately the cost of computing or of storing the latter increases with $O(m^2)$ and the cost of evaluating the solution (sometimes referred to as the *prediction*) increases with $O(m)$. Hence, one idea is to pick some functions $k(x_1, \cdot), \ldots, k(x_n, \cdot)$ (for notational convenience we chose the first n, but this assumption will be dropped at a later stage) with $n \ll m$ such that we can approximate every single $k(x_i, \cdot)$ by a linear combination of $k(x_1, \cdot), \ldots, k(x_n, \cdot)$. Without loss of generality we assume that $x_1, \ldots, x_n$ are the first n patterns of the set $\{x_1, \ldots, x_m\}$. We approximate[3]

$$k(x_i, \cdot) \approx \tilde{k}_i(\cdot) := \sum_{j=1}^{n} \alpha_{ij} k(x_j, \cdot). \tag{10.27}$$

Approximation in Feature Space

As an approximation criterion we choose proximity in the Reproducing Kernel Hilbert Space $\mathcal{H}$, hence we choose α_{ij} such that the approximation error

$$\left\| k(x_i) - \sum_{j=1}^{n} \alpha_{ij} k(x_j, \cdot) \right\|_{\mathcal{H}}^2 = k(x_i, x_i) - 2 \sum_{j=1}^{n} \alpha_{ij} k(x_i, x_j) + \sum_{j,l=1}^{n} \alpha_{ij} \alpha_{il} k(x_j, x_l)$$

is minimized. An alternative would be to approximate the values of $k(x_i, \cdot)$ on X directly. The computational cost of doing the latter is much higher however (see Problem 10.4 and [514] for details). Since we wish to optimize the overall approximation quality we have to minimize the total approximation error for all i,

2. This can be seen from the results in Table 14.1.
3. Likewise we could formalize the problem as one of approximating patterns mapped into feature space; we approximate $\Phi(x_i)$ by $\tilde{\Phi}(x_i) := \sum_{i=1}^{m} \alpha_{ij} \Phi(x_j)$ and measure the goodness-of-fit via $\|\Phi(x_i) - \tilde{\Phi}(x_i)\|^2$. For a streamlined notation and to emphasize to fact that we are approximating a *function space* we will, however, use the RKHS notation.

giving

$$\mathrm{Err}(\alpha) := \sum_{i=1}^{m} ||k(x_i, \cdot) - \tilde{k}_i||^2_{\mathcal{H}} = \sum_{i=1}^{m} K_{ii} - 2\sum_{j=1}^{n} \alpha_{ij} K_{ij} + \sum_{j,j'=1}^{n} \alpha_{ij}\alpha_{ij'} K_{jj'}. \tag{10.28}$$

Here we use (as before) $K_{ij} := k(x_i, x_j)$. Since $\mathrm{Err}(\alpha)$ is a convex quadratic function in α, all we must do is set the first derivative of $\mathrm{Err}(\alpha)$ to zero in order to find the minimizer of $\mathrm{Err}(\alpha)$. Note that in the present case $\alpha \in \mathbb{R}^{m\times n}$ is a *matrix* and therefore, with some abuse of notation, we will use ∂_α to denote the derivative with respect to all matrix elements. The minimizer of (10.28) satisfies

$$\partial_\alpha \mathrm{Err}(\alpha) = -2K^{mn} + 2\alpha K^{nn} = 0. \tag{10.29}$$

Here K^{mn} is an $m \times n$ matrix with $K^{mn}_{ij} = K_{ij}$, so K^{mn} is the left sub-matrix of K. Likewise $K^{nn} \in \mathbb{R}^{n\times n}$ is the upper left sub-matrix of K. This leads to

$$\alpha_{\mathrm{opt}} = K^{mn}(K^{nn})^{-1} \tag{10.30}$$

We can exploit this property in order to determine the minimal approximation error $\mathrm{Err}(\alpha_{\mathrm{opt}})$ and properties of the matrix $\tilde{K}$ where $\tilde{K}_{ij} = \langle \tilde{k}_i, \tilde{k}_j \rangle$. The following theorem holds.

Theorem 10.2 (Properties of $\tilde{k}$ and $\tilde{K}$) *With the above definitions and (10.30), the matrices K, $\tilde{K}$, and $K - \tilde{K}$ are positive definite and*

$$\mathrm{Err}(\alpha_{\mathrm{opt}}) = \mathrm{tr}\ K - \mathrm{tr}\ \tilde{K}, \tag{10.31}$$

where $\tilde{K} = K^{mn}(K^{nn})^{-1}(K^{mn})^\top = \alpha(K^{mn})^\top = \alpha K^{nn}\alpha^\top$.

This means that we have an approximation of K in such a way that $\tilde{K}$ is strictly smaller than K (since $K - \tilde{K}$ is positive definite as well) and, furthermore, that the approximation error in terms of $\mathcal{H}$ can be computed cheaply by finding $\mathrm{tr}\ K - \mathrm{tr}\ \tilde{K}$, that is by looking at only m elements of the $m \times m$ matrix K and $\tilde{K}$.

Bounding Norms

Finally, we obtain a bound on the operator norm of $K - \tilde{K}$ by computing the trace of the difference, since the trace is the sum of all eigenvalues, and the latter are nonnegative for positive matrices. In particular, for positive definite matrices K (and their eigenvalues λ_i) we have

$$||K|| = \max_i \lambda_i \leq ||K||_{\mathrm{Frob}} = \mathrm{tr}\ KK^\top = \left(\sum_{i=1}^{m} \lambda_i^2\right)^{\frac{1}{2}} \leq \mathrm{tr}\ K = \sum_{i=1}^{m} \lambda_i. \tag{10.32}$$

Note that the Frobenius norm $||K||_{\mathrm{Frob}}$ is simply the 2-norm of all coefficients of K. (For symmetric matrices we may decompose K into its eigensystem via $K = U^\top \Lambda U$ where U is an orthogonal matrix and Λ is a diagonal matrix. This allows us to write $\mathrm{tr}\ KK^\top = \mathrm{tr}\ U^\top \Lambda U U^\top \Lambda U = \mathrm{tr}\ \Lambda^2$.)

Proof We prove the functional form of $\tilde{K}$ first. By construction we have

$$\tilde{K}_{ij} = \langle \tilde{k}_i, \tilde{k}_j \rangle = \sum_{l,l'=1}^{n} \alpha_{il}\alpha_{jl'} \langle k(x_l, \cdot), k(x_{l'}, \cdot) \rangle = \sum_{l,l'=1}^{n} \alpha_{il}\alpha_{jl'} K_{ll'} \tag{10.33}$$

and, therefore, by construction of α

$$\tilde{K} = \alpha K^{nn} \alpha^\top = K^{mn}(K^{nn})^{-1} K^{nn} (K^{nn})^{-1} (K^{mn})^\top = K^{mn}(K^{nn})^{-1}(K^{mn})^\top.$$

Next note that for optimal α we have (with $\mathbf{k}_i$ as a shorthand for the vector $(K_{i1}, \ldots, K_{in})$)

$$\begin{aligned} \|k(x_i, \cdot) - \tilde{k}_i\|^2_{\mathcal{H}} &= K_{ii} - 2\mathbf{k}_i^\top (K^{nn})^{-1} \mathbf{k}_i + \mathbf{k}_i^\top (K^{nn})^{-1} K^{nn} (K^{nn})^{-1} \mathbf{k}_i && (10.34) \\ &= K_{ii} - \mathbf{k}_i^\top (K^{nn})^{-1} \mathbf{k}_i = K_{ii} - \tilde{K}_{ii}. && (10.35) \end{aligned}$$

Summation over i proves the first part of (10.31). What remains is to prove positive definiteness of $K, \tilde{K}$, and $K - \tilde{K}$. As Gram matrices both K and $\tilde{K}$ are positive definite. To prove that $K - \tilde{K}$ is also positive definite we show that

$$K - \tilde{K} = \bar{K} \text{ where } \bar{K}_{ij} := \langle k(x_i, \cdot) - \tilde{k}_i(\cdot), k(x_j, \cdot) - \tilde{k}_j(\cdot) \rangle \tag{10.36}$$

All we have to do is substitute the optimal value of α, i.e., $\alpha = K^{mn}(K^{nn})^{-1}$ into the definition of $\bar{K}$ to obtain

$$\bar{K} = K - 2K^{mn}\alpha + \alpha K^{nn} \alpha^\top = K - K^{mn}(K^{nn})^{-1} K^{mn\top} = K - \tilde{K}. \tag{10.37}$$

■

Note that (10.36) also means that $\langle k(x_i, \cdot) - \tilde{k}_i(\cdot), k(x_j, \cdot) - \tilde{k}_j(\cdot) \rangle = \langle k(x_i, \cdot), k(x_j, \cdot) \rangle - \langle \tilde{k}_i(\cdot), \tilde{k}_j(\cdot) \rangle$.

10.2.2 Iterative Methods and Random Sets

While (10.31) can tell us how well we are able to approximate a set of m kernel functions $k(x_i, \cdot)$ by a subset of size n, it cannot be used to *predict* how large n should be. Let us first assume that we choose to pick the subset of kernel functions $k(x_i, \cdot)$ at random to approximate the full set, as suggested in the context of Gaussian Processes [603]. We will present a more efficient method of choosing the kernel functions in the next section but, for the moment, assume that the selection process is completely random.

Given that, for some n, we have already computed the values of $(K^{nn})^{-1}$ and $\alpha_{\text{opt}} = (K^{nn})^{-1} K^{mn}$. For an additional kernel function, say $k(x_{n+1}, \cdot)$ we need to find a way to compute the values of α_{opt} and $(K^{n+1,n+1})^{-1}$ efficiently (since the difference between K^{nn} and $K^{n+1,n+1}$ is only of rank 1 such a change is commonly referred to as a rank-1 update). We may either do this directly or use a Cholesky decomposition for increased numerical stability. For details on the latter strategy see [423, 530, 247] and Problem 10.5.

Rank-1 Update

Denote by $\mathbf{k} \in \mathbb{R}^n$ the upper right vector $\left(K_{n+1,1}, \ldots, K_{n+1,n}\right)$ of the matrix $K^{n+1,n+1} \in \mathbb{R}^{(n+1)\times(n+1)}$ to be inverted, and $\kappa := K_{n+1,n+1}$. Then we have

$$\left(K^{n+1,n+1}\right)^{-1} = \begin{bmatrix} K^{nn} & \mathbf{k}^\top \\ \mathbf{k} & \kappa \end{bmatrix}^{-1} = \begin{bmatrix} (K^{nn})^{-1} + \eta^{-1}\mathbf{v}\mathbf{v}^\top & -\eta^{-1}\mathbf{v} \\ -\eta^{-1}\mathbf{v}^\top & \eta^{-1} \end{bmatrix} \tag{10.38}$$

where $\eta = (\kappa - \mathbf{k}^\top (K^{nn})^{-1}\mathbf{k})$ and $\mathbf{v} = ((K^{nn})^{-1}\mathbf{k})$. This means that computing $(K^{n+1,n+1})^{-1}$ costs $O(n^2)$ operations once $(K^{nn})^{-1}$ is known. Next we must update α. Splitting $K^{m,n+1}$ into K^{mn} and $\bar{\mathbf{k}} = (K_{1,n+1}, \ldots K_{m,n+1})$ yields

$$\alpha = K^{m,n+1}\left(K^{n+1,n+1}\right)^{-1} \tag{10.39}$$

$$= \left[K^{mn}(K^{nn})^{-1} + \eta^{-1}\left[K^{mn}\mathbf{v} - \bar{\mathbf{k}}\right]\mathbf{v}^\top, -\eta^{-1}\left[K^{mn}\mathbf{v} - \bar{\mathbf{k}}\right]\right]. \tag{10.40}$$

Computing (10.39) involves $O(mn)$ operations since $K^{mn}(K^{nn})^{-1}$ is the old value of α for the case of n basis functions and the most expensive part of the procedure. Computing $K^{mn}\mathbf{v}$ requires only $O(mn)$ operations and the approximation error $\mathrm{Err}(\alpha)$ can be computed efficiently. It is given by

$$\mathrm{Err}(\alpha) = \mathrm{tr}\ K - \mathrm{tr}\ \alpha(K^{m,n+1})^\top = \mathrm{tr}\ K - \sum_{i=1}^{m}\sum_{j=1}^{n}\alpha_{ij}K_{ij}. \tag{10.41}$$

Since $\tilde{K} = K^{m,n+1}\alpha^\top$ we only have to account for the changes in α and the additional row due to $K^{m,n+1}$. Without going into further details one can check that (10.41) can be computed in $O(m)$ time, provided the previous value of n is already known.

Overall, successive applications of a rank-1 update method to compute a sparse approximation using n kernel functions to approximate a set of m incurs a computational cost of $O\left(\sum_{n'=1}^{n} mn'\right) = O(mn^2)$. One can see that this is only a constant factor worse than a direct calculation. Besides that, the memory footprint of the algorithm is also only $O(mn)$ which can be significantly smaller than storage of the matrix K, namely $O(m^2)$.

Computational Cost

Note the similarity in computational cost between this iterative method and Conjugate Gradient Descent methods (Section 6.2.4) where the inverse of K was effectively constructed by building up a n-dimensional subspace of conjugate directions. The difference is that we never actually need to compute the full matrix K.[4]

10.2.3 Optimal and Greedy Selections

We showed that the problem of finding good coefficients α can be solved efficiently once a set of basis functions $k(x_1, \cdot), \ldots, k(x_n, \cdot)$ is available. The problem of selecting a good subset is the more difficult issue. One can show that even relatively simple problems such as one-target optimal approximation are NP-hard [381]; we cannot expect to find a solution in polynomial time.

We can take a greedy approach in the spirit of [381, 474] (see Section 6.5.3), with the difference being that we are not approximating one single target function but a set of m basis functions. This means that, rather than picking the functions

4. Strictly speaking, it is also the case that conjugate gradient descent does not require computation of K but only of $K\alpha$ for $\alpha \in \mathbb{R}^m$.

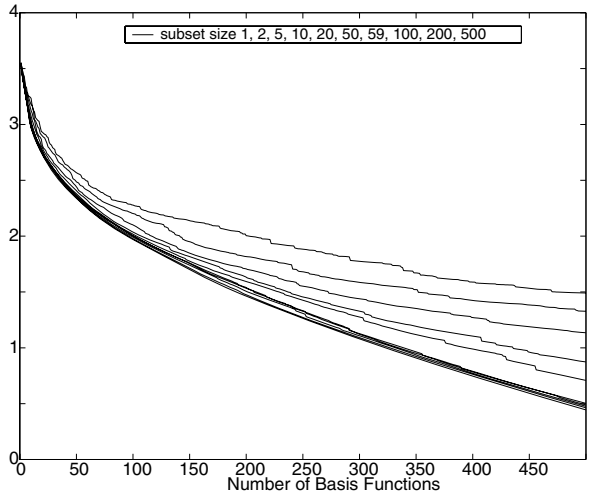

Figure 10.2 Size of the residuals $\log_{10} \operatorname{tr}(K - \tilde{K})$ for the `Abalone` dataset from the UCI repository [56]. From top to bottom: subsets of size 1, 2, 5, 10, 20, 50, 59, 100, 200. Note that, for subsets of size 50 or more, no noticeable difference in performance can be observed. After rescaling each input individually to zero mean and unit variance we used a Gaussian kernel (2.68) with $2\sigma^2 = N = 13$ where N is the dimension of the data. The size of the overall matrix was $m = 3000$.

at random, we choose one function at a time depending on which of them will decrease $\mathrm{Err}(\alpha)$ most and add this function to the set of kernels I chosen so far. Next we recompute the residual $\mathrm{Err}(\alpha)$ and continue iterating.

It would be wasteful to compute a full update for α and $(K^{n+1,n+1})^{-1}$ for every possible candidate since all we are interested in is the change in $\mathrm{Err}(\alpha)$. Simple (but tedious) algebra yields that, with the definitions $\mathbf{k}, \mathbf{v}$, and $\bar{\mathbf{k}}$ of Section 10.2.2,

$$\begin{aligned}
\mathrm{Err}(\alpha^{m,n+1}) &= \operatorname{tr} K - \operatorname{tr} \alpha^{m,n+1}(K^{m,n+1})^\top \\
&= \operatorname{tr} K - \operatorname{tr} \begin{bmatrix} K^{mn}(K^{nn})^{-1} + \eta^{-1}\left[K^{mn}\mathbf{v} - \bar{\mathbf{k}}\right]\mathbf{v}^\top \\ -\eta^{-1}\left[K^{mn}\mathbf{v} - \bar{\mathbf{k}}\right] \end{bmatrix} \begin{bmatrix} K^{mn} \\ \bar{\mathbf{k}} \end{bmatrix}^\top \\
&= \mathrm{Err}(\alpha^{mn}) - \eta^{-1}\left\|K^{mn}\mathbf{v} - \bar{\mathbf{k}}\right\|^2. \qquad (10.42)
\end{aligned}$$

Hence, our selection criterion is to find that function $k(x_i, \cdot)$ for which the decrease in the approximation error $\eta^{-1}\left\|K^{mn}\mathbf{v} - \bar{\mathbf{k}}\right\|^2$ is largest. This method still has a downside — at every step we would have to test for the $m - n$ different remaining kernel functions $k(x_i, \cdot)$ to find which would yield the largest improvement. With this the cost per iteration would be $O(mn(m - n))$ which is clearly infeasible.

Problems with Naive Greedy Methods

The key trick is not to analyze the complete set of $m - n$ functions but to pick a random subset instead. Section 6.5.1 and in particular Theorem 6.33 tell us that a random subset of size $N = 59$ is sufficiently large to yield a function $k(x_i, \cdot)$ which is, with 95% confidence, better than 95% of all other kernel functions.

Random Subsets

Figure 10.2 shows that, in practice, subsets of 59 yield results almost as good as when a much larger set or even the complete dataset is used to find the next basis function. Note the rapid decay in $\ln \mathrm{Err}(\alpha) = \ln \operatorname{tr}(K - \tilde{K})$.

We conclude this section with the pseudocode (Algorithm 10.1) needed to find a sparse greedy approximation of K and $k(x_i, \cdot)$ in the Reproducing Kernel Hilbert Space $\mathcal{H}$. Note that the cost of the algorithm is now $O(Nmn^2)$, hence it is of the

Algorithm 10.1 Sparse Greedy Matrix Approximation (SGMA)

input basis functions k_i, bound on residuals ϵ
  $n = 0, I = \{\}, \alpha = [], K^{mn} = 0$
  repeat
    $n{+}{+}$
    Draw random subset M of size N from $[m] \backslash I$
    {**Select best basis function**}
    for all $j \in M$ **do**
      $\mathbf{v} = (K^{nn})^{-1}\mathbf{k}$
      $\eta = \kappa - \mathbf{v}^\top \mathbf{k}$
      $\mathrm{Err}(\alpha^{mn}) - \mathrm{Err}(\alpha^{m,n+1}) = \eta^{-1}\|K^{mn}\mathbf{v} - \bar{\mathbf{k}}\|^2$
    end for
    Select best $\hat{\imath} \in M$ and update $I = I \cup \{\hat{\imath}\}$.
    Update $(K^{n+1,n+1})^{-1}$ and $\alpha^{m,n+1}$ from (10.38) and (10.39).
    Update $\mathrm{Err}(\alpha^{m,n+1})$
  until $\mathrm{Err}(\alpha^{m,n+1}) < \epsilon$
output $n, \alpha, I, \mathrm{Err}(\alpha^{m,n+1})$

same order as an algorithm choosing kernel functions at random.[5]

10.2.4 Experiments

To illustrate the performance of Sparse Greedy Matrix Approximation (SGMA) we compare it with a conventional low-rank approximation method, namely PCA. The latter is optimal in terms of finite dimensional approximations (see Problem 10.7), however, it comes at the expense of requiring the full set of basis functions. We show in experiments that the approximation rates when SGMA is used are not much worse than those obtained with PCA. Experimental evidence also shows that the generalization performance of the two methods is very similar (see [513] for more details). Figure 10.3 shows that under various different choices of a Hilbert space (we varied the kernel width σ) the approximation quality obtained from SGMA closely resembles that of PCA.

Since SGMA picks individual basis functions $k(x_i, \cdot)$ with corresponding observations x_i we may ask whether the so-chosen x_i are special in some way. Figure 10.4 shows the first observations for the USPS dataset of handwritten digits (Gaussian RBF kernels with width $2\sigma^2 = 0.5 \cdot N$ where $N = 16 \times 16$ pixels). Note that among the first 15 observations (and corresponding basis functions) chosen on the USPS database, all 10 digits appear at least once. The pair of ones is due to a horizontal shift of the two images with respect to each other. This makes them almost

5. If the speed of prediction is not of utmost importance, random subset selection instead of the '59-trick' may just be good enough, since it is $N = 59$ times less expensive per basis function but will typically use only four times as many basis functions. With a run-time which is quadratic in the number n of basis functions this may lead to an effective speed up, at the expense of a larger memory footprint.

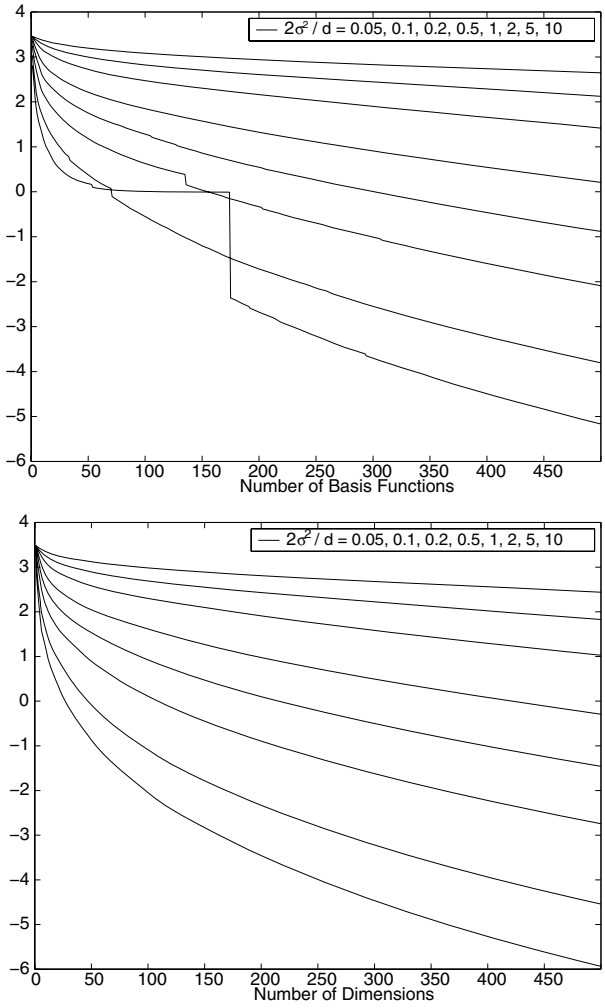

Figure 10.3 Size of the residuals for the `Abalone` dataset. Top: size of the residual trace ($\log_{10} \operatorname{tr}(K - \tilde{K})$) for projection on basis functions given by the greedy approximation scheme. Bottom: size of the residuals ($\log_{10} \sum_{j=i}^{m} \lambda_j$) for projection on subspaces given by principal component analysis. In both graphs, from top to bottom: $2\sigma^2/d = 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10$.

orthogonal to each other in feature space — their bitmaps hardly overlap at all.

Open Problem: Performance Guarantee

It is still an open question how and whether the good approximation qualities shown in Figures 10.3 can be guaranteed theoretically (3 orders of magnitude approximation with fewer than 10% of the basis functions). This need not be of any concern for the practitioner since he or she can always easily observe when the algorithm works (yet a theoretical guarantee would be nice to have).

In practice, generalization performance is more important than the question of whether the initial class of functions can be approximated well enough. As experimental evidence shows [514] the size of $\operatorname{tr}(K - \tilde{K})$, that is, the residual error, is conservative in determining the performance of a reduced rank estimator. For modest values of approximation, such as 2 orders of magnitude reduction in $\operatorname{tr}(K - \tilde{K})$, the performance is as good as the one of the estimator obtained without approximations. In some cases, such a sparse approximation may provide better

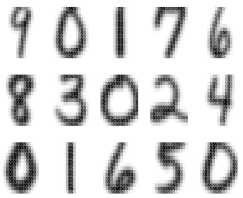

Figure 10.4 Left to Right, Top to Bottom. Patterns corresponding to the first basis functions. Note that 9 out of 10 digits are chosen among the first 10 patterns and that all patterns are sufficiently different (or shifted).

performance.

A practical use of SGMA lies in the fact that it allows us to find increasingly accurate, yet sparse approximations $\tilde{K}$ which can subsequently be used in optimization algorithms as replacements for K. It is also worth noting that the approximation and training algorithms can be coupled to obtain fast methods that do not require computation of K. Many methods find a dense expansion first and only subsequently employ a reduced set algorithm (Chapter 18) to find a more compact representation.

10.3 Interior Point Algorithms

Interior point algorithms are some of the most reliable and accurate optimization techniques and are the method of choice for small and moderately sized problems. We will discuss approximations applicable to large-scale problems in Sections 10.4, 10.5 and 10.6. We assume that the reader is familiar with the basic notions of interior point algorithms. Details can be found in Section 6.4 and references therein. In this section we focus on Support Vector specific details.

In order to deal with optimization problems which have both equality constraints and box constrained variables, we need to extend the notation of (6.72) slightly. The following optimization problem is general enough to cover classification, regression, and novelty detection:

$$\begin{array}{ll} \underset{\alpha,t}{\text{minimize}} & \frac{1}{2}\alpha^\top Q\alpha + c^\top\alpha \\ \text{subject to} & A\alpha = d \\ & \{0 \leq \alpha \leq u\} \text{ or } \{\alpha + t = u \text{ and } \alpha, t \geq 0\}. \end{array} \tag{10.43}$$

Here Q is a square matrix, typically of size $m \times m$ or $(2m) \times (2m)$, c, α, t, u are vectors of the same dimension, and A is a corresponding rectangular matrix. The dual can be found to be

$$\begin{array}{ll} \underset{\alpha,s,z,y}{\text{minimize}} & -\frac{1}{2}\alpha^\top Q\alpha + d^\top h - u^\top s \\ \text{subject to} & Q\alpha + c - A^\top h + s - z = 0 \\ & s, z \geq 0 \text{ and } h \text{ free} \end{array} \tag{10.44}$$

Furthermore, we have the Karush-Kuhn-Tucker (KKT) conditions

$$\alpha_i z_i = 0 \text{ and } s_i t_i = 0 \text{ for all } i \in [m] \tag{10.45}$$

which are satisfied at optimality.

Note that besides the primal variables α and t of the optimization problem (10.43), also the dual variables, such as s, z, and h carry information. Recall the reasoning of Section 6.3.3, where we showed that the dual-dual of a linear program is the original linear program. One may exploit this connection in the context of (10.43) and (10.44), since the latter is similar to the initial problem of minimizing regularized risks. In particular, the $A\alpha = d$ stems from the free variables (such as b or the semiparametric coefficients β_i). Consequently the dual variables h of (10.44) agree with b (or the semiparametric coefficients β_i). In practice this means that we need not perform any additional calculation in order to obtain b if we use an interior point code.

Dual-Dual Trick

10.3.1 Solving the Equations

As in Section 6.4.2 we solve the optimization problem by simultaneously satisfying primal and dual feasibility conditions and a relaxed version of (10.45). Linearization by analogy with (6.82) leads to

Linearization

$$\begin{aligned}
A\Delta\alpha &= d - A\alpha &&= \rho_{p1} \\
\Delta\alpha + \Delta t &= u - \alpha - t &&= \rho_{p2} \\
Q\Delta\alpha - A^\top \Delta h + \Delta s - \Delta z &= -Q\alpha - c + A^\top h - s + z &&= \rho_d \\
\alpha_i^{-1} z_i \Delta\alpha_i + \Delta z_i &= \mu\alpha_i^{-1} - z_i - \alpha_i^{-1}\Delta\alpha_i \Delta z_i &&=: \rho_{\mathrm{KKT1}} \\
t_i^{-1} s_i \Delta t_i + \Delta s_i &= \mu t_i^{-1} - s_i - t_i^{-1}\Delta t_i \Delta s_i &&=: \rho_{\mathrm{KKT2}}
\end{aligned} \tag{10.46}$$

Solving for $\Delta t, \Delta z, \Delta s$ yields

$$\begin{aligned}
\Delta t &= \rho_{p2} - \Delta\alpha \\
\Delta z_i &= \rho_{\mathrm{KKT1}} - \alpha_i^{-1} z_i \Delta\alpha_i \\
\Delta s_i &= \rho_{\mathrm{KKT2}} + t_i^{-1} s_i \Delta\alpha_i
\end{aligned} \tag{10.47}$$

and the *reduced KKT system* (see (6.83))

$$\begin{bmatrix} \left(Q + \mathrm{diag}(t^{-1}s + \alpha^{-1}z)\right) & -A^\top \\ -A & 0 \end{bmatrix} \begin{bmatrix} \Delta\alpha \\ \Delta h \end{bmatrix} = \begin{bmatrix} \rho_d + \rho_{\mathrm{KKT1}} - \rho_{\mathrm{KKT2}} \\ \rho_{p1} \end{bmatrix}. \tag{10.48}$$

Eq. (10.48) is best solved by a standard Cholesky decomposition of the upper left submatrix and explicit solution for the remaining parts of the linear system.[6] For details of the predictor and corrector iteration strategies, the updates of μ,

Predictor Corrector Strategy

6. Pseudocode for a Cholesky decomposition can be found in most numerical analysis textbooks such as [423]. If $\left(Q + \mathrm{diag}(t^{-1}s + \alpha^{-1}z)\right)$ should happen to be ill-conditioned, as may occur in rare cases during the iterations then it is recommended to use the pseudo-inverse or the Bunch-Kaufman decomposition [83] as a fall-back option. Linear algebra libraries such as LAPACK typically contain optimized versions of these algorithms.

convergence monitoring via the KKT gap and regarding initial conditions we refer the reader to Section 6.4.

10.3.2 Special Considerations for Classification

We now consider the particular case of SV classification and assign values to Q, c, A, and u. For standard SV classification we obtain

$$\begin{array}{lcll} Q_{ij} & = & y_i y_j k(x_i, x_j) & \text{where } Q \in \mathbb{R}^{m \times m} \\ c & = & (1, \ldots, 1) & \text{where } c \in \mathbb{R}^m \\ u & = & (C, \ldots, C) = (\frac{1}{m\lambda}, \ldots, \frac{1}{m\lambda}) & \text{where } u \in \mathbb{R}^m \\ A & = & (y_1, \ldots, y_m) & \text{where } A \in \mathbb{R}^m \\ d & = & 0 & \text{where } b \in \mathbb{R} \end{array} \tag{10.49}$$

For ν-classification (see Section 7.2), the parameters A and d are changed to

$$\begin{array}{lcl} A & = & \begin{bmatrix} y_1, \ldots, y_m \\ 1, \ldots, 1 \end{bmatrix} \\ d & = & \begin{bmatrix} 0 \\ C\nu \end{bmatrix}. \end{array} \tag{10.50}$$

In addition, we now can give meaning to the variables t, z, and s. For each α_i there exists a dual variable z_i for which $\alpha_i z_i \to 0$ as the iteration advances, due to the KKT conditions. This means that either $z_i \to 0$ or $\alpha_i \to 0$. The practical consequence of this is that we can eliminate α_i before the algorithm converges completely. All we need do is look at the size of the entries $\alpha_i^{-1} z_i$. If they exceed a certain threshold, say c, we eliminate the corresponding set of variables $(t_i, z_i, s_i, \alpha_i)$ from the optimization process, since the point will not be a Support Vector (see also Section 10.4 for details on how the optimization problem changes).

Removing Patterns

The advantage is twofold; first, this reduces computational requirements since the size of the matrix to be inverted decreases. Second, the condition of the remaining reduced KKT system improves (large entries on the main diagonal can worsen the condition significantly) which allows for faster convergence. A similar reasoning can be applied to t_i and s_i. Again, if $t_i^{-1} s_i \to \infty$ this indicates that the corresponding patterns will be a Support Vector with, however, the coefficient α_i hitting the upper boundary u_i. Elimination of this variable is slightly more complicated since we have to account for it in the equality constraints $A\alpha = d$ and update d accordingly.

As far as computational efficiency is concerned, plain interior point codes without any further modifications, are a good choice for data sets of size up to $10^3 - 10^4$ since they are simple to implement, reliable in convergence, and very precise (the size of the KKT gap is several orders of magnitude smaller than what could be achieved by SMO or other chunking algorithms). Fast numerical mathematics packages such as BLAS [316, 145] and LAPACK [11] are crucial in this case though.

Beyond that, the memory footprint and the computational cost of performing a Cholesky decomposition of Q is too expensive. We will show in Section 10.3.4 how to overcome this restriction.

10.3.3 Special Considerations for SV Regression

In the generic case we have

$$\begin{aligned} Q &= \begin{bmatrix} K & -K \\ -K & K \end{bmatrix} && \text{where } K \in \mathbb{R}^{m \times m} \\ c &= (y_1 - \varepsilon, \ldots, y_m - \varepsilon, -y_1 - \varepsilon, \ldots, -y_m - \varepsilon) && \text{where } c \in \mathbb{R}^{2m} \\ u &= (C, \ldots, C) = (\tfrac{1}{m\lambda}, \ldots, \tfrac{1}{m\lambda}) && \text{where } u \in \mathbb{R}^{2m} \\ A &= (1, \ldots, 1, -1, \ldots, -1) && \text{where } A \in \mathbb{R}^{2m} \\ d &= 0 \end{aligned} \tag{10.51}$$

The constraint matrix A changes analogously to (10.49) if we use ν regression, that is, we have two constraints rather than one. Another minor modification allows us to also deal with Huber's robust regression. All we need do is define Q as $Q = \begin{bmatrix} K + D & -K \\ -K & K + D' \end{bmatrix}$, where D is a positive definite diagonal matrix. Since the reduced KKT system only modifies Q by adding diagonal terms, we can solve both Huber's robust regression and the generic case using identical code.

The key trick in inverting Q and the matrices derived from Q by addition to the main diagonal is to exploit the redundancy of its off-diagonal elements. By an orthogonal transform

Speedup via Orthogonal Transform

$$O := \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{1} & \mathbf{1} \\ -\mathbf{1} & \mathbf{1} \end{bmatrix} \tag{10.52}$$

one obtains

$$O^\top Q O = \begin{bmatrix} 2K + \frac{D+D'}{2} & \frac{D-D'}{2} \\ \frac{D-D'}{2} & \frac{D+D'}{2} \end{bmatrix}. \tag{10.53}$$

Therefore, the $O^\top Q O$ system can be inverted essentially by inverting an $m \times m$ matrix instead of a $2m \times 2m$ system. "Essentially," since in addition to the inversion of $2K + \frac{D+D'}{2}$ we must solve for the diagonal matrices $D \pm D'$. The latter is simply an operation of computational cost $O(m)$. Furthermore, $Q^{-1} = O(O^\top Q O)^{-1} O^\top$.

All other considerations are identical to those for classification. Hence we can use the slightly more efficient matrix inversion with (10.53) as a drop-in replacement of a more pedestrian approach. This is the additional advantage we can gain from a direct implementation instead of using an off the shelf optimizer such as LOQO [556] or CPLEX [117]; in general the latter will not be able to exploit the special structure of Q.

Obtaining b,ε

Finally, note that by solving the primal and dual optimization problem simulta-

neously we also compute parameters corresponding to the initial SV optimization problem. This observation is useful as it allows us to obtain the constant term b directly, namely by setting $b = h$. See Problem 6.13 for details.

10.3.4 Large Scale Problems

The key stumbling block encountered in scaling interior point codes up to larger problems is that Q is of the size of m. Therefore, the cost of storing or of inverting Q (or any matrices derived from it) is prohibitively high. A possible solution to large problems is to use an efficient storage method for Q by low rank approximations.

Using Low Rank

Linear Systems Assume for a moment that we are using a linear SV kernel only, namely $k(x, x') = \langle x, x' \rangle$. In this case $K = XX^\top$ where $X \in \mathbb{R}^{m \times n}$ is (with slight abuse of notation) the matrix of all the training patterns, that is $X_{ij} = (x_i)_j$. Ferris and Munson [168] use this idea to find a more efficient solution for a linear SVM. They employ the Sherman-Woodbury-Morrison formula [207] (see also Section 16.4.1 for further applications) which gives

$$(V + RHR^\top)^{-1} = V^{-1} - V^{-1}R(H^{-1} + R^\top V^{-1} R)^{-1} R^\top V^{-1}, \tag{10.54}$$

Swapping Data to Disk

to invert $\left(Q + \operatorname{diag}(t^{-1}s + \alpha^{-1}z)\right)$ efficiently (see Problem 10.10). In particular they use out-of-core storage of the data (i.e. storage on the hard disk) in order to be able to deal with up to 10^8 samples.

Approximation by Low Rank Matrix

Nonlinear Systems For nonlinear SVMs, unfortunately, an application of the same technique is not directly feasible since storage requirements for K are enormous. But we can rely on the matrix approximation techniques of Section 10.2 and approximate K by $\tilde{K} = K^{mn}(K^{nn})^{-1}(K^{mn})^\top$. We only give the equations for SV classification where $Q = K$. The extension to regression is straightforward — all we must do is apply the orthogonal transform (10.53) beforehand. The problem is then solved in $O(mn^2)$ time since

$$\left(K^{mn}(K^{nn})^{-1}(K^{mn})^\top + D\right)^{-1} = \tag{10.55}$$
$$D^{-1} - D^{-1}K^{mn}\left(K^{nn} + (K^{mn})^\top D K^{mn}\right)^{-1}(K^{mn})^\top D^{-1}.$$

The expensive part is the matrix multiplications with K^{mn} and the storage of K^{mn}. As before, in the linear case, we resort to out-of-core storage (we write the matrix K^{mn} to disk once it has been computed). By this we obtain a preliminary solution using a subset consisting of only n basis functions.

Iterative Improvement If further precision is required we need to decrease the size of the problem by eliminating patterns for which the corresponding value of α can be reliably predicted. We can do this by dropping those patterns which have the largest distance from the boundary (either with $\alpha_i = 0$ or α_i at the upper constraint). These patterns are equivalent to those with the largest corresponding dual Lagrange multipliers, since these are least likely to change their value when we increase the precision of our method (see Section 10.3.2). The reduced size m of

Dimension vs. Patterns Tradeoff

the training set then allows us to store matrices with larger n and to continue the optimization on the smaller dataset. This is done until no further minimization of the regularized risk functional $R_{\text{reg}}[f]$ is observed, or until the computational restrictions of the user are met (the maximum number of basis functions we are willing to use for the solution of the optimization problem is attained).

Parallelization Note that this formulation lends itself to a parallel implementation of a SV training algorithm since the matrix multiplications can readily be implemented on a cluster of workstations by using matrix manipulation libraries such as SCALAPACK [144] or BLACS [146].

Stopping Rule Finally, if desired, we could also use the value of the primal objective function (10.43), namely $\frac{1}{2}\alpha^\top Q\alpha + c^\top\alpha$, with K rather than $\tilde{K}$ as an upper bound for the minimum of (10.43). This is due to Theorem 10.2 which states that $\alpha^\top\tilde{K}\alpha \leq \alpha^\top K\alpha$. We may not always want to use this bound, since it requires computation of K which may be too costly (but given that it is provided only as a performance guarantee this may not be a major concern).

10.4 Subset Selection Methods

In many applications the precision of the solution (in terms of the Lagrange multipliers α_i) is not a prime objective. In other situations we can reasonably assume that a large number of patterns will either become SVs with α_i constrained to the boundary, or will not become SVs at all. In any of these cases we may break the optimization problem up into small manageable subproblems and solve these iteratively. This technique is commonly referred to as chunking, or subset selection.

10.4.1 Chunking

The simplest chunking idea, introduced in [559], relies on the observation that only the SVs contribute to the final form of the hypothesis. In other words — if we were given only the SVs, we would obtain *exactly* the same final hypothesis as if we had the full training set at our disposal (see Section 7.3). Hence, knowing the SV set beforehand and, further, being able to fit it (and the dot product matrix) into memory, one could directly solve the reduced problem and thereby deal with significantly larger datasets.

Training on SVs

The catch is, that we do *not* know the SV set before solving the problem. The heuristic is to start with an arbitrary subset; a first *chunk* that fits into memory. We then train the SV algorithm on it, keep the SVs, while discarding the non-SV data in the chunk, and replace it with data on which the current estimator would make errors (for instance, data lying outside the ε-tube of the current regression). The system is then retrained and we keep on iterating until the KKT-conditions are satisfied for all samples.

10.4.2 Working Set Algorithms

The problem with chunking is that the strategy will break down on datasets where the dot-product matrix built from SVs cannot be suitably kept in memory. A possible solution to this dilemma was given in [398]. The idea is to have only a subset of the variables as a working set, and optimize the problem with respect to them while *freezing* the other variables. In other words, to perform coordinate descent on subsets of variables. This method is described in detail in [398, 266] for the case of pattern recognition. Further information can be found in [459], for example.

In the following we describe the concept behind optimization problems of type (10.43). Subsequently we will adapt it to regression and classification. Let us assume that there exists a subset $S_w \subset [m]$, also referred to as the *working set*, which is an index set determining the variables α_i we will be using for optimization purposes. Next denote by $S_f = [m] \backslash S_w$ the *fixed set* of variables which will not be modified in the current iteration. Finally, we split up Q, c, A, and u accordingly into

Splitting up Q

$$Q = \begin{bmatrix} Q_{ww} & Q_{fw} \\ Q_{wf} & Q_{ff} \end{bmatrix} \tag{10.56}$$

and $c = (c_w, c_f)$, $A = \begin{bmatrix} A_w & A_f \end{bmatrix}$, $u = (u_w, u_f)$. In this case (10.43) reads as

$$\begin{array}{ll} \underset{\alpha_w}{\text{minimize}} & \frac{1}{2}\alpha_w^\top Q_{ww}\alpha_w + \left[c_w + Q_{wf}\alpha_f\right]^\top \alpha + \\ & \left[\frac{1}{2}\alpha_f^\top Q_{ff}\alpha_f + c_f^\top \alpha_f\right] \\ \text{subject to} & A_w\alpha_w = d - A_f\alpha_f \\ & \{0 \le \alpha_w \le u_w\} \text{ or } \{\alpha_w + t_w = u_w \text{ and } \alpha_w, t_w \ge 0\} \end{array} \tag{10.57}$$

This means that we recover a standard convex program in a subset of variables, with the only difference being that the linear constraints and the linear contribution have to be changed due to their dependency on α_f. For the sake of completeness we keep the constant offset produced by α_f but it need not be taken into account during the actual optimization process.

Minimizing (10.57) with respect to α_w will also decrease (10.43). In particular, the amount of progress on the subset is identical to the progress on the whole[7]. For these subsets we may use any optimization algorithm we prefer; interior point codes for example. Algorithm 10.2 contains the details. The main difficulty in implementing subset selection strategies is how to choose S_w and S_f. We will address this issue in the next section.

Progress on Subset = Progress on Full Set

Under certain technical assumptions (see [289, 96]) Algorithm 10.2 can be shown to converge to a global minimum. Several variants of such working set algorithms

7. Such a decrease will always occur when the optimality conditions in (10.57) are not satisfied and, of course, we choose working-sets with this property.

Algorithm 10.2 Subset Selection

input kernel k, data X, precision ϵ
  Initialize $\alpha_i, \alpha_i^* = 0$
  repeat
    Compute coupling terms $Q_{wf}\alpha_f$ and $A_f\alpha_f$ for S_w.
    Solve reduced optimization problem on S_w.
    Choose new S_w from variables α_i, α_i^* not satisfying the KKT conditions.
    Compute bound on the Error in computing the minimum (e.g. by the KKT Gap)
  until Error $< \epsilon$ or $S_w = \emptyset$

have been proposed [284, 266, 398, 459, 108], most of them with slightly different selection strategies. In the following we focus on the common features among them and explain the basic ideas. For practical implementations we recommend the reader look for the most recent information since seemingly minor details in the selection strategy appear to have a significant impact on the performance. It is important to also be aware that the performance of these heuristics often depends on the dataset.

10.4.3 Selection Strategies

Recall Proposition 10.1, and in particular (10.6) and (10.8). For classification, the terms $\partial_{\xi_j} c(\max(0, 1 - y_j f(x_j)))$ and $\alpha_j(y_j f(x_j) - 1)$ give an upper bound on the deviation between the current solution and the optimal one. A similar relation applies for regression.

Contribution to KKT Gap

The central idea behind most selection strategies is to pick those patterns whose contribution to the size of the KKT Gap (10.13), (10.18) is largest. This is a reasonable strategy since, after optimization over the working set S_w, the corresponding terms of the KKT Gap will have vanished (see Problem 10.11). Unfortunately though this does not guarantee that the overall size of the KKT gap will diminish by the same amount, since the other terms KKT_i for $i \in S_f$ may increase. Still, we will have picked the set of variables for which the KKT gap size for the restricted optimization problem on S_w is largest. Note that in this context we also have to take the constraint $A\alpha = d$ into account. This means that we have to select the working set S_w such that

Lower Dimensional Subspace

$$\Omega_w := \{\alpha_w | A_w \alpha_w = d - A_f \alpha_f\} \cap \{\alpha_w | 0 \le \alpha_w \le C\} \tag{10.58}$$

is sufficiently large or, at least, does not only contain one element (see Figure 10.5 for the case of standard SV classification).

As before we focus on classification (the regression case is completely analogous), and in particular on soft margin loss functions (10.6).

KKT Selection Criterion Since $\xi = \max(\xi, 0) - \max(0, -\xi)$ we can rewrite KKT_i as

$$\mathrm{KKT}_i = (C - \alpha_i)\max(0, 1 - y_i f(x_i)) + \alpha_i \max(0, y_i f(x_i) - 1). \tag{10.59}$$

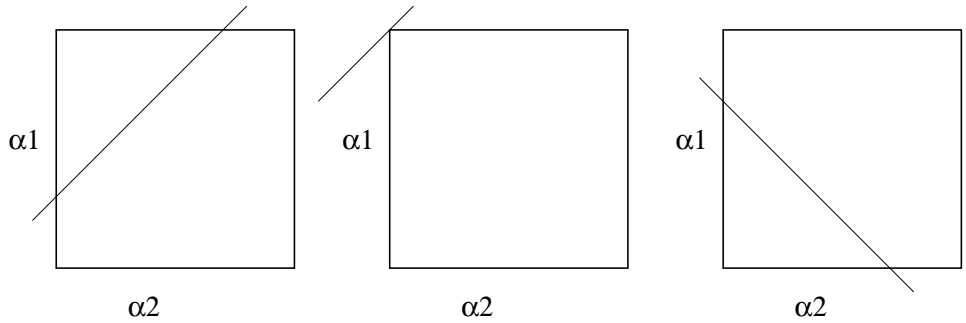

Figure 10.5 Constraints on the working set S_w in the case of classification. In all cases the box constraints $0 \leq \alpha_i \leq C$ apply. Left: two patterns of different classes result in $(+1)\alpha_1 + (-1)\alpha_2 = c$. Middle: the same case with $c = C$, thus the only feasible point is $(C, 0)$. Right: two patterns of the same class with $\alpha_1 + \alpha_2 < C$.

The size of KKT_i is used to select which variables to use for optimization purposes in the RHUL SV package [459].

Dual Constraint Satisfaction Other algorithms, such as SVMLight [266] or SVMTorch [108], see also [502], choose those patterns i in the working set for which

$$\overline{\mathrm{KKT}}_i = \mathrm{H}(C - \alpha_i)\max(0, 1 - y_i f(x_i)) + \mathrm{H}(\alpha_i)\max(0, y_i f(x_i) - 1) \tag{10.60}$$

is maximal[8], where H denotes the Heaviside function;

$$\mathrm{H}(\xi) = \begin{cases} 1 & \text{if } \xi > 0 \\ 0 & \text{otherwise} \end{cases} \tag{10.61}$$

Heaviside Function

As one can see, $C\overline{\mathrm{KKT}}_i$ is clearly an upper bound on KKT_i, and thus, if in the process of the optimization $\sum_i \overline{\mathrm{KKT}}_i$ vanishes, we can automatically ensure that the KKT gap will also decrease. The only problem with (10.60) is that it may be overly conservative in the stopping criterion and overestimate the influence of the constraint on $f(x_i)$, when the corresponding constraint on α_i is almost met.
A quite similar (albeit less popular) selection rule to (10.60) can be obtained from considering the gradient of the objective function of the optimization problem (10.43), namely $Q\alpha + c$. In this case we want to search in a direction d where $d^\top(Q\alpha + c)$ is large.

Primal Constraint Satisfaction Equations (10.59) and (10.60) suggest a third selection criterion, this time based only on the behavior of the Lagrange multipliers α_i. Rather than applying the Heaviside function to them we could also apply it to

8. This may not be immediately obvious from the derivations in [266] and [108], since their reasoning is based on the idea of [618] that we should select (feasible) directions where the gradient of the objective function is maximal. An explicit calculation of the gradient, however, reveals that both strategies lead to the same choice of a working set.

Table 10.1 Subset selection criteria for classification.

KKT gap	$(C-\alpha_i)\max(0,1-y_i f(x_i)) + \alpha_i \max(0, y_i f(x_i)-1)$
Gradient	$\mathrm{H}(C-\alpha_i)\max(0,1-y_i f(x_i)) + \mathrm{H}(\alpha_i)\max(0, y_i f(x_i)-1)$
Lagrange Multiplier	$(C-\alpha_i)\mathrm{H}(1-y_i f(x_i)) + \alpha_i \mathrm{H}(y_i f(x_i)-1)$

the factors depending on $f(x_i)$ directly. Since $\mathrm{H}(\max(0,\xi)) = \mathrm{H}(\xi)$ we obtain

$$\underline{\mathrm{KKT}}_i = (C-\alpha_i)\mathrm{H}(1-y_i f(x_i)) + \alpha_i \mathrm{H}(y_i f(x_i)-1). \tag{10.62}$$

In other words, only those patterns where the KKT conditions are violated *and* the Lagrange multipliers α_i differ from their optimal values by a large amount are chosen.

Selection Strategies

Summarizing, we may either select patterns (i) based on their contribution to the size of the KKT gap, (ii) based on the size of the gradient of the objective function at the particular location, or (iii) depending on the mismatch of the Lagrange multipliers. Table 10.1 summarizes the three strategies. Overall, strategy (i) is preferred, since it uses the tightest of all three bounds on the minimum of the objective function.

Feasible Directions In all cases we need to select a subset such that both the constraints $A\alpha = d$ and the box constraints on α are enforced. A direction e with $Ae = 0$ and $0 \leq \alpha + \delta e \leq C$ for some $\delta > 0$, that is a direction taking only points into account where the KKT conditions are violated, will surely do. In order to keep memory requirements low we will only choose a small subset $S_w \subset \{1, \ldots, m\}$ of size typically less than 100.

Balancing Samples

Balanced Sample To obtain a relatively large search space while satisfying $A\alpha = d$, we keep only coordinates d_i (and corresponding α_i) at which the gradient points away from the (possibly active) boundary constraints. Since A, in general, has a rather simple form (only entries 1 and -1) this requirement can be met by selecting a direction e where the signs of the corresponding entries in A alternate.
This can mean, for example for classification, that an equal number of positive and negative samples need be selected. For ν-SVM we additionally have to balance between points on either side of the margin within each class. The case of regression is analogous, with the only difference being that we have two "margins" rather than one.

Summing up, a simple (and very much recommended) heuristic for subset selection is to pick directions where the gradient $Q\alpha + c$ is largest, the KKT conditions for the corresponding variables are violated, and where the number of samples of either class and relation to the margin is balanced. This is what is done in [266, 108]. A proof of convergence can be found in [330]. One can show that the patterns selected by the gradient rule are identical to that chosen by (10.60) according to $\overline{\mathrm{KKT}}_i$ (see Problem 10.13).

10.5 Sequential Minimal Optimization

SMO as Special Case of Subset Selection

One algorithm, Sequential Minimal Optimization (SMO), introduced by Platt [409], puts chunking to the extreme by iteratively selecting subsets of size 2 and optimizing the target function with respect to them. It has been reported to be several orders of magnitude faster on certain data sets (up to a factor of 1000) and to exhibit better scaling properties (typically up to one order better) than classical chunking (Section 10.4.1). The key point is that for a working set of 2 the optimization subproblem can be solved analytically without explicitly invoking a quadratic optimizer[9].

SMO is one of the most easily implementable algorithms and it has a very benign memory footprint, essentially only of the size of the number of samples. In Section 8.4, we considered the special case of single-class problems; we now develop the classification and regression cases. This development includes a treatment of pattern dependent regularization and details how the algorithm can be extended to more general convex loss functions.

The exposition proceeds as follows; first we solve the generic optimization problem in two variables (Section 10.5.1) and subsequently we determine the value of the placeholders of the generic problem in the special cases of classification and regression. Finally we discuss how to adjust b properly and we determine how patterns should be selected to ensure speedy convergence (Section 10.5.5).

10.5.1 Analytic Solutions

Quadratic Program in Two Variables

We begin with a generic convex constrained optimization problem in two variables (for regression we actually have to consider four variables — $\alpha_i, \alpha_i^*, \alpha_j, \alpha_j^*$, however, only two of them may be nonzero simultaneously). By analogy to (10.43) and (10.57) we have

$$\begin{aligned} \underset{\alpha_i,\alpha_j}{\text{minimize}} \quad & \tfrac{1}{2}\left[\alpha_i^2 Q_{ii} + \alpha_j^2 Q_{jj} + 2\alpha_i\alpha_j Q_{ij}\right] + c_i\alpha_i + c_j\alpha_j \\ \text{subject to} \quad & s\alpha_i + \alpha_j = \gamma \\ & 0 \le \alpha_i \le C_i \text{ and } 0 \le \alpha_j \le C_j. \end{aligned} \tag{10.63}$$

Here $c_i, c_j, \gamma \in \mathbb{R}$, $s \in \{\pm 1\}$, and $Q \in \mathbb{R}^{2\times 2}$ are chosen suitably to take the effect of the $m-2$ variables that are kept fixed into account. The constants C_i represent

9. Note that in the following we will only consider standard SV classification and regression, since most other settings (an exception being the single-class algorithm described in Section 8.4) have more than one equality constraint and would require at least (the number of equality constraints + 1) variables per iteration in order to make any progress. In such a case (a) the difficulty of selecting a suitable set of directions would increase significantly and (b) the computational cost incurred by performing an update in a one-dimensional space would increase linearly with the number of constraints rendering SMO less attractive. See also Problem 10.14.

pattern dependent regularization parameters (as proposed in [331], among others, for unbalanced observations). Recall that we consider only optimization problems with one equality constraint. The following auxiliary lemma states the solution of (10.63).

Lemma 10.3 (Analytic Solution of Constrained Optimization) *Assume we have an optimization problem of type (10.63). Further, denote by*

$$\zeta := sc_j - c_i + \gamma s Q_{jj} - \gamma Q_{ij} \tag{10.64}$$

$$\chi := Q_{ii} + Q_{jj} - 2sQ_{ij} \tag{10.65}$$

two auxiliary variables derived from (10.63). Then, for $\chi = 0$ we have

$$\alpha_i = \begin{cases} L & \text{if } \zeta > 0 \\ H & \text{otherwise} \end{cases} \tag{10.66}$$

and for $\chi > 0$ we obtain $\alpha_i = \min(\max(L, \chi^{-1}\zeta), H)$. The case of $\chi < 0$ never occurs. Furthermore $\alpha_j = \gamma - s\alpha_i$ and L, H are defined as

$$L = \begin{cases} \max(0, s^{-1}(\gamma - C_j)) & \text{if } s > 0 \\ \max(0, s^{-1}\gamma) & \text{otherwise} \end{cases} \tag{10.67}$$

$$H = \begin{cases} \min(C_i, s^{-1}\gamma) & \text{if } s > 0 \\ \max(C_i, s^{-1}(\gamma - C_j)) & \text{otherwise} \end{cases} \tag{10.68}$$

Proof The idea is to remove α_j and the corresponding constraint from the optimization problem and to solve for α_i. We begin with constraints on α_i and the connection between α_i and α_j. Due to the equality constraint in (10.63) we have

$$\alpha_j = \gamma - s\alpha_i \tag{10.69}$$

and additionally, due to the constraints on α_j,

$$s\alpha_i = \gamma - \alpha_j \text{ and thus } \gamma \geq s\alpha_i \geq \gamma - C_j. \tag{10.70}$$

Since $C_i \geq \alpha_i \geq 0$ we may combine the two constraints into the constraint $H \geq \alpha_i \geq L$ where H and L are given by (10.67) and (10.68). Now that we have determined the constraints we look for the minimum. Elimination of $\alpha_j = \gamma - s\alpha_i$ yields

Reduction to Optimization Problem in One Variable

$$\begin{aligned} \underset{\alpha_i}{\text{minimize}} \quad & \tfrac{1}{2}\alpha_i^2(Q_{ii} + Q_{jj} - 2sQ_{ij}) + \alpha_i(c_i - sc_j + \gamma Q_{ij} - \gamma s Q_{jj}) \\ \text{subject to} \quad & L \leq \alpha_i \leq H. \end{aligned} \tag{10.71}$$

We have ignored constant terms independent of α_i since they do not influence the location of the minimum. The unconstrained objective function, which can also be written as $\frac{\chi}{2}\alpha_i^2 - \zeta\alpha_i$, has its minimum at $\chi^{-1}\zeta$. In order to ensure that the solution is also optimal for the constraint $\alpha_i \in [L, H]$ we only have to "clip" the unconstrained solution $\chi^{-1}\zeta$ to the interval, i.e. $\alpha_i = \min(\max(\chi^{-1}\zeta, L), H)$. This concludes the proof. ∎

During the optimization process it may happen, due to numerical instabilities, that the numerical value of χ is negative. In this situation we simply reset $\chi = 0$ and use (10.66). All that now remains is to find explicit values in the cases of classification and regression.

10.5.2 Classification

Proposition 10.4 (Optimal Values for Classification) *In classification the optimal values of α_i and α_j are given as follows. Denote by $\chi = K_{ii} + K_{jj} - 2y_i y_j K_{ij}$, $s = y_i y_j$, and let L, H be defined as in*

	$y_i = y_j$	$y_i \neq y_j$
α_i	$L = \max(0, \alpha_i^{\text{old}} + \alpha_j^{\text{old}} - C_j)$ $H = \min(C_i, \alpha_i^{\text{old}} + \alpha_j^{\text{old}})$	$L = \max(0, \alpha_i^{\text{old}} - \alpha_j^{\text{old}})$ $H = \min(C_i, C_j + \alpha_i^{\text{old}} - \alpha_j^{\text{old}})$

We have $\alpha_i = \min(\max(\bar{\alpha}, L, H))$ and $\alpha_j = s(\alpha_i^{\text{old}} - \alpha_i) - \alpha_j^{\text{old}}$. With the auxiliary definition $\delta := y_i((f(x_j) - y_j) - (f(x_i) - y_i))$, $\bar{\alpha}$ is given by

$$\bar{\alpha} = \begin{cases} \alpha_i^{\text{old}} + \chi^{-1}\delta & \text{if } \chi > 0 \\ -\infty & \text{if } \chi = 0 \text{ and } \delta > 0 \\ \infty & \text{if } \chi = 0 \text{ and } \delta < 0 \end{cases} \tag{10.72}$$

This means that the change in α_i and α_j depends on the difference between the approximation errors in i and j. Moreover, in the case that the unconstrained solution of the problem is identical to the constrained one ($\alpha_i = \bar{\alpha}$) the improvement in the objective function is given by $\chi^{-1}((f(x_j) - y_j) - (f(x_i) - y_i))^2$. Hence we should attempt to find pairs of patterns (i, j) where the difference in the classification errors is largest (and the constraints will still allow improvements in terms of the Lagrange multipliers).

Proof We begin with γ. In classification we have $\sum_{i=1}^m y_i \alpha_i = 0$ and thus $y_i \alpha_i + y_j \alpha_j = y_i \alpha_i^{\text{old}} + y_j \alpha_j^{\text{old}}$, or equivalently

$$y_i y_j \alpha_i + \alpha_j = y_i y_j \alpha_i^{\text{old}} + \alpha_j^{\text{old}} =: \gamma \text{ and } s = y_i y_j. \tag{10.73}$$

Now we turn our attention to Q. From (10.56) we conclude that $Q_{ii} = K_{ii}$, $Q_{jj} = K_{jj}$, and $Q_{ij} = Q_{ji} = sK_{ij}$. This leads to

$$\chi = K_{ii} + K_{jj} - 2K_{ij}. \tag{10.74}$$

Next we compute c_i and c_j. Eq. (10.57) leads to

$$c_i = -1 + y_i \left(\sum_{l \neq i,j}^m \alpha_l k(x_i, x_l) \right) = y_i(f(x_i) - b - y_i) - \alpha_i K_{ii} - \alpha_j s K_{ij} \tag{10.75}$$

and similarly for c_j. Using $y_i = y_j s$ we compute ζ as

$$\zeta = -y_i(f(x_i) - b - y_i) + \alpha_i K_{ii} + \alpha_j s K_{ij} + y_i(f(x_j) - b - y_j) \tag{10.76}$$

$$
\begin{aligned}
&+\alpha_j s K_{jj} + \alpha_i K_{ij} + (\alpha_i + s\alpha_j)(K_{ij} - K_{jj}) \\
&= y_i((f(x_j) - y_j) - (f(x_i) - y_i)) + \alpha_i \chi
\end{aligned} \tag{10.77}
$$

Substituting the values of γ, χ, and ζ into Lemma 10.3 concludes the proof. ■

10.5.3 Regression

We proceed as in classification. We have the additional difficulty, however, that for each pair of patterns x_i and x_j we have four Lagrange multipliers $\alpha_i, \alpha_i^*, \alpha_j$, and α_j^*. Hence we must possibly consider up to three different pairs of solutions[10]. Let us rewrite the restricted optimization problem in regression as follows

$$
\begin{aligned}
\underset{\alpha_i,\alpha_i^*,\alpha_j,\alpha_j^*}{\text{minimize}} \quad & \tfrac{1}{2}\begin{bmatrix} \alpha_i - \alpha_i^* \\ \alpha_j - \alpha_j^* \end{bmatrix}^\top \begin{bmatrix} K_{ii} & K_{ij} \\ K_{ij} & K_{jj} \end{bmatrix} \begin{bmatrix} \alpha_i - \alpha_i^* \\ \alpha_j - \alpha_j^* \end{bmatrix} \\
& + \begin{bmatrix} c_i \\ c_j \end{bmatrix}^\top \begin{bmatrix} \alpha_i - \alpha_i^* \\ \alpha_j - \alpha_j^* \end{bmatrix} + \varepsilon(\alpha_i + \alpha_i^* + \alpha_j + \alpha_j^*) \\
\text{subject to} \quad & 0 \le \alpha_l \le C_l \text{ and } 0 \le \alpha_l^* \le C_l^* \text{ for all } l \in \{i, j\} \\
& (\alpha_i - \alpha_i^*) + (\alpha_j - \alpha_j^*) = (\alpha_i^{\text{old}} - \alpha_i^{*\text{old}}) + (\alpha_j^{\text{old}} - \alpha_j^{*\text{old}}) =: \gamma.
\end{aligned} \tag{10.78}
$$

Here c_i, c_j are suitably chosen constants depending solely on the differences $\alpha_i - \alpha_i^*$ and $\alpha_j - \alpha_j^*$. One can check that

$$
c_i = -y_i + (f(x_i) - b - K_{ii}(\alpha_i - \alpha_i^*) - K_{ij}(\alpha_j - \alpha_j^*)) \tag{10.79}
$$

and c_j likewise. We deliberately keep the contribution due to ε separate since this is the only part where sums $\alpha_i + \alpha_i^*$ rather than differences enter the equations.

Eliminating α_j, α_j^*

As in the classification case we begin with the constraints on α_i and α_i^*. Due to the summation constraint in the optimization problem we obtain $(\alpha_i - \alpha_i^*) = \gamma - (\alpha_j - \alpha_j^*)$. Using the additional box constraints on α_i, α_i^* of (10.78) leads to

$$
L := \max(\gamma - C_j, -C_i^*) \le \alpha_i - \alpha_i^* \le \min(\gamma + C_j^*, C_i) =: H. \tag{10.80}
$$

This allows us to eliminate α_j, α_j^* and rewrite (10.78) in terms of $\beta := \alpha_i - \alpha_i^*$,

$$
\begin{aligned}
\underset{\beta}{\text{minimize}} \quad & \tfrac{1}{2}\beta^2(K_{ii} + K_{jj} - 2K_{ij}) + \beta(\gamma(K_{ij} - K_{jj}) + c_i - c_j) \\
& + \varepsilon(|\beta| + |\gamma - \beta|) \\
= \quad & \tfrac{1}{2}\beta^2\chi + \beta((f(x_i) - y_i) - (f(x_j) - y_j) - \chi(\alpha_i^{\text{old}} - \alpha_i^{*\text{old}})) \\
& + \varepsilon(|\beta| + |\gamma - \beta|) \\
\text{subject to} \quad & L \le \beta \le H.
\end{aligned} \tag{10.81}
$$

10. The number of solutions is restricted to four due to the restriction that α_i and α_i^* (or analogously α_j and α_j^*) may never both be nonzero at the same time. In addition, the constraint that $\alpha_i - \alpha_i^* + \alpha_j - \alpha_j^* = \gamma$ rules out one of these remaining four combinations.

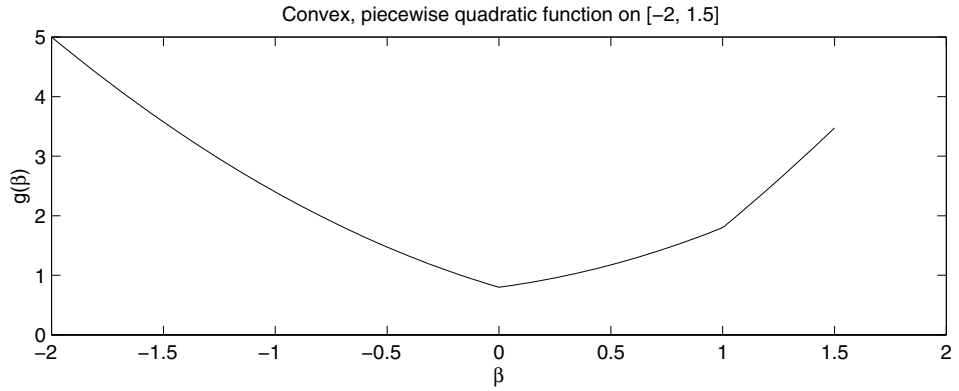

Figure 10.6 The minimum of this function occurs at $\beta = 0$ due to the change in $\varepsilon|\beta|$.

Here we used the $\chi = K_{ii} + K_{jj} - 2y_i y_j K_{ij}$, as in classification. The objective function is convex and piecewise quadratic on the three intervals

$$I_- := [L, \min(0,\gamma)],\ I_0 := [\min(0,\gamma), \max(0,\gamma)],\ I_+ := [\max(0,\gamma), H] \tag{10.82}$$

(for $\gamma = 0$ the interval I_2 vanishes). An example of such a function is given in Figure 10.6. One can check that the *unconstrained* minimum of the quadratic objective function (10.81), as defined on the intervals I_-, I_0, I_+, would be given by

Effect of Piecewise Convex Function

$$\left.\begin{matrix}\beta_- \\ \beta_0 \\ \beta_+\end{matrix}\right\} = (\alpha_i^{\text{old}} - \alpha_i^{*\text{old}}) + \frac{1}{\chi}((f(x_j) - y_j) - (f(x_i) - y_i)) + \frac{\varepsilon}{\chi}\left\{\begin{matrix} 2 & \text{if } \beta \in I_- \\ 0 & \text{if } \beta \in I_0 \\ -2 & \text{if } \beta \in I_+ \end{matrix}\right. \tag{10.83}$$

For $\chi = 0$ the same considerations as in classification apply; the optimum is found on one of the interval boundaries. Furthermore, since (10.78) is convex all we now have to do is match up the solutions β_i with the corresponding intervals I_i.

For convenience we start with β_0 and I_0. If $\beta_0 \in I_0$ we have found the optimum. Otherwise we must continue our search in the direction in which β_0 exceeds I_0. Without loss of generality assume that this is I_+. Again, if $\beta_+ \in I_+$ we may stop. Otherwise we simply "clip" β_+ to the interval boundaries of I_+. Now we have to reconstruct α from β. Due to the box constraints and the fact that $\sum_i(\alpha_i - \alpha_i^*) = 0$ we obtain

Update is Independent of b

$$\begin{matrix} \alpha_i & = & \max(0,\beta), & \alpha_j & = & \max(0,\gamma-\beta) \\ \alpha_i^* & = & \max(0,-\beta), & \alpha_j^* & = & \max(0,-\gamma+\beta). \end{matrix} \tag{10.84}$$

In order to arrive at a complete SV regression or classification algorithm, we still need a way of selecting the patterns x_i, x_j and a method specifying how to update the constant offset b efficiently. Since most pattern selection methods use b as additional information to select patterns we will start with b.

10.5.4 Computing the Offset b and Optimality Criteria

We can compute b by exploiting the KKT conditions (see Theorem 6.21). For instance in classification; at the solution, the margin must be exactly 1 for Lagrange multipliers for which the box constraints are inactive. We obtain

$$y_i f(x_i) = y_i(\langle \mathbf{w}, \Phi(x)\rangle + b) = 1 \text{ for } \alpha_i \in (0, C_i) \tag{10.85}$$

Computing b via KKT Conditions

and likewise for regression

$$f(x_i) = \langle \mathbf{w}, \Phi(x)\rangle + b = y_i - \varepsilon \text{ for } \alpha_i \in (0, C_i) \tag{10.86}$$

$$f(x_i) = \langle \mathbf{w}, \Phi(x)\rangle + b = y_i + \varepsilon \text{ for } \alpha_i^* \in (0, C_i^*). \tag{10.87}$$

Hence, if all the Lagrange multipliers α_i were optimal, we could easily find b by picking any of the unconstrained α_i or α_i^* and solving (10.85), (10.86), or (10.87).

Unfortunately, during training, not all Lagrange multipliers will be optimal, since, if they were, we would already have obtained the solution. Hence, obtaining b by the aforementioned procedure is not accurate. We resort to a technique suggested by Keerthi et al. [291, 289] in order to overcome this problem.

Sets of KKT Violation and Satisfaction

For the sake of simplicity we start with the classification setting; we first split the patterns X into the following five sets:

$$\begin{aligned} I_0 &= \{i | \alpha_i \in (0, C_i)\} \quad I_{+,0} = \{i | \alpha_i = 0, y_i = +1\} \quad I_{+,C} = \{i | \alpha_i = C_i, y_i = +1\} \\ &\qquad\qquad I_{-,0} = \{i | \alpha_i = 0, y_i = -1\} \quad I_{-,C} = \{i | \alpha_i = C_i, y_i = -1\} \end{aligned}$$

Moreover we define

$$\begin{aligned} e_{\text{hi}} &:= \min_{i \in I_0 \cup I_{+,0} \cup I_{-,C}} f(x_i) - y_i \\ e_{\text{lo}} &:= \max_{i \in I_0 \cup I_{-,0} \cup I_{+,C}} f(x_i) - y_i. \end{aligned} \tag{10.88}$$

Since the KKT conditions have to hold for a solution we can check that this corresponds to $e_{\text{hi}} \geq 0 \geq e_{\text{lo}}$. For I_0 we have already exploited this fact in (10.85). Formally we can always satisfy the conditions for e_{hi} and e_{lo} by introducing two thresholds: $b_{\text{hi}} = b - e_{\text{hi}}$ and $b_{\text{lo}} = b - e_{\text{lo}}$. Optimality in this case corresponds to $b_{\text{hi}} \leq b_{\text{lo}}$. Additionally, we may use $\frac{1}{2}(b_{\text{up}} + b_{\text{lo}})$ as an improved estimate of b.

The real benefit, however, comes from the fact that we may use e_{hi} and e_{lo} to choose patterns to focus on. The largest contribution to the discrepancy between e_{hi} and e_{lo} stems from that pair of patterns (i, j) for which

Choose Large Discrepancy with Large Possible Updates

$$\text{discrepancy}(i, j) := (f(x_i) - y_i) - (f(x_j) - y_j) \text{ where } \begin{array}{l} i \in I_0 \cup I_{-,0} \cup I_{+,C} \\ j \in I_0 \cup I_{+,0} \cup I_{-,C} \end{array} \tag{10.89}$$

is largest. This is a reasonable strategy for the following reason: from Proposition 10.4 we conclude that the potential change in the variables α_i, α_j is largest if the discrepancy $(f(x_i) - y_i) - (f(x_j) - y_j)$ is largest. The only modification is that i and j are not chosen arbitrarily any more.

Finally, we obtain another stopping criterion. Instead of requiring that the violation of the KKT condition is smaller than some tolerance Tol we may require that $e_{\text{lo}} \leq e_{\text{hi}}$ holds with some tolerance; $e_{\text{lo}} \leq e_{\text{hi}} - 2\,\text{Tol}$. In addition, we will not consider patterns where discrepancy$(i, j) < 2\,\text{Tol}$. See [290] for more details and pseudocode of their implementation.

To adapt these ideas to regression we have to modify the sets I slightly. The change is needed since we have to add or subtract ε in a way that is very similar to our treatment of the classification case, where $y_i \in \{\pm 1\}$.

1. If $\alpha_i = 0$ at optimality we must have $f(x_i) - (y_i - \varepsilon) \geq 0$.
2. For $\alpha_i \in (0, C_i)$ we must have $f(x_i) - (y_i - \varepsilon) = 0$.
3. For $\alpha_i = C_i$ we get $f(x_i) - (y_i - \varepsilon) \leq 0$.

Analogous inequalities hold for α_i^*. As before we split the patterns X into several sets according to

$$I_0 = \{i|\alpha_i \in (0, C_i)\} \quad I_{+,0} = \{i|\alpha_i = 0\} \quad I_{+,C} = \{i|\alpha_i = C_i\}$$
$$I_0^* = \{i|\alpha_i^* \in (0, C_i^*)\} \quad I_{-,0} = \{i|\alpha_i^* = 0\} \quad I_{-,C} = \{i|\alpha_i^* = C_i^*\}$$

Computing b for Regression

and introduce $e_{\text{hi}}, e_{\text{lo}}$ by

$$e_{\text{hi}} := \min\left(\min_{i \in I_0 \cup I_{+,0}} f(x_i) - (y_i - \varepsilon), \min_{i \in I_0^* \cup I_{-,C}} f(x_i) - (y_i + \varepsilon)\right) \tag{10.90}$$

$$e_{\text{lo}} := \max\left(\max_{i \in I_0 \cup I_{+,C}} f(x_i) - (y_i - \varepsilon), \max_{i \in I_0^* \cup I_{-,0}} f(x_i) - (y_i + \varepsilon)\right). \tag{10.91}$$

The equations for computing a more robust estimate of b are identical to the ones in the classification case. Note that (10.90) and (10.91) are equivalent to the ansatz in [494], the only difference being that we sacrifice a small amount of numerical efficiency for a somewhat simpler definition of the sets I (some of them are slightly larger than in [494]) and the rules regarding which e_{hi} and e_{lo} are obtained (the cases $\alpha_i = 0, \alpha_i^* = C_i^*$ and $\alpha_i^* = 0, \alpha_i = C_i$ are counted twice).

Without going into further details, we may use a definition of a discrepancy like (10.89) and then choose patterns (i, j) for optimization where this discrepancy is largest. See the original work [494] for more details. Below we give a simpler (and slightly less powerful) reasoning.

10.5.5 Selection Rules

The previous section already indicated some ways to pick the indices (i, j) such that the decrease in the objective function is maximized. We largely follow the reasoning of Platt [409, Section 12.2.2]. See also the pseudocode (Algorithms 10.3 and 10.4).

We choose a two loop approach to maximizing the objective function. The outer loop iterates over all patterns violating the KKT conditions, or possibly over those where the threshold condition of the previous section (using e_{hi} and e_{lo}) is violated.

Usually we first loop only over those with Lagrange multipliers neither on the upper nor lower boundary. Once all of these are satisfied we loop over all patterns violating the KKT conditions, to ensure self consistency on the complete dataset. This solves the problem of choosing the index i.

Full Sweep for Noisy Data

It is sometimes useful, especially when dealing with noisy data, to iterate over the complete KKT violating dataset before complete self consistency on the subset has been achieved. Otherwise considerable computational resources are spent making subsets self consistent that are not globally self consistent. The trick is to perform a full sweep through the data once only less than, say, 10% of the non bound variables change[11].

Now to select j: To make a large step towards the minimum, one looks for large steps in α_i. Since it is computationally expensive to compute χ for all possible pairs (i, j) one chooses a heuristic to maximize the change in the Lagrange multipliers α_i and thus to maximize the absolute value of the numerator in the expressions (10.72) and (10.83). This means that we are looking for patterns with large differences in their relative errors $f(x_i) - y_i$ and $f(x_j) - y_j$. The index j corresponding to the maximum absolute value is chosen for this purpose.

Second Choice Hierarchy

If this heuristic happens to fail, in other words if little progress is made by this choice, all other indices j are looked at (this is what is called "second choice hierarchy" in [409]) in the following way.

1. All indices j corresponding to non-bound examples are looked at, searching for an example to make progress on.
2. In the case that the first heuristic was unsuccessful, all other samples are analyzed until an example is found where progress can be made.
3. If both previous steps fail, SMO proceeds to the next index i.

For a more detailed discussion and further modifications of these heuristics see [409] and [494, 291].

10.6 Iterative Methods

Many training algorithms for SVMs or similar estimators can be understood as iterative methods. Their main advantage lies in the simplicity with which they can be implemented. While not all of them provide the best performance (plain gradient descent in Section 10.6.1) and some may come with restrictions on the scope of applications (Lagrangian SVM in Section 10.6.2 can be used only for quadratic soft-margin loss), the algorithms presented in this section will allow practitioners to obtain first results in a very short time. Finally, Section 10.6.3 indicates how Support Vector algorithms can be extended to online learning problems.

11. This modification is not contained in the pseudocodes, however, its implementation should not pose any further problems. See also [494, 291] for further pseudocodes.

Algorithm 10.3 Pseudocode for SMO Classification

```
function TakeStep(i, j)
  if i = j then return 0
  s = y_i y_j
  if s = 1 then
    L = max(0, α_i + α_j − C_j)
    H = min(C_i, α_i + α_j)
  else
    L = max(0, α_i − α_j)
    H = min(C_i, C_j + α_i − α_j)
  end if
  if L = H then return 0
  χ = K_ii + K_jj − 2K_ij
  if χ > 0 then
    ᾱ = α_i + χ^{-1} y_i((f(x_j) − y_j) − (f(x_i) − y_i))
    ᾱ = min(max(ᾱ, L), H)
  else if y_i((f(x_j) − y_j) − (f(x_i) − y_i)) < 0 then
    ᾱ = H
  else
    ᾱ = L
  end if
  if |α_i − ᾱ| < ε(ε + ᾱ + α_i) then return 0
  α_j += s(α_i − ᾱ) and α_i = ᾱ (note: x += y means x = x + y)
  Update b
  Update f(x_1), ..., f(x_m)
  return 1
end function

function ExamineExample(i)
  KKT_i = H(α_i) max(0, y_i f(x_i) − 1) + H(1 − α_i) max(0, 1 − y_i f(x_i))
  if KKT_i > Tol then
    if Number of nonzero and non bound α_i > 1 then
      Find j with second choice heuristic
      if TakeStep(i, j) = 1 then return 1
    end if
    for all α_j > 0 and α_j < C_j (start at random point) do
      if TakeStep(i, j) = 1 then return 1
    end for
    for all remaining α_j do
      if TakeStep(i, j) = 1 then return 1
    end for
  end if
  return 0
end function

main SMO Classification(k, X, Y, ε)
  Initialize α_i, α_i^* = 0 and b = 0, make X, Y, α global variables
  ExamineAll = 1
  while NumChanged > 0 or ExamineAll = 1 do
    NumChanged = 0
    if ExamineAll = 1 then
      for all α_i do NumChanged += ExamineExample(i)
    else
      for all α_i > 0 and α_i < C_i do NumChanged += ExamineExample(i)
    end if
    if ExamineAll = 1 then
      ExamineAll = 0
    else if NumChanged = 0 then
      ExamineAll = 1
    end if
  end while
end main
```

.

Algorithm 10.4 Pseudocode for SMO Regression

function TakeStep(i, j)
  if $i = j$ **then** return 0
  $\gamma = (\alpha_i - \alpha_i^*) + (\alpha_j - \alpha_j^*)$
  $L = \max(\gamma - C_j, -C_i^*)$ and $H = \min(\gamma + C_j^*, C_i)$
  if $L = H$ **then** return 0
  $l = \min(\gamma, 0)$ and $h = \max(\gamma, 0)$
  $\chi = K_{ii} + K_{jj} - 2K_{ij}$
  if $\chi > 0$ **then**
    $\beta_0 = (\alpha_i - \alpha_i^*) + \chi^{-1}((f(x_i) - y_i) - (f(x_j) - y_j))$
    $\beta_+ = \beta_0 - 2\frac{\varepsilon}{\chi}$ and $\beta_- = \beta_0 + 2\frac{\varepsilon}{\chi}$.
    $\beta = \max(\min(\beta_0, h), l)$ (clip β_0 to I_0)
    if $\beta = h$ **then** $\beta = \max(\min(\beta_+, H), h)$
    if $\beta = l$ **then** $\beta = \max(\min(\beta_-, l), L)$
  else if $(f(x_i) - y_i) - (f(x_j) - y_j) < 0$ **then**
    $\beta = h$
    if $(f(x_i) - y_i) - (f(x_j) - y_j) + 2\varepsilon < 0$ **then** $\beta = H$
  else
    $\beta = l$
    if $(f(x_i) - y_i) - (f(x_j) - y_j) - 2\varepsilon > 0$ **then** $\beta = L$
  end if
  if $|\beta - (\alpha_i - \alpha_i^*)| < \varepsilon(\varepsilon + |\beta| + \alpha_i + \alpha_i^*)$ **then** return 0
  $\alpha_i = \max(\beta, 0)$, $\alpha_i^* = \max(-\beta, 0)$, and $\alpha_j = \max(0, \gamma - \beta)$, $\max(0, -\gamma + \beta)$
  Update b
  Update $f(x_1), \ldots, f(x_m)$
  return 1
end function

function ExamineExample(i)
$$\overline{\mathrm{KKT}}_i = \mathrm{H}(\alpha_i)\max(0, f(x_i) - (y_i - \varepsilon)) + \mathrm{H}(\alpha_i^*)\max(0, (y_i + \varepsilon) - f(x_i)) + \mathrm{H}(C_i - \alpha_i)\max(0, (y_i - \varepsilon) - f(x_i)) + \mathrm{H}(C_i^* - \alpha_i^*)\max(0, f(x_i) - (y_i + \varepsilon))$$
  if $\overline{\mathrm{KKT}}_i > \mathrm{Tol}$ **then**
    if Number of nonzero and non bound $\alpha_i > 1$ **then**
      Find j with second choice heuristic
      if TakeStep(i, j) = 1 **then** return 1
    end if
    for all $\alpha_j > 0$ **and** $\alpha_j < C_j$ (start at random point) **do**
      if TakeStep(i, j) = 1 **then** return 1
    end for
    for all remaining α_j **do**
      if TakeStep(i, j) = 1 **then** return 1
    end for
  end if
  return 0
end function

main SMO Regression(k, X, Y, ε)
  Initialize $\alpha_i, \alpha_i^* = 0$ and $b = 0$
  ExamineAll = 1
  while NumChanged > 0 **or** ExamineAll = 1 **do**
    NumChanged = 0
    if ExamineAll = 1 **then**
      for all α_i **do** NumChanged += ExamineExample(i)
    else
      for all $\alpha_i > 0$ **and** $\alpha_i < C_i$ **do** NumChanged += ExamineExample(i)
    end if
    if ExamineAll = 1 **then**
      ExamineAll = 0
    else if NumChanged = 0 **then**
      ExamineAll = 1
    end if
  end while
end main

.

10.6.1 Gradient Descent

Most of the methods in this chapter are concerned with the *dual* optimization problem of the regularized risk functional. It is, however, perfectly legitimate to ask whether or not a primal optimization approach would also lead to good solutions. The maximum margin perceptron of Kowalczyk [309] for instance follows such an approach. Another method which can be understood as gradient descent is Boosting (see [349, 179]).

It is important to keep in mind that the choice of parametrization will have a significant impact on the performance of the algorithm (see [9] for a discussion of these issues in the context of Neural Networks). We could either choose to compute the gradient in the function space (thus the Reproducing Kernel Hilbert Space $\mathcal{H}$) of f, namely $\partial_f R_{\text{reg}}[f]$, or choose a particular parametrization $f(x) = \sum_i \alpha_i k(x_i, x)$ and compute the gradient with respect to the parameters α_i. Depending on the formulation we obtain different (and variably efficient) algorithms. We also briefly mention how the kernel AdaTron [183] fits into this context. For convenience of notation we choose the λ formulation of the regularized risk functional.

Gradient in Function Space

Let us start with gradients in function space. We use the standard RKHS regularization ([349] and, later, [221] use gradients in the space ℓ_2^m induced by the values of f on the training set) terms $\Omega[f] = \frac{1}{2}\|f\|^2_{\mathcal{H}}$. With the definitions of (4.1) this yields:

$$R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1}^{m} c(x_i, y_i, f(x_i)) + \frac{\lambda}{2}\|f\|^2_{\mathcal{H}} \tag{10.92}$$

$$\partial_f R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1}^{m} c'(x_i, y_i, f(x_i))k(x_i, \cdot) + \lambda f. \tag{10.93}$$

Consequently, we obtain the following update rules for f, given a learning rate Λ,

$$f \longleftarrow f - \Lambda\partial_f R_{\text{reg}}[f] = (1 - \Lambda\lambda)f - \Lambda\sum_{i=1}^{m} c'(x_i, y_i, f(x_i))k(x_i, \cdot). \tag{10.94}$$

Here the symbol '$\longleftarrow$' means 'is updated to'. For computational reasons we have to *represent* f as a linear combination of functions in a finite dimensional space (the Representer Theorem of Section 4.2 tells us that m basis functions $k(x_i, x)$ are sufficient for this). With the usual expansion $f(\cdot) = \sum_i \alpha_i k(x_i, \cdot)$ the update rule for the coefficients becomes

$$\alpha \longleftarrow (1 - \Lambda\lambda)\alpha - \Lambda\gamma = \alpha - \Lambda(\lambda\alpha + \gamma), \text{ where } \gamma_i = c'(x_i, y_i, f(x_i)). \tag{10.95}$$

Distinguishing between the different cases of regression, classification, and classification with a Boosting cost function [498, 221] we obtain the derivatives as described in Table 10.2.

Note that we can obtain update rules similar to the Kernel AdaTron [183] if we

Table 10.2 Cost functions and their derivatives for ε-Regression, Soft-Margin Classification, and Boosting with an exponential cost function.

$c(x, y, f(x))$	$c'(x, y, f(x))$
Regression $c = \begin{cases} f(x) - y - \varepsilon & \text{if } f(x) - y > \varepsilon \\ y - f(x) - \varepsilon & \text{if } y - f(x) > \varepsilon \\ 0 & \text{otherwise} \end{cases}$	$c' = \begin{cases} 1 & \text{if } f(x) - y > \varepsilon \\ -1 & \text{if } y - f(x) > \varepsilon \\ 0 & \text{otherwise} \end{cases}$
Classification $c = \begin{cases} 1 - yf(x) & \text{if } yf(x) < 1 \\ 0 & \text{otherwise} \end{cases}$	$c' = \begin{cases} -y & \text{if } yf(x) < 1 \\ 0 & \text{otherwise} \end{cases}$
Boosting $c = \exp(-yf(x))$	$c' = -y \exp(-yf(x))$

modify the loss function c to become

$$c(x, y, f(x)) = \begin{cases} \frac{1}{2}(1 - yf(x))^2 & \text{if } yf(x) < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{10.96}$$

AdaTron

On a per-pattern basis, this leads to update rules identical to the ones of the AdaTron. In particular, if we combine (10.96) with the online extensions of Section 10.6.3, we fully recover the update rule of the Kernel AdaTron. This means that the AdaTron uses squared soft margin loss functions as opposed to the standard soft margin loss of SVMs.[12]

Gradient in Coefficient Space

Rather than a parametrization in function space $\mathcal{H}$ we may also choose to start immediately with a parametrization in coefficient space [577, 221]. It is straightforward to see that, in the case of RKHS regularization as above (here $\|f\|^2 = \alpha^\top K\alpha$), we obtain, with the definitions of γ as in (10.94),

$$\partial_\alpha R_{\text{reg}}[f] = \frac{1}{m} K\gamma + \lambda K\alpha \tag{10.97}$$

$$\alpha \longleftarrow \alpha - \Lambda\lambda K\alpha - \Lambda K\gamma = \alpha - \Lambda K(\lambda\alpha + \gamma). \tag{10.98}$$

In other words the updates from (10.95) are multiplied by the kernel matrix K to obtain the update rules in the coefficient space. This means that we are performing gradient descent in a space with respect to the metric given by K rather than the Euclidean metric. The other difference to (10.93) is that it allows us to deal with regularization operators other than those based on the RKHS norm of f; $\Omega[f] = \sum_i |\alpha_i|$ for example, (see Section 4.9.2 and [498]). Table 10.3 gives an overview of different regularization operators and their gradients.

12. The strategy for computing b is different though. Since $\partial_b R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1}^m y_i c'(x_i, y_i, f(x_i))$ we may also update b iteratively if desired, whereas in the AdaTron we must add a constant offset to the kernel function in order to obtain an update rule.

Table 10.3 Gradients of the regularization term.

$\Omega[f]$	Regularization	Gradient wrt. α
$\frac{1}{2}\|f\|_{\mathcal{H}}^2$	standard SV regularizer	$K\alpha$
$\frac{1}{2}\|f\|_{\mathcal{H}}$	renormalized SV regularizer	$(\alpha^\top K\alpha)^{\frac{1}{2}} K\alpha$
$\sum_{i=1}^m \|\alpha_i\|$	sparsity regularizer	$(\operatorname{sgn}(\alpha_1), \ldots, \operatorname{sgn}(\alpha_m))$
$\sum_{i=1}^m \|f(x_i)\|^2$	ℓ_2 norm on data	$K^\top K\alpha$

Line Search

Since a unit step in the direction of the negative gradient of $R_{\text{reg}}[f]$ does not necessarily guarantee that $R_{\text{reg}}[f]$ will decrease, it is advantageous in many cases to perform a line-search in the direction of $\partial_\alpha R_{\text{reg}}[f]$, specifically, to seek γ such that $R_{\text{reg}}\left[f - \gamma\partial_\alpha R_{\text{reg}}[f]\right]$ is minimized. Details of how this can be achieved are in Section 6.2.1, and Algorithm 6.5. Moreover, Section 10.6.3 describes how the gradient descent approach may be adapted to online learning, that is, stochastic gradient descent.

We conclude the discussion of gradient descent algorithms by stating a lower bound on the minimum value of the regularized risk functional [221], which depends on the size of the gradient in function space.

Theorem 10.5 (Lower Bound on Primal Objective Function) *Denote by $R_{\text{emp}}[f]$ a convex and differentiable functional on a Hilbert space $\mathcal{H}$ and consider the regularized risk functional*

$$R_{\text{reg}}[f] = R_{\text{emp}}[f] + \lambda\|f\|_{\mathcal{H}}^2, \text{ where } \lambda > 0 \tag{10.99}$$

Then, for any $f, \Delta f \in \mathcal{H}$

$$R_{\text{reg}}[f] - R_{\text{reg}}[f - \Delta f] \leq \frac{1}{2}\|\nabla R_{\text{reg}}[f]\|^2. \tag{10.100}$$

Proof We assume that $f - \Delta f$ is the minimizer of $R_{\text{reg}}[f]$, since proving the inequality for the minimizer is sufficient. Since R_{emp} is convex and differentiable we know that

$$R_{\text{emp}}[f] - R_{\text{emp}}[f - \Delta f] \leq \langle \Delta f, \nabla R_{\text{emp}}[f]\rangle. \tag{10.101}$$

Therefore we may bound $\rho(f, \Delta f) := R_{\text{reg}}[f] - R_{\text{reg}}[f - \Delta f]$ by

$$\rho(f, \Delta f) \leq \langle \Delta f, \nabla R_{\text{emp}}[f]\rangle + \lambda\omega(\|f\|) - \lambda\omega(\|f - \Delta f\|). \tag{10.102}$$

It is easy to check that if $\omega(\|f\|) = \frac{1}{2}\|f\|^2$, (10.102) is minimized by

$$\lambda\Delta f = \nabla R_{\text{emp}}[f] + \lambda f = \nabla R_{\text{reg}}[f]. \tag{10.103}$$

Substituting this back into $\rho(f, \Delta f)$ proves (10.100). ■

Eq. (10.100) shows that a stopping criterion based on the size of the gradient is a feasible strategy when minimizing regularized risk functionals.

Gradient descent algorithms are relatively simple to implement but we should

keep in mind that they often do not enjoy the convergence guarantees of more sophisticated algorithms. They are useful tools for a first implementation if no other optimization code is available, however.

10.6.2 Lagrangian Support Vector Machines

Mangasarian and Musicant [348] present a particularly fast and simple algorithm which deals with classification problems involving squared slacks (this is the same problem that the AdaTron algorithm also attempts to minimize). Below we show a version thereof extended to the nonlinear case. We begin with the basic definitions

$$c(x, y, f(x)) = \begin{cases} 0 & \text{if } yf(x) \geq 1 \\ (1 - yf(x))^2 & \text{otherwise.} \end{cases} \tag{10.104}$$

The second modification needed for the algorithm is that we also regularize the constant offset b in the function expansion, i.e. $\Omega[f] = \|\mathbf{w}\|^2 + b^2$ where $f(x) = \langle \mathbf{w}, \phi(x) \rangle + b$. This reduces the number of constraints in the optimization problem at the expense of losing translation invariance in feature space. It is still an open question whether this modification is detrimental to generalization performance. In short, we have the following optimization problem;

Primal Optimization Problem

$$\begin{array}{ll} \underset{\mathbf{w}, b, \xi}{\text{minimize}} & \frac{1}{m} \sum_{i=1}^{m} \xi_i^2 + \frac{\lambda}{2} \left(\|\mathbf{w}\|^2 + b^2 \right) \\ \text{subject to} & y_i(\langle \mathbf{w}, \phi(x_i) \rangle + b) \geq 1 - \xi_i \text{ where } \xi_i \geq 0. \end{array} \tag{10.105}$$

By using the tools from Chapter 6 (see also [345, 348]) one can show that the dual optimization problem of (10.105) is given by

Dual Optimization Problem

$$\begin{array}{ll} \underset{\alpha}{\text{minimize}} & \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (K_{ij} + 1 + \lambda m \delta_{ij}) - \sum_{i=1}^{m} \alpha_i \\ \text{subject to} & \alpha_i \geq 0 \text{ for all } i \in [m] \end{array} \tag{10.106}$$

where $\mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i \Phi(x_i)$, $b = \sum_{i=1}^{m} \alpha_i$, and $\xi_i = \lambda m \alpha_i$.

In the following we develop a recursion relation to determine a solution of (10.105). For convenience we use a slightly more compact representation of the quadratic matrix in (10.106). We define

$$Q := \operatorname{diag}(y)(K + \lambda m \mathbf{1} + \vec{1}^\top \vec{1}) \operatorname{diag}(y) \tag{10.107}$$

where $\operatorname{diag}(y)$ denotes the matrix with diagonal entries y_i and $\mathbf{1}$ is the unit matrix. Since α_i are Lagrange multipliers, it follows from the KKT conditions (see Theorem 6.21) that only if the constraints $y_i(\langle \mathbf{w}, \phi(x_i) \rangle + b) \geq 1 - \xi_i$ of (10.105) are active may the Lagrange multipliers α_i be nonzero. With the definition of Q we can write these conditions as $\alpha_i > 0$ only if $(Q\alpha)_i = 1$. Summing over all indices i we have

$$\alpha^\top (Q\alpha - \vec{1}) = 0. \tag{10.108}$$

Now, if we can find some α which are both feasible (they satisfy the constraints imposed on them) and which also satisfy $\alpha^\top(Q\alpha - \vec{1}) = 0$, then we have found a solution. The key optimization algorithm trick lies in the following lemma [348].

Lemma 10.6 (Orthogonality and Clipping) *Denote by $a, b \in \mathbb{R}^m$ two arbitrary vectors. Then the following two conditions are equivalent*

$$\left\{a, b \geq 0 \text{ and } a^\top b = 0\right\} \Longleftrightarrow \{a = (a - \gamma b)_+ \text{ for all } \gamma > 0\} \tag{10.109}$$

See Problem 10.15 for a proof.

Rewriting the KKT Conditions

Consequently it is a condition on α that, for all $\gamma > 0$,

$$Q\alpha - \vec{1} = ((Q\alpha - \vec{1}) - \gamma\alpha)_+ \tag{10.110}$$

must hold. As previously mentioned [348] a solution α satisfying (10.108) is the minimizer of the constrained optimization problem (10.106). Furthermore, Lemma 10.6 implies that (10.108) is equivalent to (10.110) for all $\gamma > 0$. This suggests an iteration scheme for obtaining α whereby

$$\alpha^{i+1} = Q^{-1}(((Q\alpha^i - \vec{1}) - \gamma\alpha^i)_+ + \vec{1}). \tag{10.111}$$

The theorem below shows that (10.111) is indeed a convergent algorithm and that it converges linearly.

Theorem 10.7 (Global Convergence of Lagrangian SVM [348]) *For any symmetric positive matrix K and Q given by (10.107) under the condition that $0 < \gamma < 2\lambda m$, the iteration scheme (10.111) will converge at a linear rate to the solution $\bar{\alpha}$ and*

$$||Q\alpha^{i+1} - Q\bar{\alpha}|| \leq ||\mathbf{1} - \gamma Q^{-1}|| \cdot ||Q\alpha^i - Q\bar{\alpha}||. \tag{10.112}$$

Proof By construction $\bar{\alpha}$ is a fixed point of (10.111). Therefore we have

$$\begin{aligned} ||Q\alpha^{i+1} - Q\bar{\alpha}|| &= ||(Q\alpha^i - \vec{1} - \gamma\alpha^i)_+ - (Q\bar{\alpha} - \vec{1} - \gamma\bar{\alpha})_+|| && (10.113)\\ &\leq ||(Q - \gamma\mathbf{1})(\alpha^i - \bar{\alpha})|| && (10.114)\\ &\leq ||\mathbf{1} - \gamma Q^{-1}|| \cdot ||Q\alpha^i - Q\bar{\alpha}||. && (10.115) \end{aligned}$$

Next we bound the norm of $||\mathbf{1} - \gamma Q^{-1}||$ and, in particular, we show under which conditions it is less than 1. By construction we know that the smallest eigenvalue of Q is at least λm and, moreover, Q is a positive matrix. Hence Q^{-1} is also positive and its largest eigenvalue is bounded from above by $\frac{1}{\lambda m}$. Therefore the largest eigenvalue of $||\mathbf{1} - \gamma Q^{-1}||$ is bounded from above by $|1 - \gamma\frac{1}{\lambda m}|$ and, consequently, for all $0 < \gamma < 2\lambda m$ the algorithm will converge. ∎

Sherman Morrison Woodbury

To make practical use of (10.111) on large amounts of data we need to find a way to invert Q cheaply. Recall Section 10.3.4 where we dealt with a similar problem in the context of interior point optimization codes. Assuming that we can find a low rank approximation of K, by $\tilde{K} = K^{mn}(K^{nn})^{-1}(K^{mn})^\top$ for example, we may

replace K by $\tilde{K}$ throughout the algorithm, apply the Sherman-Woodbury-Morrison formula (10.54) and invert Q approximately.

The additional benefit is that we get a compact representation of the solution of the classification problem in a small number of basis functions, n. Thus the evaluation of the solution is much faster than if the full matrix K had been used. The approximation in this setting ignores the smallest eigenvalues of K, which will be dominated by the addition of the regularization term $\lambda m\mathbf{1}$ in the definition (10.110) of Q anyway. In analogy to (10.55) we obtain

$$\left(\tilde{K} + \lambda m 1 + \vec{1}^\top \vec{1}\right)^{-1} \tag{10.116}$$

$$= \left(K^{mn}(K^{nn})^{-1}(K^{mn})^\top + \lambda m 1 + \vec{1}^\top \vec{1}\right)^{-1} \tag{10.117}$$

$$= (\lambda m)^{-1}\mathbf{1} - (\lambda m)^{-2} \begin{bmatrix} K^{mn} & \vec{1} \end{bmatrix} Q_{\text{red}}^{-1} \begin{bmatrix} K^{mn} & \vec{1} \end{bmatrix}^\top \tag{10.118}$$

Speedup for Low Rank Approximations

where

$$Q_{\text{red}} = \left(\begin{bmatrix} K^{nn} & 0 \\ 0 & 1 \end{bmatrix} + \lambda m \begin{bmatrix} K^{mn} & \vec{1} \end{bmatrix}^\top \begin{bmatrix} K^{mn} & \vec{1} \end{bmatrix} \right). \tag{10.119}$$

Likewise, the matrix multiplications by Q can be sped up by the low rank decomposition of K. Overall, the cost of one update step is $O(n^2 m)$; significantly less than $O(m^3)$, which would be incurred if we had to invert Q exactly. The same methods that can be used to implement any interior point method (out-of-core storage of the matrix K^{mn} for example) can also be applied to Lagrangian SVM.

Linear Kernels

For the special case that we have only linear kernels $k(x, x') = \langle x, x' \rangle$, the update rule becomes particularly simple. Here we can represent K as $K = X^\top X$ where X denotes the matrix of all patterns x_i. The MATLAB code (courtesy of Mangasarian and Musicant) is given in Algorithm 10.5 (in the nonlinear case, we can adapt the algorithm easily by replacing X with $K^{mn}(K^{nn})^{-\frac{1}{2}}$, where K^{mn} and K^{nn} are defined as in Section 10.2.1).

10.6.3 Online Extensions

Online learning differs from the settings in the other chapters of this book, which study *batch* learning, insofar as it assumes that we have a (possibly infinite) stream of incoming data $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The goal is to predict y_i and incur as little loss as possible during the iterations. This goal is quite different from that of minimizing the expected risk since the distribution from which the data (x_i, y_i) is drawn may change over time and thus no single estimate $f : \mathcal{X} \to \mathcal{Y}$ may be optimal over the total time (see [32, 54, 378]).

Increasing Number of Kernels

At every step t we could attempt to perform an optimal prediction based on the minimizer of the regularized risk functional $R_{\text{reg}}[f]$ where our training set consists of $(x_1, y_1), \ldots (x_{t-1}, y_{t-1})$. Unfortunately this task is completely computationally infeasible since it would require that we solve an ever-increasing optimization problem in t variables at every instance. Hence the time required to perform the

Algorithm 10.5 Linear Lagrangian Support Vector Machines

```
function [it, opt, w, gamma] = lsvm(X,Y,lambdam,itmax,tol)

[m,n]=size(X);
gamma=1.9 * nu;
e=ones(m,1);
H=Y*[X -e];
it=0;
S=H*inv((speye(n+1)*lambdam+H'*H));
alpha=(1-S*(H'*e)) / lambdam;
oldalpha=alpha+1;
while it<itmax & norm(oldalpha-alpha)>tol
       z=(1+pl(((speye(m)*lambdam*alpha+H*(H'*alpha))-gamma*alpha)-1));
       oldalpha=alpha;
       alpha=(z-S*(H'*z))/lambdam;
       it=it+1;
end;
opt=norm(alpha-oldalpha);w=X'*Y*alpha;b=-e'*Y*alpha;

function pl = pl(x); pl = (abs(x)+x)/2;
```

prediction would increase polynomially over time due to the increasing sample size. This is clearly not desirable.

Another problem arises from the Representer Theorem (Th. 4.2). It states that the solution is a linear combination of kernel functions $k(x_i, \cdot)$ centered at the training points. Assuming that the probability of whether a point will become a kernel function does not depend on t this shows that, at best, the selection of basis functions will change while, typically, the number of basis functions selected will grow without bound (see Problem 10.18). This means that prediction will also become increasingly expensive and the computational effort is likely to grow polynomially.

From these two problems we conclude that if we want to use an online setting we should perform some sort of approximation rather than trying to solve the learning problem exactly.

Fixed Dimensional Setting

One possibility is to project every new basis function $k(x_t, \cdot)$ onto a set of existing basis functions, say $k(x_{n_1}, \cdot), \ldots k(x_{n_N}, \cdot)$ and find a solution in the so-chosen subspace. This is very similar to online learning with a neural network with fixed architecture. We thus perform learning with respect to the functional

$$R_{\text{reg}}[f] := \frac{1}{m} \sum_{i=1}^{m} c\left(x_i, y_i, \sum_{j=1}^{N} \alpha_j k(x_{n_j}, x_i)\right) + \frac{\lambda}{2} \sum_{j,j'=1}^{N} k(x_{n_j}, x_{n'_j}) \alpha_j \alpha_{j'}, \tag{10.120}$$

Computational Cost

where $f = \sum_{i=1}^{N} \alpha_j k(x_{n_j}, \cdot)$. Unfortunately the computational cost is at least $O(N^2)$ per iteration since computing the gradient of (10.120) with respect to α already requires a matrix-vector multiplication, no matter how simple we manage to keep

the sample dependent term $\frac{1}{m}\sum_{i=1}^{m} c\left(x_i, y_i, f(x_i)\right)$[13]. This shows that any gradient descent algorithm in a lower dimensional fixed space will exhibit this problem. Hence, projection algorithms do not appear to be a promising strategy.

Likewise, incremental update algorithms [93] claim to overcome this problem but cannot guarantee a bound on the number of operations required per iteration. Hence, we must resort to different methods.

Recently proposed algorithms [194, 242, 214, 329] perform perceptron-like updates for classification at each step. Some algorithms work only in the noise free case, others do not work for moving targets, and still others assume an upper bound on the complexity of the estimators. Below we present a simple method which allows the use of kernel estimators for classification, regression, and novelty detection and which copes with a large number of kernel functions efficiently.

Direct Online Algorithm

Stochastic Approximation The following method [299] addresses the problem by formulating it in the Reproducing Kernel Hilbert Space $\mathcal{H}$ directly and then by carrying out approximations during the update process. We will minimize the ordinary regularized risk functional (4.2); $R_{\text{reg}}[f] = R_{\text{emp}}[f] + \frac{\lambda}{2}\|f\|^2_{\mathcal{H}}$. Since we want to perform *stochastic* gradient descent, the empirical error term $R_{\text{emp}}[f]$ is replaced by the empirical error *estimate at instance* (x_t, y_t), namely $c(x_t, y_t, f(x_t))$. This means that at time t we have to compute the gradient of

$$R_{\text{stoch}}[f, t] := c(x_t, y_t, f(x_t)) + \frac{\lambda}{2}\|f\|^2_{\mathcal{H}} \tag{10.121}$$

and then perform gradient descent with respect to $R_{\text{stoch}}[f, t]$. Here t is either randomly chosen from $\{1, \ldots m\}$ or it is the new training instance observed at time t. Consequently the gradient of $R_{\text{stoch}}[f, t]$ with respect to f is

$$\partial_f R_{\text{stoch}}[f, t] = c'(x_t, y_t, f(x_t))k(x_t, \cdot) + \lambda f. \tag{10.122}$$

The update equations are thus straightforward,

$$f \longleftarrow f - \Lambda \partial_f R_{\text{stoch}}[f, t], \tag{10.123}$$

where $\Lambda \in \mathbb{R}^+$ is the learning rate controlling the size of updates undertaken at each iteration. We will return to the issue of adjusting (λ, Λ) at a later stage.

Descent Algorithm Substituting the definition of $R_{\text{stoch}}[f, t]$ into (10.123) we obtain

$$f \longleftarrow f - \Lambda\left(c'(x_t, y_t, f(x_t))k(x_t, \cdot) + \lambda f\right) \tag{10.124}$$

$$= (1 - \lambda\Lambda)f - \Lambda c'(x_t, y_t, f(x_t))k(x_t, \cdot). \tag{10.125}$$

While (10.124) is convenient in a theoretical analysis, it is not directly amenable to

13. If we decide to use the gradient in function space instead then the gradient itself will be cheap to compute, but projection of the gradient onto the N dimensional subspace will cost us $O(N^2)$ operations.

computation. For this purpose we have to express f as a kernel expansion

$$f(x) = \sum_i \alpha_i k(x_i, x) \tag{10.126}$$

where the x_i are (previously seen) training patterns. Then (10.126) becomes

$$\alpha_t \longleftarrow (1 - \lambda\Lambda)\alpha_t - \Lambda c'(x_t, y_t, f(x_t)) \tag{10.127}$$

$$= -\Lambda c'(x_t, y_t, f(x_t)) \qquad \text{for } \alpha_t = 0 \tag{10.128}$$

$$\alpha_i \longleftarrow (1 - \lambda\Lambda)\alpha_i \qquad \text{for } i \neq t. \tag{10.129}$$

Eq. (10.127) means that, at each iteration, the kernel expansion may grow by one term. Further, the cost of training at each step is not larger than the prediction cost. Once we have computed $f(x_t)$, α_t is obtained by the value of the derivative of c at $(x_t, y_t, f(x_t))$.

Instead of updating all coefficients α_i we may simply cache the power series $1, (1 - \lambda\Lambda), (1 - \lambda\Lambda)^2, (1 - \lambda\Lambda)^3, \ldots$ and pick suitable terms as needed. This is particularly useful if the derivatives of the loss function c only assume discrete values, say $\{-1, 0, 1\}$ as is the case when using the soft-margin type loss functions.

Truncation The problem with (10.127) and (10.129) is that, without any further measures, the number of basis functions n will grow without bound. This is not desirable since n determines the amount of computation needed for prediction. The regularization term helps us here. At each iteration the coefficients α_i with $i \neq t$ are shrunk by $(1 - \lambda\Lambda)$. Thus, after τ iterations, the coefficient α_i will be reduced to $(1 - \lambda\Lambda)^\tau \alpha_i$.

Proposition 10.8 (Truncation Error) *For a loss function $c(x, y, f(x))$ with its first derivative bounded by C and a kernel k with bounded norm $\|k(x, \cdot)\| \leq X$, the truncation error in f incurred by dropping terms α_i from the kernel expansion of f after τ update steps is bounded by $\Lambda(1 - \lambda\Lambda)^\tau CX$. In addition, the total truncation error due to dropping all terms which are at least τ steps old is bounded by*

$$\|f - f_{\text{trunc}}\|_{\mathcal{H}} \leq \sum_{i=1}^{t-\tau} \Lambda(1 - \lambda\Lambda)^{t-i} CX < \lambda^{-1}(1 - \lambda\Lambda)^\tau CX \tag{10.130}$$

Here $f_{\text{trunc}} = \sum_{i=t-\tau+1}^{t} \alpha_i k(x_i, \cdot)$. Obviously the approximation quality increases exponentially with the number of terms retained.

The regularization parameter λ can thus be used to control the storage requirements for the expansion. Moreover, it naturally allows for distributions $\mathrm{P}(x, y)$ that change over time in which case it is desirable to *forget* instances (x_i, y_i) that are much older than the average time scale of the distribution change [298].

We now proceed to applications of (10.127) and (10.129) in specific learning situations. We utilize the standard addition of the constant offset b to the function expansion, $g(x) = f(x) + b$ where $f \in \mathcal{H}$ and $b \in \mathbb{R}$. Hence we also update b into $b - \Lambda\partial_b R_{\text{stoch}}[g]$.

Classification We begin with the soft margin loss (3.3), given by $c(x, y, g(x)) =$

$\max(0, 1 - yg(x))$. In this situation the update equations become

$$(\alpha_i, \alpha_t, b) \longleftarrow \begin{cases} ((1-\Lambda\lambda)\alpha_i, y_i\Lambda, b + \Lambda y_i) & \text{if } yg(x_t) < 1 \\ ((1-\Lambda\lambda)\alpha_i, 0, b) & \text{otherwise.} \end{cases} \tag{10.131}$$

For **classification with the ν-trick**, as defined in (7.40), we also have to take care of the margin ρ, since there $c(x, y, g(x)) = \max(0, \rho - yg(x)) - \nu\rho$. On the other hand, one can show [481] (see also Problem 7.16) that the specific choice of λ has no influence on the estimate in ν-SV classification. Therefore, we may set $\lambda = 1$ and obtain

$$(\alpha_i, \alpha_t, b, \rho) \longleftarrow \begin{cases} ((1-\Lambda)\alpha_i, y_i\Lambda, b + \Lambda y_i, \rho + \Lambda(1-\nu)) & \text{if } yg(x_t) < \rho \\ ((1-\Lambda)\alpha_i, 0, b, \rho - \Lambda\nu) & \text{otherwise.} \end{cases} \tag{10.132}$$

By analogy to Propositions 8.3 and 7.5, only a fraction of ν points will be used for updates. Finally, if we choose the **hinge-loss**, $c(x, y, g(x)) = \max(0, -yg(x))$,

$$(\alpha_i, \alpha_t, b) \longleftarrow \begin{cases} ((1-\Lambda\lambda)\alpha_i, y_i\Lambda, b + \Lambda y_i) & \text{if } yg(x_t) < 0 \\ ((1-\Lambda\lambda)\alpha_i, 0, b) & \text{otherwise.} \end{cases} \tag{10.133}$$

Setting $\lambda = 0$ recovers the kernel-perceptron algorithm. For nonzero λ we obtain the kernel-perceptron with regularization.

Novelty Detection Results for novelty detection (see Chapter 8 and [475]) are similar in spirit. The **ν-setting** is most useful here, particularly where the estimator acts as a warning device (network intrusion detection for example) or when we would like to specify an upper limit on the frequency of alerts $f(x) < \rho$. The relevant loss function, as introduced in (8.6), is $c(x, y, f(x)) = \max(0, \rho - f(x)) - \nu\rho$ and usually [475] one uses $f \in \mathcal{H}$ rather than $f + b$, where $b \in \mathbb{R}$, in order to avoid trivial solutions. The update equations are

$$(\alpha_i, \alpha_t, \rho) \longleftarrow \begin{cases} ((1-\Lambda)\alpha_i, \Lambda, \rho + \Lambda(1-\nu)) & \text{if } f(x) < \rho \\ ((1-\Lambda)\alpha_i, 0, \rho - \Lambda\nu) & \text{otherwise.} \end{cases} \tag{10.134}$$

Considering the update of ρ we can see that, on average, only a fraction of ν observations will be considered for updates. Thus we only have to store a small fraction of the x_i. We can see that the learning rate Λ provides us with a handle to trade off the cost of the expansion (in terms of the number of basis functions needed) with the time horizon of the prediction; the smaller Λ, the more patterns are included since the coefficients α_i will decay only very slowly. Beyond this point, further research needs to be done to show how Λ is best adjusted (a rule of thumb is to let it decay as $\frac{1}{\sqrt{m}}$). Figure 10.7 contains initial results of the online novelty detection algorithm.

Adjusting Λ

Algorithm 10.6 describes the learning procedure for novelty detection in detail.

Regression We consider the following four settings: squared loss, the ε-insensitive loss using the ν-trick, Huber's robust loss function, and trimmed mean estimators. For convenience we only use estimates $f \in \mathcal{H}$ rather than $g = f + b$ where $b \in \mathbb{R}$. The extension to the latter case is straightforward.

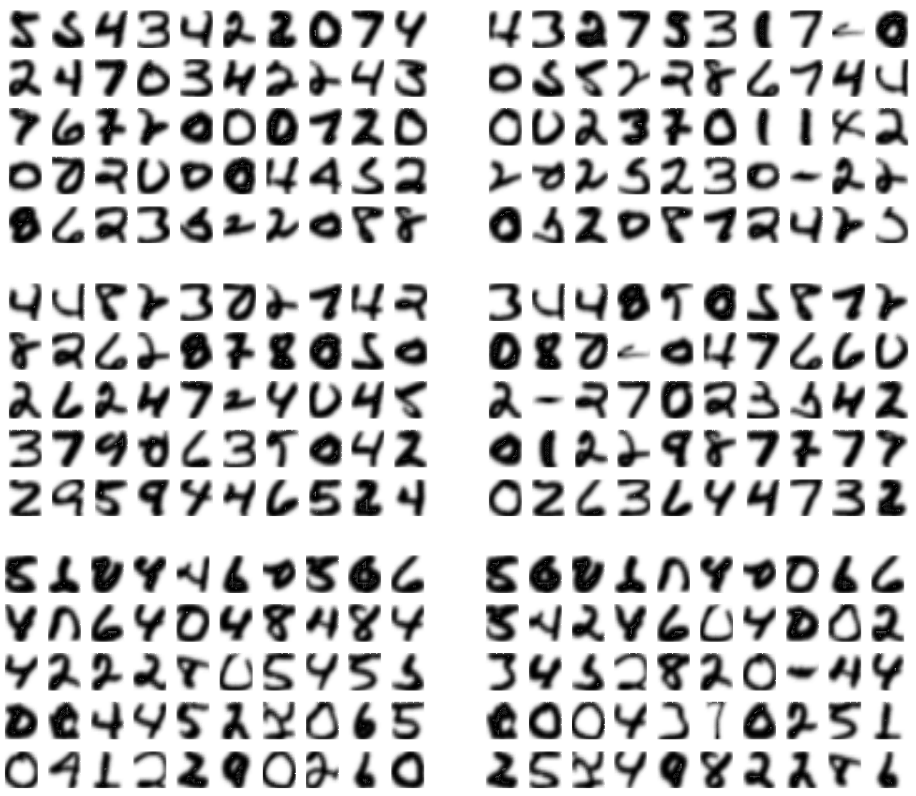

Figure 10.7 Online novelty detection on the USPS dataset (dimension $N = 256$). We use Gaussian RBF kernels with width $2\sigma^2 = 0.5N = 128$. The learning rate was adjusted to $\frac{1}{\sqrt{m}}$ where m is the number of iterations. The left column contains results after one pass through the database, the right column results after 10 random passes. From top to bottom: (top) the first 50 patterns which incurred a margin error, (middle) the 50 worst patterns according to $f(x) - \rho$ on the training set, (bottom) the 50 worst patterns on an unseen test set.

- We begin with **squared loss** (3.8) where c is given by $c(x, y, f(x)) = \frac{1}{2}(y - f(x))^2$. Consequently the update equation is

$$(\alpha_i, \alpha_t) \longleftarrow ((1 - \lambda\Lambda)\alpha_i, \Lambda(y_t - f(x_t))). \tag{10.135}$$

This means that we have to store *every* observation we make or, more precisely, the prediction error we make on every observation.

- The **ε-insensitive loss** (see (3.9)) $c(x, y, f(x)) = \max(0, |y - f(x)| - \varepsilon)$ avoids this problem but introduces a new parameter — the width of the insensitivity zone ε. By making ε a variable of the optimization problem, as shown in Section 9.3, we have

$$c(x, y, f(x)) = \max(0, |y - f(x)| - \varepsilon) + \nu\varepsilon. \tag{10.136}$$

Algorithm 10.6 Online SV Learning

input kernel k, input stream of data X, Λ, ν. time horizon T
Initialize "time" $t = 0$,
repeat
$t = t + 1$
Draw new pattern x_t and compute $f(x_t)$

$$\begin{array}{lrcl} & \alpha_i & \longleftarrow & (1-\Lambda)\alpha_i \\ \text{Update} & \alpha_t & = & \Lambda \mathrm{H}(\rho - f(x)) \\ & \rho & \longleftarrow & \rho + \Lambda(\nu - \mathrm{H}(\rho - f(x))) \end{array}$$

Truncate the expansion to T terms.
until no more data arrives

The update equations now have to be stated in terms of α_i, α_t, and ε, which is allowed to change during the optimization process. This leads to

$$(\alpha_i, \alpha_t, \varepsilon) \longleftarrow \begin{cases} ((1-\lambda\Lambda)\alpha_i, \Lambda \operatorname{sgn}(y_t - f(x_t)), \varepsilon + (1-\nu)\Lambda) & \text{if } |y_t - f(x_t)| > \varepsilon \\ ((1-\lambda\Lambda)\alpha_i, 0, \varepsilon - \Lambda\nu) & \text{otherwise.} \end{cases} \tag{10.137}$$

Meaning that every time the prediction error exceeds ε we increase the insensitivity zone by $\Lambda\nu$. Likewise, if it is smaller than ε, the insensitive zone is decreased by $\Lambda(1-\nu)$.

■ Finally, we analyze the case of regression with **Huber's robust loss**. The loss (see Table 3.1) is given by

$$c(x, y, f(x)) = \begin{cases} |y - f(x)| - \frac{1}{2}\sigma & \text{if } |y - f(x)| \geq \sigma \\ \frac{1}{2\sigma}(y - f(x))^2 & \text{otherwise.} \end{cases} \tag{10.138}$$

As before, we obtain update equations by computing the derivative of c with respect to $f(x)$.

$$(\alpha_i, \alpha_t) \longleftarrow \begin{cases} ((1-\Lambda)\alpha_i, \Lambda \operatorname{sgn}(y_t - f(x_t))) & \text{if } |y_t - f(x_t)| > \sigma \\ ((1-\Lambda)\alpha_i, \sigma^{-1}(y_t - f(x_t))) & \text{otherwise.} \end{cases} \tag{10.139}$$

■ Comparing (10.139) and (10.137) leads to the question of whether σ might not also be **adjusted adaptively**. This is a desirable goal since we may not know the amount of noise present in the data. While the ν-setting allows us to form such adaptive estimators for batch learning with the ε-insensitive loss, this goal has proven elusive for other estimators in the standard batch setting. In the online situation, however, such an extension is quite natural (see also [180]). All we need do is make σ a variable of the optimization problem and set

$$(\alpha_i, \alpha_t, \sigma) \longleftarrow \begin{cases} ((1-\Lambda)\alpha_i, \Lambda \operatorname{sgn}(y_t - f(x_t)), \sigma + \Lambda(1-\nu)) & \text{if } |y_t - f(x_t)| > \sigma \\ ((1-\Lambda)\alpha_i, \sigma^{-1}(y_t - f(x_t)), \sigma - \Lambda\nu) & \text{otherwise.} \end{cases} \tag{10.140}$$

The theoretical analysis of such online algorithms is still an area of ongoing research and we expect significant new results within the next couple of years. For first results see [175, 242, 299, 298, 194, 329]. For instance, one may show [298] that the estimate obtained by an online algorithm converges to the minimizer of the batch setting. Likewise, [242] gives performance guarantees under the assumption of bounded RKHS norms.

For practitioners, however, currently online algorithms offer an alternative to (sometimes rather tricky) batch settings and extend the domain of application available to kernel machines. It will be interesting to see whether the integration of Bayesian techniques [546, 128] leads to other novel online methods.

10.7 Summary

10.7.1 Topics We Did Not Cover

While it is impossible to cover all algorithms currently used for Kernel Machines we give an (incomplete) list of some other important methods.

Kernel Billiard This algorithm was initially proposed in [450] and subsequently modified to accommodate kernel functions in [451]. It works by simulating an ergodic dynamical system of a billiard ball bouncing off the boundaries of the version space (the version space is the set of all $\mathbf{w}$ for which the training data is classified correctly). The estimate is then obtained by averaging over the trajectories.

Bayes Point Machine The algorithm [239, 453] is somewhat similar in spirit to the Kernel Billiard. The main idea is that, by sampling from the posterior distribution of possible estimates (see Chapter 16 for a definition of these quantities), we obtain a solution close to the mean of the posterior.

Iterative Re-weighted Least Squares SVM The main idea is to use clever working set selection strategies to identify the subset of SVs that are likely to sit exactly on the margin. For those SVs, the separation inequality constraints become equalities, and, for these, the reduced QP for the working set can be solved via a linear system (by quadratically penalizing deviations from the exact equality constraints) [407]. This approach can handle additional equality constraints without significant extra effort. Accordingly, it has been generalized to situations with further equality constraints, such as the ν-SVM [408].

Maximum Margin Perceptron This algorithm works in primal space and relies on the idea that the weight vector $\mathbf{w}$ is a linear combination between vectors contained in the convex hull of patterns with $y_i = 1$ and the convex hull of patterns with $y_i = -1$. That convergence requires a finite number of steps can be proven. Moreover, the constant threshold b can be determined rather elegantly. See [309] for a variety of versions of the MMP algorithm.

AdaTron Originally introduced in [12], the kernel version of the AdaTron appeared in [183]. It is, essentially, an extension of the perceptron algorithm to the maximum margin case. As we saw in Section 10.6, similar update rules can be derived with a quadratic soft-margin loss function.

More Mathematical Programming Methods Once one is willing to go beyond the standard setting of regularization in a Reproducing Kernel Hilbert Space, one is offered a host of further Support-Vector like methods derived from optimization theory. The papers of Mangasarian and coworkers [324, 325, 188, 189] present such techniques.

10.7.2 Topics We Covered

Several algorithms can be used to solve the quadratic programming problem arising in SV regression. Most of them can be shown to share a common strategy that can be understood well through duality theory. In particular, this viewpoint leads to useful optimization and stopping criteria for many different classes of algorithm, since the Lagrange multipliers α_i are less interesting quantities than the value of the objective function itself.

Interior Point Codes A class of algorithms to exploit these properties explicitly are interior point primal-dual path following algorithms (see Section 10.3). They are relatively fast and achieve high solution precision in the case of moderately sized problems (up to approximately 3000 samples). Moreover, these algorithms can be modified easily for general convex loss functions without additional computational cost. They require computation and inversion of the kernel matrix K however, and are thus overly expensive for large problems.

Greedy Approximation We presented a way to extend this method to large scale problems which makes use of sparse greedy approximation techniques. The latter are particularly well suited to this type of algorithm since they find a low rank approximation of the dense and excessively large kernel matrix K directly, without ever requiring full computation of the latter. Moreover, we obtain sparse (however approximate) solutions, independent of the number of Support Vectors.

Chunking in its different variants is another modification to make large scale problems solvable by classical optimization methods. It requires the breaking up of the initial problem into subproblems which are then, in turn, solved separately. This is guaranteed to decrease the objective function, thus approaching the global optimum. Selection rules, in view of duality theory, are given in section 10.4.3.

Sequential Minimal Optimization (SMO) is probably one of the easiest algorithms to implement for SV optimization. It might thus be the method of choice for a first attempt to implement an SVM. It exhibits good performance, and proofs of convergence have been obtained. Recent research has pointed out several ways of improving the basic algorithm. We briefly sketched one technique which improves the estimation of the constant threshold b and thus also helps select good subsets

more easily. Pseudocode for regression and classification conclude this section.

Iterative Methods Finally, another class of algorithms can be summarized as iterative methods, such as gradient descent, Lagrangian SVM which are extremely simple but which are only applicable for a specific choice of cost function, and online support vector methods. These have the potential to make the area of large scale problems accessible to kernel methods and we expect good progress in the future. While it is far from clear what the optimal strategy might be, it is our hope that the reasoning of Section 10.6.3 will help to propel research in this area.

10.7.3 Future Developments and Code

We anticipate that future research will focus on efficient *approximate* and sparse solutions. This means that, quite often, it is not necessary to find a kernel expansion $f = \sum_{i=1}^{m} \alpha_i k(x_i, \cdot)$, where the α_i are the Lagrange multipliers of the corresponding optimization problem and, instead, a much more compact function representation can be used.

Second, we expect that both lightweight optimizers, which can be deployed on small consumer hardware, and large-scale optimizers, which take advantage of large clusters of workstations, will become available. It is our hope that, in a few years, kernel methods will be readily accessible as plug-ins and toolboxes for many statistical analysis packages.

Finally, the choice of parameters is still a problem which requires further attention. While there exist several promising approaches (see [288, 102, 268]) for assessing the generalization performance, mainly involving leave-one-out estimates or their approximation, the problem is far from solved. In particular, every new bound on the generalization performance of kernel machines will inevitably prompt the need for an improved training algorithm which can take advantage of the bound.

Some readers will miss pointers to readily available code for SVM in this book. We deliberately decided not to include such information since such information is likely to become obsolete rather quickly. Instead, we refer the reader to the kernel-machines website (http://www.kernel-machines.org) for up-to-date information on the topic.

10.8 Problems

10.1 (KKT Gap for Linear Programming Regularizers ●●)
Compute the explicit functional form of the KKT gap for Linear Programming Regularizers. Why can't you simply use the expansion coefficients α_i as in Proposition 10.1?

10.2 (KKT Gap for Sub-Optimal Offsets ●●)
Compute the functional form of the KKT gap for non-optimal parametric parts in the expansion of f, e.g., if $f(x) = \langle \Phi(x), \mathbf{w} \rangle + b$ where b is not optimal. Hint: consider

Theorem 6.22 and prove a variant of Theorem 6.27.

10.3 (Restarting for λ •)
Prove an analogous inequality for $R_{\text{reg}}[f]$ as (10.22) for λ rather than C, i.e. prove

$$R_{\text{reg}}[f_\lambda, \lambda'] \geq R_{\text{reg}}[f_{\lambda'}, \lambda'] \geq R_{\text{reg}}[f_\lambda, \lambda] \text{ for all } \lambda' > \lambda. \tag{10.141}$$

10.4 (Sparse Approximation in the Function Values ••)
State the optimal expansion for approximations of $k(x_i, \cdot)$ by $\tilde{k}_i(\cdot)$ in the space of function values on $X = \{x_1, \ldots, x_m\}$. How many operations does it cost to compute the expansion?

10.5 (Rank-1 Updates for Cholesky Decompositions ••)
Given a positive definite matrix $K \in \mathbb{R}^{n\times n}$, its Cholesky decomposition $TT^\top = K$ into triangular matrices $T \in \mathbb{R}^{n\times n}$, a vector $\mathbf{k} \in \mathbb{R}^n$, and a real number κ such that the matrix $\begin{bmatrix} K & \mathbf{k} \\ \mathbf{k}^\top & \kappa \end{bmatrix}$ is positive definite, show that the Cholesky decomposition of the larger matrix is given by

$$\begin{bmatrix} K & \mathbf{k} \\ \mathbf{k}^\top & \kappa \end{bmatrix} = \begin{bmatrix} T & 0 \\ \mathbf{t} & \tau \end{bmatrix} \begin{bmatrix} T^\top & \mathbf{t}^\top \\ 0 & \tau \end{bmatrix} \tag{10.142}$$

where

$$\mathbf{t} = T^{-1}\mathbf{k} \text{ and } \tau = \left(\kappa - \mathbf{t}^\top \mathbf{t}\right)^{\frac{1}{2}}. \tag{10.143}$$

Why would we replace the equation for τ by $\tau = \max\left(0, \left(\kappa - \mathbf{t}^\top \mathbf{t}\right)^{\frac{1}{2}}\right)$ for numerical stability? How can you compute (10.143) in $O(n^2)$ time?

10.6 (Smaller Memory Footprint for SGMA •••)
Show that rather than caching K^{mn}, $(K^{nn})^{-1}$ and $\alpha = (K^{nn})^{-1}K^{mn}$ (and updating the three matrices accordingly) we can reformulate the sparse greedy matrix approximation algorithm to use only T_n and $T_n^{-1}K^{mn}$ where T_n is the Cholesky decomposition (see Problem 10.5) of K^{nn} into a product of triangular matrices, that is $T_n T_n^\top = K^{nn}$.

In particular, show that the update for $T_n^{-1}K^{mn}$ is given by

$$T_{n+1}^{-1}K^{m,n+1} = \begin{bmatrix} T_n^{-1}K^{mn} \\ \tau^{-1}\left(\bar{k} - \mathbf{t}\left(T_n^{-1}K^{mn}\right)\right) \end{bmatrix} \tag{10.144}$$

where $\mathbf{t}$ and τ are defined as in (10.143).

10.7 (Optimality of PCA •••)
Show that for the problem of approximating a positive definite matrix K by a matrix $\tilde{K}$ of rank n such that both $\tilde{K}$ and $K - \tilde{K}$ are positive definite the solution is given by projecting onto the largest n principal components of K, i.e., by PCA. Here we consider an approximation to be optimal if the residual trace of $K - \tilde{K}$ is minimized. Show that PCA is also optimal if we consider the largest eigenvalue of $K - \tilde{K}$ as the quantity to be minimized. Hint: recall that K is a Gram matrix for some x_i, i.e., $K_{ij} = \langle x_i, x_j \rangle$.

10.8 (General Convex Cost Functions [516] ••)
Show that for a convex optimization problem

$$\begin{array}{ll} \underset{\alpha}{\text{minimize}} & \frac{1}{2}q(\alpha) + \langle c, \alpha \rangle \\ \text{subject to} & A\alpha = d \\ & l \leq \alpha \leq u \end{array} \tag{10.145}$$

with $c, \alpha, l, u \in \mathbb{R}^m$, $A \in \mathbb{R}^{n \cdot m}$, *and* $d \in \mathbb{R}^n$, *the inequalities between vectors holding component-wise, and* $q(\alpha)$ *being a convex function of* α, *the dual optimization problem is given by*

$$\begin{array}{ll} \text{maximize} & \frac{1}{2}\left(q(\alpha) - \langle \partial_\alpha q(\alpha), \alpha \rangle\right) + \langle d, h \rangle + \langle l, z \rangle - \langle u, s \rangle \\ \text{subject to} & \frac{1}{2}\partial_\alpha q(\alpha) + c - (Ay)^\top + s = z \\ & s, z \geq 0, \ h \text{ free} \end{array} \tag{10.146}$$

Moreover, the KKT conditions read

$$g_i z_i = s_i t_i = 0 \text{ for all } i \in [m]. \tag{10.147}$$

10.9 (Interior Point Algorithm for (10.145) [516] ••)
Derive an interior point algorithm for the optimization problem given in (10.145). Hint: use a quadratic approximation of $q(\alpha)$ *for each iteration and apply an interior point code to this modification. Which cost functions does this allow you to use in an SVM?*

10.10 (Sherman-Woodbury-Morrison for Linear SVM [168] ••)
Show that for linear SVMs the cost per interior point iteration is $O(mn^2)$. *Hint: use (10.54) to solve (10.48).*

10.11 (KKT Gap and Optimality on Subsets ••)
Prove that after optimization over a subset S_w *(and adjusting* b *in accordance to the subset) the corresponding contributions to the KKT gap, i.e. the terms* KKT_i *for* $i \in S_w$ *will vanish.*

10.12 (SVMTorch Selection Criteria [502, 108] •)
Derive the SVMTorch optimality criteria for SV regression; derive the equations analogous to (10.60) for the regression setting.

10.13 (Gradient Selection and KKT Conditions •)
Show that for regression and classification the patterns selected according to (10.60) are identical to those chosen by the gradient selection rule, i.e. according to $Q\alpha + c$. *Hint: show that gradient and* $\overline{\mathrm{KKT}}_i$ *differ only in the constant offset* b, *hence taking the maxima of both sets yields identical results.*

10.14 (SMO and Number of Constraints •)
Show that for SMO to make any progress we need at least $n + 1$ *variables where* n *is the number of equality constraints in* $A\alpha = d$. *What does this mean in terms of speed of optimization?*

Find a reformulation of the optimization problem which can do without any equality constraints. Hint: drop the constant offset b from the expansion. State the explicit solution to the constrained optimization problem in one variable.

Which selection criteria would you use in this case to find good patterns? Can you adapt $\mathrm{KKT}_i, \overline{\mathrm{KKT}}_i$, *and* $\underline{\mathrm{KKT}}_i$ *accordingly? Can you bound the improvement explicitly?*

10.15 (Orthogonality and Clipping •)
Prove Lemma 10.6. Hint: first prove that for $a, b \in \mathbb{R}$ *(10.109) holds. The lemma then follows by summation over the coordinates.*

10.16 (Lagrangian Support Vector Machines for Regression ••) *Derive a variant of the Lagrangian Support Vector Algorithm for regression. Hint: begin with a regularized risk functional where b is regularized and the squared* ε*-insensitive loss function* $c(x, y, f(x)) = \max(|f(x) - y| - \varepsilon, 0)^2$. *Next derive an equation analogous to (10.110).*

For the iteration scheme you may want to take advantage of orthogonal transformations such as the one given in (10.52). What is the condition on γ *in this case?*

10.17 (Laplace Approximation ••)
Use Newton's method as described in (6.12) to find an iterative minimization scheme for the regularized risk functional. See also Section 16.4.1 for details. For which cost functions is it suitable (Hint: Newton's method is a second order approach)? Can you apply the Sherman-Morrison-Woodbury formula to find quick approximate minimizers? Compare the algorithm to the Lagrangian Support Vector Machines.

10.18 (Online Learning and Number of Support Vectors ••)
Show that for a classification problem with nonzero minimal risk the number of Support Vectors increases linearly with the number of patterns, provided one chooses a regularization parameter that avoids overfitting. Hint: first show that all misclassified patterns on the training set will become Support Vectors, then show that the fraction of misclassified patterns is non-vanishing.

10.19 (Online Learning with ν-SVM •••)
Derive an online version of the ν*-SVM classification algorithm. For this purpose begin with the modified regularized risk functional as given by*

$$R_{\mathrm{reg}}[f] = \frac{1}{m}\sum_{i=1}^{m} c_\rho(x_i, y_i, f(x_i)) - \rho\nu + \frac{1}{2}\|f\|_{\mathcal{H}}^2. \tag{10.148}$$

Next replace $\frac{1}{m}\sum_{i=1}^{m} c_\rho(x_i, y_i, f(x_i))$ *by the stochastic estimate* $c_\rho(x_t, y_t, f(x_t))$. *Note that you have to perform updates not only in* f *but also in the margin* ρ.

What happens if you change ρ *rather than letting* α_i *decay in the cases where no margin error occurs? Why don't you need* λ *any more? Hint: consider the analogous case of Chapter 9.*