Gradient Based Optimization of Support Vector Machines

Dissertation

Zur Erlangung des Doktorgrades
der Naturwissenschaften
an der Fakultät für Mathematik
der Ruhr-Universität Bochum

vorgelegt von

Tobias Glasmachers
Institut für Neuroinformatik
Ruhr-Universität Bochum

April 2008

Thanks

I want to thank all people who supported me during my work on this thesis, and Eva in particular. Especially I want to thank my supervisors Christian Igel for his continuous support and advice, and Hans Ulrich Simon for actually making this thesis possible.

Contents

Introduction

Supervised learning problems occur in all areas of natural sciences, medicine, engineering, economics, and many other disciplines. A supervised learning problem is stated in terms of sample data in a input space X, with associated label information from a label space Y. The task or learning problem is to generalize from these examples to a hypothesis about the dependency of the labels on the input data, that is, to come up with a function that maps inputs to labels. Of course, we want this hypothesis to be as correct as possible, or to be correct with high probability. Here, the term *supervised* learning refers to the availability of the labels for the sample data (which are thought of as being provided by an abstract supervisor or teacher).

Learning machines are algorithms that learn to make predictions based on the finite (and thus observable) set of ground truth sample data. Given these training examples, this formally amounts to the proposition of a functional relation from the input to the label space. The main problem such algorithms face is that they need to make assumptions when generalizing from the training set to all possible cases. On the other hand it is usually required that a machine does not make too explicit and restrictive assumptions in terms of the structure (for example an explicit parametric form) of a model, in order to remain sufficiently general. Therefore, methods designed to be generally applicable to all kinds of supervised learning problems often rely on non-parametric models.

Learning machines are typically used when explicit models or human expert judges prove to be impractical or unreliable. Humans outperform learning machines in some instances due their superior capability to make use of prior knowledge, or due to the specialized task-specific information processing of the brain, whereas machines outperform humans in a number of other applications. In addition to their often superior performance the development of learning algorithms is also motivated by the possibility of automating learning and prediction for decision making.

Support Vector Machines

The present work focusses on a particular approach to the general learning problem outlined above, namely the support vector machine (SVM), introduced by Cortes and Vapnik. The different variants of support vector machines pose a general and flexible approach to various kinds of machine learning problems. Since their emergence in the late nineties they have been successfully applied to a wide range of problems.

Due to Mercer's theorem, SVMs can be conveniently embedded into the framework of kernel-induced Hilbert spaces. They embed the input space X into a high (and often infinite) dimensional Hilbert space $\mathcal{H}$ with a so-called feature map $\Phi : X \to \mathcal{H}$, and build their hypothesis based on an affine linear function

$$x \mapsto \langle w, \Phi(x) \rangle + b$$

in this space. This computation involves an important trick which enables us to deal with the high-dimensional feature space only implicitly. This limits the scaling of the

computational complexity to the number of training examples instead of the dimension of the feature space. A second important property of SVMs is that their capacity control is independent of the feature space dimension. Support vector machines therefore manage to circumvent the curse of dimensionality in high-dimensional feature spaces.

Because of the convenient mathematical embedding of support vector machines into feature Hilbert spaces, it is possible to obtain powerful learning theoretical performance bounds. Furthermore, support vector machines are known to be universally consistent learning machines, which means that SVMs can learn any classification problem arbitrarily well, given sufficient examples.

Another focus are the algorithms used for machine training. It became necessary to develop highly specialized and efficient algorithms for the task of obtaining a hypothesis from examples. These training procedures have undergone numerous investigations such that different types of convergence results are available. In the second part of this thesis we propose improved algorithms for support vector machine training, accompanied by proofs of convergence, relying on novel proof techniques.

Although support vector machines are relatively well understood, their application to a particular problem, stated in terms of a fixed dataset, is still non-trivial for at least two reasons. First, the generalization performance of the machine strongly depends on the choice of the kernel function and a regularization parameter. Second, the number of computations needed to train the machine may simply exceed the available computing power in a given time frame. Both difficulties are closely connected to the solution of optimization problems.

Machine Training

At first glance, the problem of support vector machine training seems comparatively simple, because it amounts to the solution of a convex quadratic program. Here the main difficulties arise due to the large number of variables, ranging from hundreds and thousands to millions. It is impossible to store the dense kernel matrices occurring in such problems in working memory, making the inversion of these matrices in today's computing environments prohibitive. Therefore, efficient standard techniques like interior point methods are not applicable.

Model Selection

The model selection problem for support vector machines is which kernel function to use for a given dataset, and how to adjust the capacity control via a regularization parameter. Usually we restrict ourselves to a family of kernel functions parameterized by a manageable low-dimensional parameter manifold. The biggest difficulty in the solution of the model selection problem is the choice of a reasonable objective function and a well-suited search strategy, because the "true" objective function, the generalization error, can not be computed from the available data. Thus we need to come up with efficiently computable objective functions that lead to well generalizing machines, and that do not trap search strategies in local optima. For large datasets the choice of objective function and search strategy is additionally restricted by the available computing power.

Gradient-based Optimization

Efficiency is a concern in both of these fundamentally different optimization problems. Whenever the derivative of the objective function is available, it is natural to base the search strategy on some variant of gradient descent (or ascent). For the problem of machine training the Hessian matrix of the objective function is also available, although

it regularly exceeds the available working memory. Nevertheless we will use Newton's method to solve a sequence of sub-problems, where this method directly computes the unconstrained optimum of the quadratic objective function. For the model selection problem the situation is different, because some natural objective functions are not differentiable. We will concentrate on differentiable objective functions and apply simple and robust gradient descent algorithms whenever possible.

Outline of this Thesis

This thesis is divided into three parts, closely following the topics addressed above. In the first part the basic definitions and methods are introduced. We start with the general problem of supervised learning and discuss simple algorithms. Then we will turn to different variants of support vector machines, and finally discuss problems occurring in performance measurement and the comparison of algorithms on the basis of finite datasets.

The second part is devoted to the understanding and speed-up of support vector machine training. We first present the state of the art, as constituted by the specialized SMO decomposition algorithm. By introducing a geometric decomposition of the problem, we can prove a general linear progress rate for free SMO steps. In addition, we develop and evaluate improvements for different aspects of the SMO algorithm.

In particular we present the hybrid maximum gain working set selection algorithm [Glasmachers and Igel, 2006, Glasmachers, 2008a] resulting in extremely fast training on large scale problems. This algorithm selects its working sets based on second order information and is thus superior to the already fast MVP method, which relies on the gradient only. Furthermore, this algorithm makes highly efficient use of the kernel cache, such that it saves up to 50% of the kernel function evaluations per iteration, resulting in excellent performance on large datasets where only a small fraction of the kernel matrix can be cached in working memory. The convergence of the algorithm to the optimum of the support vector machine training problem is proven.

Another approach is the analysis of the optimal step size for the SMO algorithm, given assumptions on the next iteration. We highlight the sub-optimality of the step size resulting from the solution of the SMO sub-problem. With this insight we get new possibilities for the design of SMO-type algorithms. We implement these considerations in the planning-ahead SMO algorithm [Glasmachers, 2008b]. We prove the convergence of the algorithm to the optimum and demonstrate its superiority over the standard variant.

We conclude our studies on the SMO algorithm with the application of a second order working set selection algorithm to an online learning variant of the support vector machine [Glasmachers and Igel, 2008].

In the third part of this thesis we address the problem of model selection. We start with a review of standard methods from the literature, including cross-validation, minimization of performance bounds, and maximization of the kernel target alignment, which can be used for efficient gradient-based model selection [Igel et al., 2007]. Then we develop a new differentiable objective function. This new objective function has the advantage of a natural probabilistic interpretation, such that Bayesian methods for incorporating prior knowledge or regularization can be directly applied. Three main ideas influenced the development of the novel model selection objective. The first ingredient is a systematic symmetrization of the commonly used cross-validation error. This technique can be naturally combined with an efficiently computable probabilistic interpretation of support vector machine outputs, which has the additional advantage of making the objective function differentiable. Then a simple sampling technique is applied to overcome the problems arising from the computational complexity of the symmetrization. Because the sampling technique results in a randomized objective function, standard gradient descent methods do not work well in this case. We develop a gradient descent algorithm that can deal with

the type of uncertainty we face.

The difficulty of the model selection problem is underlined by deriving sufficient criteria for the existence of multiple local optima for standard model selection objective functions [Glasmachers, 2006]. A study on gradient-based adaptation of covariance matrices of Gaussian kernels [Glasmachers and Igel, 2005] sheds light on further pitfalls in model selection.

To investigate how to approach the model selection problem, we conduct a systematic comparison of the performance of different model selection strategies on both artificial and real world problems. As an example we finally apply our methods to an interesting problem from bioinformatics, where the task is to identify the beginning of protein encoding gene sequences in sequences of nucleotides [Mersch et al., 2006, 2007, Igel et al., 2007].

Part I

Background

Chapter 1

The Learning Problem

The aim of this chapter is to introduce basic concepts, problems, and notations serving as a base for this thesis. Some simple yet powerful learning machines are presented and the concept of regularization is surveyed.

1.1 Supervised Learning

To define the general problem of statistical learning theory we need some notations. We consider two metric spaces X and Y, where X is called the input space and Y the label space. These spaces are equipped with the Borel σ-algebras induced by the corresponding open sets. We assume that there exists a fixed but unknown probability distribution ν on the product $X \times Y$, called the data generating distribution. Supervised learning deals with the extraction of properties of this distribution from a finite number of random variables, namely from the labeled training dataset

$$D = \big((x_1, y_1), \ldots, (x_\ell, y_\ell)\big) \in \mathcal{D} = \bigcup_{\ell=1}^{\infty} (X \times Y)^\ell \ ,$$

drawn i.i.d. from ν, that is, $D \sim \nu^\ell$. It turns out that the independence assumption is the most restrictive part of this general problem definition. In particular for data that stem from processes embedded in time a natural description usually violates this condition. Nevertheless, *per definition*, all supervised learning problems are representable (possibly in an unnaturally involved way) in this form.

According to Vapnik [1998], statistical learning theory can be split into three basic classes of problems. In the first two scenarios we try to describe the most important properties of ν by a measurable hypothesis $h : X \to Y$. The third learning problem is the description of the distribution ν itself in terms of approximations of its density[1].

1. A discrete (and usually finite) set Y defines a classification problem. The most basic and important case of binary classification deals with only two classes, usually labeled -1 and $+1$. This is the simplest non-trivial learning problem.

2. In the case $Y = \mathbb{R}$ we face a regression problem[2].

[1] The density, usually with respect to the Lebesgue measure or the counting measure, may not exist. Nevertheless, density estimation aims at finding a density inducing a distribution that approximates ν.

[2] Any differentiable manifold gives rise to a regression problem. However, most loss functions and algorithms make more or less explicit use of the reals or of some $\mathbb{R}^n$ as their label space.

3. Instead of estimating a map $h : X \to Y$ we approximate the density of ν (if it exists). Because we try to estimate the complete set of properties of the distribution at once, this is the most difficult case.

These problems are ordered by their difficulty. If we have a solution to the density estimation problem, that is, a density that approximates ν well, we can solve the simpler problems easily using a so called plug-in model. Unfortunately, the rate of convergence of asymptotically consistent density estimators is much slower than for example for asymptotically consistent classifiers. Therefore it does not make much sense to first estimate a density which is then used for classification. Because we are mainly interested in classification and regression problems, we will not consider density estimation here.

1.1.1 Noisy Data

As we are interested in the estimation of a map $h : X \to Y$ we can define the notion of noise. For each x such that the conditional distribution ν_x is not a Dirac distribution (that is, multiple values of y may occur for given x) we say that the problem is noisy. This is because no matter how we choose $h(x)$ there is a positive probability of predicting the wrong class y given the input x. In a noise free problem, for example in a simple classification task, it should be doubtlessly clear which class needs to be assigned to an input.

This notion of noise is nothing but a definition of the wording used in the remainder of this thesis. It does not emphasis on the reason why the distribution of y given x is blurred. In fact, classical noise in a measurement process may be a reason, but incomplete observations of an otherwise noise free problem can have the same effect, or the process generating the data may in principle be random like in standard quantum mechanics. By using the term *noisy* we do not emphasis on the reason of uncertainty, but rather want to quantify a particular property related to the difficulty of the learning problem at hand.

1.1.2 Hypotheses and Learning Machines

Let $\mathfrak{H} = \{h : X \to Y \text{ measurable}\}$ denote the set of all possible hypotheses. A learning machine is a mapping

$$A : \mathcal{D} \to \mathfrak{H} \ ,$$

mapping datasets to hypotheses[3]. This definition enables us to deal with datasets of arbitrary finite size. In this thesis, up to Chapter 8, we will consider so-called batch learning. That is, the learning machine can access the whole training dataset at all times. This is in contrast to online learning where only one example is presented at a time and the machine is required to quickly update its hypothesis with each observation of a training example. In contrast to batch learning, the hypotheses output by online learning algorithms usually depend on the order of the observations. Therefore the online learning case is not captured by the above definition of a learning machine.

In many cases we have an algorithm which outputs a hypothesis h given a dataset D. Such an algorithm realizes a statistical learning machine A. The process of determining h from D is called training of the learning machine. Therefore we will often call the dataset D the training dataset.

[3] A straight forward generalization of this definition allows for stochastic algorithms by mapping $\mathcal{D}$ to the distributions over $\mathfrak{H}$. However, we will mainly consider deterministic machines.

1.1.3 Loss Functions and Risk

The goal of supervised learning is to achieve a low generalization error, that is, a minimal loss on average. A loss function is a function

$$L : X \times Y \times Y \to \mathbb{R}^{\geq 0}$$

with the property $y = y' \Rightarrow L(x, y, y') = 0$. The loss measures the amount of error of a hypothesis $h \in \mathfrak{H}$ on a labeled example $(x, y) \in X \times Y$ as $L(x, y, h(x))$. For a correct prediction of the hypothesis the error vanishes. Most loss functions used in practice do not depend on the input x. In fact, the dependency on $x \in X$ is often left out of the definition.

Considering input space invariant loss functions, any classification loss functions takes at most $|Y| \cdot (|Y| - 1)$ values and is usually described by a quadratic cost matrix with zero diagonal. The important special case of 0-1-loss is given by the cost matrix with value 1 in all off-diagonal entries.

The by far most important loss function for regression is the squared error $(y - y')^2$. Together with the Laplace loss $|y - y'|$ this quantity can be generalized to the concept of p-loss $|y - y'|^p$.

The risk $\mathcal{R}_\nu : \mathfrak{H} \to \mathbb{R}^{\geq 0}$ of a hypothesis h is defined as the expected error

$$\mathcal{R}_\nu(h) = \mathbb{E}_\nu[L(x, y, h(x)] = \int_{X \times Y} L\big(x, y, h(x)\big)\, d\nu(x, y) \ .$$

In the context of machine learning it is often called generalization error, because it coincides with the expected loss or error of a hypothesis on an "unseen" example, that is, an example independent of D and therefore of $h = A(D)$. This quantity measures how well the hypothesis generalizes its "knowledge" to examples not used for training.

The generalization error of a learning algorithm A on a given dataset D is just the risk of the hypothesis $A(D)$. However, the dataset D is a random variable and we can not know how probable its actual realization is. Usually it makes more sense to ask for the performance of an algorithm for a particular type of problem, that is, for a data generating distribution ν and a dataset size ℓ. Given ν and ℓ we want to find a learning machine which minimizes the risk

$$\int_{(X \times Y)^\ell} \mathcal{R}_\nu(A(D))\, d\nu^\ell(D)$$

over all datasets of a particular size.

Already at this general stage it becomes apparent that supervised learning is closely connected to optimization. If the distribution ν was known, we would define the risk of a hypothesis as an objective function to minimize. We will see that this connection is indeed close for most types of learning machines discussed later on.

1.1.4 Test Error Estimate of the Generalization Error

The goal of learning is to find a hypothesis with a small generalization error. However, as this quantity is defined as the expected value of the loss of the hypothesis we need to know the data generating distribution in order to compute it. Regularly we are faced with ground truth coming from measurements or human expert judges which are impossible to model completely. Even if we have analytical expressions for the data generating distribution and the hypothesis it is in most cases impossible to solve the integral involved in the computation of the generalization error. Instead it is common to estimate the

generalization error of a hypothesis from its performance on a test set. A test set is an i.i.d. sample $\big((\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_N, \tilde{y}_N)\big) \sim \nu^N$ independent of the training set (of course, independence does not imply that training and test set are disjoint). Then the test error, defined as the average loss

$$E_{\text{test}} = \frac{1}{N} \sum_{n=1}^{N} L(\tilde{x}_n, \tilde{y}_n, h(\tilde{x}_n))$$

of the test dataset, is an unbiased estimate of the true risk $\mathcal{R}_\nu(h)$. Assume we have a classification task with 0-1-loss. The variance of E_{test} is given by

$$\text{Var}(E_{\text{test}}) = \frac{\mathcal{R}_\nu(h)(1 - \mathcal{R}_\nu(h))}{N} \leq \frac{1}{4N} \ .$$

The binomial distribution of the test error is well approximated by a normal distribution for large N and we can use the normal distribution quantiles to estimate how close E_{test} is to $\mathcal{R}_\nu(h)$. For example, we may have measured a test error of 10% on a test set of size $N = 10,000$. Then, plugging the estimate E_{test} in instead of the unknown mean $\mathcal{R}_\nu(h)$, we estimate that E_{test} deviates by at most $\gamma_{0.95}\sqrt{\frac{0.1 \cdot 0.9}{10,000}} \approx 0.005$ from its expected value with a probability of 90%. Here $\gamma_{0.95} \approx 1.645$ is the 95%-quantile of the standard normal distribution, with $\gamma_{0.05} = -\gamma_{0.95}$ due to symmetry. Thus, it is well justified to expect the true risk in the interval $[0.095, 0.105]$. If we are not satisfied with this accuracy we have to sample more test data, that is, perform additional measurements.

Because we are nearly never in the position to compute the generalization error directly we will use test errors to judge or compare hypotheses. As test errors are random variables (even for fixed training dataset) we need statistical tests for such comparisons.

1.1.5 Bayes Optimality and Consistency

If the data generating distribution was known the learning problem would reduce to the minimization of the risk. This problem depends on the distribution ν and the loss function L, but not on the data D. Because ν is assumed to be unknown (in contrast to D) it is not relevant for the solution of the supervised learning problem, but it gives some general insights.

A hypothesis minimizing the risk is called Bayes-optimal. The Bayes risk is defined as

$$\mathcal{R}^*_\nu := \inf_{h \in \mathfrak{H}} \{\mathcal{R}_\nu(h)\} \ .$$

Any hypothesis h^* realizing the Bayes risk $\mathcal{R}_\nu(h^*) = \mathcal{R}^*_\nu$ is called Bayes optimal. By definition the Bayes risk is a lower bound for the risk $\mathcal{R}_\nu(h)$ of any hypothesis h. Therefore the problem of minimizing the risk is equivalent to approximating the Bayes risk. It is important to note that in general the Bayes risk is not zero.

For example, given a classification problem with 0-1-loss, any hypothesis $h^* : X \to Y$ with

$$h^*(x) = \arg\max \Big\{ P(y \,|\, x) \ \Big| \ y \in Y \Big\}$$

is Bayes optimal, no matter how ties are broken.[4] Here, $P(y \,|\, x)$ denotes the conditional probability to observe y given x under the distribution ν.

[4] We assume that ties are broken in a measurable fashion in order to define a proper hypothesis.

Let $(D_\ell)_{\ell\in\mathbb{N}}$ be a random sequence of datasets with $|D_\ell| = \ell$, all sampled i.i.d. from ν. We call a learning machine $A : \mathcal{D} \to \mathfrak{H}$ (weakly) consistent for the problem ν if it fulfills

$$\lim_{\ell\to\infty} \mathcal{R}_\nu(A(D_\ell)) = \mathcal{R}^*_\nu \ ,$$

that is, if its loss converges to the Bayes risk in probability (see Devroye et al., 1996, Chapter 6.1). If this property holds for all distributions ν we call the learner universally consistent. Such machines indeed exist and we will see examples of universally consistent classifiers later. It is well known that for each universally consistent learning machine there exists a sequence $(\nu_n)_{n\in\mathbb{N}}$ of distributions such that the rate of convergence becomes arbitrary slow in the limit. This result is closely connected to so called "no-free-lunch" theorems [Wolpert and Macready, 1995] stating that no learning machine is superior to all other learners on all problems. In other words, universal consistency of a machine is the best we can get, but there is no guarantee how close the risk of a particular hypothesis is to the Bayes risk for a fixed distribution ν, a fixed dataset size ℓ, or for a fixed dataset D.

1.1.6 Empirical Risk Minimization

Our complete knowledge about the data generating distribution ν comes from a finite dataset $D = \big((x_1, y_1), \ldots, (x_\ell, y_\ell)\big) \in \mathcal{D}$. This dataset defines the empirical distribution

$$\hat{\nu} = \frac{1}{\ell} \sum_{i=1}^{\ell} \delta_{(x_i,y_i)}(x, y)$$

where $\delta_{(x,y)}$ denotes the Dirac peak at $(x, y) \in X \times Y$. The empirical distribution naturally leads to the definition of the empirical risk

$$\hat{\mathcal{R}}_D(h) = E_{\text{train}} = \mathbb{E}_{\hat{\nu}}[L(x, y, h(x)] = \frac{1}{\ell} \sum_{i=1}^{\ell} L\big(x_i, y_i, h(x_i)\big)$$

which is simply the error of a hypothesis h on the training data D. Therefore the empirical risk is often termed training error. Note that the formula coincides with the computation of the test error on an independent test dataset. Unlike the loss on a test set the training error is often an over-optimistic (and thus strongly biased) estimate of the risk.

In contrast to the true risk, the empirical risk can be computed given only the hypothesis and the training data. It is a straight forward learning strategy to minimize the empirical risk over a predefined hypothesis class $H \subset \mathfrak{H}$, that is, to choose

$$\hat{h} := \arg\min \Big\{ \hat{\mathcal{R}}_D(h) \ \mid \ h \in H \Big\}$$

as a hypothesis. This proceeding is commonly referred to as empirical risk minimization. It defines a learning machine[5] $A_H : \mathcal{D} \to H$ for each class H of hypotheses as long as the minimum is unique. Otherwise an additional rule is needed to break ties.

Here we have a more concrete example how learning translates to optimization, because the empirical risk objective function $\hat{\mathcal{R}}_D$ is completely known (in contrast to the true risk) and usually efficiently computable. In certain situations we can explicitly construct an algorithm realizing this learning machine, for example if H is finite or if the empirical risk minimization gives rise to a convex optimization problem. The latter usually requires the loss function to be convex. For this reason convex loss functions play an important role in learning theory.

[5] We ignore the issue whether the output of this learning machine is efficiently computable.

1.1.7 The Overfitting Problem

We should remark that empirical risk minimization with the appealing choice $H = \mathfrak{H}$ does not lead to consistent learning machines. This is obvious from an example given by Leibniz. He realized that any finite set of points blindly distributed on a sheet of paper can be perfectly explained by a polynomial curve running through all these points. Of course, a polynomial curve is no plausible *explanation* for the randomly distributed points on the paper (Leibniz, 1686, page 33).

Analogously we can fit a measurable hypothesis with vanishing empirical risk to any non-contradicting dataset,[6] and usually there is a large set of such hypotheses available. It is well known that for a distribution with non-zero Bayes risk $\mathcal{R}^*_\nu > 0$ any algorithm choosing a hypothesis $h \in \mathfrak{H}$ with minimal empirical risk can not be asymptotically consistent. Such an algorithm will perfectly explain the data of a noisy distribution. It is intuitively clear that by fitting noisy improbable events the hypothesis is forced to make mistakes on more probable unseen regions in the vicinity of such points.

Asymptotically, a consistent algorithm whose training error approximates the Bayes risk for large datasets will do better because it accounts for the presence of noise. It will not try to fit noisy inputs too well, but instead use, for example, localized majority votes (see Section 1.2.4).

This phenomenon is regularly observed when training empirical risk minimizers on large hypothesis classes. It is commonly called overfitting. The hypothesis minimizing the empirical risk is endangered to fit the training dataset D "too well" to be able to generalize to unseen points. This happens because the best empirical risk minimization can do is to assume that in the training points the probability $P^\nu(y_n|x_n)$ of observing the training label is one. The possibility of random effects is ignored.

Tychonoff was the first to realize that regularization of such ill-posed problems is a way to escape this pitfall. See Section 1.3 and the book by Vapnik [1998] for a discussion.

1.2 Simple Algorithms

There are a couple of simple learning rules, some of which can be naturally interpreted as empirical risk minimizers. In their simplest forms these algorithms build their hypothesis class from the linear functions on the space $X = \mathbb{R}^n$. The most important exception are nearest neighbor classifiers. It is much more adequate to view them as quite simple prototype learners instead of deriving them as empirical risk minimizers. They result in piecewise linear classification boundaries, selected from a relatively rich hypothesis class (compared to hypotheses built from linear functions).

The presentation of these simple algorithms has a two-fold purpose. First, these machines can serve as a baseline for the evaluation of more sophisticated methods, and, beyond that, often work well in practice. Second and more importantly, powerful and highly complex machines can be derived from the simple ones presented in this section.

The most simple learning machines are linear machines in the sense that they operate in a (usually finite dimensional) real vector space X and build their hypothesis class from the class of linear functions $f : X \to \mathbb{R}$ or linear maps $f : X \to \mathbb{R}^m$. Affine linear maps are a common extension. These can be viewed as linear maps on $X \times \mathbb{R}$ with the input space embedding $X \times \{1\} \subset X \times \mathbb{R}$.

The classes of linear and affine linear maps are directly suitable for regression with $Y = \mathbb{R}$ or $Y = \mathbb{R}^m$. Binary classification is usually realized by thresholding at zero with label space $Y = \{\pm 1\}$. An (affine) linear function f can be turned into a classifier

[6] We call a dataset non-contradicting if there is no pair $i \neq j$ such that $x_i = x_j$ and $y_i \neq y_j$.

hypothesis h by

$$h(x) = \text{sign}(f(x)) = \begin{cases} +1 & \text{if } f(x) > 0 \\ -1 & \text{if } f(x) \leq 0 \end{cases} .$$

In contrast to the usual definition our sign function will assign $\text{sign}(0) = -1$ in order to map each value to a valid label, which makes sense in the context of binary classification. We call the set $\{x \in X \mid f(x) = 0\}$ the decision boundary of the hypothesis h. Note that the functions $f(x)$ and $\lambda f(x)$ result in the same decision boundary for all $\lambda \neq 0$ and even in the same classification hypothesis for all $\lambda > 0$.

Of course, the assignment of the negative class for $f(x) = 0$ is arbitrary. To avoid the asymmetry one could introduce randomized hypotheses which toss a coin on the decision boundary. However, as the decision boundary is usually a zero set (at least if ν has a density w.r.t. the Borel measure), we need not care for the prediction directly on the decision boundary too much.

A large bundle of classical algorithms operating on linear or affine linear hypotheses is available. Many methods from parametric statistics fit into this picture. However, these hypothesis classes are relatively simple and small, in particular for finite dimensional input spaces. On the one hand this simplifies the analysis of such algorithms. On the other hand the restriction to too simple hypotheses imposes a priori limitations. This is an undesirable property in general which is also called underfitting, resulting in poor generalization performance on problems with a complicated structure.

In Section 2.1 we will introduce a unified way to turn simple linear hypotheses into flexible non-linear ones and show how to extend them to general input spaces without vector space embedding. This directly translates to a large and important class of learning machines [Vapnik, 1998].

1.2.1 Mean Classifier

One of the simplest learning machines is the mean classifier. It just computes the class-wise means or centers of mass

$$c_y = \frac{1}{|\{n \mid y_n = y\}|} \sum_{n \mid y_n = y} x_n$$

of the training data in a vector space X and classifies an input x according to the label of the nearest class center

$$h(x) = \arg\min\{d(x, c_y) \mid y \in Y\} .$$

In the case of binary classification we write c_+ and c_- for the class centers and get the hypothesis

$$h(x) = \text{sign}(f(x)) \quad \text{with} \quad f(x) = \frac{(c_+ - c_-)^T(2x - c_+ - c_-)}{(c_+ - c_-)^T(c_+ - c_-)}$$

built from the affine linear function which is -1 in the center of the negative class, $+1$ in the center of the positive class and whose gradient points from the negative to the positive class center.

1.2.2 Maximum Margin Classifier

Let us assume that we are given a linearly separable binary classification dataset. That is, there exists an affine linear function $f : X \to \mathbb{R}$ such that $y_n f(x_n) > 0$ for all

$n \in \{1, \ldots, \ell\}$. Then, a slight variation of f in an open neighborhood will also separate the training data. In fact, there exist infinitely many ways to linearly separate the classes without making an error on the training set. We have infinitely many choices of empirical risk minimizers. This enables us to define a secondary criterion which then identifies a single hypothesis out of all empirical risk minimizers.

We write the affine linear function in the form $f(x) = \langle w, x\rangle + b$ and obtain the decision boundary $S = \{x \in X \,|\, f(x) = 0\} = \{x \in X \,|\, \langle w, x\rangle = -b\}$ which is a hyperplane. It is geometrically intuitive that it is unnecessarily dangerous to pick a hypothesis such that the hyperplane is close to training points. These points seem to be correct only by chance. This intuition leads to the maximum margin classifier. We define the geometric margin of a training example (x, y) w.r.t. a function $\langle w, x\rangle + b$ as the signed distance $yf(x)/\|w\|$ to the decision boundary. This quantity is positive if (x, y) is classified correctly. The simpler concept of the functional margin is defined as $yf(x)$, which measures the distance to the decision boundary in terms of the function value. Again, the functional margin is positive for correctly classified examples, zero for examples on the decision boundary and negative for examples on the wrong side of the class boundary.

Vice versa, we assign a margin to an affine linear function as the minimal margin of all training points. This way we assign a geometric and a functional margin to the decision function. The geometric margin is even a property of the separating hyperplane, as it is invariant under scaling of the affine linear function.

The maximum margin classifier is the unique linear classifier that maximizes the geometric margin, that is, the minimal distance over all training examples from the decision boundary:

$$(w^*, b^*) = \arg\max \left\{ \min_{1 \le n \le \ell} \left\{ \frac{y_n(\langle w, x_n\rangle + b)}{\|w\|} \right\} \;\middle|\; w \in X \setminus \{0\}, b \in \mathbb{R} \right\}$$

The solution of this problem is unique up to scaling, that is, $(\lambda w^*, \lambda b^*)$ describes the same optimal decision boundary for all $\lambda > 0$. For an appropriate normalization like $\|w\| = 1$ we can instead maximize the simpler concept of the functional margin

$$(w^*, b^*) = \arg\max \left\{ \min_{1 \le n \le \ell} \left\{ y_n(\langle w, x_n\rangle + b) \right\} \;\middle|\; w \in X, b \in \mathbb{R}, \|w\| = 1 \right\} .$$

The more common normalization to minimize $\|w\|$ under the functional margin constraint $y_n f(x_n) \ge 1$ for all $n \in \{1, \ldots, \ell\}$ is equally well suited. With this approach the maximum margin classifier becomes

$$(w^*, b^*) = \arg\min \left\{ \frac{1}{2}\|w\|^2 \,\middle|\, w \in X,\, b \in \mathbb{R},\, y_n(\langle w, x_n\rangle + b) \ge 1 \text{ for } 1 \le n \le \ell \right\} .$$

Besides its intuitive plausibility maximum margin classification turns out to be a powerful concept:

Theorem 1.1 (Theorem 7.2 by Shawe-Taylor and Cristianini [2004])**.** *Fix $\delta > 0$. Suppose that a training dataset*

$$D = \big((x_1, y_1), \ldots, (x_\ell, y_\ell)\big) \sim \nu^\ell$$

drawn i.i.d. according to a distribution ν is linearly separable and let $h(x) = \langle x, w\rangle + b$ be the maximum margin classifier with geometric margin $\rho > 0$. With probability at least $1 - \delta$ over the training dataset D, the generalization error of h is bounded by

$$\frac{4}{\ell \cdot \rho}\sqrt{\operatorname{tr}(G)} + 3\sqrt{\frac{\ln(2/\delta)}{2\ell}}$$

where $G \in \mathbb{R}^{\ell \times \ell}$, $G_{ij} = \langle x_i, x_j\rangle$ is the Gram matrix of the inputs $x_n \in X$.

This theorem supports the intuition that it is the safest to use as large margins as possible when aiming at good generalization performance. We refer to the textbook by Shawe-Taylor and Cristianini [2004] and the very good book chapter by Bartlett and Shawe-Taylor [1999] for a much more detailed discussion of this type of generalization bound results.

1.2.3 Linear Regression

Linear regression with a quadratic loss function is a classical example of empirical risk minimization. We consider an input space $X \subset \mathbb{R}^n$ and a label space $Y = \mathbb{R}^m$ and try to find an affine linear map $h : X \to Y$ minimizing the quadratic loss $\|y - h(x)\|^2$ on the training data. Our model can be written as $f(x) = Mx + b$ with $M \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, resulting in

$$(M^*, b^*) = \arg\min \left\{ \sum_{i=1}^{\ell} \|Mx_i + b - y_i\|^2 \;\middle|\; M \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m \right\} .$$

This poses an unconstrained convex quadratic optimization problem and its solution can be obtained by setting the gradient of the objective function to zero. For $m = 1$, with the quantities

$$\bar{x} = \frac{1}{\ell} \sum_{i=1}^{\ell} x_i \qquad \bar{y} = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$
$$\tilde{X} = \sum_{i=1}^{\ell} (x_i - \bar{x}) x_i^T \qquad \tilde{y} = \sum_{i=1}^{\ell} (y_i - \bar{y}) x_i^T$$

constructed from the training dataset, we obtain $M^* = \tilde{y}\tilde{X}^{-1}$ and $b^* = \bar{y} - M^*\bar{x}$ where $\tilde{X}^{-1}$ is the generalized inverse of $\tilde{X}$. For $m > 1$ the same calculation holds componentwise.

If there is no exact linear relation of training inputs and labels (in particular for $\ell > n$) the method will not achieve zero training error. It is well known that linear regression computes the maximum likelihood estimate under the model assumption $y_i = Mx_i + b + \varepsilon_i$ with i.i.d. zero mean Gaussian random variables ε_i. The linear regression hypothesis yields the best approximation to the data in the least squares sense.

1.2.4 Nearest Neighbor Classifiers

For a given input $x \in X$ the k-nearest-neighbor classifier computes the k training examples closest to x and outputs the class label according to a majority vote of the corresponding training labels. The different variants of this general scheme vary in how ties are broken in the distance minimization as well as the majority voting. In addition, weighting schemes depending on the distance ranks can be applied [Devroye et al., 1996].

Although it uses simple distance computations only, the nearest neighbor classifier outputs arbitrarily complex hypotheses. The hypotheses have piecewise linear decision boundaries. Such hypotheses can be computed based on affine linear functions, but they are much more complex than a binary classification decision based on a single thresholded linear function. For $k = 1$ we get an empirical risk minimizer, while in general the number of training errors will increase for growing k.

The k-nearest neighbor classifier has undergone numerous investigations and most of its variants are well understood. The most prominent result is Stone's Theorem [Stone, 1977], from which we obtain that the algorithm is a universally consistent learner (see

Section 1.1.5) whenever the number k of neighbors grows with the training dataset size ℓ like

$$\lim_{\ell\to\infty} k(\ell) = \infty \quad \text{and} \quad \lim_{\ell\to\infty} \frac{k(\ell)}{\ell} = 0 \ .$$

The nearest neighbor classifier can be defined in any metric space, but it is usually considered in a real vector space with Euclidean norm and induced Borel σ-algebra. Due to the simplicity of the computations involved it is reasonable to consider it basic. Nevertheless it has the capacity to solve the supervised learning problem in a universally consistent manner.

Compared to hypotheses resulting from linear functions the computation of a single prediction of the nearest neighbor classifier, at least in a naïve implementation, amounts to $\mathcal{O}(\ell)$ operations because the distance of the pattern in question to all training points needs to be computed. For the same reason the memory requirement scales analogously.

1.3 Regularized Learning Machines

Empirical risk minimizers on large hypothesis classes tend to overfit the training data, that is, they achieve sub-optimal generalization performance. At the same time their decision boundaries are relatively complex, often even compared to the Bayes optimal decision boundary. It turns out that "real" problems tend to be simple, compared to what would have been possible within the general framework of a data generating distribution ν. This agrees with a principle often referred to as Occam's razor "*entia non sunt multiplicanda praeter necessitatem*" (entities should not be multiplied beyond necessity), stating that simple explanations are more plausible and simple models more explanatory and more reliable than complicated ones.

Instead of considering the problem relatively simple compared to the amount of information contained in the training data we may take the opposite point of view (or have the hope) that the training dataset is rich enough to reliably estimate important properties of the problem described by ν. Therefore we can afford not to trust empirical risk minimization too much, as we expect it to result in a too complicated solution.

Occam's razor is an empirical and intuitive statement. In the context of statistical learning machines it can be translated into mathematical terms in several ways using complexity measures like VC-dimension (see for example Section 1.3.3), Rademacher complexity, or the minimal description length principle. It is commonly agreed that simplicity should be an additional learning objective used to obtain hypotheses for prediction. However, simplicity conflicts with empirical risk minimization is general.

There are two common ways to incorporate the goal of simplicity into empirical risk minimization, that is, to resolve the above conflict: Either the class of hypotheses is restricted to a subclass of simple hypotheses, or a penalty term depending on a measure of complexity of the hypothesis is added to the empirical risk objective, scaled by a trade-off parameter. In the first case complexity is a property of a class of hypotheses, while the second approach requires that we can measure the complexity of a single hypothesis which rarely makes sense. Of course, for a nested sequence of growing hypothesis classes we can define the complexity of a hypothesis as the complexity of the smallest class containing this hypothesis. Whenever we talk about the complexity of a hypothesis we mean the complexity of the smallest class containing the hypothesis w.r.t. a fixed complexity measure.

Both proceedings regularize the learning process as they favor hypotheses that are simpler than the empirical risk minimizer. For a given measure of complexity we either control an upper bound on the complexity or the weight of the complexity term controlling the trade-off between the conflicting objectives. In both cases this parameter is

called the regularization parameter, as it controls the amount of regularization, or the amount of enforced simplicity, of the resulting hypothesis. In any case the measure of simplicity/complexity used as well as the actual choice of the value of the regularization parameter may greatly influence the resulting hypothesis.

The method of defining a nested sequence of hypotheses with bounded complexity is known as structural risk minimization [Vapnik, 1998]. This technique is not in contrast to empirical risk minimization but rather complements it. Within structural risk minimization we pick a hypothesis which minimizes the empirical risk within a class of hypothesis with simple structure.

1.3.1 The Role of Regularization for Consistency

As outlined above empirical risk minimization does not lead to universally consistent learning, that is, to learning machines whose risk converges to the Bayes risk in the limit $\ell \to \infty$ for all data generating distributions ν. This is because the problem of learning a well generalizing hypothesis from a finite dataset is ill-posed.

Consider the situation of a growing sequence of datasets D_ℓ with $|D_\ell| = \ell$. To achieve consistency we need to ensure that the larger ℓ the closer the risk $\mathcal{R}_\nu(A(D_\ell))$ of the resulting hypothesis comes to the Bayes risk. Tychonoff was the first to realize that regularization is the key to ensure this property for ill-posed problems. For example, the Moore-Penrose pseudo-inverse of a matrix can be obtained as the limit over a sequence of well-defined approximations with decaying regularization. Let $M \in \mathbb{R}^{n \times m}$ be any matrix, and let $M^{-1} \in \mathbb{R}^{m \times n}$ denote its pseudo-inverse. Then M^{-1} can be obtained as

$$\begin{aligned} M^{-1} &= \lim_{\lambda \to 0} (M^T M + \lambda I)^{-1} M^T \\ &= \lim_{\lambda \to 0} M^T (M M^T + \lambda I)^{-1} \end{aligned}$$

where I is the corresponding identity matrix and λ is the regularization parameter. Note that for all $\lambda > 0$ the inverse matrices on the right hand side are well defined.

Analogously, in the context of learning the problem of obtaining a hypothesis from a large class needs to be regularized in order to become well-posed. In general it is necessary to use a decaying amount regularization for large ℓ in order to achieve consistency. It depends on the particular type of learning machine used how this translates to asymptotic rules for the dependency of the regularization parameter on the dataset size or other quantities. Of course, the complexity of the hypothesis class must be allowed to grow with the dataset size, but on the other hand it must grow moderately in order to avoid overfitting in the limit.

A learning machine together with an asymptotic rule for the regularization parameter can lead to consistency for all possible data generating distributions ν, that is, to universally consistent learning. The first result of this type was obtained by Stone (refer to Section 1.2.4). In particular, his analysis covers the k-nearest-neighbor classifier. Later universal consistency was obtained for a number of learning machines. Steinwart [2001] investigated universal consistency of support vector machines. It turns out that regularized large margin classification can be made universally consistent provided the kernel function implies a rich enough hypothesis space (see Section 2.1.4).

1.3.2 Regularization for Finite Datasets

The asymptotic dependency of the regularization parameter on the dataset size does not tell us how to choose its value in order to achieve the best possible performance for a particular finite dataset size ℓ or a fixed dataset D describing a particular problem. But this is the most important task when applying such machines to real world data.

Occam's razor and the overfitting behavior of empirical risk minimizers give us good reasons to regularize the learning process. This means that we sacrifice some performance on the training set for simplicity of the hypothesis. In other words, we will usually make some training errors in order to achieve the desired level of simplicity. Of course, it is impossible to generally answer the question which training examples not to trust. This question will be answered implicitly by the choice of a measure of complexity on the hypothesis space. This complexity control mechanism is usually considered part of the learning machine. Further there is no a priori way of telling a good value for the regularization parameter. This problem is left to the user of the learning machine in the hope that prior knowledge and experience can guide the selection.

Often learning machines are used without sufficient experience either with the type of machine used or with the data at hand to make such an expert guess. In this situation, and as we are in general interested in the automation of supervised learning, we would like to have a data driven way to adjust the regularization parameter as well as further machine specific parameters to the needs of the distribution ν and the dataset size ℓ. Part III of this thesis is devoted to this problem.

1.3.3 Complexity Control via VC-dimension

As indicated above the VC-dimension (named after V. Vapnik and A. Chervonenkis) of a class of hypotheses is a measure of their complexity. The core concept is only applicable to the case of binary classification.

We say that a class H of hypotheses shatters a finite set of points $\{x_1, \ldots, x_n\} \subset X$ if for each binary label combination $(y_1, \ldots, y_n) \in Y^n$ there is $h \in H$ fulfilling $h(x_1) = y_1, \ldots, h(x_n) = y_n$. Vapnik and Chervonenkis [1971] define the VC-dimension of H as the cardinality of the largest set shattered by H, or infinity if H shatters arbitrary large finite sets.

The VC-dimension of a set H will be denoted by $\mathrm{VCdim}(H)$. We can interpret the VC-dimension as follows: Consider empirical risk minimization on a dataset of size ℓ with extremely noisy labels over a set H of hypotheses. For $\ell \leq \mathrm{VCdim}(H)$ it is possible that the resulting hypothesis achieves zero training error irrespectively of the actual labels. With high probability this solution just fits noise. In contrast, for $\ell > \mathrm{VCdim}(H)$ the chance to obtain zero training error decays quickly (see Theorem 4.3 by Vapnik, 1998) with the number of separations of ℓ points by hypotheses in H, resulting in the inequality

$$P(\hat{\mathcal{R}}_D(h) = 0) \leq \left(\frac{e \cdot \ell}{\mathrm{VCdim}(H)} \right)^{\mathrm{VCdim}(H)} \Big/ 2^\ell$$

where the probability is w.r.t. the uniform distribution on Y^ℓ of the labels.

When learning a noisy distribution and we hope to average the noise out, we should use a function class with a VC-dimension smaller (and usually much smaller) than ℓ. This formally translates to a bound obtained by Vapnik [1998] for a hypothesis h obtained from a training dataset D of size ℓ by minimizing the empirical risk over a class of hypotheses H. With probability at least $1 - \delta$ over the datasets D of size ℓ we have

$$\mathcal{R}_\nu(h) \leq \hat{\mathcal{R}}_D(h) + \frac{\mathcal{E}}{2} \left(1 + \sqrt{1 + \frac{\hat{\mathcal{R}}_D(h)}{\mathcal{E}}} \right)$$

with

$$\mathcal{E} \leq \frac{4}{\ell} \left(\mathrm{VCdim}(H) \left[\ln(2\ell / \mathrm{VCdim}(H)) + 1 \right] - \ln(\delta/4) \right) .$$

Note that this bound is independent of the distribution ν.

Chapter 2

Support Vector Machines

Support vector machines (SVM) are state-of-the-art learning machines, introduced by Boser et al. [1992] and Cortes and Vapnik [1995], see also the textbooks by Vapnik [1998], Schölkopf and Smola [2002], and Shawe-Taylor and Cristianini [2004]. For their success and their learning theoretical guarantees they have attracted a lot of interests.

All types of support vector machines are based on linear functions on Hilbert spaces, where the feature Hilbert space corresponding to a particular machine is defined in terms of a kernel function. Kernel methods are a flexible and powerful mechanism to introduce non-linearity into linear methods from learning and statistics. This procedure is often called "kernelization". A kernel corresponds to a Riemannian metric on the input space X, making its geometry independent of its actual embedding. This makes linear methods implicitly applicable even to spaces without canonical vector space embedding like for example strings of words or genetic data.

A second key element of support vector machines is their control of the solution complexity via a regularization term. Maybe even more importantly, the VC dimension based capacity control overcomes the traditional "curse of dimensionality" as the solution complexity becomes independent of the data embedding.

Another important property of SVMs is that their hypothesis selection poses a convex optimization problem. Therefore, once the regularization is fixed, the learning algorithm can be sure to return a globally best hypothesis w.r.t. the optimization objective. This situation is highly satisfactory compared to other approaches and simplifies the analysis of properties of the support vector machine, which are mainly properties of the optimal hypothesis.

Altogether these advantages allow for a relatively simple analysis of learning theoretical properties of SVMs. In particular, upper bounds on the generalization error have been derived.

2.1 Kernels

The maybe most interesting point of view on (positive semi-definite) kernels is that they provide a standard and uniform way to incorporate non-linear computations into a linear method, allowing for greatly increased flexibility. In this context a linear method operates on data situated in a (usually but not necessarily finite dimensional) real vector space endowed with an inner product. It uses only linear features of these data, that is, scalars obtained by applying affine linear functions to the input. Of course, these maps may depend on the training data.

Linear methods are often geometrically motivated and have a natural interpretation.

In many cases they are simple enough for a theoretical analysis. However, such methods suffer from a rather restricted class of potential solutions, as many algorithms return linear functions (or functions thereof). Furthermore, these methods can only be applied to vector valued data representations. Kernels offer a convenient and powerful way to overcome these limitations.

In the following, we will restrict ourselves to algorithms where the only operation allowed to compare input vectors is the inner product. From Riesz's theorem it follows that this is equivalent to the application of linear maps constructed from linear combinations of data points. Many classical algorithms like linear discriminant analysis, linear regression, mean classifiers and nearest neighbor classifiers (see section 1.2) fulfill this constraint.

Definition 2.1. *A function $k : X \times X \to \mathbb{R}$ is called a (Mercer) kernel function on the set X if it fulfills the following properties:*

1. *$k(x_1, x_2) = k(x_2, x_1)$ for all $x_1, x_2 \in X$.*
2. *For all $n \in \mathbb{N}$ and for all points $(x_1, \ldots, x_n) \in X^n$ the kernel Gram matrix $K \in \mathbb{R}^{n \times n}$ defined as $K_{ij} = k(x_i, x_j)$ for $i, j \in \{1, \ldots, n\}$ is positive semi-definite.*

For a kernel function k, Mercer's Theorem [Mercer, 1909] ensures the existence of a Hilbert space $\mathcal{H}$ and a feature map $\Phi : X \to \mathcal{H}$ such that the kernel computes the inner product of the features for all pairs of inputs: $k(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$. It is important to note that neither the Hilbert space $\mathcal{H}$ nor the embedding Φ are unique.

For example, if the input space X has the structure of a real manifold (usually open in some $\mathbb{R}^n$) and the kernel is a smooth function, the image $\Phi(X) \subset \mathcal{H}$ is a Riemannian manifold whose Riemannian metric can be computed in terms of second derivatives of the kernel function [Burges, 1998]. Although every Riemannian manifold is embeddable, this embedding is far from unique.

Now the following proceeding is called the kernel trick. It is also referred to as the *kernelization* of a linear algorithm: Given a linear method formulated in terms of inner products of the data and a kernel function k, replace all inner products by the kernel function. With Mercer's Theorem this is equivalent to the application of the feature map Φ first and the linear algorithm afterwards.

A natural question asks for the advantage of using a kernel method instead of a feature map and a linear method afterwards. From a purely mathematical point of view there is no advantage because the algorithms are equivalent (which was the aim of the construction). However, the computational point of view makes the decisive difference. We often consider kernel functions with very high or even infinite dimensional feature spaces. In this case a direct computation of the feature map is prohibitive or even impossible in a computer.

Another point of view on kernel methods focuses on the induced distance measure. As the input space is implicitly embedded into a Hilbert space, a kernel function defines a metric on this space. It is intuitively clear that the choice of a problem specific metric can improve the performance of learning machines. For example, in a classification task we may wish to choose a metric that clusters classes together, see for example Section 9.1.6 for a data driven approach.

2.1.1 Computations with Kernels

Kernels compute the inner product in a feature space $\mathcal{H}$ which is a bilinear function on $\mathcal{H}$. This bilinearity is regularly exploited in calculations involving linear combinations of training examples in feature space. Note that the image $\Phi(X)$ of the input space X can be an embedded Riemannian manifold in feature space, but usually does not have the structure of a vector space. Thus, linear combinations of feature vectors which stem

from X need to be represented explicitly as sums, as in general they can not be written in the form $\Phi(x)$ for some $x \in X$. For two vectors

$$v = \sum_{i=1}^{n} \alpha_i \Phi(x_i) \quad \text{and} \quad w = \sum_{j=1}^{m} \beta_j \Phi(x'_j)$$

in the span of $\Phi(X)$ we can compute the inner product

$$\begin{aligned}\langle v, w \rangle &= \Big\langle \sum_{i=1}^{n} \alpha_i \Phi(x_i), \sum_{j=1}^{m} \beta_j \Phi(x'_j) \Big\rangle \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j \langle \Phi(x_i), \Phi(x'_j) \rangle \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, x'_j)\end{aligned}$$

in terms of the kernel function. This simplifies to $\langle v, \Phi(x) \rangle = \sum_{i=1}^{n} \alpha_i k(x_i, x)$ if we multiply a linear combination v with the image of $x \in X$.

An important application is the computation of the norm or the squared norm of a vector $\|v\|^2 = \langle v, v \rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j)$. For the squared norm $\|\Phi(x)\|^2 = k(x, x)$ of a point in the image of the input space we simply get the kernel applied to the same element twice.

2.1.2 Reproducing Kernel Hilbert Spaces

As mentioned above, the embedding $\Phi : X \to \mathcal{H}$ is far from unique. In applications this is not a concern because all embeddings compute the same value for inner products of images of the inputs. However, for theoretical analysis it can be advantageous to construct a canonical feature space. This feature space should be minimal w.r.t. the subspace order, that is, no proper subspace should be a feature Hilbert space for the same input space X and kernel k. For a particular embedding to be defined below we will choose the smallest possible feature space as our canonical feature space.

Definition 2.2 (Reproducing kernel Hilbert space)**.** *Let X be a topological space and $k : X \times X \to \mathbb{R}$ a continuous kernel function. The set $B := \{k(x, \cdot) : X \to \mathbb{R}, x' \mapsto k(x, x') \,|\, x \in X\}$ consists of continuous functions on X, and thus its span in the real vector space $C(X)$ of continuous functions is a real vector space again. It is endowed with the canonical inner product $\langle \sum_{i=1}^{n} k(x_i, \cdot), \sum_{j=1}^{m} k(x'_j, \cdot) \rangle := \sum_{i=1}^{n} \sum_{j=1}^{m} k(x_i, x'_j)$. The completion of this inner product space of continuous functions is called reproducing kernel Hilbert space (RKHS) for k on X.*

For any $x \in X$ and any $f \in \mathcal{H}$ the property $\langle k(x, \cdot), f \rangle = f(x)$ follows directly from the above definition. The RKHS is named after this so-called reproducing property. The following lemma shows that the RKHS can be understood as the minimal (and therefore canonical) feature space corresponding to a kernel.

Lemma 2.3. *Consider a fixed input space X and a kernel function k as in Definition 2.2. Let $\mathcal{H}$ denote the RKHS with feature map $\Phi : X \to \mathcal{H}$ and let $\tilde{\mathcal{H}}$ be any feature Hilbert space with corresponding feature map $\tilde{\Phi} : X \to \tilde{\mathcal{H}}$ fulfilling $\langle \tilde{\Phi}(x), \tilde{\Phi}(x') \rangle_{\tilde{\mathcal{H}}} = k(x, x')$ for all $x, x' \in X$. Then $\mathcal{H}$ is isomorphic to a subspace of $\tilde{\mathcal{H}}$.*

Proof. The lemma can be proven by constructing a canonical homomorphism $A : \mathcal{H} \to \tilde{\mathcal{H}}$ and by proving its injectivity.

Consider two points $a, b \in X$ with $\tilde{\Phi}(a) = \tilde{\Phi}(b)$. Then we have

$$\begin{aligned}
&\tilde{\Phi}(a) = \tilde{\Phi}(b) \\
\Rightarrow\ &\langle \tilde{\Phi}(a), \tilde{\Phi}(x) \rangle_{\tilde{\mathcal{H}}} = \langle \tilde{\Phi}(b), \tilde{\Phi}(x) \rangle_{\tilde{\mathcal{H}}} && \forall\, x \in X \\
\Rightarrow\ &k(a, x) = k(b, x) && \forall\, x \in X \\
\Rightarrow\ &k(a, \cdot) = k(b, \cdot) \\
\Rightarrow\ &\Phi(a) = \Phi(b)\ ,
\end{aligned}$$

which shows that the map $A = \tilde{\Phi} \circ \Phi^{-1}$ is well-defined. By construction it respects the inner products (that is, $\langle v, w \rangle_{\mathcal{H}} = \langle A(v), A(w) \rangle_{\tilde{\mathcal{H}}}$). Then its linearity follows from

$$\langle A(\lambda v), A(u) \rangle_{\tilde{\mathcal{H}}} = \langle \lambda v, u \rangle_{\mathcal{H}} = \lambda \langle v, u \rangle_{\mathcal{H}} = \lambda \langle A(v), A(u) \rangle_{\tilde{\mathcal{H}}} = \langle \lambda A(v), A(u) \rangle_{\tilde{\mathcal{H}}}$$

and

$$\begin{aligned}
&\langle A(v + w), A(u) \rangle_{\tilde{\mathcal{H}}} = \langle v + w, u \rangle_{\mathcal{H}} = \langle v, u \rangle_{\mathcal{H}} + \langle w, u \rangle_{\mathcal{H}} \\
=&\langle A(v), A(u) \rangle_{\tilde{\mathcal{H}}} + \langle A(w), A(u) \rangle_{\tilde{\mathcal{H}}} = \langle A(v) + A(w), A(u) \rangle_{\tilde{\mathcal{H}}}
\end{aligned}$$

for all $u, v, w \in \mathcal{H}$ and $\lambda \in \mathbb{R}$.

We can write any $u \in \ker(A) \subset \mathcal{H}$ in the form $u = \sum_{i \in I} \lambda_i \Phi(x_i)$ with countable index set I, such that we have $A(u) = \sum_{i \in I} \lambda_i \tilde{\Phi}(x_i) = 0$. For all $x \in X$ it follows

$$\langle u, \Phi(x) \rangle_{\mathcal{H}} = \sum_{i \in I} \lambda_i k(x_i, x) = \sum_{i \in I} \lambda_i \langle \tilde{\Phi}(x_i), \tilde{\Phi}(x) \rangle_{\tilde{\mathcal{H}}} = \langle A(u), \tilde{\Phi}(x) \rangle_{\tilde{\mathcal{H}}} = 0$$

and thus $u = 0$, which shows the injectivity of A. □

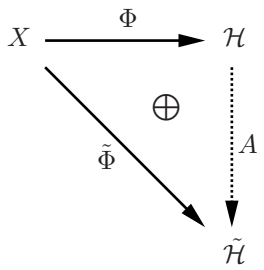

Figure 2.1: The homomorphism $A : \mathcal{H} \to \tilde{\mathcal{H}}$.

As illustrated in Figure 2.1, the property of the pair $(\mathcal{H}, \Phi)$ to be feature space and feature map for (X, k) is a universal property for the RKHS. Therefore the RKHS can be defined, up to isomorphy, as the minimal feature space for (X, k).

2.1.3 Common Kernel Families

Whenever the input space X is (an open subset of) a finite dimensional real vector space and the feature map is a smooth embedding, the image $X \cong \Phi(X) \subset \mathcal{H}$ is a Riemannian manifold. The Riemannian metric structure is induced by the kernel function [Burges, 1998]. Because every Riemannian manifold is embeddable it is clear that there are at least as many kernel functions as there are Riemannian metrics on X. This result gives an insight into the amount of flexibility we get from the choice of the kernel function.

In practice we want to profit from the kernel trick. From a practical as well as from a computational complexity point of view this means that the kernel function should be cheap to compute. This requirement limits the number of kernel functions considered in implementations. On the other hand the kernel function should induce a distance measure that makes learning easy. It is often possible to roughly deduce the form of a possible kernel function for a given task using expert knowledge. Then one first picks a parameterized family of kernel functions and determines good parameter values in a second step.

Common general purpose kernel functions, operating on $X \subset \mathbb{R}^n$, include the following: The linear kernel $k(x, x') = \langle x, x' \rangle = x^T x'$ simply computes the (standard) inner product on the input space. This is a useful concept to apply a kernel algorithm in its pure form directly to the input space. The canonical feature map is simply the identity (although any orthogonal transformation of X would do the job).

As a first non-trivial example we consider the polynomial kernel $k(x, x') = (\langle x, x' \rangle + \theta)^d$ with degree $d \in \mathbb{N}$ and offset $\theta \in \mathbb{R}$. This kernel is a straight forward generalization of the linear kernel which is obtained for $d = 1$ and $\theta = 0$. Its feature space is the space of polynomials of degree d over the input space X, with the remarkable property that the kernel function takes only $\mathcal{O}(\dim X)$ computations, while the dimension of the feature space and therefore the computation of the explicit feature map scales like $\mathcal{O}((\dim X)^d)$. Thus, algorithms based on the polynomial kernel can save a lot of time by exploiting the kernel trick.

The most frequently used kernel is the radial Gaussian kernel

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) = \exp(-\gamma \|x - x'\|^2) \tag{2.1}$$

with width parameter σ or concentration parameter γ. It corresponds to an infinite dimensional feature space, which precludes a direct representation of the feature map in coordinates. Nevertheless the radial Gaussian kernel can be computed in $\mathcal{O}(\dim X)$ operations. A standard extension is the general Gaussian kernel

$$k(x, x') = \exp\left(-\frac{1}{2}(x - x')^T M (x - x')\right) \tag{2.2}$$

with positive definite symmetric parameter matrix $M \in \mathbb{R}^{n \times n}$. This kernel is equivalent to first transforming the input space with a matrix[1] $\sqrt{M}$ and then applying the radial Gaussian kernel with parameter $\gamma = 1$. We may restrict M to diagonal matrices, resulting in the so-called diagonal Gaussian kernel for feature scaling. This kernel computes the radial Gaussian kernel with $\gamma = 1$ on the input space scaled coordinate-wise by the square roots of the diagonal entries of M. For a typical data representation with one scalar feature in each coordinate this proceeding amounts to the application of the radial Gaussian kernel to scaled features. This way we hope to increase the importance of discriminative features, while the influence of less informative features can be reduced.

There are some standard ways to obtain kernels from other kernels. Let k_1 and k_2 be two kernels on X with corresponding feature spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, and feature maps $\Phi_1 : X \to \mathcal{H}_1$ and $\Phi_2 : X \to \mathcal{H}_2$.

For non-negative λ_1, λ_2 the linear combination $k(x, x') = \lambda_1 k_1(x, x') + \lambda_2 k_2(x, x')$ is a kernel again which computes the inner product in the (scaled) direct product $\mathcal{H}_1 \oplus \mathcal{H}_2$ with feature map $\Phi(x) = \sqrt{\lambda_1}\Phi_1(x) + \sqrt{\lambda_2}\Phi_2(x)$.

The product kernel $k(x, x') = k_1(x, x') \cdot k_2(x, x')$ computes the inner product on the feature space of bilinear functions $\mathcal{H}_1 \times \mathcal{H}_2 \to \mathbb{R}$.

[1] N is a square root of M if $N^2 = M$. The square root exists for symmetric positive definite matrices but is not unique.

The normalized kernel

$$k(x, x') = \frac{k_1(x, x')}{\sqrt{k_1(x, x)k_1(x', x')}}$$

is a kernel again with the property $\|\Phi(x)\|^2 = k(x, x) = 1$. That is, the input space is mapped to the unit sphere in $\mathcal{H}_1$. For example, the Gaussian kernel is already normalized.

For a more complete overview of how to construct kernels on all kinds of (structured) spaces or data structures like strings and for other ways to obtain kernels from other kernels refer to the text books by Schölkopf and Smola [2002] or Shawe-Taylor and Cristianini [2004].

2.1.4 Universal Kernels

Universal kernels were introduced by Steinwart [2001]. They are an important concept for the analysis of kernelized versions of algorithms outputting linear hypotheses. In particular, the question under which conditions the standard support vector machine classifier as proposed by Cortes and Vapnik [1995] is universally consistent highly depends on the universality of the kernel function.

Definition 2.4 (universal kernel, see Definition 4 by Steinwart [2001])**.** *A continuous kernel $k : X \times X \to \mathbb{R}$ on a compact metric space X is called universal if the space of all functions induced by k is dense in $C(X)$, i.e., if for every function $f \in C(X)$ and every $\varepsilon > 0$ there exists a function $g(x) = \sum_{n=1}^{\ell} \lambda_n k(x_n, x)$ induced by k with $\|f - g\|_\infty \leq \varepsilon$.*

This property ensures a sufficiently rich feature space to approximate any data generating distribution ν arbitrarily well. It is clear that the RKHS of any universal kernel is infinite dimensional. Steinwart has shown that the corresponding feature maps are injective.

Among the standard kernels introduced above only the Gaussian kernel is universal [Steinwart, 2001].

2.2 Large Margin Classifiers in Kernel Feature Space

The most widespread types of support vector machines are binary classifiers. They realize a classification with large margin in a kernel induced feature space. Like the linear maximum margin classifier (see Section 1.2.2), all these machines are defined through optimization problems. We will first introduce the concepts underlying the different types of machines and then turn to their dual problems, allowing for a more standard and efficient solution.

2.2.1 Hard Margin Support Vector Machine

The simplest type of support vector machine is the maximum margin classifier in a kernel induced feature space. This maximum margin classifier with kernel trick applied is called hard margin support vector machine. Let $k : X \times X \to \mathbb{R}$ denote the positive semi-definite kernel function and let $\Phi : X \to \mathcal{H}$ be a corresponding feature map fulfilling $\langle \Phi(x_1), \Phi(x_2) \rangle_{\mathcal{H}} = k(x_1, x_2)$. Let us first assume that the transformed dataset $((\Phi(x_1), y_1), \ldots, (\Phi(x_\ell), y_\ell)) \in (\mathcal{H} \times \{\pm 1\})^\ell$ is linearly separable in $\mathcal{H}$.

As discussed in Section 1.2.2 we can obtain the large margin classifier

$$h(x) = \text{sign}(\langle w, \Phi(x) \rangle + b)$$

as the solution of the problem

$$\begin{aligned} \underset{(w,b)}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y_n(\langle w, \Phi(x_n)\rangle + b) \geq 1 \text{ for all } n \in \{1, \ldots, \ell\} \ . \end{aligned}$$

This is a convex quadratic optimization problem with linear constraints. However, the search space $\mathcal{H} \times \mathbb{R}$ for (w, b) may be infinite dimensional.

2.2.2 Regularized Large Margin Classification

The most common type of support vector machine is the regularized large margin classifier in a kernel-induced feature space, in particular the so-called 1-norm soft margin support vector machine. This construction has a two-fold purpose. First, it allows us to regularize the training of the machine in order to obtain a sufficiently simple hypothesis. Second, the constraint of linear separability of the dataset is dropped. Both points are closely connected, as we need to allow for training errors in order to reduce the complexity of the hard margin hypothesis.

To apply regularization we need two new concepts. The first one is a measure of complexity of the hypothesis $h(x) = \text{sign}(\langle w, \Phi(x)\rangle + b)$, which will be $\frac{1}{2}\|w\|^2$. As the functional margin of the hard margin SVM is fixed to one, a large norm of w corresponds to a small geometric margin $\rho = 1/\|w\|$ and therefore to an undesired hypothesis. Another point of view on this concept is that the function $\langle w, \Phi(x)\rangle + b$ is smooth on the input space for small values of $\|w\|$ and gets rougher (in terms of larger derivatives) for larger values of $\|w\|$. A better or at least more formal argument to use $\frac{1}{2}\|w\|^2$ as a measure of complexity is its connection to the VC-dimension. According to Theorem 10.3 by Vapnik [1998] the set of hypotheses obtained from linear functions with $\|w\| \leq A$ has VC-dimension bounded by $\lfloor D^2 A^2 \rfloor + 1$ on the ball with radius D around the origin.

The second ingredient is a measure of training error, that is, a loss function $L : X \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{\geq 0}$. Here the second component is the binary training label, while we feed $\langle w, \Phi(x)\rangle + b$ directly into the third component, without applying the sign function first. For the training example (x_n, y_n) we introduce the non-negative slack variable $\xi_n = \max\{0, 1 - y_n(\langle w, \Phi(x_n)\rangle + b)\}$ measuring the amount of functional margin violation and use this quantity as our loss function

$$L(x, y, \langle w, \Phi(x)\rangle + b) = \max\{0, 1 - y(\langle w, \Phi(x)\rangle + b)\} \ ,$$

commonly referred to as the Hinge loss.

Now we have two objectives $\frac{1}{2}\|w\|^2$ and $\sum_{n=1}^{\ell} \xi_n$ to minimize, quantifying hypothesis complexity and training error, respectively. As discussed in Section 1.3 we introduce a regularization parameter $C > 0$ to adjust the trade-off between empirical risk minimization and complexity control, and obtain the primal optimization problem

$$\begin{aligned} \underset{(w,b,\xi)}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{\ell} \xi_n \\ \text{s.t.} \quad & y_n(\langle w, \Phi(x_n)\rangle + b) \geq 1 - \xi_n \text{ for all } n \in \{1, \ldots, \ell\} \\ \text{and} \quad & \xi_n \geq 0 \text{ for all } n \in \{1, \ldots, \ell\} \ . \end{aligned} \tag{2.3}$$

A variant of the soft margin SVM uses the two-norm of the slack vector $\xi = (\xi_1, \ldots, \xi_\ell)^T$, corresponding to the squared Hinge loss, to measure the empirical error. Then the primal objective is defined as

$$\frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{n=1}^{\ell} \xi_n^2 \tag{2.4}$$

which is subject to minimization under the same constraints as above. By additionally dropping the non-negativity constraints for ξ_n we obtain the so-called least squares SVM.

2.2.3 Dual Problems

In the following we show how the dual problems corresponding to the primal problems defined above can be obtained from the Lagrange mechanism for the solution of constrained optimization problems. For more details refer to the books by Vapnik [1998] and Schölkopf and Smola [2002].

We start with the hard margin support vector machine. To compensate for the inequality constraints we introduce non-negative Lagrange multipliers α_n, $n \in \{1, \ldots, \ell\}$, which are summarized in the coefficient vector $\alpha = (\alpha_1, \ldots, \alpha_\ell)^T$. The corresponding Lagrangian becomes

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 + \sum_{n=1}^{\ell} \alpha_n (1 - y_n(\langle w, \Phi(x_n)\rangle + b)) \ .$$

We obtain the dual problem

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & f(\alpha) = \inf\{\mathcal{L}(w, b, \alpha) \mid w \in \mathcal{H}, b \in \mathbb{R}\} \\ \text{s.t.} \quad & \alpha_n \geq 0 \text{ for all } n \in \{1, \ldots, \ell\} \end{aligned}$$

and observe that we have

$$\begin{aligned} \frac{\partial \mathcal{L}(w, b, \alpha)}{\partial w} &= w - \sum_{n=1}^{\ell} y_n \alpha_n \Phi(x_n) = 0 \qquad &&\Rightarrow\ w = \sum_{n=1}^{\ell} y_n \alpha_n \Phi(x_n) \\ \frac{\partial \mathcal{L}(w, b, \alpha)}{\partial b} &= -\sum_{n=1}^{\ell} y_n \alpha_n = 0 \qquad &&\Rightarrow\ \sum_{n=1}^{\ell} y_n \alpha_n = y^T \alpha = 0 \end{aligned}$$

in the minimum. Resubstituting these results into the Lagrangian gives

$$\begin{aligned} f(\alpha) &= \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \Phi(x_j), \Phi(x_i)\rangle \\ &\quad + \sum_{i=1}^{\ell} \alpha_i \Big(1 - y_i \big(\langle \sum_{j=1}^{\ell} y_j \alpha_j \Phi(x_j), \Phi(x_i)\rangle + b\big)\Big) \\ &= -\frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j k(x_i, x_j) + \sum_{i=1}^{\ell} \alpha_i - b \cdot \underbrace{\sum_{i=1}^{\ell} y_i \alpha_i}_{=0} \\ &= -\frac{1}{2} \alpha^T \operatorname{diag}(y) K \operatorname{diag}(y) \alpha + \mathbf{1}^T \alpha \ . \end{aligned}$$

Here, y is the vector of training labels, $\operatorname{diag}(y)$ is the diagonal $\ell \times \ell$ matrix built from y, K is the kernel Gram matrix $K_{ij} = k(x_i, x_j)$ and $\mathbf{1} = (1, \ldots, 1)^T$. For a shorter notation we introduce the quadratic positive semi-definite matrix $Q = \operatorname{diag}(y) K \operatorname{diag}(y)$ with $Q_{ij} = y_i y_j k(x_i, x_j)$. Then the dual optimization problem is the quadratic program

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & f(\alpha) = \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T Q \alpha \\ \text{s.t.} \quad & y^T \alpha = 0 \\ \text{and} \quad & \alpha_n \geq 0 \text{ for all } n \in \{1, \ldots, \ell\} \end{aligned}$$

for $\alpha \in \mathbb{R}^\ell$. Thus, the search space of the dual problem is finite dimensional, but grows with the dataset size ℓ.

Let α^* denote a maximizer of this problem. We are originally interested in the primal solution (w^*, b^*). The vector $w^* = \sum_{n=1}^{\ell} y_n \alpha_n^* \Phi(x_n)$ directly results from α^*. Now b^* can be obtained from the KKT (Karush-Kuhn-Tucker) complementarity conditions for the dual problem. In particular we have $\alpha_n^*(1 - y_n(\langle w^*, \Phi(x_n)\rangle + b^*)) = 0$ for all $n \in \{1, \ldots, \ell\}$. Thus, for any $\alpha_i^* \neq 0$ we have $y_i(\langle w^*, \Phi(x_i)\rangle + b^*) = 1 \Leftrightarrow b^* = y_i - \langle w^*, \Phi(x_i)\rangle = y_i - \sum_{n=1}^{\ell} y_n \alpha_n^* k(x_n, x_i)$. Note that, as long as the learning problem in non-trivial, that is, if examples of both classes are given, some of the Lagrange multipliers α_i^* are positive such that b^* can be obtained this way.

The resulting hypothesis

$$\begin{aligned} h(x) &= \operatorname{sign}(\langle w^*, x\rangle + b^*) \\ &= \operatorname{sign}\left(\sum_{n=1}^{\ell} y_n \alpha_n^* k(x_n, x) + b^*\right) \\ &= \operatorname{sign}\left(\sum_{n \in SV} y_n \alpha_n^* k(x_n, x) + b^*\right) \end{aligned}$$

can be computed in terms of the kernel function, too. Due to the constraints $\alpha_n \geq 0$ many of the optimal Lagrange multipliers α_n^* are zero. These components do not enter the hypothesis at all. We call a point x_n with $\alpha_n^* > 0$ a support vector and define the index set $SV = \{n \in \{1, \ldots, \ell\} \mid \alpha_n^* > 0\}$ of support vectors. For a support vector we have $y_n(\langle w^*, \Phi(x_n)\rangle + b^*) = 1$, that is, the functional margin condition is fulfilled exactly. Therefore, the support vectors are said to lie on the margin. All other points are on the correct side of the margin: $y_n(\langle w^*, \Phi(x_n)\rangle + b^*) \geq 1$.

For many problems the hypothesis is built only from a small part $|SV| \ll \ell$ of the training data. This sparsity of the solution is often considered an important property of support vector machines, as the prediction of a single input takes only $\mathcal{O}(|SV|)$ instead of $\mathcal{O}(\ell)$ operations.

Soft margin support vector machines can be treated completely analogously. To respect the additional constraints $\xi_n \geq 0$ we introduce Lagrange multipliers $\beta_n \geq 0$ and, for the 1-norm slack penalty, we get

$$\begin{aligned} \mathcal{L}(w, b, \xi, \alpha, \beta) = \frac{1}{2}\|w\|^2 + C \sum_{i=n}^{\ell} \xi_n \\ + \sum_{i=n}^{\ell} \alpha_n (1 - \xi_n - y_n(\langle w, \Phi(x_n)\rangle + b)) - \sum_{n=1}^{\ell} \beta_n \xi_n \ . \end{aligned}$$

Setting the derivatives w.r.t. the primal variables w, b, and ξ_n to zero results in

$$\begin{aligned} \frac{\partial \mathcal{L}(w, b, \xi, \alpha, \beta)}{\partial w} &= w - \sum_{n=1}^{\ell} y_n \alpha_n \Phi(x_n) = 0 && \Rightarrow\ w = \sum_{n=1}^{\ell} y_n \alpha_n \Phi(x_n) \\ \frac{\partial \mathcal{L}(w, b, \xi, \alpha, \beta)}{\partial b} &= -\sum_{n=1}^{\ell} y_n \alpha_n = 0 && \Rightarrow\ \sum_{n=1}^{\ell} y_n \alpha_n = y^T \alpha = 0 \\ \frac{\partial \mathcal{L}(w, b, \xi, \alpha, \beta)}{\partial \xi_n} &= C - \alpha_n - \beta_n = 0 && \Rightarrow\ \alpha_n \leq C \ . \end{aligned}$$

Like for the hard margin machine we plug these results into the Lagrangian and end up

with the dual problem

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & f(\alpha) = \mathbf{1}^T\alpha - \frac{1}{2}\alpha^T Q\alpha \\ \text{s.t.} \quad & y^T\alpha = 0 \\ \text{and} \quad & 0 \leq \alpha_n \leq C \text{ for all } n \in \{1,\ldots,\ell\} \end{aligned} \tag{2.5}$$

which, up the the upper bound on α_n, coincides with the hard margin problem. Here we have two different types of support vectors: For $0 < \alpha_n < C$ the support vector is called free, while for $\alpha_n = C$ it is bounded.[2] While free support vectors are located exactly on the margin, bounded support vectors violate the margin. For $\xi_n > 1$ we see from the KKT complementarity condition that $y_n(\langle w, \Phi(x_n)\rangle + b) = 1 - \xi_n < 0$, that is, a bounded support vector may even lie on the wrong side of the decision boundary.

The analysis by Steinwart [2003] shows that in general the number of support vectors grows asymptotically linear with the dataset size ℓ. To be more exact the number of support vectors grows at least like $\mathcal{R}^*_\nu \cdot \ell$, where the lion's share of the support vectors is bounded. That is, asymptotically there is a lower bound on the sparseness for noisy problems, such that the resulting machines are not as sparse as we may wish. Nevertheless for many applications the SVM hypotheses use only a small fraction of the training inputs as support vectors.

In the limit $C \to \infty$ we obtain the hard margin solution. In fact, the solutions coincide as soon as C exceeds the value of the largest Lagrange multiplier α^*_n of the hard margin solution.

The optimal offset b can be obtained as $b^* = y_i - \sum_{n=1}^{\ell} y_n\alpha^*_n k(x_n, x_i)$ just like in the hard margin case, but in this case i must index a free support vector in order to ensure equality in the KKT complementarity condition.[3]

With the same mechanism as above we get the dual problem of the 2-norm machine

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & f(\alpha) = \mathbf{1}^T\alpha - \frac{1}{2}\alpha^T(Q + \frac{1}{C}I)\alpha \\ \text{s.t.} \quad & y^T\alpha = 0 \\ \text{and} \quad & \alpha_n \geq 0 \text{ for all } n \in \{1,\ldots,\ell\} \end{aligned}$$

with the identity matrix $I = \operatorname{diag}(\mathbf{1}) \in \mathbb{R}^{\ell\times\ell}$. The solution to this problem is always unique as the quadratic matrix $Q + \frac{1}{C}I$ is strictly positive definite due to the regularization term. In this machine all support vectors are free as there is no upper bound on α_n. At the same time they all violate the margin by $\xi_n = \alpha_n/C$. That is, a support vector with coefficient $\alpha_n > C$ is on the wrong side of the margin. Again, in the limit $C \to \infty$ we reconstruct the hard margin case.

It is interesting that this dual problem differs from the dual problem of the hard margin machine only in the quadratic term. From an optimization point of view we can usually interpret this problem as a hard-margin problem with a slightly different kernel: $\bar{k}(x, x') = k(x, x') + \frac{1}{C}\delta_{xx'}$, where δ denotes the Kronecker symbol. This is in fact a valid kernel, but we have to assume that the training dataset does not contain a point twice for this interpretation to work properly. Further, the original kernel is used in the resulting hypothesis.[4]

[2] This definition may slightly vary. A support vector with $\alpha_n = C$ may also fulfill $\xi_n = 0$. In this case it can make sense to consider the support vector free. For most problems ν this happens with probability zero. Further, the property $\alpha_n = C$ is easier to obtain from the dual solution than the condition $\xi_n > 0$.

[3] It is possible that all support vectors are bounded. In this case we can derive an interval for b, and implementations just pick the center of this interval. For the vast majority of problems this happens only if the regularization parameter C is chosen much too small, leading to severe underfitting, such that this problem rarely plays a role.

[4] This clearly shows that this interpretation is not completely valid. It should be considered a compu-

2.3 Support Vector Machines for Regression

Vapnik [1998] introduced a learning machine that outputs an affine linear hypothesis $h(x) = \langle w, \Phi(x)\rangle + b$ directly suitable for regression. This machine has the most important properties of SVM classifiers, including sparse solutions. To achieve this we need to introduce the ϵ-insensitive loss function $\max\{0, |y - y'| - \epsilon\}$ which vanishes as long as the output does not deviate from the target by more than ϵ, and then increases linearly. Again we measure the hypothesis complexity by $\frac{1}{2}\|w\|^2$. The resulting problem is

$$\underset{(w,b)}{\text{minimize}} \quad \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{\ell} \max\{0, |\langle w, \Phi(x_n)\rangle + b - y_n| - \epsilon\} \ .$$

To get a better grip on the kinks in the loss function we define slack variables ξ_n^+ and ξ_n^-, and arrive at the primal problem

$$\begin{aligned} \underset{(w,b,\xi^+,\xi^-)}{\text{minimize}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{\ell}(\xi_n^- + \xi_n^+) \\ \text{s.t.} \quad & -\epsilon - \xi_n^+ \le \langle w, \Phi(x_n)\rangle + b - y_n \le \epsilon + \xi_n^- \text{ for all } n \in \{1, \ldots, \ell\} \\ \text{and} \quad & \xi_n^+ \ge 0, \ \xi_n^- \ge 0 \text{ for all } n \in \{1, \ldots, \ell\} \end{aligned}$$

of the standard support vector machine for regression. The dual problem with variables (α^+, α^-) is

$$\begin{aligned} \underset{(\alpha^+,\alpha^-)}{\text{maximize}} \quad & f(\alpha^+, \alpha^-) = y^T(\alpha^+ - \alpha^-) - \epsilon \mathbf{1}^T(\alpha^+ + \alpha^-) \\ & \qquad\qquad - \frac{1}{2}(\alpha^+ - \alpha^-)^T K(\alpha^+ - \alpha^-) \\ \text{s.t.} \quad & \mathbf{1}^T(\alpha^+ - \alpha^-) = 0 \\ \text{and} \quad & 0 \le \alpha_n^+ \le C, \ 0 \le \alpha^- \le C \text{ for all } n \in \{1, \ldots, \ell\} \ , \end{aligned}$$

resulting in $w = \sum_{n=1}^{\ell}(\alpha_n^+ - \alpha_n^-)\Phi(x_n)$. From the KKT conditions

$$\begin{aligned} \alpha_n^-(\langle w, \Phi(x_n)\rangle + b - y_n - \epsilon - \xi_n^-) = 0 \\ \alpha_n^+(y_n - \langle w, \Phi(x_n)\rangle - b - \epsilon - \xi_n^+) = 0 \end{aligned}$$

we can compute b from any free variable α^- or α^+ as usual. Sometimes it is more intuitive to introduce the variables $\alpha_n = \alpha_n^+ - \alpha_n^-$ resulting in the problem

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & f(\alpha) = y^T\alpha - \epsilon\sum_{n=1}^{\ell}|\alpha_n| - \frac{1}{2}\alpha^T K\alpha \\ \text{s.t.} \quad & \mathbf{1}^T\alpha = 0 \\ \text{and} \quad & -C \le \alpha_n \le C \text{ for all } n \in \{1, \ldots, \ell\} \end{aligned}$$

where we trade a reduction of the number of variables by a factor of two for a non-differentiable objective function.

tational trick only. However, it the data generating distribution ν has no discrete components the trick works with full probability.

2.4 Universal Consistency of Support Vector Machines

Steinwart was the first to assess conditions for universal consistency of the machines outlined above [Steinwart, 2001, 2002]. A first result of his studies is that in order to achieve universally consistent classification with an affine linear decision boundary the RKHS and therefore the hypothesis space must be rich enough to allow for an arbitrary good approximation of the Bayes optimal hypothesis for any distribution ν. This property is captured by the definition of universal kernels, see Section 2.1.4.

Another ingredient for a universally consistent learner is the correct dependency of the regularization parameter on the number of training examples. Steinwart has proven that for the 1-norm support vector machine a sequence of regularization parameters $(C_\ell)_{\ell\in\mathbb{N}}$ fulfilling $C_\ell = C_0 \cdot \ell^{\beta-1}$ leads to consistency, where β is defined in terms of the covering numbers

$$\mathcal{N}((X,d),\varepsilon) = \inf\left\{ n \in \mathbb{N} \;\middle|\; \exists\, x_1,\ldots,x_n \text{ with } X \subset \bigcup_{i=1}^{n} B_d(x_i,\varepsilon) \right\}$$

of X, d is a metric, and $B_d(x,\varepsilon) = \{x' \in X \,|\, d(x,x') < \varepsilon\}$ is the open ε-ball around x w.r.t. d. This concept is applied to the metric

$$d_k(x,x') = \|\Phi(x) - \Phi(x')\| = \sqrt{k(x,x) - 2k(x,x') + k(x',x')}$$

induced by a kernel function k.

Theorem 2.5 (Theorem 2 by Steinwart [2002])**.** *Let $X \subset \mathbb{R}^n$ be compact and let k be a universal kernel on X with $\mathcal{N}((X,d_k),\varepsilon) \in \mathcal{O}(\varepsilon^{-\alpha})$ for some $\alpha > 0$. Suppose that we have a positive sequence $(C_\ell)_{\ell\in\mathbb{N}}$ with $\ell \cdot C_\ell \to \infty$ and $C_\ell \in \mathcal{O}(\ell^{\beta-1})$ for some $0 < \beta < \frac{1}{\alpha}$. Then for all Borel probability measures ν on $X \times Y$ and all $\varepsilon > 0$ we have*

$$\lim_{\ell\to\infty} \Pr_\ell^\nu(\{D \in (X\times Y)^\ell \,|\, \mathcal{R}_\nu(A_{C_\ell,k}(D)) \le R_\nu^* + \varepsilon\}) = 1$$

where the learner $A_{C,k}$ computes the 1-norm SVM hypothesis with regularization parameter C and kernel k, and $\Pr_\ell^\nu$ is the outer probability measure on ν^ℓ.

This is possibly the most important result in the whole area of support vector machines, as it ensures that these machines do not only work well for small datasets (as we can experience in practice), but that they can learn any problem arbitrarily well. This is a justification to consider support vector machines (with universal kernels) as generally applicable to all possible kinds of supervised learning problems.

Chapter 3

Performance Assessment

This chapter deals with the question how to compare learning algorithms. In the remainder of this thesis several algorithms will be introduced, mainly variants of existing algorithms with a particular focus. Then naturally the question arises how to compare these variants to the standard version or to completely different approaches.

3.1 Exact Analysis and Asymptotic Properties

Sometimes it is helpful to know how different algorithms perform asymptotically, most frequently expressed in $\mathcal{O}$-notation. For example, all variants of the decomposition algorithm for SVM training (see Chapter 4) get along with $\mathcal{O}(\ell)$ working memory plus a large but constant size kernel cache ($\mathcal{O}(1)$), where ℓ is the number of variables in problem (4.1). Then, empirically, the runtime for the solution of the problem up to a fixed accuracy scales roughly like $\mathcal{O}(\ell^{2.1})$ [Joachims, 1998].

In several cases (for example, most sorting algorithms) it is possible to state exact conditions under which one algorithm is superior to another and to derive a detailed mean and worst case analysis of the runtime in terms of the problem size. There are indeed several results stating polynomial runtime of decomposition algorithms [Hush and Scovel, 2003, List and Simon, 2004, 2005, Hush et al., 2006], but the bounds are not applicable to the variants of the algorithm that play a role is this thesis, and some of the bounds are probably quite loose. That is, such an analysis can not reveal the differences in the runtime of the algorithms. For a concrete problem it is completely unclear which properties indicate that a certain algorithm will be faster than another.

3.2 Benchmarks

As indicated above there is little hope to compare the algorithms presented in the forthcoming chapters within a complete analysis. Still there is a need for a comparison, although it will remain incomplete and unsatisfactory from a mathematical point of view. The best we can do is to run the algorithms on a suite of benchmark problems and measure their performance. The usage of a collection of benchmark problems has a two-fold purpose:

- In many cases benchmark problems stem from real-world applications. One may claim that such a suite of benchmark datasets poses an unbiased sample from the (not strictly defined) distribution of real-world problems. That is, the data generating distribution ν is not arbitrary, but is assumed to model typical application relevant problems from natural sciences, engineering, medicine, economics, and many

more areas of applications of machine learning techniques. That is, the limited set of benchmark problem is hoped to be a reasonable representation of typical datasets the algorithm may be confronted with, and the goal of optimization is to be fast or accurate on average in a "typical" application.

- Some benchmark problems are used regularly such that their properties are known quite well. Then it is possible to use a benchmark collection to analyze the strengths and weaknesses of a particular algorithm in comparison to standard methods. This type of comparison looks at different benchmark problems as special application cases, each testing the performance of the algorithm in the presence or absence of typical properties of the problem at hand.

3.3 Runtime Comparison

The algorithms introduced in Part II of this thesis are all tailored to the solution of the same type of quadratic programs. In this context high speed or short runtime, under certain constraints on the working memory consumption, is the most important goal.

3.3.1 Hyperparameter Dependency

Most algorithms compared depend on one or more so-called hyperparameters. These are the regularization parameter and an arbitrary number of parameters of the kernel function. It turns out that the runtime critically depends on the values of these algorithm parameters. Thus, there are mainly two comparison strategies:

- We can compare the runtime of the algorithms for different systematically chosen parameter settings. This way we get a rather complete picture of their performance. In particular we can assess the robustness of the performance w.r.t. the hyperparameters. However, this comes at the price of extremely long runtimes in the experimental evaluation. It is not clear a priori which parameter values lead to good generalization (and therefore are of greatest interest) for a particular problem. But in some regions of the parameter space far from this point the machine training may take longer by several orders of magnitude than for the well-generalizing machines. Then these parameter make up a large fraction of the overall time the empirical comparison takes. If this procedure is repeated for all problems in a suite of benchmark problems or even for a single large or difficult dataset it may simply turn out to be computationally impractical.

- Usually it is considered most relevant that the algorithm is fast for the parameter values leading to well generalizing machines. Thus we can determine reasonably good parameters with a relatively cheap (but maybe not very accurate) method. Then the performance comparison is carried out with these fixed parameters.

3.3.2 Technical Difficulties

In particular the comparison of the runtime of different algorithms turns out to be more complicated than it seems at a first glance. Usually we are interested in the comparison of abstract algorithms, not of concrete implementations. Bacause we have no alternative to simulation results we need to ensure, whenever implementations are compared, that they do not favor one or the other algorithm. Of course, this is hard to guarantee, if not impossible. The best we can do is to implement the algorithms in the very same environment, with as much overlap as possible. For example, when small but critical parts of the SMO algorithm (see Chapter 4) are replaced, we exchange the corresponding

parts of the LIBSVM implementation [Fan et al., 2005] without touching the rest of the code.

Still the performance may be subject to programming skills. But as LIBSVM is regularly claimed to be the fastest publicly available support vector machine implementation and in most cases the implementation of the algorithms is straight forward this danger is minimal.

Another pitfall, resulting from a special property of the SMO algorithm, will be discussed in Section 4.3.9.

3.4 Generalization Performance

In Part III the main performance criterion considered is the generalization error, while speed is only of minor importance. We will see that the performance of some methods strongly depends on random variables like the split of a dataset into training and validation data. As the performance of these methods turns out to have a high variance we need to repeat the simulations many times in order to achieve reliable and statistically significant comparison results.

Most typical benchmark problems are stated by fixed size samples from an unknown data generating distribution, mostly defined by experimental conditions and subject to measurement and labeling errors. All we have is this fixed amount of samples in an electronic representation, and it is not possible to sample more data as needed. Thus we can not arbitrarily repeat experiments with fresh data to reduce the variance of the random measurements. Here, the best approximation we can get results from a simple resampling or bootstrapping technique which amounts to changing the underlying distribution to the empirical distribution of the available data. This can lead to over-optimistic results, for example if we use a look-up table, containing the frequencies of the labels observed in each point, as a learning machine. This primitive machine is consistent for distributions with finite support (as defined by a fixed dataset), but does not generalize to the data generating distribution ν at all.

Therefore it is natural to go for known data generating distributions from which we can cheaply draw as many examples as needed. Of course this means that we can not use "typical" data from applications any more. On the other hand we have the advantage to completely control the data generation process, and we may even be able to calculate the Bayes optimal performance.

Here we will define several distributions used later on. All of these distributions are defined on subsets $X \subset \mathbb{R}^n$ and the Bayes error rates are easily computable. We will use these distributions to sample artificial test problems for different types of method comparisons. The problems are well suited for the application of the Gaussian kernel.

The Chess Board Problem

This problem has the parameters $n \in \mathbb{N}$ and $d \in \mathbb{N}$. Its input space is $X = [0, n)^d$. Inputs $x \in X$ are drawn from the uniform distribution on X and the label is assigned due to the rule $y = (-1)^{\sum_{i=1}^{d} \lfloor x_i \rfloor}$. For $n = 8$ and $d = 2$ we get a chess board as the input space X, where the color of a field indicates the label. The number of chess board fields is n^d which can easily grow very large. Of course, there should be significantly more training examples than fields to give a learning machine a chance to learn this structure. Therefore we use $n = 4$ and $d = 2$ as standard settings with only 16 fields to make small sample sizes of several hundred examples meaningful (see Figure 3.1).

The Bayes error of this problem is zero, as there is a deterministic rule to compute the label from the input. Therefore it is justified to train hard margin SVMs on this problem.

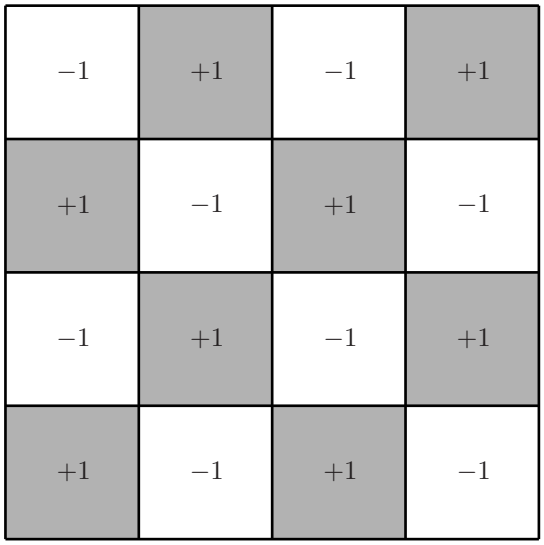

Figure 3.1: Labeling rule of the default chess board distribution on the input space $X = [0, 4) \times [0, 4)$.

However, a moderately regularized machine performs slightly better. Nevertheless all values of the regularization parameter corresponding to few or no regularization have nearly the same effect such that the performance is not very sensitive w.r.t. this parameter. It is more challenging to find good kernel parameters.

To make the adaptation of the regularization parameter more important we add some noise to the basic chess board problem. For this purpose we introduce a parameter $p \in [0, 1)$. The input x is drawn as usual from the chess board distribution. With probability p a random label (uniformly distributed in $\{-1, +1\}$ is assigned, while the default label from the chess board problem is assigned otherwise. The resulting problem has a Bayes error of $p/2$. It is referred to as the noisy chess board problem.

The Sparse Coordinate Problem

The chess board problem has the appeal that datasets generated from this distribution result in quite complicated quadratic programs for the SMO algorithm. However, because the dimension d of the chess board needs to be small in order to keep the number of fields tractable, this distribution is not well suited for assessing the adaptation of large numbers of hyperparameters. For this purpose we define a different problem with parameters $n, m, k \in \mathbb{N}$. The design of this problem is inspired by the bioinformatics problem introduced in Chapter 13.

The input space of this problem is $X = \{0, 1\}^{2n+m} \subset \mathbb{R}^{2n+m}$. When drawing an example from this distribution we first define the set $S = \{1, \ldots, 2n + m\}$. We draw the label $y \in Y = \{-1, +1\}$ and a number $i \in \{1, \ldots, n\}$. For $y = +1$ we set $S \leftarrow S \setminus \{i\}$, otherwise we set $S \leftarrow S \setminus \{n+i\}$. Then we draw $(k-1)$-times from the uniform distribution on S and immediately remove the drawn element. Finally we set $x_i = 0$ for all $i \in S$ and $x_i = 1$ otherwise. That is, the support of this distribution consists of the binary $(2n+m)$-dimensional vectors with exactly k 1s, where at least one 1 is among the first $2n$ coordinates. Most positive examples will have more 1-entries among the first n coordinates than on average, while the frequency of 1-entries for the negative class is increased in the coordinates $n + 1, \ldots, 2n$. Thus, the first $2n$ coordinates are highly predictive, while all

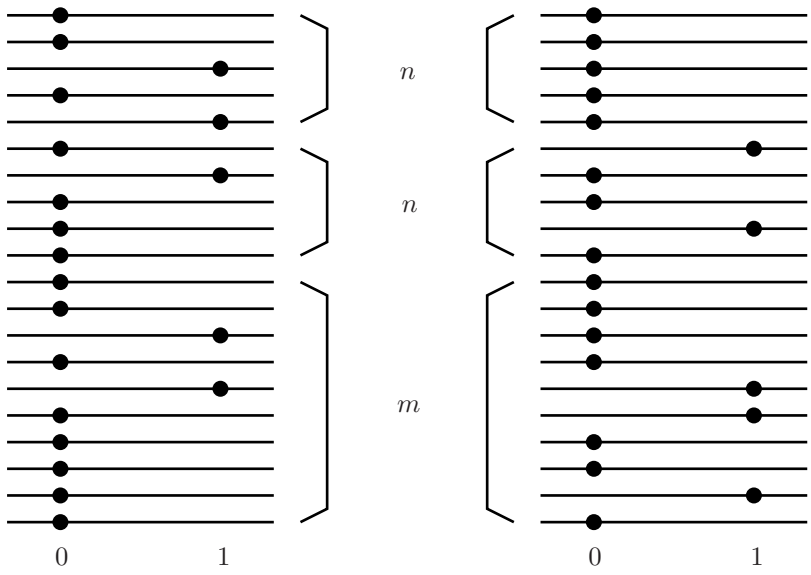

Figure 3.2: Examples of the sparse coordinate problem with parameters $(n, m, k) = (5, 10, 5)$. The example on the left belongs to the positive class with high probability, as it has two 1-entries among the first $n = 5$ coordinates and only one 1-entry among the second $n = 5$ coordinates. In contrast, the example on the right doubtlessly belongs to the negative class. Note that the last m coordinates do not encode any discriminative information at all.

other coordinates contain no useful information, see Figure 3.2.

We will use the configurations $(n, m, k) = (3, 14, 3)$ and $(n, m, k) = (3, 14, 5)$, both making up a $2 \cdot 3 + 14 = 20$-dimensional input space. The computation of the Bayes error rates is a combinatorial problem as the support of the distributions is finite. In the first case the Bayes error is $144/1026 \approx 14.04\%$, and in the second case it increases to $5262/23256 \approx 22.63\%$. To assess the ability of model selection algorithms to obtain good parameters from noisy data we can use the diagonal Gaussian feature scaling kernel with this distribution. For these problems, this kernel has 20 free parameter which can be adapted, and the task is of course to choose a wide width in the first $2 \cdot n$ coordinates and a peaked shape in the remaining m uninformative coordinates.

Part II

Support Vector Machine Training

Chapter 4

Decomposition and Sequential Minimal Optimization

The second part of this thesis deals with algorithms for the iterative solution of the quadratic programs that correspond to the dual problems of the support vector machines introduced in Chapter 2. We start with the decomposition algorithm and the specialized sequential minimal optimization (SMO) algorithm. In the remaining Chapters 5 to 8 of this part we present our own work on this topic:

As a step towards the analysis of the convergence speed of the SMO algorithm we prove a progress rate for certain types of iterations. Then we present two independent improvements of the basic algorithm, namely a working set selection policy and a technique called planning-ahead. Both improvements are accompanied by a detailed convergence analysis and simulation results showing the corresponding improvements. Finally we apply a second order working set selection policy to an SVM online learning algorithm.

4.1 The Quadratic Program of SVM Training

The dual problems of all support vector machine variants introduced in Chapter 2 share the structure

$$\begin{aligned} \text{maximize} \quad & f(\alpha) = y^T\alpha - \frac{1}{2}\alpha^T K\alpha && (4.1)\\ \text{s.t.} \quad & \mathbf{1}^T\alpha = 0 && \text{(equality constraint)}\\ \text{and} \quad & L_n \le \alpha_n \le U_n \quad \forall\ 1 \le n \le \ell && \text{(box constraint)} \end{aligned}$$

with positive semi definite matrix K. For example, in case of the 1-norm soft margin classifier we have $K_{ij} = k(x_i, x_j)$, such that we re-interpret the variables α_n because the axes corresponding to negative class examples are switched (we write α_n instead of $y_n\alpha_n$ in the original problem). The vector y coincides with the vector composed of the labels, and the lower and upper bounds are given as $L_n = \min\{0,\ y_n{\cdot}C\}$ and $U_n = \max\{0,\ y_n{\cdot}C\}$. Analogously, the other dual problems of the hard margin SVM, the 2-norm soft margin SVM, and the SVM for regression can easily be transformed into this form. Further problems with this structure include the dual problem of the one-class SVM [Schölkopf et al., 2001] for the estimation of the support of a distribution and the computation of the radius of the smallest sphere containing all training examples (see Vapnik, 1998).

The convex[1] quadratic programming Problem (4.1) is special in that there is exactly

[1] Of course, the corresponding problem to minimize $-f(\alpha)$ is convex. We stick to maximization because this form stems from dual optimization problems.

one equality constraint and the inequality constraints are aligned to the coordinate system. Usually such problems are solved with standard software from an optimization toolbox. This turns out to be difficult, especially for large datasets when the matrix $K \in \mathbb{R}^{\ell \times \ell}$ does not fit into the available working memory and its inversion becomes numerically challenging and computationally prohibitive. This is the main motivation for the investigation of algorithms specifically tailored to this type of problem.

In the following we fix some notations concerning the quadratic programming problem (4.1). Let $\mathcal{R} \subset \mathbb{R}^\ell$ denote its feasible region, that is, the compact set of points that fulfill all constraints. Geometrically speaking this is the intersection of a high dimensional box and a hyperplane. It can be written as a finite intersection of half-spaces, a so called polytope.

Let $\mathcal{R}^* \subset \mathcal{R}$ be the compact and convex set of optimal solutions, while f^* denotes the optimum, defined as the value of f on $\mathcal{R}^*$. Like the feasible region, the set $\mathcal{R}^*$ form a polytope. All optimal $\alpha, \alpha' \in \mathcal{R}^*$ differ only inside the kernel of the matrix K, that is, $\alpha - \alpha' \in \ker(K)$. Otherwise we would get the contradiction $f(\frac{1}{2}(\alpha + \alpha')) > f^*$. Therefore the gradients $\nabla f(\alpha) = y - K\alpha = y - K\alpha' = \nabla f(\alpha')$ in all optimal points coincide.

For any feasible search point $\alpha \in \mathcal{R}$, and in particular for optimal points $\alpha^* \in \mathcal{R}^*$, it is often of interest whether a particular component α_n is at the box boundary $\alpha_n \in \{L_n, U_n\}$ or not. We call variables $\alpha_n \in (L_n, U_n)$ free. This naming just ignores the equality constraint, that is, a free variables is free w.r.t. the box constraints.

4.2 The Decomposition Algorithm

For the gradient-based optimization of problem (4.1) it is necessary to compute the gradient $\nabla f(\alpha) = y - K\alpha$ of the objective function in the current search point. In general, this computation takes quadratic time as the whole matrix K needs to be evaluated. The situation is even worse if ℓ is so large (roughly $\ell > 10,000$ on today's standard hardware) that the $\ell \times \ell$ matrix K does not fit into the available working memory. For large scale problems with many thousands or even millions of training examples even large hard disks are insufficient. In this situation a decomposition algorithm can be beneficial as it requires only $\mathcal{O}(\ell)$ time and memory in each iteration.

The basic idea of decomposition is to modify only a small number of variables at a time, the so called working set. In this respect the algorithm closely sticks to the given coordinate system. The size of this set is upper bounded by a small pre-defined constant q. Then the problem is solved (near) optimal on the subspace defined by these variables. The resulting q-dimensional sub-problem has the same structure as problem (4.1), but it easily fits into the available working memory and can be solved with standard tools. This process is repeated, resulting in an iterative approximation of the optimal solution.

Equally importantly, the gradient of the objective function can be computed based on the gradient in the old search point with only $\mathcal{O}(q\ell)$ operations. Let α be the current search point. The next search point takes the form $\alpha + \delta$ where the vector δ is extremely sparse in that only at most q of its components are non-zero. For the gradient we have $\nabla f(\alpha + \delta) = \nabla f(\alpha) - K\delta$. Due to the extreme sparsity of δ the product $K\delta$ takes only $q\ell$ multiplications and additions, that is, time linear in the problem size ℓ.

The decomposition technique explicitly exploits the special structure of problem (4.1). Most importantly, the inequality constraints are nicely aligned to the coordinates such that for the solution of the sub-problem on the working set the box constraints of the inactive variables can safely be ignored. Therefore only $\mathcal{O}(q)$ constraints need to be handled for the solution of the sub-problem.

Due to its iterative structure the algorithm must check a stopping condition to terminate as soon as the solution is sufficiently accurate. Algorithm 4.1 demonstrates the

structure of the decomposition scheme.

Algorithm 4.1: General Decomposition Algorithm

Input: feasible initial point $\alpha^{(0)}$, maximum working set size $q \geq 2$, accuracy $\varepsilon \geq 0$
compute the initial gradient $G^{(0)} \leftarrow \nabla f(\alpha^{(0)}) = y - K\alpha^{(0)}$
set $t \leftarrow 1$
do
  select a working set $B^{(t)} \subset \{1, \ldots, \ell\}$ with $2 \leq |B^{(t)}| \leq q$
  solve the sub-problem induced by $B^{(t)}$ and $\alpha^{(t-1)}$, resulting in $\alpha^{(t)}$
  compute the gradient $G^{(t)}$ in the new search point
  check the stopping condition, depending on ε
  set $t \leftarrow t + 1$
loop

In general the computation of the initial gradient $G^{(0)}$ takes quadratic time ($\mathcal{O}(\ell^2)$ operations). If no additional information are available it makes sense to choose the initial solution to be $\alpha^{(0)} = 0$ resulting in the initial gradient $G^{(0)} = \nabla f(\alpha^{(0)}) = y$. Then no evaluations of the kernel function are needed to compute the initial gradient.

Each of the steps in the iterative loop of the decomposition algorithm is designed to take at most $\mathcal{O}(\ell)$ computations. More complex operations usually slow the entire algorithm down even for medium size datasets, and become intractable for large datasets.

In the notation introduced in Algorithm 4.1 all quantities are named after the iteration in which they are computed. Therefore, the sub-problem depends on the working set with index (t) which was defined in the preceding step, and on the current search point with index $(t-1)$ which still stems from the previous iteration. The idea to apply the decomposition technique to problem (4.1) and therefore to the training of support vector machines goes back to Osuna et al. [1997]. Later this technique was further improved by several authors (most noticeable Joachims, 1998, Platt, 1998, Fan et al., 2005) and soon became the default in SVM training due to its speed and scalability. The book chapter by Bottou and Lin [2007] gives a concise review including the recent development.

The decomposition algorithm works rather different from, for example, standard gradient descent. Nevertheless it is clearly a gradient-based optimization technique, as it makes use of the gradient in all of its steps. Even the second derivatives (which are independent of the search point as the objective function is quadratic) can be used, for examples for the solution of the sub-problems posed by the current search point and the working set, or for the selection of a working set.

In the following we will have a closer look at the single steps of the decomposition algorithm.

4.2.1 Working Set Selection

The selection of a working set $B \subset \{1, \ldots, \ell\}$ of at most q indices of variables α_n, $n \in B$, roughly corresponds to the selection of a search direction for optimization. Here, instead of a single direction, a low dimensional subspace spanned by the unit vectors $e_n = (0, \ldots, 1, \ldots, 0)^T$, $n \in B$, is selected. If we want to keep the current solution feasible we are forced to select at least two variables in each iteration because a single variable is always fixed by the equality constraint of problem (4.1). For a related problem with k equality constraints it is understood that we need to select at least $k + 1$ variables in order to include a feasible direction.[2]

[2]This does not hold for the special case of the ν-SVM introduced by Schölkopf et al. [2000] with two equality constraints. Refer to Chang and Lin [2001b] for how this problem can be solved with working

The decomposition algorithm produces a sequence of feasible points $\alpha^{(t)}$ which depend on the initial solution $\alpha^{(0)}$ and the sequence of working sets $B^{(t)}$. This makes clear that working set selection is critical for the speed of the decomposition algorithm as well as for its convergence properties.

We do not want the working set selection policy to choose a working set such that the current search point is optimal in the corresponding subspace. In contrast, we want to find a subspace with as much room for improvement as possible. In general, the more the solution can be improved through the optimization in the current subspace the faster will the overall algorithm converge. However, even a greedy search is usually prohibitive. Therefore working set selection algorithms need to use further heuristics in order to find good working sets.

The selection of working sets such that some progress can be made ensures that the sequence $f(\alpha^{(t)})$ is strictly monotonically increasing. Of course, this does not guarantee convergence to the optimum. To make the decomposition algorithm terminate for any predefined accuracy of the solution it is an important task to prove its convergence to the optimum for the working set selection policy used.

It has been shown by Lin [2001] that the decomposition algorithm converges to the optimum if the so called maximum violating pair (see equation (4.5)) is contained in each working set. Alternatives are the rate certifying pair approach by Hush and Scovel [2003], Hush et al. [2006] which has been generalized by List and Simon [2005] to arbitrary quadratic programs, or the closely related selection based on a sparse witness of sub-optimality [List and Simon, 2004], where the working set guarantees a certain rate of progress towards the optimum (see Section 6.1.1 for an example). These selections do not only guarantee convergence to the optimum, but even convergence at a guaranteed convergence rate (see Hush and Scovel, 2003, List and Simon, 2004, 2005, Hush et al., 2006).

4.2.2 Solution of the Sub-Problem

The sub-problem given by the current search point $\alpha^{(t-1)}$ and the current working set $B^{(t)}$ is defined as follows:

$$\begin{aligned} \text{maximize} \quad & f(\alpha) = y^T\alpha - \frac{1}{2}\alpha^T K\alpha && (4.2)\\ \text{s.t.} \quad & \sum_{n=1}^{\ell} \alpha_n = 0 && \text{(equality-constraint)}\\ & L_n \le \alpha_n \le U_n \quad \forall\ 1 \le n \le \ell && \text{(box-constraint)}\\ \text{and} \quad & \alpha_n = \alpha_n^{(t-1)} \text{ for all } n \notin B^{(t)}\ . \end{aligned}$$

That is, problem (4.1) is solved as good as possible while only the variables indexed by the working set $B^{(t)}$ are modified. Thus we should rewrite problem (4.2), considering only the α_n, $n \in B^{(t)}$, as variables. Let $\{b_1, \ldots, b_m\} = B^{(t)}$, $m \le q$, be the indices in the current working set. We define

$$\alpha_{B} = \begin{pmatrix} \alpha_{b_1} \\ \vdots \\ \alpha_{b_m} \end{pmatrix} \in \mathbb{R}^m \qquad K_{BB} = \begin{pmatrix} K_{b_1 b_1} & \ldots & K_{b_1 b_m} \\ \vdots & \ddots & \vdots \\ K_{b_m b_1} & \ldots & K_{b_m b_m} \end{pmatrix} \in \mathbb{R}^{m\times m}$$

sets of size 2. This proceeding has been generalized by List [2008].

and fix the constants

$$y_{B} = \begin{pmatrix} y_{b_1} - \sum\limits_{n \notin B^{(t)}} K_{nb_1} \alpha_n^{(t-1)} \\ \vdots \\ y_{b_m} - \sum\limits_{n \notin B^{(t)}} K_{nb_m} \alpha_n^{(t-1)} \end{pmatrix} \in \mathbb{R}^m \qquad \text{and} \qquad z_{B} = - \sum_{n \notin B^{(t)}} \alpha_n^{(t-1)} \ .$$

Then the reduced problem

$$\begin{aligned} \text{maximize} \quad & f_{B}(\alpha_{B}) = y_{B}^T \alpha_{B} - \frac{1}{2} \alpha_{B}^T K_{BB} \alpha_{B} && (4.3) \\ \text{s.t.} \quad & \sum_{n \in B^{(t)}} (\alpha_{B})_n = z_{B} && \text{(equality-constraint)} \\ \text{and} \quad & L_n \leq (\alpha_{B})_n \leq U_n \quad \forall\, n \in B^{(t)} && \text{(box-constraint)} \end{aligned}$$

for $\alpha_{B} \in \mathbb{R}^m$ is equivalent to problem (4.2) (see Joachims, 1998). The value z_{B} can easily be determined in time linear in q, but the computation of y_{B} takes time linear in q and ℓ. We will deal with problem (4.3) under the assumption that we know the gradient vector $\nabla f(\alpha^{(t-1)}) = y - K\alpha^{(t-1)}$. With this information y_{B} can be computed in $\mathcal{O}(q^2)$ operations, which is independent of ℓ.

The above problem involves at most q variables, which is usually a small fixed number. Therefore it can be solved with standard optimization techniques, for example, an interior point algorithm [Boyd and Vandenberghe, 2004]. Again, the number of operations depends on q only and is in particular independent of ℓ. In this respect the solution of the sub-problem is the fastest step in the decomposition loop. The optimal solution α_B^* replaces the variables $\alpha_n^{(t-1)}$, $n \in B^{(t)}$, to build the new search point $\alpha^{(t)}$.

4.2.3 Update of the Gradient

The gradient $\nabla f(\alpha^{(t-1)}) = y - K\alpha^{(t-1)}$ is already known, but we are interested in the new gradient $\nabla f(\alpha^{(t)}) = y - K\alpha^{(t)}$. As already stated above, this vector can be computed as an update in linear time. As the difference $\delta = \alpha^{(t)} - \alpha^{(t-1)}$ is a sparse vector with non-zero entries only in the components corresponding to the working set, we can write the new gradient as $\nabla f(\alpha^{(t)}) = \nabla f(\alpha^{(t-1)}) - K\delta$ which involves $\mathcal{O}(q \cdot \ell)$ operations.

4.2.4 Stopping Condition

The most common stopping condition used to terminate the decomposition algorithm is to check the KKT complementarity condition up to a fixed accuracy given by ε. As the gradient $\nabla f(\alpha^{(t)})$ is available the dual gap function

$$\begin{aligned} \psi(\alpha) = & \max \left\{ \frac{\partial f(\alpha)}{\partial \alpha_n} \;\middle|\; \alpha_n < U_n,\, n \in \{1, \ldots, \ell\} \right\} \\ & - \min \left\{ \frac{\partial f(\alpha)}{\partial \alpha_n} \;\middle|\; \alpha_n > L_n,\, n \in \{1, \ldots, \ell\} \right\} \end{aligned}$$

can be computed in linear time.[3] It is positive for non-optimal points $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ and zero or negative for optimal points $\alpha \in \mathcal{R}^*$ (see Lemma 1 by Keerthi et al., 2001). The condition $\psi(\alpha) \leq \varepsilon$ is then used to stop the optimization loop (usually with a value of $\varepsilon = 0.001$). Of course, in SVM training we are first of all interested in the accuracy of

[3] To make this function well-defined we use the conventions $\min(\emptyset) = +\infty$ and $\max(\emptyset) = -\infty$.

the corresponding approximate solution of the primal problem. List et al. [2007] have shown that the usage of the dual gap function ψ for stopping is a meaningful condition, as the gap of the dual SVM problem translates, up to a constant factor, into a gap for the primal problem.

4.2.5 Caching and Shrinking

In elaborate implementations the algorithm is accompanied by a kernel cache and a shrinking heuristic [Joachims, 1998]. The caching technique exploits the fact that the decomposition algorithm needs the rows (or columns, as the matrix is symmetric) of the kernel matrix K that correspond to the indices in the current working set $B^{(t)}$, for example for the update of the gradient. Each row can be computed from the kernel function in $\mathcal{O}(\ell)$ operations.

The kernel cache uses a predefined amount of working memory to store rows of the kernel matrix K which have already been computed in earlier iterations. Therefore the algorithm needs to recompute only those rows from the training data evaluating the possibly costly kernel function which have not been used recently. We will later use the fact that the most recently used rows of the kernel matrix K are available from the cache. A cached row is available in constant time. As evaluations of the kernel function can be quite costly this technique can significantly speed up the algorithm. The free choice of the size of the kernel cache makes the decomposition algorithm freely scalable w.r.t. its working memory consumption.

The shrinking heuristic removes variables α_n from the problem that are likely to end up at the box boundaries in the final solution. The algorithm simply assumes that certain variables which are already at the boundary will not be selected into the working set again. Several heuristics have been proposed how to make the decision which variables to remove [Joachims, 1998, Fan et al., 2005]. Shrinking has the effect that all computations of kernel matrix rows, the working set selection, the update of the gradient and the check of the stopping condition can be restricted to the remaining variables, which can be a small subset of the whole set of ℓ variables. The result is an enormous speed up for the subsequent iterations. The price is that, as soon as the current solution is judged to be sufficiently close to the optimum, the gradient of the removed variables has to be recomputed from scratch, taking $\mathcal{O}(\ell^2)$ operations, to verify the optimality. In case of a wrong shrinking decision the algorithm needs additional iterations to complete the optimization. Nevertheless shrinking saves a large fraction of the overall training time in most cases.

The techniques of caching and shrinking perfectly cooperate and result in an enormous speed up of the training process. This is because the cache gets more efficient whenever the shrinking heuristic removes a variable from the problem, as the kernel matrix of the shrunken problem scales quadratically with the number of remaining variables.

4.3 Minimal Working Sets

We have seen that, in order to keep the search point feasible, we have to use working sets of size $|B^{(t)}| \geq 2$. It turns out that the minimal size of exactly $q = 2$ elements is an advantageous choice for a number of reasons. The selection of exactly two variables in each working set goes back to the sequential minimal optimization (SMO) algorithm by Platt [1998]. Here the term *minimal* refers to the size of the working sets used. In general, decomposition algorithms using two-element working sets are often termed SMO-type or SMO-like algorithms, although some parts of the original SMO algorithm have been replaced by alternative approaches. In this sense SMO should be understood as a

family of methods rather than Platt's original algorithm.

The choice of $q = 2$ is not only the canonical one. More importantly, it is even better suited for the solution of problem (4.1) because it explicitly exploits the presence of a single equality constraint. Besides the simplicity of the solution of the sub-problem, this is another advantage over the more general decomposition scheme with $q > 2$:

Because of its minimal working set size the SMO algorithm makes less progress in a single iteration compared to larger working sets. On the other hand single iterations are faster. Thus, there is a trade-off between the time per iteration and the number of iterations needed to come close enough to the optimum. The decisive advantage of SMO in this context is that it can take its decisions which working set B to choose more frequently between its fast iterations. This strategy has proven beneficial in practice.

4.3.1 Search Directions

In the SMO algorithm the sub-problems are made up by only two variables. With one equality constraint the problem becomes one-dimensional, that is, the algorithm just needs to find the optimum on a line. It makes sense to talk about a search direction in this case, which can be defined as a vector of fixed length pointing along this line. We have exactly two choices for such a vector, as each search direction v gives rise to another search direction $-v$ along the same line. Thus each working set corresponds to two search directions. In the following it will facilitate the clarity of the presentation to fix the search direction associated with a working set. To achieve this we will use tuples instead of sets to index the active variables.[4] Let $\mathcal{B} = \{(i,j) \,|\, i,j \in \{1,\dots,\ell\} \text{ and } i \neq j\}$ denote the set of tuples corresponding to two-element working sets. Following Keerthi et al. [2001] we define the index sets

$$\begin{aligned} I_{\text{up}}(\alpha) &= \{n \in \{1,\dots,\ell\} \mid \alpha_n < U_n\} \\ I_{\text{down}}(\alpha) &= \{n \in \{1,\dots,\ell\} \mid \alpha_n > L_n\} \\ \mathcal{B}(\alpha) &= \{(i,j) \mid i \in I_{\text{up}}(\alpha),\, j \in I_{\text{down}}(\alpha),\, i \neq j\} \end{aligned}$$

for each feasible point $\alpha \in \mathcal{R}$. The set $I_{\text{up}}(\alpha)$ contains the indices of all variables α_n which can be increased without immediately violating the box constraint. Analogously, variables α_n with $n \in I_{\text{down}}(\alpha)$ can be decreased. The equality constraint in problem (4.1) keeps the sum of all variables constant, such that an increase of one variable must correspond to a decrease of another variable in order to keep the solution feasible. Thus, any non-trivial step with a two-variable working set must involve exactly one variable from $I_{\text{up}}(\alpha)$ and one variable from $I_{\text{down}}(\alpha)$. This corresponds to the choice of a tuple $B = (i,j) \in \mathcal{B}(\alpha)$, where by convention the first index can be increased and the second one can be decreased. With this interpretation we assign the search direction

$$v_B = v_{(i,j)} = e_i - e_j \in \mathbb{R}^\ell$$

to each tuple $B = (i,j)$ where $e_n = (0,\dots,1,\dots,0)^T$ is the n-th standard basis vector in $\mathbb{R}^\ell$. Adding this search direction, possibly scaled with a factor $\mu > 0$, to the current search point amounts to increasing α_i and decreasing α_j by μ. We will later enforce the validity of this interpretation with a technical condition on the choice of the tuples. Although we use tuples instead of sets to describe the set of active variables we will nevertheless stick to the common term working *set* whenever there is no ambiguity.

[4] The usage of tuples is in contrast to the notation in the standard literature. We will nevertheless stick to this decision because we feel that it is natural and helpful for the geometric understanding of problem (4.1) to keep the additional orientation information directly in the working set notation.

There are always exactly two tuples (i, j) and (j, i) corresponding to a working set $\{i, j\}$. As discussed above, the corresponding search directions are related as $v_{(i,j)} = -v_{(j,i)}$, spanning the same line.

On $\mathcal{R}$ we define an equivalence relation: $\alpha \sim_{\mathcal{B}} \alpha' \Leftrightarrow \mathcal{B}(\alpha) = \mathcal{B}(\alpha')$. The equivalence classes of this relation are denoted by $[\alpha]$. Each equivalence class consists of the points where the same set of working sets can be chosen, corresponding to directions not immediately pushing towards the box boundary. By definition $\mathcal{B}(\alpha')$ is independent of $\alpha' \in [\alpha]$. Thus we will use the notation $\mathcal{B}([\alpha])$ in the following. Analogously it makes sense to write $I_{\text{up}}([\alpha])$ and $I_{\text{down}}([\alpha])$ for the index sets.

With this notation Algorithm 4.2 summarizes the SMO algorithm.

Algorithm 4.2: General SMO Algorithm

Input: feasible initial point $\alpha^{(0)}$, accuracy $\varepsilon \geq 0$
compute the initial gradient $G^{(0)} \leftarrow \nabla f(\alpha^{(0)}) = y - K\alpha^{(0)}$
set $t \leftarrow 1$
do
1 select a working set $B^{(t)} \in \mathcal{B}([\alpha^{(t-1)}])$
2 solve the sub-problem induced by $B^{(t)}$ and $\alpha^{(t-1)}$, resulting in $\alpha^{(t)} = \alpha^{(t-1)} + \mu v_{B^{(t)}}$ with $\mu \geq 0$
3 compute the gradient $G^{(t)} \leftarrow \nabla f(\alpha^{(t)}) = G^{(t-1)} - \mu K v_{B^{(t)}}$
4 check the stopping condition, depending on ε
set $t \leftarrow t + 1$
loop

4.3.2 The Geometry of the Feasible Region

Let us return to the polytope $\mathcal{R}$. An interesting thing about polytopes is that they can be decomposed into sub-polytopes similar to a simplex decomposition. For this construction we first fix the interior $\overset{\circ}{\mathcal{R}}$ of $\mathcal{R}$ as the highest-dimensional open sub-polytope. Then the boundary $\partial\mathcal{R}$ is written as the unique minimal union of polytopes. These polytopes have codimension one and intersect each other in even smaller polytopes of codimension two. We continue this process with all polytopes of codimension one. The interior of each such set is an open sub-polytope of $\mathcal{R}$ and we proceed with their boundaries, again reducing the dimension by one. This process is repeated until the boundaries become zero-dimensional, forming a finite set of points. These points are then the zero-dimensional sub-polytopes. Observe how the following elementary statements connect the geometric structure of the polytope decomposition with the structure of the equivalence classes $[\alpha]$ and the search directions v_B:

- The equivalence classes $[\alpha]$ decompose the set $\mathcal{R}$ exactly into its sub-polytopes.
- Each one-dimensional open sub-polytope $[\alpha]$ of $\mathcal{R}$ is contained in a line with direction $v_{(i,j)}$ for the unique pair $\{i, j\}$ such that $\{(i, j), (j, i)\} \subset \mathcal{B}([\alpha])$.

For some geometrical considerations the embedding of the polytope $\mathcal{R}$ into $\mathbb{R}^\ell$ is not optimal. This canonical embedding is important because the box constraints are nicely aligned to the given coordinates, but sometimes it is more appropriate to consider the embedding $\mathcal{R} \subset V = \text{span}(\mathcal{R})$. The vector space V is the smallest subspace of $\mathbb{R}^\ell$ containing $\mathcal{R}$. The codimension of V in $\mathbb{R}^\ell$ is at least one due to the equality constraint

of problem (4.1). The codimension can actually be larger as we do not exclude the case $L_n = U_n$ of coinciding lower and upper bounds in the box constraints.[5]

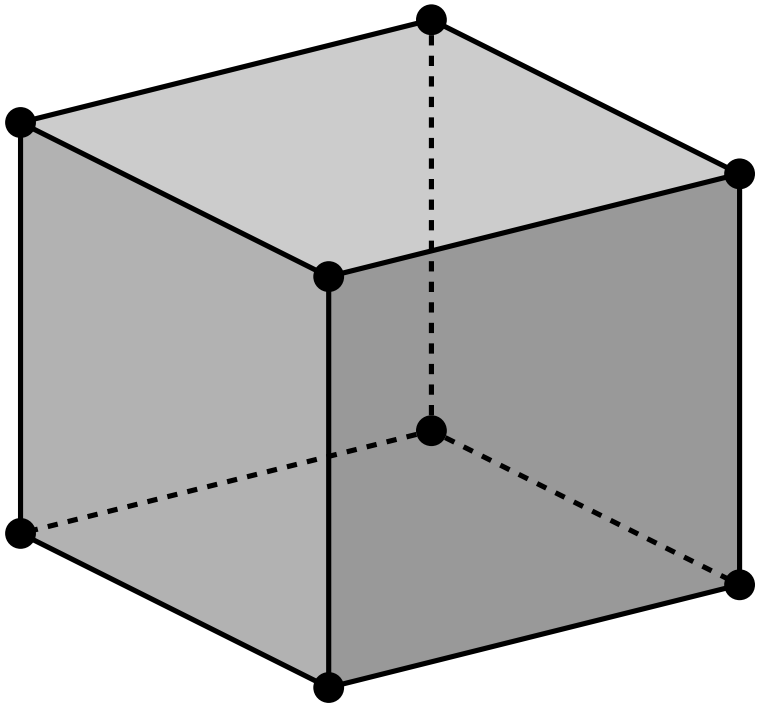

Figure 4.1: Decomposition of the polytope $\mathcal{R}$ into sub-polytopes. In this low dimensional example the polytope is illustrated as a three dimensional box. Note that in general the edges do not form right angles. Here, the equivalence classes $[\alpha]$ are eight vertices, twelve edges, six faces, and one cell. In each of these sets the SMO algorithm can choose among a certain set $\mathcal{B}([\alpha])$ of potential working sets, corresponding to directions which are parallel to the edges of the polytope.

The decomposition is illustrated in Figure 4.1. The following two lemmata provide additional insights into the connection of the polytope decomposition and the sets $\mathcal{B}([\alpha])$ of feasible search directions.

Lemma 4.1. *For $\alpha \in \mathcal{R}$ we define the set $D([\alpha]) = \{v_{(i,j)} \,|\, (i,j) \in \mathcal{B}([\alpha])$ and $(j,i) \in \mathcal{B}([\alpha])\}$. We can recover the sub-polytope $[\alpha]$ from the affine linear space*

$$E(\alpha) = \Big\{ \alpha + \sum_{v \in D([\alpha])} \lambda_v \cdot v \;\Big|\; \lambda_v \geq 0 \text{ for all } v \in D([\alpha]) \Big\} \subset V$$

as $\overline{[\alpha]} = E(\alpha) \cap \mathcal{R}$.

Proof. It is easy to see that $E(\alpha)$ is indeed an affine linear subspace, as for each $v \in D([\alpha])$ we have $-v \in D([\alpha])$ such that the non-negativity constraint on λ_v can be dropped. We fix $\alpha, \alpha' \in \mathcal{R}$ with $\alpha' \sim_{\mathcal{B}} \alpha$. This relation immediately implies $\alpha_n \in \{L_n, U_n\} \Rightarrow \alpha_n = \alpha'_n$,

[5] For examples, this can be useful for the efficient computation of the leave-one-out error, see Chapter 9.

implying $E(\alpha) \cap \mathcal{R} \subset \overline{[\alpha]}$. The other inclusion follows from the fact that a closed polytope $\overline{[\alpha]}$ is the convex hull of its vertices (zero-dimensional sub-polytopes). It is further clear that $E(\alpha) = E(\alpha') \Leftrightarrow \alpha \sim_{\mathcal{B}} \alpha'$ such that $E([\alpha])$ is well-defined. □

For obvious reasons we call the following lemma the cone-lemma.

Lemma 4.2. *For each $\alpha \in \mathcal{R}$ the feasible region is contained in the cone*

$$\mathcal{C}(\alpha) = \left\{ \alpha + \sum_{B \in \mathcal{B}([\alpha])} \lambda_B \cdot v_B \;\middle|\; \lambda_B \geq 0 \text{ for all } B \in \mathcal{B}([\alpha]) \right\} \subset V \ .$$

Proof. For $|\mathcal{R}| \leq 1$ the statement is trivial. Otherwise by convexity the feasible region contains at least one line segment and there is at least one feasible working set for α, that is, the sets $I_{\text{up}}(\alpha)$, $I_{\text{down}}(\alpha)$ and $\mathcal{B}(\alpha)$ are non-empty for all $\alpha \in \mathcal{R}$. Let $S = I_{\text{up}}(\alpha) \cup I_{\text{down}}(\alpha)$ denote the index set of non-trivial coordinates whose upper bound is strictly greater than the lower bound. The set S is independent of α. We have $n \in S$ if $L_n < U_n$ and $n \notin S$ if $L_n = U_n$, which reduces the dimension of $\mathcal{R}$ by one.

First we consider the case that $[\alpha]$ is a vertex (zero-dimensional polytope). Recall that the vectors $v_{(i,j)}$ for $i, j \in S$ are parallel to the edges (one-dimensional polytopes) in the polytope decomposition of the feasible region defined by the equivalence classes $[\alpha]$ (see Figure 4.1). The directions corresponding to two edges span a face, and by induction over the dimensions it is easy to see that the cone $\mathcal{C}(\alpha)$ spanned by all edges ending in the vertex α spans the whole polytope $\mathcal{R}$.

For a general α the cone $\mathcal{C}(\alpha)$ is invariant under translation within the subspace $E([\alpha])$. We define the equivalence relation $\alpha \sim_E \alpha'$ if $\alpha - \alpha' \in E([\alpha])$ on V and consider the quotient $\mathcal{R}/\sim_E$ which is a polytope in $V/E([\alpha])$ again. Then the equivalence class of α is a vertex in this polytope and we apply the above argument for vertices. □

With the invariance property used in the proof it is easy to see that $\mathcal{C}(\alpha) = \mathcal{C}(\alpha') \Leftrightarrow \alpha \sim_{\mathcal{B}} \alpha'$ such that we can write $\mathcal{C}([\alpha])$ for the cone associated with all $\alpha' \in [\alpha]$.

We can decompose the polytope $\mathcal{R}^*$ of optimal points analogously, but this decomposition has nothing to do with the search directions v_B and the equivalence classes $[\alpha]$. Such a decomposition is of minor interest as the only thing the decomposition algorithm ever does in an optimal point is to stop immediately.

4.3.3 Optimal Solution of the Sub-Problem

An easy consequence of the minimality of the working set size is that the solution of the sub-problem posed by the current search point $\alpha^{(t-1)} \in \mathcal{R}$ and the working set $B^{(t)} = (i, j) \in \mathcal{B}(\alpha^{(t-1)})$ becomes rather simple. This is an advantage for the implementation of the algorithm as well as for its analysis.

As already mentioned above the main simplification with two variables in the working set is that the feasible region of the sub-problem (4.2) is a line segment (see Figure 4.2). Thus the next search point can be written as $\alpha^{(t)} = \alpha^{(t-1)} + \mu v_B^{(t)}$ with $\mu \in \mathbb{R}$.

The Taylor expansion of the objective function on the feasible line segment around the current search point is

$$f(\alpha^{(t-1)} + \mu v_B^{(t)}) = f(\alpha^{(t-1)}) + v_{B^{(t)}}^T \nabla f(\alpha^{(t-1)}) \cdot \mu - \frac{1}{2} v_{B^{(t)}}^T K v_{B^{(t)}} \cdot \mu^2 \ .$$

Plugging everything in we arrive at the equivalent problem

$$\begin{aligned} \text{maximize} \quad & v_{B^{(t)}}^T \nabla f(\alpha^{(t-1)}) \cdot \mu - \frac{1}{2} v_{B^{(t)}}^T K v_{B^{(t)}} \cdot \mu^2 \\ \text{s.t.} \quad & L \leq \mu \leq U \end{aligned}$$

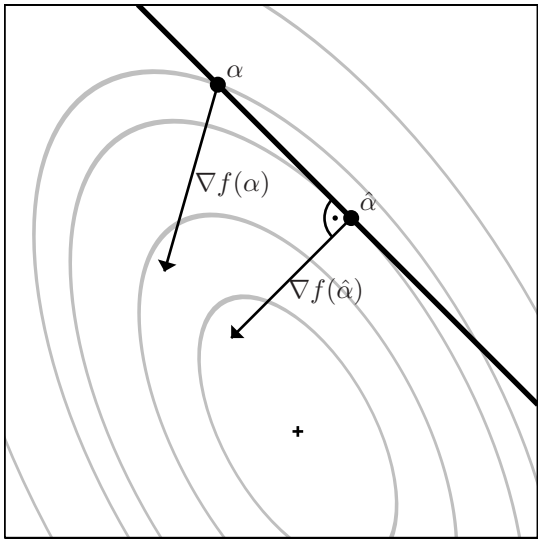

Figure 4.2: The 2-dimensional SMO sub-problem restricted to the equality constraint (solid 'feasible' line) and the box constraints (boundary). The point fulfilling the equality constraint with gradient orthogonal to the feasible line is a candidate for the solution of the sub-problem. If it is not feasible w.r.t. the box constraints it has to be moved along the line onto the box boundary, see Platt [1998].

for $\mu^{(t)} \in \mathbb{R}$ with bounds

$$\begin{aligned} L &= \max\{L_i - \alpha_i^{(t-1)}, \alpha_j^{(t-1)} - U_j\} \leq 0 \\ U &= \min\{U_i - \alpha_i^{(t-1)}, \alpha_j^{(t-1)} - L_j\} \geq 0 \ . \end{aligned}$$

Due to the sparsity of the vector $v_{B^{(t)}}$ and because the gradient is available both terms

$$\begin{aligned} v_{B^{(t)}}^T \nabla f(\alpha^{(t-1)}) &= \left(\nabla f(\alpha^{(t-1)})\right)_i - \left(\nabla f(\alpha^{(t-1)})\right)_j \\ \text{and} \quad v_{B^{(t)}}^T K v_{B^{(t)}} &= K_{ii} - 2K_{ij} + K_{jj} \end{aligned}$$

can be computed in constant time. If we ignore the box constraints the optimal solution is found by setting the derivative of the objective w.r.t. μ to zero

$$\frac{\partial f(\alpha^{(t-1)} + \mu v_B^{(t)})}{\partial \mu} = 0$$

resulting in the unconstrained optimum

$$\hat{\mu} = \frac{v_{B^{(t)}}^T \nabla f(\alpha^{(t-1)})}{v_{B^{(t)}}^T K v_{B^{(t)}}} \ .$$

As the derivative is linear in μ, the step $\hat{\mu}$ coincides with the Newton step for finding the null of the derivative. Therefore we will refer to the step $\hat{\mu}$ as the Newton step in the fixed search direction $v_{B^{(t)}}^T$ in the following.

As the matrix K is not positive definite but only positive semi-definite in general it can happen that the search direction v_B is in the kernel of the matrix K. Then the denominator vanishes. There are two cases: If the numerator $v_B^T \nabla f(\alpha)$ vanishes, too,

the objective function on the line along v_B is constant. Then we arbitrarily define the optimal step to be $\hat{\mu} = 0$ as all points on a constant objective function are equally good. The second case is that the numerator is non-zero and the objective function is linear. Then, depending on the sign of the numerator, the solution can be improved arbitrarily on a diverging sequence and we write $\hat{\mu} = +\infty$ or $\hat{\mu} = -\infty$.

The sub-problem is solved by clipping the Newton step to the bounds:

$$\mu^* = \max\left\{\min\left\{\hat{\mu}, U\right\}, L\right\} \tag{4.4}$$

is the maximizer of the above sub-problem, resulting in the new search point $\alpha^{(t)} = \alpha^{(t-1)} + \mu^* v_{B^{(t)}}$.

For $\mu^* = \hat{\mu}$ we call the SMO step free. In this case the SMO step coincides with the Newton step in direction $v_{B^{(t)}}$. Otherwise the step is said to hit the box boundary.

The simplicity of the computations involved (in particular, an iterative process like interior point methods etc. is avoided) is one of the reasons for the success of the SMO algorithm.

There is a simple and insightful geometric interpretation of the solution of the sub-problem. Consider the case $Kv_B \neq 0$, that is, v_B is not in the kernel of K. If we ignore the box-constraints all optimal point starting from arbitrary $\alpha \in V$ fulfill the equation $v_B^T \nabla f(\alpha) = v_B^T(y - K\alpha) = 0$. This is an affine linear equation for α and the solutions form a hyperplane $H_B \subset V$. Then the solution of the sub-problem posed by α and B results in the unique intersection of the line $\alpha + \mathbb{R} \cdot v_B$ with the hyperplane H_B.

If we additionally consider the box constraint we get an interval constraint $L \leq \mu \leq U$ on the line $\alpha + \mu \cdot v_B$ for $\mu \in \mathbb{R}$. If the point $\alpha + \mathbb{R} \cdot v_B \cap H_B$ turns out to be infeasible, then the closest feasible point (either $\mu = L$ or $\mu = U$) is optimal.

In the case $Kv_B = 0$, that is, v_B is in the kernel of K, there are again two cases. For $v_B^T \nabla f(\alpha) = v_B^T(y - K\alpha) = 0$ the objective function on the line $\alpha + \mathbb{R} \cdot v_B$ is constant. This is equivalent to $v_B^T y = 0$. Then the sub-problem is solved by the point α itself and no progress is made. For $v_B^T y \neq 0$ the unconstrained solution is described by $\hat{\mu} = \pm\infty$ resulting in $\mu^* = U$ or $\mu^* = L$, that is, the optimal point is always on the box boundary.

To get a better grip on this behavior we interpret the step size μ^* as a family $\mu_B^* : \mathcal{R} \to \mathbb{R}$ of functions.

Lemma 4.3. *For each $B \in \mathcal{B}$ the function $\mu_B^* : \mathcal{R} \to \mathbb{R}$ is continuous.*

Proof. We write $B = (i, j)$ and apply the definitions of μ^* (equation (4.4)), $\hat{\mu}$, and the upper and lower bounds L and U to explicitly compute the step size in the form

$$\mu_B^*(\alpha) = \max\left\{ \min\left\{ \frac{v_B^T(y - K\alpha)}{v_B^T K v_B}, \right.\right.$$
$$\left. U_i - \alpha_i^{(t-1)},\, \alpha_j^{(t-1)} - L_j \right\},$$
$$\left. L_i - \alpha_i,\, \alpha_j - U_j \right\} .$$

The only problems could occur if the denominator vanishes. As discussed above we have two cases: For $v_B^T(y - K\alpha) = 0$ we define the quotient to be zero such that the function μ_B^* is constantly zero. Otherwise the fraction is either $+\infty$ for $v_B^T(y - K\alpha) > 0$ or $-\infty$ for $v_B^T(y - K\alpha) < 0$. In these cases the fraction is either not the minimizer or not the maximizer of the outer min and max operations, and the remaining terms are clearly continuous. From the above formula we can see that μ_B^* is piecewise linear. □

Analogously we introduce the family $\hat{\mu}_B : \mathcal{R} \to \mathbb{R} \cup \{\pm\infty\}$, of functions computing the Newton step size.

4.3.4 Maximum Violating Pair Working Set Selection

A lot of working set selection schemes for SMO-type algorithms have been proposed in the literature [Platt, 1998, Keerthi et al., 2001, Hush and Scovel, 2003, List and Simon, 2004, Fan et al., 2005, Glasmachers and Igel, 2006]. The most important one is the maximum (or most) violating pair (MVP) selection [Joachims, 1998, Keerthi et al., 2001]

$$B = \arg\max \left\{ v_B^T \nabla f(\alpha) \;\middle|\; B \in \mathcal{B}(\alpha) \right\} . \tag{4.5}$$

Its name comes from the fact that the pair B of variables selected by this policy most strongly violates the KKT condition (see Keerthi et al., 2001)

$$\alpha \in \mathcal{R}^* \quad \Leftrightarrow \quad (\nabla f(\alpha))_i \leq (\nabla f(\alpha))_j \;\; \forall i \in I_{\text{up}}(\alpha), j \in I_{\text{down}}(\alpha)$$

for optimality of problem (4.1). Therefore it is reasonable to select this pair for optimization. Further this policy selects the search direction v_B with $B \in \mathcal{B}(\alpha)$ which best reflects the gradient of the objective function in that the angle between the gradient $\nabla f(\alpha)$ and the search direction v_B is minimal. Thus it is the best sparse search direction we can deduce from the first order Taylor approximation of the objective function.

4.3.5 Second Order Working Set Selection

To improve on the MVP strategy it is natural to consider the second order Taylor approximation of the objective function. As the objective function is quadratic this Taylor approximation is already exact. However, this involves the Hessian $-K$ which is assumed not to fit into the available working memory.

We introduce the family $g_B : \mathcal{R} \to \mathbb{R}^{\geq 0}$, $B \in \mathcal{B}$ of functions which compute the progress or functional gain of a SMO step starting in α with working set B [Glasmachers and Igel, 2006]. The point $\alpha' = \alpha + \mu_B^*(\alpha) v_B$ is the next search point resulting from this step. Then $g_B(\alpha) = f(\alpha') - f(\alpha)$ computes the progress or gain this step makes in terms of the objective function f.

Lemma 4.4. *For each $B \in \mathcal{B}$ the function $g_B : \mathcal{R} \to \mathbb{R}^{\geq 0}$ is continuous.*

Proof. We have $g_B(\alpha) = f(\alpha + \mu_B^*(\alpha) v_B) - f(\alpha)$. By Lemma 4.3 μ_B^* is continuous which shows the assertion. □

In the computation of g_B we have to consider several different cases corresponding to free and bounded SMO steps. If we decide to just ignore the box constraints the situation is simpler. We define the family

$$\tilde{g}_B : \mathcal{R} \to \mathbb{R}^{\geq 0} \cup \{\infty\}, \qquad \alpha \mapsto \frac{1}{2} \frac{(v_B^T \nabla f(\alpha))^2}{v_B^T K v_B} = f(\alpha + \hat{\mu}_B(\alpha) v_B) - f(\alpha) \tag{4.6}$$

of functions computing the gain corresponding to the Newton step $\hat{\mu}$. Therefore we call this function the Newton step gain. It is easy to see that for each $B \in \mathcal{B}$ the Newton step gain is an upper bound for the SMO gain g_B with equality for free SMO steps, that is, for $\mu_B^*(\alpha) = \hat{\mu}_B(\alpha)$. Otherwise the Newton step gain corresponds to an infeasible step.

The MVP strategy has recently been superseded by the second order selection [Fan et al., 2005]

$$\begin{aligned} &B = (i, j) \text{ with} \\ &i = \arg\max \left\{ \frac{\partial f}{\partial \alpha_n}(\alpha) \;\middle|\; n \in I_{\text{up}}(\alpha) \right\} \\ &j = \arg\max \left\{ \tilde{g}_{(i,n)}(\alpha) \;\middle|\; n \in I_{\text{down}}(\alpha) \right\} . \end{aligned} \tag{4.7}$$

This selection strategy was never given an accurate name. It is implemented in the current version 2.85 of the popular support vector machine software LIBSVM. According to the title of its first publication we will refer to it simply as second order (SO) working set selection policy.

The selection of the first index i coincides with the MVP policy. Once this index is fixed the function $\tilde{g}_{(i,n)}(\alpha)$ is maximized to determine the second index.[6]

This working set selection can be interpreted as follows: The function $\tilde{g}_B(\alpha)$ is the optimum of the second order Taylor series of the functional gain. As the objective function is quadratic the Taylor approximation is exact. The algorithm selects the working set using the second order approximation of the objective function. The computation of this gain is similar to the solution of the sub-problem itself, but neglecting the box constraint.

For a fixed working set the Newton step gain can be computed in constant time. The selection could be done by maximizing the Newton step gain over all possible working sets $B \in \mathcal{B}(\alpha)$. Because the number of working sets is quadratic in ℓ in general this would require $\mathcal{O}(\ell^2)$ operations. Fan et al. [2005] propose to use the first index selected by the MVP strategy and then to select the second index involving the second order information, resulting in linear scaling of the runtime. This strategy is indeed superior to the MVP strategy (see Chapter 6 or refer to Fan et al., 2005).

4.3.6 Properties of Working Set Selection Policies

A working set selection (WSS) policy assigns a tuple $B \in \mathcal{B}(\alpha)$ to each feasible non-optimal point α. This is a convenient situation, in particular for the analysis of the SMO algorithm, but in general we will allow the selection to depend on further information.

The working set selection strategies defined above have the property to select B such that $v_B^T \nabla f(\alpha) > 0$. That is, the objective function grows in the search direction selected, resulting in a positive step size $\mu^* > 0$. This is in accordance with the interpretation of the search directions v_B for $B \in \mathcal{B}(\alpha)$ that there is a positive $r > 0$ such that for all $\mu \in [0, r]$ the point $\alpha + \mu v_B$ is feasible. In general this needs not holds for any negative value of μ.

Let us check the other possibilities. The equality $v_B^T \nabla f(\alpha) = 0$ can be excluded for any reasonable working set selection strategy because then the step $\mu^* = 0$ would be trivial and the algorithm would be stuck, selecting the same working set again and again without ever reaching a new search point. For $v_B^T \nabla f(\alpha) < 0$ there are two cases: First, let us assume that the search direction $-v_B \in \mathcal{B}(\alpha)$ is available. Then we can, w.l.o.g., request the algorithm to return $-v_B$ (which corresponds to switching the entries of the tuple B). This choice guarantees $\mu^* > 0$. If the direction $-v_B \notin \mathcal{B}(\alpha)$ is not available we get $\mu^* = 0$ and SMO gets stuck again.

Therefore it poses no restriction to assume that any reasonable working set selection algorithm returns a working set $B \in \mathcal{B}(\alpha)$ fulfilling $v_B^T \nabla f(\alpha) > 0$. This prerequisite is the price we have to pay for the consideration of search directions v_B and tuples $B = (i, j)$ instead of working sets $\{i, j\}$. The technical condition $v_{(i,j)}^T \nabla f(\alpha) > 0$ enables us to assign a unique direction to each (useful) working set $\{i, j\}$.

Definition 4.5 (Keerthi and Gilbert [2002])**.** *A tuple $B \in \mathcal{B}(\alpha)$ fulfilling $v_B^T \nabla f(\alpha) > 0$ is called a violating pair.*

The name comes from the fact that a violating pair is sufficient to show that the KKT conditions of optimality are violated. This property implies positive gain $g_B(\alpha) > 0$. In other words, the step in direction v_B starting from α makes progress. It is well known that

[6] For non-optimal $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ this algorithm never selects $i = j$ such that the working set selection is well defined.

for all non-optimal points $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ there exists a violating pair (This is a prerequisite for the convergence of SMO to the optimum). This concept is a key ingredient of the following general definition of working set selection:

Definition 4.6. *A working set selection policy is based on a state space $\mathcal{S}$ which can be written as a product $\mathcal{S} = (\mathcal{R} \setminus \mathcal{R}^*) \times \tilde{\mathcal{S}}$. Then a working set selection policy is a map $W : \mathcal{S} \to \mathcal{B}$ fulfilling $W(\alpha, \tilde{s}) \in \mathcal{B}(\alpha)$ and $v_{W(\alpha,\tilde{s})}^T \nabla f(\alpha) > 0$. We also consider maps $W : \mathcal{R} \times \tilde{\mathcal{S}} \to \mathcal{B}$ such that $W|_{(\mathcal{R}\setminus\mathcal{R}^*)\times\tilde{\mathcal{S}}}$ fulfills the above properties of a working set selection policy.*

This definition basically ensures that in non-optimal points a working set selection policy selects violating pairs.

The state $s = (\alpha, \tilde{s})$ plays the role of the initial state of the working set selection algorithm. In most cases, including MVP and SO selection, the state will consist only of the current search point α, resulting in $\mathcal{S} = \mathcal{R} \setminus \mathcal{R}^*$, under slight misuse of notation with a trivial second dummy component $\tilde{\mathcal{S}} = \{\bullet\}$ and the identification $S = (\mathcal{R} \setminus \mathcal{R}^*) \times \tilde{S} \cong \mathcal{R} \setminus \mathcal{R}^*$.

Both MVP and SO are examples of valid working set selection strategies in this sense with state space $\mathcal{S} = \mathcal{R} \setminus \mathcal{R}^*$. That is, both strategies base their decisions solely on properties of the current search point (including the derivative of the objective function in this point) to select violating pairs. We will see more complex examples later on.

4.3.7 Stopping Criterion

The stopping criterion used by the SMO algorithm is inherited from the more general decomposition scheme. However, with the new terminology introduced with the SMO algorithm, in particular with the notion of search directions v_B, it makes sense to reformulate the stopping condition. We define the function

$$\psi : \mathcal{R} \to \mathbb{R}, \qquad \alpha \mapsto \max \left\{ v_B^T \nabla f(\alpha) \;\middle|\; B \in \mathcal{B}(\alpha) \right\}$$

which nearly[7] coincides with the function ψ defined in section 4.2.4. From this notation it is obvious that the MVP working set selection strategy selects the maximizer of this function.

4.3.8 Kernel Evaluations in SMO

Like in the general decomposition scheme we need at least the rows of the kernel matrix K that correspond to the working set $B^{(t)}$ in each iteration t to compute the gradient update. In general it will slow the overall algorithm down if we make use of further parts of the kernel matrix for other steps like the working set selection or the solution of the sub-problem, as these parts may need to be computed from the kernel function and the kernel cache gets less efficient.

The solution of the sub-problem turns out to be no problem, as for a working set $B = (i, j)$ only the kernel matrix entries K_{ii}, K_{jj} and $K_{ij} = K_{ji}$ are needed. These are included in the i-th and the j-th row of the matrix which are needed for the gradient update anyway.

Working set selection based on second order terms of the objective function, which directly correspond to the kernel matrix, seems to be more problematic at the first glance.

[7] In fact, the only difference is that in this definition the indices i and j in the working set $B = (i, j)$ must be distinct. This changes the maximizer only in extremely rare cases where an optimal point has exactly one free variable. In short, the maximum can only deviate for optimal points, but the decisive property $\psi(\alpha) > 0 \Leftrightarrow \alpha \in \mathcal{R} \setminus \mathcal{R}^*$ remains valid.

In the SO strategy we are faced with the computation of all terms $\tilde{g}_{(i,n)}(\alpha)$ for all $n \in I_{\text{down}}(\alpha)$ with fixed first index i, involving the kernel matrix entries K_{ii}, K_{nn} and $K_{in} = K_{ni}$. The diagonal entries K_{ii} and K_{nn} can be precomputed before the optimization loop starts in $\mathcal{O}(\ell)$ time and memory, which is only a small overhead. The entries K_{in} are found in the i-th row of K which is needed for the gradient update anyway as i is already fixed in the first component of the working set. Thus, it seems that additional kernel matrix values are needed for the working set selection, but this actually reduces to the diagonal entries which pose no problem. The only difference is that the i-th row of K needs to be computed several steps earlier than the gradient update.

4.3.9 Ambiguity of Working Set Selection

As noted above it is canonical to start the SMO algorithm from $\alpha^{(0)} = (0, \ldots, 0)^T$ in order to avoid the quadratic time complexity of the computation of the initial gradient. Then, in the SVM context, this gradient coincides with the vector y of labels. All components have equal absolute value and point into the box. That is, all variables corresponding to positive examples maximize the function $\frac{\partial f}{\partial \alpha_n}(0)$ used to determine the first index $i^{(1)}$ of both the MVP and SO working sets $W_{\text{MVP}}(0)$ and $W_{\text{SO}}(0)$. For MVP even the second index $j^{(1)}$ can be chosen arbitrarily from the variables corresponding to negative class examples. This ambiguity poses no principle problem but of course different initial choices lead to different optimization paths. In experiments it turns out that this initial choice can significantly influence the overall number of iterations and the runtime of the SMO algorithm. For example, the software LIBSVM simply chooses the highest index $i^{(1)} = \max(I_{\text{up}}(\alpha^{(0)}))$. Therefore the results may depend on the order in which the examples are presented to the machine.

This effect plays an important role for the experimental evaluations presented in the following chapters. To get rid of the undesired variability resulting from the order of the training examples it is natural to take a large enough sample of identical trials which differ only in the order of the training dataset which is drawn i.i.d. from the permutations on $\{1, \ldots, \ell\}$. Then standard statistical tests can be used to assess the results.

4.3.10 Convergence Results

It is easy to see that the SMO algorithm increases the objective function in each iteration. Of course, convergence to the optimum is not ensured and, in fact, Lin [2001] has given an example how the sequence can converge to a non-optimal point although a certifying pair is selected in each iteration. As convergence to the optimum for exact accuracy $\varepsilon = 0$ is equivalent to finite termination of the SMO algorithm for any accuracy $\varepsilon > 0$ this is an important property.

For several working set selection strategies including MVP and SO proofs of convergence to the optimum are available [Lin, 2001, Keerthi and Gilbert, 2002, Takahashi and Nishi, 2005, Fan et al., 2005, Chen et al., 2006] and we will present further proofs in the remainder of this thesis. There are several general proof techniques around:

1. Chen et al. [2006] have shown that if there exists $c > 0$ such that for all $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ an algorithm selects a working set $B \in \mathcal{B}(\alpha)$ with $v_B^T \nabla f(\alpha) \geq c \cdot \psi(\alpha)$ then it converges to the optimum. This definition happens to be equivalent to Definition 5.1 (see Chapter 5).

2. Another approach is to select a rate certifying pair [Hush and Scovel, 2003, Hush et al., 2006]. This approach has the advantage that it automatically supplies us with a convergence rate for the overall algorithm.

3. List and Simon [2004] propose to select working sets according to a so called q-sparse witness of sub-optimality. In this approach, the continuity of the witness functions plays a crucial role (see Section 6.1.1).

Besides the guaranteed convergence it is interesting how fast the SMO algorithm approaches the optimum. We are in general interested in guarantees on convergence rates and upper bounds on the number of iterations to reach a certain predefined accuracy. The approaches by Hush and Scovel [2003] and List and Simon [2004] are based on a worst case analysis. They therefore provide us with such rates and bounds by design. In contrast, the MVP and SO working set selections considered by Chen et al. [2006] are more practically relevant because they show high speed in experiments, but little is known about their worst case behavior. Only the above asymptotic result on their convergence speed is available. From a theoretical point of view the main challenge in the analysis of these policies is that they result from the maximization of discontinuous functions. In Chapter 5 we will show a possibility how to deal with this difficulty.

Chapter 5

Progress Rate for Free SMO Steps

The aim of this chapter is to prove a progress rate for free SMO steps for a quite general class of working set selection policies. This result generalizes a theorem by Chen et al. [2006] and is proven with completely different methods. Besides the generalization of the result the new proof has the advantage that the geometric structure of the problem is clearly brought to light, but at the cost that the progress rate remains implicit.

5.1 Prerequisites

We start with the class of working set selection policies considered throughout this chapter. For a fixed working set selection policy based on the state space $\mathcal{R} \setminus \mathcal{R}^*$ the SMO gain g and the Newton step gain $\tilde{g}$ depend only on the search point. We stress this fact by writing

$$g_W : \mathcal{R} \setminus \mathcal{R}^* \to \mathbb{R}^{\geq 0} \qquad g_W(\alpha) := g_B(\alpha) \text{ with } B = W(\alpha)$$

and analogously

$$\tilde{g}_W : \mathcal{R} \setminus \mathcal{R}^* \to \mathbb{R}^{\geq 0} \cup \{\infty\} \qquad \tilde{g}_W(\alpha) := \tilde{g}_B(\alpha) \text{ with } B = W(\alpha) \ .$$

Here, we index the gain functions with the working set selection policy W instead of the working set B. Therefore we obtain a single function instead of a family of functions. With this notation we define the class of working set selection policies considered in this chapter as follows:

Definition 5.1. *We say that a working set selection $W : \mathcal{R} \setminus \mathcal{R}^* \to \mathcal{B}$ (with state space $\mathcal{R} \setminus \mathcal{R}^*$) is a σ-working set selection (σ-WSS) with $\sigma > 0$ if $\tilde{g}_W(\alpha) \geq \sigma \cdot (\psi(\alpha))^2$ for all $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$.*

In the following we will need the constants

$$\sigma_{\max} = \max\{v_B^T K v_B \mid B \in \mathcal{B}\}$$
$$\sigma_{\min} = \min\{v_B^T K v_B \mid B \in \mathcal{B} \text{ with } v_B^T K v_B > 0\} \ .$$

We will denote the MVP selection and the SO selection by W_{MVP} and W_{SO}, respectively.

Lemma 5.2. *The working set selections W_{MVP} and W_{SO} are a $1/(2\sigma_{\max})$-selection and a $\sigma_{\min}/(2\sigma_{\max}^2)$-selection, respectively.*

Proof. Let B denote the working set selected in a search point $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$. First we treat the case $v_B \in \ker(K)$. Because both policies select violating pairs we have $v_B^T \nabla f(\alpha) > 0$ and thus $\tilde{g}_B(\alpha) = \infty$, which exceeds $\sigma \cdot (\psi(\alpha))^2$ for any value of σ.

For $v_B \notin \ker(K)$ we have $v_B^T K v_B > 0$. As noted before, the MVP working set selection is closely related to the function ψ. The inequality $\tilde{g}_{W_{\mathrm{MVP}}}(\alpha) \geq 1/2\sigma_{\max} \cdot (\psi(\alpha))^2$ holds by the definitions of $\tilde{g}_B$ and ψ. For the SO strategy and $v_B \notin \ker(K)$ we make use of a result by Fan et al. [2005, section 3] who derive the inequality

$$v_{W_{\mathrm{SO}}(\alpha)}^T \nabla f(\alpha) \geq \sqrt{\sigma_{\min}/\sigma_{\max}}\, \psi(\alpha) \ .$$

From the definition of $\tilde{g}_B(\alpha)$ applied to the working set $W_{\mathrm{SO}}(\alpha)$ we get the bound

$$\begin{aligned}
\tilde{g}_{W_{\mathrm{SO}}}(\alpha) &= \frac{1}{2} \frac{(v_{W_{\mathrm{SO}}(\alpha)}^T \nabla f(\alpha))^2}{v_{W_{\mathrm{SO}}(\alpha)}^T K v_{W_{\mathrm{SO}}(\alpha)}} \\
&\geq \frac{1}{2\,\sigma_{\max}} (v_{W_{\mathrm{SO}}(\alpha)}^T \nabla f(\alpha))^2 \\
&\geq \frac{1}{2\,\sigma_{\max}} \left(\sqrt{\frac{\sigma_{\min}}{\sigma_{\max}}} \right)^2 \cdot (\psi(\alpha))^2 \\
&= \frac{\sigma_{\min}}{2\,\sigma_{\max}^2} \cdot (\psi(\alpha))^2 \ .
\end{aligned}$$

□

For a σ-selection W we have

$$\tilde{g}_W(\alpha) = \frac{1}{2} \frac{(v_{W(\alpha)}^T \nabla f(\alpha))^2}{v_{W(\alpha)}^T K v_{W(\alpha)}} \geq \sigma (\psi(\alpha))^2 \ .$$

For strictly positive K the denominator never vanishes and we get

$$v_{W(\alpha)}^T \nabla f(\alpha) \geq \sqrt{2\sigma_{\min}\sigma}\, \psi(\alpha)$$

for non-optimal points. Thus, the definition of σ-selection is equivalent to WSS-4 by Fan et al. [2005], while Chen et al. [2006] use the term WSS-2, or constant-factor violating pair.

Next we will introduce the linear convergence result by Chen et al. [2006] which is mainly based on a linear progress rate for free SMO steps in sufficiently late iterations. This result is based on the following technical condition:

Definition 5.3. *We call an instance of problem* (4.1) *non-degenerate if all optimal solutions $\alpha^* \in \mathcal{R}^*$ fulfill*

$$\begin{aligned}
\alpha_n^* = L_n \ &\Rightarrow \ \frac{\partial f}{\partial \alpha_n}(\alpha^*) < \min \left\{ \frac{\partial f}{\partial \alpha_i}(\alpha^*) \ \middle| \ i \in I_{up}(\alpha^*) \right\} \\
\alpha_n^* = U_n \ &\Rightarrow \ \frac{\partial f}{\partial \alpha_n}(\alpha^*) > \max \left\{ \frac{\partial f}{\partial \alpha_j}(\alpha^*) \ \middle| \ j \in I_{down}(\alpha^*) \right\}
\end{aligned}$$

for all $n \in \{1, \ldots, \ell\}$ with $L_n < U_n$.

In the cases $\alpha^* = (L_1, \ldots, L_\ell)^T$ and $\alpha^* = (U_1, \ldots, U_\ell)^T$ the above quantities are not well defined because either $I_{\mathrm{up}}(\alpha^*)$ or $I_{\mathrm{down}}(\alpha^*)$ is empty. However, in these cases we have $|\mathcal{R}| = |\mathcal{R}^*| = 1$ which makes optimization including convergence to the optimum trivial. We can capture this case with the conventions $\max(\emptyset) = -\infty$ and $\min(\emptyset) = +\infty$.

Chen et al. [2006] have proven that in case the non-degeneracy condition holds there exists $\bar{k}$ such that for all $k > \bar{k}$ the SMO step in iteration k is free. This is a strong result because the analysis of the SMO algorithm becomes much simpler for free steps. Loosely speaking, the condition ensures that an optimum is not at the bound "by chance". That is, non-degeneracy ensures that a point with the same gradient can only be optimal if it is not moved away from the boundary.

Lemma 5.4. *Consider a non-degenerate instance of problem* (4.1)*. Then the optimal solution is unique:* $|\mathcal{R}^*| = 1$.

We postpone the simple proof to the next section. This lemma shows that non-degeneracy is a strong assumption which excludes relevant cases.

Under the additional precondition that the matrix K is strictly positive definite (which induces $|\mathcal{R}^*| = 1$ again) Chen et al. [2006] derive an asymptotic linear convergence rate $c < 1$ for the SMO algorithm: For points α before and α' after the SMO step in iteration $k > \bar{k}$ we have

$$(f^* - f(\alpha')) \leq c \cdot (f^* - f(\alpha)) \ .$$

In other words, a *free* SMO step in iteration $k > \bar{k}$ makes a certain uniformly guaranteed rate of progress to the optimum. Although the preconditions turn out to be quite strong, this is an interesting result. Chen et al. [2006] provide such a rate c in an explicit form involving the problem dimension ℓ, the eigenvalues $v_B^T K v_B$ as well as eigenvalues of the full matrix K^{-1}, and of course the constant σ of the working set selection policy.

It should be noted that this convergence rate result holds for working set selection policies that are not tailored towards their worst case analysis. Instead, the SO policy maximizes the mean case performance of the software LIBSVM, see Fan et al. [2005] for a comparison. Thus, the result holds for a case of high practical relevance. Therefore it can hardly be expected to be as general as corresponding results for rate certifying pairs.

We will generalize the progress rate for free SMO steps in that we will drop both preconditions of non-degeneracy and strict positive definiteness of K. In particular, the statements in this chapter cover the case $|\mathcal{R}^*| > 1$. Further, we will consider all iterations performing free steps, not only late iterations $k > \bar{k}$.[1]

The following definition is central for this chapter because it formalizes the way how σ-WSS policies are studied.

Definition 5.5. *We say that a function* $\varphi : \mathcal{R} \setminus \mathcal{R}^* \to \mathbb{R} \cup \{\infty\}$ *has property* $(*)$ *if it is positive and lower semi-continuous. We extend this definition to functions* $\varphi : \mathcal{R} \to \mathbb{R} \cup \{\infty\}$ *which are positive and lower semi-continuous on* $\mathcal{R} \setminus \mathcal{R}^*$*, that is, whose restriction* $\varphi|_{\mathcal{R} \setminus \mathcal{R}^*}$ *has property* $(*)$.

We will use two facts about lower semi-continuous functions. First, a lower semi-continuous function attains its minimum on compact sets. Second, if a lower semi-continuous function is positive in a point α then it is positive in an open neighborhood $U \ni \alpha$.

Lemma 5.6. *The function* ψ *(defined in Section 4.3.7) has property* $(*)$.

Proof. Obviously ψ is continuous on each equivalence class $[\alpha]$. Now, the boundary of a class $[\alpha]$ is the union of those classes $[\alpha']$ with $\mathcal{B}([\alpha']) \subset \mathcal{B}([\alpha])$. Because the argument of the maximum operation in the definition of ψ is a subset of $\mathcal{B}([\alpha])$ on the boundary the maximum can only drop down. Thus, ψ is lower semi-continuous. Further, it is well known that ψ is positive for non-optimal points. □

[1] It is straight forward to apply the analysis by Chen et al. [2006] to earlier iterations performing free steps, but it is not possible to drop the other prerequisites.

From the above lemma together with Lemma 5.2 it directly follows that for any σ-selection W the Newton-step gain $\tilde{g}_W$ is lower bounded by the function $\sigma(\psi(\alpha))^2$ which has property $(*)$.

5.2 Normal Form

We consider only free SMO steps in this chapter. Therefore we can simply drop the box constraints. Then the embedding $\mathcal{R} \subset V$ is much more natural than $\mathcal{R} \subset \mathbb{R}^\ell$. We introduce further important subspaces that are closely related to the structure of the objective function.

Without loss of generality we translate the coordinate system such that $\mathcal{R}^*$ contains the origin. Note that such a transformation does not change the form of problem (4.1). Of course, the values of L_n, U_n and y change and f^* appears as a constant summand in the objective function. For simplicity we will denote the new quantities by the same names and drop the constant summand, which is irrelevant for maximization. Thus, we have $f^* = 0$ by convention.

We decompose the vector space V into an orthogonal direct sum $V = V_K \oplus V_L \oplus V_Z$ explained in the following. Let further $V_\perp = \{\alpha \in \mathbb{R}^\ell \,|\, \beta^T\alpha = 0 \text{ for all } \beta \in V\}$ denote the orthogonal complement of V in $\mathbb{R}^\ell$. We can decompose a vector $\alpha \in \mathbb{R}^\ell$ into its components $\alpha = \alpha_\perp + \alpha_V$ such that $\alpha_\perp \in V_\perp$ and $\alpha_V \in V$. We define the $\ell \times \ell$-matrix $\tilde{K}$ which describes the endomorphism $\tilde{K} : V \to V$, $\alpha \mapsto K\alpha - (K\alpha)_\perp = (K\alpha)_V$. Like K, the matrix $\tilde{K}$ is symmetric and positive semi definite: By construction we have $v^T K v = v^T \tilde{K} v$ for $v \in V$ and $v^T \tilde{K} v = 0$ for $v \in V_\perp$. The spaces $\ker(K) \cap V$ and $\ker(\tilde{K})$ coincide. Analogously we define the vector $\tilde{y} = y - y_\perp = y_V$. Again we have $y^T v = \tilde{y}^T v$ for all $v \in V$.

Let V_K be the sum of all eigenspaces of $\tilde{K}$ in V corresponding to positive eigenvalues. The remaining space $\ker(\tilde{K})$ is decomposed into $V_Z = \{\alpha \in \ker(\tilde{K}) \,|\, \tilde{y}^T\alpha = 0\}$ and $V_L = \{\alpha \in \ker(\tilde{K}) \,|\, \beta^T\alpha = 0 \text{ for all } \beta \in V_Z\}$ which is the orthogonal complement of V_Z in $\ker(\tilde{K})$. For a vector $\alpha \in V$ we define the decomposition $\alpha = \alpha_K + \alpha_L + \alpha_Z$ into its components $\alpha_K \in V_K$, $\alpha_L \in V_L$, and $\alpha_Z \in V_Z$. The vector space V_L is spanned by the single vector $\tilde{y}_L$, which is just the component of $\tilde{y}$ in V_L. If $\tilde{y}_L$ vanishes we get $V_L = \{0\}$. With these conventions it is natural to rewrite the objective function as

$$f(\alpha) = \tilde{y}_L^T\alpha_L + \tilde{y}_K^T\alpha_K - \frac{1}{2}\alpha_K^T\tilde{K}\alpha_K \ . \tag{5.1}$$

This is the desired normal form of the objective function.

From this notation is becomes clear that the objective function as well as its gradient $\nabla f(\alpha) = \tilde{y}_L + \tilde{y}_K - \tilde{K}\alpha_K = \tilde{y} - \tilde{K}\alpha$ are invariant under translation within V_Z. In fact we have $\mathcal{R}^* = V_Z \cap \mathcal{R}$. Further note that the summands $\tilde{y}_L$ and $\tilde{y}_K - \tilde{K}\alpha_K$ of the gradient are orthogonal and therefore can never annihilate each other.

The normal form has a two-fold purpose: First, it focuses on the space $V \subset \mathbb{R}^\ell$ as the canonical embedding of $\mathcal{R}$. Second, the decomposition $V = V_K \oplus V_L \oplus V_Z$ reflects the qualitative behavior of the objective function on the different subspaces: On V_K the objective function is quadratic with strictly negative definite Hessian $-\tilde{K}$, on V_L it is linear with non-vanishing gradient $\tilde{y}_L$, and on V_Z it is constant.

With these tools it is easy to prove Lemma 5.4.

Proof of Lemma 5.4. We have $\mathcal{R}^* = V_Z \cap \mathcal{R}$. Assume $|\mathcal{R}^*| > 1$, then $\mathcal{R}^*$ contains a maximal line segment $\{(1-\mu)\alpha + \mu\alpha' \,|\, \mu \in [0,1]\}$ such that the classes $[\alpha]$, $[\alpha'']$, and $[\alpha']$ are disjoint for $\alpha'' = (1-\mu)\alpha + \mu\alpha'$ with $\mu \in (0,1)$. Because of $[\alpha] \subset \partial[\alpha'']$ the set $\mathcal{B}([\alpha])$ is a proper subset of $\mathcal{B}([\alpha''])$ and we fix an index $n \in \mathcal{B}([\alpha'']) \setminus \mathcal{B}([\alpha])$. We have either $\alpha_n = L_n(\alpha)$ or $\alpha_n = U_n(\alpha)$, imposing one of the inequalities from Definition 5.3

on the gradient $\nabla f(\alpha)$. In contrast, the variable α_n'' is free. As the gradient coincides for all optimal point we have $\nabla f(\alpha'') = \nabla f(\alpha)$. Obviously both of the two possible inequalities on $\nabla f(\alpha)$ contradict the optimality of α'' as we can construct a violating pair involving n. □

5.3 Linear Rate for General Free SMO steps

In the following we will use algebraic properties of the quadratic objective function to define useful invariances. Then only the topological structure of the resulting equivalence classes is needed to apply analytic arguments involving the lower semi-continuity of certain functions.

The following lemma is the core result which directly implies the progress rate for free SMO steps.

Lemma 5.7. *For a σ-selection W the quotient*

$$q(\alpha) = \frac{\tilde{g}_W(\alpha)}{f^* - f(\alpha)}$$

is uniformly bounded from below by a positive constant $R > 0$ for all $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$.

Proof. From the normal form of the objective function it becomes clear that the gradient of the objective function coincides for all $\alpha^* \in \mathcal{R}^*$. It takes the value $\tilde{y}$. In the following we will only consider the set $\mathcal{R} \setminus \mathcal{R}^*$. Therefore we will restrict the equivalence classes $[\alpha]$ to this set, that is, we consider the quotient $(\mathcal{R} \setminus \mathcal{R}^*)/\sim_{\mathcal{B}}$. Note that because of this a class $[\alpha]$ is a polytope minus an affine subspace. We decompose $\mathcal{R} \setminus \mathcal{R}^* = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$ into its disjoint subsets

$$\begin{aligned}
\mathcal{R}_1 &= \left\{ \alpha \in \mathcal{R} \setminus \mathcal{R}^* \;\middle|\; \overline{[\alpha]} \cap \mathcal{R}^* = \emptyset \right\} \\
\mathcal{R}_2 &= \left\{ \alpha \in \mathcal{R} \setminus \mathcal{R}^* \;\middle|\; \overline{[\alpha]} \cap \mathcal{R}^* \neq \emptyset \text{ and } v_B^T \tilde{y} \leq 0 \text{ for all } B \in \mathcal{B}([\alpha]) \right\} \\
\mathcal{R}_3 &= \left\{ \alpha \in \mathcal{R} \setminus \mathcal{R}^* \;\middle|\; \overline{[\alpha]} \cap \mathcal{R}^* \neq \emptyset \text{ and } \exists B \in \mathcal{B}([\alpha]) \text{ with } v_B^T \tilde{y} > 0 \right\} .
\end{aligned}$$

Each of these sets is a union of equivalence classes $[\alpha]$.

The function $q : \mathcal{R} \setminus \mathcal{R}^* \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ inherits property $(*)$ from $\tilde{g}_W$. In the following we will uniformly bound q from below on each of the sets $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. In all three cases we will use that in the compact set $\mathcal{R}$ the complement of every open neighborhood of $\mathcal{R}^*$ is compact and lies in $\mathcal{R} \setminus \mathcal{R}^*$. Then a function with property $(*)$ attains its minimum on this set, which is positive because the function is positive in every point. This minimum is then a lower bound for the function on this set. Thus, to derive a uniform lower bound it remains to bound the function from below in a neighborhood of $\mathcal{R}^*$.

The first and most simple application of this argument is the following one: The set $\mathcal{R}_1$ is compact and its complement $\mathcal{R} \setminus \mathcal{R}_1$ is an open neighborhood of $\mathcal{R}^*$. Therefore, q attains its positive minimum m_1 on this set.

Next we consider the set $\mathcal{R}_2$. We define $\phi : \mathcal{R}_2 \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ as

$$\phi(\alpha) = \frac{2\,\sigma\,(\psi(\alpha))^2}{\alpha^T \tilde{K} \alpha} \quad .$$

The function ϕ inherits property $(*)$ from ψ. Because W is a σ-selection we have $\sigma\,(\psi(\alpha))^2 \leq \tilde{g}_W(\alpha)$. We know from Lemma 4.1 that each equivalence class $[\alpha] \subset \mathcal{R}_2$

is spanned by the directions $D([\alpha]) = \{v_{(i,j)} \,|\, (i,j) \in \mathcal{B}([\alpha]) \text{ and } (j,i) \in \mathcal{B}([\alpha])\}$. We conclude from the definition of $\mathcal{R}_2$ and from $v_B \in D([\alpha]) \Rightarrow -v_B \in D([\alpha])$ that we have $v_B^T \tilde{y} = 0$ for these directions, and therefore $\tilde{y}^T \alpha = 0$. That is, on $\mathcal{R}_2$ the linear term of the normal form of the objective function vanishes and we have $f(\alpha) = -\frac{1}{2}\alpha^T \tilde{K} \alpha$. Then we rewrite the denominator of q as $\frac{1}{2}\alpha^T \tilde{K} \alpha$ and obtain the inequality $\phi(\alpha) \leq q(\alpha)$. The invariance properties of ϕ discussed later on enable us to prove a positive lower bound for ϕ on $\mathcal{R}_2$, which then serves as a lower bound for q.

We collect the directions

$$S = \{v_B \,|\, B \in \mathcal{B} \text{ and } v_B^T \tilde{y} < 0\} \ .$$

By the continuity of the gradient the set $U = \{\alpha \in \mathcal{R} \,|\, v^T \nabla f(\alpha) < 0 \ \forall\, v \in S\}$ is an V_Z-invariant open neighborhood of $\mathcal{R}^*$ of $\mathcal{R}$. We define its subset $U_2 = U \cap \mathcal{R}_2$. By property $(*)$ ϕ attains its positive minimum w on the compact set $\mathcal{R} \setminus U$. On U the directions $v \in S$ are never selected. Thus, we have $v_{W(\alpha)}^T \tilde{y} = 0$ on U_2. In particular, $\max\{v_B^T \tilde{y} \,|\, B \in \mathcal{B}([\alpha])\} = 0$ for all $[\alpha] \subset \mathcal{R}_2$.

If we have even $\tilde{y} = 0$ then this implies $S = \emptyset$, $V_L = \{0\}$, $V_Z = \ker(\tilde{K})$, $U_2 = \mathcal{R}_2$ and $\mathcal{R}_3 = \emptyset$. This is the situation exploited in Theorems 7 and 8 by Chen et al. [2006] for the problem restricted to the equivalence class $[\alpha^*]$ of the unique optimal solution α^*. Then the constant R can be made explicit. We will give an alternative proof in the terms of lower semi-continuity and compactness in the following.

On U_2 we define two equivalence relations: First, $\alpha \sim_Z \alpha'$ if $\alpha \sim_{\mathcal{B}} \alpha'$ and $\alpha - \alpha' \in V_Z$. We denote its classes by $\bar{\alpha}$. To refine this relation we say that $\alpha \sim_M \alpha'$ if $\alpha \sim_{\mathcal{B}} \alpha'$ and there exists $s \in \mathbb{R}^{\neq 0}$ such that $\bar{\alpha} = s \cdot \bar{\alpha}'$.[2] The classes of the latter relation will be denoted by $\tilde{\alpha}$. Now we consider the quotient $Q = U_2 / \sim_M$ of equivalence classes, see Figure 5.1, equipped with the quotient topology. Because we can not separate an equivalence class $[\alpha]$ from its boundary this space is not even Hausdorff, but it is compact:

By the definition of the quotient topology the canonical quotient projection $\pi_M : U_2 \to Q$ is continuous. Let V_2 denote the smallest subspace of V containing U_2 (and thus $\mathcal{R}_2$). This subspace is orthogonal to $\tilde{y}$ and contains $\mathcal{R}^*$. In $\mathcal{R} \cap V_2$ there exists an open neighborhood of $\mathcal{R}^*$ of the form $\{\alpha \in \mathcal{R} \cap V_2 \,|\, d(\alpha, V_Z) < r\} \subset U_2 \cup \mathcal{R}^*$ for some $0 < r < d(\mathcal{R}_1, V_Z)$. Then, the compact set $C = \{\alpha \in \mathcal{R} \cap V_2 \,|\, d(\alpha, V_Z) = r/2\}$ is contained in U_2 and, by construction, intersects all equivalence classes $\tilde{\alpha}$ of $\mathcal{R}_2$. Thus, Q is compact as the continuous projection of this compact set.

The map ϕ has important invariance properties on U_2. It is easy to see that both $\psi(\alpha)$ and $\alpha^T \tilde{K} \alpha$ are invariant under translation within V_Z inside the equivalence class $[\alpha]$. Therefore ϕ is well defined on $U_2 / \sim_Z$. Now, for $\alpha \sim_M \alpha'$ with corresponding $s \in \mathbb{R}^{\neq 0}$ such that $\bar{\alpha} = s \cdot \bar{\alpha}'$ we get $\alpha^T \tilde{K} \alpha = s^2 \cdot \alpha'^T \tilde{K} \alpha'$. For all $B \in \mathcal{B}([\alpha])$ with $v_B^T \tilde{y} = 0$ we get $v_B^T \nabla f(\alpha) = v_B^T \tilde{K} \alpha$ and thus $v_B^T \nabla f(\alpha) = s \cdot v_B^T \nabla f(\alpha')$. Together with $[\alpha] = [\alpha'] \Rightarrow \mathcal{B}([\alpha]) = \mathcal{B}([\alpha'])$ this implies $\psi(\alpha) = s \cdot \psi(\alpha')$. Plugging this into the definition of ϕ we see that the factor s^2 cancels out, resulting in $\phi(\alpha) = \phi(\alpha')$. Thus, ϕ is well defined on the quotient $Q = U_2 / \sim_M$ and we have $\phi(U_2) = \phi(C)$. From the compactness of C (or Q) we conclude again from property $(*)$ that ϕ attains its positive minimum w' on U_2. The positive constant $m_2 = \min\{w, w'\}$ is a lower bound for ϕ on $\mathcal{R}_2$.

On $\mathcal{R}_3$ there is even a uniform lower bound for $\tilde{g}_W$. We consider an equivalence class $[\alpha] \subset \mathcal{R}_3$ and choose $v_{[\alpha]} = v_B$ for some $B \in \mathcal{B}([\alpha])$ such that $v_{[\alpha]}^T \tilde{y} > 0$. By continuity of the gradient we know that there exists a neighborhood $\tilde{U}_{[\alpha]} \subset \mathcal{R}$ of $\mathcal{R}^*$ such that

[2] On each equivalence class $[\alpha]$ the map from $\bar{\alpha} \in U_2 / \sim_Z$ to $\alpha + V_Z \in V / V_Z$ is injective. Therefore, the scalar multiplication in the vector space V/V_Z carries over to the quotient $U_2 / \sim_Z$ and, for appropriate s, the multiplication $s \cdot \bar{\alpha}$ is well defined. $\bar{\alpha} = s \cdot \bar{\alpha}'$ is just a convenient way to express $\alpha - s \cdot \alpha' \in V_Z$. By the way, note that we never have $s = 0$ because $0 \cdot \alpha \in \mathcal{R}^*$. The quotient w.r.t. $\sim_M$ can be thought of as the natural representation of $\mathcal{R}_2$ in the projective space $\mathbb{P}(V/V_Z)$.

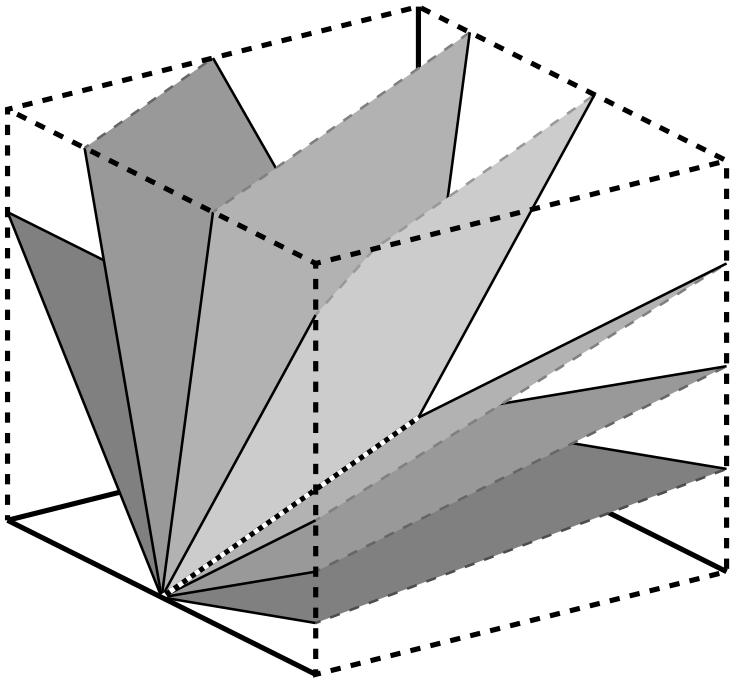

Figure 5.1: Example of the polytope $\mathcal{R}_2$, here as a three dimensional half-open box. For simplicity and because of the low dimension available in a figure we illustrate the case $\tilde{y} = 0$. The set of optimal points is indicated by the dotted line. This line segment lies within the bottom face and ends in one edge and one vertex. The edges belonging to $\mathcal{R}_2$ are drawn solid, while the dashed edges as well as the vertices at their ends belong to $\mathcal{R}_1$. All faces but the top and the front right face touch the optimal line segment and belong to $\mathcal{R}_2$. The figure illustrates the equivalence classes $\tilde{\alpha}$ defined by the relation $\sim_M$. The quotient $Q = U_2/\sim_M$ consists of these equivalence classes. Each gray shaded plane in the inner of the box is one class, as well as the edges of these planes which lie in the box faces belonging to $\mathcal{R}_2$. In contrast to the other edges, these are drawn in black.

$v_{[\alpha]}^T \nabla f(\alpha') > 0$ for all $\alpha' \in \tilde{U}_{[\alpha]} \cap [\alpha]$. Then, $\tilde{U}_{[\alpha]}$ contains a smaller neighborhood $U_{[\alpha]}$ such that $v_{[\alpha]}^T \nabla f(\alpha') > v_{[\alpha]}^T \tilde{y}/2$ for all $\alpha' \in U_{[\alpha]} \cap [\alpha]$ and we have $v_{[\alpha]}^T \nabla f(\alpha') \geq v_{[\alpha]}^T \tilde{y}/2$ for $\alpha' \in \overline{U_{[\alpha]}} \cap [\alpha]$. By construction it holds $\psi(\alpha) \geq v_{[\alpha]}^T \nabla f(\alpha') \geq v_{[\alpha]}^T \tilde{y}/2$. The minimum $w = \min\{v_{[\alpha]}^T \tilde{y}/2 \,|\, [\alpha] \subset \mathcal{R}_3\}$ over the finite set of classes $[\alpha] \subset \mathcal{R}_3$ is positive. By property $(*)$ ψ attains its positive minimum w' on the compact closure of the set

$$\mathcal{R}_3 \setminus \Big(\bigcap_{[\alpha] \subset \mathcal{R}_3} U_{[\alpha]} \Big)$$

Thus, $\min\{w, w'\}$ is a positive lower bound for ψ on $\mathcal{R}_3$. Then $\sigma\,(\min\{w, w'\})^2$ is a lower bound for $\tilde{g}_W$ and $m_3 = \sigma\,(\min\{w, w'\})^2/(f^* - \min\{f(\alpha) \,|\, \alpha \in \mathcal{R}\}) > 0$ is a lower bound for q on $\mathcal{R}_3$.

Finally, $R = \min\{m_1, m_2, m_3\} > 0$ is the desired positive constant. □

Now we are in the position to state a progress rate for free SMO steps:

Theorem 5.8. *Consider problem* (4.1) *and the SMO algorithm with σ-selection W. There exists a constant $c < 1$ such that each free SMO step, starting in $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ and ending in $\alpha' \in \mathcal{R}$ makes progress according to*

$$f^* - f(\alpha') \leq c \cdot (f^* - f(\alpha)) \ .$$

Proof. For a free SMO step we have $\tilde{g}_W(\alpha) = f(\alpha') - f(\alpha)$. With $R > 0$ from Lemma 5.7 and $c = 1 - R$ the result follows from

$$\begin{aligned} & (1-c) = R \leq \frac{g_W(\alpha)}{f^* - f(\alpha)} \\ \Leftrightarrow \ & (1-c)(f^* - f(\alpha)) \leq \tilde{g}_W(\alpha) = f(\alpha') - f(\alpha) \\ \Leftrightarrow \ & f^* - f(\alpha') \leq c \cdot (f^* - f(\alpha)) \ . \end{aligned}$$

□

If we want to apply the result to a specific algorithm in order to obtain an asymptotic linear convergence rate we need to make sure that free SMO steps are carried out in at least a fixed fraction of the iterations. Currently, the only result of this type, as discussed above, makes use of the non-degeneracy assumption. It is not clear whether it is possible to circumvent this assumption, or to replace it with a significantly weaker one. This question, although closely related to this work, remains open. Although this is completely unclear we suspect that in general there is no such rate of free SMO steps, such that a worst case analysis has to impose some preconditions. However, it would be desirable to replace the concept of non-degeneracy with a less restrictive approach.

Finally we want to comment on the utility of this algorithm for proofs of convergence to the optimum (without a linear progress rate). For variants of the SMO-algorithm using non-standard working set selection such a proof can be challenging. Especially for hybrid algorithm using different branches in different situations, even for algorithms depending on additional state information kept beyond a single iteration, it can be helpful to cover the special case that infinitely many of its steps are free SMO steps. In this case the above results can be used to prove the convergence to the optimum. Note that this assumption is extremely robust is the sense that all we need to request from the other (possibly infinitely many) iterations is that they do not reduce the objective function. We do not even need the assumption that they carry out SMO steps at all. We will see an application of this idea in Chapter 7.

Chapter 6

Maximum Gain Policies

In this chapter we will introduce an alternative to the second order (SO) working set selection algorithm by Fan et al. [2005]. We call this algorithm *Hybrid Maximum Gain* (HMG) selection [Glasmachers and Igel, 2006]. The SMO algorithm with this selection is highly efficient for support vector machine training on large-scale problems. In particular we will consider a problem as "large-scale" if the matrix K does not fit into the available cache. At the time of writing this is the case for problems stated by 10,000 training patterns or more, which is common in many application domains.

6.1 Maximum-Gain Working Set Selection

We start with the introduction of *Maximum Gain* (MG) selection. In a way, this policy is the ancestor of the hybrid strategy introduced later on. It realizes two important design principles:

- The MG working set selection was one of the first algorithms using second order information for working set selection. In fact, at the time the algorithm was developed (around 2004/2005) the SO algorithm was not published yet.
- Due to the special set of $\mathcal{O}(\ell)$ candidate working sets considered for selection the number of cache misses of the kernel matrix cache is significantly reduced.

First we turn to a related strategy we will refer to as greedy working set selection. The results for the greedy policy serve as a motivation for the MG selection policy.

6.1.1 The Sparse Witness Property of the Gain

The greedy working set selection picks the working set that achieves the largest immediate gain:

$$W_{\text{greedy}}(\alpha) = \arg\max\{g_B(\alpha) \mid B \in \mathcal{B}(\alpha)\} \ .$$

This is a natural strategy as it is the best we can get without looking into the future (which will be the topic of Chapter 7). However, it takes $\mathcal{O}(\ell^2)$ operations to compute the greedy working set. This is obvious as the whole kernel matrix needs to be evaluated. It is a much better strategy to evaluate just a subset of size $\mathcal{O}(\ell)$ of all working sets because then $\mathcal{O}(\ell)$ iterations can be carried out in the same time a single greedy step takes (which is clearly preferable in practice).

Nevertheless the algorithm is of theoretical interest. We use Theorem 6.2 by List and Simon [2004] to prove the convergence of SMO with greedy working set selection. For this purpose we need the concept of a 2-sparse witness of sub-optimality.

Definition 6.1 (2-sparse witness of sub-optimality, List and Simon [2004])**.** *A family of functions* $(C_B)_{B\in\mathcal{B}}$

$$C_B : \mathcal{R} \to \mathbb{R}^{\geq 0}$$

fulfilling the conditions

(C1) for each $B \in \mathcal{B}$ *the function* C_B *is continuous*

(C2) $C_B(\alpha) = 0$ *if and only if* α *is optimal for the sub-problem induced by* α *and* B

(C3) for each $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ *there exists* $B \in \mathcal{B}(\alpha)$ *such that* $C_B(\alpha) > 0$

is called a 2-sparse witness of sub-optimality.

A 2-sparse witness $(C_B)_{B\in\mathcal{B}}$ of sub-optimality induces a working set selection policy by the rule

$$W_C(\alpha) = \arg\max\{C_B(\alpha) \mid B \in \mathcal{B}(\alpha)\} \ .$$

List and Simon [2004] call this the induced decomposition algorithm. Now we can quote a general convergence theorem for decomposition methods induced by a 2-sparse witness of sub-optimality:[1]

Theorem 6.2 (List and Simon, 2004)**.** *We consider problem* (4.1) *and a 2-sparse witness* $(C_B)_{B\in\mathcal{B}}$ *of sub-optimality. Let* $(\alpha^{(t)})_{t\in\mathbb{N}}$ *denote a sequence of feasible points generated by the SMO algorithm with the working set selection induced by* $(C_B)_{B\in\mathcal{B}}$*, and let* $(\alpha^{(t)})_{t\in S}$*,* $S \subset \mathbb{N}$*, be a convergent sub-sequence with limit point* $\alpha^{(\infty)}$*. Then, the limit point is optimal, that is,* $f(\alpha^{(\infty)}) = f^*$*.*

The following lemma allows for the application of this theorem.

Lemma 6.3. *The gain* $(g_B)_{B\in\mathcal{B}}$ *is a 2-sparse witness of sub-optimality.*

Proof. Property (C1) is proven in Lemma 4.4, while Property (C2) is fulfilled directly per construction of the gain function. Assume Property (C3) is not fulfilled. Then there exists a point $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ (for some instance of problem (4.1) with given ℓ, K, y and lower and upper bounds L_n and U_n) such that no working set allows for any progress. This contradicts the fact that there exist working set selection strategies such that SMO converges (refer to the articles by Lin [2001], Keerthi and Gilbert [2002], Takahashi and Nishi [2005], and numerous others). □

The greedy selection coincides with the working set selection policy induced by the gain as a 2-sparse witness of sub-optimality.

Corollary 6.4. *We consider problem* (4.1)*. Let* $(\alpha^{(t)})_{t\in\mathbb{N}}$ *be a sequence of feasible points produced by the SMO algorithm with the greedy working set selection policy. Then, every limit point* $\alpha^{(\infty)}$ *of a convergent sub-sequence* $(\alpha^{(t)})_{t\in S}$*,* $S \subset \mathbb{N}$*, is optimal.*

Here, we are only interested in the fact that the greedy strategy which maximizes the gain in each single iteration converges to the optimum. It is straightforward to use the more general version of the theorem by List and Simon [2004] to extend the construction for working sets limited in size to some $q > 2$. Then we can generalize this approach to the selection of working sets of size $q = k + 1$ for problems with k equality constraints, resulting in a time complexity of $\mathcal{O}(\ell^{k+1})$ operations for working set selection (which is intractable in most cases, but still polynomial in ℓ).

[1] Theorem 1 in List and Simon [2004] is more general as it is not restricted to problems with a single equality constraint.

6.1.2 The Maximum Gain Algorithm

As noted above the greedy selection takes more than linear time, such that it slows the entire decomposition algorithm down. Therefore, to make use of the above result, we should restrict the greedy search over all $B \in \mathcal{B}(\alpha)$ to a subset of size $\mathcal{O}(\ell)$. This is difficult as there is no canonical subset.

Our choice is guided by the observation of the usage of the kernel matrix as discussed in Section 4.3.8. If we fix one index i of the working set we need the diagonal entries of the kernel matrix and the i-th row to compute the gain $g_{(i,n)}(\alpha)$ for all $1 \leq n \leq \ell$. Of course, we do not want the algorithm to spend its time on costly evaluations of the kernel function. Therefore we should pick an index i such that the corresponding i-th row is already present in the kernel cache. We know that (for any reasonable caching strategy) the two rows corresponding to the most recent working set are available from the cache. Therefore, with $B' = (i', j')$ denoting the working set of the previous iteration, we choose the subset

$$Z(\alpha, (i', j')) = \{(i, j) \in \mathcal{B}(\alpha) \mid \{i, j\} \cap \{i', j'\} \neq \emptyset\}$$

of size $|Z(\alpha, (i', j'))| \leq 2\ell - 1$ and maximize the gain over this subset:

$$W_{\mathrm{MG}}(\alpha, (i', j')) = \arg\max\{g_B(\alpha) \mid B \in Z(\alpha, (i', j'))\} \ .$$

This poses the maximum gain (MG) working set selection strategy.

The MG algorithm selects tuples in consecutive iterations such that one index remains in the working set, even if it changes its position. This property is formally captured by the following definition.

Definition 6.5. *We call two tuples $B = (i, j) \in \mathcal{B}$ and $B' = (i', j') \in \mathcal{B}$ related if $\{i, j\} \cap \{i', j'\} \neq \emptyset$.*

Then MG maximizes the gain over all working sets related to the previous working set. In the first iteration there is no previous working set available. We can simply use the MVP strategy in this case, or alternatively the SO strategy, or even pick a random tuple.

The property to select related working sets in consecutive iterations has an important consequence for the efficiency of the kernel cache. Per assumption the row of the kernel matrix K corresponding to the index that was already present in the previous working set is cached. Therefore in each iteration at most one row of the kernel matrix needs to be computed from the kernel function. For large problems where only a small fraction of the kernel matrix fits in the cache this is a decisive advantage. Usually the rate of cache misses is close to one in the early iterations before the shrinking heuristic has a chance to reduce the problem to a manageable size. The fact that shrinking has not reduced the problem size yet has the second effect that the kernel matrix rows are still very long. In many cases the evaluation of the kernel function then takes a large fraction of the overall computations. For these reasons early SMO iterations can take rather long, compared to the mean iteration time. Then the usage of the MG strategy is a decisive advantage reducing the time for a single iteration by nearly 50%.

6.1.3 Generalization of the Algorithm

In the following, we discuss some of the potential variants of the basic MG algorithm.

It is possible to relax the constraint of selecting a tuple related to the previously selected one. Instead of reusing one of the previously picked indices we could try all combinations with, say, the 10 most recently used working sets. Then the gain is maximized

over a set of at most $20 \cdot \ell$ pairs. In the extreme case we can evaluate all pairs corresponding to cached rows of K. Then the working set selection algorithm becomes quite time consuming in comparison to the gradient update. In simple experiments we observed that the number of iterations does not decrease accordingly. Therefore we recommend to use only the two rows of the matrix from the previous working set. Refer to Section 6.3.5 for a comparison.

A small change speeding up the working set selection is fixing one element of the working set in each iteration. When alternating the fixed position, each element is used two times successively. Only $\ell - 2$ pairs need to be evaluated in each iteration. Though leading to more iterations this policy can bring a minor speed up for small problems (see Section 6.3.5).

The algorithm can be extended to compute the gain for tuples of size $q > 2$. It is a severe disadvantage that such sub-problems can not be solved analytically and an iterative solver (for example, an interior point method) has to be used for the solution of the sub-problem. Note that this becomes necessary also for every gain computation during the working set selection. To keep the complexity of the working set selection linear in ℓ only one element new to the working set can be evaluated. Due to this limitation this method becomes even more cache friendly. The enlarged working set size may decrease the number of iterations required, but at the cost of the usage of an inner solver loop. This should increase the speed of the decomposition algorithm only on large problems with extremely complicated kernels, where the kernel matrix does not fit into the cache and the kernel evaluations in each iteration take much longer than the working set selection.

6.1.4 Convergence of the MG Algorithm

The analysis of the MG algorithm is complicated by the fact that single SMO iterations are not independent. The working set selection depends on the current search point and the previous working set. However, the state space $\mathcal{S} = (\mathcal{R} \setminus \mathcal{R}^*) \times \mathcal{B}$ of the MG algorithm is still manageable.

The SMO algorithm with MG selection is pretty fast in practice. However, there exist rare cases where it gets stuck. This happens if the linear size subset $Z(\alpha, (i', j'))$ does not contain a violating pair. Thus, the MG selection policy does not realize a valid working set selection algorithm in the sense of Definition 4.6. We will demonstrate this problem on a low-dimensional counter-example, showing that in general the SMO algorithm with MG selection does not converge to the optimum.

For $\ell < 3$ the selection of a working set related to the previous working set poses no restriction. Thus, to find a counter example, we have to use $\ell \geq 4$. Indeed, using four training examples is already sufficient. We consider the following instance of problem (4.1) with $C = \frac{1}{10}$ (see Figure 6.1):

$$K = \begin{pmatrix} 2 & \sqrt{3} & -1 & -\sqrt{3} \\ \sqrt{3} & 4 & -\sqrt{3} & -3 \\ -1 & -\sqrt{3} & 2 & \sqrt{3} \\ -\sqrt{3} & -3 & \sqrt{3} & 4 \end{pmatrix} \qquad y = \begin{pmatrix} -1 \\ -1 \\ +1 \\ +1 \end{pmatrix}$$

$$L_1 = -C \quad L_2 = -C \quad L_3 = 0 \quad L_4 = 0$$
$$U_1 = 0 \quad U_2 = 0 \quad U_3 = C \quad U_4 = C$$

The matrix K is positive definite with one-dimensional eigenspace for the eigenvalue 9 and three-dimensional eigenspace for the eigenvalue 1. We assume the previous working set to be $B^{(1)} = (3, 1)$ resulting in the point $\alpha^{(1)} = (-C, 0, C, 0)^T$. This configuration

can indeed occur in SVM training, starting from $\alpha^{(0)} = (0, 0, 0, 0)^T$ and greedily choosing the working set $(3, 1)$ in the first iteration. It is thus possible that the SMO algorithm reaches this state in the SVM context. We compute the gradient

$$\begin{aligned}\nabla f(\alpha^{(1)}) = y - K\alpha^{(1)} &= \left(-\frac{7}{10}, -1 + \frac{\sqrt{3}}{5}, \frac{7}{10}, 1 - \frac{\sqrt{3}}{5}\right)^T \\ &\approx (-0.7, -0.65, 0.7, 0.65)^T\end{aligned}$$

(which is orthogonal to the equality constraint normal vector $\mathbf{1}$). The sub-problems defined by all working sets with exception $B^{(2)} = (4, 2)$ are already optimal, and the working sets $B^{(1)}$ and $B^{(2)}$ are obviously not related. Therefore the MG algorithm can not select a violating pair.

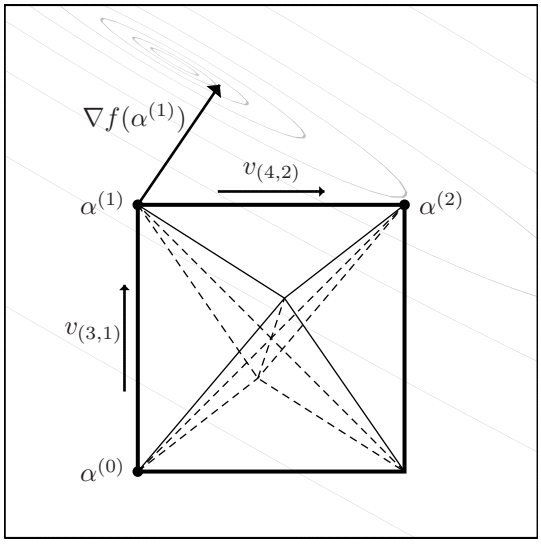

Figure 6.1: Illustration of the counter example. The feasible region $\mathcal{R}$ forms an octahedron within the 3-dimensional space V. The possible search directions are parallel to the edges of the octahedron. The SMO algorithm starts in $\alpha^{(0)} = (0, 0, 0, 0)^T$. During the first two iterations the greedy policy reaches the points $\alpha^{(1)} = (-C, 0, C, 0)^T$ and $\alpha^{(2)} = (-C, -C, C, C)^T$. The MG algorithm gets stuck after the first iteration at $\alpha^{(1)}$. The plane spanned by the directions $v_{(3,1)} = (-1, 0, 1, 0)^T$ and $v_{(4,2)} = (0, -1, 0, 1)^T$ defined by the working sets $(3, 1)$ and $(4, 2)$, respectively, is drawn. The gray ellipses are level sets of the objective function f within this plane. In the point $\alpha^{(1)}$ the gradient $\nabla f(\alpha^{(1)})$ (which lies within the plane) has an angle of less than $\pi/2$ only with the horizontally drawn edge corresponding to the working set $(4, 2)$.

From Figure 6.1 we can see that the same example works for all points on the edge $\alpha^{(1)} = (-C, -a, C, a)^T$ for $a \in [0, C)$. Lemma 6.6 states that indeed the edges of the feasible octahedron are the only candidates for the MG algorithm to get stuck. To be more precise, the above situation can only occur if the previous step moved both active variables to the box boundary.

Lemma 6.6 (see Glasmachers [2008a])**.** *Let $\alpha \in \mathcal{R} \setminus \mathcal{R}^*$ be a non-optimal feasible point and i an index such that the corresponding variable $\alpha_i \in (L_i, U_i)$ is free. Then there exists j such that either $B = (i, j)$ or $B = (j, i)$ is a violating pair. This pair fulfills the*

inequality

$$v_B^T \nabla f(\alpha) \geq \frac{1}{2}\psi(\alpha) = \frac{1}{2}\max\left\{ v_{\tilde{B}}^T \nabla f(\alpha) \;\middle|\; \tilde{B} \in \mathcal{B}(\alpha) \right\} .$$

Proof. Let $\tilde{B} = (m, n)$ be the maximizer of the right hand side, that is, (m, n) is the most violating pair. In particular we have $\alpha_m < U_m$ and $\alpha_n > L_n$. From $v_{(m,i)} + v_{(i,n)} = v_{(m,n)}$ we conclude $v_{(m,i)}^T \nabla f(\alpha) + v_{(i,n)}^T \nabla f(\alpha) = v_{(m,n)}^T \nabla f(\alpha) > 0$ which implies that at least one of the summands is positive. This already shows that one of the pairs (m, i) or (i, n) is violating. We define the value $M = \frac{1}{2}\left(\frac{\partial f}{\partial \alpha_m}(\alpha) + \frac{\partial f}{\partial \alpha_n}(\alpha)\right)$ and write

$$\frac{\partial f}{\partial \alpha_m}(\alpha) = \left(M + \frac{1}{2} v_{(m,n)}^T \nabla f(\alpha) \right) \quad \text{and} \quad \frac{\partial f}{\partial \alpha_n}(\alpha) = \left(M - \frac{1}{2} v_{(m,n)}^T \nabla f(\alpha) \right) .$$

Depending on the derivative $\frac{\partial f}{\partial \alpha_i}(\alpha)$ we distinguish two cases: For $\frac{\partial f}{\partial \alpha_i}(\alpha) \leq M$ the computation

$$v_{(m,i)}^T \nabla f(\alpha) = \frac{\partial f(\alpha)}{\partial \alpha_m} - \frac{\partial f(\alpha)}{\partial \alpha_i} \geq \left(M + \frac{1}{2} v_{(m,n)}^T \nabla f(\alpha) \right) - M = \frac{1}{2} v_{(m,n)}^T \nabla f(\alpha)$$

reveals the desired inequality for $v_{(m,i)}^T \nabla f(\alpha)$ which immediately implies that (m, i) is a violating pair. It is completely analogous to obtain $v_{(i,n)}^T \nabla f(\alpha) \geq \frac{1}{2} v_{(m,n)}^T \nabla f(\alpha)$ with violating pair (i, n) from $\frac{\partial f}{\partial \alpha_i}(\alpha) \geq M$. □

The application of this lemma to the MG selection policy is obvious:

Corollary 6.7. *Let $B' = (i', j')$ be the working set of the previous SMO step resulting in the current point α. If $\alpha_{i'}$ or $\alpha_{j'}$ is free then there exists a violating pair B related to B'.*

Proof. First we consider the case that $\alpha_{i'}$ is free. Then Lemma 6.6 states that there exists n such that either (i', n) or (n, i') is a violating pair. Obviously these pairs are related to (i', j'). The proof is completely analogous if α_j is the free variable. □

6.2 Hybridization with Fall-Back Strategy

The counter example clearly indicates the weakness of the MG algorithm. At the same time Corollary 6.7 makes sure that in the majority of cases progress can be made. With this guidance we will modify the selection in order to inherit the efficiency of the MG algorithm and to ensure convergence to the optimum at the same time.

It is straight forward to use a different selection whenever both variables indexed by the previous working set end at the box boundary. In this situation we use a so-called fall-back strategy. For the case that it should be necessary to use the fall-back strategy in all iterations it makes sense to consider a strategy which is known to make SMO converge to the optimum. In practice such cases are rare. Therefore it does not make a difference whether we use MVP or SO as a fall-back strategy.

To ensure convergence to the optimum it turns out that we need an extension of the above concept. Even if the previous step moved both variables indexed by the previous working set close enough to the box boundary we use the fall-back strategy. This poses the HMG working set selection algorithm, which is formally defined in Algorithm 6.1.

For the training of a 1-norm SVM the distances d_i and d_j equal the product of the scaling parameter η and the regularization parameter C. If in iteration $t > 1$ both variables indexed by the previous working set $B^{(t-1)}$ are no more than $\eta \cdot C$, $0 < \eta \ll 1$,

Algorithm 6.1: Hybrid Maximum-Gain Working Set Selection

Input: current point $\alpha^{(t-1)} \in \mathcal{R}$, iteration $t \in \mathbb{N}$, $0 < \eta \ll 1$
Output: working set $B^{(t)}$
if $t = 1$ **then**
  return the working set according to the fall-back strategy
end
else
  Let $B^{(t-1)} = (i, j)$ denote the previous working set.
  Set $d_i = \eta(U_i - L_i)$ and $d_j = \eta(U_j - L_j)$.
  if $L_i + d_i \leq \alpha_i^{(t-1)} \leq U_i - d_i$ **or** $L_j + d_j \leq \alpha_j^{(t-1)} \leq U_j - d_j$ **then**
    return the working set according to the MG strategy
  end
  else
    return the working set according to the fall-back strategy
  end
end

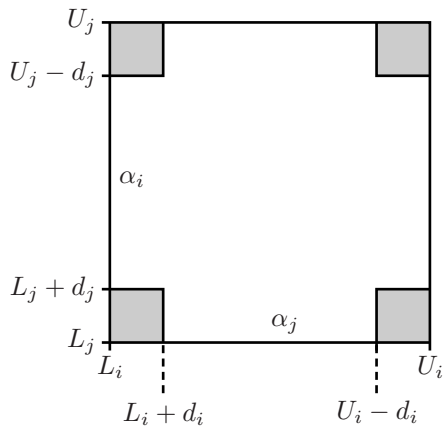

Figure 6.2: Illustration of the HMG algorithm. The plain defined by the previous working set $B^{(t-1)} = (i, j)$ is drawn. If the algorithm ends up in one of the gray corners then the fall-back algorithm is used in iteration t.

from the bounds then the fall-back algorithm is used. Otherwise the working set is selected according to MG. Figure 6.2 illustrates the HMG decision rule.

The parameter η is arbitrarily set to 10^{-8}. As long as this parameter is sufficiently small it should not be considered a parameter of the HMG strategy which is subject to tuning. It merely results from the convergence proof given in the next section. In practice the value of η does not play any role, because

1. the fall-back strategy is used rarely as long as η is small.

2. in the vast majority of cases when the fall-back strategy is used the previous variables are exactly at the box boundary. This typically happens in early iterations when the SMO algorithm moves variables corresponding to training examples with different labels from $(0, 0)$ to $(C, -C)$.

6.2.1 Convergence of the HMG Algorithm

In our convergence analysis we consider perfect accuracy $\varepsilon = 0$. If the SMO algorithm stops then the search point exactly fulfills the KKT conditions and is thus optimal. However, in general, the SMO algorithm does not stop and instead produces an infinite sequence $(\alpha^{(t)})_{t\in\mathbb{N}}$ of feasible points. We will discuss the convergence of this infinite sequence.

The Bolzano-Weierstraß property states that every sequence on a compact set contains a convergent sub-sequence. Because of the compactness of $\mathcal{R}$ the sequence $(\alpha^{(t)})_{t\in\mathbb{N}}$ always contains a convergent sub-sequence denoted $(\alpha^{(t)})_{t\in S}$, $S \subset \mathbb{N}$, with limit point $\alpha^{(\infty)}$. From the construction of the decomposition algorithm it follows that the sequence $(f(\alpha^{(t)}))_{t\in\mathbb{N}}$ increases monotonically, and it is bounded from above by the optimum f^*. It therefore converges and its limit $f(\alpha^{(\infty)})$ does not depend on the choice of the convergent sub-sequence. The crucial question is whether this limit coincides with f^*. The gain sequence $g_{B^{(t)}}(\alpha^{(t-1)}) = f(\alpha^{(t)}) - f(\alpha^{(t-1)})$ is non-negative and converges to zero.

List and Simon [2004] introduced the technical restriction that all principal 2×2 minors of K are positive definite. In our notation this means $v_B^T K v_B > 0$ for all $B \in \mathcal{B}$. For

For Theorem 6.2, which was shown by List and Simon [2004], and for the proof of Lemma 6.9 (and thus of Theorem 6.8) we adopt this requirement. The assumption is not very restrictive because it does not require the whole matrix K to be positive definite. Instead it just requires that the finitely many search directions v_B must not lie in the kernel of K. If in contrast K is indeed positive definite (for example for Gaussian kernels with distinct training examples) then this property is inherited by the principal minors. Usually there is only a zero set of training datasets that violate this condition if a universal kernel is used. This is obviously true if the input space is an open subset of some $\mathbb{R}^n$ and the distribution generating the training data has a density w.r.t. the Lebesgue measure (thus, $\nu(\{x\} \times Y) = 0$ for all $x \in X$).

In this section we will prove the following theorem:

Theorem 6.8. *We consider problem* (4.1) *and assume* $v_B^T K v_B > 0$ *for all* $B \in \mathcal{B}$. *Let* $(\alpha^{(t)})_{t\in\mathbb{N}}$ *be a sequence of points produced by the SMO algorithm using the HMG policy, and let* $(\alpha^{(t)})_{t\in S}$ *for some* $S \subset \mathbb{N}$ *be a convergent sub-sequence. Then the limit point is optimal:*

$$\lim_{\substack{t\to\infty \\ t\in S}} \alpha^{(t)} = \alpha^{(\infty)} \in \mathcal{R}^* \ .$$

Following List and Simon [2004] one can bound the distance $\|\alpha^{(t)} - \alpha^{(t-1)}\|$ in terms of the gain:

Lemma 6.9. *We consider problem* (4.1) *and assume* $v_B^T K v_B > 0$ *for all* $B \in \mathcal{B}$. *Let* $(\alpha^{(t)})_{t\in\mathbb{N}}$ *be a sequence of points produced by the SMO algorithm with an arbitrary working set selection policy. Then, the sequence* $\left(\|\alpha^{(t)} - \alpha^{(t-1)}\|\right)_{t\in\mathbb{N}}$ *converges to* 0.

Proof. The gain sequence $\left(g_{B^{(t)}}(\alpha^{(t-1)})\right)_{t\in\mathbb{N}}$ converges to zero as it is non-negative and its sum is bounded from above. Recall the Taylor expansion

$$f(\alpha^{(t-1)} + \mu v_{B^{(t)}}) = f(\alpha^{(t-1)} + \hat{\mu} v_{B^{(t)}}) - \frac{1}{2}(\mu - \hat{\mu})^2 v_{B^{(t)}}^T K v_{B^{(t)}}$$

and $\alpha^{(t)} - \alpha^{(t-1)} = \mu^* v_{B^{(t)}}$ with $0 < \mu^* \le \hat{\mu}$. From $\|v_{B^{(t)}}\| = \sqrt{2}$ we get $\|\alpha^{(t)} - \alpha^{(t-1)}\| = \sqrt{2}\mu^*$. Plugging $g_{B^{(t)}}(\alpha^{(t-1)}) = f(\alpha^{(t-1)} + \mu^* v_{B^{(t)}}) - f(\alpha^{(t-1)})$ into the Taylor expansion

reveals the gain formula

$$\begin{aligned} g_{B^{(t)}}(\alpha^{(t-1)}) &= \frac{1}{2}((\hat{\mu})^2 - (\mu^* - \hat{\mu})^2)\, v_{B^{(t)}}^T K v_{B^{(t)}} \\ &= \frac{1}{2}(2\mu^*\hat{\mu} - (\mu^*)^2)\, v_{B^{(t)}}^T K v_{B^{(t)}} \\ &\geq \frac{1}{2}(\mu^*)^2\, v_{B^{(t)}}^T K v_{B^{(t)}} \quad . \end{aligned}$$

Let $\lambda_{\min} = \min_{B \in \mathcal{B}}\{\frac{1}{2} v_B^T K v_B\} > 0$ denote the minimal eigenvalue of all 2×2 principle minors of K. It follows

$$\begin{aligned} & g_{B^{(t)}}(\alpha^{(t-1)}) \geq \frac{\lambda_{\min}}{2} \|\alpha^{(t)} - \alpha^{(t-1)}\|^2 \\ \Leftrightarrow \quad & \|\alpha^{(t)} - \alpha^{(t-1)}\| \leq \sqrt{\frac{2}{\lambda_{\min}}\, g_{B^{(t)}}(\alpha^{(t-1)})} \underset{t\to\infty}{\to} 0 \quad . \end{aligned}$$

□

Before we can prove our main result we need the following lemma.

Lemma 6.10. *We consider problem* (4.1) *and assume* $v_B^T K v_B > 0$ *for all* $B \in \mathcal{B}$. *Let* $(\alpha^{(t)})_{t\in\mathbb{N}}$ *be a sequence of points produced by the SMO algorithm using the HMG policy, and let the index set* $S \subset \mathbb{N}$ *correspond to a convergent sub-sequence* $(\alpha^{(t)})_{t\in S}$ *with limit point* $\alpha^{(\infty)}$.

(*i*) *Let*

$$I_S := \left\{ B \in \mathcal{B} \;\middle|\; |\{t \in S \,|\, B^{(t)} = B\}| = \infty \right\}$$

denote the set of working sets selected infinitely often. Then, no gain can be achieved in the limit point $\alpha^{(\infty)}$ *using working sets* $B \in I_S$.

(*ii*) *Let*

$$R_S := \left\{ B \in \mathcal{B} \setminus I_S \;\middle|\; B \text{ is related to some } \tilde{B} \in I_S \right\} \quad .$$

denote the set of working sets related to working sets in I_S. *If the HMG algorithm chooses MG working sets in all iterations* $t+1$ *with* $t \in S$ *then no gain can be achieved in the limit point* $\alpha^{(\infty)}$ *using working sets* $B \in R_S$.

Proof. First, we prove (*i*). Let us assume these exists $B \in I_S$ on which positive gain can be achieved in the limit point $\alpha^{(\infty)}$. Then we have $\varepsilon := g_B(\alpha^{(\infty)}) > 0$. By Lemma 4.4 g_B is continuous. Thus there exists t_0 such that $g_B(\alpha^{(t)}) > \varepsilon/2$ for all $t \in S$, $t > t_0$. Because B is selected infinitely often it follows $f(\alpha^{(\infty)}) = \infty$. This is a contradiction to the existence of the upper bound f^*.

To prove (*ii*) we define the index set $S_{(+1)} := \{t+1 \,|\, t \in S\}$. Using Lemma 6.9 we conclude that the sequence $(\alpha^{(t)})_{t\in S_{(+1)}}$ converges to $\alpha^{(\infty)}$. Let us assume that the limit point can be improved using a working set $B \in R_S$ resulting in $\varepsilon := g_B(\alpha^{(\infty)}) > 0$. Because g_B is continuous there exists t_0 such that $g_B(\alpha^{(t)}) > \varepsilon/2$ for all $t \in S_{(+1)}$, $t > t_0$. By the convergence of the sequence $(f(\alpha^{(t)}))_{t\in\mathbb{N}}$ we find t_1 such that for all $t > t_1$ it holds $g_{B^{(t)}}(\alpha^{(t-1)}) < \varepsilon/2$. The definitions above ensure that there is a working set $\tilde{B} \in I_S$ related to B which is chosen in an iteration $t > \max\{t_0, t_1\}$, $t \in S$. Then in iteration $t+1 \in S_{(+1)}$ due to the MG policy the working set B (or another working set resulting in larger gain) is selected. We conclude that the gain achieved in iteration $t+1$ is greater and smaller than $\varepsilon/2$ at the same time which is a contradiction. Thus, $\alpha^{(\infty)}$ can not be improved using a working set $B \in R_S$. □

Proof of Theorem 6.8. We distinguish two cases. If the MG algorithm is used only finitely often then there exists t_0 such that for all $t > t_0$ the fall-back strategy is used. Per assumption the SMO algorithm with this selection policy converges to the optimum. Otherwise we consider the set

$$T := \{t \in \mathbb{N} \mid \text{MG is used in iteration } t+1\}$$

of iterations after which the MG selection is used. The compactness of $\mathcal{R}$ ensures the existence of a subset $S \subset T$ such that the sub-sequence $(\alpha^{(t)})_{t \in S}$ converges to some limit point $\alpha^{(\infty)}$. With the corresponding sets I_S and R_S defined in the proof of Lemma 6.10 we conclude that $\alpha^{(\infty)}$ can not be improved using working sets $B \in I_S \cup R_S$.

Now let us assume that the limit point can be improved using any other working set. Then Lemma 6.6 implies that all coordinates $\alpha_n^{(\infty)}$ for all n contained in one of the working set $B \in I_S$ are at the bounds. By the definition of the HMG algorithm this contradicts the assumption that the MG policy is used on the whole sequence $(\alpha^{(t)})_{t \in S}$. Thus the limit point $\alpha^{(\infty)}$ can not be improved on any working set. This is equivalent to its optimality because we know that for each non-optimal point there exists a violating pair.

From the strict increase and the convergence of the sequence $(f(\alpha^{(t)}))_{t \in \mathbb{N}}$ it follows that the limit point of every convergent sub-sequence $(\alpha^{(t)})_{t \in \tilde{S}}$ is optimal. □

6.3 Efficiency on Large Scale Problems

This section is devoted to an experimental comparison of different working set selection policies. In order to investigate the cache-friendly HMG strategy the main focus is on large problems.

The experiments were carried out using the software LIBSVM [Chang and Lin, 2001a, Fan et al., 2005]. The version 2.71 implements the MVP algorithm, while the more recent version 2.8 uses SO working set selection. The HMG policy was implemented into LIBSVM to allow for a direct comparison.

We consider radial Gaussian kernel functions (2.1). If not stated otherwise, the SVM was given 40 MB of working memory to store matrix rows. The accuracy of the stopping criterion was set to $\varepsilon = 0.001$. The shrinking heuristic for speeding up the SMO algorithm is turned on. All of these settings correspond to the LIBSVM default configuration.

For the determination of the runtime of the algorithms we used a 1533 MHz AMD Athlon-XP system running Fedora Linux.

In most experiments we measured both the number of iterations needed and the runtime of the algorithm. Although the runtime depends highly on implementation issues and programming skills, this quantity is in the end the most relevant in applications. In this setting the runtime is indeed comparable as only LIBSVM's working set selection was replaced.

The comparison of the working set selection algorithms involves one major difficulty: We use different stopping criteria for the algorithms. This is because in MVP and SO working selection the computation of $\psi(\alpha)$ is a by-product, but for the MG strategy it is not. Therefore we stop the MG algorithm as soon as the Newton step size $\hat{\mu}$ falls below the threshold ε. It is easy to see that, like the standard condition based on ψ, this condition defines a filter for $\mathcal{R}^*$:

$$\bigcap_{\varepsilon > 0} \{\alpha \in \mathcal{R} \mid \hat{\mu}_{W_{\mathrm{MG}}(\alpha)}(\alpha) < \varepsilon\} = \mathcal{R}^* \ .$$

That is, for $\varepsilon \to 0$ the algorithm approximates the optimum arbitrary well. This leads to the situation that the stopping condition used for HMG depends on the usage of MG or

MVP. As the comparability of the runtime depends on an efficient implementation each algorithm in the comparison uses its own stopping criterion. The computation of the final value of the objective function reveals that the two stopping conditions are roughly comparable (see Section 6.3.2 and Table 6.2).

As discussed in Section 4.3.9, the order in which the training examples are presented influences the initial working set and thereby considerably the speed of optimization. Whether a certain ordering leads to fast or slow convergence depends on the working set selection method used. Therefore, we always consider the median over 10 independent trials with different initial working sets drawn i.i.d. if not stated otherwise. In each trial the different algorithms start from the same working set. Whenever we claim that one algorithm requires less iterations or time these results are highly significant (two-tailed Wilcoxon rank sum test, $p < 0.001$).

Besides the overall performance of the working set selection strategies we investigated the influence of a variety of conditions. The experiments compared different values of the Gaussian kernel parameter σ, the regularization parameter C, and the cache size. Finally, we compared variants of the HMG strategy using different numbers of cached matrix rows for working set selection.

6.3.1 Dataset Description

Four benchmark problems were considered. The 60,000 training examples of the MNIST handwritten digit database [LeCun et al., 1998] were split into two classes containing the digits $\{0, 1, 2, 3, 4\}$ and $\{5, 6, 7, 8, 9\}$, respectively. Every digit is represented as a 28×28 pixel array making up a 784 dimensional input space.

The next two datasets are available from the UCI repository [Blake and Merz, 1998]. The spam-database contains 4,601 examples with 57 features extracted from e-mails. There are 1,813 positive examples (spam) and 2,788 negative ones. We transformed every feature to zero mean and unit variance. Because of the small training set, HMG is not likely to excel at this benchmark.

The connect-4 opening database contains 67,557 game states of the connect-4 game after 8 moves together with the labels 'win', 'loose', or 'draw' for the optimal strategy. For binary classification the 'draw' examples were removed resulting in 61,108 data points. Every situation was transformed into a 42-dimensional vector containing the entries 1, 0, or -1 for the first player, no player, or the second player occupying the corresponding field, respectively. The representation is sparse as in every vector only 8 components are non-zero. The data were split roughly into two halves making up training and test data. For the experiments only the training data were used.

The face dataset contains 20,000 positive and 200,000 negative training examples. Every example originates from the comparison of two face images. Two pictures of the same person were compared to generate positive examples, while comparisons of pictures of different persons make up negative examples. The face comparison is based on 89 similarity features. These real-world data were provided by the Viisage Technology AG and are not available to the public.

If not stated otherwise, the hyperparameters C and σ were fixed to values giving well generalizing classifiers. These were determined by a coarse grid search optimizing the error on independent test data. The results of the grid searches are given in Table 6.1.

Training an SVM using the large datasets face and MNIST takes fairly long. Therefore these two problems were not considered in all experiments.

We want to pay special attention to the size of the kernel matrices in comparison to the cache size (see Table 6.1). The datasets cover a wide range of kernel matrix sizes which fit into the cache by nearly 50 % to only 0.02 %. It is a hopeless approach to adapt the cache size in order to fit larger parts of the kernel matrix into working memory. Because

the space requirement for the kernel matrix grows quadratically with ℓ, large scale real world problems exceed any physically available cache.

dataset	dim.	ℓ	cache	σ	C	SV	BSV
spam-database	57	4,601	47.2 %	10	50	18.5 %	11.7 %
connect-4	42	30,555	1.07 %	1.5	4.5	27.0 %	7.7 %
MNIST	784	60,000	0.28 %	3,500	50	10.5 %	4.6 %
face	89	220,000	0.02 %	3	5	2.6 %	1.2 %

Table 6.1: SVM parameters used in the comparison together with solution statistics. The column "dim." gives the input space dimension while ℓ is the number of training examples. The "cache"-column shows how much of the kernel matrix fits into the kernel cache. The fractions of support vectors and bounded support vectors are denoted by "SV" and "BSV". These percentage values can slightly differ between the algorithms because of the finite accuracy of the solutions.

6.3.2 Comparison of Working Set Selection Strategies

We trained 1-norm SVMs on all datasets presented in the previous section. We monitored the number of iterations and the time until the stopping criterion was met. The results are shown in Table 6.2. The final target function values $f(\alpha^*)$ are also presented to prove the comparability of the stopping criteria (for the starting state it holds $f(\alpha^{(0)}) = 0$). Indeed, the final values are extremely close and which algorithm is most accurate depends on the problem.

dataset (ℓ)	algorithm	iterations	runtime	$f(\alpha^*)$
spam-database (4,601)	MVP	36,610	11.21 s	27,019.138
	SO	**9,228**	**8.44** s	27,019.140
	HMG	10,563	9.17 s	27,019.140
connect-4 (30,555)	MVP	65,167	916 s	13,557.542
	SO	**45,504**	734 s	13,557.542
	HMG	50,281	**633 s**	13,557.536
MNIST (60,000)	MVP	185,162	13,657 s	143,199.142
	SO	**110,441**	9,957 s	143,199.146
	HMG	152,873	**7,485 s**	143,199.160
face (220,000)	MVP	37,137	14,239 s	15,812.666
	SO	**32,783**	14,025 s	15,812.666
	HMG	42,303	**11,278 s**	15,812.664

Table 6.2: Comparison of the number of iterations of the decomposition algorithm and training times for the different working set selection approaches. In each case the best value is highlighted. The differences are highly significant (Wilcoxon rank sum test, $p < 0.001$). Additionally, the final value of the objective function showing the comparability of the results is given.

It becomes clear from the experiments that the MVP algorithm performs worst. This is no surprise because it does not take second order information into account. In the following we will concentrate on the comparison of the second order algorithms SO and HMG.

As the smallest problem considered, the spam-database consists of 4,601 training examples. The matrix K requires about 81 MB of working memory. The cache size of

40 MB should be sufficient when using the shrinking technique. The SO algorithm profits from the fact that the kernel matrix fits into the cache after the first shrinking event. It takes less iterations and (in the mean) the same time per iteration as the HMG algorithm and is thus the fastest in the end.

In the connect-4 problem the kernel matrix does not fit into the cache even if shrinking is used to reduce the problem size. Thus, even in late iterations kernel evaluations can occur. Here, HMG outperforms the old and the new LIBSVM algorithm. This situation is even more pronounced for the MNIST dataset and the face problem. Only a small fraction of K fits into the cache making expensive kernel evaluations necessary. Note that for all of these large problems the SO algorithm minimizes the number of iterations while HMG minimizes the training time. The HMG algorithm is the fastest on the three large scale problems, because it makes use of the kernel cache more efficiently.

6.3.3 Analysis of Different Parameter Regimes

The choice of the regularization parameter C and the parameter σ of the Gaussian kernel (2.1) influence the quadratic problem induced by the data. We analyzed this dependency using grid search on the connect-4 problem. The results are plotted in Figure 6.3.

All parameter configurations where MVP or SO outperform HMG have one important property in common, namely, that it is a bad idea to reselect an element of the previous working set. This is true when after most iterations both coordinates indexed by the working set are already optimal. This can happen for different reasons:

- For $\sigma \to 0$, that is, for very narrow kernels, the feature vectors corresponding to the training examples become more and more orthogonal and the quadratic problem (4.1) becomes (almost) separable.
- For increasing values of σ the example points become more and more similar in the feature space until they are hard to distinguish. This increases the conditioning number of the kernel matrix K. Thus, the solution of problem (4.1) is likely to lie in a corner or on a low dimensional edge of the box constraining the problem, that is, many components of the optimal solution are at the boundary. This is even more likely for small values of C.

We argue that in practice parameter settings leading to those situations are not really relevant because they tend to produce degenerate solutions. Either almost all examples are selected as support vectors (and the SVM converges to 1-nearest-neighbor classification) or the information available are used inefficiently, setting most support vector coefficients to C. Both extremes are usually not intended in SVM learning. In our experiments, HMG performs best in the parameter regime giving well generalizing solutions.

6.3.4 Influence of the Cache Size

The speed (in terms of runtime, not iterations) of the SVM algorithm depends on the fraction of matrix rows fitting into the cache. We used the connect-4 dataset to test the dependency of speed on the cache size. The full representation of the matrix K requires nearly 3.5 GB of working memory for this problem. We trained SVMs with 20 MB (0.56% of the matrix), 40 MB (1.12%), 100 MB (2.8%) and 200 MB (5.6%) cache. Because the shrinking heuristic reduces the amount of memory required for the storage of the relevant part of K, the percentage values should be viewed with care.

If all variables ending up at the box constraints are removed, the storage size of the matrix K is about 134 MB. This matrix already fits into the 200 MB cache.

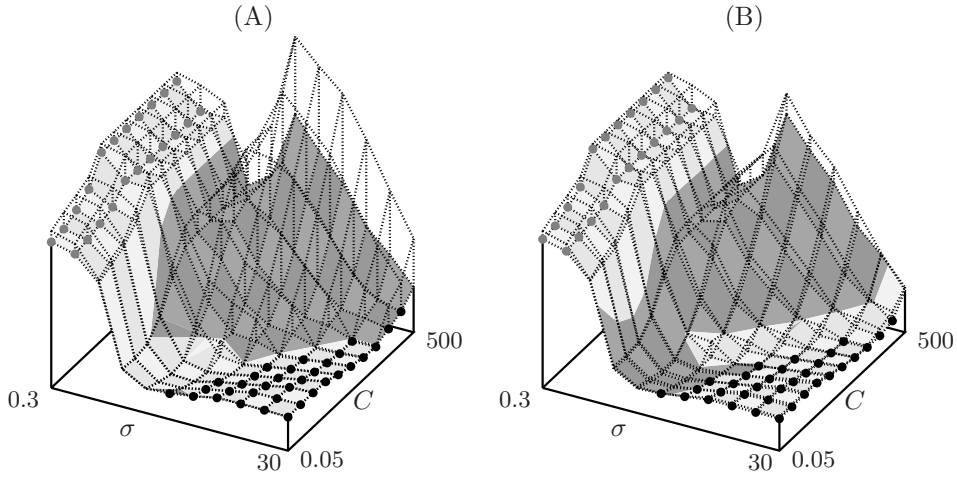

Figure 6.3: Influence of the hyperparameters C and σ on the runtime for the connect-4 dataset. The plots show on logarithmic scales the runtime of SVM training depending on C and σ. The comparison of HMG to MVP is plotted in (A), while plot (B) shows the comparison of HMG to SO. The shaded shapes indicate the method needing less runtime, in light and dark gray for MVP/SO and HMG, respectively. Only the lower surface corresponding to the faster algorithm is drawn solid while the higher surface is indicated by the dotted grid lines. Both $\gamma = 1/(2\sigma^2)$ and C are considered in a range of factor $10,000$ containing the well generalizing parameter regime, see Table 6.2. The dots mark degenerate (and thus not desirable) solutions. The gray dots indicate that the solution uses at least 99 % of the training data as support vectors. If at least 99 % of the support vectors are at the upper bound C the solution is marked with a black dot.

cache size	MVP	SO	HMG
20 MB	958 s	766 s	656 s
40 MB	916 s	734 s	633 s
100 MB	758 s	649 s	583 s
200 MB	603 s	547 s	555 s

Table 6.3: The training time for the connect-4 task for the different working set selection algorithms depending on the cache size.

The results listed in Table 6.3 and plotted in Figure 6.4 clearly show that for small cache sizes the HMG algorithm is advantageous while for a large cache the SO algorithm catches up.

This result coincides with the expectations. As long as there is a considerable chance to find a matrix row in the cache it is not necessary to use the cache friendly HMG strategy. In this case it is reasonable to minimize the number of iterations. This is best achieved by the SO algorithm. If the cache is too small to store a relevant part of the kernel matrix it becomes advantageous to use HMG, because HMG results in at most one cache miss per iteration. Therefore the HMG algorithm should be used for large scale problems.

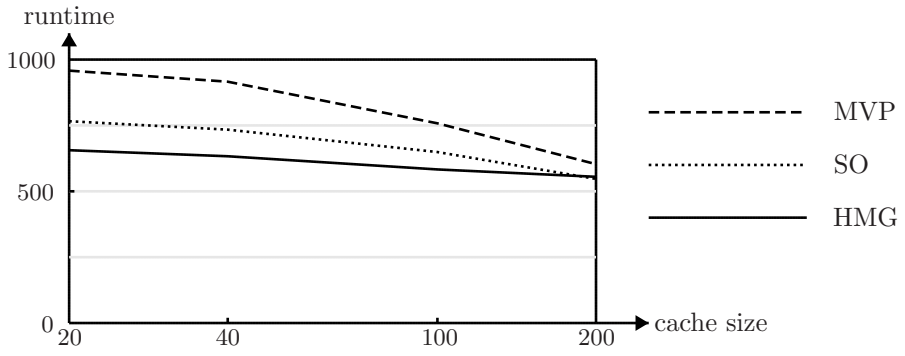

Figure 6.4: The training time in seconds for the connect-4 dataset is plotted over the cache size in MB.

6.3.5 Number of Matrix Rows Considered

In the definition of the HMG algorithm we restrict ourselves to computing the gain using the two cached matrix rows corresponding to the previous working set. This seems to be an arbitrary restriction. To determine the influence of the number of rows considered we compared the HMG algorithm to two modified versions.

We computed the gain using only one matrix row corresponding to one element of the working set. The element chosen was alternated in every iteration such that a selected variable was used in exactly two successive iterations. This policy reduces the time required for the working set selection by about 50%. It can be considered as the minimal strategy avoiding asymmetries between the variables. The results comparing the usage of one and two rows are shown in Table 6.4.

	two rows			one row		
dataset	iterations	runtime	$f(\alpha^*)$	iterations	runtime	$f(\alpha^*)$
spam-database	**10,563**	**9.17 s**	27,019.140	14,023	10.99 s	27,019.134
connect-4	**50,281**	**633 s**	13,557.536	83,305	4,967 s	13,557.529

Table 6.4: Comparison of the one-row and the two-row strategy. The better values are printed in bold face. The differences are highly significant (Wilcoxon rank sum test, $p < 0.001$). The final target function values are lower using the one-row strategy.

Although the stopping criteria used are the same we get different final target function values. This happens due to the reduced number of pairs over which the maximum is taken in the one-row strategy. The values listed indicate that the experiments are roughly comparable, but the one-row strategy produces less accurate solutions in general.

The two-row strategy performed clearly better. The reasons for the poor performance of the one-row strategy are the higher number of iterations and, what is worse, the far higher number of unshrinking events. Because of the reduced amount of pairs considered this strategy is endangered to wrongly detect optimality on the shrunk problem causing a costly unshrinking process. Simple experiments indicated that the one-row strategy can compete if the problem is small. However, in this case both HMG strategies were outperformed by the SO algorithm.

The other strategy tested is to compute the gain for every cached matrix element

available. Of course this algorithm is extremely time consuming and thus not practical for applications. This test gives us the minimum number of iterations the HMG working set algorithm can achieve, as it is the strategy using all information available. It thus provides a bound on the performance of possible extensions of the algorithm using more than two cached matrix rows to determine the working set. In the case where the whole matrix K fits into the cache and all rows have already been computed, the strategy coincides with the exact greedy policy w.r.t. the gain. We compared the results to the two-row strategy, see Table 6.5.

	two rows		whole cache		iterations
dataset	iterations	$f(\alpha^*)$	iterations	$f(\alpha^*)$	saved
spam-database	10,563	27,019.140	7,280	27,019.143	31 %
connect-4	50,281	13,557.536	51,285	13,557.538	0 %

Table 6.5: Comparison of the strategies using two matrix rows and the whole matrix cache available.

Again it is difficult to compare the experiments because the algorithm using all cached rows available generally stops later than the two-row strategy. We will nevertheless interpret the results, although this difficulty indicates that the bound on the possible performance of HMG algorithms using more than two rows may not be tight.

The behavior is obviously problem specific. On the connect-4 task both strategies nearly showed the same performance. This suggests that on some problems the two-row strategy cannot be outperformed by HMG strategies using more than two matrix rows. In contrast for the spam-database, the whole-cache strategy saved 31 percent of the iterations. This is a considerable amount which was bought dearly using all cached matrix rows for the working set selection. In practice one would like to use a faster method, which, for example, looks at a small fixed number of matrix rows. The reduction of the number of iterations will presumably be less pronounced for such strategies. Additionally, the non-trivial question for the row selection policy arises. Thus, for simplicity as well as performance we stick to the two-row HMG algorithm.

Chapter 7

Planning Ahead

Due to its quadratic complexity, greedy working set selection seems to set a clear limit for the single step performance of the SMO algorithm. The working set selection strategies discussed in the previous chapters pose heuristics that try to get close to the greedy policy in linear time. In this chapter we will develop an improvement of another part of the SMO loop. We will investigate the size of the SMO step, usually resulting from the solution of the sub-problem given by the current solution and the working set. The solution of the sub-problem as discussed in Section 4.3.3 is clearly optimal in a greedy point of view. However, if we consider the progress made in several consecutive iterations this step size may turn out to be sub-optimal.

In contrast to working set selection there has not been any work on the improvement of the step size of the SMO algorithm. Of course, it is not possible to considerably speed up a computation taking $\mathcal{O}(1)$ operations, but we will see in the following how we may replace the optimal (greedy) truncated Newton step (4.4) with another approach. The presentation in this chapter follows the article by Glasmachers [2008b].

7.1 Computationally Cheap Second Order Optimization

As discussed in Section 4.3 it turns out that for most SVM training problems a minimal working set size is beneficial. This means that the decomposition algorithm can choose its optimization directions very frequently, while in each iteration it takes only a minimal amount of information on the dependencies between variables (in terms of second derivatives $\frac{\partial^2 f(\alpha)}{\partial \alpha_i \partial \alpha_j} = -K_{ij}$ of the objective function) into account. We want to stick to this successful strategy, but at the same time we want to countervail an undesired effect which can be observed in simple experiments.

7.1.1 Typical Behavior of the SMO Algorithm

As a motivation for the following computations we need to make some statements about the overall behavior of the SMO algorithm.

From an empirical point of view we can describe the qualitative behavior of SMO roughly as follows. In the early iterations many steps move variables α_n to the lower or upper bounds L_n and U_n. After a while, these steps become rare and most iterations are spent on a relatively small number of variables performing free steps. In this phase the shrinking heuristics removes most bounded variables from the problem. Then working

set selection, gradient update and stopping condition need to be computed only on the relatively small set of active variables, leading to extremely fast iterations.

Many common benchmark problems and real world applications are simple in the sense that the algorithm performs only a number of iterations in the order of magnitude of ℓ to achieve the standard accuracy of $\varepsilon = 0.001$. In this case only few variables (if any) are changed many times. This indicates that there are only few free support vectors or that the dependencies between variables are weak, making SMO optimization easy. On the other hand, for harder problems the algorithm spends most of its iterations on free SMO steps to resolve complicated dependencies between the free variables. In fact, in some cases we can observe large blocks of iterations spent on a small number of variables. With MVP working set selection this behavior is even more pronounced than with the second order variants.

Due to its finite number of optimization directions the SMO algorithm is prone to oscillate while compensating the second order cross terms of the objective function. The presence of the box constraints further complicates this problem. In fact we can observe how the algorithm sometimes oscillates for many (even thousands of) iterations until one variable finally hits the boundary. This oscillatory behavior observed in case of difficult problems is the main motivation for the consideration presented below.

7.1.2 The Planning Step

Without loss of generality we consider the iteration $t = 1$ in this section to keep the notation simple.

Assume we are given the current working set $B^{(1)} = (i^{(1)}, j^{(1)})$ and for some reason we already know the working set $B^{(2)} = (i^{(2)}, j^{(2)})$ to be selected in the next iteration. In addition, we presume that the solutions of both sub-problems involved are not at the bounds. That is, we can simply ignore the box constraints. With the quantities

$$\begin{aligned} Q_{tt} &= v_{B^{(t)}}^T K v_{B^{(t)}} \\ l_t &= v_{B^{(t)}}^T \nabla f(\alpha^{(t-1)}) \\ \tilde{L}_t &= \max\{L_{i^{(t)}} - \alpha_{i^{(t)}}^{(t-1)}, \alpha_{j^{(t)}}^{(t-1)} - U_{j^{(t)}}\} \\ \tilde{U}_t &= \min\{U_{i^{(t)}} - \alpha_{i^{(t)}}^{(t-1)}, \alpha_{j^{(t)}}^{(t-1)} - L_{j^{(t)}}\} \end{aligned}$$

for $t \in \{1, 2\}$ we know from equation (4.6) with $\hat{\mu}_1 = l_1/Q_{11}$ and $\hat{\mu}_2 = l_2/Q_{22}$ that both free steps together result in the gain

$$g^{\text{2-step}} := f(\alpha^{(2)}) - f(\alpha^{(0)}) = \frac{1}{2} Q_{11} \hat{\mu}_1^2 + \frac{1}{2} Q_{22} \hat{\mu}_2^2 \ . \tag{7.1}$$

Under the assumption that we already know the working set $B^{(2)}$ we can of course precompute the second step. To stress this point of view we introduce the quantities

$$w_t = v_{B^{(t)}} \nabla f(\alpha^{(0)}) \qquad \text{for } t \in \{1, 2\}$$

which only depend on $\alpha^{(0)}$ (and not on $\alpha^{(1)}$) and are thus known in iteration $t = 1$. We rewrite

$$\begin{aligned} l_1 &= w_1 \\ l_2 &= w_2 - Q_{12} \mu_1 \\ \text{with} \quad Q_{12} &= Q_{21} = K_{i^{(1)} i^{(2)}} - K_{i^{(1)} j^{(2)}} - K_{j^{(1)} i^{(2)}} + K_{j^{(1)} j^{(2)}} = v_{B^{(1)}}^T K v_{B^{(2)}} \ . \end{aligned}$$

Then we can express the step size

$$\hat{\mu}_2 = l_2/Q_{22} = w_2/Q_{22} - Q_{12}/Q_{22}\mu_1 \tag{7.2}$$

in these terms. The above notation suggests the introduction of the 2×2 matrix

$$Q = \begin{pmatrix} Q_{11} & Q_{21} \\ Q_{12} & Q_{22} \end{pmatrix}$$

which is symmetric and positive semi-definite. If we drop the assumption that both steps involved are Newton steps the computation of the gain gets slightly more complicated:

$$g^{\text{2-step}}(\mu_1, \mu_2) := f(\alpha^{(2)}) - f(\alpha^{(0)}) = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}^T \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}^T Q \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \ .$$

Plugging everything in, and in particular substituting μ_2 with $\hat{\mu}_2$ according to equation (7.2) we express the gain as a function of the single variable μ_1 resulting in

$$g^{\text{2-step}}(\mu_1) = -\frac{1}{2} \cdot \frac{\det(Q)}{Q_{22}} \mu_1^2 + \frac{Q_{22}w_1 - Q_{12}w_2}{Q_{22}} \mu_1 + \frac{1}{2} \cdot \frac{w_2^2}{Q_{22}} \ . \tag{7.3}$$

For $\mu_1 = w_1/Q_{11} = l_1/Q_{11}$ we obtain the gain computed in (7.1), but of course the maximizer of the quadratic function (7.3) will in general differ from this value. Thus, under the above assumption that we already know the next working set $B^{(2)}$ we can achieve a better functional gain by computing the optimal planning-ahead step size

$$\mu_1^{\text{pa}} = \frac{Q_{22}w_1 - Q_{12}w_2}{\det(Q)} \tag{7.4}$$

where we again assume that we do not hit the box constraints which are dropped. It is easy to incorporate the constraints into the computation, but this has two drawbacks in our situation: It leads to a large number of different cases and, much worse, it complicates the convergence proof. Therefore the algorithms resulting from these considerations have to handle the box constrained case separately.

Figure 7.1 illustrates the resulting step. We call the step $\mu_1^{\text{pa}} \cdot v_{B^{(1)}}$ the planning-ahead step, because we need to simulate the current and the next step in order to determine the step size μ_1^{pa}. Analogously we refer to μ_1^{pa} as the planning-ahead step size.

Just like the usual SMO update this step can be computed in constant time, that is, independent of the problem dimension ℓ. However, the kernel values of an up to 4×4 principal minor of the kernel Gram matrix K are needed for the computation, in contrast to a 2×2 minor for the standard SMO update step. Thus, this approach takes more second order cross terms of the objective function into account. To be more precise, additional to the 2×2 sub-matrices involved in the solution of the independent sub-problems in two subsequent standard SMO iterations this approach honors the dependency between the two search directions. Alternatively one could directly solve the up to four-dimensional sub-problem induced by the union of the working sets. In general this proceeding would result in an even larger gain, but at the cost of a less frequent working set selection leading to poorer search directions. This is because, in contrast to our initial assumption, we can of course not know the working set $B^{(2)}$ selected in the next iteration.

Note the asymmetry of the functional gain as well as the optimal step size w.r.t. the iteration indices 1 and 2. We control the length of the first step, which influences the length of the second step. The asymmetry results from the fact that the size of the second step is chosen greedily in contrast to the first one. The first step is optimal given the next working set $B^{(2)}$ and planning one step ahead, while the second step is optimal in the usual sense of doing a single greedy step without any planning-ahead.

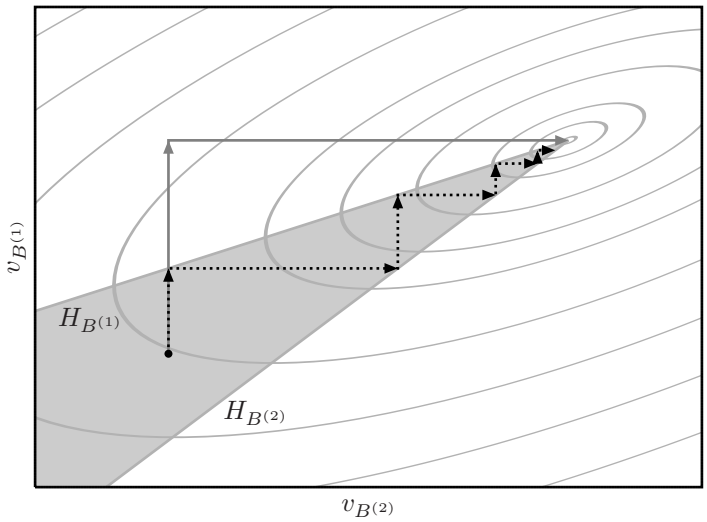

Figure 7.1: Optimization path with (dark gray) and without (dotted) planning ahead in a minimal scenario composed of only two possible working sets $B^{(1)}$ and $B^{(2)}$. The light gray ellipses indicate level sets of the objective function f. From these it is obvious that the first step of the dark gray path results in a decrease of the objective function. While the usual SMO procedure oscillates inside a cone bounded by the hyperplanes $H_{B^{(t)}}$ (see Section 4.3.3), planning ahead one step results in the optimal step size to solve this low dimensional problem.

Another interesting property is that for $\mu_1^{\text{pa}} \notin [0, 2\,l_1/Q_{11}]$ the first step actually results in a decay of the objective function value, that is, $f(\alpha^{(1)}) < f(\alpha^{(0)})$, see Figure 7.2. Nevertheless, such steps can be extremely beneficial in some situations, see Figure 7.1. Of course, by construction both planned-ahead steps together result in an increase of the objective function, which is even maximized for the given working sets.

7.1.3 Algorithms

In this section we will turn the above consideration into algorithms. We will present a first simple version and a refinement which focuses on the convergence of the overall algorithm to an optimal solution. These modifications are all based on the SMO Algorithm 4.2. Thus, we will only state replacements for the working set selection step and the solution of the sub-problem.

In the previous section it is left open how we can know the choice of the working set in the forthcoming iteration. If we try to compute this working set given a step size μ it turns out that we need to run the working set selection algorithm. That is, the next working set depends on the current step size and it takes linear time to determine the working set for a given step size. This makes a search for the optimal combination of step size and working set impractical.

We propose a simple heuristic instead. For two reasons we suggest to re-use the previous working set: First, the chance that the corresponding kernel evaluations are cached is highest for this working set. Second, as already stated in Section 7.1.1, the SMO algorithm sometimes tends to oscillate within a small number of variables. Figure 7.1 gives a low-dimensional example. Now, if we are in a phase of oscillation, the previous working set is a good candidate for planning-ahead.

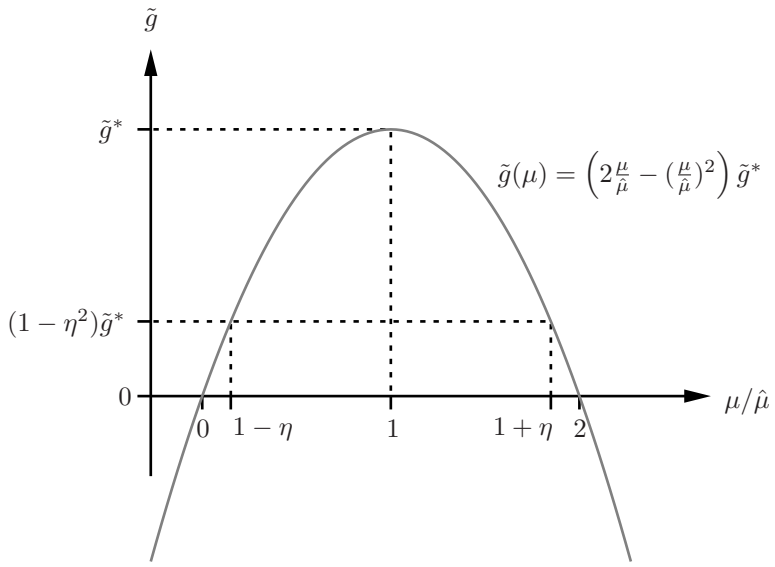

Figure 7.2: Gain of the step with size μ compared to the Newton step size $\hat{\mu}$. As long as the quantity $\mu/\hat{\mu}$ is in the open interval $(0, 2)$ the step makes some progress. For $\eta < 1$ and $\mu/\hat{\mu} \in [1-\eta, 1+\eta]$ the gain is even strictly lower bounded by a fraction of $1-\eta^2$ of the Newton step gain.

Algorithm 7.1: Modification of step 2 of the SMO algorithm

```
if previous iteration performed a SMO step (eq. (4.4)) then
    Compute the planning-ahead step size μ_t^pa (eq. (7.4))
          assuming B^(t−1) as the next working set
    if the current or the planned step ends at the box boundary then
        perform a SMO step (eq. (4.4))
    else
        perform the step of size μ_t^pa as planned
    end
else
    perform a SMO step (eq. (4.4))
end
```

These considerations result in a new algorithm. It differs from the SMO algorithm only in the solution of the sub-problem. The basic idea is to use the previous working set for planning-ahead. However, we revert to the standard SMO step if the previous step was used for planning-ahead or the planned steps are not free. This proceeding is formalized in Algorithm 7.1, which is a replacement for the solution of the sub-problem (step 2) in Algorithm 4.2.

As already indicated in Section 7.1.2 the algorithm uses planning-ahead only if both steps involved do not hit the box boundaries. This means that we need to check the box constraints while planning ahead, and we turn to the standard SMO algorithm whenever the precomputed steps become infeasible. Thus, there is no need to incorporate the box constraints into the planning-ahead step size given in equation (7.4).

The algorithm works well in experiments. However, it is hard to prove its convergence

Algorithm 7.2: Modification of step 1 of the SMO algorithm

Input: $\eta \in (0,1)$
Input: μ_{t-1}: step size of the previous iteration $t-1$
Input: $\hat{\mu}_{t-1}$: Newton step size of the previous iteration $t-1$
if *previous step was a standard SMO step* **then**
  `// use SO selection, see equation` (4.7)
  $i^{(t)} \leftarrow \arg\max\{\frac{\partial f}{\partial \alpha_n}(\alpha^{(t-1)}) \mid n \in I_{\text{up}}(\alpha^{(t-1)})\}$
  $j^{(t)} \leftarrow \arg\max\{\tilde{g}_{(i^{(t)},n)}(\alpha^{(t-1)}) \mid n \in I_{\text{down}}(\alpha^{(t-1)})\}$
  $B^{(t)} \leftarrow (i^{(t)}, j^{(t)})$
else
  if $1-\eta \leq \mu_{t-1}/\hat{\mu}_{t-1} \leq 1+\eta$ **then**
    `// SO selection with additional candidate` $B^{(t-2)}$
    $i^{(t)} \leftarrow \arg\max\{\frac{\partial f}{\partial \alpha_n}(\alpha^{(t-1)}) \mid n \in I_{\text{up}}(\alpha^{(t-1)})\}$
    $j^{(t)} \leftarrow \arg\max\{\tilde{g}_{(i^{(t)},n)}(\alpha^{(t-1)}) \mid n \in I_{\text{down}}(\alpha^{(t-1)})\}$
    $B^{(t)} \leftarrow (i^{(t)}, j^{(t)})$
    if $\tilde{g}_{B^{(t-2)}}(\alpha^{(t-1)}) > \tilde{g}_{B^{(t)}}(\alpha^{(t-1)})$ **then**
      $B^{(t)} \leftarrow B^{(t-2)}$
    end
  else
    `// selection with additional candidate` $B^{(t-2)}$
    `// based on` g `instead of` $\tilde{g}$
    $i^{(t)} \leftarrow \arg\max\{\frac{\partial f}{\partial \alpha_n}(\alpha^{(t-1)}) \mid n \in I_{\text{up}}(\alpha^{(t-1)})\}$
    $j^{(t)} \leftarrow \arg\max\{g_{(i^{(t)},n)}(\alpha^{(t-1)}) \mid n \in I_{\text{down}}(\alpha^{(t-1)})\}$
    $B^{(t)} \leftarrow (i^{(t)}, j^{(t)})$
    if $g_{B^{(t-2)}}(\alpha^{(t-1)}) > g_{B^{(t)}}(\alpha^{(t-1)})$ **then**
      $B^{(t)} \leftarrow B^{(t-2)}$
    end
  end
end

to an optimal solution for a number of reasons. The main difficulty involved is that the planning-ahead step together with the subsequent iteration is not guaranteed to improve the search point. This problem basically results from the lack of knowledge of the outcome of the next working set selection. If we would know the next working set both steps together are known to have positive gain $g^{\text{2-step}}$. As we do not know the next working set but assume that the previous one is selected in the next iteration we can improve the situation by replacing the working set selection (step 1) of Algorithm 4.2 with Algorithm 7.2. For simplicity, this working set selection is based on the highly efficient SO working set selection.

At a first glance this algorithm looks more complicated than it is. The selection basically ensures that the planning-ahead step and the next SMO step together have a positive gain: Recall that for $\mu_{t-1}/\hat{\mu}_{t-1} \in [1-\eta, 1+\eta]$ the planning step itself makes some progress (see Figure 7.2). The following SMO step has always positive gain. Now consider the case that the planning-step does not make a guaranteed progress, that is, $\mu_{t-1}/\hat{\mu}_{t-1} \notin [1-\eta, 1+\eta]$. The planned double-step gain (7.3) is by construction lower bounded by the Newton step gain. Thus, if the previous working set is re-used in the following iteration the total gain is positive. Now the usage of the SMO gain g instead of the Newton step gain $\tilde{g}$ for working set selection ensures that this gain can only increase if another working set is actually selected in the step following planning-ahead. Thus,

Algorithm 7.3: Modification of step 2 of the SMO Algorithm. The only difference compared to Algorithm 7.1 is the first condition that the SMO step must be *free*.

if *previous iteration performed a free SMO step* **then**
  Compute the planning-ahead step size μ_t^{pa} (eq. (7.4))
    assuming $B^{(t-1)}$ as the next working set
  if *the current or the planned step ends at the box boundary* **then**
    perform a SMO step (eq. (4.4))
  else
    perform the step of size μ_t^{pa} as planned
  end
else
  perform a SMO step (eq. (4.4))
end

both steps together have positive gain in any case. In the following we will arbitrarily fix $\eta = 0.9$. Thus, we will not consider η as a free hyper-parameter of Algorithm 7.2.

It obviously makes sense to provide the working set which was used for planning-ahead as a candidate to the working set selection algorithm. As explained above, this property together with the usage of the SMO gain function g instead of the approximation $\tilde{g}$ ensures positive gain of the double-step. Of course, positive gain is not sufficient to show the convergence to an optimal point. The following section is devoted to the convergence proof.

Although planning-ahead is done in constant time, it takes considerably longer than the computation of the Newton step. For simple problems where planning-ahead does not play a role because most steps end up at the box bounds the unsuccessful planning steps can unnecessarily slow down the algorithm. As discussed in Section 7.1.1 this is mainly the case at the beginning of the optimization. We introduce the following simple heuristic: If the previous iteration was a free SMO step, then we perform planning ahead, otherwise we perform another standard SMO step. Thus, we use the previous SMO step as a predictor for the question whether planning ahead makes sense in the current iteration. Algorithm 7.3 captures this idea.

The SMO algorithm with modified steps 1 and 2 as defined in Algorithms 7.3 and 7.2, respectively, will be referred to as the planning-ahead SMO (PA-SMO) algorithm in the following. For completeness, we state the whole PA-SMO algorithm at the end of the chapter.

It is not really clear whether the consideration of the working set $B^{(t-1)}$ for planning ahead is a good choice. In fact, it would be good to know whether this choice has a critical impact on the performance. To evaluate this impact we need to introduce a variant of PA-SMO. Because the planning-step takes only constant time we could afford to perform $N > 1$ planning steps with the working sets $B^{(t-n)}$ for $1 \leq n \leq N$ and choose the step size with the largest double-step gain. In this case we should also provide these sets to the working set selection algorithm as additional candidates. We call this variant the multiple planning-ahead algorithm using the $N > 1$ most recent working sets.

7.1.4 Convergence of the Method

In this section we will make use of the methods developed in Chapter 5 to show that the PA-SMO algorithm converges to the optimum f^* of problem (4.1).

Lemma 7.1. *Consider a σ-selection W and a sequence $(\alpha^{(t)})_{t\in\mathbb{N}}$ in $\mathcal{R}$ with $(f(\alpha^{(t)})_{t\in\mathbb{N}})$ monotonically increasing. Let there exist a constant $c > 0$ and an infinite index set $T_c \subset \mathbb{N}$*

such that the steps from $\alpha^{(t-1)}$ to $\alpha^{(t)}$ have the property

$$f(\alpha^{(t)}) - f(\alpha^{(t-1)}) \geq c \cdot \tilde{g}_W(\alpha^{(t-1)})$$

for all $t \in T_c$. Then we have $\lim_{t\to\infty} f(\alpha^{(t)}) = f^*$.

Proof. Because of the compactness of $\mathcal{R}$ there exists a convergent sub-sequence $(\alpha^{(t-1)})_{t\in S}$ for some $S \subset T_c$. We will denote its limit point by $\alpha^{(\infty)}$. Assume the limit point is not optimal. Then $\sigma \cdot (\psi(\alpha^{(\infty)}))^2 > 0$ by property $(*)$. The lower semi-continuity of ψ implies $\sigma \cdot (\psi(\alpha))^2 > 0$ for all α in an open neighborhood U' of $\alpha^{(\infty)}$. We choose a smaller open neighborhood U of $\alpha^{(\infty)}$ such that its closure $\overline{U}$ is contained in U'. Again by lower semi-continuity $\sigma \cdot (\psi(\alpha))^2$ attains its minimum $m > 0$ on the compact set $\overline{U}$. There is t_0 such that $\alpha^{(t)} \in U$ for all $t \in S$ with $t > t_0$. Then we have

$$\begin{aligned}
f(\alpha^{(\infty)}) &\geq f(\alpha^{(t_0)}) + \sum_{t\in S, t>t_0} f(\alpha^{(t)}) - f(\alpha^{(t-1)}) \\
&\geq f(\alpha^{(t_0)}) + \sum_{t\in S, t>t_0} c \cdot \tilde{g}(\alpha^{(t-1)}) \\
&\geq f(\alpha^{(t_0)}) + \sum_{t\in S, t>t_0} c \cdot \sigma \cdot (\psi(\alpha^{(t-1)}))^2 \\
&\geq f(\alpha^{(t_0)}) + \sum_{t\in S, t>t_0} c \cdot m = \infty > f^*
\end{aligned}$$

which is a contradiction. Thus, $\alpha^{(\infty)}$ is optimal. □

With the additional assumption that infinitely many SMO steps end up free we could use Lemma 7.1 to show the convergence of Algorithm 4.2 to an optimal solution (This was already proven by Fan et al. [2005] without this assumption).

Corollary 7.2. *Consider the sequence $(\alpha^{(t)})_{t\in\mathbb{N}}$ in $\mathcal{R}$ with $(f(\alpha^{(t)}))_{t\in\mathbb{N}}$ monotonically increasing. If there are infinitely many t such that the step from $\alpha^{(t-1)}$ to $\alpha^{(t)}$ is a free SMO step resulting from a σ-selection W then we have $\lim_{t\to\infty} f(\alpha^{(t)}) = f^*$.*

Proof. For a free SMO step we have $f(\alpha^{(t)}) - f(\alpha^{(t-1)}) = \tilde{g}_W(\alpha^{(t-1)})$. Thus we can simply apply the above lemma with $c = 1$. □

The allurement of this approach is that we do not need any assumption on the steps which differ from free SMO steps as long as the objective function does not decrease. This is an ideal prerequisite to tackle the convergence of hybrid algorithms like the PA-SMO algorithm which need to distinguish qualitatively different branches. Consequently, the following lemma will be helpful when applying the above results to PA-SMO.

Lemma 7.3. *Consider two iterations t and $t+1$ of the PA-SMO algorithm where planning-ahead is active in iteration t. The double-step gain $g^{2\text{-step}} = f(\alpha^{(t+1)}) - f(\alpha^{(t-1)})$ is then lower bounded by $(1-\eta^2) \cdot \tilde{g}_W(\alpha^{(t-1)})$.*

Proof. Let $\hat{\mu}_t$ denote the Newton step size in iteration t and let $\tilde{g}^* = \tilde{g}_W(\alpha^{(t-1)})$ be the gain achieved by this (possibly infeasible) step. Just like in Algorithm 7.2 we distinguish two cases:

1. The step size μ_t^{pa} satisfies $1 - \eta \leq \mu_t^{\text{pa}}/\hat{\mu}_t^{\text{pa}} \leq 1 + \eta$:
   We write the gain in iteration t in the form $\left(2\mu_t^{\text{pa}}/\mu^* - (\mu_t^{\text{pa}}/\mu^*)^2\right) \cdot \tilde{g}^*$, see Figure 7.2. Together with the strict increase of the objective function in iteration $t+1$ we get $g^{2\text{-step}} \geq (1-\eta^2) \cdot \tilde{g}^*$.

2. The step size μ_t satisfies $\mu_t^{\text{pa}}/\hat{\mu}_t \notin [1-\eta, 1+\eta]$:
   By construction the planned ahead gain (7.3) is lower bounded by $\tilde{g}^*$ (see Section 7.1.2). The planning-step assumes that the working set $B^{(t-1)}$ is selected in iteration $t+1$. However, another working set may actually be chosen. Because Algorithm 7.2 uses the SMO gain $g_B(\alpha^{(t)})$ for working set selection the gain may only improve due to the choice of $B^{(t+1)} \neq B^{(t-1)}$. Therefore $g^{2\text{-step}}$ is even lower bounded by $\tilde{g}^*$. With $0 < 1-\eta^2 \leq 1$ the desired bound follows.

□

The first case seems to complicate things unnecessarily. Further, it reduces the progress by a factor of $1-\eta^2$. We could simply skip the second if-condition in Algorithm 7.2 and in all cases turn to the else-part. That is, we could always rely on the gain g_B for working set selection. However, the usage of the exact gain $g_B(\alpha)$ instead of $\tilde{g}_B(\alpha)$ turns out to be an unfavorable choice for working set selection in practice. For performance reasons we want to allow the algorithm to use the working set selection objective $\tilde{g}_B(\alpha)$ as often as possible. Thus we have to cover case 1 in Lemma 7.3, too.

Theorem 7.4. *Let $\alpha^{(t)}$ denote the sequence of feasible points produced by the PA-SMO algorithm working at perfect accuracy $\varepsilon = 0$. Then the algorithm either stops in finite time with an optimal solution or produces an infinite sequence with $\lim_{t\to\infty} f(\alpha^{(t)}) = f^*$.*

Proof. Because the SMO algorithm checks the exact KKT conditions the finite stopping case is trivial. For the infinite case we distinguish two cases. If the sequence contains only finitely many steps which are planning ahead then there exists $t_0 > 0$ such that in all iterations $t > t_0$ the algorithm coincides with the standard SMO Algorithm 4.2 with SO working set selection and the convergence proof given by Fan et al. [2005] holds. Otherwise there exists an infinite sequence $(t_n)_{n\in\mathbb{N}}$ of planning steps. Now there are at least two possibilities to apply the above results. An easy one is as follows: From Lemma 7.3 we obtain a constant $c = 1-\eta^2$ such that Lemma 7.1 implies the desired property. Alternatively we can argue that the double-step gain is non-negative by Lemma 7.3. Algorithm 7.3 ensures that the SMO steps in iterations $t_n - 1$, $n \in \mathbb{N}$ just before the planning-ahead steps are free. Then we can apply Corollary 7.2. However, the second variant of the proof does not hold if we replace Algorithm 7.3 by Algorithm 7.1. □

As already noted above, Lemma 7.1 and Corollary 7.2 resolve the separate handling of different cases by the algorithm in a general manner. In the case of an infinite sequence of planning-ahead steps the proof does not consider the other iterations at all. This technique is similar to the convergence proof for the Hybrid Maximum-Gain second order algorithm introduced in Chapter 6 which needs to cover different cases to ensure convergence, too.

7.2 Experiments

The main emphasis of the experiments is to compare the PA-SMO algorithm to the standard (greedy) SMO algorithm. Version 2.84 of the software LIBSVM [Fan et al., 2005] implements Algorithm 4.2 with SO working set selection. For comparison, we implemented the modifications described in Algorithm 7.2 and Algorithm 7.3 directly into LIBSVM.

As discussed in Section 4.3.9 it is necessary to repeat the experiments with different permutations of the datasets in order to achieve reliable results. To reduce random effects, we created 100 random permutations of each dataset. All measurements reported are mean values over these 100 permutations. Because the permutations were drawn i.i.d. we can apply standard significance tests to our measurements.

dataset	ℓ	C	γ	SV	BSV
banana	5,300	100	0.25	1,223	1,199
breast-cancer	277	0.6	0.1	178	131
diabetis	768	0.5	0.05	445	414
flare-solar	1,066	1.5	0.1	744	709
german	1,000	1	0.05	620	426
heart	270	1	0.005	158	149
image	2,310	100	0.1	301	84
ringnorm	7,400	2	0.1	625	86
splice	3,175	10	0.01	1,426	7
thyroid	215	500	0.05	17	3
titanic	2,201	1,000	0.1	934	915
twonorm	7,400	0.5	0.02	734	662
waveform	5,000	1	0.05	1,262	980
chess-board-1000	1,000	1,000,000	0.5	41	3
chess-board-10000	10,000	1,000,000	0.5	129	84
chess-board-100000	100,000	1,000,000	0.5	556	504
connect-4	61,108	4.5	0.2	13,485	5,994
king-rook-vs-king	28,056	10	0.5	5,815	206
tic-tac-toe	958	200	0.02	104	0
internet-ads	2,358	10	0.03	1,350	6
ionosphere	351	3	0.4	190	8
spam-database	4,601	10	0.005	1,982	583

Table 7.1: Datasets used for the comparison. The dataset size, the regularization parameter C and the kernel parameter γ are given. The last two columns list the resulting total number of support vectors and the number of bounded support vectors. Due to the finite accuracy of the solutions these mean values are not always integers. For clarity we provide rounded values.

We collected a set of 22 datasets for the performance comparison. For the 13 benchmark datasets introduced in the study by Rätsch et al. [2001] we merged training and test sets. The artificial chess-board problem (see Section 3.4) was considered because it corresponds to quadratic programs which are difficult to solve for SMO-type decomposition algorithms. Because this problem is described by a known distribution, we are in the position to sample datasets of any size from it. We arbitrarily fixed three datasets consisting of $1,000$, $10,000$, and $100,000$ examples. Six more datasets were taken from the UCI benchmark repository [Blake and Merz, 1998]: The datasets `connect-4`, `king-rook-vs-king`, and `tic-tac-toe` are extracted from games, while `ionosphere`, `spambase`, and `internet-ads` stem from real world applications.

In all experiments we use the Gaussian kernel (2.1) with the single kernel parameter $\gamma > 0$. The complexity control parameter C and the kernel parameter γ were selected with grid search on the cross-validation error to ensure that the parameters are in a region where the resulting classifiers generalize reasonably well, see Table 7.1. All experiments were carried out on a Xeon 3 GHz CPU running Fedora Linux.

7.2.1 Comparison to Standard SMO

We performed 100 runs (corresponding to the 100 permutations) per dataset for both algorithms and measured the runtime and the number of iterations. The results are summarized in Table 7.2.

dataset	time			iterations		
	SMO		PA-SMO	SMO		PA-SMO
banana	2.07		2.08	23,295	>	19,721
breast-cancer	0.02		0.02	313	>	292
diabetis	0.08		0.10	361	>	358
flare-solar	0.18		0.19	792	>	744
german	0.20	>	0.19	908	>	879
heart	0.02		0.02	113		112
image	0.45		0.46	6,553	>	6,359
ringnorm	2.41		2.27	1,569	>	1,537
splice	4.04	>	3.92	6,643	>	5,854
thyroid	0.02		0.01	744	>	667
titanic	0.54	>	0.47	3,375	>	1,653
twonorm	2.67		2.65	641		642
waveform	3.03		2.99	1,610	>	1,539
chess-board-1000	3.86	>	2.98	1,883,310	>	1,186,963
chess-board-10000	76.72		75.36	32,130,476	>	24,997,371
chess-board-100000	475.37	>	428.18	145,364,030	>	105,199,379
connect4-0.2	1,268.04		1,243.56	82,076	>	77,690
king-rook-vs-king	272.80		273.06	69,410	>	64,067
tic-tac-toe	0.10		0.10	8,321	>	7,786
internet-ads	2.38		2.31	2,785	>	2,750
ionosphere	0.03		0.04	411	>	408
spambase	8.36		8.36	9,641	>	9,171

Table 7.2: Comparison of standard SMO (Algorithm 4.2) and planning-ahead SMO (Algorithm 7.4). Mean time in seconds and number of iterations are listed. The ">" sign indicates that the left value is statistically significantly larger than the right value (paired Wilcoxon rank rum test, $p = 0.05$ over 100 permutations of the datasets). The left value is in no case significantly smaller than the right one.

There is a clear trend in these results. For some datasets the PA-SMO algorithm significantly outperforms the SMO algorithm, while for other datasets there is no significant difference. Most important, PA-SMO performs *in no case* worse than standard SMO.

The number of iterations is significantly reduced in nearly all cases. This result is not surprising. It basically means that the algorithm works as expected. However, early iterations working on the whole problem take much longer than late iterations after shrinking has more or less identified the free variables. Therefore it is natural that the number of iterations is only a weak indicator for the runtime. The runtime of the PA-SMO algorithm is usually slightly reduced in the mean. This difference is significant in 5 cases. However, the striking argument for the algorithm is that it never performs worse than the standard SMO algorithm.

Although both algorithms use the same stopping condition the dual objective values achieved slightly vary. A careful check of these values reveals that the PA-SMO algorithm consistently achieves better solutions (paired Wilcoxon rank sum test, $p = 0.05$) for all datasets but `chess-board-100000`. Thus, the speed up is not a trivial effect of reduced solution quality. The tests reveal that the contrary is the case, that is, the new algorithm outputs better solutions in less time.

7.2.2 Influence of Planning-Ahead vs. Working Set Selection

It is interesting to look a little bit behind the scenes. Recall that we changed two parts of the SMO algorithm. The truncated Newton step was replaced by the planning-ahead Algorithm 7.3 and the working set selection was modified accordingly by Algorithm 7.2. It is possible to use the second modification without the first one, but hardly vice versa. Therefore, we ran the SMO algorithm with the modified working set selection but without planning ahead to get a grip on the influence of these changes on the overall performance. That is, we made sure that the algorithm selects the working set used two iterations ago if it is a feasible direction and maximizes the Newton step gain $\tilde{g}$. While the results of the comparison to standard SMO were completely ambiguous, the PA-SMO algorithm turned out to be clearly superior. Thus, the reason for the speed up of PA-SMO is not the slightly changed working set selection, but planning-ahead.

7.2.3 Planning-Ahead Step Sizes

To understand how planning-ahead is really used by the algorithm we measured the quantity $\mu^{\text{pa}}/\hat{\mu} - 1$, that is, the size of the planning-ahead step relative to the Newton step. For free SMO steps this quantity is always 0, for larger steps it is positive, for smaller steps negative, and for steps in the opposite direction it is even smaller than -1. We present some representative histograms in Figure 7.3. These histograms reveal that most planning-steps are only slightly increased compared to the Newton step size, but there are cases where the algorithm chooses a step which is enlarged by a factor of several thousands. However, few steps are reduced or even reversed, if any.

Obviously the step size histograms are far from symmetric. Therefore it is natural to ask whether a simple increase of the Newton step size can be a good strategy. By heretically using

$$\mu_t = \max\left\{\min\left\{1.1 \cdot \hat{\mu}, \tilde{U}_t\right\}, \tilde{L}_t\right\}$$

instead of equation (4.4) we still achieve $1 - 0.1^2 = 99\%$ of the SMO gain in each iteration (see Figure 7.2) and avoid the drawback of the more complicated computations involved when planning-ahead. Further, this strategy can be implemented into an existing SMO solver in just a few seconds. Experiments indicate that it is surprisingly successful, no matter if the original working set selection or Algorithm 7.2 is used. For most simple problems it performs as good as the much more refined PA-SMO strategy. However, for the extremely difficult `chess-board` problem this strategy performs significantly worse.

7.2.4 Multiple Planning-Ahead

We now turn to the variant of the PA-SMO algorithm which uses more than one recent working set for planning-ahead. This variant, as explained at the end of Section 7.1.3, plans ahead with multiple candidate working sets. Further, these working sets are additional candidates for the working set selection. We can expect that the number of iterations decreases the more working sets are used this way. However, the computations per iteration of course increase, such that too many working sets will slow the entire algorithm down. Thus, there is a trade-off between the number of iterations and the time needed per iterations. Now the interesting question is whether there is a uniform best number of working sets for all problems.

We performed experiments with the 2, 3, 5, 10, and 20 most recent working sets. It turned out that the strategies considering the most recent two or three working sets perform comparable to standard PA-SMO, and even slightly better. For 5, and more drastically, for 10 or 20 working set evaluations the performance drops, see Figure 7.4.

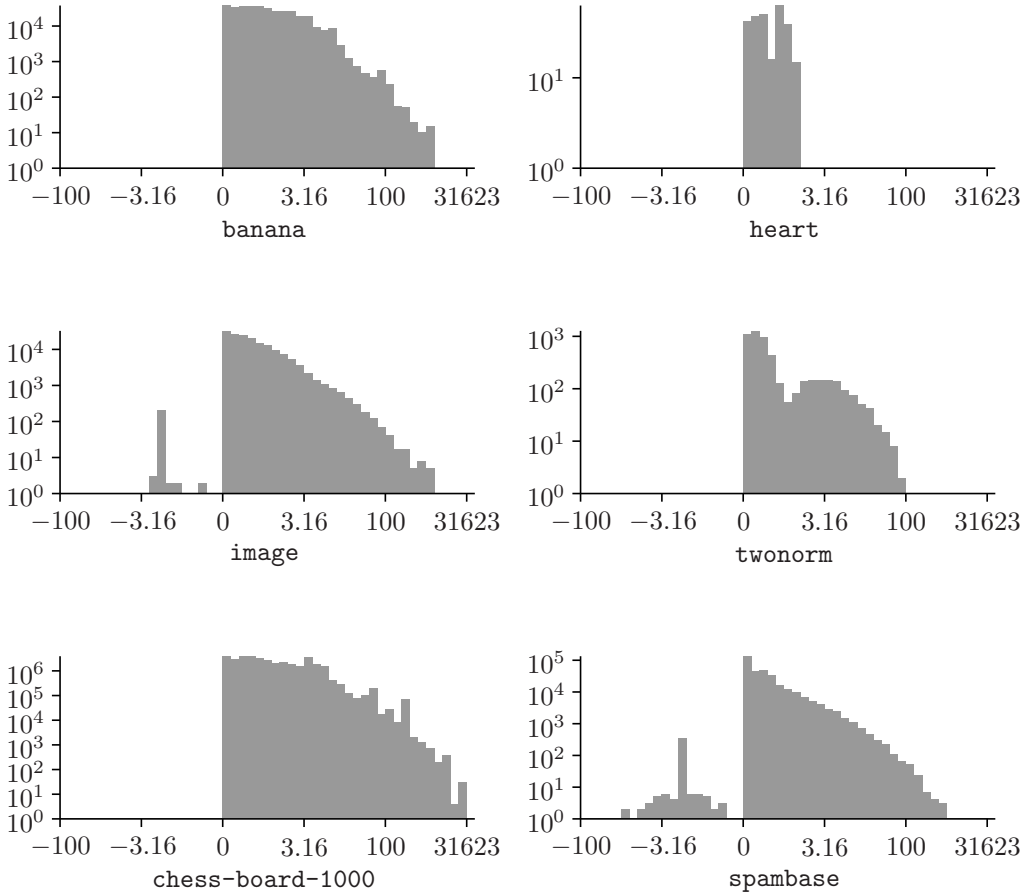

Figure 7.3: Histograms (number of iterations) of the planning-step size μ^{pa} divided by the Newton step size $\hat{\mu}$, minus 1. On both axes a logarithmic scale is used to increase the resolution for small values (to achieve this effect for the x-axis we used the parameterization $t \mapsto \mathrm{sign}(t) \cdot (10^{t^2/2} - 1)$ which is symmetric around the Newton step size corresponding to the origin of $t = \mu^{\mathrm{pa}}/\hat{\mu} - 1$, with a high resolution around this point). The rightmost bin counts all steps which exceed the scale, which is actually the case for the `chess-board-1000` dataset.

This result makes clear that we do not lose much when completely ignoring the multiple working set selection strategy, and at the same time we stay at the safe side. Therefore it seems reasonable to stick to the standard form of the PA-SMO algorithm. On the other hand we can get another small improvement if the two or three most recent working sets are considered.

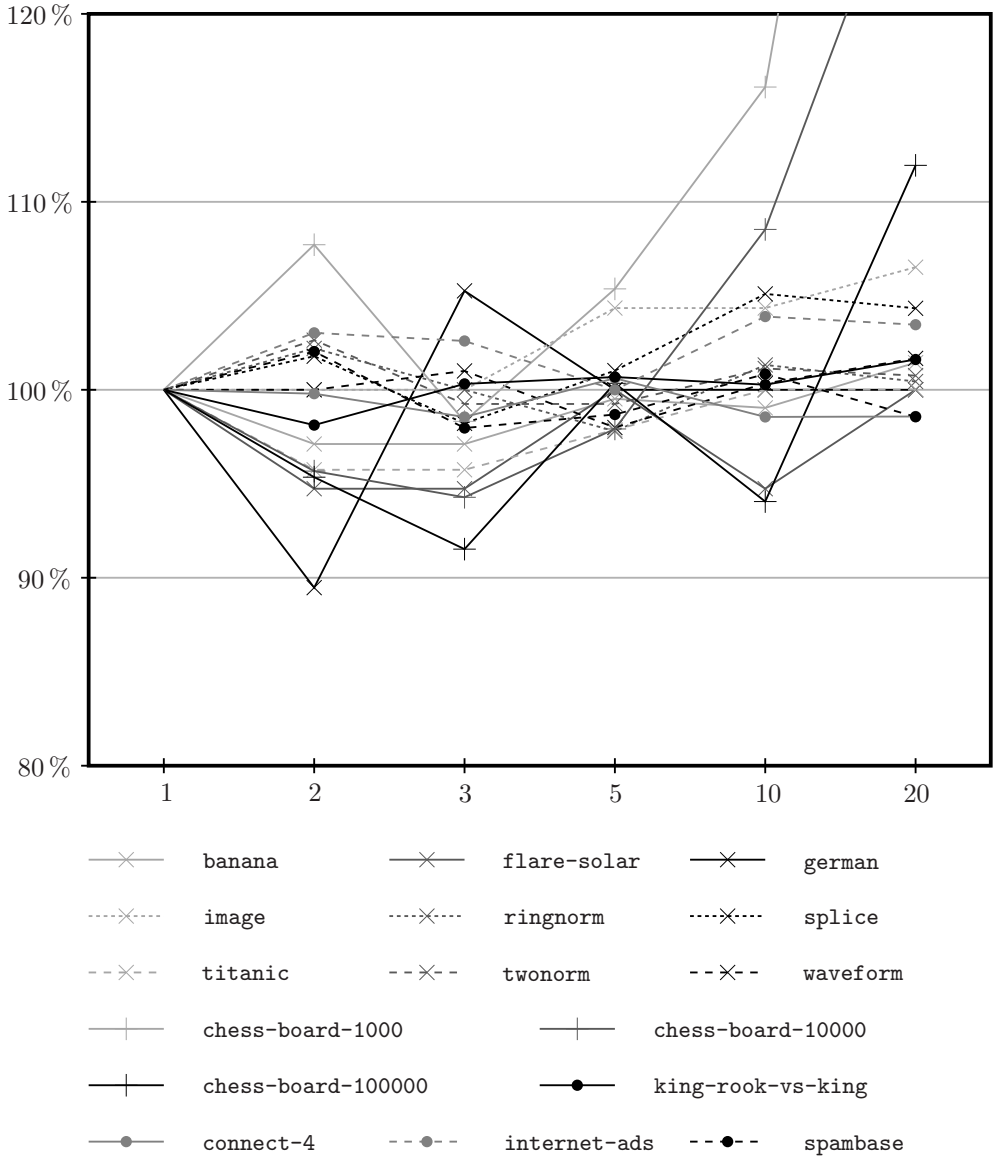

Figure 7.4: The curves show the performance (mean runtime) of the PA-SMO algorithm with 1, 2, 3, 5, 10, and 20 most recent working sets. All results are normalized with the runtime of the standard variant (only the most recent working set is used for planning ahead). Only datasets with runtimes above 100 ms are plotted, because otherwise the deviation due to the measurement accuracy overshadows the effect.

Algorithm 7.4: The complete PA-SMO Algorithm

Input: feasible initial point $\alpha^{(0)}$, accuracy $\varepsilon \geq 0$, $\eta \in (0,1)$
$G^{(0)} \leftarrow \nabla f(\alpha^{(0)}) = y - K\alpha^{(0)}$, $p \leftarrow$ true, $t \leftarrow 1$
do
  if p = *true* **then**
    $i^{(t)} \leftarrow \arg\max\{G_n^{(t-1)} \mid n \in I_{\text{up}}(\alpha^{(t-1)})\}$
    $j^{(t)} \leftarrow \arg\max\{\tilde{g}_{(i^{(t)},n)}(\alpha^{(t-1)}) \mid n \in I_{\text{down}}(\alpha^{(t-1)})\}$
    $B^{(t)} \leftarrow (i^{(t)}, j^{(t)})$
  else
    if $1 - \eta \leq \mu_{t-1}/\hat{\mu}_{t-1} \leq 1 + \eta$ **then**
      $i^{(t)} \leftarrow \arg\max\{G_n^{(t-1)} \mid n \in I_{\text{up}}(\alpha^{(t-1)})\}$
      $j^{(t)} \leftarrow \arg\max\{\tilde{g}_{(i^{(t)},n)}(\alpha^{(t-1)}) \mid n \in I_{\text{down}}(\alpha^{(t-1)})\}$
      $B^{(t)} \leftarrow (i^{(t)}, j^{(t)})$
      if $\tilde{g}_{B^{(t-2)}}(\alpha^{(t-1)}) > \tilde{g}_{B^{(t)}}(\alpha^{(t-1)})$ **then** $B^{(t)} \leftarrow B^{(t-2)}$
    else
      $i^{(t)} \leftarrow \arg\max\{G_n^{(t-1)} \mid n \in I_{\text{up}}(\alpha^{(t-1)})\}$
      $j^{(t)} \leftarrow \arg\max\{g_{(i^{(t)},n)}(\alpha^{(t-1)}) \mid n \in I_{\text{down}}(\alpha^{(t-1)})\}$
      $B^{(t)} \leftarrow (i^{(t)}, j^{(t)})$
      if $g_{B^{(t-2)}}(\alpha^{(t-1)}) > g_{B^{(t)}}(\alpha^{(t-1)})$ **then** $B^{(t)} \leftarrow B^{(t-2)}$
    end
  end
  $\hat{\mu}_t \leftarrow l_t/Q_{tt} = (v_{B^{(t)}}^T G^{(t-1)})/(v_{B^{(t)}}^T K v_{B^{(t)}})$
  if *previous iteration performed a free SMO step* **then**
    Compute the planning-ahead step size $\mu_t^{\text{pa}} = \frac{Q_{22}w_1 - Q_{12}w_2}{\det(Q)}$ (eq. (7.4))
        assuming $B^{(t-1)}$ as the next working set
    if *the current or the planned step ends at the box boundary* **then**
      $\mu_t \leftarrow \max\left\{\min\left\{\hat{\mu}_t, \tilde{U}_t\right\}, \tilde{L}_t\right\}$ (eq. (4.4))
      $p \leftarrow$ false
    else
      $\mu_t \leftarrow \mu_t^{\text{pa}}$
      $p \leftarrow$ true
    end
  else
    $\mu_t \leftarrow \max\left\{\min\left\{\hat{\mu}_t, \tilde{U}_t\right\}, \tilde{L}_t\right\}$ (eq. (4.4))
    $p \leftarrow$ false
  end
  $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \mu_t \cdot v_{B^{(t)}}$
  $G^{(t)} \leftarrow G^{(t-1)} - \mu_t K v_{B^{(t)}}$
  stop if $\psi(\alpha^{(t)}) \leq \varepsilon$
  $t \leftarrow t + 1$
loop

Chapter 8

Application to Online and Active Learning

The applicability of support vector machines to large scale problems is limited by the training time, which scales at least quadratically with the number of input patterns. Although progress has been made during the last decade to significantly speed up SVM learning [Joachims, 1998, Platt, 1998, Fan et al., 2005, Glasmachers and Igel, 2006], there is an increasing interest in approximation methods [Vishwanathan et al., 2003, Bordes et al., 2005, Tsang et al., 2005, Keerthi et al., 2006]. The aim of approximation methods is to come up with a quickly computable rough estimate of the SVM solution, usually without any accuracy guarantees. The goal is just to come reasonably close to the generalization accuracy of the fully trained SVM. Such a proceeding can be useful for real-time applications where the training time is limited a priori or for extremely large problems. Usually such methods either can not compute the exact SVM solution or are slower than the SMO algorithm for the standard accuracy, but on the other hand they are faster than SMO in producing a rough approximation.

We consider the LASVM algorithm proposed by Bordes et al. [2005], which performs SMO steps during learning. The algorithm can be used for online learning, where the machine is presented a continuous stream of fresh training examples and the task is to update the classifier quickly after every observed pattern. This is an important property because standard SVM training algorithms like SMO are not well suited to online learning. In active learning mode, this continuous stream of examples is produced by a heuristic picking informative training patterns. In this case the machine decides from which pattern it wants to learn next.

A major advancement of SMO-type algorithms in recent years has been the introduction of second order working set selection algorithms [Fan et al., 2005, Glasmachers and Igel, 2006]. Here we present our contribution to transfer these results to online and active learning by incorporating them into the LASVM algorithm. The chapter follows the article by Glasmachers and Igel [2008].

8.1 The LASVM Algorithm

The LASVM algorithm [Bordes et al., 2005] holds a set $\tilde{S}$ of support vectors on which the current classifier is based. It uses SMO to modify both the dual variables α_i, $i \in \tilde{S}$, and the set $\tilde{S}$ itself. There are two types of update steps, called `process` and `reprocess` using different kinds of working sets B. In `process` one element of B comes from the stream of online or active learning examples, while in `reprocess` it is selected from $\tilde{S}$.

The other element is always chosen from $\tilde{S}$. If the coefficient of a new example ends up non-zero after `process` the example is added to $\tilde{S}$. After `reprocess` an example is removed from $\tilde{S}$ if its coefficient is zero and the current solution can not be improved with a working set formed from this example and any other candidate from $\tilde{S}$. This rule coincides with the shrinking heuristic used by LIBSVM. LASVM uses MVP working set selection to choose all of the indices which are not fixed by the stream.

The active learning LASVM algorithm starts with initializing $\tilde{S}$ with $\ell_{\text{init}} \ll \ell$ elements of each class using `process` repeatedly. A predefined number of epochs follows, consisting of ℓ iterations each. Every iteration is divided into three steps: An active selection step, a `process` step, and a `reprocess` step. After the last epoch an optional finishing step similar to standard SMO optimization can be conducted. The duration of this finishing step cannot be controlled in advance, as it repeats `reprocess` steps until the KKT conditions are fulfilled with a predefined accuracy ε on the support vector set $\tilde{S}$. That is, given the algorithm has identified the set $\tilde{S}$ of support vectors correctly, the finishing step basically coincides with the SMO algorithm.

For many problems the resulting classifier performs already sufficiently well in terms of generalization error after only a single epoch of training [Bordes et al., 2005]. In addition, the resulting kernel expansion after the finishing step is usually sparser than the exact SVM solution.

Bordes et al. [2005] have shown that in the limit of arbitrary many epochs LASVM converges to the exact SVM solution. This statement holds for the online learning case and for the active learning scheme with slight modifications.[1]

8.2 Working Set Selection for LASVM

Both MVP and SO working set selection are simple in that they depend on the current search point α only. Therefore they are well suited for the LASVM algorithm where other state information like the previously selected working set in HMG may not make sense (for example because a corresponding index has been removed from $\tilde{S}$). We have to make further minor changes to these algorithms to apply them within LASVM.

Whenever LASVM chooses an index into its working set, it is restricted to the set $\tilde{S}$. The easiest way to incorporate this modification into our notation is to restrict the potential working sets

$$\begin{aligned} I_{\text{up}}(\alpha, \tilde{S}) =& \{n \in \tilde{S} \mid \alpha_n < U_n\} \\ I_{\text{down}}(\alpha, \tilde{S}) =& \{n \in \tilde{S} \mid \alpha_n > L_n\} \\ \mathcal{B}(\alpha, \tilde{S}) =& \{(i,j) \mid i \in I_{\text{up}}(\alpha, \tilde{S}),\, j \in I_{\text{down}}(\alpha, \tilde{S}),\, i \neq j\} \ . \end{aligned}$$

Then MVP and SO working set selection become

$$B = \arg\max \left\{ v_B^T \nabla f(\alpha) \;\middle|\; B \in \mathcal{B}(\alpha, \tilde{S}) \right\}$$

and

$$\begin{aligned} &B = (i,j) \text{ with} \\ &i = \arg\max \left\{ \frac{\partial f}{\partial \alpha_n}(\alpha) \;\middle|\; n \in I_{\text{up}}(\alpha, \tilde{S}) \right\} \\ &j = \arg\max \left\{ \tilde{g}_{(i,n)}(\alpha) \;\middle|\; n \in I_{\text{down}}(\alpha, \tilde{S}) \right\} \ , \end{aligned}$$

[1] For example, it is sufficient to ensure that a σ-selection is used at least once per epoch. This may require the active learning scheme to change its strategy from time to time. The resulting rate of convergence is slow, but such a modification still ensures convergence to the optimum.

respectively, formally depending on the state $(\alpha, \tilde{S})$ and therefore on the state space $(\mathcal{R} \setminus \mathcal{R}^*) \times \mathbb{P}(\{1, \ldots, \ell\})$. Note that for LASVM the complexity of both working set selection policies is linear in $|\tilde{S}|$, not in ℓ.

For a `process` step the stream of online or active learning examples determines one of the entries of the working set. If the corresponding coefficient is at the lower bound it is fixed as the first component of the working set, and the second component needs to be selected from the set $I_{\text{down}}(\alpha, \tilde{S})$. Analogously, if the coefficient is at the upper bound the index is fixed to the second position in the working set and we can restrict the search for a partner index to the set $I_{\text{up}}(\alpha, \tilde{S})$. If the variable is free we have to consider both possibilities. Then the terms $v_B^T \nabla f(\alpha)$ for MVP or $\tilde{g}_B(\alpha)$ for SO are maximized over these sets to determine the remaining entry of the working set.

8.3 Experiments and Results

We experimentally compared the performance of LASVM using either MVP or SO working set selection. In the work of Bordes et al. [2005] the original LASVM was compared to an old version of LIBSVM using MVP working set selection. Thus, we had to clarify whether LASVM is still faster after the new working set selection is incorporated into both machines. Therefore, we conducted the experiments also with LIBSVM 2.8 using SO working set selection [Fan et al., 2005].

We considered four benchmark datasets, namely `Adult`, `Banana`, `USPS+N`, and `Chessboard`. The first three benchmarks were also used by Bordes et al. [2005]. These are difficult datasets in the sense that it takes several learning epochs for LASVM to come close to the performance of a standard 1-norm SVM. On datasets where LASVM basically converges in a single epoch we observed no significant differences between the two working set selection strategies in LASVM. In addition, the `Chessboard` distribution (see Section 3.4) was considered because a simple dataset of $5,000$ examples from this distribution leads to a dual SVM problem that is already rather difficult for SMO-type algorithms. Gaussian kernels (2.1) were used for all datasets. The cache for the kernel matrix rows was set to the default size of 256 MB if not stated otherwise. Some datasets consist of 10 different partitionings into training and test sets. In these cases all our results refer to mean values over the partitions [Bordes et al., 2005].

We measured runtimes for different cache sizes, classification rate on a test set not used for training, and primal (2.3) as well as dual (2.5) objective value. First, we conducted 50 epochs of training to investigate the convergence behavior of the two LASVM variants. Second, we ran LASVM for a single epoch of training. This corresponds to a realistic application scenario. These results are compared to the baseline batch learning classifiers computed by LIBSVM with standard accuracy.

The learning trajectories over 50 epochs shown in Figure 8.1 demonstrate that the new working set selection noticeably speeds up LASVM. The SO algorithm leads to much faster learning than MVP. The longer LASVM needs to compute a sufficiently good hypothesis the more pronounced are the differences. The performance indicators listed in Table 8.1 clearly show that the solution quality considerably increases at the same time.

It seems that the selection of better working sets leads to a faster convergence of the set $\tilde{S}$. Wrongly adding a training example to $\tilde{S}$ can decrease the overall speed as subsequent iterations scale linear in $|\tilde{S}|$. Even worse, this may lead to other wrong choices in future decisions. Therefore a well founded working set selection can play a key role in speeding up the entire algorithm. This is in accordance with the number of support vectors used by both methods, see Table 8.1. After a single learning epoch, the SO solutions are extremely sparse compared to machines produced using MVP. With MVP selection, LASVM adds lots of coordinates to the set $\tilde{S}$ of candidate support vectors

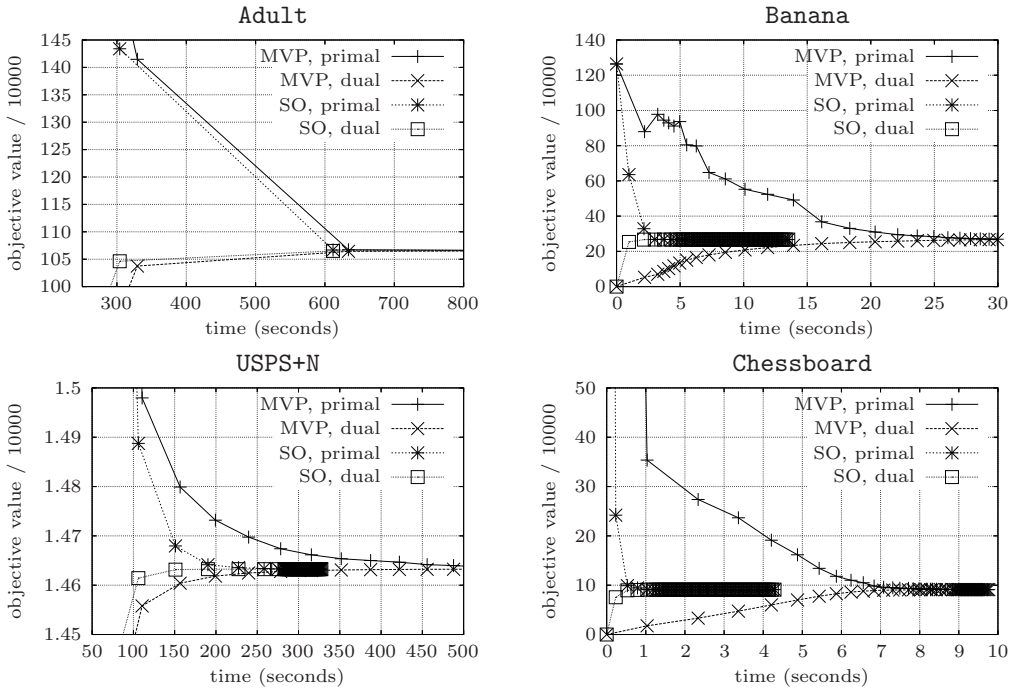

Figure 8.1: Performance of LASVM with most violating pair (MVP) and second order (SO) working set selection. The plots illustrate the evolution of primal and dual objective over 50 epochs of training on the benchmark problems. The upper (decreasing) and lower (increasing) curves represent the primal and dual objective, respectively, converging to the same optimal value.

that later turn out to be non-support vectors. These wrong decisions, which are most prominent in the experiments with the datasets `Banana` and `Chessboard`, considerably slow the algorithm down. Therefore, besides improved training speed and accuracy, we consider the sparseness of intermediate solutions as a decisive difference between MVP and SO working set selection. In three of four cases the solutions of LASVM with SO working set selection were even more sparse than the exact SVM solutions, whereas LASVM with MVP can lead to machines with almost four times as many support vectors compared to the LIBSVM solutions when considering a single learning epoch for LASVM. Thus, the main result of this comparison is non-ambiguous: The SO selection improves LASVM in all measured performance criteria.

The comparison of LASVM with LIBSVM 2.8, both using SO working set selection, is more difficult. Our selection of difficult datasets as discussed above clearly biases this comparison. On simple datasets where LASVM needs only a few epochs to reach the same objective value as LIBSVM the LASVM algorithm has the same classification error [Bordes et al., 2005]. When comparing the classification rates in Table 8.1 we have to keep in mind that LASVM was only trained for one epoch and had usually not reached a sufficiently accurate solution, whereas LIBSVM was trained with its standard stopping criterion. Of course, the LASVM solution will approach the exact SVM solution found by LIBSVM (i.e., having approximately the same classification rate and number of support vectors) in subsequent epochs. Still, even for some of our difficult datasets we see only small differences in classification rate between LIBSVM and LASVM with SO working set

dataset	parameters	algorithm	#SV	cl. rate	primal	dual
`Adult`		MVP	14033	74.84%	1,187,350	1,036,160
$\ell = 32,562$	$C = 100$	SO	11167	82.10%	1,189,174	1,046,430
1 partition	$\gamma = 0.005$	LIBSVM	11345	85.13%	1,065,429	1,065,408
`Banana`		MVP	484	64.79%	915,624	52,984
$\ell = 4,000$	$C = 316$	SO	145	80.01%	375,718	253,562
10 partitions	$\gamma = 0.5$	LIBSVM	162	90.04%	267,278	265,545
`USPS+N`		MVP	3870	99.09%	17,214	14,207
$\ell = 7,329$	$C = 10$	SO	2501	99.36%	15,867	14,454
10 partitions	$\gamma = 2$	LIBSVM	2747	99.51%	14,635	14,633
`Chessboard`		MVP	3395	97.86%	782,969	60,771
(5000)	$C = 1000$	SO	944	98.91%	143,614	83,686
1 partition	$\gamma = 2$	LIBSVM	873	99.40%	90,959	90,946

Table 8.1: Classification rate, primal and dual objective value, and the number of support vectors for LASVM with MVP and SO working set selection for all four datasets. For comparison, the LIBSVM results represent solutions with guaranteed KKT violations of at most $\varepsilon = 0.001$ (see Chapter 4).

selection after a single epoch, whereas LASVM with MVP performs considerably worse.[2]

On the `Adult` dataset the LIBSVM algorithm outperforms LASVM in terms of training time although it computes a more accurate solution. This result indicates that LIBSVM profits more than LASVM from the SO working set selection for this problem. This is possible because the number of LASVM iterations is fixed by the epoch length while LIBSVM may stop after less iterations.

On the other three datasets LASVM is clearly faster than LIBSVM, especially for small cache sizes, see Figure 8.2. The small cache behavior is an indicator of how an algorithm performs on large scale datasets. By design it is one of LASVM's advantages to work well with comparatively small cache sizes, or, with extremely large datasets. Therefore it is not surprising that LASVM is much more robust to a cache size reduction than LIBSVM. This is a general property of LASVM independent of the working set selection algorithm.

[2]In general it is dangerous to compare the test error achieved by different SVM training algorithms. The primal and dual objective values are much more reliable quantities. In this case the differences are statistically significant.

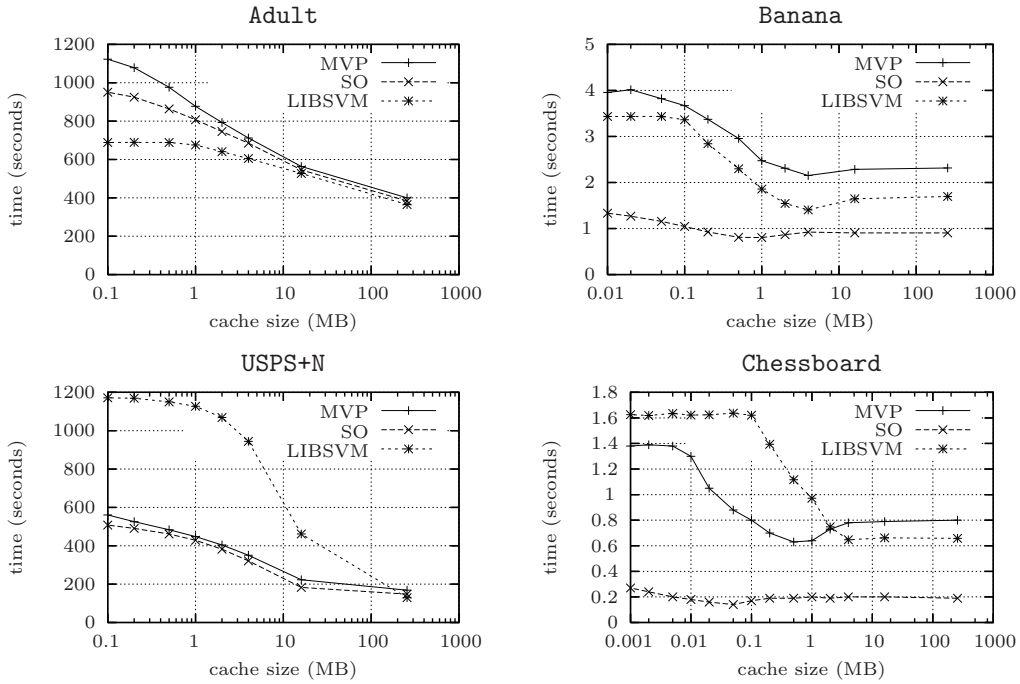

Figure 8.2: Dependency of the runtime on the cache size for a single epoch. The usage of a small cache simulates the behavior on large problems.

Part III

Model Selection for Support Vector Machines

Chapter 9

The Model Selection Problem

Assume we are given a dataset $D = \big((x_1, y_1), \ldots, (x_\ell, y_\ell)\big) \in (X \times Y)^\ell$, say, for binary classification, and we want to apply a support vector machine to this problem. Until now the question how to come up with a kernel function $k : X \times X \to \mathbb{R}$ and a value $C > 0$ for the regularization parameter suitable for the particular dataset was left open. This problem is known as the model selection problem for support vector machines, which is the topic of the third part of this thesis.

It is understood that the choice of the kernel and the regularization parameter value has a considerable influence on the quality of the resulting hypothesis. Therefore we look for a general recipe how to robustly obtain a well generalizing model. Such an algorithm would make SVMs generally applicable as a black box to all kinds of practical problems, requiring only a minimum of expert knowledge about the machine and the problem at hand.

A converse approach is to explicitly incorporate available expert knowledge into the kernel function. There is an elaborate theory guiding the explicit construction of specialized kernel functions. We refer to the books by Schölkopf and Smola [2002] and Shawe-Taylor and Cristianini [2004] for a bunch of such construction methods. Here we concentrate on a different aspect, namely the adaptation of the remaining free parameters of a family of kernel functions, possibly resulting from such a constructive approach.

As we consider the application of support vector machines to supervised learning problems we will assume in the following that the decision to apply SVMs was already taken. For simplicity we restrict ourselves to 1-norm SVMs for binary classification. Then, as already stated above, the model selection problem can be cast into the following two questions:

- Which kernel function should be used?
- How much regularization is needed?

These questions are not independent, and a globally valid answer does certainly not exist, as there are no uniformly best regularization parameter and kernel function for all problems. Thus, to give a general answer to these questions we have to provide a way how to select the kernel and the regularization parameter as good as possible for each individual problem, or for whole classes of problems, in a data-driven manner.

Usually prior knowledge can guide us to fix a parameterized family of candidate kernel functions that are believed to capture the most important information needed to separate the classes. If we have few or no knowledge about the data at all and a (canonical) vector space embedding is available the Gaussian kernel is used regularly. In either case we end up with a parameterized finite dimensional family of kernel functions. Then the model

selection problem reduces to the selection of finitely many parameter values. In contrast to the SVM model parameters $(w, b)\widehat{=}(\alpha, b)$ the kernel parameters (denoted by θ) and the regularization parameter C are called hyperparameters in the machine learning literature.

To actually select values for the SVM hyperparameters we need a selection criterion and a search strategy. The generalization error is of course the objective to minimize, but as the data generating distribution is unknown it can not be computed. Instead most approaches try to minimize related quantities which can be computed from the available training data. We will see that the selection of an objective function and a search strategy is limited by efficiency considerations.

In case of two hyperparameters a simple grid search with cross-validation based objective function is a common technique. A fixed grid is defined in the plain spanned by C and θ, or usually an equidistant grid for $\log(C)$ and $\log(\gamma)$ for the Gaussian kernel (2.1) (with $\theta = \gamma$ in this case), and the point with the lowest cross-validation error (see Section 9.1.1) is picked and defines the model. Computationally, this is only possible for low dimensional kernel families as the search strategy scales exponentially in the number of parameters. In higher dimensions evolution strategies and other direct search methods are a viable alternative. In this work we will mainly deal with (nearly everywhere) differentiable objective functions. Then it is possible to apply efficient gradient-based algorithms to approximate the (local) optimum of the objective function.

Two types of questions arise:

- Which objective function leads to the best generalization performance?
- Which optimization strategy finds the best (local) optimum of the objective function?

Again it turns out that both questions are connected. We can only evaluate a combination of objective function and search strategy and not every strategy is applicable (or makes sense to be applied) to each objective function.

We aim at a general solution of the (parametric) model selection problem. Such a potential solution consists of an objective function together with a suitable search strategy. In the following we will refer to such a pair as a model selection strategy or a model selection algorithm. The most important goal for such a model selection strategy is a low generalization error of the resulting machine. As the training data are random variables and some of the error measures and search strategies are randomized the mean or expected generalization error is to be minimized, but robustness is also important.

Efficiency, mainly in terms of a manageable runtime, is another concern. This becomes important for large problems, as some model selection schemes do not scale well with the number ℓ of training examples. On the other hand model selection is relatively easy if the underlying distribution is well sampled, such that crude but easy to compute objective functions can do the job. Therefore, when considering efficiency, we have to keep in mind that we can allow for computationally intensive methods for difficult problems arising from small datasets, while we need fast methods for large datasets.

In this chapter we will concentrate on objective functions for model selection and at first ignore the corresponding search strategies for their optimization. The chapter is organized as follows: First, we concisely review important model selection objective functions from the literature. Then we turn to the value of a probabilistic interpretation of SVM outputs for the model selection problem, which allows us to derive a new approach for the construction of randomized model selection objective functions. For our goal to apply efficient gradient-based optimization algorithms to approximate optima of these functions we need to compute their derivatives, which is non-trivial in some cases. Finally we investigate the issue of multiple local optima.

9.1 Objective Functions

If we make a wish list for the construction of a model selection objective function, then it surely lists the following properties: A perfect objective function should have only a single local optimum or even better it should be convex, and this optimum should coincide with the global optimum of the generalization error, which can in principle possess multiple local optima. This is even more important than a monotonic relation of the objective function to the generalization error. However, experience shows that there is little hope to build such a function from data with high probability, at least for a small dataset size. Nevertheless we should keep these properties in mind, as they ensure that we can easily identify parameters leading to well generalizing classifiers.

In the following we introduce common model selection objective functions. Some of these are quite generally applicable to any classifier, which is treated as a black box. Other approaches are only applicable to support vector machine classifiers as they make explicit use of the linearity of the separation in feature space or of the concepts of margins and support vectors. Further the objectives can be distinguished according to whether they need to train machines on a subset of the training data to compute nearly unbiased estimates of the generalization error or whether the training set is used as a whole.

Even if an objective function is designed to be an estimate or a bound of the generalization error there are at least two effects leading to random or systematic deviations of the objective function values from the generalization error. Of course, an upper bound of a quantity systematically deviates from this quantity. For hold-out based estimates (see below) we have a systematic deviation due to the reduction of the size of the training set. A second source of deviation is the introduction of splits of the dataset into training and validation data. For a fixed split we get a systematic deviation, while this deviation is random if we use a new (random) split for each evaluation. For all of these effects it is unclear how the deviations from the true generalization error translate to deviations in the position of (local) optimal points. The analysis is further complicated by the fact that the available training dataset itself is a random variable.

We will refer to these considerations when discussing the different model selection objective functions below. For this discussion of deviations, mainly due to splits of the training dataset into a smaller training set and a validation or hold-out set, we are mainly interested in their bias and variance. That is, when bias and variance of objective function values are discussed below, these terms refer to the bias and the variance of the generalization error estimates, usually with respect to the split of the dataset into subsets.

9.1.1 Hold-Out Sets and Cross-Validation

The probably most simple type of objective function is the error on a hold-out set. Assume we use a fraction of, for example, 1/5 of the training data as a hold-out set. Then we train a machine on the remaining 4/5 of the data and compute the fraction of errors on the hold-out set. This error measure is an unbiased estimate of the generalization error of a machine trained on a dataset of size $(4/5)\ell$ sampled i.i.d. from ν. Usually the optimal parameters for this machine will be quite good for a machine trained on the whole dataset, or for a typical dataset of the full size ℓ, such that this objective function seems to allow for a simple and viable solution to the model selection problem.

But it turns out that the hold-out error has a high variance, especially for small datasets and for small hold-out sets. On the other hand, the larger the hold-out set the smaller becomes the remaining training set imposing a larger bias in the estimation. Further the asymmetry between the roles of the reduced training set and the hold-out set is dissatisfactory. Cross-validation is a simple procedure which improves on these points. However, the split of the data into training and hold-out set remains arbitrary.

In a k-fold cross-validation procedure based on any error measure the data D are split into k disjoint subsets $D_1, \ldots, D_k$ of roughly equal size. Then for each $i \in \{1, \ldots, k\}$ a machine is trained on the dataset $D \setminus D_i$ and the error E_i on the corresponding hold-out set D_i is computed. Finally the total error $\sum_{i=1}^{k} E_i$ is the so-called k-fold cross-validation error. In this procedure the underlying error function is evaluated exactly once on each training example. In many cases the cross-validation procedure is used in conjunction with the classification error or 0-1-loss, counting the wrongly classified examples, but, like the hold-out error, this general scheme is applicable to other error measure, like for example the quadratic loss for regression problems. In the machine learning literature, common values for the parameter k range from three to ten, and the choice $k = 5$ can be considered the default value [Hastie et al., 2001].

Compared to the simple hold-out error the variance of this estimate is reduced, but as the data used in the different splits are of course not independent the reduction of the variance is not by a factor of $\frac{1}{k}$. In general the cost for the computation of the cross-validation error is k times the cost of the computation of the hold-out error. Especially for small datasets the cross-validation error can heavily depend on the split $D_1, \ldots, D_k$ of the dataset and thus has a relatively high variance w.r.t. the choice of the split. This is a dissatisfactory situation as there is no such thing as a canonical data split.

9.1.2 Leave-One-Out Error

We can vary the parameter k of the cross-validation procedure to small numbers reducing the variance of the estimate to large numbers reducing the bias. The leave-one-out (LOO) procedure is an extreme variant of cross-validation for $k = \ell$. That is, the machine is trained on all but one training example and the single hold-out example is predicted. This process is repeated for each training example. From these single predictions the leave-one-out procedure computes an unbiased estimate of the generalization error of a machine trained on $\ell - 1$ examples, which is nearly unbiased for machines trained with ℓ examples (which we are finally interested in). In contrast to general cross-validation, in this procedure the split of the dataset is canonical. The variance of the resulting estimate is nevertheless high, because the prediction of a single hold-out example is extremely unreliable and the combined estimates are highly dependent. Thus, this generalization error estimate is unreliable, such that it is not clear whether we can profit from its small bias. Nevertheless, this procedure is appealing in many respects:

A simple observation which is promising in the context of SVMs is that the hypothesis remains unchanged if a non-support vector is removed from the training set. Further, non-support vectors are classified correctly by the SVM as they have a functional margin of at least one from the separating hyperplane. Thus, a non-support vector can not produce an error in the leave-one-out procedure. This is why the LOO procedure can become significantly less computationally intensive if the hypothesis is a sparse combination of the training examples. Furthermore, the training of each machine on $\ell - 1$ examples can use a warm start from the machine trained on all examples.[1]

Another interesting property is that the hypothesis changes only slightly if only one example is removed from the training set. This allows for a relatively simple analysis of the prediction on the hold-out example, mainly in terms of upper bounds on the resulting error. Consequently a number of bounds for the leave-one-out error have been proven [Vapnik, 1998, Vapnik and Chapelle, 2000, Chapelle et al., 2002]. These are regularly

[1] This can be accomplished by setting the upper and the lower bound of the Lagrange multiplier α_n of the hold-out example (x_n, y_n) in problem (4.1) to zero. Then α_n must be set to zero, too, resulting in a need to change other multipliers in order to keep the equality constraint valid. This can be accomplished in at most linear time and is very fast in most cases. Often the training starting from the warm start solution takes a small fraction of the computations of a complete training from scratch.

termed generalization bounds which is misleading in a two-fold manner: First and most important these quantities bound the leave-one-out error and not the generalization error. Second they do not only hold with a certain probability (due to the random training set) like bounds on the generalization error always do (compare to, for example, Theorem 1.1).

The leave-one-out error as well as its upper bounds are regularly proposed as model selection objective functions (see for example Vapnik [1998], Vapnik and Chapelle [2000], Chapelle et al. [2002], Chung et al. [2003], Keerthi [2002]). Some of the bounds are very loose, while others turn out to be quite tight. As Chung et al. [2003] pointed out, this is of minor relevance for the model selection problem. The most important question is whether the optimal solutions of these functions result in machines with low generalization error with high probability. Experiments indicate that the tightness of the available bounds is often a poor indicator of the quality of their (local) optima.

Note that in most cases the application of such bounds as model selection objectives violates the prerequisites of the functions being bounds. For example, one needs an a priori choice of a class of hypotheses to bound the generalization error in terms of its VC dimension. Minimizing the radius margin quotient (see below) over the hyperparameters results in an a priori unknown range of $\|w\|$, violating the preconditions of the bound. Nevertheless, the radius margin quotient can turn out to be a valuable objective function for efficient model selection for hard margin SVMs. Refer to the book chapter by Bartlett and Shawe-Taylor [1999] for approaches how to get around this dilemma.

9.1.3 Number of Support Vectors

As discussed above the hold-out error of a non-support vector in the leave-one-out procedure is known to be zero. Therefore it is trivial to upper bound the number of errors with the number of support vectors. However, this bound can be quite loose. It is, just like the leave-one-out error, integer valued and therefore not differentiable. Further it usually prefers hypotheses resulting from little or no regularization at all which makes it implausible as a model selection objective.

9.1.4 Radius Margin Bound

Minimization of the radius margin quotient R^2/ρ^2 can be used for model selection of *hard margin* SVMs. Here, R is the radius of the smallest sphere containing the training data, and ρ is the (geometric) margin. For hard margin machines the only hyperparameters are the parameters of the family of kernel functions. The regularization parameter can be included for the 2-norm machine by the computational trick to consider C as a kernel parameter (see Section 2.2.3 or refer to Chapelle et al. [2002]).

According to Theorem 10.3 by Vapnik [1998] the set of hypotheses obtained from linear functions $x \mapsto \langle w, \Phi(x)\rangle$ with $\|w\| = \frac{1}{\rho} \leq A$ has VC-dimension bounded by $\lfloor R^2 A^2 \rfloor + 1$ on the ball with radius R around the origin (see also Section 2.2.2). In this line of thought Vapnik has proven the following theorem which is based on the leave-one-out procedure. It is the main motivation for the minimization of the radius margin quotient.

Theorem 9.1 (Theorem 10.6 by Vapnik [1998])**.** *For optimal hyperplanes passing through the origin the following inequality*

$$\ell \cdot \mathbb{E}[\mathcal{R}^{\nu}_{\ell-1}] \leq \mathbb{E}\left[\frac{R^2_{\ell}}{\rho^2_{\ell}}\right]$$

holds true, where $\mathcal{R}^{\nu}_{\ell-1}$ is the generalization error of the hard margin SVM trained on a dataset of $\ell - 1$ examples, and R_ℓ and ρ_ℓ are (random) values that for a given training set size of ℓ define the maximal norm of support vectors $\Phi(x)$ and the margin.

Note that the left hand side of the above inequality is the expected value of the leave-one-out error. This theorem was later extended to general hyperplanes not necessarily passing through the origin, and to balls centered in arbitrary points [Chapelle et al., 2002]. Then the statement becomes applicable to standard hard margin SVMs, and the reduction of the radius by removing the restriction that the ball is centered in the origin gets us a tighter bound. Therefore we use the more general formulation in the following, such that the radius R refers to the smallest (non-centered) ball containing all training examples.

The radius margin quotient depends differentiably on the kernel function and is therefore appropriate for efficient gradient-based optimization of a large number of hyperparameters [Chapelle et al., 2002]. However, there is no differentiable extension to 1-norm SVMs.

9.1.5 Span Bound

In the radius margin bound the role of the radius is to bound the distance $\|\Phi(x_i) - \Phi(x_j)\| \leq 2R$ between two training examples. In most cases this is a crude estimate. The geometrically motivated concept of the span of support vectors introduced by Vapnik and Chapelle [2000] improves this situation. Consequently the resulting span bound is much tighter than the radius margin bound for many problems, but it is much more costly to compute. Therefore an estimate of the span bound has been proposed, which holds under the assumption that the sets of free and bounded support vectors do not change in the leave-one-out procedure. Although this assumption is clearly wrong the approximation works well. With this approximation Theorem 2.3 by Vapnik and Chapelle [2000] states

$$y_n f(x_n) - y_n f^{\hat{n}}(x_n) = \alpha_n^2 S_n^2$$

where $f(x) = \sum_{i=1}^{\ell} y_i \alpha_i k(x_i, x) + b$ is the function corresponding to the SVM hypothesis obtained from the full training dataset, while $f^{\hat{n}}$ is the function obtained from training with the n-th example left out. The quantity S_n, called the span of the feature space representation $\Phi(x_n)$ of the example x_n, is defined as the distance of $\Phi(x_n)$ to the affine subspace spanned by the remaining support vectors.

The LOO procedure makes an error if the margin $y_n f^{\hat{n}}(x_n)$ of x_n w.r.t. the leave-one-out hypothesis is negative. Therefore

$$|\{n \in \{1, \ldots, \ell\} \mid y_n f(x_n) \leq \alpha_n^2 S_n^2\}|$$

is an upper bound for the leave-one-out error under the above assumption. However, to obtain a strict bound on the LOO error we need to drop the unrealistic assumption that the sets of free and bounded support vectors do not change during LOO. This case is covered by Theorem 2.2 in the original work by Vapnik and Chapelle [2000] on the span bound. The resulting bound for the 1-norm SVM is not differentiable, as it depends on the number of bounded support vectors. This bound is neither tight nor conceptually satisfactory, as it counts all bounded support vectors as errors.

Instead of the original span bound we propose a closely related quantity motivated by the calculations above, which bounds the LOO error and is continuous and nearly everywhere differentiable:

$$S = \sum_{n=1}^{\ell} 1 - \begin{cases} f^{\hat{n}}(x_n)/f(x_n) & \text{for } y_n f^{\hat{n}}(x_n) > 0 \\ 0 & \text{otherwise .} \end{cases}$$

Per definition, the LOO procedure makes an error if the margin $y_n f^{\hat{n}}(x_n)$ is negative. The above bound just counts these cases. The other extreme case are non-support vectors

for which the functions $f^{\hat{n}}$ and f coincide. These terms do not contribute to the sum at all. Using the inequality $y_n f(x_n) > y_n f^{\hat{n}}(x_n)$ for the margins we conclude that the cases in between contribute a value in the interval $(0, 1)$ which is strictly larger than the vanishing contribution to the LOO error.

The complexity of the computation of S is comparable to the LOO error as we have to construct $f^{\hat{n}}$ for each support vector x_n. Although this new quantity does not coincide with the span bound proposed by Vapnik and Chapelle [2000] we will, because of the conceptual similarity, refer to it under the same name in this thesis.

9.1.6 Kernel Target Alignment

The kernel target alignment (KTA)

$$\hat{A} := \frac{\langle yy^T, K\rangle}{\sqrt{\langle yy^T, yy^T\rangle\langle K, K\rangle}} = \frac{\langle yy^T, K\rangle}{\ell \cdot \sqrt{\langle K, K\rangle}}$$

is an objective function first defined by Cristianini et al. [2002]. The inner product for matrices is defined as $\langle M, N\rangle = \mathrm{tr}(M^T N) = \sum_{i,j=1}^{\ell} M_{ij} N_{ij}$, $K \in \mathbb{R}^{\ell\times\ell}$ with $K_{ij} = k(x_i, x_j)$ is the kernel Gram matrix, and y is the label vector, with yy^T denoting the outer product with itself.

The kernel target alignment measures how well the classes are clustered by the metric induced by the kernel. The maximization of this measure can be interpreted as the choice of a metric which clusters the classes together while separating different classes as good as possible. This goal is expressed in the numerator $\langle yy^T, K\rangle$ where the inner product K_{ij} of each pair x_i, x_j of examples is aimed to be similar to the product $y_i y_j$ of their labels, that is, the inner product should be large for coinciding and small (or negative) for different classes. The denominator basically ensures that $\hat{A}$ is invariant under scaling of the kernel. With respect to the inner product on kernel Gram matrices the KTA is the cosine of the angle between the Gram matrix K induced by the kernel and the "optimal" Gram matrix yy^T corresponding to perfect clustering of the data. For normalized kernels like the Gaussian kernel it is possible to drop the denominator. The resulting measure $P = \langle yy^T, K\rangle$ is known as kernel polarization [Baram, 2005].

The KTA is completely independent of the actual learning machine used. On the one hand this is an advantage as the same kernel can be plugged into multiple learning machines. Further the derivative of the KTA is cheap to compute and can be used for extremely efficient gradient-based adaptation of the kernel parameters, see Igel et al. [2007].

On the other hand there is hope that machine specific objective functions can lead to a choice of the kernel function that best suites that particular type of machine. The independence of the learning machine makes it necessary to revert to geometrical intuition instead of relying on learning performance bounds. Practically the independence of the type of learning machine has the consequence that it is not possible to choose the SVM regularization parameter C by maximizing $\hat{A}$.

9.1.7 Bootstrapping

Let us return to the cross-validation procedure. As discussed above the computation of the cross-validation error involves each example exactly once as a validation example, breaking the asymmetry between training and hold-out examples. However, the split of the data is not canonical and we observe that the resulting objective function can strongly depend on the split of the dataset into subsets.

Conceptually it is easy to overcome this problem. Let us define the set

$$J_n = \left\{ I \subset \{1, \ldots, \ell\} \;\middle|\; |I| = n \right\}$$

of index subsets of size n, leading to the objective function

$$\bar{B} = \frac{1}{|J_n|} \sum_{I \in J_n} \left(\sum_{i \in I^C} L(x_i, y_i, h_I(x_i)) \right)$$

where L is a loss function and h_I is the hypothesis constructed from the data indexed by I. Each summand of this objective function computes the hold-out error of the hold-out set I^C, denoting the complement of I in $\{1, \ldots, \ell\}$, evaluated on the hypothesis h_I. In the style of the k-fold cross-validation procedure we can choose $n = \lfloor \frac{k-1}{k} \ell \rfloor$, and, as usual, we consider $k = 5$ as the default.

This objective function has the conceptual advantage to be completely symmetric w.r.t. the split of the dataset into training and hold-out set and computes the probably best possible estimate of the generalization error of a machine trained on n examples. It has the disadvantage that the set J_n grows like $|J_n| = \binom{\ell}{n}$ which is clearly computationally intractable.

Instead we introduce the random variable

$$\hat{B} : J_n \to \mathbb{R} \qquad I \mapsto \sum_{i \in I^C} L(x_i, y_i, h_I(x_i))$$

with the uniform distribution for $I \in J_n$. For each fixed training set I we get the hold-out objective function $\hat{B}(I)$ which is a function of the hyperparameters. We write $\hat{B}$ for the randomized objective function which picks a random $I \in J_n$ for each of its evaluations. It clearly holds $\mathbb{E}[\hat{B}] = \bar{B}$. This way we are able to avoid a systematic bias resulting from a fixed split of the data like in the cross-validation procedure, at the cost of a randomized objective function. We will refer to this objective function with 0-1-loss as the bootstrapping error. For the minimization of $\bar{B}$ based on evaluations of $\hat{B}$ we need a search strategy that can deal with a non-differentiable and noisy objective function. This randomized objective functions is as cheap to evaluate as the simple hold-out error, but we have to be prepared for relatively long optimization runs due to the need for the search algorithm to handle the uncertainty in the objective function evaluations, for example when it is necessary to average over many evaluation in order to obtain sufficiently reliable information. Therefore, a search algorithm that reliably identifies a minimum of $\bar{B}$ is usually computationally infeasible for large datasets.

Of course there are a lot of ways conceptually in between the randomized hold-out error $\hat{B}$, standard cross-validation, and full bootstrapping. We can choose a subset of J_n of considerably smaller size (which should be as symmetric as possible), or take the mean over a few index sets sampled from J_n, possibly again in a symmetrized fashion. The most straight-forward example is to randomly pick a new split of the dataset for each evaluation of the cross-validation error. The resulting objective function requires a search strategy that can deal with uncertain function evaluations, just like for the minimization of $\hat{B}$. We could even define a (weighted) mean over all choices of n. Such proceedings can lead to a large number of deterministic or randomized objective functions.

A concept related to the bootstrapping error, but leading to a differentiable objective function with an interesting probabilistic interpretation, will be introduced in Section 9.2.2.

9.2 Probabilistic Interpretation of SVMs

The only noticeable approach in the machine learning literature to obtain probabilities $P(\text{class} \,|\, \text{input})$ from the outputs of an SVM hypothesis is the work by Platt [1999]. However, Platt's motivation was not the model selection problem, but to obtain a uniform way to post-process SVM outputs. In this section we will discuss why such a probabilistic approach can be helpful to assess the model selection problem. Then we make use of the probabilistic interpretation of the SVM outputs to construct model selection objective functions that considerably differ from the approaches presented in the previous section.

There is a number of good reasons for a probabilistic interpretation of the output of a support vector machine classifier. We may simply want to know how "sure" the hypothesis is of its decision if the function $\langle w, \Phi(x) \rangle + b$ outputs a particular value for the given sample x. Usually we just use the sign of this value for prediction, but sometimes it is useful to ask for the probability that this decision is correct, or for the probability of a sample being of a particular class. This happens in a number of situations:

- If we know the probability of correct prediction for all basic classifiers in an ensemble of learning machines we can combine their outputs with simple probabilistic rules (under the assumption of independence) to obtain an optimal combination of machines.
- Even if we have a probabilistic output only for a subset of machines in an ensemble, we can still adapt the voting scheme used to combine the single decisions by adjusting the weights given to the votes of the probabilistic machines accordingly.
- If we want to compare classifiers it is important to know which machine is how sure, for example, about a whole test set. This is helpful for the comparison of different types of machines, but also for a single machine trained with different hyperparameters. If we do not apply the sign operation to the SVM hypothesis we already get real valued outputs. However, these are not normalized, such that the comparison of the absolute values output by two machines, possibly trained with different kernels or regularization parameters, is meaningless. In contrast, probabilities of being correct are comparable among different machines. This is why a probabilistic interpretation of the outputs of SVM hypotheses may help to solve the model selection problem.
- If a machine outputs the probability of an example belonging to a class we can compute the likelihood of this machine w.r.t. a dataset. Then Bayesian methods become directly applicable as it is straight forward how to incorporate prior distributions on the hyperparameters into the decision. Such priors can encode prior knowledge about the model selection problem at hand.

In the following we will always consider binary classification. Then we are interested in the probability of being a positive example $P(y = +1 \,|\, x)$, while we of course have the identity $P(y = -1 \,|\, x) = 1 - P(y = +1 \,|\, x)$ for the negative class. In general these probabilities will depend on x in a way which is not captured by the support vector machine output $\langle w, \Phi(x) \rangle + b$. The method proposed by Vapnik [1998, Section 11.11], takes this into account, but it is quite complicated, rarely used, and requires a lot of overhead compared to SVM training.

9.2.1 Platt's Method

For the reasons of simplicity and robustness Platt [1999] formulates the idea to model the above probability based on the output of the function $x \mapsto \langle w, \Phi(x) \rangle + b$ only. That

is, he uses a model of the form $P(y = +1 \,|\, x) = \sigma(\langle w, \Phi(x)\rangle + b)$. Of course it is much simpler to estimate a single function $\sigma : \mathbb{R} \to [0, 1]$ than a general model $X \to [0, 1]$ for the probability of positive label.

The basic interpretation underlying this model is that for a large absolute value of the output, corresponding to an example far from the decision boundary, the support vector machine can be quite sure about its prediction. Therefore it is plausible to restrict the above model to monotonically increasing functions σ. We may further require $\sigma(0) = \frac{1}{2}$ as we expect the chance of correct prediction to be worst exactly on the decision boundary. However, Platt does not impose the last restriction. Still it is not clear how sure the machine can be about its prediction for a given distance from the hyperplane. This information needs to be learned from the data and will be encoded in the slope of the function σ.

As the true function $x \mapsto P(y = +1 \,|\, x)$ does in most cases not factor over the SVM hypothesis $\langle w, \Phi(x)\rangle + b$ this model sacrifices some accuracy a priori. On the other hand learning σ is more robust and the prediction of the probability is cheap as we already know the SVM hypothesis and a model of σ will usually be fast to evaluate.

For a given dataset $D = \big((x_1, y_1), \ldots, (x_N, y_N)\big)$ of N i.i.d. samples from ν the likelihood of the above model is

$$\prod_{y_i=+1} \Big(\sigma(\langle w, \Phi(x_i)\rangle + b)\Big) \cdot \prod_{y_i=-1} \Big(1 - \sigma(\langle w, \Phi(x_i)\rangle + b)\Big)$$

with logarithm

$$\mathcal{L} = \sum_{y_i=+1} \log\Big(\sigma(\langle w, \Phi(x_i)\rangle + b)\Big) + \sum_{y_i=-1} \log\Big(1 - \sigma(\langle w, \Phi(x_i)\rangle + b)\Big) \ .$$

Platt proposes the parametric form

$$\sigma_{r,s}(t) = \frac{1}{1 + \exp(st + r)}$$

of sigmoidal functions. With the constraint $s < 0$ these functions are monotonically increasing. This model was probably chosen due to its simplicity. Its plausibility is questionable, but in simple experiments the error rates it predicts can be fitted quite well. However, the model has the freedom to assign arbitrary probabilities to the SVM decision boundary $t = 0$ for $r \neq 0$.

To obtain a probabilistic model from a readily trained SVM hypothesis $\langle w, \Phi(x)\rangle + b$ we have to fix the parameters r and s of the sigmoid first. In general we should not use the maximum likelihood estimate for the training dataset, as the outputs of the hypothesis on these points are not typical. In contrast, the SVM will output the exact values $+1$ or -1 for many examples, as they are free support vectors, which is of course not expected for test examples. Thus this approach can lead to a strongly biased fit.

We can use a hold-out set not used for training instead, such that the data used to fix r and s are independent of the training data. This has the disadvantage that we need to train the machine on a reduced training set in order to optimize the sigmoid parameters and that the likelihood can only be estimated on the smaller hold-out set. To reduce the second effect Platt proposes to use a cross-validation procedure. If the parameters are once determined the final SVM can be trained on the whole training set.

Further, Platt regularizes the training process as the sigmoid can still overfit in the cross-validation procedure. He uses a Bayesian regularization with a uniform prior on the probability of positive class (see Platt, 1999) reducing the steepness of the resulting maximum a-posteriori estimate of the function σ.

9.2.2 Combination with Bootstrapping

Experiments indicate that Platt had good reasons to regularize the parameter selection of the sigmoids. Indeed, the parameter $s < 0$ of the sigmoids can vary significantly over different data splits used for cross-validation. As already discussed in Section 9.1.7 we can eliminate overfitting to the particular split used in a cross-validation procedure with a simple bootstrapping technique, that is, by taking the mean over all possible splits of the data. Then the variations coming from the maximum likelihood sigmoids average out and we do not need to introduce any priors for regularization.

For conceptual reasons we will use the simpler sigmoid model $\sigma_s(t) = \frac{1}{1+\exp(st)}$ in the following, such that the parameter r in Platt's more general model is fixed to zero. This model always fulfills $\sigma(0) = \frac{1}{2}$ which is a reasonable assumption that in particular holds true in the limit $\ell \to \infty$ if the prerequisites for universal consistency are fulfilled (see Section 2.4).

With the above ingredients we define a probabilistic SVM as a standard 1-norm SVM together with a sigmoid probability model σ_s. The hyperparameters of this model are the kernel parameters, the complexity control parameter C and the sigmoid parameter s. Now we ask how to select these parameters based on a training dataset.

A simple answer is that we can maximize the mean log-likelihood

$$\bar{\mathcal{L}} = \frac{1}{|J_n|} \sum_{I \in J_n} \mathcal{L}(I^C, \sigma_s, f_{C,\theta,I})$$

with the set J_n defined in Section 9.1.7. Here, $f_{C,\theta,I}(x) = \langle w, \Phi(x) \rangle + b$ is the function computed by the SVM trained on the examples indexed by I with kernel k_θ and regularization parameter C. Then $\mathcal{L}(I^C, \sigma_s, f_{C,\theta,I})$ denotes the likelihood of the model $\sigma_s \circ f_{C,\theta,I} : X \to [0,1]$, evaluated on the hold-out set indexed by I^C.

The computational complexity is analogously to the bootstrapping technique introduced in Section 9.1.7 and thus intractable. Therefore we define the randomized objective function

$$\hat{\mathcal{L}} = \mathcal{L}(I^C, \sigma_s, f_{C,\theta,I})$$

with random index set I drawn i.i.d. from the uniform distribution on J_n. The construction is completely analogous to the randomized bootstrapping error introduced above. This novel model selection objective function inherits the advantage of symmetry together with the disadvantage of randomization from the bootstrapping error, but has the additional advantages of a natural probabilistic interpretation and the possibility to apply gradient-based optimization, as $\hat{\mathcal{L}}$ is differentiable whenever $f_{C,\theta,I}$ is.

When searching for good hyperparameters for our model we can in practice restrict ourselves to the parameters C and θ. This is because for a given function $f_{C,\theta,I}$, which is the result of SVM training, it is computationally cheap to compute the corresponding optimal value of s by gradient ascent on the log-likelihood. Then, for an efficient gradient-based maximization of $\bar{\mathcal{L}}$ we need to compute the derivatives of $\hat{\mathcal{L}}$ w.r.t. C and θ. In Chapter 11 we will present an algorithm for gradient ascent/descent that can deal with the uncertainty due to the random index set I used for each gradient computation.

Besides the incorporation of the randomization technique to get rid of systematic effects resulting from a fixed split of the dataset, the main difference to Platt's original work is that we use the likelihood as a model selection objective function. This point of view clearly goes beyond the need to post-process the SVM outputs, for example to combine them with other machines.

A related approach was proposed by Keerthi et al. [2007] for smoothing hold-out based error measures like the cross-validation error. However, in this work the sigmoidal

approach proposed by Platt is used just as a computational trick to smoothen the objective function in order to make efficient gradient-based optimization applicable, and in fact the resulting objective functions differ from the likelihood function. In contrast, the main motivation for our approach comes from the probabilistic interpretation of the likelihood performance measure, with the goal to incorporate prior knowledge encoded in prior distributions over the hyperparameters, as discussed in the following.

9.2.3 Combination with a Prior

The objective function $\hat{\mathcal{L}}$ has a natural interpretation as the log-likelihood of the hold-out set. Therefore it is straight forward to incorporate prior knowledge or to impose regularization in terms of a prior distribution on the hyperparameters. However, our application of prior distributions is fundamentally different from most other probabilistic interpretations found in the literature.

Most Bayesian approaches to support vector machines, see for example the work of Gold and Sollich [2003] and references therein, introduce a probabilistic interpretation through the back-door by interpreting the primal SVM optimization objective or some other quantity as a negative log posterior. This leads to a nice framework if some issues like non-normalized measures are ignored. Many approaches use priors on all kinds of parameters to integrate these parameters out (and thus to get rid of them at the cost of the introduction of a prior). Our approach is different in all of these respects. First, we start with the meaningful probabilistic interpretation of the outputs of the SVM hypothesis, that is, with the probability of misclassification, without introducing ill-founded probabilistic interpretations of other quantities. Second, we do not replace parameters with priors, but only alter the model selection objective function, just like the introduction of a complexity measure changes the objective function of empirical risk minimization. A similar approach, defining priors directly on the hyperparameters, was developed by Cawley and Talbot [2007] for the least squares SVM.

Assume we are given a prior distribution $P(C, \theta)$ on the hyperparameters, and let $\tilde{D} = \big((x_1, y_1), \ldots, (x_N, y_N)\big)$ be a validation dataset. We know from Bayes rule that the posterior probability $P((C, \theta) \,|\, \tilde{D})$ of the validation dataset is proportional to the likelihood $P(\tilde{D} \,|\, (C, \theta))$ times the prior probability $P(C, \theta)$ of the parameters, where the constant of proportionality is just the probability of observing $\tilde{D}$ (which is not in our hands). Then, up to a constant, the logarithm of the posterior is the log-likelihood plus the logarithm of the prior, such that the log-posterior objective function is the sum $\mathcal{L} + \log(P(C, \theta))$. Depending on the prior the second summand can be interpreted as encoding prior knowledge or imposing regularization, or both.

In contrast to the structural risk minimization scenario it is satisfactory that we do not need a parameter to fix the relative importance of the likelihood and the prior, as the normalization of both summands is automatically obtained from the normalization of the distributions to a total probability of one. Still there is a large freedom in the choice of the prior.

9.3 Derivatives of the Model Selection Objectives

Now we have a large repertoire of model selection objective functions, and most of them are differentiable w.r.t. the SVM hyperparameters nearly everywhere, such that gradient-based optimization algorithms are in principle applicable to approximate their optimal points. To apply any gradient-based search strategy we have to compute the derivative of the objective function. As this is non-trivial in some cases we will explicitly state these derivatives in this section. In particular we summarize how to compute the derivatives of

the radius margin quotient and the kernel target alignment w.r.t. the kernel (or a generic parameter θ of the family of kernels), and the derivatives of the span bound and the logarithmic likelihood w.r.t. the regularization parameter C of the 1-norm SVM and the kernel parameters. These calculations are largely based on articles by Chapelle et al. [2002] and Keerthi et al. [2007].

9.3.1 Derivative of the 1-norm SVM Hypothesis

The derivative of the parameters (α, b) of the 1-norm SVM hypothesis is a prerequisite for the computation of the derivatives of the span bound and the likelihood. Throughout this section we assume that the parameterization of the family of kernel functions is continuously differentiable, that is, the derivative $\frac{\partial k_\theta(x,x')}{\partial \theta}$ is well defined and continuous. Then the parameters (α, b) happen to depend continuously on the regularization parameter C and the kernel, and the dependency is differentiable up to a low-dimensional subset.[2] This subset is the set of hyperparameter settings such that a vector with coefficient α_n at the lower or upper bound lies exactly on the margin. As this subset is low-dimensional we just consider the case that all training examples x_n that happen to lie exactly on the margin, fulfilling $y_n f(x_n) = 1$, are *free* support vectors.

For the calculation we will assume that the solution α of the quadratic program (4.1) is exact, that is, exactly fulfills the KKT conditions, which are only fulfilled with accuracy $\varepsilon > 0$ in practice. Although this prerequisite is of course not fulfilled it turns out that the deviations are negligible or average out when solving the model selection problem up to a reasonable accuracy.

Along the lines of the computation carried out by Keerthi et al. [2007] (see also Chapelle et al. [2002] for the 2-norm case) we derive an analytic expression for the unbounded variables of the 1-norm SVM classifier. We define the index sets

$$\begin{aligned} u =& \left\{ i \in \{1, \ldots, \ell\} \mid 0 < y_i \alpha_i < C \Leftrightarrow y_i f(x_i) = 1 \right\} \\ g =& \left\{ i \in \{1, \ldots, \ell\} \mid y_i \alpha_i = C \Leftrightarrow y_i f(x_i) < 1 \right\} \\ n =& \left\{ i \in \{1, \ldots, \ell\} \mid \alpha_i = 0 \Leftrightarrow y_i f(x_i) > 1 \right\} . \end{aligned}$$

The above assumption that only free support vectors lie on the margin guarantees that the conditions defining the sets are indeed equivalent. The same assumption makes sure that the sets u, g, and n do not change in an open neighborhood of the current hyperparameters. We immediately conclude that the derivatives of the coefficients α_i corresponding to non-support vectors vanish. For coefficients α_i corresponding to bounded support vectors the derivative w.r.t. a kernel parameter vanishes, too, while the derivative w.r.t. C is just y_i. It remains to compute the derivatives of the parameters (α_u, b).

Let $\mathbf{1}_u$ and $\mathbf{1}_g$ denote vectors of all ones of the corresponding dimensions $|u|$ and $|g|$. We split the kernel matrix and the label vector into sub-matrices and sub-vectors

$$K = \begin{pmatrix} K_{uu} & K_{ug} & K_{un} \\ K_{gu} & K_{gg} & K_{gn} \\ K_{nu} & K_{ng} & K_{nn} \end{pmatrix} \qquad y = \begin{pmatrix} y_u \\ y_g \\ y_n \end{pmatrix}$$

where we order the variables according to u, g, and n. From the equality constraint and the fact that unbounded support vectors evaluate exactly to their labels we get the

[2] If the optimal solution of problem (4.1) is not unique the definition of the dependency of α from the hyperparameters is problematic. As the SVM hypothesis does not depend on the choice of the optimal α the dependency of the hypothesis on the hyperparameters is anyway well-defined.

equation

$$\begin{pmatrix} K_{uu} & K_{gu} & \mathbf{1}_u \\ \mathbf{1}_u^T & \mathbf{1}_g^T & 0 \end{pmatrix} \begin{pmatrix} \alpha_u \\ \alpha_g \\ b \end{pmatrix} = \begin{pmatrix} y_u \\ 0 \end{pmatrix} \quad .$$

With the notation

$$H = \begin{pmatrix} K_{uu} & \mathbf{1}_u \\ \mathbf{1}_u^T & 0 \end{pmatrix}$$

we get the expression

$$\begin{pmatrix} \alpha_u \\ b \end{pmatrix} = H^{-1} \left[\begin{pmatrix} y_u \\ 0 \end{pmatrix} - \begin{pmatrix} K_{gu} \\ \mathbf{1}_g^T \end{pmatrix} \alpha_g \right]$$

for the unbounded SVM parameters. Note that we have $\alpha_g = C \cdot y_g$ for the bounded variables.

Let $\frac{\partial v}{\partial \theta}$ denote the component-wise partial derivative of a vector or a matrix v. With the identity[3] $\left(\frac{\partial H^{-1}}{\partial \theta}\right) = -H^{-1} \frac{\partial H}{\partial \theta} H^{-1}$ we get the derivative

$$\begin{aligned} \frac{\partial \begin{pmatrix} \alpha_u \\ b \end{pmatrix}}{\partial \theta} &= \frac{\partial H^{-1}}{\partial \theta} \left[\begin{pmatrix} y_u \\ 0 \end{pmatrix} - C \begin{pmatrix} K_{gu} \\ \mathbf{1}_g \end{pmatrix} y_g \right] - C \cdot H^{-1} \begin{pmatrix} \frac{\partial K_{gu}}{\partial \theta} \\ 0 \end{pmatrix} y_g \\ &= -H^{-1} \frac{\partial H}{\partial \theta} \underbrace{H^{-1} \left[\begin{pmatrix} y_u \\ 0 \end{pmatrix} - C \begin{pmatrix} K_{gu} \\ \mathbf{1}_g \end{pmatrix} y_g \right]}_{= \begin{pmatrix} \alpha_u \\ b \end{pmatrix}} - C \cdot H^{-1} \begin{pmatrix} \frac{\partial K_{gu}}{\partial \theta} \\ 0 \end{pmatrix} y_g \\ &= -H^{-1} \left[\frac{\partial H}{\partial \theta} \begin{pmatrix} \alpha_u \\ b \end{pmatrix} + C \begin{pmatrix} \frac{\partial K_{gu}}{\partial \theta} \\ 0 \end{pmatrix} y_g \right] \end{aligned}$$

w.r.t. a kernel parameter θ. For the dependency on the regularization parameter C we get

$$\frac{\partial \begin{pmatrix} \alpha_u \\ b \end{pmatrix}}{\partial C} = -H^{-1} \left[\begin{pmatrix} K_{gu} \\ \mathbf{1}_g^T \end{pmatrix} y_g \right]$$

for the unbounded variables. To compute the derivative of the output of the SVM hypothesis $f(x)$ on a given input x we have to plug the above terms into the derivative of the affine linear kernel expansion $f(x) = \sum_{i=1}^{\ell} \alpha_i k(x, x_i) + b$. We get

$$\frac{\partial f}{\partial \theta}(x) = \left[\sum_{i=1}^{\ell} \frac{\partial \alpha_i}{\partial \theta} k(x, x_i) + \frac{\partial k(x, x_i)}{\partial \theta} \alpha_i \right] + \frac{\partial b}{\partial \theta}$$

for a kernel parameter θ and

$$\frac{\partial f}{\partial C}(x) = \left[\sum_{i=1}^{\ell} \frac{\partial \alpha_i}{\partial C} k(x, x_i) \right] + \frac{\partial b}{\partial C}$$

[3]The formula can be obtained by calculating the derivative of the unit matrix in the form HH^{-1} (refer to the article by Chapelle et al., 2002).

for the regularization parameter C.

In the worst case the complexity of this calculation is governed by the computation of H^{-1}, taking $\mathcal{O}(|u|^3)$ operations. Luckily for most problems we have $|u| \ll \ell$, but as we want to apply this computation for model selection we have to be prepared for intermediate (usually non-optimal) hyperparameter settings for which most examples become free support vectors. Then the above computation is intractable for large datasets, and we have to switch to model selection strategies that get along without the derivative of the hypothesis.

9.3.2 Derivative of the Radius Margin Quotient

The radius of the smallest ball containing all training examples is given by the solution of the quadratic program

$$\begin{aligned} \underset{\beta}{\text{maximize}} \quad & R^2 = \sum_{n=1}^{\ell} \beta_n K_{nn} - \beta^T K \beta \\ \text{s.t.} \quad & \mathbf{1}^T \beta = 1 \\ \text{and} \quad & \beta_n \geq 0 \quad \forall \ 1 \leq n \leq \ell \end{aligned}$$

for $\beta \in \mathbb{R}^\ell$, (see Vapnik, 1998). Chapelle et al. [2002] have shown that, under the assumption that the solutions α and β of the quadratic programs for the computation of the margin and the radius are exact, the derivatives are given by

$$\frac{\partial \|w\|^2}{\partial \theta} = \frac{\partial (1/\rho^2)}{\partial \theta} = \alpha^T \frac{\partial K}{\partial \theta} \alpha$$

and

$$\frac{\partial R^2}{\partial \theta} = \sum_{n=1}^{\ell} \beta_n \frac{\partial K_{nn}}{\partial \theta} - \beta^T \frac{\partial K}{\partial \theta} \beta \ .$$

We conclude

$$\begin{aligned} \frac{\partial (R^2/\rho^2)}{\partial \theta} &= R^2 \frac{\partial (1/\rho^2)}{\partial \theta} + \frac{1}{\rho^2} \frac{\partial R^2}{\partial \theta} \\ &= \Big(\sum_{n=1}^{\ell} \beta_n K_{nn} - \beta^T K \beta \Big) \Big(\alpha^T \frac{\partial K}{\partial \theta} \alpha \Big) + \Big(y^T \alpha \Big) \Big(\sum_{n=1}^{\ell} \beta_n \frac{\partial K_{nn}}{\partial \theta} - \beta^T \frac{\partial K}{\partial \theta} \beta \Big) \ . \end{aligned}$$

It is remarkable that, once the radius margin quotient is computed and the quadratic programs are solved, the derivative is obtained more or less as a by-product. All we need is the derivative of the kernel w.r.t. its parameters. Therefore the complexity of this computation scales like two solution of convex quadratic programs plus $\mathcal{O}(\ell^2)$ operations per independent parameter θ (where we assume that the derivative of the kernel w.r.t. its parameters scales at most linear with the number of parameters).

9.3.3 Derivative of the Kernel Target Alignment

The computation of the derivative of the kernel target alignment

$$\hat{A} = \frac{\langle yy^T, K \rangle}{\sqrt{\langle yy^T, yy^T \rangle \langle K, K \rangle}} = \frac{\sum_{i,j=1}^{\ell} y_i y_j K_{ij}}{\ell \sqrt{\sum_{i,j=1}^{\ell} (K_{ij})^2}}$$

is straight forward. By quotient and chain rule we get

$$\frac{\partial \hat{A}}{\partial \theta} = \frac{\Big(\sum\limits_{i,j=1}^{\ell} y_i y_j \frac{\partial K_{ij}}{\partial \theta}\Big)\sqrt{\sum\limits_{i,j=1}^{\ell}(K_{ij})^2} - \Big(\sum\limits_{i,j=1}^{\ell} y_i y_j K_{ij}\Big)\frac{\sum\limits_{i,j=1}^{\ell} 2K_{ij}\frac{\partial K_{ij}}{\partial \theta}}{2\sqrt{\sum\limits_{i,j=1}^{\ell}(K_{ij})^2}}}{\ell \sum\limits_{i,j=1}^{\ell}(K_{ij})^2}$$

$$= \frac{\Big(\sum\limits_{i,j=1}^{\ell} y_i y_j \frac{\partial K_{ij}}{\partial \theta}\Big)\Big(\sum\limits_{i,j=1}^{\ell}(K_{ij})^2\Big) - \Big(\sum\limits_{i,j=1}^{\ell} y_i y_j K_{ij}\Big)\Big(\sum\limits_{i,j=1}^{\ell} K_{ij}\frac{\partial K_{ij}}{\partial \theta}\Big)}{\ell\Big(\sum\limits_{i,j=1}^{\ell}(K_{ij})^2\Big)^{3/2}}$$

whose computation takes $\mathcal{O}(\ell^2)$ operations per parameter θ. Because this computation does not involve any optimization problems it is very fast in practice.

9.3.4 Derivative of the Span Bound

It is trivial to reduce the derivative

$$\frac{\partial S}{\partial \theta} = \sum_{\substack{n\in\{1,\dots,\ell\}\\ y_n f^{\hat{n}}(x_n)>0}} \left(f^{\hat{n}}(x_n)\frac{\partial f}{\partial \theta}(x_n) - \frac{\partial f^{\hat{n}}}{\partial \theta}(x_n) f(x_n)\right) \Big/ (f(x_n))^2$$

of the span bound to the derivative of the hypotheses f and $f^{\hat{n}}$ computed in Section 9.3.1. The formula for the regularization parameter C instead of a kernel parameter θ is analogous. In a naïve implementation the computation of this derivative costs up to $\mathcal{O}(\ell^4)$ operations per parameter, which is prohibitive even for medium size datasets with several thousands of training examples.

9.3.5 Derivative of Likelihood and Posterior

For fixed $I \in J_n$ we split the complement set I^C into the subsets $I^C_+ = \{i \in I^C \,|\, y_i = +1\}$ and $I^C_- = \{i \in I^C \,|\, y_i = -1\}$. Then the maximum log-likelihood can be written as

$$\hat{\mathcal{L}} \;=\; \max\left\{ \sum_{i\in I^C_+} \log\Big(\sigma_s(f_{C,\theta,I}(x_i))\Big) + \sum_{i\in I^C_-} \log\Big(1-\sigma_s(f_{C,\theta,I}(x_i))\Big) \;\Bigg|\; s\in\mathbb{R}\right\} \;.$$

Let s^* denote the value of the sigmoid parameter for which the maximum is attained. We introduce the notations $z_i = f_{C,\theta,I}(x_i)$ and $p_i = \sigma_{s^*}(z_i)$ and, using $\frac{\partial \hat{\mathcal{L}}}{\partial s} = 0$ in the maximum, write the derivative in the form

$$\frac{\partial \hat{\mathcal{L}}}{\partial \theta} = \sum_{i\in I^C_+} \frac{1}{p_i}\cdot\frac{\partial p_i}{\partial z_i}\cdot\frac{\partial z_i}{\partial \theta} + \sum_{i\in I^C_-} \frac{1}{1-p_i}\cdot\frac{\partial p_i}{\partial z_i}\cdot\frac{\partial z_i}{\partial \theta} \;.$$

From the form of the sigmoid we obtain $\frac{\partial p_i}{\partial z_i} = -s^* p_i(1-p_i)$ and the remaining derivative $\frac{\partial z_i}{\partial \theta}$ of the SVM hypothesis w.r.t. the kernel parameter has already been computed in Section 9.3.1. The derivative w.r.t. C is completely analogous.

The log-posterior is just the sum of the log-likelihood and the logarithm of the prior. Of course, the derivative then depends on the density function of the prior distribution.

The computation of the derivatives of $f_{C,\theta,I}(x_i)$ w.r.t. the hyperparameters takes the lion's share of the time. Most priors are cheap to compute and therefore do not play a role for the overall runtime.

9.4 Multi-Modality of Model Selection Objectives

There is a general pitfall when we select the free hyperparameters of the model selection problem by optimizing an objective function, in particular if gradient-based search strategies are applied to approximate an optimal point: The objective functions are usually non-convex and can exhibit multiple local optima, such that the search strategy is endangered to converge to a local optimal point that turns out to be globally sub-optimal. In fact we can observe in experiments that such local optimal points can correspond to machines with clearly sub-optimal generalization error, down to the poor performance of random guessing or a simple majority vote. In this respect model selection for support vector machines is a much harder problem than machine training. In this section we take a closer look at the multi-modality of model selection objective functions, that is, at the presence of multiple local optima.

Because multiple local optima in general preclude efficient global search we should be prepared for sub-optimal solutions of the model selection problem. It is well known that some search strategies are more prone to getting stuck in local optima than others. Therefore it is impossible to judge objective function and search strategy independently.

Of course, model selection objective functions depend on the training data. Therefore it is easy to demonstrate the multi-modality of such functions on example problems. In this section we introduce a different approach. We prove the presence of local optima for a wide range of learning problems for two of the objective functions presented above. We will investigate the radius margin quotient and the kernel polarization measure for hard margin support vector machines with radial Gaussian kernel (2.1) in the limit of extremely narrow kernels. In this limit our results are nearly independent of the training dataset, allowing for a uniform analysis. The proceeding directly carries over to 2-norm SVMs. The presentation follows the article by Glasmachers [2006].

The results rely on certain properties of the kernel Gram matrix for radial Gaussian kernels. The kernel allows for the computation of inner products between pairs of training examples in $\mathcal{H}$. Usually, this is the only computation which is affordable in feature space, and thus model selection algorithms are restricted to the information represented in the positive semi-definite symmetric kernel Gram matrix $K \in \mathbb{R}^{\ell \times \ell}$ with

$$K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_i, x_j) \ .$$

Hence we consider quantities which depend on the training data only in terms of the kernel matrix K. This condition is fulfilled for all model selection objective functions presented above. In case of the radius margin quotient the functional relation is only implicitly available as the computation of R and ρ each requires the solution of a quadratic program depending on K.

9.4.1 Properties of the Radial Gaussian kernel

The derivative of the kernel (2.1) with respect to γ is

$$k'_\gamma(x, z) = \frac{\partial}{\partial \gamma} k_\gamma(x, z) = -\|x - z\|^2 \exp(-\gamma \|x - z\|^2) \ .$$

As already stated in Section 2.1.3 the Gaussian kernel is normalized, mapping the input space to the unit sphere in $\mathcal{H}$. Thus, the diagonal entries of K do not depend on γ. This is not the case for the off-diagonal entries. For every pair of different training examples $x \neq z$ it holds $k_\gamma(x, z) \in (0, 1)$. As γ increases, the feature vectors $\Phi_\gamma(x)$ and $\Phi_\gamma(z)$ become more and more orthogonal (that is, if γ is multiplied with two the kernel values are squared and thus become smaller). Let us now consider two pairs of training examples

$x_1 \neq z_1$ and $x_2 \neq z_2$ fulfilling the strict[4] inequality $\|x_1 - z_1\| < \|x_2 - z_2\|$. It follows $k_\gamma(x_1, z_1) > k_\gamma(x_2, z_2)$ for all $\gamma > 0$. An interesting property to observe here is

$$\frac{k_\gamma(x_2, z_2)}{k_\gamma(x_1, z_1)} = \left(\frac{k_1(x_2, z_2)}{k_1(x_1, z_1)}\right)^\gamma \quad , \tag{9.1}$$

that is by increasing γ the quotient becomes arbitrarily small. It is important to note that not only the absolute values of the off-diagonal entries of K decrease during this process, but that they become more and more different in the sense that their quotients decay (or explode if the pairs are switched).

We can compute a similar quotient using the derivative of the kernel function instead of the kernel function itself:

$$\frac{k'_\gamma(x_2, z_2)}{k'_\gamma(x_1, z_1)} = \left(\frac{\|x_1 - z_1\|^2}{\|x_2 - z_2\|^2}\right)^{\gamma-1} \cdot \left(\frac{k'_1(x_2, z_2)}{k'_1(x_1, z_1)}\right)^\gamma \quad . \tag{9.2}$$

For $\gamma \to \infty$ this quotient decays to zero even faster. We want to reserve the indices p and q for the pair of training examples with smallest[5] input space distance, that is,

$$(p, q) = \arg\min \left\{ \|x_i - x_j\| \;\middle|\; 1 \leq i < j \leq \ell \right\} \quad .$$

To stress the dependency of the model selection objectives from the finite number of kernel matrix entries depending on γ, we use the matrix notation

$$K_{ij}(\gamma) = k_\gamma(x_i, x_j) \quad \text{and} \quad K'_{ij}(\gamma) = k'_\gamma(x_i, x_j) \quad .$$

It is clear from equations (9.1) and (9.2) how these matrices look like in the limit case $\gamma \to \infty$. The diagonal entries are fixed to $K_{ii}(\gamma) = 1$ and $K'_{ii}(\gamma) = 0$, while the off-diagonal entries quickly decay to zero. Among the off-diagonal entries $K_{pq}(\gamma) = K_{qp}(\gamma)$ and $K'_{pq}(\gamma) = K'_{qp}(\gamma)$ become arbitrarily dominant over all other entries. This is formalized in the following lemma:

Lemma 9.2. *For the quotients of off-diagonal entries of K and K' it holds*

$$\lim_{\gamma\to\infty} \sum_{\substack{1\leq i<j\leq \ell \\ (i,j)\neq(p,q)}} K_{ij}(\gamma) \Big/ K_{pq}(\gamma) = 0 \quad \textit{and} \quad \lim_{\gamma\to\infty} \sum_{\substack{1\leq i<j\leq \ell \\ (i,j)\neq(p,q)}} K'_{ij}(\gamma) \Big/ K'_{pq}(\gamma) = 0 \quad .$$

Recall that model selection objective functions depend on the information represented in the kernel Gram matrix K. The above lemma states that among the non-trivial off-diagonal entries of K a single term becomes arbitrarily dominant over all others. Thus it is not surprising that the model selection objective functions considered below and their derivatives heavily depend on this single entry.

9.4.2 Limit Case Behavior of Model Selection Objectives

As a consequence of the form of the kernel matrix and its derivative we show that both radius margin quotient $(R/\rho)^2$ and kernel polarization P are governed by the pair of labels (y_p, y_q) if the kernel parameter γ becomes large. For the analysis of the radius margin quotient we introduce the notation

$$\ell_+ = \left| \left\{ n \in \{1, \ldots, \ell\} \;\middle|\; y_n = +1 \right\} \right| \quad \text{and} \quad \ell_- = \left| \left\{ n \in \{1, \ldots, \ell\} \;\middle|\; y_n = -1 \right\} \right| \quad .$$

[4] For simplicity we assume $\|x_1 - z_1\| \neq \|x_2 - z_2\|$ from now on.

[5] We assume these indices to be unique. A generalization to multiple pairs is straight forward.

Theorem 9.3. *Under the conditions $\ell_+ \geq 2$ and $\ell_- \geq 2$ the radius margin quotient R^2/ρ^2 has a local optimum (minimum) at the boundary $\gamma \to \infty$ if and only if $y_p \neq y_q$.*

Proof. We will compute the derivative of R^2/ρ^2 in the limit $\gamma \to \infty$. As stated in Section 9.3.2 the radius R can be obtained from the solution β^* of the quadratic problem

$$R^2 = \max_{\beta} \left(\sum_{i=1}^{\ell} \beta_i K_{ii}(\gamma) - \sum_{i,j=1}^{\ell} \beta_i \beta_j K_{ij}(\gamma) \right)$$

under the constraints $\sum_{i=1}^{\ell} \beta_i = 1$ and $\beta_i \geq 0 \, \forall \, i = 1, ..., \ell$. In the limit we have $\lim_{\gamma \to \infty} K_{ij}(\gamma) = \delta_{ij}$. The resulting objective and the constraints equally depend on all β_i. Thus all β_i^* take on the same value leaving only a one dimensional problem open. From the equality constraint we obtain the feasible solution $\beta_i^* = 1/\ell$, from which we compute $R^2 = 1 - 1/\ell$.

The maximum α^* of the dual SVM objective $y^T \alpha - \frac{1}{2} \alpha^T K(\gamma) \alpha$ under the equality constraint $\mathbf{1}^T \alpha = 0$ and the inequality constraints $y_n \alpha_n \geq 0 \; \forall \, n \in \{1, \ldots, \ell\}$ defines the margin by $1/\rho^2 = y^T \alpha^* = \sum_{i,j=1}^{\ell} \alpha_i^* \alpha_j^* K_{ij}(\gamma)$. After plugging in $K_{ij}(\gamma) = \delta_{ij}$, the resulting problem depends equally on all α_n with the same label y_n. From this observation we conclude that the solution is of the form $\alpha_n^* = \tilde{\alpha}_{y_n}$. The two dimensional problem is then reduced to one dimension by the equality constraint and can be solved setting the derivative of the objective to zero. We get the feasible solution

$$\alpha_n^* = \begin{cases} \tilde{\alpha}_{(+1)} = 2\ell_-/\ell & \text{if } y_n = +1 \\ \tilde{\alpha}_{(-1)} = -2\ell_+/\ell & \text{if } y_n = -1 \end{cases} \qquad \text{and thus obtain} \qquad 1/\rho^2 = 4 \cdot \frac{\ell_+ \ell_-}{\ell} \; .$$

We observe that all training examples are support vectors. Following Chapelle et al. [2002], the derivative of the radius margin quotient can be computed as

$$\begin{aligned} \frac{\partial}{\partial \gamma} \frac{R^2}{\rho^2} &= R^2 \left(-\sum_{i,j=1}^{\ell} \alpha_i^* \alpha_j^* K'_{ij}(\gamma) \right) + \frac{1}{\rho^2} \left(\sum_{i=1}^{\ell} \beta_i^* K'_{ii}(\gamma) - \sum_{i,j=1}^{\ell} \beta_i^* \beta_j^* K'_{ij}(\gamma) \right) \\ &= -2 \sum_{1 \leq i < j \leq \ell} \left(\alpha_i^* \alpha_j^* R^2 + \beta_i^* \beta_j^* / \rho^2 \right) \cdot K'_{ij}(\gamma) \; . \end{aligned}$$

From Lemma 9.2 we know that in the limit $\gamma \to \infty$ this sum is governed by the term

$$\alpha_p^* \alpha_q^* R^2 + \beta_p^* \beta_q^* / \rho^2 \; . \tag{9.3}$$

Note that due to $K'_{ij}(\gamma) < 0$ for all $\gamma > 0$ this term has the same sign as $\frac{\partial}{\partial \gamma} \frac{R^2}{\rho^2}$. From the prerequisites it follows $(\ell - 1)(\min(\ell_+, \ell_-))^2 > \ell_+ \ell_-$. Together with the limits of α^* and β^* computed above, we get the inequality

$$|\alpha_p^* \alpha_q^*| \, R^2 \;\; \geq \;\; \frac{4(\ell - 1)(\min(\ell_+, \ell_-))^2}{\ell^3} \;\; > \;\; \frac{4\ell_+ \ell_-}{\ell^3} = \beta_p^* \beta_q^* / \rho^2$$

which shows that the left summand of expression (9.3) dominates the right one. Thus the derivative of the radius margin quotient R^2/ρ^2 has the same sign as $\alpha_p^* \alpha_q^*$ which coincides with the sign of $y_p y_q$ and which is negative if and only if $y_p \neq y_q$. At the boundary $\gamma \to \infty$, a negative (positive) derivative indicates a minimum (maximum). □

Theorem 9.4. *The kernel polarization P has a local optimum (maximum) at the boundary $\gamma \to \infty$ if and only if $y_p \neq y_q$.*

Proof. We compute the derivative

$$\frac{\partial}{\partial\gamma}P \;=\; \sum_{i,j=1}^{\ell} y_i y_j K'_{ij}(\gamma) \;=\; 2\cdot\sum_{1\le i<j\le\ell} y_i y_j K'_{ij}(\gamma)$$

and observe from Lemma 9.2 that for large γ the term is governed by $y_p y_q K'_{pq}(\gamma)$. It is positive (negative) if and only if the labels y_p and y_q differ (are equal), indicating a maximum (minimum) at the boundary $\gamma\to\infty$. □

Although the kernel target alignment $\hat{A}$ differs from kernel polarization only by a normalization term, its analysis turns out to be more complicated.

The theorems show that near the boundary $\gamma\to\infty$ the term $y_p y_q$ controls the path taken by a gradient descent algorithm. It is clear that this term does not represent much information about the underlying distribution ν. At least for noisy distributions it is a highly random quantity. Further the proofs of the theorems show that lots of local optima may exist near the boundary, governed by label combinations of proximate training examples.

As parameter search is usually carried out over several orders of magnitude, boundary extrema may play a role. For the particular problem addressed by the above theorems it is possible to sail around the arising difficulties easily. For example, we can use a heuristic as proposed by Jaakkola et al. [1999] for an initial choice of the concentration parameter γ of the Gaussian kernel. With this initialization a search algorithm should avoid the boundary optima with high probability. Here we just want to sensitize for the fact that standard model selection objective functions are in general multi-modal.

9.4.3 Simulation Results

It is clear that the proofs poorly catch the true multi-modality of model selection objective functions. Besides the boundary optima computed in the proofs useful objective functions can exhibit lots of local optima far from the boundary, see for example the article by Friedrichs and Igel [2005]. The plots in Figure 9.1 prototypically illustrate the objective landscapes resulting from several of the objective functions introduced above.

The plots in this figure give a good impression of the typical effects we observe when applying the corresponding objectives functions to model selection. The rough error surface of the cross-validation error possesses lots of local optima and, what is worse (and not visible in this illustration) strongly depends on the particular data split. In contrast the LOO error uses a single canonical split, but the high variance of the measurements complicates its minimization.

Compared to these non-differentiable functions the span bound is much smoother, and we can observe that this accurate upper bound closely follows the LOO error. However, due to the typical gouges of the error surface it can easily mislead gradient-based optimization algorithms.

Obviously, the surface generated by the bootstrap error is in good correspondence with the test error. However, this measure is not differentiable, and note that the optimal value of the kernel parameter γ is shifted due to the reduced training set size. The same holds for the negative logarithmic likelihood which looks like a monotonically transformed version of the bootstrap error. It should enable us to come close to the optimum of the bootstrap error, but using more efficient gradient-based optimization.

Finally, the plot of the test error surface gives a good impression of the nature of the model selection problem. The rather large basin of well-generalizing solutions can easily be found, but to reliably identify good parameter values within this basin is a much harder task. The relatively flat but sub-optimal plateau on the right front, corresponding

to large values of γ (narrowly peaked kernels), with classifiers basically converging to the 1-nearest-neighbor solution, is more pronounced for noisy problems. Note that these plots give only little insight into the nature of the problem and the qualitative behavior of the objective functions in high-dimensional search spaces.

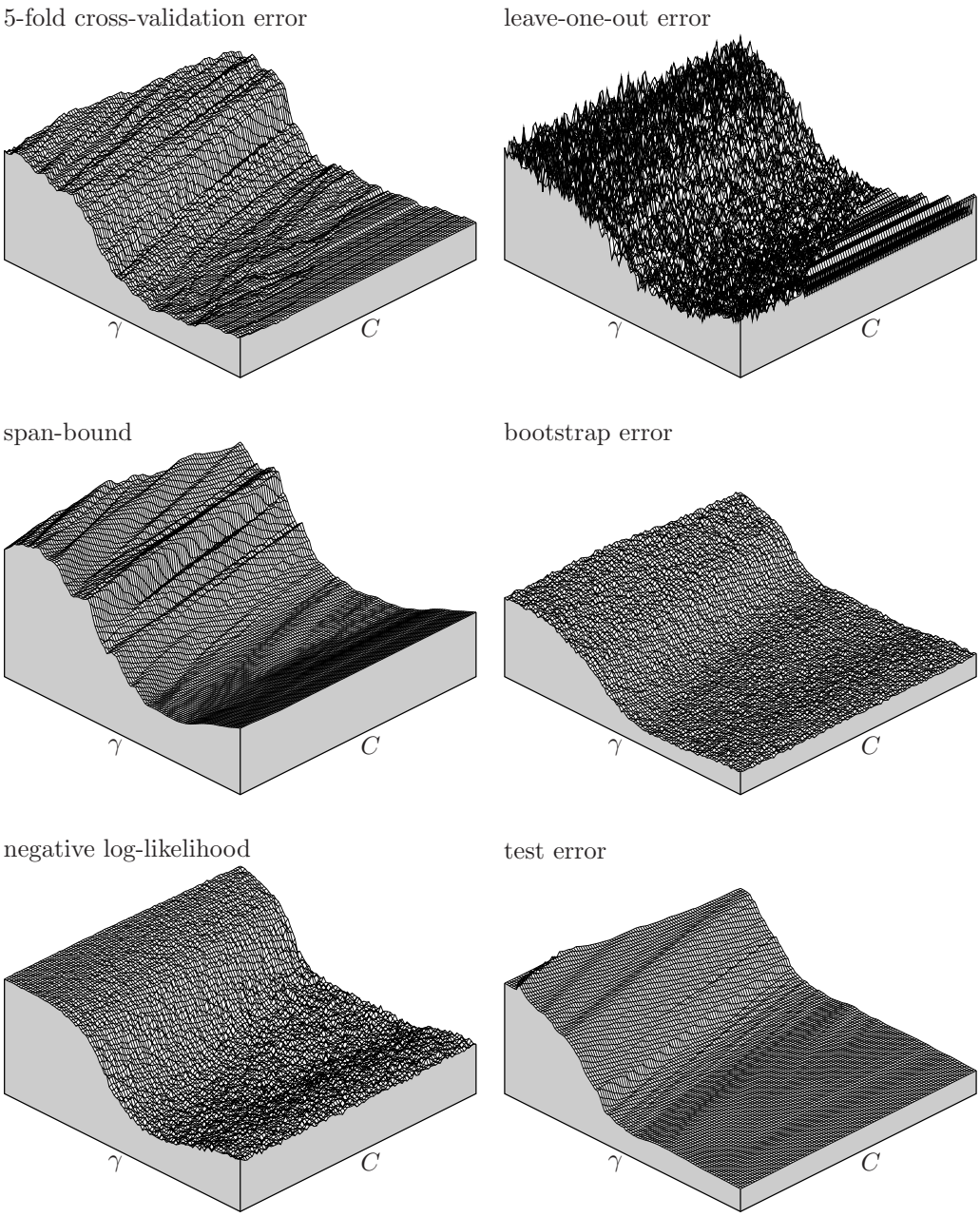

Figure 9.1: Illustration of the qualitative behavior of model selection objective functions. The parameters C of the 1-norm SVM and γ of the radial Gaussian kernel (2.1) are varied on a logarithmic grid of size 100×100 for a noise-free toy problem arising from the chess board distribution, see Section 3.4. As discussed in Section 9.1.7 it is impossible to compute the exact bootstrap error and the likelihood. To obtain these plots we took the means over 100 i.i.d. random draws per grid point (which amounts to one million machine trainings in total). For this reason there is a small component of white noise in the surface plots which could be further reduced by averaging over even more data partitions.

Chapter 10

Adaptation of General Gaussian Kernels

In this chapter we will further investigate properties of the radius margin quotient for model selection on the family (2.2) of Gaussian kernels with arbitrary covariance matrix. The main focus is on the introduction of a parameterization of the parameter manifold with a two-fold purpose: First it allows for unconstrained optimization, and second it induces a natural metric on this manifold, simplifying the task of the gradient-based search algorithm. The results presented here can be found in the article by Glasmachers and Igel [2005].

In this application we will consider hard margin support vector machines and concentrate on the effects of the adaptation of the kernel function. This is a meaningful proceeding in particular if we have prior knowledge indicating that the problem at hand is (nearly) noise free.

We want to select the parameters of a general Gaussian kernel[1] operating on an input space $X \subset \mathbb{R}^n$

$$k_B(x, z) = e^{-\frac{1}{2}(Bx-Bz)^T (Bx-Bz)} \quad ,$$

where $B \in \mathbb{R}^{n \times n}$ is symmetric and positive definite. The most elaborated methods for adjusting these kernels are gradient-based approaches [Chapelle et al., 2002, Chung et al., 2003, Gold and Sollich, 2003, Keerthi, 2002] that restrict B to diagonal matrices. However, only by dropping this restriction one can achieve invariance against linear transformations of the input space. Indeed it has been shown empirically by direct search that adapting the full covariance matrix of Gaussian kernels improves the performance on benchmark problems [Friedrichs and Igel, 2005]. Therefore, we compute the gradient for optimizing B in the manifold of positive definite symmetric matrices. We will decouple the adaptation of shape and orientation from the size of the kernel, which becomes necessary, for example, to overcome inherent problems when minimizing the radius margin quotient for hard margin support vector machines.

[1] The more common notation $k(x, z) = \exp(-\frac{1}{2}(x - z)^T Q (x - z))$ introduced in equation (2.2) is recovered by using the map $B \mapsto B^T B = Q$ which is a diffeomorphism on the manifold of symmetric positive definite matrices. This is important because many implementations of kernel-based methods support only Gaussian kernels with $B^T B = \lambda I$, where I is the unit matrix. Transforming the input space according to B and using the standard Gaussian kernel for training is a simple way to overcome this restriction in practice.

10.1 Kernel Parameterization

Most standard gradient descent techniques are well suited for unconstrained optimization. In order to ensure the feasibility of the parameter matrices we use a parameterization that allows for unconstrained optimization, avoiding the incorporation of constraint handling into existing optimization algorithms. Let

$$\mathfrak{m} := \{A \in \mathbb{R}^{n\times n} \mid A = A^T\}$$

be the real vector space of symmetric $n \times n$ matrices. The manifold

$$M := \{B \in \mathbb{R}^{n\times n} \mid x^T Bx > 0 \ \forall\, x \in \mathbb{R}^n \setminus \{0\} \text{ and } B = B^T\}$$

of positive definite symmetric $n \times n$ matrices can be parameterized with the exponential map[2]

$$\exp : \mathfrak{m} \to M \qquad A \mapsto \sum_{i=0}^{\infty} \frac{A^i}{i!} \ .$$

It holds $\exp(0) = I$ and $\frac{\partial}{\partial a_{ij}}\Big|_{A=0} \exp(A) = \frac{\partial A}{\partial a_{ij}}$ for each of the $n(n+1)/2$ hyperparameters $a_{ij} = a_{ji}$. The idea is to use at each point B a parameterization that maps the origin $0 \in \mathfrak{m}$ to B. We define the map

$$M \to M \qquad H \mapsto HBH \ ,$$

which is a diffeomorphism mapping the unit matrix I to B. To compute the gradient of the kernel k_B w.r.t. its parameters we express B by $\exp(A)B\exp(A)$ with $A = 0$. The trick is that in $A = 0$ the partial derivatives of $\exp(A)$ can be computed easily. We consider $\frac{\partial}{\partial a_{ij}}\Big|_{A=0} k_{\exp(A)B\exp(A)}(x, z)$ for each hyperparameter $a_{ij} = a_{ji}$:

$$\begin{aligned}
\xi_{ij} &:= \frac{\partial}{\partial a_{ij}}\bigg|_{A=0} k_{\exp(A)B\exp(A)}(x, z) \\
&= \frac{\partial}{\partial a_{ij}}\bigg|_{A=0} e^{-\frac{1}{2}\cdot (x-z)^T \exp(A)^T B^T \exp(A)^T \exp(A) B \exp(A)(x-z)} \\
&= -\frac{1}{2} k_B(x, z) \cdot (x - z)^T \frac{\partial}{\partial a_{ij}}\bigg|_{A=0} \Big(\exp(A)B(\exp(A))^2 B\exp(A)\Big)(x - z) \\
&= -\frac{1}{2} k_B(x, z) \cdot (x - z)^T \left(\frac{\partial A}{\partial a_{ij}} B^2 + 2B\frac{\partial A}{\partial a_{ij}} B + B^2 \frac{\partial A}{\partial a_{ij}}\right)(x - z) \ .
\end{aligned}$$

Setting $S := \dfrac{\partial A}{\partial a_{ij}} B + B\dfrac{\partial A}{\partial a_{ij}}$ and using $S = S^T$ it follows

$$\begin{aligned}
\xi_{ij} &= -\frac{1}{2} k_B(x, z) \cdot (x - z)^T (SB + BS)(x - z) \\
&= -\frac{1}{2} k_B(x, z) \cdot \left[(x - z)^T S(B(x - z)) + (B(x - z))^T S(x - z)\right] \\
&= -\frac{1}{2} k_B(x, z) \cdot (x - z)^T (S + S^T)(B(x - z)) \\
&= -k_B(x, z) \cdot (x - z)^T SB(x - z) \ .
\end{aligned}$$

[2] The manifold M is a subset of the Lie group $\mathrm{GL}_n(\mathbb{R})$ and the vector space $\mathfrak{m}$ a corresponding subspace of the Lie algebra $\mathfrak{gl}_n(\mathbb{R})$, cf. Baker [2002].

For example, a simple steepest-descent step with learning rate $\eta > 0$ would lead to the new matrix $\exp(-\eta\xi)B\exp(-\eta\xi)$.

In the following we will distinguish three qualitatively different and in fact independent properties of the kernel:

- We define the size (or mean width) of the kernel in terms of the smallest volume where a certain amount, say 95 %, of the kernel is concentrated. The size is controlled by the determinant of B, that is, by a single scalar value.
- The shape is the relative size of the eigenvalues of B, best described as an element of the projective space $\mathbb{P}(\mathbb{R}^n)$, decoupling this quantity from the kernel size. The dimensionality of this quantity is thus $n-1$.
- Finally the orientation is described by the eigenspaces of B, ordered by the magnitude of their corresponding eigenvalues. The orientation can be written as an orthogonal matrix with $n(n-1)/2$ free parameters.

This decomposition is naturally reflected by a decomposition of $\mathfrak{m}$ into orthogonal subspaces. Changes of the kernel size are controlled by the trace $\mathrm{tr}(\xi)$, the shape corresponds to the remaining diagonal of ξ, while non-zero off-diagonal entries of ξ change the orientation of the kernel.

As we will see, it is sometimes reasonable to restrict the adaptation to kernels with a fixed size. This is achieved by considering the one co-dimensional linear subspace

$$\mathfrak{n} := \{A \in \mathfrak{m} \mid \mathrm{tr}(A) = 0\} \subset \mathfrak{m}$$

of matrices A fulfilling $\det(\exp(A)) = 1$. The gradient descent can be restricted to this subspace[3] by orthogonally projecting the gradient matrices ξ to $\mathfrak{n}$, subtracting $\mathrm{tr}(\xi)/n$ from the diagonal entries of ξ.

10.2 Application to the Radius Margin Quotient

We design a simple test scenario as follows: The standard chess board distribution with 4×4 fields as introduced in Section 3.4 is transformed with a linear map

$$B = D^T \cdot \begin{pmatrix} 3 & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \cdot D$$

where in each trial an orthogonal matrix D is drawn from the uniform distribution on the (compact Lie group $\mathrm{O}_n(\mathbb{R})$ of) orthogonal $n \times n$ matrices. That is, $(B^{-1}x, y)$ is distributed according to the chess board distribution. Due to the symmetry of the chess board problem it is clear that the kernels with parameters close to B should lead to the best generalization. The dataset size of $\ell = 500$ used for the experimental evaluation is sufficient to sample the underlying distribution quite well, with approximately 31.25 training examples in each chess board field. As this problem is noise free it is well justified to learn it with a hard margin SVM. For this type of machine we can use efficient gradient descent of the radius margin quotient for model selection, see Section 9.1.4.

To get an insight into the effect of the adaptation of size, shape, and orientation of the kernels we consider five different kernel parameterizations:

[3] $\mathfrak{n} = \mathfrak{m} \cap \mathfrak{sl}_n(\mathbb{R})$ is the subspace of the Lie-Algebra $\mathfrak{sl}_n(\mathbb{R})$ corresponding to $\mathfrak{m}$.

	constraint	# variables	impact on kernel
(A)	$B = \lambda I$	1	size
(B)	B diagonal, $\det(B)$ const.	$n-1$	shape
(C)	B diagonal	n	size & shape
(D)	$\det(B)$ const.	$n(n+1)/2-1$	shape & orientation
(E)	none	$n(n+1)/2$	size, shape, & orientation

The optimization with the variant iRprop$^+$ [Igel and Hüsken, 2003] of the Rprop algorithm starts from an isotropic kernel whose width is initialized to the median of the distances from the positive examples to the negative examples, a heuristic suggested by Jaakkola et al. [1999]. In cases (B) and (D) the optimization is restricted to kernels of fixed size by projecting the derivatives to $\mathfrak{n}$. We measure the radius margin objective, the error on a test set, and the number of support vectors used to build the solution.

Of course, in all five scenarios the radius-margin quotient decreases, see Figure 10.1. However, in all cases where the size of the kernels is not kept constant, (A), (C), and (E), the number of support vectors drastically increases due to an inherent disadvantage of the optimization criterion: Having a training dataset consisting of ℓ elements and a normalized kernel, the radius is bounded by $R \leq \sqrt{1-1/\ell} \approx 1 - 1/(2\ell)$. In many applications this bound is almost reached, such that the derivative of R is comparatively small and the gradient of the quotient w.r.t. the kernel parameters is governed by the gradient of the margin ρ. Then, the margin can easily be enlarged by increasing $\det(B)$, that is, by concentrating the kernel mass to smaller areas in input space. This leads to solutions with smaller radius margin quotient but increasing number of support vectors. These complex solutions using nearly all points as support vectors are highly adapted to the training dataset and are not desirable because they tend to overfit, leading to worse test error in cases (A) and (C). This effect can be avoided by early stopping, by using a different optimization criterion, or by controlling the kernel size (e.g., by fixing the trace in the gradient steps to zero).

Comparing (C) with (E) and in particular (B) with (D) demonstrates in accordance to Friedrichs and Igel [2005] that better results can be achieved when the kernel adaptation is not restricted to diagonal matrices. The final test error in case (D) is significantly (20 trials, Wilcoxon rank-sum test, $p < 0.01$) lower than in all other cases.

Although the parameter space dimensions of the different kernel properties suggest that the control of the kernel size with only a single parameter should be easiest and most robust, the specific properties of the radius margin quotient lead to clearly sub-optimal solutions. This is in particular problematic because this result holds for case (A) of the standard radial Gaussian kernel. This investigation shows that the efficient minimization of the radius margin quotient can lead to undesirable solutions for this important type of kernel.

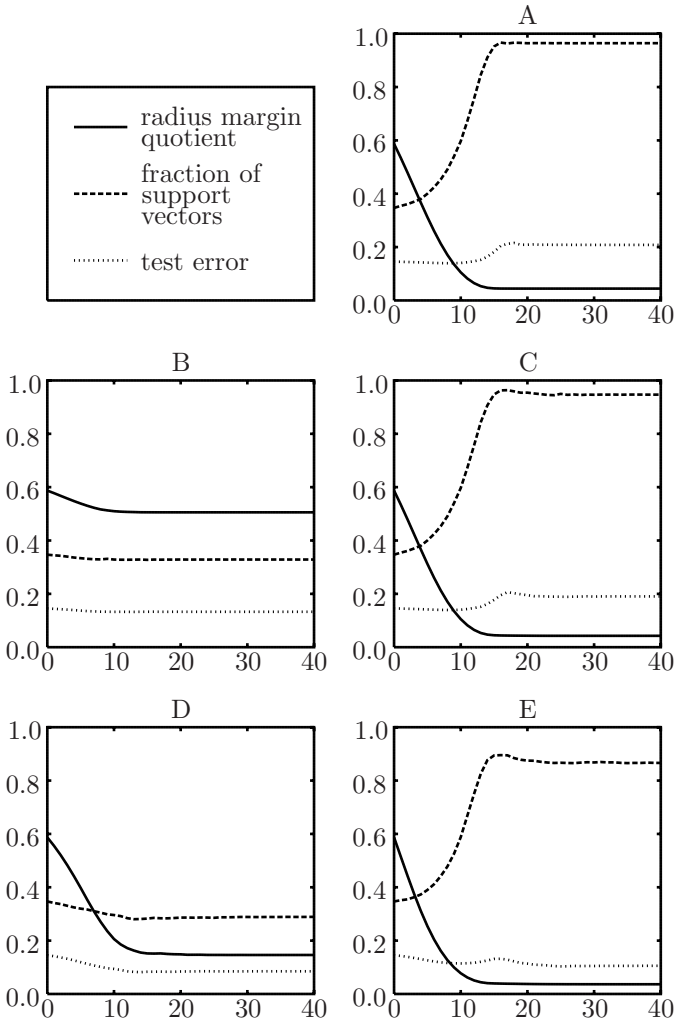

Figure 10.1: The diagrams show the (scaled) radius margin quotient $10^{-4} \cdot (R/\gamma)^2$, the fraction of support vectors, and the test error on a large separate dataset over the number of gradient descent steps. All quantities are averaged over 20 trials. From left to right: (A) multiples of the unit matrix, (B) diagonal with fixed trace, (C) diagonal, (D) fixed trace, (E) no constraints.

Chapter 11

Gradient Descent in the Presence of Noise

It is straight forward to use direct search methods for the minimization of hold-out error based objective functions like the cross-validation error or gradient-based algorithms for the optimization of the span-bound or the kernel target alignment. In contrast, maximization of the logarithmic likelihood $\hat{\mathcal{L}}$ and, if a prior distribution on the hyperparameters is available, the logarithmic posterior, can not be achieved with standard approaches. The reason is that the sampling technique used for the evaluation of these measures introduces uncertainty into the gradient information. In this chapter we will develop a new algorithm for gradient descent (or ascent) in the presence of the type of noise resulting from the sampling of dataset splits.

As already mentioned in the introduction, gradient descent and variants thereof are iterative strategies for unconstrained real-valued optimization. The objective function $g : \mathbb{R}^n \to \mathbb{R}$ is assumed to be differentiable. Then the aim of gradient descent is to iteratively approximate a (local) minimum of the objective (or error) function. In each iteration the algorithm computes the gradient of the objective function in the current search point and uses the negative gradient to construct a new search point with a (usually) lower value. In this context it is generally assumed that a (standard) inner product on the search vector space is given. This inner product is used to embed the gradient directly into the search vector space. This vector then coincides with the vector of partial derivatives w.r.t. the given orthonormal standard basis. As long as gradient information is available it is usually most efficient to approximate a local minimum with a gradient descent algorithm.

With the evaluation of the gradient of the randomized log-likelihood $\hat{\mathcal{L}}$ the situation is complicated by the fact that we have a possibly not very reliable estimate of the gradient of the "true" objective function $\bar{\mathcal{L}}$. Then the optimization algorithm must average over multiple evaluation in order to obtain more reliable information. We could try to resolve this situation by defining a new objective function which is just the mean over $N \gg 1$ evaluations of $\hat{\mathcal{L}}$ in the hope that N is large enough to make standard gradient descent algorithms work well. Such a proceeding has several disadvantages. We need to choose N a priori, without having any guidance for this choice. Thus we have an additional parameter to fix. Further, a good choice for the parameter N may depend on the search point. Therefore it is a much better idea to keep the averaging process out of the objective function and instead let the optimization algorithm handle this duty. This is the main motivation for the development of a gradient descent algorithm that can handle uncertain gradient information.

11.1 Problem Formulation

Let us make our assumptions explicit. Assume we are given a differentiable function on $\mathbb{R}^n$ and we want to find its minimum, or at least a local minimum. In a given search point $x \in \mathbb{R}^n$ we can evaluate the function value as well as its gradient, but these evaluations are subject to some kind of randomness. That is, the objective function as well as its gradient are given as the expected values of the distribution of the noisy measurements.

This situation is captured by a probability distribution for each $x \in \mathbb{R}^n$. Let μ_x be a distribution on $\mathbb{R} \times \mathbb{R}^n$ such that the random variable $(f(x), \nabla f(x))$ is distributed according to μ_x. That is, for measurable $V \subset \mathbb{R}$ and $D \subset \mathbb{R}^n$ the probability to have $f(x) \in V$ and $\nabla f(x) \in D$ is $\mu_x(V \times D)$. We make the following assumptions:

1. For each $x \in \mathbb{R}^n$ the expected value and the variance of $f(x)$ and $\nabla f(x)$ exist.
2. The function $g : \mathbb{R}^n \to \mathbb{R}$, $x \mapsto \mathbb{E}[f(x)]$, is continuously differentiable.
3. For the consistency of the above interpretation we need $\nabla g(x) = \mathbb{E}[\nabla f(x)]$ for all $x \in \mathbb{R}^n$.
4. For each $x \in \mathbb{R}^n$, μ_x has a continuous density $p_x : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ and the function $p : \mathbb{R}^n \times (\mathbb{R} \times \mathbb{R}^n) \to \mathbb{R}$, $(x, v, d) \mapsto p_x(v, d)$ is continuous.

These restrictions are rather weak. In particular, we do not assume a certain parametric noise model or a lower bound on the signal to noise ratio. The fourth condition seems unnecessarily strong, but has the advantage to allow for a clear argumentation and avoids technical difficulties. It can be considerably weakened if needed, but this would unnecessarily complicate the argumentation below.

The aim of this chapter is to develop an algorithm which, based on the random variable $\nabla f(x)$ only, approximates a local minimum of g. That is, if g has a unique minimum we want to solve the optimization problem

$$\underset{x}{\text{minimize}}\ g(x) \tag{11.1}$$

for $x \in \mathbb{R}^n$ based on the random gradient $\nabla f(x)$. Before we approach this goal we will introduce a simple gradient descent method that in some sense will serve as a design pattern for our new algorithm.

11.2 Resilient Backpropagation

The resilient backpropagation (Rprop) algorithm is a robust gradient descent method which aims at the efficient iterative approximation of a local minimum of a differentiable objective function $g : \mathbb{R}^n \to \mathbb{R}$ based on evaluations of the gradient $\nabla g(x)$ in the current search point x. The algorithm was developed by Riedmiller and Braun [1992] and later refined in several ways to further improve its speed and robustness (for example by Igel and Hüsken [2003]). The algorithm was originally designed to train feed forward artificial neural networks, but it is applicable to gradient-based optimization in general. The basic variant of the Rprop algorithm is stated as Algorithm 11.1.

The algorithm proceeds as follows: In each coordinate, it makes a step in the direction of the negative gradient. That is, if the partial derivative of the objective function with respect to a given coordinate is positive the algorithm makes a step to the left, while it steps to the right in case of a negative derivative. The step size is subject to a simple heuristic strategy adaptation. If two successive steps have the same direction, the step size is increased assuming that further steps in this direction follow and the optimization will reach the minimum faster with longer steps. If the current gradient indicates to

Algorithm 11.1: The resilient backpropagation algorithm

Input: initial $x \in \mathbb{R}^n$, initial step sizes $\delta_i > 0$ for $i \in \{1, \ldots, n\}$, strategy parameters $0 < \eta_- < 1 < \eta_+$

$P \leftarrow (0, \ldots, 0)^T$

do

  $G \leftarrow \nabla g(x)$

  check stopping condition

  forall $i \in \{1, \ldots, n\}$ **do**

    if $P_i \cdot G_i > 0$ **then**

      $\delta_i \leftarrow \eta_+ \cdot \delta_i$

    end

    else if $P_i \cdot G_i < 0$ **then**

      $\delta_i \leftarrow \eta_- \cdot \delta_i$

    end

    $x_i \leftarrow x_i - \text{sign}(G_i) \cdot \delta_i$

  end

  $P \leftarrow G$

loop

switch the direction then the previous step has jumped over a minimum and the step size is decreased. Of course, this heuristic leaves the interplay of the coordinates completely out of consideration. The idea is that this strategy adaptation makes the algorithm oscillate around the local minimum with decaying step size, resulting in convergence to this minimum. It is generally difficult to prove the convergence of optimization schemes involving strategy adaptation. This is because the current iteration does not only depend on the search point, but also on the optimization history. For a variant of the Rprop algorithm convergence to a local minimum has been proven [Anastasiadis et al., 2005].

The default values for the strategy adaptation parameters are $\eta_- = 0.5$ and $\eta_+ = 1.2$, halving the step size after a jump over a minimum and enlarging it by 20% otherwise. During optimization the algorithm adapts the step size to its needs, where the adaptation is by a factor growing exponentially with the number of iterations. Furthermore, the qualitative behavior of the algorithm is quite robust w.r.t. the actual choice of its strategy parameters. Therefore Rprop can be considered more or less a parameter free optimization strategy.

The main drawback of the Rprop algorithm is that it sticks to the given coordinate system. Thus it can suffer from badly conditioned Hessians of the objective function around a local optimum, resulting in slow oscillations. In this case the algorithm tends to first optimize along the principal axis of the Hessian with maximal eigenvalue and then successively continue with more difficult eigenspaces. This behavior is in principle only avoidable by algorithms which compute the Hessian of the objective function (if available) or estimate this Hessian from successive gradient evaluations (quasi Newton methods), see for example the article by Igel et al. [2005]. This proceeding is realized for example by the popular Broyden-Fletcher-Goldfarb-Shanno (BFGS) method and implicitly by the conjugate gradient method. However, if the actual Hessian varies too much with the search point the Hessian estimation can endanger the robustness of the method because the Newton step $\Delta x = -H^{-1} \nabla f(x)$ involves the inverse of the Hessian estimation H.

The robustness of the Rprop algorithm w.r.t. its strategy parameters and the objective function mainly results from the following properties:

- Only the sign of the partial derivatives is used. The absolute value is discarded, making the algorithm invariant against strictly monotone transformations of the

objective function.

- In contrast to the classical gradient descent technique the step size is adapted for each coordinate individually. This allows the algorithm to respond to differently scaled input coordinates, although it leaves dependencies between the coordinates unhonored. The step size is adapted quickly to the local properties of the objective function.

However, the strategy adaptation completely breaks down in the presence of randomness as introduced above. In this case the algorithm detects lots of spurious minima resulting in an undesired decay of the step size, leading to convergence to a sub-optimal point. In the following we will develop a completely new algorithm realizing the robust design principles of the Rprop algorithm, which at the same time works well if the gradients are random variables.

11.3 The NoisyRprop Algorithm

For the optimization of problem (11.1) robustness will play an even more important role than for the minimization of (deterministic) functions. This is because random effects can have a distorting influence on the selection of the search direction and the strategy adaptation.

For the motivation as well as for the analysis of the algorithm we will assume that g has a unique minimum. This consideration is sufficient to understand the behavior of the algorithm near a local minimum. Just like Rprop, the new algorithm will ignore the dependencies among the variables. That is, the algorithm will be designed as if all but one variable were fixed and we were interested in the minimum of the corresponding one-dimensional sub-problem. Among other questions we will investigate the effect of this design in Section 11.4.

11.3.1 Fixed Step Size Analysis

The first observation in the presence of noise is that the assumption of the Rprop algorithm that the previous step jumped over a minimum in case of a change of the sign of the derivative becomes invalid. Thus we can not use the same strategy adaptation as the original Rprop algorithm. Nevertheless we can still use the coordinate wise step sizes $\delta_i > 0$ and the rule to add or to subtract the step size in case of a negative or positive derivative, respectively.

It is important to understand the behavior of this algorithm in case of a fixed step size. W.l.o.g. let the minimum be located in the origin. Assume that the probability $q(x_i) = P((\nabla f(x))_i > 0)$ of a step to the left is monotonically increasing and takes the value $\frac{1}{2}$ only at $x_i = 0$. From the existence of the density p_x we know that the probability of a step to the right is $1 - q(x_i)$. Then the algorithm performs a random walk on a grid $\delta_i \cdot \mathbb{Z}$ translated by the initial position of x_i. This random walk has a drift towards the origin. From the continuity and the monotonicity of q we conclude the existence of $d > 0$ and $\varepsilon > 0$ such that $q(x) \leq \frac{1}{2} - \varepsilon$ for $x \leq -d$ and $q(x) \geq \frac{1}{2} + \varepsilon$ for $x \geq d$. Then the random walk does not escape to infinity with probability one and we get a stationary limit distribution on $x_i + \delta_i \cdot \mathbb{Z}$ which fulfills the linear equation

$$\begin{aligned} P(x = x_i + \delta_i k) = {} & q(x_i + \delta_i(k-1)) \cdot P(x = x_i + \delta_i(k-1)) \\ & + (1 - q(x_i + \delta_i(k+1))) \cdot P(x = x_i + \delta_i(k+1)) \end{aligned}$$

for $P(x = x_i + \delta_i k)$. From the bound on q it is clear that the probabilities decay at least exponentially outside the interval $[-d, d]$.

The most interesting observation is that this stationary distribution is more concentrated for smaller step sizes. This follows from a simple calculation. Assume we are at some $x_i \geq d + \delta_i$. Then the probability to go left is lower bounded by $\frac{1}{2} + \varepsilon$, while the probability to go right is upper bounded by $\frac{1}{2} - \varepsilon$. Let Q_1 denote the quotient of these probabilities which is lower bounded by $E = \frac{\frac{1}{2}+\varepsilon}{\frac{1}{2}-\varepsilon} > 1$, resulting in the decay $P(x = d + \delta_i k) \leq E^{-k} \cdot P(x = d)$. If we instead perform two steps of length $\delta_i/2$ there is a non-vanishing chance to make one step right and one step left (that is, in total, to stay). The chance to go left by δ_i is lower bounded by $(\frac{1}{2} + \varepsilon)^2$ while we go right with a probability of at most $(\frac{1}{2} - \varepsilon)^2$. The corresponding quotient is lower bounded by $E^2 > E$, such that the stationary probabilities decay at least like $P(x = d+\delta_i k) \leq E^{-2k} \cdot P(x = d)$. The argument carries over one-to-one to the left tail of the stationary distribution.

11.3.2 Adaptation of the Step Size

So it seems that all we need to do to well approximate the optimum is to choose a sufficiently small step size and let the algorithm run sufficiently long to well approximate the stationary distribution. Although this is indeed the main idea of the resulting algorithm we are not done yet. Of course, it is in most cases not appropriate or practically impossible to guess the correct step size for the desired solution accuracy. Such a proceeding would not only be impractical, but even cause rather long optimization times in some cases. It is a much better idea to let the algorithm choose the step size itself according to a strategy adaptation rule. This step size adaptation should have the following properties:

1. If the step size is much too small compared to the concentration of the stationary distribution, that is, we are far from the peak of the stationary distribution, then the optimization will clearly tend towards the minimum (that is, either to the left or to the right) with small steps. In this case we can speed up the optimization by increasing the step size.

2. If the optimization oscillates around the optimum and is mainly sampling the stationary distribution then we already achieved the best we can get with the current step size. In this case we should decrease the step size in order to sharpen the stationary distribution.

Of course, because of the unknown characteristics of the distributions μ_x it is impossible to judge the above cases from a single evaluation of $\nabla f(x)$. Instead, the algorithm will adapt its strategy only after an episode of N iterations. Let $x_i(k)$ for $k \in \{0, \ldots, N\}$ be the search points observed within the episode.

In the following we will define several statistics over these episodes and adapt the strategy parameters whenever the statistics take extreme values. Assume we want to control a certain "behavior" of the algorithm which is reflected by a statistic S over an episode. Let large values of S indicate desirable behavior, and let this behavior in general result from a large episode length. Let us define the null hypothesis of "perfect" behavior, which will be rejected in case S is found below its 10% quantile, given the null hypothesis (which will correspond to some limit distribution). Whenever we decide to reject the null hypothesis we increase the episode length. However, even for sufficiently long episodes resulting in desired behavior the length is increased in 10% of the cases. That is, the strategy adaptation drives the algorithm to arbitrary good behavior, but at the cost of uncontrolled growth of the episode length. Therefore we should decrease the episode length accordingly whenever the statistic is above, say, its 85% quantile, where it is important that $100\% - 85\% > 10\%$. By adjusting the quantiles we can control the distribution of the episode length and thus the quality of the desired behavior relative to its currently required accuracy and at the same time keep the episode length as low as

possible. Further assume that the statistic S becomes more reliable with growing episode length. As the algorithm will need to gradually increase the episode length in order to further improve the search point the episode length will nevertheless grow unlimited, but in this case because the algorithm iteratively requires higher and higher accuracy of its behavior in order to further concentrate the stationary distribution. Thus, the episode length grows only at the necessary rate to make progress, depending on the local properties of the objective function and the distribution of the gradient evaluations. We will apply this type of argument three times to different strategy parameters in the following.

Let us return to the step size control which is closely connected to the episode length. The first case of a too small step size is easy to detect. Let $\bar{q}$ be the mean probability of a step to the left over the episode, then the distribution of the accumulated steps $x_i(N) - x_i(0)$ is roughly binomial distributed with mean $\delta_i N(1-\bar{q})$ and variance $4\delta_i^2 N\bar{q}(1-\bar{q})$, which can be well approximated by a normal distribution (for N not too small). That is, under the hypothesis $\bar{q} = \frac{1}{2}$ the statistic

$$S_1 = \frac{x_i(N) - x_i(0)}{\delta_i\sqrt{N}} \quad .$$

is approximately standard normally distributed. We will discard the assumption $\bar{q} = \frac{1}{2}$ if $|S_1|$ is above its $1-\xi_1$ quantile. In this case the algorithm should increase the step size. However, even in case of $\bar{q} = \frac{1}{2}$ the algorithm will increase the step size in a fraction of ξ_1 of all cases. As this is an undesired effect we decrease the step size if $|S_1|$ falls below its ξ_1' quantile for some $\xi_1' > \xi_1$.

Assume we are close enough to the optimum for the approximation $q(x_i) = \frac{1}{2}$, then the algorithm performs a random walk which can be well approximated by a Wiener process for N large enough. We want to distinguish this situation from a process with an oscillation around the optimum. Obviously the spread of the points $x_i(k)$ should be significantly different in these cases. It seems natural to use the distance of the leftmost and the rightmost visited point as a statistic for this problem. We will use a slight variation, that is, we fix I equidistant iterations of the episode. Then each sum of N/I iterations is, in the Wiener process approximation, normally distributed with zero mean and variance $\delta_i^2 N/I$. We compute the statistic

$$S_2 = \frac{1}{\delta_i\sqrt{N}} \cdot \left(\max_{1\le k\le I} \left\{ x_i(\lfloor k\frac{I}{N} \rfloor) \right\} - \min_{1\le k\le I} \left\{ x_i(\lfloor k\frac{I}{N} \rfloor) \right\} \right) \quad .$$

If this quantity falls below the ξ_2 quantile of its distribution we will decrease the step size. On the other hand, if it is found above its $1-\xi_2'$ quantile for $\xi_2' > \xi_2$ we increase the step size again.

Whenever the step size is decreased, the episode length should be increased accordingly to guarantee that the stationary distribution is still equally well sampled. On the other hand, if the step size is increased the episodes can be shortened again.

11.3.3 Mean versus Median Optimization

Up to now the algorithm searches for x_i with $q(x_i) = \frac{1}{2}$. This is the point where the median of $(\nabla f(x))_i$ vanishes. However, for non-symmetric distributions μ_x the median will in general not coincide with the mean and the algorithm will tend to the wrong point. Therefore we need to control the skewness of the distribution, or to be more exact, the deviation of the median from the mean. But, assume we detect that the mean and the median of the distribution do not coincide with high probability - what can we do about it? The simple answer is that we evaluate the current search point multiple times without

moving x_i and average the gradient. The central limit theorem tells us that for large M the distribution of

$$\frac{\sum_{j=1}^{M}\Big((\nabla f(x)^{(j)})_i - \mathbb{E}[(\nabla f(x))_i]\Big)}{\sqrt{M}}$$

will converge to a normal distribution. Here, $\nabla f(x)^{(j)}$ is the j-th i.i.d. draw from μ_x. We make use of the observation that averaging over a number of samples does not only have the effect to reduce the variance, but it reduces the distance of the mean and the median of the distribution even more strongly. Then, in the limit $M \to \infty$ the median and the mean coincide.

It is clear that, in order to make the algorithm converge to a point with vanishing mean gradient $\mathbb{E}[(\nabla f(x))_i]$, the number M of averages needs to go to infinity in general. In terms of our algorithm this makes clear that M should be subject to strategy adaptation, too. Therefore we need a statistic to monitor in order to decide whether to increase or to decrease M. If M is large enough then the average derivative is roughly normally distributed. Let $G_i^{(k)} = \sum_{j=1}^{M}(\nabla f(x)^{(j,k)})_i$ be the averaged derivative in iteration k of the current episode. Under the hypothesis that the distribution of $G_i^{(k)}$ is symmetrical with zero mean the distribution of $|G_i^{(k)}|$ coincides for positive ($G_i^{(k)} > 0$) and for negative ($G_i^{(k)} < 0$) samples. We form the sums of the gradients pointing to the left and to the right

$$L = -\sum_{k=1}^{N} \min\{0, G_i^{(k)}\} \qquad R = \sum_{k=1}^{N} \max\{0, G_i^{(k)}\} \ ,$$

count the corresponding events

$$l = \big|\big\{k \in \{1,\dots,N\} \mid G_i^{(k)} < 0\big\}\big| \qquad r = \big|\big\{k \in \{1,\dots,N\} \mid G_i^{(k)} > 0\big\}\big|$$

and evaluate the statistic

$$S_3 = \sqrt{N} \cdot \frac{R/r - L/l}{(R+L)/N}$$

of the averaged derivatives which is approximately normally distributed under the Wiener process assumption. If $|S_3|$ exceeds its $1-\xi_3$ quantile we increase M and if it falls below its ξ_3' quantile we decrease M for some $\xi_3' > \xi_3$. An increase of M of course leads to an increase of the number of derivative evaluations per episode. To take a well founded decision at the end of an episode of a fixed step size requires a certain number of derivative evaluations, such that it is natural to keep $N \cdot M$ constant. That is, each time the number M of averages is increased, the episode length is decreased accordingly and vice versa.

11.3.4 The Algorithm

Now we turn the above heuristics into an algorithm, called NoisyRprop. It is given in Algorithm 11.2 in full detail. For this algorithm we fix the above defined probabilities more of less arbitrarily to $\xi_1 = 0.1$, $\xi_1' = 0.2$, $\xi_2 = 0.1$, $\xi_2' = 0.25$, $\xi_3 = 0.1$ and $\xi_3' = 0.15$. This leads to the approximate quantiles

value	quantile
$1-\xi_1$	$q^{S_1}_{1-\xi_1} \approx 1.64$
ξ'_1	$q^{S_1}_{\xi_1} \approx 0.25$
ξ_2	$q^{S_2}_{\xi_2} \approx 0.95$
$1-\xi'_2$	$q^{S_2}_{1-\xi_2} \approx 1.77$
$1-\xi_3$	$q^{S_3}_{1-\xi_3} \approx 2.5$
ξ'_3	$q^{S_3}_{\xi_3} = 0.29$

We set $I = 100$ and define a lower bound $N_{\min} = 40$ for N which will be used as an initial value for the episode length N. We start the algorithm without averaging, setting $M = 1$ to its lower bound.

For a fixed number I it is easy to numerically estimate the quantiles of the distribution of S_2. In a simple experiment we estimated the quantiles from $10,000,000$ i.i.d. samples of S_2 for $I = 100$ which should give extremely reliable results compared to the required accuracy.

To avoid rounding values like N and M we increase and decrease all quantities by the factor 2 only such that N and M will always remain whole numbers.

In each iteration the algorithm evaluates the random variable $\nabla f(x)$ once. That is, the computation of the average derivatives can take multiple iterations, depending on the coordinate index. This allows us to control the number of gradient evaluation in terms of iterations and guarantees that all partial derivatives are evaluated equally often. This is important in the typical case that a single function evaluation is computationally costly, but the derivatives can be computed with moderate overhead. On the other hand this has the effect that one variable may change while the average derivative is computed for another one. However, this is in accordance with the design principle that all coordinates are optimized independently of each other.

11.3.5 Averaging versus Random Walk

Whenever we halve the step size δ_i the stationary distribution gets more concentrated. But as we do not know how much more concentrated it gets we stay on the safe side by doubling the episode length.

Of course, instead of the episode length N_i we could increase the number M_i of averages, which stabilizes the gradient evaluations $G_i^{(k)}$. However, it is hoped that the first method has advantages over the second: First, we need to make as many steps as possible to sample the stationary distribution well, and the Wiener process approximation is more exact for longer episodes. Second, the distribution gets sampled much finer (although much less reliable) if we move the search point as often as possible. Last but not least we can move the search point much faster which is beneficial if we are far from the optimum.

11.4 Experimental Evaluation

Experiments were conducted for simple test cases in order to verify whether the various strategy adaptation rules work. To keep things simple all distributions used result from parametric families of functions. The algorithm was initialized with step sizes of 0.01 and a search point in a distance of about 1.0 to 1.1 from the optimum. This simulates the choice of a small step size compared to the quality of the initial guess of the optimum. For all problems the algorithm was run for $100,000,000$ iterations.

Algorithm 11.2: The NoisyRprop algorithm

Input: initial $x \in \mathbb{R}^n$, initial step sizes $\delta_i > 0$ for $i \in \{1, \ldots, n\}$, $N_{\min}$
$N \leftarrow (N_{\min}, \ldots, N_{\min})^T \in \mathbb{N}^n$; $M \leftarrow (1, \ldots, 1)^T \in \mathbb{N}^n$
$k \leftarrow (0, \ldots, 0)^T \in \mathbb{N}^n$; $j \leftarrow (0, \ldots, 0)^T \in \mathbb{N}^n$
$\tilde{x} \leftarrow x$; $x^{\min} \leftarrow x$; $x^{\max} \leftarrow x$
$l \leftarrow (0, \ldots, 0)^T \in \mathbb{N}^n$; $r \leftarrow (0, \ldots, 0)^T \in \mathbb{N}^n$
$L \leftarrow (0, \ldots, 0)^T \in \mathbb{R}^n$; $R \leftarrow (0, \ldots, 0)^T \in \mathbb{R}^n$
$G \leftarrow (0, \ldots, 0)^T \in \mathbb{R}^n$
do
  $G \leftarrow G + \nabla g(x)$
  forall $i \in \{1, \ldots, n\}$ **do**
    $j_i \leftarrow j_i + 1$
    if $j_i = M_i$ **then**
      `// standard Rprop step`
      if $G_i > 0$ **then**
        $x_i \leftarrow x_i - \delta_i$
        $L \leftarrow L + G_i$
        $l \leftarrow l + 1$
      end
      else if $G_i < 0$ **then**
        $x_i \leftarrow x_i + \delta_i$
        $R \leftarrow R - G_i$
        $r \leftarrow r + 1$
      end
      if $k = \lfloor tI/N \rfloor$ *for some* $t \in \{1, \ldots, 100\}$ **then**
        $x_i^{\min} = \min\{x_i^{\min}, x_i\}$
        $x_i^{\max} = \max\{x_i^{\max}, x_i\}$
      end
      if $k_i = N_i$ **then**
        `// strategy adaptation`
        $\tilde{N} \leftarrow N_i$
        $S_1 \leftarrow \big(x_i - \tilde{x}_i\big) \big/ \big(\delta_i \sqrt{N_i}\big)$
        if $|S_1| > q_{1-\xi_1}^{S_1}$ **then** $\tilde{N} \leftarrow \tilde{N}/2$
        else
          if $|S_1| < q_{\xi_1}^{S_1}$ **then** $\tilde{N} \leftarrow 2\tilde{N}$
          $S_2 \leftarrow \big(x_i^{\max} - x_i^{\min}\big) \big/ \big(\delta_i \sqrt{N}\big)$
          if $S_2 < q_{\xi_2}^{S_2}$ **then** $\delta_i \leftarrow \delta_i/2$; $\tilde{N} \leftarrow 2\tilde{N}$
          else if $S_2 > q_{1-\xi_2}^{S_2}$ **then** $\delta_i \leftarrow 2\delta_i$; $\tilde{N} \leftarrow \tilde{N}/2$
          $S_3 = \big(R_i/r_i - L_i/l_i\big) \big/ \big(\sqrt{N_i M_i}\,(R_i + L_i)\big)$
          if $|S_3| < q_{\xi_3}^{S_3} \wedge M_i \geq 2$ **then** $M_i \leftarrow M_i/2$; $\tilde{N} \leftarrow 2\tilde{N}$
          else if $|S_3| > q_{1-\xi_3}^{S_3}$ **then** $M_i \leftarrow 2M_i$; $\tilde{N} \leftarrow \tilde{N}/2$
          $N_i \leftarrow \max\{N_{\min}, \tilde{N}\}$
          $\tilde{x}_i \leftarrow x_i$; $x_i^{\min} \leftarrow x_i$; $x_i^{\max} \leftarrow x_i$
          $l_i \leftarrow 0$; $r_i \leftarrow 0$; $L_i \leftarrow 0$; $R_i \leftarrow 0$
        end
      end
      $G_i \leftarrow 0$; $j_i \leftarrow 0$; $k_i \leftarrow k_i + 1$
    end
  end
  check stopping condition
loop

As simple test cases we define the following four families of continuous and nearly everywhere differentiable functions

$$
\begin{aligned}
f_m^{(1)} &: \mathbb{R} \to \mathbb{R} & x &\mapsto \begin{cases} -|x| & \text{for } m \leq \frac{1}{4} \\ |x| & \text{for } m \geq \frac{1}{4} \end{cases} \\
f_{m,n}^{(2)} &: \mathbb{R}^2 \to \mathbb{R} & (x,y) &\mapsto (x-m)^2 + (y-n)^2 \\
f_{m,n}^{(3)} &: \mathbb{R}^2 \to \mathbb{R} & (x,y) &\mapsto (x-m)^2 + \frac{9}{5}(x-m)(y-n) + (y-n)^2 \\
f_m^{(4)} &: \mathbb{R} \to \mathbb{R} & x &\mapsto \begin{cases} 3(m-x) & \text{for } x \leq m \\ x-m & \text{for } x \geq m \end{cases}
\end{aligned}
$$

and compute their expected values $g^{(i)}(x) = \int_0^1 f_m(x)\,dm$ or $g^{(i)}(x) = \int_0^1 \int_0^1 f_m(x)\,dm\,dn$ w.r.t the uniform distributions over the parameter spaces $m \in [0,1]$ and $(m,n) \in [0,1] \times [0,1]$, respectively. The resulting convex objective functions are

$$
\begin{aligned}
g^{(1)}(x) &= \frac{1}{2}|x| \\
g^{(2)}(x) &= x^2 - x + y^2 - y + \frac{2}{3} \\
g^{(3)}(x) &= x^2 + y^2 + \frac{9}{5}xy - \frac{19}{10}x - \frac{19}{10}y + \frac{67}{60} \\
g^{(4)}(x) &= \begin{cases} -3x + \frac{3}{2} & \text{for } x \leq 0 \\ 2x^2 - 3x + \frac{3}{2} & \text{for } 0 \leq x \leq 1 \\ x - \frac{1}{2} & \text{for } 1 \leq x \end{cases} .
\end{aligned}
$$

These functions have the following properties: $f^{(1)}$ is an interesting case because the sign of the derivative is correct in 75% of the cases independent of the current search point. Therefore the optimization does not get harder when the algorithm approaches the optimum. After a phase of initial strategy adaptation we can expect a constant mean progress (measured on the logarithmic distance to the optimum) for fixed length intervals of iterations. The resulting behavior is shown in Figure 11.1. We can observe that, despite of randomness, the algorithm can exactly locate the optimum.

The test cases $f^{(2)}$ and $f^{(3)}$ involve two variables each. For $f^{(2)}$ the problem is separable, that is, the variables can be optimized independently, while for $f^{(3)}$ the optimal value for one variable strongly depends on the other. The Hessian of this function has eigenvalues 1.9 and 0.1 resulting in a conditioning number of 19, where the eigenspaces are spanned by the vectors $(1,1)^T$ and $(-1,1)^T$ which involves both coordinates. Thus this problem should be harder for NoisyRprop as the algorithm completely ignores this situation. The performance on these distributions is shown in Figure 11.2. Compared to the first experiment it is interesting that the curves oscillate much stronger. Here we can see the random walk on the contracting stationary distributions which are much wider than for $f^{(1)}$. The algorithm indeed suffers from the dependency of the variables.

Finally, $f^{(4)}$ is an example of an asymmetric gradient distribution, that is, the negative derivatives have three times larger absolute values than the positive derivatives. For this problem the median derivative vanishes in $x = 0.5$, while the mean derivative vanishes in $x = 0.75$ (the optimal point). Figure 11.3 shows that the optimization does not suffer from the asymmetry at all. Although in the first phase the search tends towards $x = 0.5$ it takes the algorithm only two or three episodes to adapt its strategy to sufficiently symmetrize the derivatives. Again, the plots show the typical oscillation of the solution quality with the roughly linear decay of the error in the log-log-plot which is typical for this type of objective function with arbitrary bad signal to noise ratio in the optimum.

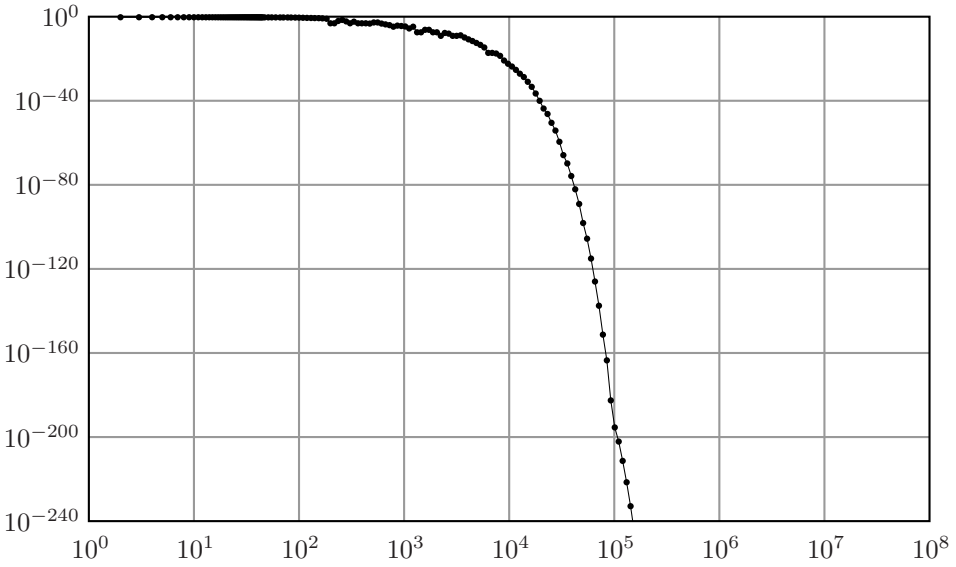

Figure 11.1: Distance to the optimum over the number of iterations for test case $f^{(1)}$. As soon as the strategy parameters are well adapted the optimization works very fast. After about 230,000 iterations the search point becomes exactly zero due to the limited resolution of double accuracy numbers which can not represent positive numbers below approximately 10^{-309}.

Optimization of noisy objectives is a common topic when applying evolution strategies. In contrast to the NoisyRprop approach, such direct search algorithms evaluate only the function value and do not make use of the gradient. A comparison of 0-th order search strategies with 1-st order gradient descent involves some difficulties not present in optimization problems without randomness. For example, if we add the term $10,000 \cdot \sin(20,000\pi m)$ to the objective function $f^{(1)}$, neither the derivative $\frac{\partial f^{(1)}(x)}{\partial x}$ nor the objective $g^{(1)}$ change. In contrast, the direct search algorithm would have a hard time averaging over lots of evaluations before a global trend became apparent, while the information evaluated by the NoisyRprop algorithm remains the same. This example shows that we would need to come up with a specific noise model when comparing NoisyRprop to direct search. Therefore such a comparison is not in the scope of this work.

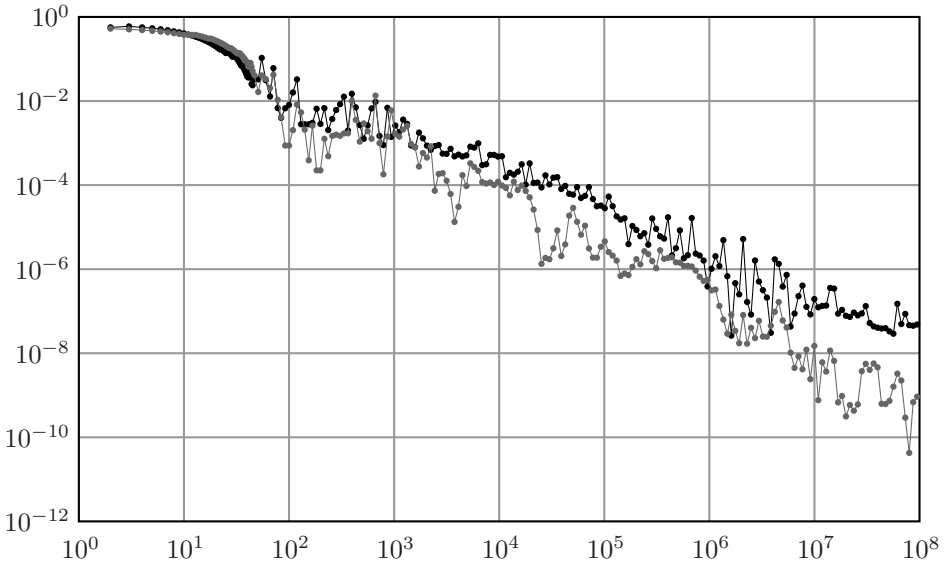

Figure 11.2: Distance to the optimum over the number of iterations for test cases $f^{(2)}$ (gray) and $f^{(3)}$ (black). The first segment of the curves corresponds to the first episode before any strategy adaptation is applied. As expected the optimization is slower for the dependent variables in problem $f^{(3)}$.

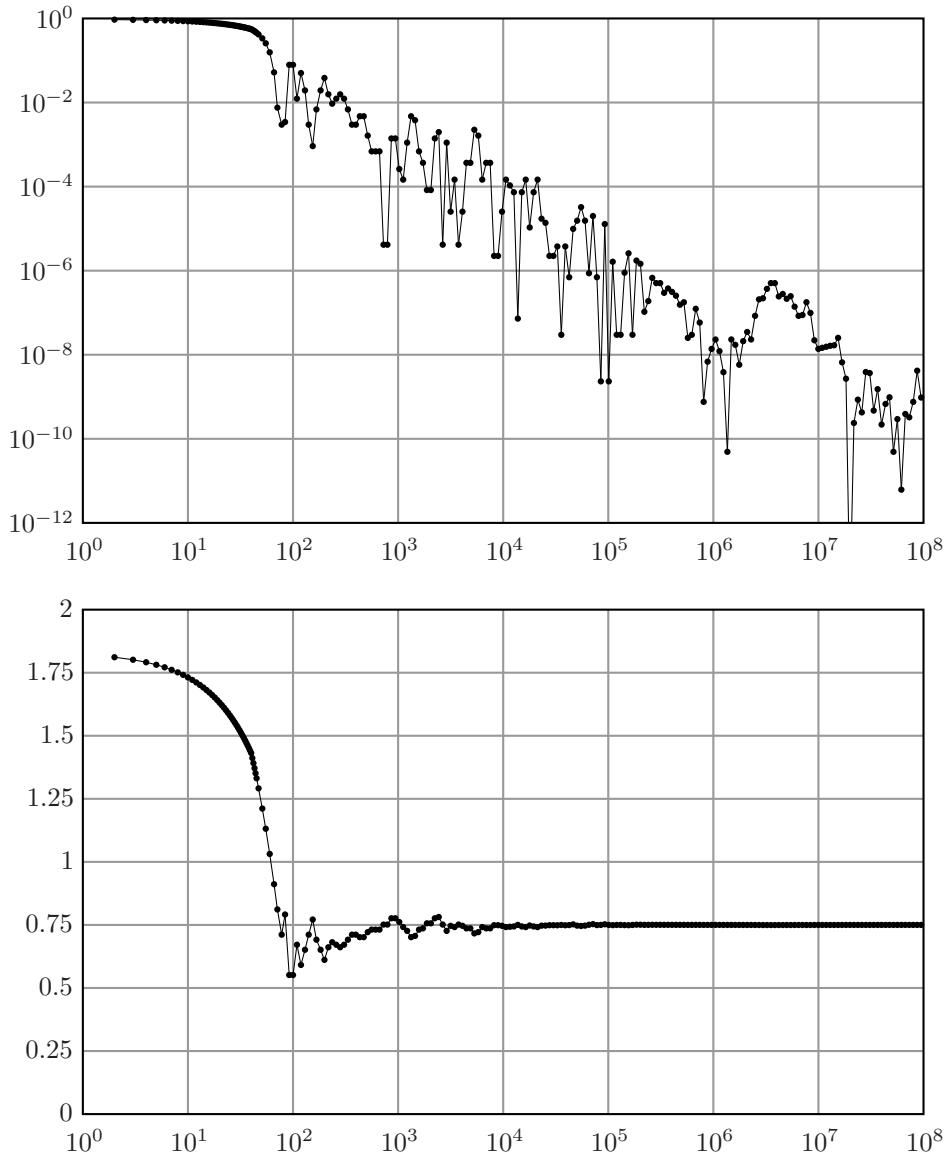

Figure 11.3: Distance to the optimum and search point over the number of iterations for test case $f^{(4)}$. We can observe how the algorithm first finds the point $x = 0.5$ where the median of the gradient vanishes, then gradually increases the number of averages and finally converges to the desired value of 0.75.

Chapter 12

Comparison of Model Selection Strategies

The best comparison of model selection strategies is to evaluate these methods on a suite of benchmark problems (see also the discussion in Chapter 3). In this chapter we empirically evaluate five strategies derived from the model selection objective functions introduced in Chapter 9. These strategies are compared on the artificial problems defined in Section 3.4 and on a collection of 10 datasets introduced in the study by Rätsch et al. [2001]. Most of these datasets stem from real-world problems. To obtain reliable results we restrict ourselves to 10 out of 13 datasets that consist of more than 500 examples. We apply 1-norm SVMs with Gaussian kernels and the goal of model selection is to find good values for the regularization parameter and the kernel parameters. We will evaluate these parameters with the error rate the final machine achieves on a test dataset.

12.1 Model Selection Strategies

The following five model selection strategies are suitable for the selection of the regularization parameter C of the 1-norm SVM and an arbitrary number of kernel parameters. They consist of an objective function *and* a suitable search strategy. Some of the iterative search strategies have parameters, most noticeable the starting point of the optimization, initial (average) step sizes, and the number of iterations used for the optimization. These values are set to reasonable default values, and in particular the number of iterations is large enough to allow the search strategies to come reasonable close to a local optimum, such that the study enables us to judge the quality (in terms of test error) of the optimum. In general longer runs of the algorithms will of course give slightly different results, but the number of iterations is chosen large enough such that it is reasonable to assume that these results will not be qualitatively different.

- `5-fold-CV`
  The cross-validation error (see Section 9.1.1) based on the 0-1-loss is not differentiable. As it is a standard objective function it is included in this comparison for completeness. Usually this objective function is used in conjunction with simple grid search, scaling exponentially in the number of variables. Instead we use the covariance matrix adaptation evolution strategy (CMA-ES) [Hansen and Ostermeier, 2001, Hansen et al., 2003, Friedrichs and Igel, 2005, Hansen, 2006] to descend the cross-validation error, which makes this error measure applicable in high-dimensional search spaces.

Each performance evaluation requires the training of 5 machines on 80% of the available data. Because the search strategy is very efficient for real-valued optimization, it is applicable to large datasets and high-dimensional parameter spaces. We used the standard settings for the CMA-ES and conducted $100\sqrt{n}$ iterations (each corresponding to the generation of one offspring population consisting of $\max\{5, \lfloor 3 \cdot \log(n) \rfloor + 4\}$ candidate solutions in the evolution loop), where n denotes the dimension of the search space.

- `KTA`
  This strategy first uses efficient gradient ascent (using the iRprop$^+$ algorithm, refer to Igel and Hüsken [2003]) on the kernel target alignment (see Section 9.1.6 or refer to the articles by Cristianini and Shawe-Taylor [2000] or Igel et al. [2007]) to obtain values for the kernel parameters. For this approach it is not necessary to train any support vector machine classifiers, resulting in a fast algorithm. Thus it makes sense to choose a fast method for the adaptation of the regularization parameter, too, which is determined by one-dimensional grid search on the 5-fold cross-validation error. Like for the `5-fold-CV` strategy we let the gradient descent algorithm perform $100\sqrt{n}$ iterations.

- `span-bound`
  The gradient-based iRprop$^+$ algorithm is used to descent the span bound objective function (see Section 9.1.5). As the computation of the span bound and its derivative is computationally demanding this strategy can only be applied to small datasets. Again we conduct $100\sqrt{n}$ gradient descent iterations.

- `likelihood`
  We use the NoisyRprop algorithm introduced in Chapter 11 to ascend the randomized logarithmic likelihood objective function $\hat{\mathcal{L}}$ (see Section 9.2.2). Each evaluation of the derivative requires the training of one support vector machine classifier and the inversion of a sub-matrix of the kernel Gram matrix. While this is still tractable for medium size problems the uncertainty in the function evaluation requires relatively many gradient ascent iterations, resulting in long runtimes. We let the NoisyRprop algorithm perform $10,000\sqrt{n}$ iterations. This computational complexity is comparable to $100\sqrt{n}$ evaluations of a 100-fold cross-validation error. Therefore, like for `span-bound`, the applicability of this strategy is limited to small problems.

- `posterior`
  We regularize the `likelihood` approach with a prior on the regularization parameter C and the Gaussian kernel (see Section 9.2.3). The algorithm requires the derivative of the logarithm of this prior.

  Without prior knowledge it is reasonable to choose the improper Jeffreys prior [Jeffreys, 1946] for C, that is, the (improper) uninformative prior for $\log(C)$. This prior does not favor any value of C over another. This is not always desirable as the derivative of the likelihood w.r.t. $\log(C)$ vanishes for hard margin solutions, such that there is a large plateau of the objective function in this area. We use a simple trick to escape this hard margin plateau for too large choices of C by setting the derivative of the prior w.r.t. $\log(C)$ to a small negative value. This effectively allows the sign-based NoisyRprop algorithm to quickly escape the plateau without affecting the optimization if the derivative of the likelihood w.r.t. C is non-zero.

  We choose a Gaussian prior for the parameter $\log(\gamma)$ of the Gaussian kernel. We compute the median m of the distances of examples of different class, a heuristic proposed by Jaakkola et al. [1999] for the parameter σ. This translates to γ as

$\gamma_m = 1/2m^2$. We arbitrarily fix the width of the prior to 2 such that the density of the prior for $\log(\gamma)$ becomes

$$p(\log(\gamma)) = \frac{1}{\sqrt{2\pi} \cdot 2} \exp\left(-\frac{(\log(\gamma) - \log(\gamma_m))^2}{2 \cdot 2^2} \right) \quad .$$

This reflects the heuristic assumption that a reasonable value for $\log(\gamma)$ should roughly be in the range of 2 around Jaakkola's heuristic. Translated to the width of the resulting Gaussian kernel this means that Jaakkola's guess should give a good kernel width, with a typical multiplicative error of about e^2. Of course, this choice is completely arbitrary and can be considerably wrong in some cases. In fact the width of the prior controls the amount of regularization of the model selection process. The best justification for the above prior is that is seems to work well for a large class of problems. These considerations directly carry over to the diagonal Gaussian kernel, where we apply a Gaussian prior to the parameter space on a logarithmic scale centered around the radial kernel with Jaakkola's heuristic. The prior can even be generalized to general Gaussian kernels with arbitrary covariance matrix. Refer to Chapter 10 for an appropriate parameterization of the parameter manifold.

As we are primary interested in the generalization performance of these strategies we use training datasets of $\ell = 200$ examples for the evaluation in order to keep the runtime tractable for all methods. All experiments were carried out with the Shark machine learning library [Igel et al., 2008].

12.2 Experiments on Artificial Problems

The first evaluation of the model selection strategies defined above is carried out on artificial data generating distributions defined in Section 3.4. We will refer to the sparse coordinate problems with parameters $(n, m, k) = (3, 14, 3)$ and $(n, m, k) = (3, 14, 5)$ as problem `sparse-3` and problem `sparse-5`, respectively. Gaussian kernels with free parameters on the diagonal of the covariance matrix, which can be used to freely scale the individual features (and therefore rank their relative importance) are a natural choice for these datasets.

Further we consider the chess board problem with its default parameters $(n, d) = (4, 2)$, referred to as problem `chess`, and a noisy variant for which the label was drawn randomly in 20% of the cases, referred to as problem `noisy-chess`. We use the radial Gaussian kernel in these cases.

As the distributions are completely known we are in the position to sample an arbitrary number of examples. For the evaluation we conducted 100 independent trials with each of the methods, where for each trial a training set of 200 examples and a large test set consisting of 100,000 examples were sampled. That is, in total every method was applied 100 times to generate reliable results, and the 100 resulting classifiers were judged on a total of 10 million test samples to estimate their generalization performance.

The results, including test errors and statistical tests, are summarized in Table 12.1.

12.3 Experiments on Benchmark Datasets

Although we get a first impression from the results on artificial data, we can argue that the sparse coordinate and chess board toy problems may not have much in common with typical application problems. To assess the performance of the model selection strategies

sparse-3

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.159	±0.0143	—	>>>	>>>	>>>	>>>
KTA	(2)	0.149	±0.0056	<<<	—	>>>	>>>	>>>
span-bound	(3)	0.157	±0.0603	<<<	<<<	—	>>>	>>>
likelihood	(4)	0.143	±0.0046	<<<	<<<	<<<	—	>>
posterior	(5)	0.141	±0.0017	<<<	<<<	<<<	<<	—

sparse-5

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.281	±0.0215	—	>>>	>>>	>>>	>>>
KTA	(2)	0.264	±0.0364	<<<	—	0.67	0.14	>>>
span-bound	(3)	0.275	±0.0608	<<<	0.33	—	0.16	>>>
likelihood	(4)	0.272	±0.0513	<<<	0.86	0.84	—	>>>
posterior	(5)	0.246	±0.0170	<<<	<<<	<<<	<<<	—

chess

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.107	±0.0212	—	<<<	<<<	>	>
KTA	(2)	0.125	±0.0158	>>>	—	<<<	>>>	>>>
span-bound	(3)	0.374	±0.0823	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.099	±0.0174	<	<<<	<<<	—	0.49
posterior	(5)	0.111	±0.0707	<	<<<	<<<	0.51	—

noisy-chess

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.240	±0.0419	—	<<<	<<<	>>	0.18
KTA	(2)	0.249	±0.0238	>>>	—	<<<	>>>	0.94
span-bound	(3)	0.430	±0.0369	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.230	±0.0217	<<	<<<	<<<	—	<<
posterior	(5)	0.290	±0.1093	0.82	0.06	<<<	>>	—

Table 12.1: Absolute and relative performance of the different model selection strategies on the artificial distributions. The errors are given as mean and standard deviation over the 100 independent trials. To increase legibility, the comparison matrix on the right uses the symbols <, <<, and <<< to indicate that the method in this row performs significantly better than the method in this column with significance levels 0.05, 0.01, and 0.001, respectively. A one-sided Mann-Whitney U-Test (also known as Wilcoxon rank sum test) is used for the comparison. Analogously, the symbols >, >>, and >>> indicate that the method in that row performs significantly worse with the corresponding significance level. If the differences are not significant to a level of at least 0.05 the significance level is reported.

	dataset	number of examples	input space dimension
1	banana	5300	2
2	diabetis	768	8
3	flare-solar	1066	9
4	german	1000	20
5	image	2310	19
6	ringnorm	7400	20
7	splice	3175	60
8	titanic	2201	3
9	twonorm	7400	20
10	waveform	5000	21

Table 12.2: Number of examples and input space dimension of the benchmark datasets.

on real-world data we conducted experiments on benchmark problems. The characteristics of the 10 benchmark datasets are summarized in Table 12.2.

As the number of examples available is limited we have to alter the proceeding used for the artificial problems. In particular, we can not sample large test sets, and not even 100 independent training sets from the underlying distribution. We could achieve this by sampling from the empirical distribution, but as discussed in Section 3.4 such a proceeding changes the problem and gives over-optimistic results, as the problem is greatly simplified. Instead we use 100 splits of the data into training and test sets. In each trial the available dataset is split into a random training set of size $\ell = 200$ and a test set consisting of the remaining examples. This comes at a two-fold price:

First, the test sets are of limited size and thus the variance of the test error can not be controlled at wish. Therefore the resulting significance levels are expected to be less accurate than for the artificial data with 100,000 test examples. Second and much worse, the trials are not independent. Therefore we can not apply standard significance tests to the results. We will nevertheless present test results, but note that these are obtained *as if the trials were independent*. Thus, the significance levels can not be trusted, but still the results allow for an estimate of the relative performance of the different model selection strategies.

We used both the radial and the diagonal feature scaling Gaussian kernel in the experiments. In the first case there are only two parameters to adapt, namely the regularization parameter C and the single kernel parameter γ. The number of parameters of the diagonal Gaussian kernel coincides with the input space dimension listed in Table 12.2. In most cases the dimension of the search space for the diagonal kernel is much larger than for the radial kernel.

The results with radial kernel are listed in Tables 12.3 and 12.4. Tables 12.5 and 12.6 present the corresponding results with diagonal kernel.

12.4 Discussion

By extensive simulations we collected a large number of results. We organize the discussion and the interpretation of these results as follows: First, we will have a closer look at the highly reliable results obtained on the artificial datasets. Then we will turn to the ten benchmark problems. Because of the large deviations we will in a first step discuss the experiments with the radial kernel and the diagonal kernel independently, and then compare the effects when using different kernels, which corresponds to different dimen-

banana

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.116	±0.0088	—	<<<	<<<	>	>
KTA	(2)	0.122	±0.0114	>>>	—	<<	>>>	>>>
span-bound	(3)	0.127	±0.0121	>>>	>>	—	>>>	>>>
likelihood	(4)	0.114	±0.0083	<	<<<	<<<	—	0.53
posterior	(5)	0.114	±0.0076	<	<<<	<<<	0.47	—

diabetis

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.252	±0.0263	—	<<<	<	0.90	>
KTA	(2)	0.272	±0.0202	>>>	—	>>>	>>>	>>>
span-bound	(3)	0.268	±0.0463	>	<<<	—	>>	>>>
likelihood	(4)	0.247	±0.0214	0.10	<<<	<<	—	0.74
posterior	(5)	0.244	±0.0173	<	<<<	<<<	0.26	—

flare-solar

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.352	±0.0215	—	<<<	0.47	>	>
KTA	(2)	0.369	±0.0430	>>>	—	>>>	>>>	>>>
span-bound	(3)	0.351	±0.0137	0.53	<<<	—	>	>
likelihood	(4)	0.347	±0.0179	<	<<<	<	—	0.49
posterior	(5)	0.349	±0.0238	<	<<<	<	0.51	—

german

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.277	±0.0216	—	>	<<<	>>>	>>>
KTA	(2)	0.271	±0.0180	<	—	<<<	>	>>>
span-bound	(3)	0.300	±0.0079	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.266	±0.0184	<<<	<	<<<	—	0.90
posterior	(5)	0.262	±0.0158	<<<	<<<	<<<	0.10	—

image

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.098	±0.0161	—	>	<	>	>>
KTA	(2)	0.094	±0.0160	<	—	<<<	0.28	0.75
span-bound	(3)	0.103	±0.0192	>	>>>	—	>>>	>>>
likelihood	(4)	0.094	±0.0128	<	0.72	<<<	—	0.92
posterior	(5)	0.091	±0.0125	<<	0.25	<<<	0.08	—

Table 12.3: Absolute and relative performance of the different model selection strategies on the benchmark problems 1 to 5 with radial Gaussian kernel. Refer to Table 12.1 for the meaning of the symbols, but note that due to the statistical dependency of the datasets used in the 100 trials the significance levels most probably need adjustment.

ringnorm

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.019	±0.0036	—	>>	<<<	<<	0.13
KTA	(2)	0.017	±0.0016	<<	—	<<<	<<<	<<
span-bound	(3)	0.090	±0.1573	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.044	±0.0838	>>	>>>	<<<	—	0.91
posterior	(5)	0.029	±0.0229	0.87	>>	<<<	0.09	—

splice

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.215	±0.1063	—	<	<<<	>>>	>>>
KTA	(2)	0.190	±0.0453	>	—	<<<	>>>	>>>
span-bound	(3)	0.480	±0.0494	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.172	±0.0328	<<<	<<<	<<<	—	0.22
posterior	(5)	0.170	±0.0116	<<<	<<<	<<<	0.78	—

titanic

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.226	±0.0081	—	0.25	0.82	0.58	0.39
KTA	(2)	0.227	±0.0093	0.75	—	0.91	0.79	0.66
span-bound	(3)	0.226	±0.0097	0.18	0.09	—	0.25	0.12
likelihood	(4)	0.226	±0.0086	0.42	0.21	0.75	—	0.31
posterior	(5)	0.227	±0.0088	0.61	0.35	0.88	0.69	—

twonorm

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.029	±0.0058	—	0.71	<<<	0.93	>>
KTA	(2)	0.028	±0.0030	0.29	—	<<<	0.84	>>
span-bound	(3)	0.416	±0.1790	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.032	±0.0473	0.07	0.16	<<<	—	0.93
posterior	(5)	0.027	±0.0034	<<	<<	<<<	0.07	—

waveform

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.119	±0.0243	—	0.88	<<<	>>	>>>
KTA	(2)	0.115	±0.0090	0.12	—	<<<	>	>>
span-bound	(3)	0.261	±0.0935	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.113	±0.0088	<<	<	<<<	—	0.84
posterior	(5)	0.111	±0.0080	<<<	<<	<<<	0.16	—

Table 12.4: Absolute and relative performance of the different model selection strategies on the benchmark problems 6 to 10 with radial Gaussian kernel. Refer to Table 12.1 for the meaning of the symbols, but note that due to the statistical dependency of the datasets used in the 100 trials the significance levels most probably need adjustment.

banana

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.118	±0.0099	—	<<<	<<<	>>	>
KTA	(2)	0.124	±0.0126	>>>	—	<	>>>	>>>
span-bound	(3)	0.127	±0.0122	>>>	>	—	>>>	>>>
likelihood	(4)	0.115	±0.0082	<<	<<<	<<<	—	0.31
posterior	(5)	0.115	±0.0084	<	<<<	<<<	0.69	—

diabetis

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.277	±0.0334	—	0.92	>	0.13	>>
KTA	(2)	0.267	±0.0199	0.08	—	0.90	<<<	>
span-bound	(3)	0.281	±0.0690	<	0.10	—	<<<	0.47
likelihood	(4)	0.277	±0.0221	0.87	>>>	>>>	—	>>>
posterior	(5)	0.263	±0.0207	<<	<	0.53	<<<	—

flare-solar

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.355	±0.0261	—	0.43	0.58	0.88	0.93
KTA	(2)	0.358	±0.0325	0.57	—	0.65	0.89	0.94
span-bound	(3)	0.352	±0.0173	0.42	0.35	—	0.83	0.92
likelihood	(4)	0.350	±0.0189	0.12	0.11	0.17	—	0.65
posterior	(5)	0.349	±0.0197	0.07	0.06	0.09	0.35	—

german

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.311	±0.0171	—	>>>	>>>	<<<	>>>
KTA	(2)	0.300	±0.0456	<<<	—	<	<<<	<
span-bound	(3)	0.300	±0.0079	<<<	>	—	<<<	0.31
likelihood	(4)	0.385	±0.1486	>>>	>>>	>>>	—	>>>
posterior	(5)	0.302	±0.0222	<<<	>	0.69	<<<	—

image

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.066	±0.0145	—	>>>	<<<	>>	<
KTA	(2)	0.055	±0.0380	<<<	—	<<<	<<<	<<<
span-bound	(3)	0.122	±0.1363	>>>	>>>	—	>>>	>>
likelihood	(4)	0.090	±0.1092	<<	>>>	<<<	—	<<<
posterior	(5)	0.070	±0.0143	>	>>>	<<	>>>	—

Table 12.5: Absolute and relative performance of the different model selection strategies on the benchmark problems 1 to 5 with diagonal Gaussian kernel. Refer to Table 12.1 for the meaning of the symbols, but note that due to the statistical dependency of the datasets used in the 100 trials the significance levels most probably need adjustment.

ringnorm

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.053	±0.0460	—	>>>	>>>	>	>
KTA	(2)	0.030	±0.0068	<<<	—	<<<	<<<	<<<
span-bound	(3)	0.115	±0.1727	<<<	>>>	—	<<	<<<
likelihood	(4)	0.087	±0.1318	<	>>>	>>	—	0.45
posterior	(5)	0.045	±0.0090	<	>>>	>>>	0.55	—

splice

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.478	±0.0217	—	>>>	<<<	0.60	>>>
KTA	(2)	0.181	±0.1213	<<<	—	<<<	<<<	0.94
span-bound	(3)	0.485	±0.0289	>>>	>>>	—	0.51	>>>
likelihood	(4)	0.363	±0.1929	0.40	>>>	0.49	—	>>>
posterior	(5)	0.137	±0.0196	<<<	0.06	<<<	<<<	—

titanic

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.226	±0.0107	—	0.06	0.21	0.19	0.32
KTA	(2)	0.232	±0.0468	0.94	—	0.82	0.76	0.90
span-bound	(3)	0.226	±0.0094	0.80	0.18	—	0.48	0.65
likelihood	(4)	0.226	±0.0106	0.81	0.24	0.52	—	0.67
posterior	(5)	0.226	±0.0083	0.68	0.10	0.36	0.34	—

twonorm

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.100	±0.1170	—	>>>	<<<	>>>	>>>
KTA	(2)	0.052	±0.0090	<<<	—	<<<	0.49	>>>
span-bound	(3)	0.437	±0.1571	>>>	>>>	—	>>>	>>>
likelihood	(4)	0.069	±0.0885	<<<	0.51	<<<	—	>>>
posterior	(5)	0.049	±0.0467	<<<	<<<	<<<	<<<	—

waveform

method		error		(1)	(2)	(3)	(4)	(5)
5-fold-CV	(1)	0.254	±0.0880	—	>>>	<<	>>>	>>>
KTA	(2)	0.132	±0.0136	<<<	—	<<<	<<<	<<<
span-bound	(3)	0.305	±0.0800	>>	>>>	—	>>>	>>>
likelihood	(4)	0.196	±0.1518	<<<	>>>	<<<	—	>>
posterior	(5)	0.143	±0.0142	<<<	>>>	<<<	<<	—

Table 12.6: Absolute and relative performance of the different model selection strategies on the benchmark problems 6 to 10 with diagonal Gaussian kernel. Refer to Table 12.1 for the meaning of the symbols, but note that due to the statistical dependency of the datasets used in the 100 trials the significance levels most probably need adjustment.

sions of the search spaces for the model selection problem. As the results considerably differ from dataset to dataset we will discuss special cases when appropriate. Finally we will try to extract general trends from the experimental results.

Before starting with the analysis of the results presented in the various tables it is appropriate to give some additional comments to the numbers in the tables, in particular to the means and standard deviations of the test errors. In some cases the standard deviations of the test errors over the 100 trials are astonishingly high. This is usually because the search algorithm completely fails in some trials, resulting in a test error of either constant classification or guessing performance. In the other trials the methods usually perform reasonably well. However, it is hard to judge whether such a behavior is preferable over a more robust but generally weaker strategy. Instead of the mean performance we generally use the significance levels of the Mann-Whitney U-test as a basis to compare the results.

12.4.1 Results on the Artificial Toy Problems

Let us have a look at the test results obtained on the four artificial datasets. Obviously, the `likelihood` strategy performs well. Due to its close relation to the `posterior` strategy we could expect similar performance of the `posterior` results, but of course the performance of this strategy additionally depends on the choice of the prior. For example, Jaakkola's heuristic for the choice of the Gaussian kernel obviously does not work well for the `noisy-chess` problem, where the median of the distances of examples of different class is too small due to the label noise. A higher quantile than the median should have been used to estimate a good kernel width instead, but this would have required the incorporation of prior knowledge which was assumed to be absent in this comparison. Besides the quality of Jaakkola's heuristic it is important to note that the regularizing effect of the hyperparameter prior, avoiding extreme parameter values and in particular boundary optima, is more important for the flexible diagonal Gaussian kernel used with the sparse coordinate problems than for the radial kernel used with the chess board problems. Therefore it is not surprising that the `posterior` method is more successful on the high-dimensional search spaces of the sparse coordinate problems.

As a side note it is interesting to observe that, despite the relatively small training dataset size of only $\ell = 200$ examples, the `posterior` strategy comes very close to the Bayes optimal error rate of the `sparse-3` problem of about 0.1404 (computed in Section 3.4).

We observe that the introduction of a non-trivial prior distribution on the space of kernels, in particular for high-dimensional search spaces, can improve the performance and the robustness of the model selection strategy, but a misleading prior may of course countervail this effect.

The `span-bound` strategy did not perform well. In particular for the chess board problems the gradient descent optimization gets regularly stuck in sub-optimal local minima. This is in concordance with the interpretation of the plots in Figure 9.1 where it is obvious how gradient descent can be mislead on this objective function.

Only the strategies `5-fold-CV` and KTA remain computationally feasible even for much larger dataset sizes. Especially the cross-validation strategy performs poorly in the high-dimensional search spaces of the sparse coordinate problems, but both methods give acceptable results when we take their low computational complexity into account. The results do not indicate which of these two strategy is more suitable. We would still prefer the kernel target alignment over cross validation, as gradient descent on this measure is extremely fast.

12.4.2 Benchmark Problems with Radial Kernel

Now we turn to the results on the benchmark problems. First we restrict our argumentation to nine datasets, as for the `titanic` dataset no method performed superior to any other.

The general trend in the results on the remaining benchmark problems learned with radial Gaussian kernel is quite similar to the results on the artificial benchmark problems. Again the `likelihood` strategy performs quite well, but it is slightly outperformed by the `posterior` method, which performs best. This result supports the impression that these objective functions are a good choice for model selection.

It is interesting to observe, especially on the first five datasets, that the span bound objective function does not lead to good hyperparameters even if it does not suffer from sub-optimal local optima far from the region of reasonable solutions. The low standard deviations for these datasets clearly indicate that the `span-bound` strategy systematically leads to sub-optimal solutions, as its mean performance is not distracted by outliers. On the other problems the mean performance is again drastically reduced by completely mislead trials achieving only guessing performance. This is most pronounced for the `splice` dataset.

Like for the artificial problems the fast strategies `cross-validation` and KTA perform reasonably well in all cases, with none of them being clearly preferable over the other. An exception is only the `ringnorm` problem where the simple strategies clearly outperform the three other methods. A closer look at the single trials reveals that all strategies achieve comparable performance in the majority of trials, but that the computationally intensive strategies `span-bound`, `likelihood`, and `posterior` suffer from outliers much more often. That is, on this problem all strategies are nearly equally good if we look at the medians, but the simple strategies turn out to be most robust in this case.

12.4.3 Benchmark Problems with Diagonal Kernel

We proceed with the last two tables, presenting the results of the experiments with the diagonal Gaussian kernel. Just like for the radial kernel we can not draw conclusions from the insignificant results obtained for the `titanic` dataset. The same holds for the `flare-solar` problem with diagonal kernel, where we can not derive any significant differences between the methods.

As above, we start with the `likelihood` strategy. In contrast to the other settings this strategy does not work well. In nearly all cases the `posterior` method achieves better results. This is easy to explain with the regularizing effect of the hyperparameter prior, which seems to be necessary in the high-dimensional search spaces corresponding to the diagonal Gaussian feature scaling kernel. In total it becomes clear that the likelihood method is not very robust, but together with the regularizing effect of the hyperparameter prior the `posterior` strategy is still among the best methods in this comparison.

Like in the other cases the `span-bound` strategy turns out to be clearly inferior to the other strategies in a majority of the cases. The `ringnorm` dataset is an exception. Although it achieves the worst test error in the mean the test results indicate that the `span-bound` method outperforms all but the `KTA` strategy on this dataset. This happens because in most trials the strategy achieves good results, but in about 20% of the cases it is completely mislead, resulting in a test error of about 50%.

The comparison of the fast strategies, `cross-validation` and `KTA`, is more interesting. The `cross-validation` method does not perform satisfactory with the diagonal Gaussian kernel. The reason for this behavior is that this relatively flexible kernel overfits to the fixed split of the dataset, detecting spurious features in the relatively small

hold-out sets. In contrast, the maximization of the kernel target alignment turns out to be the most robust strategy in the field, as it performs best in many cases and is significantly outperformed only by the posterior strategy on some datasets. The only exception is the low-dimensional `banana` dataset. The experiments indicate that especially in high-dimensional search spaces (in this case with 10 or more kernel parameters) the `KTA` strategy is most robust and performs clearly best.

12.4.4 Radial vs. Diagonal Kernel

By comparing the experiments with the radial Gaussian kernel with the results obtained for the diagonal kernel we can learn something about the robustness of the different model selection strategies. Of course, it is much simpler to achieve reliable results on low-dimensional search spaces than on high-dimensional ones, as in high-dimensional spaces there are intuitively more directions to escape to undesired boundary optima.[1] The `cross-validation` strategy clearly suffers from this effect, but it is most pronounced for the `likelihood` strategy. In contrast, the `posterior` strategy suffers much less from the high search space dimension. This is of course expected, as the regularizing effect of the prior was designed exactly this way. For the `span-bound` strategy the family of kernels used makes nearly no difference as any effect is overshadowed by the large number of test error outliers. The `KTA` method is clearly best suited for model selection in high-dimensional search spaces.

Another aspect of this comparison is that we would generally expect that the classifiers achieve better results with the more flexible diagonal Gaussian kernel, because this kernel allows to scale the relative importance of each coordinate individually and can thus rely on the highly discriminative features. It turns out that in the collection of benchmark problems this effect does not always play a role. This is because in some datasets it makes sense to use an equal weighting of the coordinates. Then the additional flexibility of the diagonal kernel can be a drawback, as it endangers the robustness of the model selection process, and the same method usually results in a higher test error.

12.4.5 Final Evaluation

Some of the results obtained are quite clear, while others remain ambiguous. Although there are good explanations for most of the effects observed, the reasons remain unclear in some special cases. Therefore it is not easy to come up with a clear conclusion. Nevertheless we would conclude the following recommendations from this study: Whenever it is computationally feasible, in particular for small datasets when the model selection problem is most difficult from a conceptual point of view, the `posterior` strategy should be given a try. Of course this requires that one can construct a meaningful prior on the kernel parameters. For larger datasets, when this method becomes intractable, the `KTA` strategy is a good alternative which is directly applicable even to datasets of tens of thousands of training examples. We should also keep in mind that the results remain ambiguous. Therefore it may be a good idea to try multiple strategies in order to obtain a robust estimate of well generalizing hyperparameters.

However, due to the finite character of this study the results have to be viewed with care. There is little justification to generalize the results to other datasets, although it is hoped that the benchmark problems are more or less representative. The generalization of the results to completely other types of kernels seems to be even more dangerous, but

[1] The scaling of the number of local optima with the search space dimension is a non-trivial issue. In this case it is plausible that boundary optima exist in some of the parameter coordinates independent of each other, such that the number of such undesired boundary optima increases with the search space dimension.

this is in principle not avoidable in an empirical study. At this point it is most evident that general theoretical results about the performance of model selection strategies are needed, but because the available generalization bounds turn out to be still too loose it is not clear how to approach this challenging problem.

Chapter 13

Application to Classification of Prokaryotic TIS

This final chapter describes the application of model selection methods for SVMs to a biologically relevant signal detection problem, namely the extraction of protein-encoding sequences from nucleotide sequences. This is an elementary task in bioinformatics. To extract the sequences, it is necessary to detect locations at which coding regions start. These locations are called *translation initiation sites* (TISs).

The exact localization of gene starts[1] in DNA sequences is crucial for the correct annotation of bacterial genomes. This task is difficult to achieve with conventional gene finders, which are usually restricted to the identification of long coding regions. Support vector machines have proven helpful for the automated solution of this problem [Zien et al., 2000, Meinicke et al., 2004, Li and Jiang, 2005, Mersch et al., 2006, 2007, Igel et al., 2007].

13.1 Problem Description

In the analysis of bacterial gene data it is an important sub-task to identify the positions in the DNA (or RNA) nucleotide sequence where the protein encoding triplet code starts. These positions are called translation initiation sites (TISs).

We consider (sense strand) DNA sequences represented as strings over the alphabet $\mathcal{A} = \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}$. A TIS contains the start codon (triplet, substring of length three) ATG or rarely GTG or TTG (in the data used here there is only one known case where also ATT serves as a start codon). The start codon marks the position at which the translation starts.

Not every ATG triplet is a start codon. Therefore it must be decided whether a particular ATG corresponds to a start codon or not. It is biologically plausible that the neighborhood of nucleotides around potential TISs, probably combined with additional features, should serve as the basis of decision making.

The problem of detecting translation initiation sites is naturally cast into a binary classification task by considering fixed size parts of DNA sequences containing a potential start codon as input. The task is to predict whether a codon represents a gene start or not.

When discussing TIS detection, we have to distinguish between eukaryotes and prokaryotes, that is, between organisms in which the genetic material is organized into membrane-bound nuclei and organisms without a cell nucleus. In contrast to prediction of eukaryotic

[1] In this context we use the terms *gene start* and *translation initiation site* as synonyms.

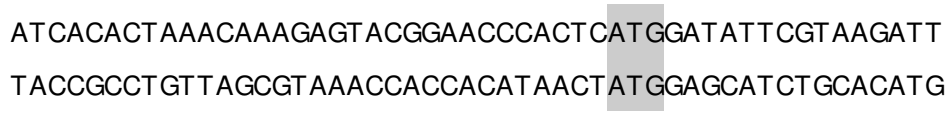

Figure 13.1: Examples of sub-sequences extracted from *E. coli* genes. The sequences of length 50 are aligned such that the potential start codon is located at the same position in all examples. The first sequence has a positive label, while the second sequence is a negative example. That is, in the first example the codon ATG encodes a gene start, while in the second example this codon is formed by chance or to encode an amino acid. The bigger part of the sub-sequence is located upstream (left) of the potential start codon. The assumption that these nucleotides have the largest influence on whether a coding region starts or not is biologically plausible.

TISs there is no biological justification for using a general learning machine across different species for prediction of prokaryotic TISs. For this reason, learning of prokaryotic TISs is always restricted to a limited amount of species-specific examples and model selection methods have to cope with small datasets.

Our model selection experiments were carried out on the same data considered by Mersch et al. [2006, 2007], and Igel et al. [2007]. This dataset was obtained from the *Escherichia coli* genes from the EcoGene database [Rudd, 2000]. Only those entries with biochemically verified amino-terminus were considered (that is, there is biochemical proof that proteins are built starting from the TIS). The neighboring nucleotides were looked up in the GenBank file `U00096.gbk` [Blattner et al., 1997]. From the 732 positive examples (as noted above, we have to deal with small datasets compared to the analysis of eukaryotic sequence databases) we created associated negative examples. For the negative examples we extracted sequences centered around a codon from the set $\{\mathsf{ATG}, \mathsf{GTG}, \mathsf{TTG}\}$ and accepted them if the codon is in-frame (translated by a multiple of three) with one of the start sites used as a positive case, its distance from a real TIS is less than 80 nucleotides, and no in-frame stop codon occurs in between. This corresponds to a (human or otherwise) preprocessed dataset, where the learning machine is only asked in difficult to decide cases. Thus, this data selection generates a challenging benchmark.

We finally obtained a set of 1248 negative examples, which is in the same order of magnitude as the number of positive examples (the selection of difficult cases avoids a flood of trivial negative examples). The length of the sub-sequences used as inputs is arbitrarily chosen to be 50 nucleotides, with 32 located upstream and 18 downstream, including the start codon (see Figure 13.1 for an illustration). This leads to a canonical alignment of the sub-sequences with the potential start codon at the same position in all sequence examples.

13.2 Oligo Kernels

Several types of kernels have been proposed for sequence data, usually operating on text or DNA sequences (refer to the textbooks by Schölkopf and Smola, 2002, Schölkopf et al., 2004, and Shawe-Taylor and Cristianini, 2004). Here we will apply oligo kernels introduced by Meinicke et al. [2004] and variants thereof. The term oligo(mer) refers to short, single stranded DNA fragments. Oligo kernels allow to control the amount of position dependency of single nucleotides in the similarity computation carried out by the kernel, and are therefore well suited for the problem at hand. For example, measuring the similarity of two sequences by the standard Hamming distance is fully position-dependent

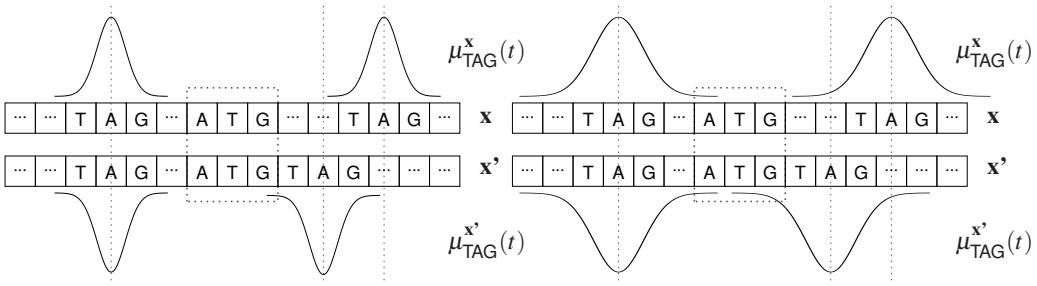

Figure 13.2: Example of two sequences x and x' and the corresponding oligo functions for $\omega = \mathsf{TAG}$ for small (left) and large (right) smoothing parameter σ_3.

(either two symbols at a given position are identical or not), whereas comparing just the frequencies of the symbols is completely position-independent (the position of a symbol within a sequence does not matter, just how often it occurs). The gradual control is a decisive feature compared to other string kernels for biological sequences, which usually provide either position-dependent [Degroeve et al., 2002] or completely position-independent representations [Leslie et al., 2002, Markowetz et al., 2003]. The locality improved kernel [Zien et al., 2000] is among the first approaches allowing a gradual control of the position-dependency. Measuring the similarity between sequences using kernels based on the edit distance between the sequences is an alternative proceeding in which the position-dependency can be controlled [Li and Jiang, 2005].

From a machine learning perspective the different variants of oligo kernels pose interesting problems, with the number of parameters varying in a wide range.

Oligo kernels are constructed with an explicit feature map Φ, mapping the input space $X \subset \bigcup_{n\in\mathbb{N}} \mathcal{A}^n$ into the Hilbert space $\mathcal{H} = L^2(\mathbb{R})$ of square integrable functions w.r.t. the standard Lebesgue measure on the reals. The image $\Phi(X)$ consists of so-called oligo functions, posing finite sums of Gaussians. Then the inner product in the Hilbert space $L^2(\mathbb{R})$ defines the oligo kernel function.

In our context, subsequences $\omega \in \mathcal{A}^K$ of length K are called *K-mers* or oligomers of length K. For a sequence $x \in X$ containing the K-mer $\omega \in \mathcal{A}^K$ at positions $S^x_\omega = \{p_1, p_2, \dots\}$, the oligo function is given by

$$\mu^x_\omega : \mathbb{R} \to \mathbb{R}, \qquad t \mapsto \sum_{p\in S^x_\omega} \exp\left(-\frac{(t-p)^2}{2\sigma_K^2}\right) \ ,$$

see Figure 13.2 for an example. The smoothing parameter σ_K adjusts the width of the Gaussians centered on the observed oligomer positions and determines the degree of position-dependency of the feature space representation.

For a sequence $x \in X$ the occurrences of all K-mers contained in $\mathcal{A}^K = \{\omega_1, \dots, \omega_m\}$ can be represented by an oligo function $\mathbb{R} \to \mathbb{R}^m$ with $m = |\mathcal{A}|^K$. This yields the final feature space representation $\Phi^K(x) = (\mu^x_{\omega_1}, \dots, \mu^x_{\omega_m})^T \in \left(L^2(\mathbb{R})\right)^m$ of that sequence. The feature space representation of a sequence thus becomes a vector-valued function with a smoothed representation of the occurrences of a certain K-mer in each component. For

two sequences x and x' we obtain the kernel function

$$\begin{aligned}\left\langle \Phi^K(x), \Phi^K(x') \right\rangle &= \int \Phi^K(x)(t) \cdot \Phi^K(x')(t) dt \\ &= \sum_{\omega \in \mathcal{A}^K} \sum_{p \in S_\omega^x} \sum_{q \in S_\omega^{x'}} \int \exp\left(-\frac{(t-p)^2}{2\sigma_K^2}\right) \exp\left(-\frac{(t-q)^2}{2\sigma_K^2}\right) dt \\ &\propto \sum_{\omega \in \mathcal{A}^K} \sum_{p \in S_\omega^x} \sum_{q \in S_\omega^{x'}} \exp\left(-\frac{1}{4\sigma_K^2}(p-q)^2\right) \\ &=: k_K(x, x')\end{aligned}$$

which is proportional to the inner product in $\left(L^2(\mathbb{R})\right)^m$. The computation of this kernel takes $\mathcal{O}(n \cdot n')$ operations where n and n' denote the lengths of the sequences x and x'. Thus, the computation of a kernel value is rather expensive.

The feature space representations of two sequences may have different norms. In order to improve comparability between sequences of different lengths, we can compute the normalized oligo kernel

$$\bar{k}_K(x, x') = \frac{k_K(x, x')}{\sqrt{k_K(x, x) k_K(x', x')}} \ .$$

From a biological point of view it is natural to consider the case $K = 3$, as each amino acid is encoded by a triplet of nucleotides. However, several studies [Meinicke et al., 2004, Mersch et al., 2006, 2007, Igel et al., 2007] have shown that it is beneficial to use more than one value of K. This consideration leads to the combined oligo kernel

$$k_{\text{combined}}(x, x') = \sum_{K=1}^{6} \bar{k}_K(x, x') \ ,$$

where the position dependency of each oligomer length can be adjusted with its own parameter σ_K. It is natural to extend this definition by the introduction of additional non-negative weights λ_K to the weighted kernel

$$k_{\text{weighted}}(x, x') = \sum_{K=1}^{6} \lambda_K \bar{k}_K(x, x') \ .$$

Finally we want to encode the biological knowledge of the importance of the triplet code. Some triplets may be more discriminative than others for the task of TIS classification. To represent this relative importance we modify the 3-mer kernel by introducing non-negative weights for each of the 64 different 3-mers $\omega \in \mathcal{A}^3$ by writing

$$k(x, x') = \sum_{\omega \in \mathcal{A}^3} \lambda_\omega \sum_{p \in S_\omega^x} \sum_{q \in S_\omega^{x'}} \exp\left(-\frac{1}{4\sigma_3^2}(p-q)^2\right) \ .$$

We refer to the normalization of this kernel as the triplet weighting oligo kernel.

Many other parameterizations would be plausible. For example, we may introduce additional weights for the 50 positions in the aligned sub-sequences. For simplicity we will restrict ourselves to two standard cases in the following.

13.3 Model Selection for TIS classification

In the following we will build 1-norm SVMs either with the simple 3-mer kernel or with the rather flexible triplet weighting oligo kernel. Analogously to the previous chapter

the main focus of the empirical evaluation is on the comparison of the model selection strategies, once in the presence of only two parameters C and σ_3, and once for a 66-dimensional search space when the 64 weights λ_ω are included.

We apply the model selection schemes `5-fold-CV`, `KTA`, `span bound`, and `likelihood` as defined in the previous chapter to the problem of TIS classification with 1-norm SVMs. In contrast to the previous chapter we do not consider the `posterior` strategy because it is not clear how to come up with a meaningful prior on the space of oligo kernels if we assume that no prior knowledge about the problem at hand or typical parameter ranges for the kernel parameters is available.

To reduce random effects, we generate 10 different partitionings of the data into training and test sets. Each training set contains 400 sequences plus the associated negatives, the corresponding test set 332 sequences plus the associated negatives.

13.4 Experimental Results

3-mer kernel

method		error		(1)	(2)	(3)	(4)
`5-fold-CV`	(1)	0.073	±0.0069	—	0.93	<<	0.58
`KTA`	(2)	0.068	±0.0045	0.07	—	<<<	0.13
`span bound`	(3)	0.109	±0.0856	>>	>>>	—	>>
`likelihood`	(4)	0.072	±0.0064	0.42	0.87	<<	—

triplet weighting kernel

method		error		(1)	(2)	(3)	(4)
`5-fold-CV`	(1)	0.074	±0.0068	—	<<<	0.09	0.93
`KTA`	(2)	0.106	±0.0575	>>>	—	>	>>>
`span-bound`	(3)	0.107	±0.0865	0.91	<	—	>>
`likelihood`	(4)	0.068	±0.0091	0.07	<<<	<<	—

Table 13.1: Absolute and relative performance of the four model selection strategies on the TIS detection problem with the simple 3-mer kernel and the highly flexible triplet weighting kernel. Refer to Table 12.1 for the meaning of the symbols, but note that due to the statistical dependency of the datasets used in the different trials the significance levels most probably need adjustment.

The results of the experiments are summarized in Table 13.1. Interestingly, the outcomes considerably differ from the results obtained in the comparison study presented in the previous chapter.

For the 3-mer kernel all strategies but `span-bound` perform nearly equally well, with the `KTA` slightly in the lead. Although until now the `KTA` strategy has proven most robust in high-dimensional search spaces, it does not achieve satisfactory results for the triplet weighting kernel. This is because the maximum of the kernel target alignment is located at the boundary of the parameter space, such that this strategy sets about 3/4 of the weights exactly to zero. The test results indicate that this is not a good strategy, as the remaining triplets do not contribute sufficient discriminative information to achieve competable performance.

Like in the study presented in the previous chapter the mean performance of the `span-bound` strategy suffers from outliers, that is, the strategy completely fails in some cases. If we consider the median performance instead this method performs much better,

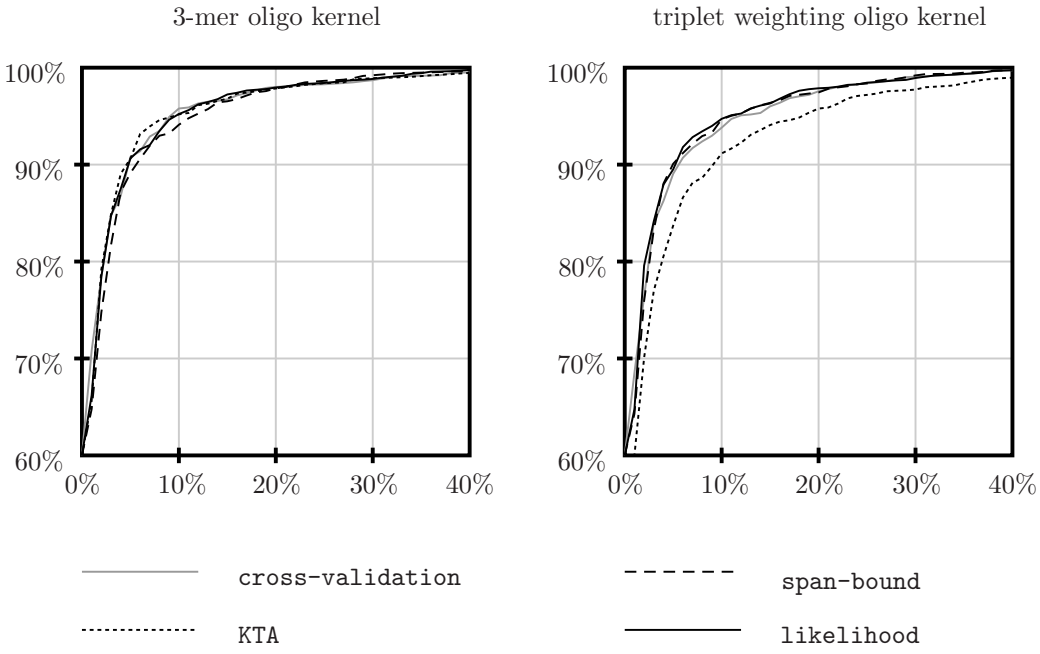

Figure 13.3: Receiver operating characteristic (ROC) curves of the classifiers obtained with the different model selection strategies. The plots show the dependency of the sensitivity (true positive rate) over one minus the specificity (false positive rate) of the classifiers. The different error rates were obtained by varying the threshold b in the SVM hypothesis $x \mapsto \langle w, x \rangle + b$, which is a common technique to obtain ROC curves for SVMs. The medians over 10 trials are plotted.

but still consistently worse than the `likelihood` method. Finally, the results indicate that the `likelihood` method outperforms the `cross-validation` strategy only slightly. The reason is probably that, for the problem of TIS classification, the `cross-validation` strategy does not overfit to its fixed dataset split too much. Still, the `likelihood` strategy performs better, while the `cross-validation` strategy is much faster.

The ROC curves shown in Figure 13.3 give an even more detailed impression of the performance of the classifiers constructed by the different strategies. Note that the median performance is used for these plots, such that the outliers in the `span-bound` trials are not visible. It becomes clear that the `KTA` strategy is not suitable for the triplet weighting kernel. The differences in the median performance of the other strategies are much smaller.

In accordance with the earlier studies by Mersch et al. [2006, 2007] and Igel et al. [2007] the kernel width parameters are chosen quite small. For example, the `likelihood` strategy uses $\sigma_3 = 1.27$ for the 3-mer kernel and $\sigma_3 = 1.15$ for the triplet weighting kernel. Thus, the match of the triplets is done with relatively high positional accuracy.

The triplet weighting kernel was introduced with the belief that the classification accuracy can be improved due to the different relative importance of the triplets. Although the results did not improve significantly, it is still interesting to ask which triplets were considered most important. For this analysis we use the results of the most successful strategy, which is the `likelihood` method. The codons with the largest weights are listed in Table 13.2.

triplet	GGC	ATG	TCG	GAG	TTG	GGA
weight	5.64	4.81	3.34	2.66	2.61	2.44
triplet	AGG	GGT	GGG	GCT	CGC	GAT
weight	2.28	2.04	2.02	1.96	1.89	1.89

Table 13.2: The 12 most important triplets selected by the `likelihood` method, and the (geometric) means of their weights.

Two of the three potential start codons are given considerably high weights. Furthermore, most of the codons appearing in the Shine-Dalgarno sequence TAAGGAGGT [Shine and Dalgarno, 1974], which is known to be important for the detection of TISs in prokaryotes, are judged to be highly discriminative. These results are in accordance with biological knowledge, such that there is hope that the results of similar experiments can be transferred back to obtain insights about the mechanisms underlying the biological process of TIS detection.

Summary and Final Discussion of the Results

In this section we summarize our results and discuss them from a broader perspective. We embed the methods presented in this thesis into a wider picture, showing possible directions for future extensions.

Short Summary

Gradient-based optimization techniques are applied to two independent problems in the area of support vector machine (SVM) learning. The first problem, machine training, requires efficient high-dimensional quadratic programming. The second problem, model selection, amounts to the construction of an objective function and non-convex optimization. The main contributions to SVM training are in the area of algorithm design and corresponding convergence proofs. The model selection problem is assessed with completely new approaches. We highlight these achievements with a number of basic theoretical results and empirical evaluations.

Improvements of the SMO Algorithm

State of the art algorithms for support vector machine training, in particular sequential minimal optimization (SMO) type algorithms, are already highly developed and extremely efficient. Our linear progress rate result for free SMO steps reveals an interesting property of such algorithms under moderate assumptions only on the working set selection strategy. In contrast to the earlier result by Chen et al. [2006] the proof relies on geometrically motivated analytical and topological arguments.

We present improvements of the basic SMO algorithm, demonstrate their performance, and prove the convergence of the resulting algorithms to the optimum of the dual support vector machine optimization problem. In particular, the hybrid maximum gain (HMG) working set selection strategy is the fastest method available for large scale problems (that is, where speed is a concern). Its decisive advantage is the efficient usage of the kernel cache, which is achieved by reselecting one variable from the previously used working set whenever possible. This algorithm results in fast iterations, especially on large datasets. However, it also makes the convergence proof more complicated, because the SMO algorithm with HMG working set selection needs to keep track of the previous working set in addition to the current search point, thus extending the state space on which the algorithm operates.

Another speed-up is achieved by planning the forthcoming SMO step ahead in order to compute a good step size. This result is surprising because it is commonly assumed that the optimal solution of each sub-problem results in the best step size. At a conceptual level we show that non-greedy algorithms for the choice of the SMO step size can outperform the greedy standard approach.

The different improvements of the SMO algorithm, in particular HMG working set selection and planning-ahead, have one important property in common. They both rely on additional state information from the previous iteration, in contrast to the standard SMO algorithm which makes its decisions (its choice of working set and step size) based on the current search point and properties thereof only.

In general it is a completely open question which state information is valuable for SMO-type algorithms. A general answer, if it exists, would be insightful, but challenging to obtain. Furthermore, algorithms relying on larger state spaces require new types of convergence proofs, with a need for generally applicable proof techniques. The convergence proofs of both the HMG algorithm and the planning-ahead technique need to handle different cases depending on the type of step taken. These steps in turn (partially) depend on the additional history information. In particular, the simple proof of convergence of the planning-ahead algorithm to the optimum is a step in this direction and may turn out to be useful for future algorithms.

Finally we apply a modern second order working set selection to an SVM variant for online learning. This straight forward application once again demonstrates the benefits of efficient gradient-based optimization relying on second order working set selection, resulting in improved optimization speed and increased sparseness of the approximate solution.

Robust Selection of Well-generalizing Models

SVMs are an elegant and general approach to supervised learning, and achieve low error rates on many real-world learning problems. However, little is known about how to *reliably and efficiently* choose the kernel function and the regularization parameter for a particular task. For conceptual reasons, we rank generalization performance and robustness over efficiency in our studies.

After reviewing standard model selection techniques for support vector machines we develop a novel objective function, combining three ingredients: First, a sampling technique is used to overcome the asymmetry of the popular cross-validation procedure. Second, a probabilistic interpretation of SVM outputs is used to compute the likelihood of the machine on a hold-out set, thus providing a natural interpretation of objective function values. Third, due to the usage of a differentiable sigmoidal model for the conditional probabilities the resulting objective function is differentiable, such that we can apply gradient-based optimization. The application of probability estimates to SVM model selection goes beyond the original work on this method, and our empirical evaluation shows the value of this approach. Following this line of thought we obtain a simple and natural way to incorporate either prior knowledge or regularization via prior distributions on the SVM hyperparameters into the model selection process. This approach is fundamentally different from other Bayesian interpretations of support vector machines, because it directly introduces the probability of correct classification of hold-out sets instead of reinterpreting a given objective function as the negative logarithm of a posterior probability. The NoisyRprop algorithm is introduced for the optimization of the randomized likelihood and posterior objective functions. This new algorithm is generally applicable to the descent of uncertain gradients.

The presence of multiple local optima in common objective functions further highlights the difficulty of the model selection problem. This is not surprising, because the generalization error itself is often found to have this undesirable property. Instead of assessing this problem on the basis of concrete examples, we derive a general sufficient criterion for the multi-modality of the radius margin quotient and the kernel polarization measure.

For lack of alternatives we compare the different candidate strategies for model selec-

tion empirically on a set of artificial toy problems and on a suite of benchmark datasets. This comparison of model selection methods for standard 1-norm SVM classification takes conceptually different approaches into account. Generalization bounds pose natural and in many cases differentiable objective functions for model selection. In contrast, hold-out set based error estimates are more generally applicable, but they require approximations or the probabilistic interpretation mentioned above to obtain differentiable objective functions. As a third concept, the kernel target alignment is completely independent of the learning machine.

It turns out that the preliminary conclusions drawn from the experimental comparison study can not be carried over to completely different types of problems. This is most evident from the qualitatively different performance of the model selection strategies on benchmark problems with Gaussian kernels and on the bioinformatics problem of TIS classification. It is clear that the different methods perform well in certain areas, and that no method is uniformly superior to all others. Nonetheless, the likelihood measure, with and without incorporation of a prior, performs quite well. The extremely efficient maximization of the kernel target alignment measure gives surprisingly good results. This was not expected because this method does not incorporate the type of kernel machine into its decision making process at all. Only gradient-based optimization of the span bound can clearly be excluded, as it regularly gets stuck in local optima, leading to poorly generalizing classifiers. From the experimental results it is impossible to give a general recommendation for hyperparameter selection. In contrast, we obtained counter-examples for all strategies, revealing that none of the methods in the comparison gives good results in all cases.

There are no theoretical results comparing the kernel target alignment either to hold-out error based objective functions or to bounds. Even for the large and important groups of hold-out based objective functions and generalization bounds, the available results concentrate on deviations from the true generalization error. Little is known about the distributions of local optima of such objective functions and the resulting generalization errors, which is the quantity of interest for model selection. Such results would enhance our understanding of the model selection problem for support vector machines and the strengths and limitations of currently available methods.

List of Symbols

α	vector of Lagrange multipliers, variable of the dual SVM problem	29, 40
α^*	optimal point of the dual SVM problem	41
$\alpha^{(t)}$	candidate solution computed in iteration t	42
$[\alpha]$	equivalence class of α w.r.t. the sets $\mathcal{B}(\alpha)$	47
A	learning algorithm	11
$\hat{A}$	kernel target alignment (KTA)	109
$\mathcal{A}$	4-element alphabet of nucleotides	156
b	offset of the affine linear hypothesis	17
B	working set	42, 46
$B^{(t)}$	working set selected in iteration t	42
$\mathcal{B}$	set of all possible working sets	46
$\mathcal{B}(\alpha)$	set of all feasible working sets in the search point α	46
$\bar{B}$	bootstrap error	110
$\hat{B}$	randomized bootstrap error	110
C	SVM regularization parameter	28
$C(X)$	vector space of continuous functions on the input space X	27
D	dataset	10
$\mathcal{D}$	set of possible datasets	10
ε	accuracy parameter of the decomposition algorithm	42
E_{test}	error on a test set	13
E_{train}	error on the training set	14
f	1.) objective function of the dual SVM problem	29, 40
f	2.) SVM function $f(x) = \langle w, x\rangle + b$	108, 116
f^*	optimum (optimal value) of the dual SVM problem	41
γ	concentration parameter of the radial Gaussian kernel	26
g_B	SMO gain for the working set B	52
g_W	SMO gain for the working set selection policy W	57
$\tilde{g}_B$	Newton step gain for the working set B	52
$\tilde{g}_W$	Newton step gain for the working set selection policy W	57
h	hypothesis $X \to Y$	10, 30
h^*	Bayes optimal hypothesis	13
$\hat{h}$	hypothesis minimizing the empirical risk	14
H_B	hyperplane of optimal points corresponding to the working set B	51
$\mathfrak{H}$	set of all measurable hypotheses	11
$\mathcal{H}$	kernel-induced Hilbert space	23
$I_{\text{up}}(\alpha)$	index set used to build working sets	46
$I_{\text{down}}(\alpha)$	index set used to build working sets	46

J_n	set of index sets $I \subset \{1, \ldots, \ell\}$ of size $\lvert I \rvert = n$	110
k	positive semi-definite kernel function	23
K	positive semi-definite kernel Gram matrix	23
$\mathcal{L}$	Lagrangian function	29, 30
L	loss function	12
ℓ	number of training examples, size of the dual SVM problem	10, 40
L_n	lower bound on the dual variable α_n	40
$\bar{\mathcal{L}}$	log-likelihood model selection objective function	113
$\hat{\mathcal{L}}$	randomized log-likelihood function	113
μ	SMO step size	47
$\hat{\mu}$	Newton step size	50
μ^*	step size resulting from the solution of the SMO sub-problem	51
ν	data generating distribution	10
$\hat{\nu}$	empirical distribution	14
$\mathcal{O}$	[Landau symbol] $g \in \mathcal{O}(f)$ iff $\exists\, c \in \mathbb{R}$ s.t. $g/f \to c$	
Φ	feature map $X \to \mathcal{H}$	23
ψ	dual gap function	44, 54
q	upper bound on the working set size	41
ρ	geometric margin of a linear separation	17, 107
$\mathcal{R}$	feasible region of the dual SVM problem	41
$\mathcal{R}^*$	set of optimal points for the dual SVM problem	41
R	radius of the smallest ball containing all training examples	107
$\mathcal{R}_\nu$	risk w.r.t. the distribution ν	12
$\hat{\mathcal{R}}_D$	empirical risk w.r.t. the dataset D	14
$\mathcal{R}^*_\nu$	Bayes risk	13
U_n	upper bound on the dual variable α_n	40
V	vector space for the embedding of the feasible region $\mathcal{R}$	47
v_B	search direction corresponding to the working set B	46
VCdim	Vapnik-Chervonenkis-dimension	21
w	normal vector of the separating hyperplane	17
W	working set selection policy	54
ξ	vector of slack variables	28
X	input space	10
Y	label/target/output space	10

Bibliography

A. D. Anastasiadis, G. D. Magoulas, and M. N. Vrahatis. New Globally Convergent Training Scheme Based on the Resilient Propagation Algorithm. *Neurocomputing*, 64: 253–270, 2005.

A. Baker. *Matrix groups: An introduction to Lie group theory*. Springer-Verlag, 2002.

Y. Baram. Learning by kernel polarization. *Neural Computation*, 17(6):1264–1275, 2005.

P. Bartlett and J. Shawe-Taylor. Generalization Performance of Support Vector Machines and Other Pattern Classifiers. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

F. R. Blattner, G. Plunkett, C. A. Bloch, N. T. Perna, V. Burland, M. Riley, J. Collado-Vides, J. D. Glasner, C. K. Rode, G. F. Mayhew, J. Gregor, N. W. Davis, H. A. Kirkpatrick, M. A. Goeden, D. J. Rose, B. Mau, and Y. Shao. The complete genome sequence of Escherichia coli K-12. *Science*, 277:1453–1462, 1997. ftp://ftp.ncbi.nlm.nih.gov/genbank/genomes/Bacteria/Escherichia_coli_K12/.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast Kernel Classifiers with Online and Active Learning. *Journal of Machine Learning Research*, 5:1579–1619, 2005.

B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT'92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

L. Bottou and C.-J. Lin. Support Vector Machine Solvers. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 1–28. MIT Press, 2007.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

C. J. C. Burges. Geometry and Invariance in Kernel Based Methods. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

G. C. Cawley and N. L. C. Talbot. Preventing Over-Fitting during Model Selection via Bayesian Regularisation of the Hyper-Parameters. *Journal of Machine Learning Research*, 8:841–861, 2007.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001a. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

C.-C. Chang and C.-J. Lin. Training ν-support vector classifiers: Theory and Algorithms. *Neural Computation*, 13(9):2119–2147, 2001b.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing Multiple Parameters for Support Vector Machines. *Machine Learning*, 46(1):131–159, 2002.

P.-H. Chen, R.-E. Fan, and C.-J. Lin. A Study on SMO-type Decomposition Methods for Support Vector Machines. *IEEE Transactions on Neural Networks*, 17:893–908, 2006.

K.-M. Chung, W.-C. Kao, C.-L. Sun, and C.-J. Lin. Radius Margin Bounds for Support Vector Machines with RBF Kernel. *Neural Computation*, 15(11):2643–2681, 2003.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

N. Cristianini, J. Shawe-Taylor, and J. Kandola. On kernel target alignment. In *Proceedings of Neural Information Processing Systems, NIPS'01*, pages 367–373. MIT Press, 2002.

S. Degroeve, B. De Beats, Y. Van de Peer, and P. Rouzé. Feature subset selection for splice site prediction. *Bioinformatics*, 18 Suppl 2:75–83, 2002.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996.

R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working Set Selection Using the Second Order Information for Training Support Vector Machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

F. Friedrichs and C. Igel. Evolutionary Tuning of Multiple SVM Parameters. *Neurocomputing*, 64(C):107–117, 2005.

T. Glasmachers. Degeneracy in Model Selection for SVMs with Radial Gaussian Kernel. In M. Verleysen, editor, *Proceedings of the 14th European Symposium on Artificial Neural Networks (ESANN)*. d-side publications, 2006.

T. Glasmachers. On related violating pairs for working set selection in SMO algorithms. In M. Verleysen, editor, *Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN)*. d-side publications, 2008a. Accepted.

T. Glasmachers. The Planning-Ahead SMO Algorithm. Submitted, 2008b.

T. Glasmachers and C. Igel. Gradient-based Adaptation of General Gaussian Kernels. *Neural Computation*, 17(10):2099–2105, 2005.

T. Glasmachers and C. Igel. Maximum-Gain Working Set Selection for Support Vector Machines. *Journal of Machine Learning Research*, 7:1437–1466, 2006.

T. Glasmachers and C. Igel. Second-Order SMO Improves SVM Online and Active Learning. *Neural Computation*, 20(2):374–382, 2008.

C. Gold and P. Sollich. Model Selection for Support Vector Machine Classification. *Neurocomputing*, 55(1-2):221–249, 2003.

N. Hansen. The CMA evolution strategy: a comparing review. In J. A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer-Verlag, 2006.

N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

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

D. Hush, P. Kelly, C. Scovel, and I. Steinwart. QP algorithms with guaranteed accuracy and run time for support vector machines. *Journal of Machine Learning Research*, 7: 733–769, 2006.

C. Igel and M. Hüsken. Empirical Evaluation of the Improved Rprop Learning Algorithm. *Neurocomputing*, 50:105–123, 2003.

C. Igel, M. Toussaint, and W. Weishui. Rprop using the natural gradient compared to Levenberg-Marquardt optimization. *Trends and Applications in Constructive Approximation. International Series of Numerical Mathematics*, 151:259–272, 2005.

C. Igel, T. Glasmachers, B. Mersch, N. Pfeifer, and P. Meinicke. Gradient-Based Optimization of Kernel-Target Alignment for Sequence Kernels Applied to Bacterial Gene Start Detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(2):216–226, 2007.

C. Igel, V. Heidrich-Meisner, and T. Glasmachers. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

T. Jaakkola, M. Diekhaus, and D. Haussler. Using the Fisher Kernel Method to Detect Remote Protein Homologies. *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, 1999.

H. Jeffreys. An Invariant Form for the Prior Probability in Estimation Problems. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 186(1007):453–461, 1946.

T. Joachims. Making Large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, chapter 11, pages 169–184. MIT Press, 1998.

S. S. Keerthi. Efficient Tuning of SVM Hyperparameters Using Radius/Margin Bound and Iterative Algorithms. *IEEE Transactions on Neural Networks*, 13(5):1225–1229, 2002.

S. S. Keerthi and E. G. Gilbert. Convergence of a Generalized SMO Algorithm for SVM Classifier Design. *Machine Learning*, 46:351–360, 2002.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computation*, 13:637–649, 2001.

S. S. Keerthi, O. Chapelle, and D. DeCoste. Building Support Vector Machines with Reduced Classifier Complexity. *Journal of Machine Learning Research*, 8:1–22, 2006.

S. S. Keerthi, V. Sindhwani, and O. Chapelle. An Efficient Method for Gradient-Based Adaptation of Hyperparameters in SVM Models. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19, NIPS'06*, pages 673–680. MIT Press, 2007.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

G. W. Leibniz. *Discours de métaphysique*. 1686.

C. Leslie, E. Eskin, and W. S. Noble. The Spectrum Kernel: A string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, H. Lauerdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575. World Scientific, 2002.

H. Li and T. Jiang. A class of edit kernels for SVMs to predict translation initiation sites in eukaryotic mRNAs. *Journal of Computational Biology*, 12(6):702–718, 2005.

C.-J. Lin. On the Convergence of the Decomposition Method for Support Vector Machines. *IEEE Transactions on Neural Networks*, 12:1288–1298, 2001.

N. List. Generalized SMO-style decomposition algorithms. In *Proceedings of the 20th Annual Conference on Computational Learning Theory*, 2008. to appear.

N. List and H. U. Simon. A General Convergence Theorem for the Decomposition Method. In J. Shawe-Taylor and Y. Singer, editors, *Proceedings of the 17th Annual Conference on Learning Theory, COLT 2004*, volume 3120 of *LNCS*, pages 363–377. Springer-Verlag, 2004.

N. List and H. U. Simon. General Polynomial Time Decomposition Algorithms. In P. Auer and R. Meir, editors, *Proceedings of the 18th Annual Conference on Learning Theory, COLT 2005*, volume 3559 of *LNCS*, pages 308–322. Springer-Verlag, 2005.

N. List, D. Hush, C. Scovel, and I. Steinwart. Gaps in Support Vector Optimization. In *Learning Theory, 20th Annual Conference on Learning Theory, COLT 2007. Lecture Notes in Computer Science*, volume 4539, pages 336–348, 2007.

F. Markowetz, L. Edler, and M. Vingron. Support Vector Machines for protein fold class prediction. *Biometrical Journal*, 45(3):377–389, 2003.

P. Meinicke, M. Tech, B. Morgenstern, and R. Merkl. Oligo kernels for datamining on biological sequences: A case study on prokaryotic translation initiation sites. *BMC Bioinformatics*, 5:169, 2004.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. In *Philosophical Transactions of the Royal Society of London*, 1909.

B. Mersch, T. Glasmachers, P. Meinicke, and C. Igel. Evolutionary Optimization of Sequence Kernels for Detection of Bacterial Gene Starts. In *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN 2006)*. Springer-Verlag, 2006.

B. Mersch, T. Glasmachers, P. Meinicke, and C. Igel. Evolutionary Optimization of Sequence Kernels for Detection of Bacterial Gene Starts. *International Journal of Neural Systems*, 17(5):369–381, 2007. Selected paper of ICANN 2006.

E. Osuna, R. Freund, and F. Girosi. Improved Training Algorithm for Support Vector Machines. In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII*, pages 276–285. IEEE Press, 1997.

J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 12, pages 185–208. MIT Press, 1998.

J. Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classifiers*, pages 61–74, 1999.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft Margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001.

M. Riedmiller and H. Braun. Rprop: A Fast Adaptive Learning Algorithm. *Proc. of the Int. Symposium on Computer and Information Science VII*, 1992.

K. E. Rudd. EcoGene: a genome sequence database for Escherichia coli K-12. *Nucleic Acids Research*, 28:60–64, 2000.
http://bmb.med.miami.edu/EcoGene/EcoWeb/.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New Support Vector Algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, 13(7):1443–1471, 2001.

B. Schölkopf, K. Tsuda, and J.-P. Vert, editors. *Kernel Methods in Computational Biology*. Computational Molecular Biology. MIT Press, 2004.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

J. Shine and L. Dalgarno. The 3'-terminal sequence of Escherichia coli 16S ribosomal RNA: Complementarity to nonsense triplets and ribosome binding sites. *PNAS*, 71: 1342–1346, 1974.

I. Steinwart. On the Influence of the Kernel on the Consistency of Support Vector Machines. *Journal of Machine Learning Research*, 2:67–93, 2001.

I. Steinwart. Support Vector Machines are Universally Consistent. *J. Complexity*, 18(3): 768–791, 2002.

I. Steinwart. Sparseness of Support Vector Machines—Some Asymptotically Sharp Bounds. In *Proceedings of Neural Information Processing Systems, NIPS'03*, 2003.

C. J. Stone. Consistent Nonparametric Regression. *The Annals of Statistics*, 5(4):595–620, 1977.

N. Takahashi and T. Nishi. Rigorous Proof of Termination of SMO Algorithm for Support Vector Machines. *IEEE Transaction on Neural Networks*, 16(3):774–776, 2005.

I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

V. Vapnik. *Statistical Learning Theory*. Wiley, New-York, 1998.

V. Vapnik and O. Chapelle. Bounds on Error Expectation for Support Vector Machines. *Neural Computation*, 12:2013–2036, 2000.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. SimpleSVM. In T. Fawcett and N. Mishra, editors, *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, pages 760–767. AAAI Press, 2003.

D. H. Wolpert and W. G. Macready. No Free Lunch Theorems for Search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K. R. Müller. Engineering Support Vector Machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, 2000.

Lebenslauf

Persönliche Daten

Name	Tobias Glasmachers
Geburtsdatum	6. Oktober 1976
Geburtsort	Bochum
E-Mail	tobias.glasmachers@neuroinformatik.rub.de

Ausbildung und beruflicher Werdegang

1983 - 1987	Gemeinschaftsgrundschule Wetter Volmarstein
1987 - 1996	Christian-Rohlfs-Gymnasium Hagen Haspe
Juni 1996	Schulabschluss mit Abitur
1996 - 1997	Zivildienst in der Behindertenwerkstatt der Evangelischen Stiftung Volmarstein
1997 - 2004	Studium der Mathematik mit Nebenfach Physik an der Ruhr-Universität Bochum
Juni 2004	Abschluss des Studiums mit Diplom
2004 - heute	Wissenschaftlicher Mitarbeiter am Institut für Neuroinformatik an der Ruhr-Universität Bochum