Institut für Logik, Komplexität
und Deduktionssysteme
Fakultät für Informatik
Universität Karlsruhe

Bayesian methods for Support Vector machines and Gaussian processes

Diplomarbeit von
Matthias Seeger

Betreuer und Erstgutachter:	Dr Christopher K. I. Williams Division of Informatics University of Edinburgh, UK
Betreuer und Zweitgutachter:	Prof Dr Wolfram Menzel Institut für Logik, Komplexität und Deduktionssysteme

Tag der Anmeldung: 1. Mai 1999
Tag der Abgabe: 22. Oktober 1999

Ich erkläre hiermit, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 22. Oktober 1999

Matthias Seeger

Zusammenfassung

In dieser Arbeit formulieren wir einen einheitlichen begrifflichen Rahmen für die probabilistische Behandlung von Kern- oder Spline-Glättungs-Methoden, zu denen populäre Architekturen wie Gaußprozesse und Support-Vector-Maschinen zählen. Wir identifizieren das Problem nicht normalisierter Verlustfunktionen und schlagen eine allgemeine, zumindest approximative Lösungsmethode vor. Der Effekt, den die Verwendung solcher nicht normalisierter Verlustfunktionen induzieren kann, wird am Beispiel des Support-Vector-Klassifikators intuitiv verdeutlicht, wobei wir den direkten Vergleich mit dem Bayesschen Gaußprozess-Klassifikator als nichtparametrische Verallgemeinerung logistischer Regression suchen. Diese Interpretation setzt Support-Vector-Klassifikation in Bezug zu Boosting-Techniken.

Im Hauptteil dieser Arbeit stellen wir einen neuen Bayesschen Modell-Selektionsalgorithmus für Gaußprozessmodelle mit allgemeinen Verlustfunktionen vor, der auf der variationellen Idee basiert. Dieser Algorithmus ist allgemeiner einsetzbar als bisher vorgeschlagene Bayessche Techniken. Wir zeigen anhand der Resultate einer Reihe von Klassifikationsexperimenten auf Datenmengen natürlichen Ursprungs, daß der neue Algorithmus leistungsmäßig mit den besten bekannten Verfahren für Modell-Selektion von Kernmethoden vergleichbar ist.

Eine weitere Zielsetzung dieser Arbeit war, eine leicht verständliche Brücke zu schlagen zwischen den Feldern probabilistischer Bayesscher Verfahren und Statistischer Lerntheorie, und zu diesem Zweck haben wir eine Menge Text tutorieller Natur hinzugefügt. Wir hoffen, daß dieser Teil der Arbeit für Wissenschaftler aus beiden Bereichen von Nutzen ist.

Teile dieser Arbeit werden unter dem Titel “Bayesian model selection for Support Vector machines, Gaussian processes and other kernel classifiers” auf der jährlichen Konferenz für *Neural Information Processing Systems (NIPS)* 1999 in Denver, Colorado, USA vorgestellt werden, das Papier kann in den proceedings der Konferenz eingesehen werden.

Abstract

We present a common probabilistic framework for *kernel* or *spline smoothing* methods, including popular architectures such as Gaussian processes and Support Vector machines. We identify the problem of unnormalized loss functions and suggest a general technique to overcome this problem at least approximately. We give an intuitive interpretation of the effect an unnormalized loss function can induce, by comparing Support Vector classification (SVC) with Gaussian process classification (GPC) as a nonparametric generalization of logistic regression. This interpretation relates SVC to boosting techniques.

We propose a variational Bayesian model selection algorithm for general normalized loss functions. This algorithm has a wider applicability than other previously suggested Bayesian techniques and exhibits comparable performance in cases where both techniques are applicable. We present and discuss results of a substantial number of experiments in which we applied the variational algorithm to common real-world classification tasks and compared it to a range of other known methods.

The wider scope of this thesis is to provide a bridge between the fields of probabilistic Bayesian techniques and Statistical Learning Theory, and we present some material of tutorial nature which we hope will be useful to researchers of both fields.

Parts of this work will be presented at the annual conference on *Neural Information Processing Systems (NIPS)* 1999 in Denver, Colorado, USA, under the title "Bayesian model selection for Support Vector machines, Gaussian processes and other kernel classifiers" and will be contained in the corresponding conference proceedings.

Acknowledgments

This thesis was written while the author visited the Institute for Adaptive and Neural Computation (ANC), Division of Informatics, University of Edinburgh, Scotland, from January to September 1999.

My special thanks go to Dr Chris Williams (ANC, Edinburgh) who acted as supervisor "across borders" and had to spend a lot of efforts to get things going. His great knowledge about (and belief in!) Bayesian techniques of all sorts and kernel methods, his incredible overview of the literature, his steady interest in my work, his innumerably many hints and suggestions and so many very long discussions have shaped this work considerably. I am looking forward to continue my work with him in Edinburgh.

Many thanks also to Dr Amos Storkey (ANC, Edinburgh) for discussions outside and sometimes inside the pub from which I learned a lot. Chris and Amos also bravely fought their way through this monster of a thesis, gave valuable advice in great detail and corrected many of my somewhat too German expressions.

Thanks to Dr Peter Sollich (Kings College, London) for helpful discussions about Bayesian techniques for SVM and related stuff, and to Stephen Felderhof, Nick Adams, Dr Stephen Eglen, Will Lowe (all ANC) and William Chesters (DAI, Edinburgh).

My thesis supervisor in Karlsruhe, Germany, and "Betreuer der Diplomarbeit" was Prof Dr Wolfram Menzel. I owe many thanks to him for acting very flexible and tolerant in this somewhat unusual project of an "Auslandsdiplomarbeit". He took many efforts with respect to organization, suggested many possibilities for funding (one of which succeeded, see below) and showed much interest in my work done abroad. We also had some discussions which were very valuable.

I would also like to thank David Willshaw, head of the ANC, who together with Chris made possible my visit in Edinburgh, and the Division of Informatics for waiving my bench fees in Edinburgh and funding my visit of the Kernel workshop in Dortmund, Germany, in summer 1999.

I gratefully acknowledge a scholarship which I was awarded by the *Prof Dr Erich Müller Stiftung* to cover living expenses abroad.

Last, but not least, many thanks to old friends in Karlsruhe and new friends in Edinburgh with whom I had so much fun during this exciting time.

This thesis is dedicated to my mother and my sisters, and was written in memory of my father.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we give an informal overview over the topics we are concerned about in this thesis. We then define the notation used in the rest of the text and review architectures, methods and algorithms of central importance to an understanding of this work. Experienced readers might want to skip this chapter and use it in a lookup manner.

1.1 Overview

Kernel or *spline smoothing methods* are powerful nonparametric statistical models that, while free from unreasonable parametric restrictions, allow to specify prior knowledge about an unknown relation between observables in a convenient and simple way. Unless otherwise stated, we will concentrate here on the problem of *classification* or *pattern recognition*. It is, however, straightforward to apply most of our results to *regression estimation* as well. In any reasonable complex parametric model, like for example a neural network, simple prior distributions on the adjustable parameters lead to an extremely complicated distribution over the actual function the model computes, and this can usually only be investigated in limit cases. Now, by observing that for two-layer sigmoid networks with a growing number of hidden units this output distribution convergences against a simple *Gaussian process (GP)*, Neal [Nea96], Williams and Rasmussen [WR96] focused considerable and ongoing interest on these kernel models within the neural learning community.

1.1.1 History of Support Vector machines

Another class of kernel methods, namely *Support Vector machines*, have been developed from a very different viewpoint. Vapnik and Chervonenkis were concerned with the question under what conditions the *ill-posed problem* of learning the probabilistic dependence between an input and a response variable[1] can actually be solved uniquely, and what paradigm should be proposed to construct learning algorithms for this task? A candidate for such a paradigm was quickly found in the widely used principle of *empirical risk minimization (ERM)*: From all functions in the hypothesis space, choose one that minimizes the empirical risk, i.e. the loss averaged over the *empirical distribution* induced by the training sample. It is well-known that this principle fails badly when applied to reasonably "complex" hypothesis spaces, resulting in "overfitted" solutions that follow random noise on the training sample rather than abstract from such and generalize. The problem of overfitting can easily be understood by looking at a correspondence to *function interpolation*, see for example [Bis95]. In search for a complexity measure for possibly infinite hypothesis spaces, Vapnik and Chervonenkis proposed the *Vapnik-Chervonenkis dimension*, a combinatorial property of a function set that can be calculated or bounded for most of the commonly used learning architectures. They proved that once a function family has finite VC dimension, the probability of an ε deviation of the minimal empirical risk from the minimal risk over that family converges to zero exponentially fast in the number of training examples, and the exponent of this convergence depends only on the VC dimension and the accuracy ε. They also showed the converse, namely that no paradigm at all can construct algorithms that learn classes with infinite VC dimension. In this sense, ERM is optimal as learning paradigm over suitably restricted hypothesis spaces. These results laid the foundation of *Statistical Learning Theory*.

Instead of demonstrating their theory on neural networks, Vapnik and Chervonenkis focussed on linear discriminants and proved VC bounds for such families. It turns out that to restrict the VC dimension of a class of seperating hyperplanes one can for example demand that the minimal distance to all datapoints is bounded below by a fixed constant. This idea, namely that a certain well-defined distribution (called the *margin distribution*) and statistics thereof (in our case the minimum sample margin, i.e. distance from

[1] This problem can formally be defined by choosing a loss function and a hypothesis space (both choices are guided by our prior belief into the nature of the combination of underlying cause and random noise for the random correspondence to be learned), and then ask for a function in the space that minimizes the *expected loss* or *risk*, where the expectation is over the true, unknown distribution of input and response variable.

the data points) are strongly connected to the mysterious property of *generalization capability* (the most important qualitative performance measure for an adaptive system) is currently a hot topic in statistical and computational learning theory and by no means completely understood. We will return to the notion of margins below.

Surprisingly, only very much later, the idea of large margin linear discriminants was taken up again (by Vapnik) and generalized to the nonlinear case. This generalization, sometimes referred to as "kernel trick", builds on an easy consequence of Hilbert space theory and has been widely used long before in the Statistics community, but the combination with large margin machines was novel and resulted in the Support Vector machine, a new and extremely powerful statistical tool.

There are excellent reviews on Statistical Learning Theory and Support Vector machines (see for example [Vap95],[Vap98], [Bur98b]), and we will not try to compete with them here. Large margin theory is far out of the scope of this work, although we feel that by having included these lines we might have given an insight into the fascinating scientific "birth" of Support Vector machines and maybe awakened some genuine interest in the reader.

1.1.2 Bayesian techniques for kernel methods

This thesis reviews and clarifies the common roots of Gaussian process and Support Vector models, analyzes the actual differences between the domains and finally exploits the common viewpoint by proposing new techniques of Bayesian nature that can be used successfully in both fields.

Only very recently there has been interest in such Bayesian methods for Support Vector classification while Bayesian methods for Gaussian process models are successful and widely established. Let us have a look at possible reasons for this divergence. As we will show below, and as is well known, the only difference between the domains from a modeling viewpoint are the loss functions used. Support Vector classification employs losses of the ε-insensitive type [Vap98] while Gaussian process models make use of smooth differentiable loss functions. How does this affect the applicability of typical Bayesian methods?

Firstly, in its exact form, Bayesian analysis is usually intractable, and sensible yet feasible approximations have to applied. Traditionally these focus on gradient and curvature information of the log probability manifold of the posterior which is (as we argue below) not possible if nondifferentiable loss

functions of the ε-insensitive type are used. However, we show how to overcome this problem using variational techniques.

Secondly, at present the running-time scaling behaviour of Bayesian methods for kernel classifiers (like Gaussian processes) is cubic in the number of training points which reduces the applicability of these models severely. This scaling has to be contrasted with the behaviour of fast SVC implementations like [Pla98] with seems to be somewhat quadratic in the number of Support Vectors, usually only a small fraction of the training set, and is for many datasets essentially linear in the training set size[2]. However, very powerful approximative techniques for Bayesian numerical analysis have been developed (see [Ski89]) for,

> ... if Bayesians are unable to solve the problems properly,
> other methods will gain credit for improper solutions.
>
> *John Skilling*

These methods have been used for Gaussian process regression and classification in [Gib97], and we are currently exploring their applicability within our work presented here. Note that an efficient and numerically stable implementation of these methods is *not at all* straightforward, which might explain why they have not been widely used so far in the Bayesian community.

Thirdly, Gaussian process discriminants lack a very appealing property of the Support Vector machine, its *sparse solution*. The final discriminant is a function of only a (typically) small fraction of the data points, the so-called *Support Vectors*, and completely independent of the rest of the dataset. Of course, it is not known from the beginning which of the points will be the Support Vectors, but nevertheless the assumption that the final set will be small can greatly speed up the training process, and prediction over large test sets benefits from the sparseness if running time is concerned. However, the sparseness property is based on the ε-insensitive loss functions and not on the use of special prior covariance kernels, therefore applying Bayesian techniques for kernel model selection does not affect this property at all.

Fourthly, a final gap between Support Vector classification and probabilistic generative Gaussian process models remains unbridged. The ε-insensitive loss does not correspond to a noise distribution since it cannot be normalized in the sense discussed below. "Hence, a direct Bayesian probabilistic interpretation of SVM is not fully possible (at least in the simple MAP approach that

[2]The worst-case scaling of SMO as one of the fastest SVC training algorithms is more than quadratic in the training set size, and the occurrence of such superquadratic scaling is not extremely unlikeli (Alex Smola, personnal communication).

we have sketched)." [OW99], and we will not try to sketch anything more complicated, although there are possible probabilistic generative interpretations of SVC [Sol99]. Rather than closing it, we will explore this gap more closely and thereby show the relations between GPC and SVC in a new light. For practical Bayesian analysis however, we will resort to a probabilistic kernel regression model to which Support Vector classification can be regarded as an approximation.

The thesis is organized as follows. The remainder of this chapter introduces notation, models and paradigms. Gaussian process regression, classification and Support Vector classification are reviewed. The following chapter develops a common framework for Gaussian process and Support Vector classifiers. We also give an argument that might help to understand the difference between Support Vector classification models and probabilistic kernel regression classifiers like such based on Gaussian processes. The third chapter is tutorial in nature and can be skipped by experienced readers, although we think that it reviews some interesting "new views on old stuff" which seem not to be widely known. Chapter four is the main part of the thesis and discusses Bayesian model selection for architectures within the framework developed in earlier chapters, with special consideration of Support Vector classification. A new variational algorithm is suggested, analyzed and compared with related methods. Experiments and results are described, and several extensions are proposed for future work. The Appendix contains longer calculations and discussions that would disturb the flow in the main text. The *STATSIM* system which was used to implement all experiments related to this thesis, is also briefly described there.

1.2 Models, definitions and notation

1.2.1 Context-free notation

Most of the notation used in this thesis is defined here. However, we cannot avoid postponing some definitions to later sections of the introduction, since they require the appropriate context to be introduced.

We will denote vectors in boldface (e.g. $\boldsymbol{x}$) and matrices in calligraphic letters (e.g. $\boldsymbol{A}$). x_i refers to the i-th component in $\boldsymbol{x} = (x_i)_i$, likewise $\boldsymbol{A} = (\alpha_{ij})_{ij}$. $\boldsymbol{A}^t$ denotes transposition, $|\boldsymbol{A}|$ the absolute value of the determinant, $\mathrm{tr}\,\boldsymbol{A}$ the trace (i.e. the sum over the diagonal elements), $\mathrm{diag}\,\boldsymbol{A}$ the diagonal of $\boldsymbol{A}$ as vector and $\mathrm{diag}\,\boldsymbol{x}$ the matrix with diagonal $\boldsymbol{x}$ and 0 elsewhere. Given any set $A \subset \mathbb{R}$, I_A denotes the indicator function of A, i.e. $I_A(x) = 1$ if $x \in A$, and

0 elsewhere. If $\rho(x)$ is a predicate (or event), we write $I_{\{\rho\}}(x) = 1$ if $\rho(x)$ is true, 0 otherwise. By $[u]_+$ we denote the hinge function $[u]_+ = uI_{\{u \geq 0\}}$.

The probability spaces we deal with are usually finite-dimensional Euclidean ones. We consider only probability measures which have a density with respect to Lebesgue measure. A (measurable) random variable $\boldsymbol{x}$ over such a space induces a probability measure itself, which we denote by $P_{\boldsymbol{x}}$ or often simply P. The density of $\boldsymbol{x}$ is also denoted by $P(\boldsymbol{x})$. Note that $P(\boldsymbol{x})$ and $P(\boldsymbol{y})$ are different densities if $\boldsymbol{x} \neq \boldsymbol{y}$. For convenience, we use $\boldsymbol{x}$ to denote both the random variable and its actual value. This is used also for random processes: $y(\cdot)$ might denote both a random process and its actual value, i.e. a function. Measures and densities can be conditioned by events E, which we denote by $P(\boldsymbol{x}|E)$. Special E are point events for random variables. As an example, $P(\boldsymbol{x}|\boldsymbol{y})$ denotes the density of $\boldsymbol{x}$, conditioned on the event that the random variable $\boldsymbol{y}$ attains the value $\boldsymbol{y}$. $N(\mu, \sigma^2)$ denotes the univariate Gaussian distribution, $N(x|\mu, \sigma^2)$ its density. $N(\boldsymbol{\mu}, \Sigma)$ is the multidimensional counterpart.

K and R usually denote symmetric, positive definite kernels with associated covariance matrices $\boldsymbol{K}$ and $\boldsymbol{R}$, evaluated on the data points. If both symbols appear together, we usually have $K = CR$, where C is a constant called *variance parameter*. Implicit in this notation is the assumption that R has no parametric prefactor.

1.2.2 The problem

Let X be a probability space (e.g. $X = \mathbb{R}^d$) and $D = (\boldsymbol{X}, \boldsymbol{t}) = \{(\boldsymbol{x}_1, t_1), \ldots, (\boldsymbol{x}_n, t_n)\}$, $\boldsymbol{x}_i \in X$, $t_i \in \{-1, +1\}$ a noisy independent, indentically distributed (i.i.d.) sample from a *latent function* $y : X \to \mathbb{R}$, where $P(t|y)$ denotes the noise distribution. In the most general setting of the prediction problem, the noise distribution is unknown, but in many settings we have a very concrete idea about the nature of the noise. Let $\boldsymbol{y} = (y_i)_i$, $y_i = y(\boldsymbol{x}_i)$, for a given function or random process $y(\cdot)$. Given further points $\boldsymbol{x}_*$ we wish to predict t_* so as to minimize the error probability $P(t_*|\boldsymbol{x}_*, D)$, or (more difficult) to estimate this probability. In the following, we drop the conditioning on points from X for notational convenience and assume that all distributions are implicitely conditioned on all relevant points from X, for example $P(t_*|\boldsymbol{t})$ on $\boldsymbol{X}$ and $\boldsymbol{x}_*$.

There are two major paradigm to attack this problem:

- Sampling paradigm
- Diagnostic paradigm

Methods following the sampling paradigm model the class conditional input distributions $P(\boldsymbol{x}|t),\ t \in \{-1, +1\}$ explicitely and use Bayes formula to compute the predictive distributions. The advantage of this approach is that one can estimate the class conditional densities quite reliably from small datasets if the model family is reasonably broad. However, the way via the class conditional densities is often wasteful: we don't need to know the complete shape of the class distributions, but only the boundary between them to do a good classification job. Instead of the inductive step to estimate the class densities and the deductive step to compute the predictive probabilities and label the test set, it seems more reasonable to only employ the transductive step to estimate the predictive probabilities directly (see [Vap95]), which is what diagnostic methods do. No dataset is ever large enough to obtain point estimates of the predictive probabilities $P(t|\boldsymbol{x})$ (if X is large), and most diagostic methods will therefore estimate *smoothed* versions of these probabilities. This thesis deals with diagnostic, nonparametric (or *predictive*) methods, and we refer to [Rip96] for more details on the other paradigms.

Two major subclasses within the diagnostic, nonparametric paradigm are:

- Bayesian methods
- Nonprobabilistic (discriminative) methods

We will refer to nonprobabilistic methods as *discriminative* methods because they focus on selecting a discriminant between classes which need not have a probabilistic meaning. We will briefly consider each of these classes in what follows. We will sometimes refer to Bayesian methods as *generative* Bayesian methods, to emphasize the fact that they use a generative model for the data.

1.2.3 The Bayesian paradigm

Generative Bayesian methods start by encoding all knowledge we might have about the problem setting into a *prior distribution*. Once we have done that, the problem is solved in theory, since every inference we might want to perform upon the basis of the data follows by "turning the Bayesian handle", i.e. performing simple and mechanical operations of probability theory:

1. Conditioning the prior distribution on the data D, resulting in a *posterior distribution*
2. Marginalization (i.e. integration) over all variables in the posterior we are not interested in at the moment. The remaining *marginal posterior* is the complete solution to the inference problem

The most general way to encode our prior knowledge about the problem above is to place a stochastic process prior $P(y(\cdot))$ on the space of latent functions. A stochastic process $y(\cdot)$ can be defined as a random variable over a probability space containing functions, but it is more intuitive to think of it as a collection of random variables $y(\boldsymbol{x}),\ \boldsymbol{x} \in X$. How can we construct such a process? Since the $y(\boldsymbol{x})$ are random variables over the same probability space Ω, we can look at *realizations* (or *sample paths*) $\boldsymbol{x} \mapsto (y(\boldsymbol{x}))(\omega)$ for fixed $\omega \in \Omega$. However, it is much easier to specify the *finite-dimensional distributions (fdd)* of the process, being the joint distributions $P_{(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)}(y_{\boldsymbol{x}_1},\ldots,y_{\boldsymbol{x}_m})$ for every finite ordered subset $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)$ of X. It is important to remark that the specification of all fdd's does not suffice to uniquely define a random process, in general there is a whole family of processes sharing the same fdd's (called *versions* of the specification of the fdd's). Uniqueness can be enforced by concentrating on a restricted class of processes, see for example [GS92], chapter 8.6. In this thesis, we won't study properties of sample paths of processes, and we therefore take the liberty to identify all versions of a specification of fdd's and call this family a random process.

Consider assigning to each finite ordered subset $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)$ of X a joint probability distribution $P_{(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)}(y_{\boldsymbol{x}_1},\ldots,y_{\boldsymbol{x}_m})$ over $\mathbb{R}^m$. We require this assignment to be *consistent* in the following sense: For any finite ordered subset $X_1 \subset X$ and element $\boldsymbol{x} \in X \setminus X_1$ we require that

$$\int P_{X_1 \cup \{\mathbf{x}\}}\, dy(\boldsymbol{x}) = P_{X_1}. \tag{1.1}$$

Furthermore, for any finite ordered subset $X_1 \subset X$ and permutation π we require that

$$P_{\pi X_1}(\pi y(X_1)) = P_{X_1}(y(X_1)). \tag{1.2}$$

Here we used the notation $y(X_1) = (y_{\boldsymbol{x}})_{\boldsymbol{x} \in X_1}$. Note that all mentioned sets are ordered. These requirements are called *Kolmogorov consistency conditions* and are obviously a necessary property of the definition of fdd's. One can show that these conditions are also sufficient ([GS92], chapter 8.6), and since we indentified version families with processes, we have shown how to construct a process using only the familiar notion of finite-dimensional distributions.

Now, there is an immense variety of possible constructions of a process prior. For example, we might choose a parametric family $\{y(\boldsymbol{x};\theta) \mid \theta \in \Theta\}$ and a prior distribution $P(\theta)$ over the space Θ which is usually finite-dimensional. We then define $y(\cdot) = y(\cdot;\theta),\ \theta \sim P(\theta)$. This is referred to as a *parametric* statistical architecture. Examples are linear discriminants with fixed basis functions or multi-layer perceptrons.[3] Note that given the value of θ, $y(\cdot)$ is completely determined.

Another possibility is to use a Gaussian process prior. We will return to this option shortly, but remark here that in contrast to a parametric prior a Gaussian process cannot in general be determined (or parameterized) by a finite-dimensional random variable. If $y(\cdot)$ is a Gaussian process, there is in general no finite-dimensional random variable θ such that given θ, $y(\cdot)$ is completely determined. However, a countably infinite number of real parameters does the job, which is quite remarkable since X might not be countable.

What is the Bayesian solution to our problem, given a prior on $y(\cdot)$? We first compute the posterior distribution over all hidden (or latent) variables involved:

$$P(\boldsymbol{y}, y_*|D) \tag{1.3}$$

We then marginalize over $\boldsymbol{y}$ to obtain

$$P(y_*|D) = \int P(y_*|\boldsymbol{y})P(\boldsymbol{y}|D)\, d\boldsymbol{y}. \tag{1.4}$$

This follows from $P(\boldsymbol{y}, y_*|D) = P(y_*|\boldsymbol{y}, D)P(\boldsymbol{y}|D) = P(y_*|\boldsymbol{y})P(\boldsymbol{y}|D)$, where we used the fact that y_* and D are conditionally independent given $\boldsymbol{y}$. $P(y_*|\boldsymbol{y})$ depends only on the prior, and we can use *Bayes formula* to compute $P(\boldsymbol{y}|D)$:

$$P(\boldsymbol{y}|D) = \frac{P(D|\boldsymbol{y})P(\boldsymbol{y})}{P(D)}, \tag{1.5}$$

where the likelihood $P(D|\boldsymbol{y}) = \prod_i P(t_i|y_i)$ and $P(D)$ is a normalization constant. $P(t_*|D)$ can then be obtained by averaging $P(t_*|y_*)$ over $P(y_*|D)$.

1.2.4 The discriminative paradigm

Methods under the *discriminative* paradigm are sometimes called *distribution-free*, since they make no assumptions about unknown variables

[3]Strange enough, the latter are sometimes referred to as *nonparametric*, just because they (usually) have a lot of parameters. We don't follow this inconsistent nomenclature.

whatsoever. Their approach to the prediction problem is to choose a *loss function* $g(t, y)$, being an approximation to the *misclassification loss* $I_{\{ty \leq 0\}}$ and then to search for a discriminant $y(\cdot)$ which minimizes $\mathrm{E}g(t_*, y(\boldsymbol{x}_*))$ for the points $\boldsymbol{x}_*$ of interest (see [Wah98]). The expectation is over the true distribution $p_{true} = P(t|\boldsymbol{x})$, induced by the latent function and the unknown noise distribution. Of course, this criterion is not accessible, but approximations based on the training sample can be used. These approximations are usually consistent in the limit of large n in that the minimum argument of the approximation tends to the minimum argument of the true criterion. The behaviour for finite, rather small samples is often less well understood. [Wah98] and [WLZ99] suggest using proxies to the *generalized comparative Kullback-Leibler distance (GCKL)*:

$$GCKL(p_{true}, y(\cdot)) = \mathrm{E}_{true}\left[\frac{1}{n}\sum_{i=1}^{n} g(t_i, y(\boldsymbol{x}_i))\right], \tag{1.6}$$

where the expectation is over future, unobserved t_i. The corresponding points $\boldsymbol{x}_i$ are fixed to the points in the training set, so we rely on these being a typical sample of the unknown distribution $P(\boldsymbol{x})$ and the relations between the latent values y_i at these points being characteristic for the relations between latent points elsewhere. Note that if $g(t, y)$ is the misclassification loss, the GCKL is the expected misclassification rate for $y(\cdot)$ on unobserved instances if they have the same distribution on the $\boldsymbol{x}_i$ as the training set. Many loss functions g are actually *upper bounds* of this special misclassification loss (including the Support Vector classification loss discussed below), and therefore minimizing the GCKL of such a loss function w.r.t. free model parameters can be regarded as a *variational method* to approach low generalization error. We will discussion the principles of variational methods in detail below. The GCKL itself is of course not accessible since the true law p_{true} is unknown, but computational proxies based on the technique of *generalized approximative cross validation* can be found, see [Wah98] and [WLZ99].

A huge number of other distribution-free methods are analyzed in the excellent book [DGL96]. Maybe the most promising general technique is the principle of *Structural Risk Minimization (SRM)* (see [Vap98]) which is based on the idea of VC dimension mentioned above. Suppose we have a hypothesis space of (in general) infinite VC dimension. *PAC* theory proposes frequentist bounds on the minimal expected loss which come with two adjustable parameters, namely the *accuracy* ε and the *confidence* δ. PAC means *probably approximately correct*, and PAC bounds assure that "probably" (with confidence δ) the empirical risk is "approximately" (at most ε away from) the "correct" expected risk, and this statement holds for all input distributions

and all underlying hypotheses from the hypothesis space. The probabilities are taken over the i.i.d. sampling from the true underlying distribution. It is important to note that this notion of frequentist probabilities is fundamentally different from Bayesian beliefs. Within the frequentist framework, the value of any information we extract from given data (in the form of an estimator) can only be assessed over the ensemble of *all* datasets from an unknown source. To actually perform such an assessment, we would need many datasets from exactly the same source. Some frequentist techniques split given data to be able to assess their estimators at least approximately, but this is clearly wasteful. The idea behind frequentist techniques like PAC or *tail* bounds or statistical tests with frequentist confidence regions is that even though we have no access to the source ensemble, we can exploit the fact that our dataset has been generated i.i.d. from the source and therefore exhibits a certain form of regularity. This phenomenon is referred to as *asymptotic equipartition property (AEP)* (see [CT91]) and is conceptually closely related to the notion of *ergodicity*. The AEP holds also for some non i.i.d. sources, and frequentist tests can in general be applied to these sources, but the methods become more and more complicated and need large datasets to reach satisfying conclusions. In contrast to that, Bayesian prior beliefs are assessed using (necessarily subjective) experience which can be gained from expert knowledge or previous experiments. These beliefs are conditioned on the given data, without the need to refer to the data ensemble or to waste data for assessment of the analysis. The posterior probability distribution (or characteristics thereof) is the most complete statement of conclusions that can be drawn from the data (given the model), and we do not need to employ awkward, nonprobabilistic constructions like confidence intervals and p values.

Typical VC bounds on the minimal expected loss consist of the sum of the minimal empirical loss (a function of the training sample) and a complexity penalty depending on the VC dimension of the hypothesis class, but they only apply to spaces of finite complexity. The idea behind SRM is to choose a sequence of nested subsets of the hypothesis space. Each of these subsets has finite VC dimension, and the union of all subsets is the whole space. We start from the smallest subset and gradually move up in the hierarchy. In every set, we detect the minimizer of the empirical risk and evaluate the bound. We keep track of the minimizer which achieved the smallest value of the bound so far. Since the penalty term is monotonically increasing in the VC dimension, we only need to consider finitely many of the subsets (the loss function is bounded from below, and so is the expected loss term). The discriminant selected in this way has low guaranteed risk.

The original formulation of SRM required the hypothesis class hierarchy to be specified in advance, without considering the actual data. This is not only extremely inefficient, but is also clearly violated by the method Support Vector machines are trained: The minimum margin which is maximized by the training algorithm is a function of the sample. This dilemma was resolved in [STBWA96] where the important notion of *luckiness functions* was introduced. In a nutshell, a luckiness function ranks samples such that the higher luckiness value a sample achieves, the better (i.e. smaller) the penalty terms in certain VC-style upper bounds on the risk are. These bounds which are parameterized by the luckiness value, are still distribution-free. If the distribution is actually very "unusual", the luckiness value of a typical sample will be high and the value of the bound is large. The important point about luckiness-based bounds is that they can be biased in the following sense. Suppose we have some prior knowledge or expectations about the behaviour of the luckiness value under the true unknown distribution. We can then choose, as direct consequence of this prior knowledge, the values of a sequence of free parameters of the bound which can actually be interpreted as a prior distribution. The bound will be the tighter, the closer this sequence is to the true distribution of the luckiness variable. We think that this is actually a quite close correspondence to the Bayesian paradigm, although the luckiness framework is far more complicated than the Bayesian one. As non-experts in the former field, we hope that some day a synthesis will be achieved that combines the conceptually simple and elegant Bayesian method with somewhat more robust *PAC*-based techniques A very promising approach was shown by McAllester (see [McA99b], [McA99a]), but see also [HKS94].

Support Vector classification is an example of a discriminative technique. The loss function used there is

$$g(t_i, y_i) = [1 - t_i y_i]_+, \tag{1.7}$$

which will be referred to as *SVC loss*. The SVC loss is an upper bound to the misclassification loss (see [Wah98]). Furthermore, the size of the minimum margin is a luckiness function (as is the negative VC dimension). However, we will depart from this point of view and regard the penalty term in the SVC criterion as coming from a Bayesian-style prior distribution.

1.2.5 Comparison of the paradigms

Now, what paradigm should we follow given a special instantiation of the above problem? There's a long, fierce and ongoing debate between followers of either paradigm. Bayesians argue that there is no other proper and

consistent method for inference than the Bayesian one. Not quantifying prior knowledge in distributions and including them in the inference process means that information is lost. This is wasteful and can lead to false conclusions, especially for small training sample sizes. Advocats of the discriminative paradigm criticize the *subjectivity* introduced into the problem by choosing a prior. Furthermore, the Bayesian approach might fail if the assumed prior is far from the true distribution having generated the data. Both problems can be alleviated by choosing hierarchical priors. There are also guidelines, like the *maximum entropy principle* (see [Jay82]), that allow us to choose a prior with "minimum subjectivity", given constraints that many people (including the critiques) would agree about.

Maybe the most serious drawback about the Bayesian paradigm (at least from a practical viewpoint) is its immense computational complexity. Marginalization involves integration over spaces of huge dimension, and at present no known numerical technique is able to perform such computations reliably and efficiently. Thus, we are forced to use crude approximations, and the error introduced by these often cannot be determined to reasonable accuracy since the true posterior distributions are not accessible. However, as we and most Bayesians would argue, it should be more reasonable to begin with doing the right thing and gradually apply approximations where absolutely necessary, than to throw all prior knowledge away and rely on concentration properties of large, i.i.d. samples only.

Methods in the discriminative paradigm have the advantage that they are robust against false assumptions by the simple fact that they make no assumptions. However, by ignoring possibly available prior information about the problem to solve, they tend to have worse performance on small and medium sized samples. By concentrating entirely on the generalization error, they fail to answer other questions related to inference, while Bayesian methods are at least able to give subjective answers (an important example is the computation of error bars, i.e. the variance of the predictive distribution in the Bayesian framework, see for example [Bis95], chapter 6.5). However, as mentioned above, recent developments in the PAC theory have shown that distribution-free bounds on the generalization error can actually depend on the training sample, and such bounds are usually, if appropriately adjusted using prior knowledge, very much tighter than bounds that ignore such knowledge.

1.2.6 Models of classification noise

Let us introduce some common models for classification noise $P(t|y)$. The *Bernoulli* noise model (also *binomial* or *logit* noise model) is most commonly used for two-class classification. If $\pi(\boldsymbol{x}) = P(t = +1|\boldsymbol{x})$, the model treats t given π as $Bernoulli(\pi)$ variable, i.e.

$$P(t|\pi) = \pi^{(1+t)/2}(1-\pi)^{(1-t)/2}. \tag{1.8}$$

Instead of π, we model the *logit* $\log \pi/(1-\pi)$ as latent function:

$$y(\boldsymbol{x}) = logit(\boldsymbol{x}) = \log \frac{P(t=+1|\boldsymbol{x})}{P(t=-1|\boldsymbol{x})} \tag{1.9}$$

If $\sigma(u) = (1 + \exp(-u))^{-1}$ denotes the logistic function, we have $\pi(\boldsymbol{x}) = \sigma(-y(\boldsymbol{x}))$ and

$$P(t|y) = \sigma(ty) \tag{1.10}$$

The choice of the logit as the latent function is motivated by the theory of *nonparametric generalized linear models* (GLIM, see [GS94],[MN83]). These models impose a Gaussian process prior (as introduced in the next section) on the latent function $y(\cdot)$ and use noise distributions $P(t|y)$ from the exponential family, so that the Gaussian process classification model with Bernoulli noise is actually a special case of a GLIM. If $\mu(\boldsymbol{x}) = \mathrm{E}t|y(\boldsymbol{x})$, the connection between μ and the latent function is given by the *link function* $G(\mu(\boldsymbol{x})) = y(\boldsymbol{x})$. The most common link is to represent the *natural parameter* of the exponential family by $y(\cdot)$, referred to as *canonical link*. This choice has statistical as well as algorithmic advantages. As an example, the natural parameter of the Gaussian distribution is its mean, so that GP regression (as discussed in the next section) is a trivial GLIM. The natural parameter of the Bernoulli distribution is the *logit*. The standard method to fit a GLIM to data is the *Fisher scoring* method which, when applied to our model, is equivalent to the Laplace method discussed below. However, most applications of Fisher scoring in the statistical literature differ from the Bayesian GP classification in the treatment of hyperparameters. The former use classical statistical techniques like *cross validation* (as discussed in detail below) to adjust such parameters.

Another common model is *probit* noise (based on the cumulative distribution function (c.d.f.) of a Gaussian, instead of the logistic function, see [Nea97]). Probit noise can be generated very elegantly, as shown in [OW00],[OW99]. Consider adding stationary white noise $\xi(\boldsymbol{x})$ to the latent function $y(\boldsymbol{x})$,

and then thresholding the sum at zero, i.e. $P(t|y,\xi) = \Theta(t(y+\xi))$ where $\Theta(u) = I_{\{u\geq 0\}}$ is the *Heavisyde step function*. If $\xi(\boldsymbol{x})$ is a white zero-mean Gaussian process with variance σ^2, independent of $y(\boldsymbol{x})$, the induced noise model is

$$P(t|y) = \int P(t|y,\xi)P(\xi)\,d\xi = \Phi\left(\frac{ty}{\sigma}\right), \tag{1.11}$$

where Φ is the c.d.f. of $N(0,1)$. Moving from the noiseless case $P(t|y) = \Theta(ty)$ to the probit noise therefore amounts to simply add Gaussian noise to the latent variable. Adding Laplace or t distributed noise, i.e. distributions with heavier tails, results in more robust noise models. Finally, Opper and Winther [OW00],[OW99] discuss a *flip noise* model where $P(t|y,\xi) = \Theta(t\xi y)$, $\xi(\boldsymbol{x}) \in \{-1,+1\}$ is white and stationary. If $\kappa = P(\xi = +1)$, we have $P(t|y) = \kappa + (1-2\kappa)\Theta(ty)$.

1.3 Bayesian Gaussian processes

1.3.1 Construction of Gaussian processes

We follow [Wil97] and [WB98]. A *Gaussian process* is a collection of random variables, indexed by X, such that each joint distribution of finitely many of these variables is Gaussian. Such a process $y(\cdot)$ is completely determined by the mean function $\boldsymbol{x} \mapsto \mathrm{E}[y(\boldsymbol{x})]$ and the *covariance kernel* $K(\boldsymbol{x},\boldsymbol{x}') = \mathrm{E}[y(\boldsymbol{x})y(\boldsymbol{x}')]$. If we plan to use Gaussian processes as prior distributions, we can savely assume that their mean functions are 0, since knowing a priori any deviation from 0, it is easy to subtract this off. Note that assuming a zero-mean Gaussian process does *not* mean that we expect sample functions to be close to zero over wide ranges of X. It means no more and no less that, given no data and asked about our opinion where $y(\boldsymbol{x})$ might lie for a fixed $\boldsymbol{x}$, we have no reason to prefer positive over negative values or vice versa.

We can start from a positive definite symmetric kernel $K(\boldsymbol{x},\boldsymbol{x}')$. Since K is positive definite, for every finite ordered subset $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m) \subset X$ the covariance matrix $\boldsymbol{K} = (K(\boldsymbol{x}_i,\boldsymbol{x}_j))_{ij}$ is positive definite. We now assign $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m) \mapsto N(\boldsymbol{0},\boldsymbol{K})$. It is easy to see that these mappings are consistent with respect to marginalization and therefore, by the theorem indicated in 1.2.3, induce a unique Gaussian random process.

In the following, we will denote the covariance kernel of the prior Gaussian process by K, the covariance matrix evaluated over the training sample by $\boldsymbol{K}$. For a prediction point $\boldsymbol{x}_*$, we set $k_* = K(\boldsymbol{x}_*,\boldsymbol{x}_*)$ and $\boldsymbol{k}(\boldsymbol{x}_*) = (K(\boldsymbol{x}_i,\boldsymbol{x}_*))_i$.

1.3.2 Remarks on choice or design of kernels

When using a Gaussian process prior, all prior knowledge we have must be faithfully encoded in the covariance kernel. We will not go into modeling details here, but briefly mention some work in that direction. The basic idea behind most approaches is to use a more or less hierarchical design. Instead of choosing a fixed kernel K, we choose a parametric family $\{K(\boldsymbol{x}, \boldsymbol{x}'|\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}$ and a prior distribution over the *hyperparameter vector* $\boldsymbol{\theta}$, sometimes referred to as *hyperprior*. In principle we can continue and parameterize the hyperprior by hyper-hyperparameters, and so on. Note that this leads effectively to a non-Gaussian process prior, by integrating $\boldsymbol{\theta}$ out. Also, hierarchical design is very related to the way human experts attack modeling problems. If we in principle expect a certain property to hold for typical sample functions, but are not sure about some quantitative aspects of this property, we simply introduce a new hidden variable mapping these aspects to numerical values. Given a fixed value for this variable, the property should be easy to encode. Neal [Nea97] and MacKay [Mac97] give intuitive introductions to kernel design and show how to encode properties like smoothness, periodicity, trends and many more. Williams and Vivarelli [WV00] give kernels that encode degrees of mean-square differentiability which maps to expected roughness of sample functions. Another approach is to start from a parametric function family (see section 1.2.3) and choose priors on the weights in such a way that a Gaussian process prior over the function results. This sometimes works for finite architectures (i.e. having a finite number of weights), an example would be linear regression with fixed basis functions (see [Wil97]). However, in general the function distribution of more complex architectures will not be Gaussian if the weight priors are chosen in a simple way, but we often can achieve a Gaussian process in the limit of infinitely many weights. Such convergence can usually be proved by using the *central limit theorem*. Neal [Nea96] provides a thorough discussion of this technique, also showing up its limits. Williams [Wil96] calculated the kernel corresponding to radial basis function networks and multi-layer perceptrons, in the limit of an infinitely large hidden layer. Apart from the Adaptive Systems community, there are many other fields (like geology, meteorology) where Gaussian processes have been studied extensively, and there is a huge literature we will not try to review here. The reader might consider [Cre93] for spatial processes or the references given above for further bibliographies.

While most of the methods for choosing kernels mentioned above are rather adhoc, there are more principled approaches. Burges [Bur98a] shows how to incorporate certain invariance properties into a kernel, using ideas of differ-

ential geometry, see also [SSSV97]. Jaakkola and Haussler [JH98],[JHD99] propose the method of *Fisher kernels* and use generative models of the class densities over X to construct kernels. This widens the scope of kernel methods significantly. While the standard kernels usually assume X to be a real space of fixed, finite dimension, Fisher kernels have been used to discriminate between variable-length sequences, using Hidden Markov Models as generative models.

1.3.3 But why Gaussian?

There are several good reasons for preferring the family of Gaussian processes over other random processes, when choosing a prior distribution without using an underlying parametric model such as a parameterized function class. The most important one is maybe that using Gaussian process priors renders a subsequent Bayesian analysis tractable. Conditioning and marginalization of Gaussians gives Gaussians again, and simple matrix algebra suffices to compute the corresponding parameters. No other known parametric family of multivariate distributions has these closeness properties.

More justification comes from the principle of *maximum entropy* (see [Jay82],[CT91]). Suppose we use our knowledge to construct a covariance kernel K for the prior process. We now choose, among all random processes with zero mean and covariance K the one that is otherwise most uncertain. Uncertainty of a process can be measured by the *differential entropies* of its joint distributions. But it is well-known that among all distributions with a fixed covariance matrix the Gaussian maximizes the differential entropy. The maximum entropy principle therefore suggests choosing the Gaussian process with covariance K as prior distribution.

Finally, one might justify the use of Gaussian process priors by the fact that such prior distributions result if standard architectures like radial basis function networks or multi-layer perceptrons are blown up to infinite size. However, this argument is rather weak since convergence against a Gaussian process is only achieved if the priors on the architecture weights are chosen appropriately. Consider a multi-layer perceptron with one hidden layer. For the central limit theorem to be applicable, the variances of the hidden-to-output weights must converge to zero as the network grows. The final response of such a large network is the combination of a huge number of very small effects. Neal [Nea96] suggests and analyzes the use of weight priors that allows a finite number of the weights to be of the same order as the final response. More specific, under this prior the expected number of weights larger

than some threshold is bounded away from zero, even in the infinite network. Such a parameterization is maybe more reasonable if some of the network units are expected to develop feature detection capabilities. In this case, the limit of the network output is a random process which is not Gaussian.

1.3.4 Bayesian regression – an easy warmup

Even though the rest of this thesis deals with two-class classification only, we briefly develop the Bayesian analysis of the regression estimation problem with Gaussian noise, mainly because it is one of the rare special cases beyond linear regression where an exact Bayesian analysis is feasible.

Additive Gaussian noise is often justified by the presence of a large number of very small and widely independent random effects which add up to produce the final noise, and by the central limit theorem. Another argument can be formulated if we regard the corruption by noise as an information-theoretic communication channel (see [CT91]). The input y_i is corrupted by independent noise n_i to form the output $t_i = y_i + n_i$. Both input and noise are power-constrained in the sense that $\mathrm{E}[y_i^2]$ and $\mathrm{E}[n_i^2]$ are bounded. One can show that among all constrained noise distributions the Gaussian one gives rise to the smallest capacity of this channel. In other words, no other noise distribution leads to a smaller maximum mutual information between input y_i and output t_i. In this sense, Gaussian noise is the worst we can expect, and modeling unknown noise as Gaussian will at least satisfy the pessimists among the critiques. However, Gaussian noise is clearly inappropriate if we expect single random effects of the output's order of magnitude to occur (see also subsection 1.3.3), such effects result in so-called *outliers*. In these situations, distributions with larger tails like the t distribution or Huber distributions (see [Nea97],[Hub81]) should be used.

Let the noise be coloured Gaussian, i.e. $P(\boldsymbol{t}|\boldsymbol{y}) = N(\boldsymbol{y}, \boldsymbol{F}),\ \boldsymbol{F} = (F(\boldsymbol{x}_i, \boldsymbol{x}_j))_{ij}$ and F the covariance of the noise process. Then, $P(\boldsymbol{t}, \boldsymbol{y})$ is Gaussian with zero mean and a covariance made up of the blocks $\boldsymbol{K} + \boldsymbol{F}$ and three times $\boldsymbol{K}$. By conditioning on $\boldsymbol{t}$ we arrive at

$$P(\boldsymbol{y}|\boldsymbol{t}) = N\left(\boldsymbol{K}(\boldsymbol{K}+\boldsymbol{F})^{-1}\boldsymbol{t}, \boldsymbol{K}(\boldsymbol{K}+\boldsymbol{F})^{-1}\boldsymbol{F}\right) \tag{1.12}$$

$P(\boldsymbol{y}, y_*)$ is jointly Gaussian, and by conditioning we have $P(y_*|\boldsymbol{y}) = N(\boldsymbol{q}^t\boldsymbol{y}, \rho^2)$ with $\boldsymbol{q} = \boldsymbol{K}^{-1}\boldsymbol{k}(\boldsymbol{x}_*)$ and $\rho^2 = k_* - \boldsymbol{q}^t\boldsymbol{k}(\boldsymbol{x}_*)$. Furthermore,

$$P(y_*|\boldsymbol{t}) = \int P(y_*|\boldsymbol{y})P(\boldsymbol{y}|\boldsymbol{t})\, d\boldsymbol{y}. \tag{1.13}$$

Having a look at this equation, we see that y_* given $\boldsymbol{t}$ has the same distribution as $x+\boldsymbol{q}^t\boldsymbol{y}$ where $\boldsymbol{y} \sim P(\boldsymbol{y}|\boldsymbol{t})$ and $x \sim N(0, \rho^2)$, independent of $\boldsymbol{y}$. Therefore, given $\boldsymbol{t}$, y_* is normal with mean $\boldsymbol{q}^t\boldsymbol{K}(\boldsymbol{K}+\boldsymbol{F})^{-1}\boldsymbol{t} = \boldsymbol{k}(\boldsymbol{x}_*)^t(\boldsymbol{K}+\boldsymbol{F})^{-1}\boldsymbol{t}$ and variance $\rho^2 + \boldsymbol{q}^t\boldsymbol{K}(\boldsymbol{K}+\boldsymbol{F})^{-1}\boldsymbol{F}\boldsymbol{q} = k_* - \boldsymbol{k}(\boldsymbol{x}_*)^t(\boldsymbol{K}+\boldsymbol{F})^{-1}\boldsymbol{k}(\boldsymbol{x}_*)$. The predictive distribution $P(y_*|\boldsymbol{t})$ contitutes a complete solution of the prediction problem. Prediction with white noise corresponds to the special case $\boldsymbol{F} = \sigma^2\boldsymbol{I}$. See [Wil97],[WR96], [Ras96] for more details on Gaussian process regression with Gaussian noise. Neal [Nea97] discusses the regression problem with non-Gaussian, t distributed noise and suggests a *Markov Chain Monte Carlo* solution.

The Bayesian analysis is not complete at this point if we use a kernel family indexed by the hyperparameter vector $\boldsymbol{\theta}$. However, the remaining exact computations needed are not tractable. Therefore, we have to employ approximative techniques which are basically the same as in the classification case and will be described below.

1.3.5 Bayesian classification

As opposed to the regression case, the exact Bayesian analysis of two-class classification is infeasible since the integrals are not analytically tractable. A set of different techniques have been proposed to approximate the predictive distribution or moments thereof, based on *Laplace approximations* [WB98], *Markov chain Monte Carlo* [Nea97], variational techniques [Gib97] or *mean field* approximations [OW99]. We will describe the Laplace approximation in detail, since we need it in following arguments. In this subsection, we will assume that the hyperparameter vector $\boldsymbol{\theta}$ is fixed. We will specialize to the Bernoulli noise model (see subsection 1.2.6), although the presented techniques are more general.

The Laplace approximation is a general technique that applies to a wide range of positive densities. Let $p(\boldsymbol{x}) = \exp(-\Psi(\boldsymbol{x}))$ where Ψ is two times differentiable everywhere. Let $\hat{\boldsymbol{x}}$ be a local minimum of Ψ. We can expand Ψ around the local mode of $p(\boldsymbol{x})$ into a Taylor series. Dropping terms higher than second order, we have

$$\Psi(\boldsymbol{x}) \approx \Psi(\hat{\boldsymbol{x}}) + \frac{1}{2}(\boldsymbol{x}-\hat{\boldsymbol{x}})^t\boldsymbol{H}(\boldsymbol{x}-\hat{\boldsymbol{x}}), \tag{1.14}$$

where $\boldsymbol{H} = \nabla\nabla(-\log p(\boldsymbol{x}))$, evaluated at the mode $\hat{\boldsymbol{x}}$. Plugging this into $\exp(-\Psi(\boldsymbol{x}))$ and normalizing, we see that $p(\boldsymbol{x})$ can be approximated by the Gaussian with mean $\hat{\boldsymbol{x}}$ and covariance matrix $\boldsymbol{H}^{-1}$. At first sight, this approximation seems to be too inaccurate to ever be useful. Indeed, even if $\hat{\boldsymbol{x}}$ is

a global minimum of Ψ, using the Laplace method with very "non-Gaussian" $p(\boldsymbol{x})$ can cause severe problems. However, we should bear in mind that our goal is to approximate very peaky, low entropy distributions (namely, posterior distributions) in very high-dimensional spaces, where volume ratios tend to behave completely different compared to familiar low-dimensional settings. MacKay [Mac91],[Mac95] discusses some aspects of Gaussian approximations of distributions in high-dimensional spaces. From the practical point of view, we use Laplace approximations because they reduce complicated distributions to simple Gaussian ones and render previously intractable computations feasible. Apart from that, applying Laplace approximations in Bayesian analysis works surprisingly well in a large number of situations. Even though we cannot conclude that distributions are in general reasonably well represented by their Laplace approximations, these defiencies often don't seem to have a destructive effect on the final results.

The Laplace method is not the only way to approximate a high-dimensional density. It fails if Ψ is not differentiable everywhere or the second derivatives at the minimum $\hat{\boldsymbol{x}}$ don't exist. Finally, there might be other Gaussians representing $p(\boldsymbol{x})$ "better"[4] than the Laplace solution does. The latter uses only local information (namely, the curvature of the log probability manifold) concentrated at the mode and is usually suboptimal. Both problems are addressed by variational techniques like *variational free energy minimization* which will be discussed in detail below. The merits of the Laplace approximation lie in its simplicity and high computational speed, as compared to rival methods.

The negative log of the posterior $P(\boldsymbol{y}|\boldsymbol{t}) = P(\boldsymbol{t},\boldsymbol{y})/P(\boldsymbol{t})$ is, up to the constant $\log P(\boldsymbol{t})$, given by

$$\begin{aligned}\Psi = \Psi(\boldsymbol{y}) &= -\log P(\boldsymbol{y},\boldsymbol{t}) = -\log P(\boldsymbol{t}|\boldsymbol{y}) - \log N(\boldsymbol{y}|\boldsymbol{0},\boldsymbol{K}) \\ &= -\log P(\boldsymbol{t}|\boldsymbol{y}) + \frac{1}{2}\boldsymbol{y}^t\boldsymbol{K}^{-1}\boldsymbol{y} + \frac{1}{2}\log|\boldsymbol{K}| + \frac{n}{2}\log 2\pi.\end{aligned} \tag{1.15}$$

If $\hat{\boldsymbol{y}} = \operatorname{argmin}\Psi$ denotes the posterior mode, the Gaussian approximation rendered by the Laplace method is

$$P_a(\boldsymbol{y}|\boldsymbol{t}) = N\left(\hat{\boldsymbol{y}}, (\boldsymbol{W} + \boldsymbol{K}^{-1})^{-1}\right), \tag{1.16}$$

where $\boldsymbol{W} = \nabla\nabla(-\log P(\boldsymbol{t}|\boldsymbol{y}))$, evaluated at $\hat{\boldsymbol{y}}$, is a diagonal matrix. The predictive distribution is given by (1.13) and can be computed the same way as shown in subsection 1.3.4. We end up with

$$P_a(y_*|\boldsymbol{t}) = N\left(\boldsymbol{k}(\boldsymbol{x}_*)^t\boldsymbol{K}^{-1}\hat{\boldsymbol{y}}, k_* - \boldsymbol{k}(\boldsymbol{x}_*)^t\left(\boldsymbol{I} + \boldsymbol{W}\boldsymbol{K}\right)^{-1}\boldsymbol{W}\boldsymbol{k}(\boldsymbol{x}_*)\right). \tag{1.17}$$

[4] A suitable quality criterion will be introduced and justified later.

Given the model and the approximations we did so far, (1.17) represents the predictive distribution, i.e. the most complete solution to the classification problem. The MAP discriminant is $\boldsymbol{k}(\boldsymbol{x}_*)^t \boldsymbol{K}^{-1} \hat{\boldsymbol{y}}$, and the variance of $P_a(y_*|\boldsymbol{t})$ can be seen as error bar. Note that the mode of $P_a(y_*|\boldsymbol{t})$ is linearly related to the mode $\hat{\boldsymbol{y}}$ of $P_a(\boldsymbol{y}|\boldsymbol{t})$ which is also the true posterior mode. The same linear relation holds between the true predictive mean and the true posterior mean. However, since we cannot compute the true posterior mean (which is usually different from the posterior mode) in a tractable way, we need to apply some sort of approximation, such as the Laplace method, to be able to calculate an approximative predictive mean or mode[5]. Other approximations like for example mean field techniques (see [OW00]) make a Gaussian ansatz only for the predictive distribution, but the computation of the mean field parameters is typically much more involved than the minimization of (1.15). The variational method described in this thesis makes a Gaussian ansatz for the posterior $P(\boldsymbol{y}|\boldsymbol{t})$ too, but takes more information about the posterior into account than just its mode and the local curvature around it.

We have not said anything about how accurate we expect this Gaussian approximation of the posterior to be. From the convexity of (1.15) we know that the true posterior is unimodal, but it might be very skew so that any Gaussian approximation is poor in the sense that a sensible distance (such as the *relative entropy*, see subsection 3.1.2) between the posterior and any Gaussian approximation is large. Even if the posterior can be approximated well by a Gaussian, the choice made by the Laplace technique might be poor (see [BB97] for a toy example), this problem is attacked by the variational method in this thesis. However, what we are really interested in is a Gaussian approximation of the univariate predictive distribution $P(y_*|\boldsymbol{t})$, and we noted above that some methods (see [OW00]) indeed make a Gaussian ansatz only for this distribution. We have seen that our posterior ansatz leads to a Gaussian approximation of the predictive distribution and therefore is a stronger assumption, but the two are very related in the following sense. If $\boldsymbol{y} \sim P(\boldsymbol{y}|D)$ and $y_* \sim P(y_*|D)$, we have the linear relation $y_* = \boldsymbol{q}(\boldsymbol{x}_*)^t \boldsymbol{y} + r$ where $\boldsymbol{q}(\boldsymbol{x}_*) = \boldsymbol{K}^{-1} \boldsymbol{k}(\boldsymbol{x}_*)$ and r is independent Gaussian noise with variance $k_* - \boldsymbol{q}(\boldsymbol{x}_*)^t \boldsymbol{k}(\boldsymbol{x}_*)$. If y_* is Gaussian for all $\boldsymbol{x}_*$, so is $\boldsymbol{q}(\boldsymbol{x}_*)^t \boldsymbol{y}$. By Cramer's theorem (see [Fel71]), a multivariate variable is jointly Gaussian iff the projection of this variable onto every direction is Gaussian. While in general the $\boldsymbol{q}(\boldsymbol{x}_*)$ will not cover all directions, it is nevertheless clear that the Gaussian ansatz for the posterior is not much stronger than the assumption that all predictive distributions are Gaussian.

[5]The predictive mode is usually not even a function of the posterior mode, so we can't hope to determine it without approximations either.

Let us have a closer look at (1.17). If $\boldsymbol{W}$ is invertible, the variance can be written as $k_* - \boldsymbol{k}(\boldsymbol{x}_*)^t(\boldsymbol{W}^{-1} + \boldsymbol{K})^{-1}\boldsymbol{k}(\boldsymbol{x}_*)$ which has the same form as in the regression case, with $\boldsymbol{W}^{-1}$ replacing the noise covariance $\boldsymbol{E}$. The following applies to the Bernoulli noise model introduced above. We have $w_i = d^2(-\log \sigma(t_i y_i))/dy_i^2 = \sigma(y_i)(1 - \sigma(y_i))$. Figure 1.1 shows w_i plotted against y_i. How can we interpret that? Suppose that after we have found $\hat{\boldsymbol{y}}$ we observe that most of the components in $\hat{\boldsymbol{y}}$ are close to 0, i.e. most of the training patterns actually lie in the critical region between the classes. Then, the elements in the "noise" matrix $\boldsymbol{W}^{-1}$ will be fairly small and the predictive variance will be rather small. On the other hand, if most of the components in $\hat{\boldsymbol{y}}$ are rather large, i.e. most of the patterns either deep in the region of their class or judged as strong outliers by the discriminant, $\boldsymbol{W}^{-1}$ has large elements, and the predictive variance will be large, for many cases $\boldsymbol{x}_*$ close to the prior variance k_*. This is intuitively clear since patterns close to the decision boundary contain much more discriminative information (see discussion about Support Vectors below) than patterns far from the boundary, and a prediction based on more informative data should also be more confident than one based on a less informative sample.

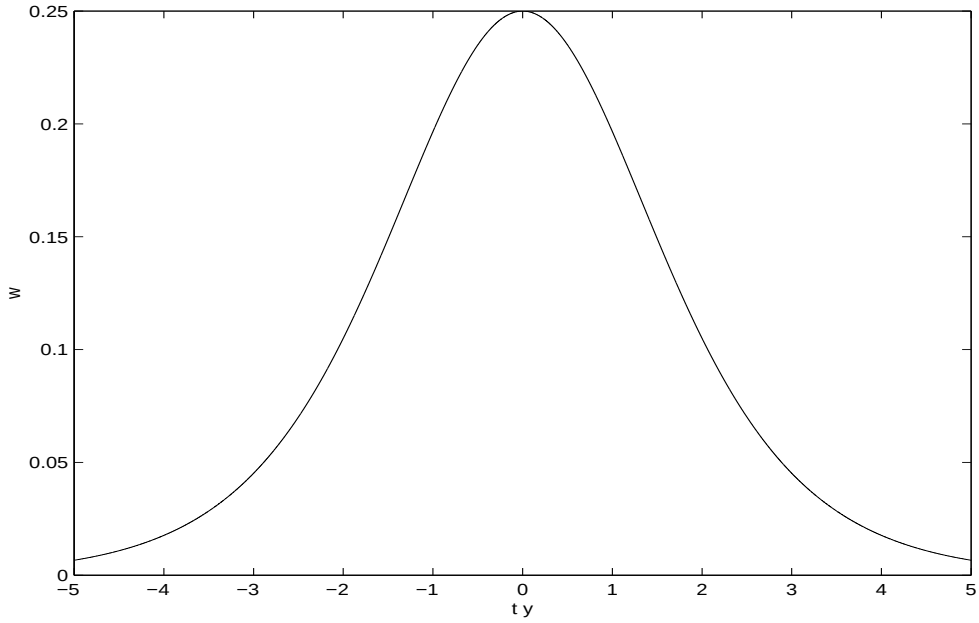

Figure 1.1: Component of $\boldsymbol{W}$ matrix against ty for the corresponding datapoint.

An approximation to the predictive distribution for the class label $P(t_*|\boldsymbol{t})$ can be computed by averaging the noise distribution $P(t_*|y_*)$ over the Gaussian

$P_a(y_*|\boldsymbol{t})$. The analytically intractable integral can easily be approximated by numerical techniques like *Gaussian quadrature* (see [PTVF92]). If the noise distribution $P(t|y)$ is strictly monotonic increasing in ty, there exist a unique threshold point a such that $P(+1|y=a) = P(-1|y=a)$, and the optimal discriminant based on $P_a(y_*|\boldsymbol{t})$ is $g(\boldsymbol{x}_*) = \mathrm{sgn}(\boldsymbol{k}(\boldsymbol{x}_*)^t \boldsymbol{K}^{-1}\hat{\boldsymbol{y}} - a)$. In case of Bernoulli noise, we have $a = 0$.

The mode $\hat{\boldsymbol{y}}$ can be found very efficiently using the *Newton-Raphson* algorithm, which is equivalent to the *Fisher scoring* method mentioned above since we use the canonical link (recall subsection 1.2.6). Note that if the negative logarithm of the noise distribution $P(t|y)$ is convex in y for both $t = \pm 1$, then the optimization problem of finding the mode is a convex one.

Finally, we have to deal with the hyperparameters $\boldsymbol{\theta}$. The exact method would be to compute the posterior $P(\boldsymbol{\theta}|D)$ and to average all quantities obtained so far by this distribution. Neither of these computations are tractable, so again approximations must be employed. Sophisticated Markov Chain Monte Carlo methods like the *hybrid MC* algorithm (see [Nea96]) can be used to approximate the integrals (see [WB98]). We can also search for the mode $\hat{\boldsymbol{\theta}}$ of the posterior $P(\boldsymbol{\theta}|D)$ and plug this parameter in, i.e. $P(y_*|D) \approx P(y_*|D, \hat{\boldsymbol{\theta}})$, etc. This *maximum a-posteriori (MAP)* approach can be justified in the limit of large datasets, since $P(\boldsymbol{\theta}|D)$ converges to a delta or point distribution for all reasonable models. We will discuss the MAP approach in more detail below.

1.4 Support Vector classification

We follow [Vap98], [Bur98b]. If not otherwise stated, all facts mentioned here can be found in these references. Vapnik [Vap98] gives a fascinating overview of the history of Statistical Learning Theory and SVM.

Support Vector classification has its origin in a perceptron learning algorithm developed by Vapnik and Chervonenkis. They argued, using bounds derived within the newly created field of Statistical Learning Theory, that choosing the hyperplane which seperates a (linearly seperable) sample with maximum *minimal margin*[6] results in a discriminant with low generalization error. The minimal margin of a seperating hyperplane $y(\boldsymbol{x}) = (\boldsymbol{\omega}, \boldsymbol{x}) + b,\ \|\boldsymbol{\omega}\| = 1$ with

[6] In earlier literature, the *minimal* margin is sometimes simply referred to as margin. However, the margin (as defined in more recent papers) is actually a random variable whose distribution seems to be closely related to generalization error. The minimal margin is the sample minimum of this variable.

respect to the sample D is defined as

$$\min_{i=1,\ldots,n} t_i y(\boldsymbol{x}_i) \tag{1.18}$$

where $y(\cdot)$ seperates the data without error. We will write $y_i = y(\boldsymbol{x}_i)$ in the sequel. Note that $t_i y_i$ denotes the perpendicular distance of $\boldsymbol{x}_i$ from the seperating plane. The reasoning behind this is that if the data points $\boldsymbol{x}_i$ are restricted to lie within a ball of radius $D/2$, then the VC dimension of the set of hyperplanes with minimal margin M is bounded above by $\lceil D^2/M^2 \rceil + 1$ (see [Vap98]). Much later, the *optimum margin hyperplanes* were generalized in two directions with the result being the Support Vector machine. First of all, nonlinear decision boundaries can be achieved by mapping the input vector $\boldsymbol{x}$ via Φ into a high, possibly infinite dimensional *feature space* with an inner product. Since in the original algorithm, input vectors occured only in inner products, the nonlinear version is simply obtained by replacing such $\boldsymbol{x}$ by $\Phi(\boldsymbol{x})$ and inner products in X by inner products in the feature space. This seems to be a bad idea since neither Φ nor feature space inner products can usually be computed efficiently, but a certain rich class of Φ mappings have associated positive semidefinite kernels K such that $K(\boldsymbol{x}, \boldsymbol{x}') = (\Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}'))$. As we will see below, every positive semidefinite kernel satisfying certain regularity conditions actually induces a feature space and a mapping Φ. We will now derive the SVC algorithm using this nonlinear generalization, and later show how it arises as the special case of a more general setting.

Instead of constraining $\boldsymbol{\omega}$ to unit length, we can equivalently fix the minimal margin to 1 (it must be positive if the data set is linearly seperable). Given a *canonical hyperplane* $(\boldsymbol{\omega}, b)$, i.e.

$$t_i y_i \geq 1, \ i = 1, \ldots, n \tag{1.19}$$

with equality for at least one point from each class, the same hyperplane with normal vector constrained to unit length has margin $1/\|\boldsymbol{\omega}\|$. The SVC problem therefore is to find a canonical hyperplane with minimal $(1/2)\|\boldsymbol{\omega}\|^2$.

The second generalization was to allow for errors, i.e. violations of (1.19), but penalize errors $[1 - t_i y_i]_+$ using a monotonically increasing function of these which is added to the criterion function. The fully generalized SVC problem is to find a discriminant $y(\boldsymbol{x}) = (\boldsymbol{\omega}, \Phi(\boldsymbol{x})) + b$ which minimizes the functional

$$C \sum_i [1 - t_i y_i]_+ + \frac{1}{2}\|\boldsymbol{\omega}\|^2. \tag{1.20}$$

This is a *quadratic programming* problem as can be shown by introducing nonnegative slack variables ξ_i, relaxing (1.19) to

$$t_i y_i \geq 1 - \xi_i, \; i = 1, \ldots, n \tag{1.21}$$

and minimizing $(1/2)\|\boldsymbol{\omega}\|^2 + C \sum_i \xi_i$ with respect to (1.21). Note that the sum is an upper bound on the number of training errors. Applying standard optimization techniques (see [Fle80]), we introduce nonnegative *Lagrange multipliers* $\alpha_i \geq 0$ to account for the constraints (1.21) and $\nu_i \geq 0$ for the nonnegativity of the slack variables. The multipliers are referred to as *dual variables*, as opposed to the *primal variables* $\boldsymbol{\omega}$ and b. The *Lagrangian* has a saddlepoint[7] at the solution of the original problem. Differentiating the Lagrangian with respect to the primal variables and equating to zero, we can express $\boldsymbol{\omega}$ in terms of the dual variables:

$$\boldsymbol{\omega} = \sum_{i=1}^{n} \alpha_i t_i \Phi(\boldsymbol{x}_i). \tag{1.22}$$

The primal variable b gives us the equality constraint

$$\sum_{i=1}^{n} \alpha_i t_i = 0. \tag{1.23}$$

Substituting these into the Lagrangian, we arrive at the *Wolfe dual* which depends only on the dual variables $\boldsymbol{\alpha}$:

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j (\Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j)). \tag{1.24}$$

However, the combination of feasibility conditions[8] and *Karush-Kuhn-Tucker (KKT) conditions* introduces new constraints $\alpha_i \leq C$ on the dual variables. The boundedness of the dual variables is a direct consequence of the linear penalty (slack variables enter linearly into the criterion), as is the fact that slack variables and their Lagrange multipliers do not appear in the Wolfe dual. By using the kernel K and the corresponding covariance matrix $\boldsymbol{K}$, the dual can be written as

$$\boldsymbol{\alpha}^t \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^t \boldsymbol{T} \boldsymbol{K} \boldsymbol{T} \boldsymbol{\alpha}, \tag{1.25}$$

[7]The solution is a minimum point of the Lagrangian w.r.t. the primal variables $\boldsymbol{\omega}, b$ and a maximum w.r.t. the dual variables.

[8]A vector is called *feasible* if it fulfils all the constraints of the problem. The set of all such vectors is called the *feasible set*. This set is convex in our situation.

where $T = \operatorname{diag} \boldsymbol{t}$. Substituting (1.22) into the hyperplane equation we have

$$y(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i t_i (\Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x})) + b = \boldsymbol{k}(\boldsymbol{x})^t \boldsymbol{T}\boldsymbol{\alpha} + b \tag{1.26}$$

where $\boldsymbol{k}(\boldsymbol{x}) = (K(\boldsymbol{x}_i, \boldsymbol{x}))_i$, as defined above.

The dual problem, namely to minimize (1.25) subject to the *box constraints* $\mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1}$ and (1.23) has a *unique* solution $\hat{\boldsymbol{\alpha}}$, being the only feasible vector $\boldsymbol{\alpha}$ that fulfils the KKT conditions

$$\begin{aligned} \alpha_i = 0 &\Longrightarrow t_i y_i \geq 1 \\ \alpha_i \in (0, C) &\Longrightarrow t_i y_i = 1 \\ \alpha_i = C &\Longrightarrow t_i y_i \leq 1 \end{aligned} \tag{1.27}$$

These conditions can be used to determine the bias parameter $\hat{b}$ at the solution. The input points $\boldsymbol{x}_i$ with $\hat{\alpha}_i > 0$ are called *Support Vectors*. Those with $\hat{\alpha}_i \in (0, C)$ are sometimes referred to as *essential Support Vectors*. The additive expansion $y(\cdot) = \boldsymbol{k}(\cdot)^t \boldsymbol{T}\hat{\boldsymbol{\alpha}}$ depends only on the Support Vectors, as does the training process of the Support Vector Machine. Note also that a falsely classified training point must have its dual variable at the upper bound C. On not too noisy datasets, the solution $\hat{\boldsymbol{\alpha}}$ is usually *sparse* in the sense that the ratio of the number of Support Vectors to the training set size is of the same order of magnitude as the Bayes error of the class distributions.

Chapter 2

A common framework

In this chapter we will introduce the *spline smoothing* framework (see [Wah90],[Wah98]) and show how Gaussian process and Support Vector classification arise as special cases. We then analyze the remaining difference between Support Vector classifiers and probabilistic kernel regression classifiers such as GPC and suggest an intuitive interpretation for the utility and the effects this difference might show in practice.

2.1 Spline smoothing methods

Throughout this section, we heavily rely on [Wah90] which contains details and references to the material presented here, unless otherwise stated. The framework given there has been applied to Support Vector machines in [Wah98], [WLZ99]. Other, albeit related unifications have been given for example by [OW99],[SSM98]. Jaakkola and Haussler [JH99] define the class of *kernel regression classifiers* and analyzes the relation between primal and dual criterion, using terms from *convex analysis*. The primal criterion for *probabilistic* kernel regression models is the log posterior of the hidden variables $\boldsymbol{y}$. He develops generic training and error estimation methods for this class.

2.1.1 Some facts from Hilbert space theory

We start with a *reproducing kernel Hilbert space (RKHS)* H_R, i.e. a Hilbert space such that all Dirac evaluation functionals are bounded. A Hilbert space is a Banach space with an inner product which is complete in the sense that

all Cauchy sequences converge against elements of the space (see [Hal57]). Without loss of generality we assume that H_R is a subspace of $L_2 = L_2(X)$, the space of square integrable functions. Note that L_2 itself is no RKHS. The Dirac functional $\delta_{\boldsymbol{a}},\ \boldsymbol{a} \in X$ maps f to $f(\boldsymbol{a})$. By the *Riesz representation theorem*, all these functionals have representers in H_R, and we can define the *reproducing kernel (RK)* R of H_R as

$$y(\boldsymbol{x}) = (R(\cdot, \boldsymbol{x}), y), \quad y \in H_R. \tag{2.1}$$

Note that $R(\boldsymbol{x}, \boldsymbol{x}') = (R(\cdot, \boldsymbol{x}), R(\cdot, \boldsymbol{x}'))$, i.e. R "reproduces" itself. The RK is always a symmetric, positive definite form. On the other hand, we can start with any positive definite form R and show that there is a *unique* RKHS with RK R, this fact is known as *Moore-Aronszajn theorem*. In fact, the inner product of this space is easily obtained from R, by a process that is simply the function space generalization of the diagonalization of a positive definite matrix. Suppose that the kernel is continuous and its square has finite trace:

$$\int \int R^2(\boldsymbol{x}, \boldsymbol{x}')\, d\boldsymbol{x} d\boldsymbol{x}' < \infty. \tag{2.2}$$

Then, by the *Mercer-Hilbert-Schmidt theorems*, there exists an orthonormal sequence of eigenfunctions $\{\Phi_\nu\}$ in L_2 and eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots$ with

$$\int R(\boldsymbol{x}, \boldsymbol{x}')\Phi_\nu(\boldsymbol{x}')\, d\boldsymbol{x}' = \lambda_\nu \Phi_\nu(\boldsymbol{x}) \tag{2.3}$$

and we can write the kernel as

$$R(\boldsymbol{x}, \boldsymbol{x}') = \sum_\nu \lambda_\nu \Phi_\nu(\boldsymbol{x})\Phi_\nu(\boldsymbol{x}'). \tag{2.4}$$

Let $\{y_\nu\}$ denote the spectrum of $y \in L_2$ w.r.t. the eigensystem, i.e. $y_\nu = \int y(\boldsymbol{x})\Phi_\nu(\boldsymbol{x})\, d\boldsymbol{x}$. We can define an inner product on L_2 by

$$(y, z)_R = \sum_\nu \frac{y_\nu z_\nu}{\lambda_\nu}, \tag{2.5}$$

and H_R can be characterized as

$$H_R = \{y \in L_2 \,|\, (y, y)_R < \infty\}. \tag{2.6}$$

By the well-known *Kolmogorov consistency theorem*, for every positive definite form there exists a unique Gaussian process with this form as covariance

kernel, so we can conclude that there is a simple 1-1 relationship between Gaussian processes and RKHS. It is interesting to note a subtilety hidden in this relationship, as pointed out by [Wah90], p.5. If we draw a sample function $y(\boldsymbol{x})$ from the Gaussian process defined by the kernel K, it is not contained in H_K with probability 1 (i.e. its norm, as defined by (2.5), is infinite). In light of this fact, it may be preferable to put into foreground the correspondence between H_K and the Hilbert space spanned by the process $y(\boldsymbol{x})$ (see [Wah90], p.14–15): these are isometrically isomorphic.

2.1.2 The general spline smoothing problem

A special case of the general spline smoothing problem is stated as follows. Let $\Phi_1, \ldots, \Phi_M$ be functions in L_2[1] such that the matrix $\boldsymbol{U} \in \mathbb{R}^{n \times M}$, $\boldsymbol{U}_{i\nu} = \Phi_\nu(\boldsymbol{x}_i)$ has full rank $M < n$. Let $g(t_i, y_i)$ be a loss function. Find the minimizer $y_\lambda \in \text{span}\{\Phi_\nu\} + H_R$ of the functional

$$\frac{1}{n} \sum_i g(t_i, y_i) + \lambda \|\boldsymbol{P} y(\cdot)\|_R^2, \tag{2.7}$$

where $\|\cdot\|_R$ denotes the norm in H_R, defined by (2.5), and $\boldsymbol{P}$ is the projection onto H_R. Kimeldorf and Wahba have shown that various very general data models result in smoothing problems of this kind if the kernel R is chosen in a proper way, see [Wah90] for details. The special version (2.7) can be found in [Wah98]. The reader might be curious what all this has to do with splines. If g is the squared-error loss and the inner product of H_R is constructed using differential operators up to a certain order, the minimizer of (2.7) is a polynomial spline (this has been shown by Schönberg [Sch64]). See [Wah90] for a derivation.

The *representer theorem* is of central importance for the spline smoothing problem because it transforms the optimization problem (2.7) into a simple finite-dimensional one. It states that y_λ has a representation of the form

$$y_\lambda(\boldsymbol{x}) = \sum_m d_m \Phi_m(\boldsymbol{x}) + \sum_i c_i R(\boldsymbol{x}_i, \boldsymbol{x}). \tag{2.8}$$

Let $\boldsymbol{R}$ denote the covariance matrix of R, evaluated at the training points, so $\boldsymbol{R}_{ij} = R(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Let $\boldsymbol{r}(\boldsymbol{x})$ be defined by $\boldsymbol{r}_i(\boldsymbol{x}) = R(\boldsymbol{x}, \boldsymbol{x}_i)$. Much more can be said about $\boldsymbol{d}$ and $\boldsymbol{c}$ if one specializes to a concrete loss function g. If g is strictly convex w.r.t. y_i, then (2.8) is a convex optimization problem. If

[1] The base functions should not be confused with the eigenfunctions of R.

g is differentiable w.r.t. y_i and $\boldsymbol{R}$ is nonsingular, we can show that $\boldsymbol{U}^t\boldsymbol{c} = 0$, so $\boldsymbol{c}$ must be a generalized divided difference of the Φ_m (see [Wah90], p.32). In the special case of quadratic loss functionals, i.e. $g(t_i, y_i) = (t_i - y_i)^2$, (2.8) can be explicitely solved for $\boldsymbol{c}$ and $\boldsymbol{d}$, and both are linear in the targets $\boldsymbol{t}$ (see [Wah90], p.11).

2.1.3 Gaussian process classification as spline smoothing problem

The correspondence between the two regimes can now easily be formulated. Choose $M = 0$ and $\boldsymbol{P}$ as identity on H_R, $g(t_i, y_i) = -\log P(t_i|y_i)$, so (2.7) becomes

$$-\frac{1}{n}\log P(\boldsymbol{t}|\boldsymbol{y}) + \lambda\|y(\cdot)\|_R^2, \tag{2.9}$$

where $\|\cdot\|_R$ is the norm of H_R. The representer theorem gives $y_\lambda(\boldsymbol{x}) = \sum_i c_i R(\boldsymbol{x}, \boldsymbol{x}_i)$. Substituting this into (2.9) results in the problem to find $\boldsymbol{c}$ which minimizes

$$-\log P(\boldsymbol{t}|\boldsymbol{y}) + (n\lambda)\boldsymbol{c}^t\boldsymbol{R}\boldsymbol{c}, \tag{2.10}$$

and if $\hat{\boldsymbol{c}}$ denotes the minimizer, the best prediction at $\boldsymbol{x}_*$ is $\hat{y}_* = \boldsymbol{r}(\boldsymbol{x}_*)^t\hat{\boldsymbol{c}}$.

If we set $K = (2n\lambda)^{-1}R$ and $\boldsymbol{y} = \boldsymbol{K}(2n\lambda)\boldsymbol{c} = \boldsymbol{R}\boldsymbol{c}$, we see that minimizing (2.10) w.r.t. $\boldsymbol{c}$ is equivalent to minimizing Ψ (1.15) w.r.t $\boldsymbol{y}$, and the predictions $\hat{y}_*$ are identical. $\boldsymbol{c}$ will also be referred to as *dual variables*, and the minimization of (2.10) as *dual problem*.

2.1.4 Support Vector classification as spline smoothing problem

We choose $M = 1$[2] and an arbitrary constant base function, say $\Phi_1 \equiv 1$. $\boldsymbol{P}$ maps y to $y - y(\boldsymbol{0})$, and the RKHS H_R in consideration only contains functions with $y(\boldsymbol{0}) = 0$. If $y \in H_R$, it can be expanded into the eigenfunctions of R:

$$y(\boldsymbol{x}) = \sum_\nu \omega_\nu \sqrt{\lambda_\nu}\Phi_\nu(\boldsymbol{x}) \tag{2.11}$$

[2]Chosing $M > 1$ leads to *semiparametric Support Vector Machines*. The choice $M = 0$ is discussed below.

If we define the map $\Phi(\boldsymbol{x}) = (\sqrt{\lambda_\nu}\Phi_\nu(\boldsymbol{x}))_\nu$ from X into l_2[3] and set $\boldsymbol{\omega} = (\omega_\nu)_\nu$, this can be written as an inner product in l_2:

$$y(\boldsymbol{x}) = (\boldsymbol{\omega}, \Phi(\boldsymbol{x})) \tag{2.12}$$

We have $R(\boldsymbol{x}, \boldsymbol{x}') = (\Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}'))$, $y_\nu = \int y(\boldsymbol{x})\Phi_\nu(\boldsymbol{x})\, d\boldsymbol{x} = \omega_\nu\sqrt{\lambda_\nu}$ and $\|y\|^2_{H_R} = \sum_\nu y_\nu^2/\lambda_\nu = \sum_\nu \omega_\nu^2 = \|\boldsymbol{\omega}\|^2$. Under these constraints, (2.7) becomes the familiar Support Vector optimization problem: Find $y(\boldsymbol{x}) = (\boldsymbol{\omega}, \Phi(\boldsymbol{x}))+b$ to minimize the functional

$$\frac{1}{n}\sum_i g(t_i, y(\boldsymbol{x}_i)) + \lambda\|\boldsymbol{\omega}\|^2. \tag{2.13}$$

See (for example) [Wah98] for a variety of loss functions g and their role as "computational proxies" to intractable ones. We only consider *Support Vector classification (SVC)* here, but the result applies to regression as well. We use the SVC loss function (1.7) and reparameterize the trade-off constant

$$C = \frac{1}{2n\lambda}, \tag{2.14}$$

which transforms (2.13) into (1.20).

2.1.5 The bias parameter

Strictly speaking, Support Vector machines are semiparametric models since they use a bias parameter b. Semiparametric Gaussian process models have been considered in the Statistics literature (see [GS94]), and for a single bias parameter the extension is straightforward. However, to render subsequent arguments more clear, we prefer to drop the bias parameter b and consider "nonparametric SVM". From a theoretical point of view, the bias parameter is redundant in a certain sense, as we will show. [Wah90] uses a parametric extension of the basic RKHS if the kernel in consideration has a null space. In this case, if the null space is finite dimensional, we can bridge the gap by a parametric extension whose base spans this space. However, we only consider kernels without null space here. If we retained b, we would have to place a prior on it. This would be zero-mean, since we have no reason to prefer one of the half-spaces for $y(\cdot)$. Furthermore, we don't expect very large values of b a priori since this results in discriminant functions predicting one

[3] l_2 is the Hilbert space of real sequences of "finite energy". Note that $\sum_\nu(\sqrt{\lambda_\nu}\Phi_\nu(\boldsymbol{x}))^2 = R(\boldsymbol{x}, \boldsymbol{x}) < \infty$.

of the targets with high probability almost everywhere.[4] Therefore, we can assume finite variance. The maximum entropy prior (see section 1.3.3) for a fixed variance is Gaussian. But placing a zero-mean Gaussian prior on b is equivalent to dropping the parametric extension of the model and adding a single positive hyperparameter to the kernel.[5]

Formally, the complete a-priori uncertainness about the value of the bias parameter would be a good reason to place an *noninformative prior* on b (see [Ber85]). However, such priors are usually *improper* in the sense that they cannot be normalized, and this is subject to severe criticism by Bayesians. We discuss such issues in subsection 3.2.3.

Remark One might argue that a fixed large constant added to the kernel comes closest to the effect of an improper prior on b. While this is true in theory, it leads to a very badly conditioned covariance matrix, and numerical errors would spoil all computations based on it.

Dropping the bias parameter might be, from the practical point of view, a bad idea. Some optimization techniques (like *Sequential Minimal Optimization (SMO)*) are very much faster with than without the equality constraint (1.23) introduced by the bias parameter. Afterwards, by employing the KKT conditions, the value of b is easily calculated. We argue that one can apply model selection algorithms for kernel families $\{K(\boldsymbol{x}, \boldsymbol{x}')|\boldsymbol{\theta}\}$ and SVM *without* bias parameter to SVM *with* a bias b as follows: We add a parameter v, constrained to be positive, to the kernel and drop b. Then, we run the model selection algorithm for SVMs without bias parameter which will give us values $(\hat{\boldsymbol{\theta}}, \hat{v})$ to plug into the kernel. Now, we simply discard $\hat{v}$ and train the original SVM with bias parameter using the kernel parameters $\hat{\boldsymbol{\theta}}$. This is motivated by looking at the modified kernel $K(\boldsymbol{x}, \boldsymbol{x}') + v$ which specifies our prior knowledge about the latent function. Now, this is the covariance of the sum of two random functions, one is specified by the original kernel K, and the other one is an unknown constant whose value is a zero-mean random variable of variance v. The latter function is exactly the bias about which we don't make any assumptions at all since the final value $\hat{v}$ of v is suggested by the data. If the data indicates that a substantial value of b is necessary for a good separation, then $\hat{v}$ will be large, and the effect of the formerly explicit bias is now implicitely realized by enforcing constraints on the posterior mode via the kernel[6]. Now, if we discard the v parameter and allow for

[4] Again, this assumptions does not apply for all possible kernels, but for kernels that encode some sort of locality by attaining very small values for distant points, it holds.

[5] Note that, if the bias parameter is dropped, the equality constraint in the dual SVM problem (see (1.23)) vanishes.

[6] This is maybe easier to see when looking at the *dual* variables $\boldsymbol{\alpha}$. Note that such an

an explicit bias parameter, we just shift the responsibility for creating the bias from the posterior mode into the explicit parameter. This argument is backed by empirical findings. On only one of the experimental datasets we observed a substantial value $\hat{v}$, as computed by the algorithm below for a machine without bias parameter and kernel $K + v$. Comparing this machine to one with explicit bias and kernel K, we found no difference in test error[7]. Training a machine without bias and kernel K gave worse performance, as expected.

2.1.6 The smoothing parameter

Consider the smoothing parameter λ in (2.7). While in the Bayesian literature this parameter is usually "hidden" in the kernel (i.e. one of the kernel hyperparameters) it is surely one of the most important free parameters since it controls the trade-off between the relevance of the loss and the prior (or penalty) term. Recall

$$C = \frac{1}{2n\lambda}. \tag{2.15}$$

If we replace the kernel R by $K = CR$, i.e. absorb the smoothing parameter into the kernel, the spline smoothing problem (2.7) is equivalent to the minimization of

$$\sum_i g(t_i, y_i) + \|\boldsymbol{P}y(\cdot)\|_K^2, \tag{2.16}$$

where $\|\cdot\|_K$ denotes the norm of the RKHS with reproducing kernel K.

Remark In practical applications the smoothing parameter should be dealt with separately from the other kernel parameters. C can become quite large and calculations based on $\boldsymbol{R}$ are much more stable than using the covariance matrix $\boldsymbol{K}$.

Let us investigate the effect of C in Gaussian process classification. A large C will render the likelihood term in (2.9) dominant, so the optimization will focus on achieving a high likelihood at the expense of a possibly very rough logit $y(\cdot)$. A too large C leads to the common problem of *overfitting* the data and results in poor performance on unseen data. If C is small, the prior

enforcement is not possible (and not necessary) when using an explicit bias, because of the equality constraint (1.23) introduced by the bias.

[7]However, the training of the second machine took substantially less time than the training of the machine without explicit bias, using SMO.

term in (2.9) dominates. Typical samples from the posterior will therefore largely be determined by our prior beliefs and be only slightly influenced by the new information contained in the data. The commonly used GP priors favor smooth slowly varying functions over rough ones, in accordance with the principle of *Occam's razor*. In this case, the posterior of the logit will very slowly vary around its mean which is nearly the same as the prior zero mean. The predictions will therefore remain unsure at most points. When C is absorbed into the covariance kernel, the variance of $y(\boldsymbol{x})$ at a fixed point $\boldsymbol{x}$ is proportional to C. Looking at the extreme cases of C, we see that this scaling property fits nicely into the interpretation. We will also refer to C as *variance parameter*.

We can also consider models with smoothing parameter C *and* kernel variance parameter (η, say). This seperation does not affect the MAP solution, i.e. the solutions for $C/\eta = const$ are identical, but it might play an important role in second-level inference of the hyperparameters. C can be interpreted as "noise variance". However, if the loss function corresponds to a noise model in the sense discussed in the next subsection, using a seperate smoothing parameter C is clearly a bad idea, since in general this destroys the correspondence: If $P(t|y)$ is a noise distribution, $P(t|y)^C$ cannot in general be normalized independently of y, and this can lead to severe problems, as discussed next.

2.1.7 Unnormalized loss functions

Having gone so far, can we actually formulate SVC as a probabilistic kernel regression classifier, such as GPC? The straightforward way fails because there is no noise distribution $P(t|y)$ such that $g(t, y) \propto -\log P(t|y)$ for the SVC loss (1.7). The normalization factor $Z(y)$ of the noise distributions $\exp(-g(t, y))/Z(y)$ depends on y. Using the normalized noise distributions attributes to change the loss to $g(t, y) + \log Z(y)$, and the corresponding negative log posterior is different from the criterion (1.20) relevant for SVC. One might try to incorporate the normalization constants into the prior, i.e. using $P(\boldsymbol{y}) \propto \prod_i Z(y_i) N(\boldsymbol{y}|0, \boldsymbol{K})$. However, there will in general be no stochastic process having these distributions as marginals (recall that the distributions have to be consistent w.r.t. marginalization – see section 1.2.3). We will refer to losses which have no associated noise distribution as *unnormalized loss functions*.

There are several possibilities to deal with kernel regression classifiers (or spline smoothing classifiers) that have unnormalized loss functions. We can approach such classifiers from the discriminative paradigm (see section 1.2.4),

accepting the fact that they are not proper generative models for the data. Wahba [Wah98],[WLZ99] recommends minimizing the GCKL (1.6) or proxies thereof. This works in two stages. For fixed kernel parameters $\boldsymbol{\theta}$ we minimize a penalized sample version of the GCKL, also referred to as *penalized likelihood*. This criterion is simply (1.20). Penalization is necessary for capacity control, since the infinite number of degrees of freedom of a random process cannot be determined by a finite sample so as to achieve good generalization. Although simple penalties that enforce smoothness or slow variation are most commonly used, we argue that the selection of the penalty kernel should be guided by prior knowledge about the expected nature of the hidden function.

Quite recently it has been shown that *boosting* techniques (see [SS98],[FS96],[Bre97], [Bre96]) like discrete or real *AdaBoost* can be regarded as stepwise optimization of a criterion that is very similar to the GCKL with the loss function $\exp(-ty)$. This function is a differentiable upper bound on the misclassification loss and behaves quite similarly to the negative log likelihood of the Bernoulli model around 0.[8] We will return to this discussion in subsection 2.2.1. The most common technique to train Support Vector classifiers is to choose the kernel parameters $\boldsymbol{\theta}$ by cross validation, using the *rotation estimator* of the generalization error (see [DGL96], chapter 31). Since the SVC loss is an upper bound on the misclassification loss, this method is quite similar to the GCKL approach mentioned above. From this discussion, one might gain the impression that loss functions like SVC or AdaBoost loss are chosen primarily out of algorithmic convenience, their more or less sole theoretical justification being the fact that they are tight upper bounds on the misclassification loss in the critical region. Recent studies show that this is far from true. While the generalization error is the obvious criterion to minimize, optimization of *estimators* of this quantity, like the rotation estimate, even if they are *unbiased*, usually neglects a lot of information provided by the sample. For example, the rotation estimate is only concerned about the fraction of errors among the left-out examples (averaged over a rotation), it completely ignores information about the certainty with which an example is correctly or incorrectly labelled. Recent discriminative learning algorithms try to incorporate this information by estimating the *margin distribution* or characteristics thereof, and then optimize free parameters with regard to these estimates. We will take up some aspects of this idea below, but a thorough discussion would lead us far out of the scope of this work. Useful references are [SFBL98],[MBB98]. Another possibility for choosing $\boldsymbol{\theta}$ is to minimize PAC bounds on the generalization error w.r.t. the kernel parameters. However, since the bounds are usually far from the true error (as

[8] The two functions are equivalent up to second order in a Taylor expansion around 0.

estimated on toy problems), there is only a vague hope that the minimum points of the generalization error and of the bound are close. The reason for the looseness of such bounds lies in their worst-case nature (see [HKS94] for a discussion).

Jaakkola and Haussler [JH99] define the class of *kernel regression classifiers* and shows how to obtain such classifiers from probabilistic generative models. He minimizes a computationally cheap upper bound on the leave-one-out rotation estimator to choose free parameters. He suggests a sequential Bayesian algorithm to approximate the posterior which is applicable to any *probabilistic* kernel regression classifier, like GPC or (parametric) logistic regression.

Opper and Winther [OW99] also discuss the problem of defining a noise model for Support Vector classification. They start from the probit noise model (see subsection 1.2.6) $P(t|y,\xi) = \Theta(t(y+\xi))$ where Θ is the Heavyside step, and the additive noise ξ is Laplace distributed, i.e. $P(\xi) = C\exp(-C\xi),\ \xi \geq 0$. Maximizing the joint $P(\boldsymbol{t}, \boldsymbol{y}, \boldsymbol{\xi})$ leads to a trivial solution, however. This can be fixed "ad hoc" by introducing a margin, i.e. replacing the noise $P(t|y,\xi)$ by $\Theta(t(y+\xi)-1)$. But this does not induce a normalized distribution $P(t|y)$. In fact, the induced "noise distribution" is $\exp(-C[1-ty]_+)$, whose negative log is the SVC loss defined in (1.7). Opper and Winther suggest to base Bayesian analysis for SVC on the related noise distribution $P(t|y,\xi) = \Theta(t(y+\xi))$ where $\xi \sim (C/2)\exp(-C|\xi|)$[9]. This distribution induces

$$P(t|y) = \Theta(ty) - \mathrm{sgn}(ty)\frac{1}{2}\exp(-C|ty|). \tag{2.17}$$

Another work we are aware of is [Kwo99]. This contains an approach to apply the evidence framework (discussed below) to Support Vector classification, but it deals with an unnormalized noise distribution. In fact, the author claims that the noise distribution is normalized by showing that $\int_0^\infty \exp(-\xi_i)d\xi_i = 1$ where $\xi_i = [1 - t_i y_i]_+$ (which is of course true), but this tells us nothing about the value of $\exp([1-y_i]_+) + \exp([1+y_i]_+)$ which is not constant w.r.t. y_i. Applying Bayesian techniques to unnormalized distributions usually leads to systematic errors, as discussed below.

The very recent work [JMJ99] describes Support Vector classification as a special instance of a general *minimum relative entropy (MRE) discrimination* framework. The soft margin constraints result from a prior distribution imposed on the margin widths at every datapoint, and the form of this prior determines the penalty for margin violations. The analogy is not perfect,

[9]Note that we do *not* impose a positivity constraint on ξ. Doing so would lead to a trivial solution of the training problem, as remarked above.

since the MRE criterion differs from the dual criterion of SVC by a potential term which prevents any of the dual variables to attain the maximum value C (or 1 if the variance parameter has been absorbed into the kernel), but the MRE criterion can be seen as a good approximation of the SVC dual. The MRE discrimination framework is promising since it attempts a link between Bayesian techniques (which are generally motivated using *maximum entropy (ME)* or MRE settings) and discriminative methods, and this might lead to important synergetic effects.

Even if we are not able to find a proper generative process model in which the SVC solution plays the role of the posterior mode, we can consider this solution as an approximation to the mode in a related Gaussian process classification model. The rationale behind this is that the Support Vector discriminant has some apparent advantages over general Gaussian process predictors, namely its sparsity which allows fast evaluation of the discriminant on test data and the existence of very efficient algorithms to solve the training problem. How should the related Gaussian process model be chosen? First it seems reasonable to employ the same kernel family as for the Support Vector machine. Second we should try to match the unnormalized SVC loss as closely as possible by the negative log of the related noise model.

Let us collect some options we have at this point. The normalization factor of the SVC loss is $Z(y) = \exp(-C[1-y]_+) + \exp(-C[1+y]_+)$. For not too small C this is close to 1 except inside the margin region and around its maximum points $y = \pm 1$. Its maximum is $1/\sigma(2C) < 2$, and it has the value $2\exp(-C)$ at its minimum point 0 (for $C > \log 2$). We therefore expect the *normalized SVC loss* $C[1-ty]_+ + \log Z(y)$ to be a good approximation to the unnormalized one, at least for C in a certain range[10]. This corresponds to the noise distribution

$$P(t|y) = Z(y)^{-1} \exp\left(-C[1-ty]_+\right). \tag{2.18}$$

Opper and Winther [OW99] suggest the related noise model (2.17) which we will call *probit noise*. Of course, we can also employ the GLIM Gaussian process model with Bernoulli noise. Figure 2.1 gives a comparative plot. For all but very small values of C the normalized SVC loss is closest to the unnormalized one almost everywhere.

Viewing the SVC optimization as an approximation to first-level inference in a related probabilistic kernel regression model is not completely satisfying. However, this approximation is not the only one that is used to make full Bayesian analysis computationally tractable. Very recently, a generative

[10] We will return to this point below.

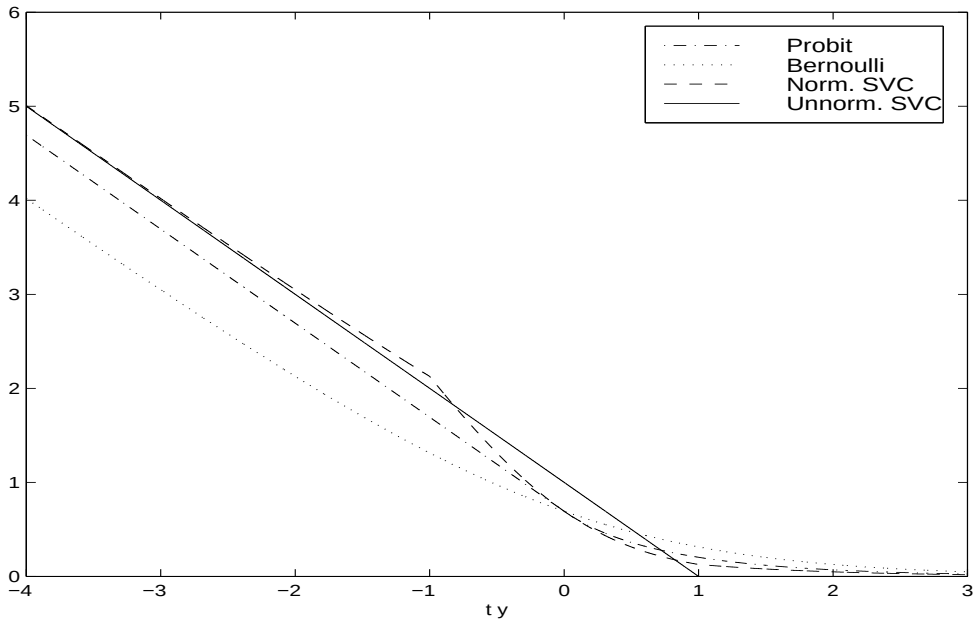

Figure 2.1: Several normalized loss functions versus the unnormalized SVC loss. Here, $C = 1$.

model for Support Vector classification has been proposed. We will discuss this model and some implications in the next subsection. Although the approach is very promising, there remain some consistency problems, and we finally decided not to base our algorithms on this model. Once these problems are fixed, it should be easy to rederive the algorithms within the new model.

2.1.8 A generative model for SVC

As shown above, the simple model design applied to kernel regression classifiers with normalized loss functions cannot be transferred to the unnormalized case. The standard sampling mechanism employed there is to sample some input points from an input distribution and independently a latent function from a process prior. The targets are sampled afterwards at the input points from the noise distribution and are conditionally independent, given the input points and the latent function. Sollich [Sol00] (see also [Sol99]) exploits the idea of nonstandard sampling mechanisms to construct a generative model for Support Vector classification, as follows. Let $Q(\boldsymbol{x})$ be a distribution from which examples $\boldsymbol{x} \in X$ are sampled. Let

$Q(t|\boldsymbol{x}, y(\cdot)) = \kappa(C)\exp(-Cg(ty(\boldsymbol{x})))$ and

$$Z(y(\boldsymbol{x})) = Q(t=1|\boldsymbol{x}, y(\cdot)) + Q(t=-1|\boldsymbol{x}, y(\cdot)). \tag{2.19}$$

The normalization constant is chosen such that $Z(y(\boldsymbol{x})) \leq 1$, $\kappa(C) = \sigma(2C)$ is the tightest choice. Our problem is that Z is not constant, therefore the "likelihood" $Q(D|y(\cdot)) = \prod_i (Q(t_i|\boldsymbol{x}_i, y(\cdot))Q(\boldsymbol{x}_i))$ is not normalized. But we have

$$\begin{aligned}
&\int Q(D|y(\cdot))P(y(\cdot))dDdy(\cdot) \\
&= \int \left(\prod_i \int Q(t_i|\boldsymbol{x}_i, y(\cdot))Q(\boldsymbol{x}_i)\, d(t_i, \boldsymbol{x}_i) \right) P(y(\cdot))\, dy(\cdot) \\
&= \int \left(\prod_i \int Z(y(\boldsymbol{x}_i))Q(\boldsymbol{x}_i)\, d\boldsymbol{x}_i \right) P(y(\cdot))\, dy(\cdot) \\
&= \int \left(\int Z(y(\boldsymbol{x}))Q(\boldsymbol{x})\, d\boldsymbol{x} \right)^n P(y(\cdot))\, dy(\cdot) = \mathcal{N},
\end{aligned} \tag{2.20}$$

and

$$P(D, y(\cdot)) = Q(D|y(\cdot))Q(y(\cdot))/\mathcal{N} \tag{2.21}$$

is properly normalized. Furthermore, the form of $\mathcal{N}$ already suggests how to sample from this joint distribution: First draw the hidden function $y(\cdot)$ from the Gaussian process prior $P(y(\cdot))$. Then, for each i, sample $\boldsymbol{x}_i \sim Q(\boldsymbol{x}_i)$ and draw $z_i \in (-1, +1, 0)$ independently, using the distribution $(Q(-1|\boldsymbol{x}_i, y(\cdot)), Q(+1|\boldsymbol{x}_i, y(\cdot)), 1 - Z(y(\boldsymbol{x}_i))$. If $z_i = 0$, restart the whole procedure, i.e. with sampling a new hidden function $y(\cdot)$. Otherwise, assign $t_i = z_i$. Let A denote the event that D is generated in the first run. Since $P(D, y(\cdot)) = P(D, y(\cdot)|\neg A)$, we have $P(D, y(\cdot)) = P(D, y(\cdot)|A)$, i.e. $(D, y(\cdot))$ is independent of A. Thus, $P(D, y(\cdot)) = P(D, y(\cdot), A)/P(A) = Q(D|y(\cdot))Q(y(\cdot))/\mathcal{N}$ since $P(A) = \mathcal{N}$. The number of trials we need until a $(D, y(\cdot))$ is produced, is geometrically distributed with success probability $P(A) = \mathcal{N}$, i.e. has mean $(1 - \mathcal{N})/\mathcal{N}$ and variance $(1 - \mathcal{N})/\mathcal{N}^2$, so the sampling terminates in finite expected time.

Let us have a closer look at the variables related to this model. The sampling mechanism induces *effective* distributions on the variables that finally "survive", which are different from the distributions used in each run. The effective latent function prior is $\propto P(y(\cdot))N(y(\cdot))^n$ where

$$N(y(\cdot)) = \int Z(y(\boldsymbol{x}))Q(\boldsymbol{x})\, d\boldsymbol{x}. \tag{2.22}$$

For $C > \ln 2$, $Z(y(\boldsymbol{x}))$ is smaller in the margin region $y(\boldsymbol{x}) \in (-1, 1)$ than elsewhere (see figure 2.2). Thus, functions with many values inside this gap (in the regions of significant $Q(\boldsymbol{x})$ mass) are discouraged by a low weighting in the effective prior. In terms of the sampling mechanism, they are less likely to survive in generating a data set of size n. Note that the effective prior depends on the dataset size n. This is presently the major problem with this model, as we will see below. Although the effective prior is non-Gaussian, the prior $P(y(\cdot))$ used for construction remains Gaussian. Therefore, conditioned on fixed hyperparameter values $\boldsymbol{\theta}$, first-level approximate inference of $\boldsymbol{y}$ works as in the case of models with normalized loss, the normalization factor $\mathcal{N}$ can be ignored. Second-level inference of the hyperparameters will be more complicated though, since the normalization factor has to be taken into account. Sollich [Sol00] suggests to estimate this factor by replacing (2.22) by the sample average. The estimate of $\mathcal{N}$ is a Gaussian expectation (over the prior) and can be approximated using Monte Carlo chaining.

The effective distribution of the inputs is $\propto Q(\boldsymbol{x})\nu(y(\boldsymbol{x}))$. This is somewhat surprising, but there might be an interesting interpretation for this effect. The present practice with Gaussian process classification models is to widely ignore the distribution of the inputs, i.e. implicitely assume a rather broad input distribution, uniform over the range of interest. However, if we start from a generative viewpoint, i.e. try to model the two classes seperately, we typically end up with a roughly bimodal distribution of the inputs. The reweighting by $\nu(y(\boldsymbol{x}))$ could be interpreted as a "correction" of the flat input distribution $Q(\boldsymbol{x})$. It leads to a dependence of the input points on the latent function. Although this dependence is obviously given in practice, it is completely ignored so far in common Gaussian process models by assuming independence of $y(\cdot)$ and the input points of the sample. The dependence implies that input points convey information about the latent function even if the corresponding target values are completely unknown. This information is the basis of any unsupervised learning method, but has been widely ignored so far in common Gaussian process models. This idea has been pointed out to us by Amos Storkey. A more detailed exploration is out of the scope of this thesis.

For $C < \ln 2$, the explanation given above fails. However, such small values of C *are* not plausible, a brief look at the induced loss $Cg(t, y)$ (which is much too flat for small C) reveals this fact. It is not yet clear if using a seperate smoothing parameter C together with the loss function (apart from a kernel variance parameter usually included in $\boldsymbol{\theta}$, see discussion in subsection (2.1.6)) has any significant advantage at all over just fixing it a-priori to a plausible value. For example, in the case of SVC loss we might fix $C = 1$, which

guarantees that the loss has the same asymptotic behaviour for large $|y|$ as the Bernoulli loss.

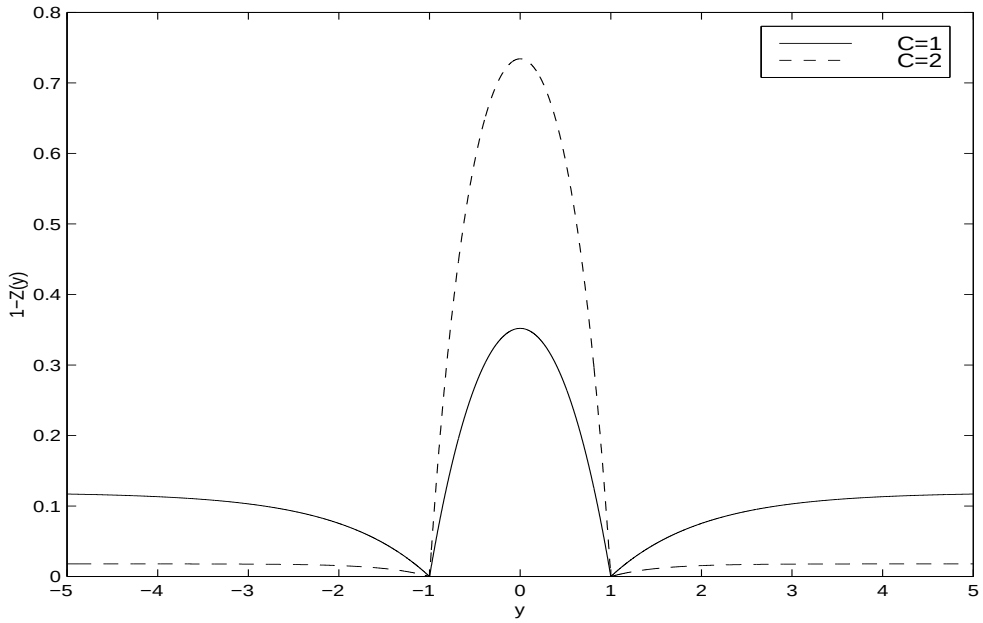

Figure 2.2: Shows $1-Z(y_i)$ against y_i where $Z(y_i)$ is the normalization factor of the single-case likelihood. A point $\boldsymbol{x}_i$ is rejected with probability $1 - Z(y(\boldsymbol{x}_i))$ during sampling.

The generative model described above is very recent and has, in its present form, some apparent problems. If the input distribution $Q(\boldsymbol{x})$ is unknown (which is the case in all but toy problems), $N(y(\cdot))$ and $\mathcal{N}$ cannot be computed exactly. Sollich [Sol00] suggests to estimate $\mathcal{N}$ using the given training sample. Pretending that the input points $\boldsymbol{x}_i$ are distributed as $Q(\boldsymbol{x})$, we arrive at an intractable Gaussian integral as estimator for $\mathcal{N}$ which can be attacked by common approximations or sampling techniques. However, the points $\boldsymbol{x}_i$ are *not* $Q(\boldsymbol{x})$ distributed, as discussed above. It is not clear if the averaging over the prior $Q(y(\cdot))$ washes out the bias of this estimator. A more severe problem with the model is, however, that it does not marginalize correctly, in the following sense. Suppose we are given a training set D consisting of n cases. Can we really be sure that D has been generated as described above, or might it be the case that originally a larger set $(D, \tilde{D})$ of $n + m$ cases has been drawn from which we only see n elements? Within a probabilistic kernel regression model, we need not to be concerned about that because $\int P(D, \tilde{D})\, d\tilde{D} = P(D)$, i.e. the marginal likelihood of the data is *consistent w.r.t. marginalization*. Indeed, this consistency is fundamentally

required in Bayesian analysis. Consistent inference in a world "full of hidden variables" can only work properly if we can marginalize out everything we are not interested in. Within the model described above we have

$$\int P(D, \tilde{D}|\boldsymbol{\theta})\, d\tilde{D} = \int P(D, y(\cdot)|\boldsymbol{\theta}) \frac{N(y(\cdot))^m \mathcal{N}_n}{\mathcal{N}_{n+m}}\, dy(\cdot) \neq P(D|\boldsymbol{\theta}) \qquad (2.23)$$

in general. For large m, the factor $N(y(\cdot))^m \mathcal{N}_n / \mathcal{N}_{n+m}$ is expected to vary by many orders of magnitude as we run over different $y(\cdot)$, therefore equality in (2.23) cannot be expected. This does not mean that $P(D|\boldsymbol{\theta})$ cannot be used for model selection. If n is reasonably large, the mass of $P(D, y(\cdot)|\boldsymbol{\theta})$ is usually concentrated in a small area. If the factor $N(y(\cdot))^m \mathcal{N}_n / \mathcal{N}_{n+m}$ is ≈ 1 in this area, equality in (2.23) might hold at least approximately. However, more work has to be done to come up with justifications like this.

2.2 Continuous reweighting as intuitive interpretation of the SVC model

Can we gain insight into the apparent difference between SVC and probabilistic kernel regression classifiers? A sensible approach is to analyze the behaviour of the two model classes on some data. The goal of a classifier built upon a generative probabilistic model is to maximize a trade-off between the likelihood of the data and a prior probability matching the hidden function against expectations we have a-priori. As opposed to that, SVC focuses on maximizing the minimal margin over the sample which relates the algorithm to a class of *boosting* techniques. Indeed, it has been empirically observed that the performances of large margin machines and boosting classifiers are often strongly correlated (see [MOR98]). We will try to make this relationship more explicit. However, our arguments will just cumulate into an intuitive interpretation of the SVC model (which might prove useful to some and pointless to others), *not* into a theorem and a proof.

2.2.1 Boosting and the margin distribution

The *margin* is defined as the random variable $ty(\boldsymbol{x})$, for a fixed function $y(\cdot)$. Note the "sources of randomness" involved here, namely the input variable $\boldsymbol{x}$ and the noisy target t conditioned on $y(\boldsymbol{x})$. Replacing the unknown data distribution by the empirical distribution of the sample, we can estimate the

margin distribution by the *sample margin distribution*

$$P_D\left[ty(\boldsymbol{x}) \leq \theta\right] = \frac{1}{n}\sum_{i=1}^{n} I_{\{t_i y(\mathbf{x}_i) \leq \theta\}}. \tag{2.24}$$

For fixed θ, this gives us the fraction of examples which are not discriminated with margin θ by $y(\cdot)$. Now, for certain hypothesis spaces theorems have been proved that bound the generalization error $P[ty(\boldsymbol{x}) \leq 0]$ by the sum of $P_D[ty(\boldsymbol{x}) \leq \theta]$ and a complexity penalty term that is monotonically decreasing in θ. This has been shown for example for two-layer neural networks [Bar98] and convex committees (including these generated by boosting techniques) [SFBL98]. Shawe-Taylor et al [STBWA96] proved a general theorem of this kind which gives nontrivial bounds for all hypothesis classes whose *fat-shattering dimension* scales favorably with the minimum margin attained on typical datasets. The fat-shattering dimension is a scale-sensitive generalization of the VC dimension. The class of Support Vector machines is covered by this theorem. Anthony [Ant97] gives a tutorial introduction into this branch of PAC theory.

The history of boosting algorithms is a fascinating one. The boosting problem in *Computational Learning Theory (COLT)* asks whether there is a polynomial time algorithm that is able to *strongly learn* sets using *weak learning algorithms* as oracles. For a detailed discussion see [KV94]. The intuitive distinction about a weak and a strong learner is that the former one is able, after having learned a polynomial-size training set, to predict future cases slightly better than uninformed pure random choices would do, while the latter performs with any desired accuracy and confidence (these terms in the context of the PAC framework are discussed in subsection 1.2.4). Indeed, Schapire proposed a boosting algorithm, thereby showing that the classes of weak and strong learners are identical. The algorithm is complicated and hardly suitable for practical use, see [KV94] for a textbook presentation. Shortly after, a much simpler algorithm was discovered by Freund (see [FS96]) and termed *AdaBoost*. This algorithm is maybe one of the most important discoveries in recent COLT and, apart from solving the theoretical boosting problem in a very elegant way, has an immense practical value. AdaBoost operates on sets of relatively poor learning methods having large bias or variance or both, and usually creates a highly competitive committee using these weak learners as experts. It does so with negligible time and memory requirements, apart from training the weak learners.

We will not give a detailed description of AdaBoost here (see [FS96], [SS98],[FHT98]), but only sketch the principal idea. The algorithm maintains a distribution over the sample which is uniform at the beginning. A

weak learner is trained on a copy of the sample which is *reweighted* according to this distribution. This can be done by feeding a resampled version of the original dataset into the unmodified training algorithm of the weak learner, or, more efficiently, to modify the training algorithm to account for the weights. The trained classifier is then tested on the sample. The learner is multiplied by a coefficient which depends on the test error, and added to the committee built up so far. The weights corresponding to examples on which the weak learner failed are multiplied by a factor > 1 which again depends on the test error. The iteration is finished by renormalizing the weight distribution and the linear coefficients of the committee. The idea behind the reweighting of the sample is to focus the next weak learner on examples that have been poorly classified so far by the committee. AdaBoost has been generalized to real-valued, *confidence-rated* weak predictors in [SS98]. This algorithm is referred to as *real AdaBoost*.

AdaBoost was originally designed to drive the error of the committee on the original sample to zero as quickly as possible. This was based on the assumption that a set of small committees has a lower complexity (e.g. VC dimension) than a set of larger ones, and on VC bounds on the generalization error in terms of training error plus complexity penalty. Ironically (and luckily), the algorithm performs much better than ever could be expected on this basis. It was soon discovered empirically that AdaBoost usually continues to drive the generalization error down, by adding new components to the committee, *after* the training error has reached the zero level. Thus, committees with an order of magnitude larger number of components achieve a smaller generalization error than small ones, even though *both* have zero training error. An explanation was given in [SFBL98], showing that the number of components in a convex committee is not a relevant complexity measure, but the scale θ for the margin is. An algorithm required to minimize the cost function $P_D[ty(\boldsymbol{x}) \leq \theta]$ for large θ is not able to make fine distinctions between the functions in a hypothesis class, and the effective complexity of the class is reduced. They further showed that algorithms like AdaBoost implicitely minimize criteria like (2.24). Mason et al [MBB98] generalized this result to more general margin cost functions, thereby showing how the cost functions of different boosting algorithms might influence the generalization error of the committees they produce. There seems to be a trade-off between cost functions close to the misclassification loss $I_{\{yf(\mathbf{x})\leq 0\}}$ (high resolution, therefore high effective complexity) and larger margin cost functions.

2.2.2 Additive logistic regression

Friedman et al [FHT98] connect boosting algorithms related to AdaBoost with procedures to fit *additive logistic regression* models. They show that algorithms like discrete and real AdaBoost can be regarded as stage-wise optimization of the criterion

$$J(y(\cdot)) = \mathrm{E}\left[\exp(-ty(\boldsymbol{x}))\right], \tag{2.25}$$

where the expectation is a L_2 population expectation or a sample average. The population minimizer of this criterion is $(1/2)\mathrm{logit}(\boldsymbol{x})$,[11] therefore optimizing (2.25) is equivalent to fitting a logistic regression model. The authors also show that replacing the AdaBoost criterion by related ones like the expected negative binomial log likelihood

$$L(y(\cdot)) = \mathrm{E}\left[\log\left(1 + \exp(-ty(\boldsymbol{x}))\right)\right] \tag{2.26}$$

gives very similar results. The algorithm using (2.26) is referred to as *LogitBoost*.

We sketch the idea of the proof for real AdaBoost. The reweighting factors of real AdaBoost, in the notation of [FHT98], are

$$w(t|\boldsymbol{x}, y(\cdot)) = \frac{\exp(-ty(\boldsymbol{x}))}{\mathrm{E}[\exp(-ty(\boldsymbol{x}))|\boldsymbol{x}]}. \tag{2.27}$$

The proof relies on two properties of the exponential criterion (2.25). First of all we have

$$\exp(-t(y(\boldsymbol{x}) + \delta y(\boldsymbol{x}))) = \exp(-ty(\boldsymbol{x}))\exp(-t\delta y(\boldsymbol{x})), \tag{2.28}$$

and secondly the Taylor expansions around $y = 0$ of the negative binomial log likelihood (which is minimized when fitting additive logistic regression models) and the exponential criterion are identical up to second order, so that

$$\begin{aligned}
&\mathrm{E}\left[\exp(-t(y(\boldsymbol{x}) + \delta y(\boldsymbol{x})))|\boldsymbol{x}\right] \\
&= \mathrm{E}\left[\exp(-ty(\boldsymbol{x}))\left(\log(1 + \exp(-t\delta y(\boldsymbol{x}))) + O((\delta y(\boldsymbol{x}))^3)\right)|\boldsymbol{x}\right] \\
&\approx \mathrm{E}\exp(-ty(\boldsymbol{x}))|\boldsymbol{x}\mathrm{E}_w\left[\log(1 + \exp(-t\delta y(\boldsymbol{x})))|\boldsymbol{x}\right].
\end{aligned} \tag{2.29}$$

Here, E_w denotes expectation over the reweighted sample, i.e. $\mathrm{E}_w[h(t, \boldsymbol{x})|\boldsymbol{x}] = \mathrm{E}[h(t, \boldsymbol{x})w(t|\boldsymbol{x}, y(\cdot))|\boldsymbol{x}]$. By reweighting the sample, we therefore manage to exchange the exponential loss (for which we don't have a fitting algorithm) by the usual binomial loss, and the component $\delta y(\cdot)$ can be trained by simple logistic regression.

[11] The *logit* transformation was introduced in subsection 1.2.6.

2.2.3 LogitBoost and Gaussian process classification

We will now have a closer look at the training algorithm of a Gaussian process classification model with Bernoulli noise $P(t_i|y_i) = \sigma(t_i y_i)$. To be consistent with the notation in [FHT98] and [WB98], we introduce the notation $t_i^* = (1/2)(t_i + 1) \in \{0, 1\}$. The negative log likelihood is $-\log P(t_i|y_i) = \log(1 + \exp(-t_i y_i)) = -t_i^* y_i + \log(1 + \exp(y_i))$, and the criterion Ψ from (1.15) is

$$\Psi = -\log P(\boldsymbol{y}, \boldsymbol{t}) = -\boldsymbol{t}^{*t}\boldsymbol{y} + \sum_{i=1}^{n} \log(1 + \exp(y_i)) - \log N(\boldsymbol{y}|0, \boldsymbol{K}). \quad (2.30)$$

The model is fitted using *Fisher scoring* (see [MN83], [GS94]). Since we are using the canonical link in the generalized linear model, this is equivalent to minimizing Ψ by the Newton-Raphson algorithm, i.e. using updates of the form $\boldsymbol{y}_{new} = \boldsymbol{y} + \delta\boldsymbol{y}$ where

$$\delta\boldsymbol{y} = -\left(\nabla\nabla\Psi\right)^{-1} \nabla\Psi \quad (2.31)$$

and Hessian and gradient are evaluated at $\boldsymbol{y}$. It is easy to show that

$$\delta\boldsymbol{y} = \left(\boldsymbol{W} + \boldsymbol{K}^{-1}\right)^{-1} \left(\boldsymbol{t}^* - \sigma(\boldsymbol{y}) - \boldsymbol{K}^{-1}\boldsymbol{y}\right), \quad (2.32)$$

where $\sigma(\boldsymbol{y})$ denotes the vector with the components $\sigma(y_i)$ and $\boldsymbol{W} = \mathrm{diag}(\sigma(y_i)(1 - \sigma(y_i)))_i$. We denote $\pi_i = P(t_i = +1|y_i) = \sigma(y_i)$.

Since the relation between $\boldsymbol{y}$ and the discriminant $y(\boldsymbol{x})$ is linear for a kernel classifier, we can also write this as

$$y_{new}(\boldsymbol{x}) = y(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^t \boldsymbol{K}^{-1} \delta\boldsymbol{y} \quad (2.33)$$

where $\boldsymbol{k}(\boldsymbol{x})$ is the vector with the components $K(\boldsymbol{x}, \boldsymbol{x}_i)$. Denote the Newton correction of $y(\boldsymbol{x})$ by $\delta y(\boldsymbol{x})$. It is interesting to note a connection to the LogitBoost algorithm proposed in [FHT98]. The latter algorithm uses a maximum likelihood criterion without a penalty or a prior. This is possible because the model class for the discriminant is strongly restricted (additive combinations of weak classifiers like decision trees). We can approach maximum likelihood by choosing a very large variance parameter C (or equivalently a very small smoothness parameter λ). Recall that $K = CR$ for a kernel R that has no variance parameter. We then have

$$\delta y(\boldsymbol{x}) = \boldsymbol{r}(\boldsymbol{x})^t \boldsymbol{R}^{-1} \left(\boldsymbol{W} + C^{-1}\boldsymbol{R}^{-1}\right)^{-1} \left(\boldsymbol{t}^* - \sigma(\boldsymbol{y}) - C^{-1}\boldsymbol{R}^{-1}\boldsymbol{y}\right). \quad (2.34)$$

Setting $\boldsymbol{z} = \boldsymbol{W}^{-1}(\boldsymbol{t}^* - \sigma(\boldsymbol{y}))$ (z is referred to as *working response variable*, and $\boldsymbol{z}$ is a sample of this variable at the input points), we see that in the

limit $C \to \infty$ $\delta y(\boldsymbol{x})$ is the ordinary least-squares regression of z on $\boldsymbol{x}$. A look at the LogitBoost algorithm of [FHT98] reveals that the update of $y(\boldsymbol{x})$ used there is exactly the same, except for the fact that $\delta y(\boldsymbol{x})$ is a *weighted* least-squares regression of z on $\boldsymbol{x}$ where the weights of the training points $(\boldsymbol{x}_i, t_i)$ are given by $w_i = \pi_i(1-\pi_i) = \sigma(y_i)(1-\sigma(y_i))$. Working with finite C destroys this direct correspondence which is quite obvious since fitting a Gaussian process regression model by (unpenalized) maximum likelihood does not give sensible results because of the richness of the function class. However, we note that there are boosting-type algorithms that are in a certain sense quite similar to Gaussian process classification except for the fact that the dataset is reweighted before fitting each correction $\delta y(\boldsymbol{x})$.

2.2.4 Continuous reweighting

Now consider an additive model $y(\boldsymbol{x}) = \delta y_1(\boldsymbol{x}) + \delta y_2(\boldsymbol{x}) + \ldots$ (the number of components is not specified in advance) where the component discriminants are "weak" in the sense that their model class is restricted. Define the criterion

$$J(y(\cdot)) = \mathrm{E}\left[[1 - ty(\boldsymbol{x})]_+\right]. \tag{2.35}$$

Note the similarity to the GCKL of (1.6). As argued in subsection 1.2.4, J is an upper bound on the expected misclassification error. Introduce the reweighting distribution w by

$$\tilde{w}(t|\boldsymbol{x}, y(\cdot)) = \frac{[1 - ty(\boldsymbol{x})]_+}{\log(1 + \exp(-ty(\boldsymbol{x})))}, \quad w(t|\boldsymbol{x}, y(\cdot)) = \frac{\tilde{w}(t|\boldsymbol{x}, y(\cdot))}{\mathrm{E}\tilde{w}(t|\boldsymbol{x}, y(\cdot))|\boldsymbol{x}}. \tag{2.36}$$

In practice, reweighting the data can be done by resampling from the dataset in a *bootstrap* manner, but it is much more efficient to reweight the criterion based on the data instead, if the training algorithm of the components allows for such a modification. Let $\tilde{w}_i = \tilde{w}(t_i|\boldsymbol{x}_i, y(\cdot))$ and $w_i = \tilde{w}_i/S$ where $S = \sum_i \tilde{w}_i$. Conditioning on $\boldsymbol{x}$ we have

$$\mathrm{E}\left[[1 - ty(\boldsymbol{x})]_+|\boldsymbol{x}\right] = \mathrm{E}\tilde{w}(t|\boldsymbol{x}, y(\cdot))|\boldsymbol{x}\mathrm{E}_w\left[\log(1 + \exp(-ty(\boldsymbol{x})))|\boldsymbol{x}\right] \tag{2.37}$$

or, in terms of the sample average,

$$\frac{1}{n}\sum_{i=1}^{n}[1 - t_i y_i]_+ = \frac{S}{n}\sum_{i=1}^{n} w_i \log(1 + \exp(-t_i y_i)). \tag{2.38}$$

The right hand side is proportional to the weighted likelihood term of a additive logistic regression model.

Note that this situation is quite different from the one encountered in the derivation of real AdaBoost above (see equations (2.28) and (2.29)). Our loss here is neither multiplicative nor close to the negative log Bernoulli likelihood around 0. We therefore propose a *continuous reweighting* of the criterion to transfer the SVC loss into the Bernoulli classification loss. In light of the aim behind the reweighting of the dataset the latter is clearly favorable since the constraints imposed by the reweighting are applied in a continuous fashion and not only after each (possibly large) Newton step has been performed. Now, in practical terms this is not very useful since we don't have an algorithm to perform this "continuously reweighted fitting" but, as we argue below, it might be useful for understanding the mechanisms going on in Support Vector training.

Figure 2.3 shows a comparative plot of the loss functions considered so far. Note that the negative log likelihood has been scaled by $1/\log 2$. Note also that the derivative of the exponential loss is unbounded for $ty \to -\infty$ while the derivatives of all other loss functions remain bounded. This means that AdaBoost is quite sensitive to outliers and therefore supposed to perform badly on very noisy data. This has indeed been observed and a *soft margin* version of AdaBoost has been suggested (see [MOR98]).

Let us have a closer look at the reweighting distribution $w(t|\boldsymbol{x}, y(\cdot))$ of (2.36). Figure 2.4 shows a plot of $\tilde{w}$ (i.e. the unnormalized w) as a function of $ty(\boldsymbol{x})$. This function is 0 for $ty \geq 1$ and converges to 1 for $ty \to -\infty$.

Figure 2.5 compares the (unnormalized) reweighting factor of AdaBoost with the factor (2.36) applied to the criterion J involving the Support Vector loss. A direct comparison in the region around 0 is difficult since both factors can be arbitrarily rescaled, but the differences are easy to see. First of all, the AdaBoost weight grows without bound for $ty \to -\infty$ while $\tilde{w}$ remains bounded. This means that if there are outliers (in very noisy data) AdaBoost will concentrate the training of the weak classifiers on these examples. This may lead to an overcomplex decision boundary, i.e. to overfitting. Indeed, this *hard margin* behaviour has been observed in [MOR98]. $\tilde{w}$ is largest in the region between -1 and 0, so the training is concentrated on patterns within the margin region (recall that for our *canonical hyperplanes* $y(\cdot)$ the margin region is defined by $ty(\boldsymbol{x}) \in [-1, +1]$). Then, the AdaBoost weighting factor is positive everywhere while $\tilde{w}$ is 0 for $ty \geq 1$. This reflects the fact that as long as a pattern lies out of the margin region (on the correct side), it doesn't affect the training of the next weak classifier at all. Note how this

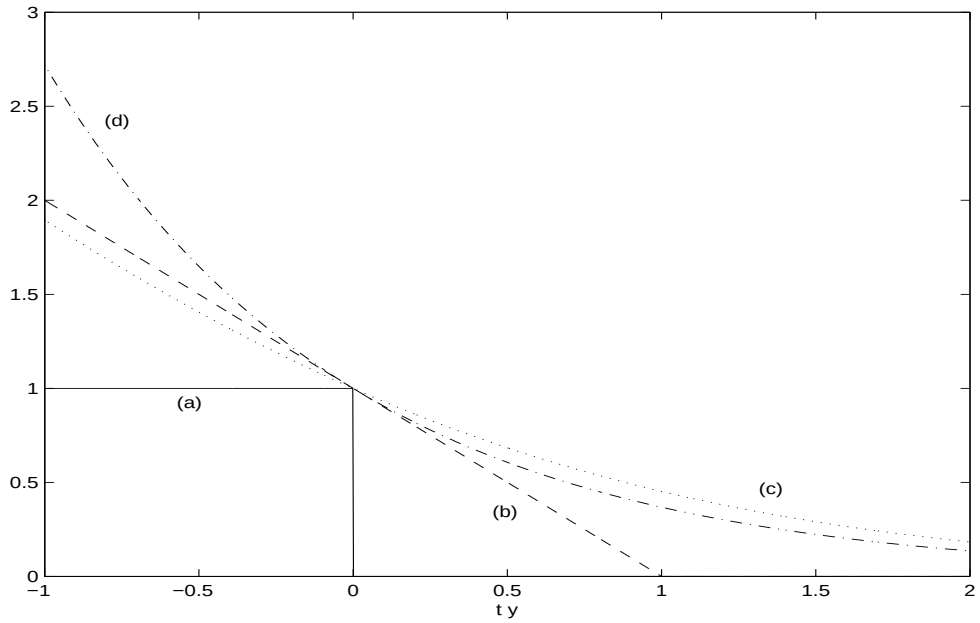

Figure 2.3: Loss functions considered so far. (a) is the misclassification loss, (b) the SVC loss, (c) the scaled negative log likelihood and (d) is the exponential loss used in real AdaBoost.

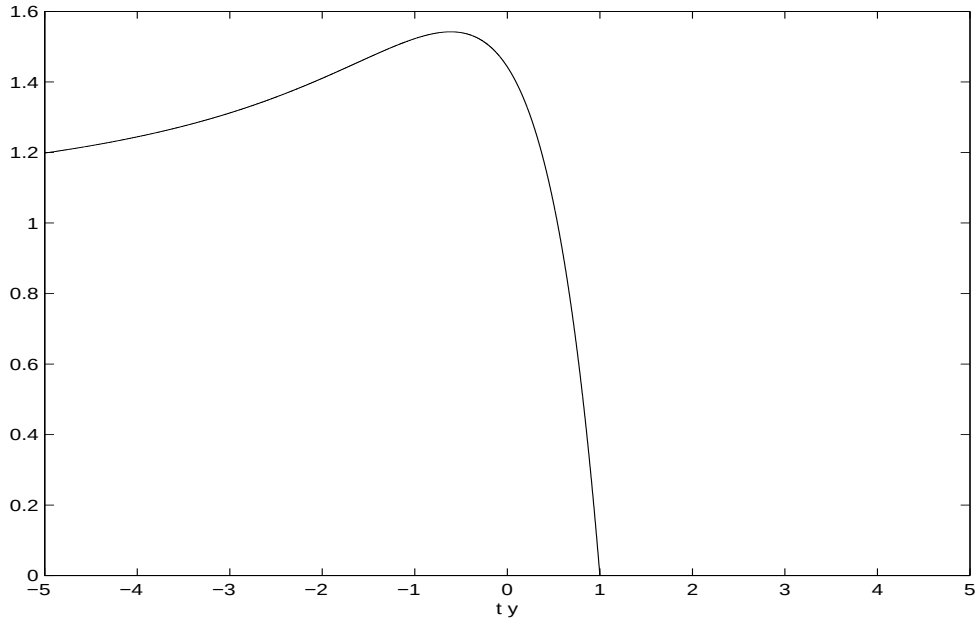

Figure 2.4: Unnormalized reweighting distribution $\tilde{w}$ as a function of ty.

fact nicely corresponds to the notion of active and inactive constraints encountered during the optimization of the Support Vector criterion. A pattern lying out of the margin region corresponds to an inactive constraint, and as long as it remains inactive, this constraint can be completely ignored. Interestingly enough, a similar cutoff technique has been proposed for boosting algorithms (*influence trimming*, see [Fri99], p.9) with the aim of faster computation. In Support Vector training the cutoff point is actually adapted to the data (see below) in a theoretically more satisfying way than the heuristic procedures suggested for incluence trimming. It has been reported that, using influence trimming, between 90-95% of the observations are typically ignored over large fractions of the training period. This corresponds to the surprising effectiveness of *chunking algorithms* used in Support Vector training (see [Pla98],[Joa98]).

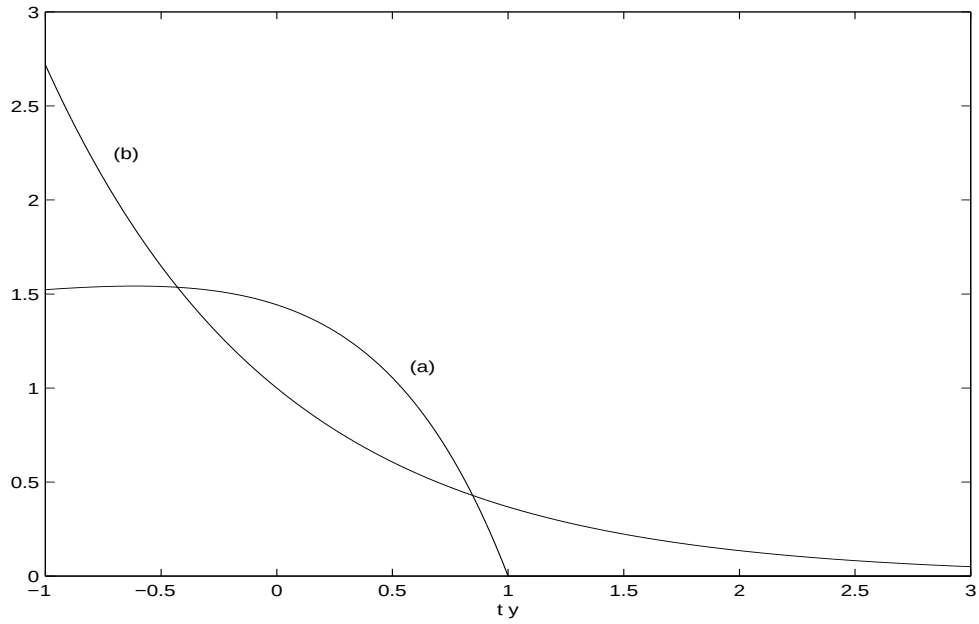

Figure 2.5: The reweighting factors of the SVC loss criterion (a) and AdaBoost (b).

2.2.5 Incorporating the prior

It is clear that when using a spline smoother as discriminant, we cannot simply apply maximum likelihood training. As shown in subsection 2.2.3, the "weak discriminants" $\delta y(\cdot)$ are also spline smoothers and therefore form a very broad model class. The likelihood must be penalized in a certain way

so as to avoid overcomplex discriminant functions, and this can be done by imposing a Gaussian process prior on $y(\cdot)$, as discussed above. Adding a penalty somewhat destroys the direct relationship between techniques like LogitBoost and GPC training, but we argue that the idea of continuously reweighting the loss term can be applied here too. First of all, the reweighting only affects the loss part of the criterion and leaves the prior term unchanged (recall that reweighting means that we change the true distribution $P(t|\boldsymbol{x})$). Then we note that the most commonly used priors encode characteristics like smoothness, sometimes an expected periodicity or an expected linear trend. Considering an additive model, i.e. a linear combination of weak discriminants, it is therefore perfectly reasonable to impose the same prior on the components than on the whole committee. This argument applies especially to the case of a spline smoother where the "weak" discriminants have the same functional form as the whole. However, we exclude the smoothing parameter (or variance parameter) from this argument since it is reasonable to use different smoothing parameters for the various $\delta y(\cdot)$, depending on the characteristics of the (reweighted) dataset they have to fit. A smoothing parameter that varies continuously with $y(\cdot)$ would fit even better.

Let us write the Support Vector classification criterion in the following form (adopted from [FHT98]):

$$C\mathrm{E}\left[[1-ty(\boldsymbol{x})]_+\right] - \log P(y(\cdot)). \tag{2.39}$$

Using a sample average and conditioning on $\boldsymbol{x}$, this becomes

$$\frac{C}{n}\sum_{i=1}^{n}[1-t_i y_i]_+ - \frac{1}{n}\log N(\boldsymbol{y}|0,\boldsymbol{R}). \tag{2.40}$$

At this point it is suitable to separate the variance parameter C from the kernel, but by using $K = CR$ it can always be absorbed. Applying the reweighting leaves the prior term unchanged (note that R does not have a variance parameter), and using (2.37) we can write (2.39) as

$$\begin{aligned}&(C\mathrm{E}\tilde{w}(t|\boldsymbol{x})|\boldsymbol{x})\,\mathrm{E}_w\big[\log(1+\exp(-ty(\boldsymbol{x})))\big] - \log P(y(\cdot))\\ &= \mathrm{E}_w\big[(C\mathrm{E}\tilde{w}(t|\boldsymbol{x})|\boldsymbol{x})\log(1+\exp(-ty(\boldsymbol{x}))) - \log P(y(\cdot))\big]\end{aligned} \tag{2.41}$$

or, in terms of a sample average conditioned on the $\boldsymbol{x}_i$, as

$$\begin{aligned}&\frac{CS}{n}\sum_{i=1}^{n} w_i \log(1+\exp(-t_i y_i)) - \frac{1}{n}\log N(\boldsymbol{y}|0,\boldsymbol{R})\\ &= \sum_{i=1}^{n} w_i\left(\frac{CS}{n}\log(1+\exp(-t_i y_i)) - \frac{1}{n}\log N(\boldsymbol{y}|0,\boldsymbol{R})\right).\end{aligned} \tag{2.42}$$

Note that we write $\tilde{w}(t|\boldsymbol{x})$ instead of $\tilde{w}(t|\boldsymbol{x}, y(\cdot))$ for clarity but keep in mind that the weights depend on the old $y(\cdot)$. Choosing $C\mathrm{E}\tilde{w}(t|\boldsymbol{x})|\boldsymbol{x}$ (or CS/n) as continuously varying variance parameter, we see that SVC training can regarded as fitting a Gaussian process classification model under continuous reweighting of the dataset. More insight can be gained by expanding $\boldsymbol{y}_{new} = \boldsymbol{y} + \delta\boldsymbol{y}$. Then, (2.42) can be written as

$$\sum_{i=1}^{n} w_i \left(\frac{CS}{n} \log(1 + \exp(-t_i(y_i + \delta y_i))) - \frac{1}{n} \log N(\delta\boldsymbol{y}| - \boldsymbol{y}, \boldsymbol{R}) \right). \quad (2.43)$$

We note that the local prior for $\delta y(\cdot)$ encourages a behaviour in opposition to $y(\cdot)$, at least w.r.t. these training points that receive zero weight. This is in fact what is done during Support Vector training: If a point is safely classified out of the margin region, i.e. $t_i y(\boldsymbol{x}_i) > 1$, then, in order to enlarge the margin, a certain force will be applied to encourage $t_i y(\boldsymbol{x}_i)$ to decrease, therefore encourage $t_i \delta y(\boldsymbol{x}_i) < 0$. In fact, if it should ever happen during training that all training points are correctly classified with margin, this "prior" force is the striking difference between any standard Perceptron learning algorithm and the Support Vector algorithm enforcing the largest margin possible on the training set.

Let us have a closer look on the variance parameter CS/n. Looking at figure 2.4 we note that it lies roughly between 0 and $1.6C$. It is largest if most of the patterns lie in the margin region and very small if most of the patterns are currently classified with margin. This fits nicely into the interpretation of the local tuning of $\delta y(\cdot)$. The likelihood term should receive the highest relative importance when there are a lot of patterns in the margin region, i.e. a lot of patterns $\delta y(\cdot)$ should concentrate on. To move these patterns out of the margin region, i.e. to act somewhat against the force put forward by the local prior is encouraged in this way. If there are only very few patterns in the margin region, the tuning of $\delta y(\cdot)$ will concentrate on these, but at the same time the small relative importance of the likelihood term will strongly encourage (via the local prior) an enlargement of the margin.

Using these observations we can conclude that the interpretation of SVC as continuously reweighted version of a logistic kernel regression discriminant might prove useful to understand the differences in behaviour of SVC and GPC and to explain the often observed similarities in performance between SVC and boosting architectures, especially those using a soft margin criterion.

2.3 Comparing GPC and SVC prediction

Recall the Gaussian process classification model: GPC is a special case of a nonparametric generalized linear model with Bernoulli-distributed noise, and the natural parameter of the Bernoulli distribution is the logit $y = \log(P(t = +1|\boldsymbol{x})/P(t = -1|\boldsymbol{x}))$, so it is "natural" to model the logit $y(\boldsymbol{x})$ by our Gaussian process:

$$P(t_i|y_i) = \sigma(t_i y_i) \tag{2.44}$$

where $\sigma(\cdot)$ is the logistic function, and

$$-\log P(\boldsymbol{t}|\boldsymbol{y}) = \sum_i \log(1 + \exp(-t_i y_i)). \tag{2.45}$$

At $\hat{\boldsymbol{y}}$ the derivative of Ψ (see (1.15)) vanishes, and by setting $\hat{\boldsymbol{\alpha}}_{GP} = (\sigma(-t_i \hat{y}_i))_i$ we arrive at

$$\hat{\boldsymbol{y}} = \boldsymbol{K}\boldsymbol{T}\hat{\boldsymbol{\alpha}}_{GP}, \tag{2.46}$$

where $\boldsymbol{T} = \operatorname{diag} \boldsymbol{t}$. Denote the measures obtained by applying the Laplace approximation around $\hat{\boldsymbol{y}}$ by P_a. The approximative predictive distribution $P_a(y_*|\boldsymbol{t})$ is normal with mean $\boldsymbol{k}(\boldsymbol{x}_*)^t \boldsymbol{K}^{-1} \hat{\boldsymbol{y}} = \boldsymbol{k}(\boldsymbol{x}_*)^t \boldsymbol{T} \hat{\boldsymbol{\alpha}}_{GP}$, and we predict $t_* = +1$ iff this mean is greater than 0.

Also recall the discussion in section 1.4, especially (1.27) and (1.26). Since we agreed to drop the bias term and absorbed the variance parameter C into the kernel, we have $\hat{\boldsymbol{y}} = \boldsymbol{K}\boldsymbol{T}\hat{\boldsymbol{\alpha}}_{SVM}$, and the predictor for y_* is $\boldsymbol{k}(\boldsymbol{x}_*)^t \boldsymbol{T} \hat{\boldsymbol{\alpha}}_{SVM}$, where $\hat{\boldsymbol{\alpha}}_{SVM}$ denotes the vector of dual variables at the optimum. Both GPC and SVC predict a new target by reweighting the vector of training targets to arrive at a weight vector $\hat{\boldsymbol{c}} = \boldsymbol{T}\hat{\boldsymbol{\alpha}}$ which is then used to form an average over the correlations between the new input point and the training points. That does not come as a surprise since both methods are special cases of spline smoothers, as shown in section 2.1. The difference between the two methods lies in the reweighting factor $\hat{\alpha}_i$ for a target t_i. For the GP case, this is $\sigma(-t_i \hat{y}_i) \in (0, 1)$. For fixed t_i this is a smooth function of $\hat{y}_i$ which is symmetric to 0 and never attains the extreme values 0 (the discriminant ignores the value of $K(\boldsymbol{x}_*, \boldsymbol{x}_i)$ even if it is significant) or 1. For SVC, the reweighting factor is the dual variable $\hat{\alpha}_i \in [0, 1]$. The KKT conditions (1.27) show that for a case which is correctly classified with margin, i.e. $t_i \hat{y}_i > 1$, the "distance" $K(\boldsymbol{x}_*, \boldsymbol{x}_i)$ has no effect on the prediction at $\boldsymbol{x}_*$. On the other hand, a case within the margin ($t_i \hat{y}_i < 1$) has maximum effect on predictions, resulting in a term $t_i K(\boldsymbol{x}_*, \boldsymbol{x}_i)$ in the expansion of $y(\boldsymbol{x}_*)$. If a case is on the

margin, i.e. $t_i\hat{y}_i = 1$, the effect on predictions lies between these extremes. The reweighting factors of both methods as functions of $\hat{y}_i$ for target $t_i = +1$ are plotted in figure 2.6.

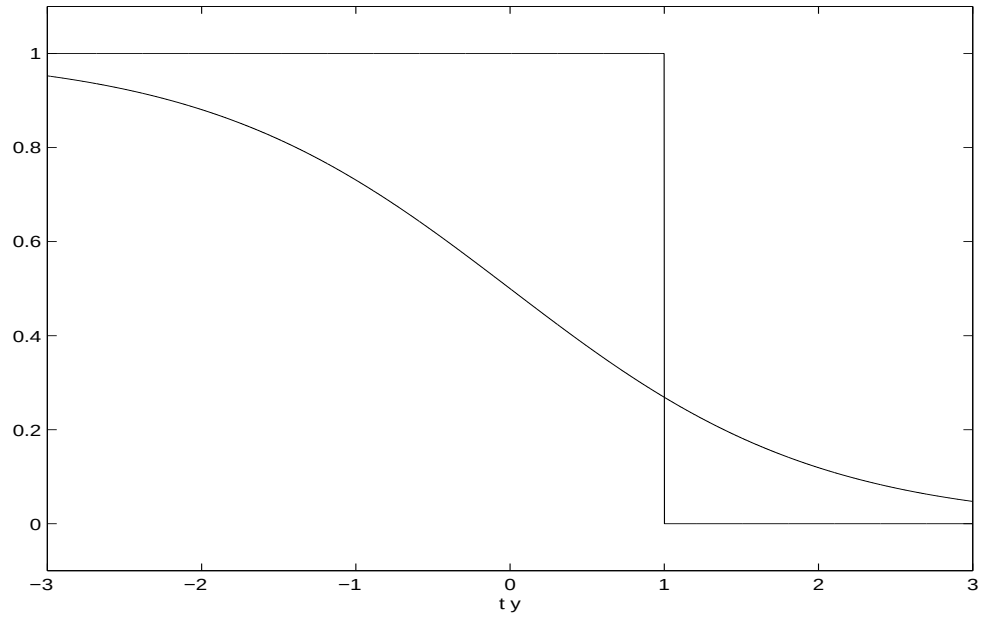

Figure 2.6: Reweighting factors as functions of $\hat{y}_i$ for SVC and GPC, target value $t_i = +1$. The GPC reweighting factor varies smoothly with $\hat{y}_i$.

We have to be careful not to misinterpret the relationship between the predictions of the two methods which actually looks simpler than it really is. The $\hat{\boldsymbol{y}}$ vectors can be quite different, being *implicitely* defined by the systems $\hat{\boldsymbol{y}} = \boldsymbol{KT}\hat{\boldsymbol{\alpha}}_{GP}(\hat{\boldsymbol{y}})$ and $\hat{\boldsymbol{y}} = \boldsymbol{KT}\hat{\boldsymbol{\alpha}}_{SVM}(\hat{\boldsymbol{y}})$ respectively, these systems encode via the definition of the dual variables, the completely different expectations we have in the predictions. Jaakkola and Haussler [JH99] provide some further insight into the relationship between the primal and dual variables.

Chapter 3

Variational and Bayesian techniques

In this chapter, we give a principled derivation of variational techniques for probabilistic inference and learning. Motivations from convex analysis, statistical physics and information theory are given. This chapter is tutorial in nature and provides a firm basis for the technique introduced in the subsequent chapter. However, the generality offered here is not necessary for the understanding of the latter, and readers not interested in the details can safely skip this chapter. We also briefly discuss Bayesian terms like MAP and the evidence framework.

3.1 Variational techniques for probabilistic inference

Following the work of Jaakkola [Jaa97], we will derive the variational principle using simple terms from *convex analysis*. Variational probabilistic inference is equivalent to *variational free energy minimization* in *statistical physics*, being an instance of the powerful *maximum entropy principle*, based on *information theory*. We also heavily rely on the notes [Jaa99].

Another bridge to information and coding theory can be drawn via the *minimum description length principle*, by applying the so called *bits-back encoding scheme*.

3.1.1 Convex duality and variational bounds

This subsection is taken from [Jaa97]. We stress that the variational approximation can in principle be introduced using three lines of algebra, but a more detailed analysis gives better insight. The convex duality principle is much more general than the variational probabilistic inference technique and can be applied as a general approximation method, suggesting metrics to measure the approximation error in an automatic way. See [Roc70] for an introduction to convex analysis, [CT91] for all information-theoretic terms.

$f : X \to \mathbb{R}$ is *(strictly) convex* iff for any $\boldsymbol{x}, \boldsymbol{x}' \in X,\ \boldsymbol{x} \neq \boldsymbol{x}',\ \lambda \in (0,1)$ we have $f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{x}') \leq (<)\lambda f(\boldsymbol{x}) + (1-\lambda)f(\boldsymbol{x}')$. This is equivalent to $f(\mathrm{E}\boldsymbol{X}) \leq (<)\mathrm{E}f(\boldsymbol{X})$ for any random variable $\boldsymbol{X} \in X$ not concentrated on a single point, this fact being known as *Jensen's inequality*. For convenience, we assume that f is differentiable. We define $epi(f) = \{(\boldsymbol{x}, y) \,|\, f(\boldsymbol{x}) \leq y\}$ which is convex. Every convex set S can be represented as intersection of all halfspaces that contain S, therefore a parameterization of this set of halfspaces is a unique representation of $epi(f)$ and f. In our case, we only need to consider halfspaces $(\boldsymbol{\xi}, \mu)$ of the form $\{(\boldsymbol{x}, y) \,|\, \boldsymbol{x}^t\boldsymbol{\xi} - y - \mu \leq 0\}$. $(\boldsymbol{\xi}, \mu)$ contains $epi(f)$ iff $\boldsymbol{x}^t\boldsymbol{\xi} - f(\boldsymbol{x}) - \mu \leq 0$ for all $\boldsymbol{x}$, i.e. iff

$$\mu \geq \max_{\mathbf{x} \in X} \left(\boldsymbol{x}^t\boldsymbol{\xi} - f(\boldsymbol{x})\right) = f^*(\boldsymbol{\xi}). \tag{3.1}$$

f^* is called the *convex dual* of f. It is defined over Ξ, the dual space of X[1]. f^* is convex itself, and $f^{**} = f$, i.e.

$$f(\boldsymbol{x}) = \max_{\xi \in \Xi} \left(\boldsymbol{x}^t\boldsymbol{\xi} - f^*(\boldsymbol{\xi})\right). \tag{3.2}$$

The fundamental relation between f and its dual f^* is

$$\boldsymbol{x}^t\boldsymbol{\xi} \leq f(\boldsymbol{x}) + f^*(\boldsymbol{\xi}). \tag{3.3}$$

This gives us an affine family of lower bounds, indexed by $\boldsymbol{\xi}$:

$$f(\boldsymbol{x}) \geq \boldsymbol{x}^t\boldsymbol{\xi} - f^*(\boldsymbol{\xi}) = f_\xi(\boldsymbol{x}). \tag{3.4}$$

$\boldsymbol{\xi}$ will be called the *variational parameter*. The halfspaces on the graph of f^* (i.e. $\mu = f^*(\boldsymbol{\xi})$) are the *critical* ones in the sense that every system of halfspaces whose intersection is $epi(f)$ must contain all these. If f is differentiable, we can get more insight into the nature of Ξ. Let f be strictly convex. For any fixed $\boldsymbol{\xi}$ there exists one and only one $\boldsymbol{x}$ such that the maximum in

[1] If $X = \mathbb{R}^p$, Ξ is isomorphic to $\mathbb{R}^p$ and we will identify the two spaces.

(3.1) is attained at $\boldsymbol{x}$, thus $0 = \nabla_{\mathbf{x}}(\boldsymbol{x}^t\boldsymbol{\xi} - f(\boldsymbol{x})) = \boldsymbol{\xi} - \nabla_{\mathbf{x}} f(\boldsymbol{x})$. Therefore, the conjugate space Ξ is simply the *gradient space* of f. Substiting this into (3.2), we arrive (after some algebra) at

$$f(\boldsymbol{x}) = \max_{\boldsymbol{x}' \in X} \left((\boldsymbol{x} - \boldsymbol{x}')^t \nabla_{\mathbf{x}'} f(\boldsymbol{x}') + f(\boldsymbol{x}') \right), \tag{3.5}$$

which is a maximum over tangent hyperplanes for f. The critical halfspaces defining f are therefore the tangent hyperplanes which are affine lower bounds of f. For strictly convex f, the maximum in (3.5) is attained at exactly one $\boldsymbol{x}'$.

Is there a general way to characterize the accuracy of the affine variational bounds in terms of the parameter $\boldsymbol{\xi}$? If f is strictly convex, then for every $\boldsymbol{\xi}'$ there exists exactly one $\boldsymbol{x}'$ such that $f(\boldsymbol{x}') = f_{\xi'}(\boldsymbol{x}')$. $\boldsymbol{x}'$ and $\boldsymbol{\xi}'$ are called *corresponding points*. Define the distance from $\boldsymbol{\xi}$ to $\boldsymbol{\xi}'$ as $D_{f^*}(\boldsymbol{\xi} \to \boldsymbol{\xi}') = f(\boldsymbol{x}') - f_{\xi}(\boldsymbol{x}')$, i.e. the approximation error made by using the suboptimal $\boldsymbol{\xi}$ instead of $\boldsymbol{\xi}'$. Figure 3.1 illustrates this definition. Using the duality between f and f^*, we can define $D_f(\boldsymbol{x} \to \boldsymbol{x}')$ analogously. If $\boldsymbol{x}$ and $\boldsymbol{\xi}$ are other corresponding points, we have

$$D_f(\boldsymbol{\xi} \to \boldsymbol{\xi}') = D_{f^*}(\boldsymbol{\xi}' \to \boldsymbol{\xi}) \tag{3.6}$$

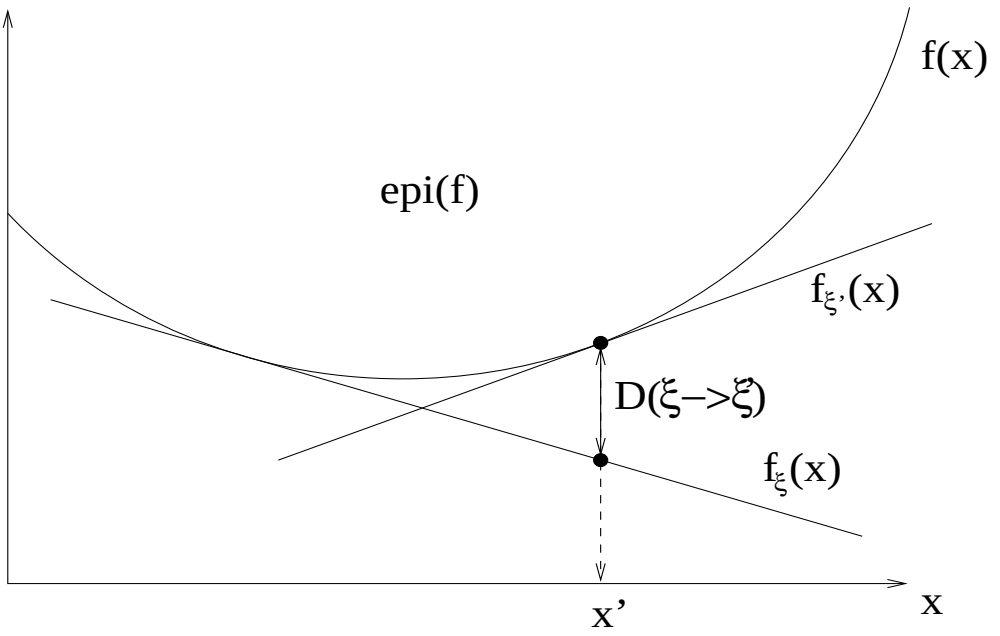

Figure 3.1: The dual metric.

How can we use convex duality to solve concrete approximation problems? Suppose, given a convex function f and a fixed point $\boldsymbol{x}'$, we would like to approximate $f(\boldsymbol{x}')$. If the computation of $f(\boldsymbol{x}')$ is infeasible, we can try to approximate the value by $f_\xi(\boldsymbol{x}')$ for some $\boldsymbol{\xi}$. This makes sense only if there is a substantial subset $\tilde{\Xi} \subset \Xi$ such that the evaluation of the approximation at $\boldsymbol{x}'$ for any $\boldsymbol{\xi} \in \tilde{\Xi}$ is feasible. Assuming this is so, we have to find a method to choose $\boldsymbol{\xi}$. If $\boldsymbol{\xi}'$ corresponding to $\boldsymbol{x}'$ is in $\tilde{\Xi}$, we have $f(\boldsymbol{x}') = f_{\xi'}(\boldsymbol{x}')$ and the approximation is exact. Otherwise, it is sensible to choose $\boldsymbol{\xi} \in \tilde{\Xi}$ such as to minimize the induced metric $D_{f^*}(\boldsymbol{\xi} \to \boldsymbol{\xi}')$. This approach is completely automatic. Given the space X and the convex function f, we only have to determine the space Ξ, the dual f^*, then select a subset $\tilde{\Xi}$ such that the bounds $f_\xi(\boldsymbol{x}')$ and the metric $D_{f^*}(\boldsymbol{\xi} \to \boldsymbol{\xi}')$ can be evaluated feasibly for all $\boldsymbol{\xi} \in \tilde{\Xi}$. We will show below how the variational method for probabilistic inference and learning as well as the EM algorithm arise naturally as special cases within the convex duality framework.

3.1.2 Maximum entropy and variational free energy minimization

The maximum entropy principle has its origin in *statistical physics* where it is applied for example to thermodynamical systems. Its usage for selection of Bayesian priors has been advocated by Jaynes (see [Jay82]). A comprehensive introduction is given in [CT91], chapter 11. The idea behind the maximum entropy principle is to avoid assigning different probabilities to configurations of a system that we have no reason to differentiate. We will first introduce the abstract method and then give a simple physical example.

Recall the method of *Lagrange multipliers* (see for example [Fle80]). Given a criterion $f(\boldsymbol{x})$ to minimize under some equality constraints $c_i(\boldsymbol{x}) = \alpha_i,\ i \in E$ and some inequality constraints $c_i(\boldsymbol{x}) \geq 0,\ i \in I,\ C = E \cup I$, we introduce dual variables $\boldsymbol{\lambda} = (\lambda_i)_{i \in C}$ and construct the *Lagrangian*

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{i \in C} \lambda_i c_i(\boldsymbol{x}). \tag{3.7}$$

If $\boldsymbol{x}^*$ is a solution to the original (primal) problem, then there exist a $\boldsymbol{\lambda}^*$ such that $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is a saddlepoint of L, i.e. $\boldsymbol{x}^*$ minimizes $L(\cdot, \boldsymbol{\lambda}^*)$ and $\boldsymbol{\lambda}^*$ maximizes $L(\boldsymbol{x}^*, \cdot)$. By setting $\partial L / \partial \boldsymbol{x}$ equal to zero and solving for $\boldsymbol{x}$, we arrive at a representation of $\boldsymbol{x}^*$ in terms of $\boldsymbol{\lambda}^*$. The latter can usually be determined as a value that satisfies all the constraints. The Karush-Kuhn-Tucker (KKT) conditions constrain the dual variables corresponding to the inequality constraints: $\lambda_i^* \geq 0$ and $\lambda_i^* c_i(\boldsymbol{x}^*) = 0$ for all $i \in I$.

The *maximum entropy problem* is to find, among all distributions $P(\boldsymbol{x})$ which fulfil all *moment constraints*

$$\mathrm{E}_P\left[r_i(\boldsymbol{x})\right] = \alpha_i, \tag{3.8}$$

one that maximizes the *differential entropy*

$$H(P) = \mathrm{E}_P\left[-\log P(\boldsymbol{x})\right]. \tag{3.9}$$

As mentioned above, inequality constraints can also be dealt with, but we avoid them here for clarity. See [CT91] for a thorough discussion of entropy and related information-theoretic terms. We always have the "implicit" constraint $r_0 \equiv 1, \alpha_0 = 1$, ensuring that P is a distribution. The Lagrangian is

$$L(P, \boldsymbol{\lambda}) = \int P(\boldsymbol{x}) \log P(\boldsymbol{x})\, d\boldsymbol{x} - \sum_i \int P(\boldsymbol{x}) r_i(\boldsymbol{x})\, d\boldsymbol{x}. \tag{3.10}$$

Solving $\partial L / \partial P(\boldsymbol{x}) = 0$ for $P(\boldsymbol{x})$ gives

$$P_\lambda(\boldsymbol{x}) = Z(\boldsymbol{\lambda})^{-1} \exp\left(\sum_{i \neq 0} \lambda_i r_i(\boldsymbol{x})\right) \tag{3.11}$$

where the normalization constant is

$$Z(\boldsymbol{\lambda}) = \int \exp\left(\sum_{i \neq 0} \lambda_i r_i(\boldsymbol{x})\right) d\boldsymbol{x}, \tag{3.12}$$

called the *partition function*. Note that λ_0 is not contained in $\boldsymbol{\lambda}$. Knowing the log partition function (w.r.t. $\boldsymbol{\lambda}$) is highly important since we can express all expectations of products of the $r_i(\boldsymbol{x})$ over P_λ in terms of higher order derivatives of $\log Z(\boldsymbol{\lambda})$. For example, we have $(\partial^2 / \partial\lambda_i \partial\lambda_j) \log Z(\boldsymbol{\lambda}) = \mathrm{Cov}_{P_\lambda}\left[r_i(\boldsymbol{x}) r_j(\boldsymbol{x})\right]$. Therefore, $\log Z(\boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}$.

We choose $\boldsymbol{\lambda}^*$ such that (3.11) fulfils all the constraints, and write $P^* = P_{\lambda^*}$. If P' is any other distribution satisfying the constraints, one can show that $H(P^*) - H(P') = D(P' || P^*) \geq 0$, where D denotes the *(differential) relative entropy* (see [CT91]), therefore the maximum $H(P^*)$ is unique (the distribution P^* might not be unique). We can use the convex duality principle to derive another expression for $\boldsymbol{\lambda}^*$ which is sometimes useful. One can show that

$$-H(P^*; \boldsymbol{\alpha}) = \sum_i \lambda_i^* \alpha_i - \log Z(\boldsymbol{\lambda}^*). \tag{3.13}$$

Since the log partition function is convex, the function $\boldsymbol{\lambda}^t\boldsymbol{\alpha} - \log Z(\boldsymbol{\lambda})$ is concave. Its maximum point is obtained by equating the gradient w.r.t. $\boldsymbol{\lambda}$ to zero, and this point is $\boldsymbol{\lambda}^*$. Therefore:

$$-H(P^*;\boldsymbol{\alpha}) = \max_{\lambda} \left(\boldsymbol{\lambda}^t\boldsymbol{\alpha} - \log Z(\boldsymbol{\lambda})\right), \tag{3.14}$$

i.e. the negative entropy $-H(P^*;\boldsymbol{\alpha})$ as a function of $\boldsymbol{\alpha}$ and the log partition function are a convex duality pair. If the right-hand side of (3.14) and/or its gradient can be efficiently computed, any optimizer can be used to find $\boldsymbol{\lambda}^*$.

Now consider a physical system, consisting of N identical subsystems. At any time, each system is in exactly one of the configurations $\boldsymbol{x} \in X$, with the corresponding energy $E(\boldsymbol{x})$. Let $N(\boldsymbol{x})$ be the number of subsystems in configuration $\boldsymbol{x}$ at a fixed time, and $P(\boldsymbol{x}) = N(\boldsymbol{x})/N$ the empirical distribution. The total energy of the system E_{tot} is constant over time:

$$\frac{1}{N}E_{tot} = \mathrm{E}\left[E(\boldsymbol{x})\right] \tag{3.15}$$

The vector of configurations for each system is called *microstate*, the distribution $P(\boldsymbol{x})$ *macrostate*. Every microstate corresponds to a macrostate, and the most probable macrostate is one with the most microstates associated (given that all microstates are equiprobable). For large N, the most probable macrostate converges in probability against the maximum entropy distribution, given the moment constraint (3.15) (in fact, much stronger statements can be proved using the theory of *types*, see [CT91], chapter 12). This distribution is the *Boltzmann distribution*

$$P^*(\boldsymbol{x}) = Z(P^*)^{-1}\exp\left(-\beta E(\boldsymbol{x})\right), \tag{3.16}$$

where $\beta = 1/T$ is the inverse temperature of the system, which stands in 1-1 relation to $\lim_{N\to\infty} E_{tot}/N$. P^* is found by maximizing the Lagrangian

$$(-\beta\Phi(P)) = -\beta\mathrm{E}_P\left[E(\boldsymbol{x})\right] + H(P). \tag{3.17}$$

$\Phi(P)$ is called *variational free energy*. $\Phi(P^*)$, evaluated at the Boltzmann distribution, is called *free energy*. We have

$$(-\beta\Phi(P^*)) = \log Z(P^*). \tag{3.18}$$

For large systems, it is usually infeasible to compute the free energy. But since $(-\beta\Phi(P)) \leq (-\beta\Phi(P^*))$, the variational free energy for any distribution P gives an upper bound on the free energy. The larger (3.17), the tighter the bound is.

3.1.3 Variational approximation of probabilistic inference

Here we establish a correspondence between variational free energy minimization and approximative probabilistic inference. We also show up connections to convex duality. Let $P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)$ be a probability model over visible or evidence variables $\boldsymbol{x}_v$ and hidden or latent variables $\boldsymbol{x}_h$. Our goal is to compute $\log P_0(\boldsymbol{x}_v)$, the log probability of the evidence we have gained. Since we create an "artificial" physical system, we can set $\beta = 1$. Define the energy of a configuration $\boldsymbol{x}_h$ as $E(\boldsymbol{x}_h) = -\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)$. This can be motivated from information theory, since an optimal code needs approximately $E(\boldsymbol{x}_h)$ "nats" to encode $(\boldsymbol{x}_v, \boldsymbol{x}_h)$, and this is directly related to the energy we need to send this message over a continuous channel (e.g. a Gaussian one) with constant noise level. More details about the relation between probability models and codes are given below. We then have

$$-\Phi(P) = \mathrm{E}_P \left[\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)\right] + H(P). \tag{3.19}$$

The partition function is

$$Z = \int \exp(-E(\boldsymbol{x}_h))\, d\boldsymbol{x}_h = \int P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)\, d\boldsymbol{x}_h = P_0(\boldsymbol{x}_v), \tag{3.20}$$

and the Boltzmann distribution is the posterior $P(\boldsymbol{x}_h|\boldsymbol{x}_v)$. As above, the value of (3.19) for any distribution $P(\boldsymbol{x}_h)$ is a lower bound on the desired log partition function:

$$\log P_0(\boldsymbol{x}_v) \geq \mathrm{E}_P \left[\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)\right] + H(P) \tag{3.21}$$

This result can also be derived using convex duality (see subsection 3.1.1). Let the space X consist of all functions $\phi_{\mathbf{x}_v}(\boldsymbol{x}_h) = \log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)$. A vector of this space can be regarded as a real vector whose components are indexed by $\boldsymbol{x}_h$. The dual space Ξ is the space of distributions $P(\boldsymbol{x}_h)$. Define f on X as $f(\phi) = \log \int \exp(\phi(\boldsymbol{x}_h))\, d\boldsymbol{x}_h$. One can show (see [Jaa97], section 2.A) that f is convex. The convex dual $f^*(P)$ is the negative entropy $-H(P)$. Using (3.4), we have for each $P \in \Xi$:

$$\begin{aligned} \log P_0(\boldsymbol{x}_v) = f(\phi_{\mathbf{x}_v}) &\geq f_P(\phi_{\mathbf{x}_v}) = (\phi_{\mathbf{x}_v}, P) - f^*(P) \\ &= \int P(\boldsymbol{x}_h) \log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)\, d\boldsymbol{x}_h + H(P), \end{aligned} \tag{3.22}$$

which is just (3.21). We can also easily compute the induced metric, as introduced in subsection 3.1.1. Let P^* and $\phi_{\mathbf{x}_v}$ be a corresponding pair. Then:

$$\begin{aligned} D_{f^*}(P \to P^*) &= f(\phi_{\mathbf{x}_v}) - f_P(\phi_{\mathbf{x}_v}) \\ &= \log P_0(\boldsymbol{x}_v) - \int P(\boldsymbol{x}_h) \log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)\, d\boldsymbol{x}_h - H(P) \\ &= \int P(\boldsymbol{x}_h) \log \left(\frac{P(\boldsymbol{x}_h)}{P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)/P_0(\boldsymbol{x}_v)} \right) = D(P||P_0(\cdot|\boldsymbol{x}_v)), \end{aligned} \tag{3.23}$$

which is the relative entropy between the variational distribution P and the posterior $P_0(\boldsymbol{x}_h|\boldsymbol{x}_v) = P^*$. The convex duality framework therefore suggests to choose a family $\tilde{\Xi}$ of distributions such that $D(P||P_0(\cdot|\boldsymbol{x}_v))$ can be efficiently computed up to additive constants for each $P \in \tilde{\Xi}$, then to minimize the metric with respect to $P \in \tilde{\Xi}$. Since the relative entropy is convex, this optimization problem has a unique solution if $\tilde{\Xi}$ is convex. Otherwise, we can search for a local minimum. By definition, $D(P||P_0(\cdot|\boldsymbol{x}_v))$ is the difference between the exact log marginal $\log P_0(\boldsymbol{x}_v)$ (corresponding to the negative free energy) and (3.19) (corresponding to the negative variational free energy). The way via convex duality seems to be overcomplex in this case, but we think it gives a useful (graphical) intuition about the nature of the variational approximation (see figure 3.1).

3.1.4 From inference to learning: The EM algorithm

In the last section, we dealt with first-level inference. Given some evidence $\boldsymbol{x}_v$ and a *fixed* probability model $P_0(\boldsymbol{x}_v, \boldsymbol{x}_h)$, we derived a tractable approximation to $\log P_0(\boldsymbol{x}_v)$. Now we start from a *family* of models $P_0(\boldsymbol{x}_v, \boldsymbol{x}_h|\boldsymbol{\theta}),\ \boldsymbol{\theta} \in \Theta$ and try to infer $\boldsymbol{\theta}$ given evidence $\boldsymbol{x}_v$. From the Bayesian point of view, the best model is a sum over all members of the family, weighted by the posterior distribution of $\boldsymbol{\theta}$. However, if we are restricted to use one model out of the family, we have to look for another criterion of choice. The *maximum a-posteriori (MAP)* method is to choose $\boldsymbol{\theta}$ such as to maximize the posterior of $\boldsymbol{\theta}$. The basic challenge for MAP is to maximize the log likelihood function $\log P_0(\boldsymbol{x}_v|\boldsymbol{\theta})$, since the prior of $\boldsymbol{\theta}$ is usually easy to optimize. The *maximum-likelihood (ML)* method is to choose $\boldsymbol{\theta}$ to maximize the log likelihood, which corresponds to MAP with an improper prior. We discuss these methods in more detail in subsection 3.2.1. Another idea would be to sample $\boldsymbol{\theta}$ from the posterior, this is referred to as the *Gibbs* method. The Gibbs method has some theoretical advantages over the MAP method, but if the posterior

is highly concentrated around its mode, the methods behave similarly. The Gibbs method can be implemented using a Monte Carlo sampler.

The *expectation-maximization (EM) algorithm* is a general tool for finding local maxima of marginal likelihoods like $\log P_0(\boldsymbol{x}_v|\boldsymbol{\theta})$ (see [DLR77],[HN97]). It requires the computation of the expectation of the complete-data log likelihood $\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h|\boldsymbol{\theta})$ over the posterior $P_0(\boldsymbol{x}_h|\boldsymbol{x}_v, \boldsymbol{\theta}_0)$ and the maximization of this criterion w.r.t. $\boldsymbol{\theta}$. In some cases, this computation is feasible while the direct optimization of the marginal likelihood is not. However, usually the posterior expectation is also infeasible and has to be approximated, for example using the variational technique. Let $\tilde{\Xi}$ be a set of variational distributions such that the computation of $\mathrm{E}_P[\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h|\boldsymbol{\theta})]$ (as a function of $\boldsymbol{\theta}$) and its gradient is feasible for all $P \in \tilde{\Xi}$. Furthermore we require that the variational inference procedure discussed in subsection 3.1.3 is feasible for every $\boldsymbol{\theta}$. Then, a generalized form of the EM algorithm can be used to find a local maximum $(\hat{P}, \hat{\boldsymbol{\theta}})$ of the criterion

$$L(P, \boldsymbol{\theta}) = \mathrm{E}_P \left[\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h|\boldsymbol{\theta})\right] + H(P). \tag{3.24}$$

Recall that this is the variational lower bound to $\log P_0(\boldsymbol{x}_v|\boldsymbol{\theta})$. If the posterior $P_0(\boldsymbol{x}_h|\boldsymbol{x}_v, \boldsymbol{\theta})$ is in $\tilde{\Xi}$ for all $\boldsymbol{\theta}$, this method coincides with the original EM algorithm. However, it is very important to note that in general $\hat{\boldsymbol{\theta}}$ will not be a local maximum point of the marginal likelihood. If the conditional posteriors for hyperparameter values around the true maximizer can only be poorly represented by any of the variational distributions, the algorithm might prefer a value $\hat{\boldsymbol{\theta}}$ far from the true maximum point if the posterior conditioned on $\hat{\boldsymbol{\theta}}$ can be fitted much better by a variational density. We can argue that if $\tilde{\Xi}$ is a broad class, then $\hat{\boldsymbol{\theta}}$ should have a reasonably large likelihood, as compared to the scores attained by local maxima of the marginal likelihood function. Furthermore, our goal is usually *not* to obtain a close approximation to a maximum point $\boldsymbol{\theta}$, but to find a good proxy for the posterior

$$P_0(\boldsymbol{x}_h|\boldsymbol{x}_v) = \int P_0(\boldsymbol{x}_h|\boldsymbol{x}_v, \boldsymbol{\theta}) P_0(\boldsymbol{\theta}|\boldsymbol{x}_v)\, d\boldsymbol{\theta} \tag{3.25}$$

within the tractable class $\tilde{\Xi}$, and $\hat{P}$ is surely a reasonable candidate. We will first state the algorithm in the form given by [HN97] and afterwards show how it can be derived as special case of a very simple optimization algorithm for convex functions.

1. Choose any $\boldsymbol{\theta}_0$ as starting point.
2. Maximize $L(P, \boldsymbol{\theta}_{t-1})$ w.r.t. $P \in \tilde{\Xi}$. Let P_t be the maximum point, i.e. the best variational approximation to $P(\boldsymbol{x}_h|\boldsymbol{x}_v, \boldsymbol{\theta}_{t-1})$.

3. Maximize $L(P_t, \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$. Let $\boldsymbol{\theta}_t$ be the maximum point. Go to 2.

This procedure convergences against a local maximum of L. Both maximization steps need not be performed to high accuracy, especially during early iterations. The global maximum in step 2 is the posterior $P_0(\boldsymbol{x}_h|\boldsymbol{x}_v, \boldsymbol{\theta}_{t-1})$ which however is usually not contained in $\tilde{\Xi}$.

Recall the discussion of subsection 3.1.1. Suppose we want to find a local maximum of $f(\boldsymbol{x})$ over some subset $X_c \subset X$. A very simple algorithm for this purpose can be described as follows. Start with any $\boldsymbol{x}_0 \in X_c$. Given $\boldsymbol{x}_{t-1}$, find the corresponding $\boldsymbol{\xi}$. If f is differentiable (as we usually assume), we have $\boldsymbol{\xi} = \nabla f(\boldsymbol{x}_{t-1})$. Consider the affine lower bound f_ξ. Now, if we find a $\boldsymbol{x}_t$ such that $f_\xi(\boldsymbol{x}_t) > f_\xi(\boldsymbol{x}_{t-1})$, we have $f(\boldsymbol{x}_t) \geq f_\xi(\boldsymbol{x}_t) > f_\xi(\boldsymbol{x}_{t-1}) = f(\boldsymbol{x}_{t-1})$, and $\boldsymbol{x}_{t-1} \to \boldsymbol{x}_t$ means progress. This algorithm converges to a local maximum where $\boldsymbol{\xi} \approx \mathbf{0}$. One can show that the EM algorithm is a special case of this simple procedure.

Now suppose it is infeasible for general $\boldsymbol{x}_{t-1} \in X_c$ to find the corresponding $\boldsymbol{\xi}$ and/or compute the affine bound f_ξ. If there is a subset $\tilde{\Xi} \subset \Xi$ such that the bounds can be computed efficiently for all $\boldsymbol{\xi} \in \tilde{\Xi}$, the following algorithm can be used to find a joint local maximum of $f_\xi(\boldsymbol{x}),\ \boldsymbol{x} \in X_c, \boldsymbol{\xi} \in \tilde{\Xi}$:

1. Choose any $\boldsymbol{x}_0 \in X_c$.

2. Maximize $f_\xi(\boldsymbol{x}_{t-1})$ w.r.t. $\boldsymbol{\xi} \in \tilde{\Xi}$. This is equivalent to minimizing the metric $D(\boldsymbol{\xi} \to \boldsymbol{\xi}')$ where $\boldsymbol{\xi}'$ corresponds to $\boldsymbol{x}_{t-1}$ (and need not be in $\tilde{\Xi}$). Let $\boldsymbol{\xi}_t$ be the maximum point.

3. Maximize $f_{\xi_t}(\boldsymbol{x})$ w.r.t. $\boldsymbol{x} \in X_c$. Let $\boldsymbol{x}_t$ be the maximum point. Go to 2.

Figure 3.2 shows the operations involved in one step of the iteration, namely finding the tangent hyperplane closest (w.r.t. to the D_{f^*} metric) to the tangent at $\boldsymbol{x}_{t-1}$, and searching for a constrained maximum point on this (former) tangent. In this figure, the local maximum of f and the lower bound f_ξ are the same, but that is not true in general.

Much like in subsection 3.1.3, we specialize X_c to be the set of functions $\phi_\theta(\boldsymbol{x}_h) = \log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h|\boldsymbol{\theta})$ which can be identified with Θ. As above, we use $f(\phi) = \log \int \exp(\phi(\boldsymbol{x}_h))\, d\boldsymbol{x}_h$ which is convex. The dual space Ξ is the gradient space of f. It is easy to show that $f(\phi_\theta) = \log P_0(\boldsymbol{x}_v|\boldsymbol{\theta})$ and $\nabla f(\phi_\theta) = P(\boldsymbol{x}_h|\boldsymbol{x}_v, \boldsymbol{\theta})$, i.e. the posterior. We choose $\tilde{\Xi}$ as space of variational distributions $P(\boldsymbol{x}_h)$. Since the convex dual of f is $f^*(P) = -H(P)$,

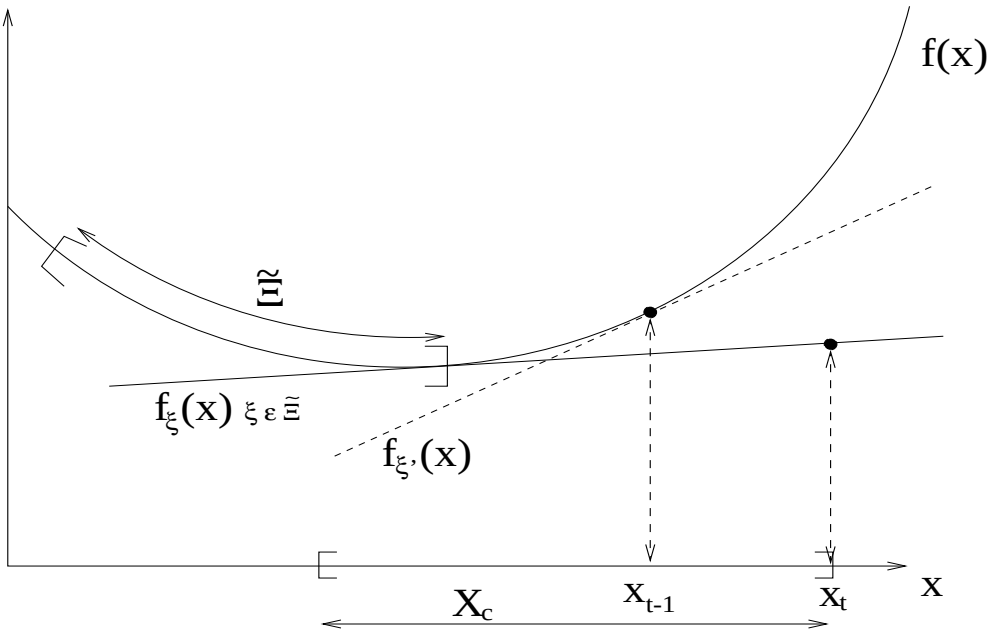

Figure 3.2: Iteration of convex maximization algorithm.

we have the affine bounds

$$f_P(\phi_\theta) = (P, \phi_\theta) + H(P) = \mathrm{E}_P \left[\log P_0(\boldsymbol{x}_v, \boldsymbol{x}_h|\boldsymbol{\theta})\right] + H(P) \tag{3.26}$$

which coincide with (3.24). This view should provide some useful graphical intuition about the nature of the generalized EM algorithm.

3.1.5 Minimum description length and the bits-back encoder

During this subsection, unlike all others, log denotes the logarithm to basis 2. Using the natural logarithm doesn't invalidate any statement here, but we would have to use the measurement unit "nats" instead of the familiar bits[2].

We can justify the variational technique from an information-theoretic viewpoint, applying the *minimum description length (MDL)* principle (see [Ris86]) and the *bits-back encoding scheme* of [HVC93]. MDL builds on *Kolmogorov complexity* (see for example [CT91]) which suggests to measure the stochastic complexity of a string of symbols by the length of the shortest computer program which prints the string and halts. Given that the decoder

[2]We would have to deal with a nats-back encoder then.

is an universal computer, transmitting the algorithmic description of a sequence instead of the sequence itself is clearly a universal optimal source code, which links Kolmogorov complexity to coding and information theory.

To compress a sequence substantially, we have to find hidden structures that can be described very much shorter than the sequence itself. This process is called *modeling*. Any nontrivial model will include different sources of randomness. First of all, adaptive models are usually parameterized by *weights*, representing hidden variables. When a model is used to generate data, the actual values of the hidden variables are sampled at random. The (prior) distribution of the weights is part of the structural model description. Secondly, there are "destructive" random sources called *noise*. From an information-theoretic viewpoint, noise transforms the exact predictions of the data-generating model into the actual sample, and by doing so *destroys* information about the values of the hidden variables in the model. The higher the "noise level", the more information is destroyed by the noise transformation. Since information is a symmetric measure, the noise also destroys information the model weights contains about the data. The noise distributions are also part of the structural model description. We conclude that a model resulting in an efficient description of a dataset has to be chosen such that

1. There is a short description for the weights.
2. The noise level is low, i.e. given the weights (and therefore the deterministic predictions of the model), there is a short description of the data.

Within Kolmogorov complexity theory, the description of the model structure and weights is called *Kolmogorov sufficient statistic* (see [CT91], chapter 7): It achieves a description of the sequence up to (structureless) pure chance. The idea of a trade-off between model complexity and accurate (low noise) description of the data is sometimes called *Occam's razor* (see [Mac91],[Mac95]). It is transcendent in natural science, stating that of two theories describing an empirical phenomenon equally well, the conceptually easier one should always be preferred. Occam's razor is also the basis of *Occam learning* in COLT (see [KV94]).

It therefore makes sense to prefer models which, if their structure is known to both sender and receiver, allow to construct the shortest possible description of given data. Suppose we are given a structure model $\mathcal{H}$, parameterized by a weight vector $\boldsymbol{w}$, and a dataset $\tilde{D}$. The goal is to transmit a short description of $\tilde{D}$ (which is quantized using equidistant bins of length δD)

from a sender S to a receiver R. Both S and R know $\mathcal{H}$ and δD. We will show that both the negative marginal log likelihood $-\log P(\tilde{D}|\mathcal{H})$ and its variational approximation are (hopefully reasonably tight) upper bounds on the minimum description length of $\tilde{D}$ given $\mathcal{H}$[3]. The MDL principle thus justifies the usage of these criteria for model selection.

The bits-back encoding scheme of [HVC93] has the reputation of being somewhat difficult to understand, because of the reason that S can (and has to) actually transfer *more* information to R than just a unique description of $\tilde{D}$. But looking at real-world situations of how a communication channel is used, this assumption is very natural. Thus, assume there is another random variable $I \sim P(I)$, independent of the random sources named above, and the prior $P(I)$ is known to S and R. At the beginning of the communication, S samples a value I, and S wants to tell R about this value, *apart from* transmitting $\tilde{D}$. The statement "transmitting k bits of information about I" can be made concrete by assuming that both parties know a partition of the range of I into 2^k subsets, each having mass 2^{-k} under the measure $P(I)$. S can tell R about the subset I lies in by sending the k-bit index of the subset, and there is no shorter message inducing the same knowledge level at the side of R. For example, if $I \sim U([0,1))$, S might just send the first k bits of the binary expansion of I.

If T is a value in the range of a discrete random variable with distribution $P(T)$, we will use the statement "T can be encoded using $-\log P(T)$ bits". This is not strictly true, but there always exist codes with word length $< -\log P(T) + 2$ bits for T, as shown in [CT91]. Conversely, we can use one of these optimal coding trees to generate a sample T from $P(T)$ using a sequence of fair coins. Given that the outcome is $T = t$, the number of random bits we actually used is also $< -\log P(t) + 2$. However, universal constants can be ignored within the MDL framework.

The bits-back coding scheme works as follows:

1. S computes the posterior distribution $P(\boldsymbol{w}|\tilde{D})$ and samples a weight vector $\tilde{\boldsymbol{w}}$ from this distribution, quantized to equidistant bins of width $\delta\boldsymbol{w}$. The sampling requires $-\log(P(\tilde{\boldsymbol{w}}|\tilde{D})\delta\boldsymbol{w})$ unbiased random bits. Here, $\delta\boldsymbol{w}$ is the quantization width for weights upon which S and R agreed beforehand.

2. S encodes $\tilde{\boldsymbol{w}}$ using the (quantized) prior distribution $P(\boldsymbol{w})$. This needs $-\log(P(\tilde{\boldsymbol{w}})\delta\boldsymbol{w})$ bits.

[3] Up to a constant only depending on the quantization width δD.

3. S encodes the data $\tilde{D}$ using the (quantized) distribution $P(D|\tilde{\boldsymbol{w}})$. This needs $-\log(P(\tilde{D}|\tilde{\boldsymbol{w}})\delta D)$ bits. S sends both messages, at a total cost of

$$-\log(P(\tilde{\boldsymbol{w}})\delta\boldsymbol{w}) - \log(P(\tilde{D}|\tilde{\boldsymbol{w}})\delta D) \text{ bits.} \tag{3.27}$$

4. R decodes $\tilde{\boldsymbol{w}}$ using the prior $P(\boldsymbol{w})$ and $\tilde{D}$ using the noise distribution $P(D|\tilde{\boldsymbol{w}})$.

Now, something is wrong here. The cost (3.27) depends on the quantization $\delta\boldsymbol{w}$ and is huge for reasonably small accuracy. But note that R, after having decoded $\tilde{\boldsymbol{w}}$ and $\tilde{D}$, is able to reconstruct the sequence of bits S used to sample $\tilde{\boldsymbol{w}}$, by computing the posterior $P(\boldsymbol{w}|\tilde{D})$ and building the same code tree used by S to construct $\tilde{\boldsymbol{w}}$. Now, what prevents S from using these bits to send another message to R, containing information about I? The bits S uses to sample $\tilde{\boldsymbol{w}}$ have to be fair coins from the viewpoint of R only (not knowing I), and can therefore be used to communicate $-\log(P(\tilde{\boldsymbol{w}}|\tilde{D})\delta\boldsymbol{w})$ bits of information. Thus, we get a lot of *bits back*, and the cost of describing $\tilde{D}$ is only

$$\begin{aligned} &-\log(P(\tilde{\boldsymbol{w}})\delta\boldsymbol{w}) - \log(P(\tilde{D}|\tilde{\boldsymbol{w}})\delta D) + \log(P(\tilde{\boldsymbol{w}}|\tilde{D})\delta\boldsymbol{w}) \\ &= -\log P(\tilde{D}) - \log\delta D. \end{aligned} \tag{3.28}$$

But what if the computation of the posterior $P(\boldsymbol{w}|\tilde{D})$ is not feasible[4]? Suppose there is a class $Q(\boldsymbol{w}|\boldsymbol{\theta})$ of approximating distributions such that the computation of a certain distance between $Q(\boldsymbol{w}|\boldsymbol{\theta})$ and the posterior $P(\boldsymbol{w}|\tilde{D})$ is feasible for every $\boldsymbol{\theta}$. We will specify the distance afterwards. We now apply the bits-back scheme as above, but replace the true posterior by the best approximation $Q(\boldsymbol{w}|\tilde{\boldsymbol{\theta}})$. The apparent cost is given by (3.27), but we get $-\log(Q(\tilde{\boldsymbol{w}}|\tilde{\boldsymbol{\theta}})\delta\boldsymbol{w})$ bits back, and the cost of describing $\tilde{D}$ is (for fixed $\tilde{\boldsymbol{w}}$)

$$\begin{aligned} &-\log(P(\tilde{\boldsymbol{w}})\delta\boldsymbol{w}) - \log(P(\tilde{D}|\tilde{\boldsymbol{w}})\delta D) + \log(Q(\tilde{\boldsymbol{w}}|\tilde{\boldsymbol{\theta}})\delta\boldsymbol{w}) \\ &= -\log P(\tilde{D}) - \log\delta D - \log\frac{P(\tilde{\boldsymbol{w}}|\tilde{D})}{Q(\tilde{\boldsymbol{w}}|\tilde{\boldsymbol{\theta}})}. \end{aligned} \tag{3.29}$$

Since $\tilde{\boldsymbol{w}} \sim Q(\boldsymbol{w}|\tilde{\boldsymbol{\theta}})$ (if I is unknown), the expected description length is

$$-\log P(\tilde{D}) - \log\delta D + D(Q(\cdot|\tilde{\boldsymbol{\theta}})||P(\cdot|\tilde{D})) \tag{3.30}$$

[4]Since MDL is not concerned about feasibility, we simply restrict the model (and encoder and decoder) such that only certain *admissable* operations are allowed.

which is easily recognized to be the variational free energy (3.19) using the variational distribution $Q(\cdot|\tilde{\boldsymbol{\theta}})$, up to a δD term. The best way to choose $\tilde{\boldsymbol{\theta}}$ is to minimize the relative entropy in (3.30) or, equivalently, (3.30) itself, which fixes the distance left unspecified above.

Hinton and van Camp [HVC93] called this special case of the variational technique *ensemble learning*. Instead of using a single fixed weight, we employ an ensemble whose weights are given by the variational distribution $Q(\cdot|\tilde{\boldsymbol{\theta}})$. The optimal ensemble is the Bayesian one, using the posterior itself as weighting distribution. If $\boldsymbol{w}$ is continuous, the most common variational distributions are Gaussians. We can also employ mixtures of Gaussians (see [HVC93]). In the latter case, the variational free energy is not analytically tractable, but there is a simple upper bound that can be minimized instead.

3.2 Bayesian techniques

We only give a short discussion here such as to render the following chapter self-contained. There are a lot of good sources for further reading, e.g. [Bis95],[Mac91],[Nea96],[Mac95]. And if you have a really tricky question, you are encouraged to visit David MacKay's well-maintained *Bayesian FAQ* page at: `http://wol.ra.phy.cam.ac.uk/mackay/Bayes_FAQ.html`

3.2.1 The evidence framework

As discussed in the introduction above, the Bayesian framework requires us to specify prior distributions over latent variables, like for example weights of a network. Priors are usually constructed in a *hierarchical* fashion (see section 1.3.2). We choose a family of possible prior distributions, parameterized by a *hyperparameter*. Specifying a *hyperprior* over this parameter induces an effective prior over the latent variables. This procedure can be iterated, i.e. we can write the hyperprior as a parametric mixture too, and so on. An advantage of the hierarchical approach is that we can generate very complicated effective priors using simple manageable hyperpriors. A disadvantage is that we have to deal with more variables and more (possibly intractable) Bayesian sums or integrals. Consequently, more approximations have to be employed. However, as we will see, some of these approximations reveal another advantage of hierarchical design, namely the adaption of hyperparameters to the data. This seems odd, since the priors should be chosen before any data is inspected, but it will become clear that the priors are indeed specified

completely before any data comes into play. What is adapted is merely a "working approximation" to the hyperprior.

The evidence framework proposed in [Mac91] shows how to perform an approximate Bayesian analysis using hierarchical priors. For simplicity, we will assume a one-stage hierarchy. Following [WB98], we will show how to apply the framework to probabilistic kernel regression classifiers. Recall the material of subsection 1.3.2. We choose a hyperprior $P(\boldsymbol{\theta})$. Details about the choice of hyperpriors are given in a seperate subsection below. If $P_a(t_*|D, \boldsymbol{\theta})$ is an approximation to the conditioned predictive distribution, we can average this distribution over the posterior $P(\boldsymbol{\theta}|D)$ to get an approximation to the true predictive distribution $P(t_*|D)$.

For reasonably complex models, we can neither normalize the posterior nor compute the expectation. However, we can search for the *maximum a-posteriori (MAP)* value $\hat{\boldsymbol{\theta}} = \text{argmax}_\theta \, P(\boldsymbol{\theta}|D)$ and approximate the predictive distribution by $P_a(t_*|D, \hat{\boldsymbol{\theta}})$. This is sometimes referred to as *maximum likelihood II* approach (see [Ber85]). In the limit of large datasets, $P(\boldsymbol{\theta}|D)$ converges in probability against a delta distribution, and the maximum points for a sequence of growing datasets converge in probability against the position of the peak, so the approximation is asymptotically exact. Much more important is that the procedure often works well in practice for finite, small dataset sizes, i.e. $P_a(t_*|D, \hat{\boldsymbol{\theta}})$ can be used as good approximation of the true predictive distribution. However, MAP has some severe caveats we have to be aware of. The method is *not* invariant against nonlinear transformations of the hyperparameter random variable, except in the special case of *maximum likelihood (ML)* which corresponds to using a constant improper prior on the hyperparameters. Let $\hat{\boldsymbol{\theta}}$ be the maximizer of the posterior $P(\boldsymbol{\theta}|D)$. If f is a nonlinear invertable function, $\boldsymbol{\gamma} = f(\boldsymbol{\theta})$ and $\hat{\boldsymbol{\gamma}}$ the maximizer of $P(\boldsymbol{\gamma}|D)$, then (in general) $\hat{\boldsymbol{\gamma}} \neq f(\hat{\boldsymbol{\theta}})$. Many Bayesians do not consider MAP a Bayesian technique for this reason (among others), and we have to repeat that a sound justification can only be given in the limit of large datasets. In the case of a finite, maybe rather small dataset it is not possible to recommend a particular kind of parameterization, since the best choice depends on the model. The parameterization should be chosen such that the posterior is not skew, although this is hard to test in complex models. Plots of $P(\boldsymbol{\theta})$ and $-\log P(\boldsymbol{\theta})$ (as part of the negative log posterior criterion to be minimized) are helpful to visualize the contribution of the prior to MAP in contrast to ML. The performance of the MAP discriminant might also be compared to MCMC prediction if the latter can be performed feasibly. A gross degradation in performance might indicate a poor parameterization of the hyperparameters, but other possible reasons are shortcomings in the

model or poor quality of approximations. Another possibility to detect such shortcomings is indicated in subsection 3.2.4.

We remark that exactly the same problem occurs with the parameterization of latent variables if we determine a MAP configuration and predict using this configuration alone instead of averaging all configurations. In the case where Gaussian approximations like the Laplace technique or certain variational methods are involved, one should always aim to choose the parameterization of the latent variables such that the posterior is as much Gaussian-shaped as possible. In certain special cases, an optimal choice can be advocated (see [Mac98]).

Given that the log hyperprior $\log P(\boldsymbol{\theta})$ can easily be maximized w.r.t. $\boldsymbol{\theta}$, the basic challenge of MAP is to maximize the *log evidence* for D:

$$\log P(D|\boldsymbol{\theta}) = \log \int P(\boldsymbol{t}, \boldsymbol{y}|\boldsymbol{\theta})\, d\boldsymbol{y} \tag{3.31}$$

Unfortunately, there is a name clash with the *graphical models* literature which would call D the evidence and $P(D|\boldsymbol{\theta})$ the marginal likelihood of the evidence (see subsection 3.1.4). The basic tool for evidence maximization is the EM algorithm, as discussed in subsection 3.1.4, where the posterior $P(\boldsymbol{y}|D, \boldsymbol{\theta}_{t-1})$ in the E step is either computed exactly (like in Gaussian process regression with Gaussian noise, see subsection 1.3.4) or approximately (see subsection 1.3.5).

Note that, as discussed in subsection 3.1.5, the evidence maximization approach can also be motivated from an MDL viewpoint, i.e. using arguments of information-theoretic origin.

3.2.2 Monte Carlo methods

Monte Carlo techniques are of immense importance in a growing number of fields including statistical physics and probabilistic inference. Consequently, the literature is huge, and attempting a review here is beyond our scope. Neal [Nea93] gives a detailed review and Neal [Nea96] is an excellent source for Bayesian learning for neural networks using *Markov Chain Monte Carlo (MCMC)* methods. Radford Neal provides free, sophisticated MCMC software for neural networks and Gaussian processes at `http://www.cs.toronto.edu/~radford`, and BUGS is a free package for Gibbs sampling[5] (see `http://www.mrc-bsu.cam.ac.uk/bugs/`).

[5] *Gibbs sampling* is a particularly simple and very popular kind of MCMC technique.

If $\boldsymbol{x} \sim P(\boldsymbol{x})$ and P and f are sufficiently regular, then

$$\frac{1}{N}\sum_{n=1}^{N} f(\boldsymbol{x}_n) \to \mathrm{E}[f(\boldsymbol{x})] \tag{3.32}$$

with probability 1, if the $\boldsymbol{x}_i$ are independent samples from P. Moreover, the variance of the left-hand side is (in probability) $O(1/N)$ if f is bounded, thus the speed of convergence is independent of details like the dimension of $\boldsymbol{x}$. Generating independent samples from P is usually difficult, but if we generate a sequence of dependent samples in a way such that points far away from each other in the sequence are "quasi" independent, (3.32) still works, either by using all dependent points to compute the average or by picking points mutually far from each other in the sequence and use the subsequence only. Of course, the $O(1/N)$ convergence behaviour is usually lost. This argument can be made precise, by estimating the correlation function of the process $(f(\boldsymbol{x}_i))_i$ (see [Nea93]).

Another problem is how to sample from P. Often, P cannot even be normalized, and even if $P(\boldsymbol{x})$ can be computed efficiently for every $\boldsymbol{x}$, it is not clear how to produce independent samples from P. MCMC uses an ergodic Markov Chain with equilibrium or stationary distribution P. After a burn-in phase, the chain settles down and produces (dependent) samples from its equilibrium distribution. This sounds fairly easy, but the problem of MCMC sampling remains a challenging one. No rigorous, general criterion to determine the length of the burn-in phase has been found so far, although criteria have been proposed which work under special conditions (see [PW96]). Cowles and Carlin [CC96] give an overview over practical convergence diagnostics. Furthermore, the lags between "quasi" independent points in the sequence can usually only be strongly overestimated. Techniques to systematically decrease the length of the burn-in phase and the lags have been proposed. While they work for large classes of distributions, P in a concrete problem often has extremely nasty properties[6] (for example if P is a posterior), and one can never be sure that the MCMC sampler doesn't fail completely for P. However, as with so many MC algorithms, MCMC is competetive in situations where everything else that is known is *definitively* incorrect or infeasible.

3.2.3 Choice of hyperpriors

Let us begin by classifying parameters of a univariate density. If $x \sim p(x|\theta)$ and $x + a \sim p(x|\theta + a)$ for any a, θ is called *location parameter*. For example,

[6] One would not need MCMC if P was simple.

the mean is a location parameter of a Gaussian. If $x \sim p(x|\rho)$ where $\rho > 0$, and if $ax \sim p(x|a\rho)$ for any $a > 0$, ρ is called *scale parameter*. The standard deviation of a zero-mean Gaussian or the mean of a Gamma distribution are examples for scale parameters. If $x \sim p(x|\theta, \rho),\ \rho > 0$, θ is a location parameter for every fixed ρ and $ax \sim p(x|a\theta, a\rho)$ for every $a > 0$, θ and ρ are a location-scale parameter pair.

The most commonly used hyperprior distributions are *noninformative priors*. By definition, these priors contain no information at all about the parameter, and usually they are not even distributions, since their integral over the range of the parameter is not finite. Such priors are often called *improper*. The use of improper priors is heavily criticized by Bayesians, since such a procedure makes no sense at all within the Bayesian philosophy. Improper priors are a constant source of errors in applications of "Bayesian" inference, for example an MCMC implementation will almost surely fail if improper priors are used. Other approaches like the evidence framework discussed above are more robust against this misuse, but in general improper priors should be regarded merely as a concession to people who criticize the subjectivity in Bayesian analysis, and anybody else should try to avoid them whenever possible. The standard improper prior for location parameters is the flat uniform one $p(\theta) \equiv 1$. Noninformative priors for scale parameters should be uniform on a log scale, i.e $p(\log \rho) \equiv 1$ or $p(\rho) = 1/\rho$. Both priors cannot be normalized. These choices can be motivated using arguments about invariances, see [Bis95], [Ber85].

The use of *conjugate priors* is very popular in Bayesian statistics. A class of distribution Γ is called *conjugate* w.r.t. a family of distributions $P(\boldsymbol{x}|\theta)$ if for any choice $P(\theta) \in \Gamma$ and any $\boldsymbol{x}$ the posterior $P(\theta|\boldsymbol{x})$ is also in Γ. For example, if a class of prior distributions is conjugate w.r.t. the family of noise distributions of a model, the posterior distribution is in the same class as the prior, and the parameters usually can be determined easily. An example would be Gaussian process regression with Gaussian noise, see subsection 1.3.4. In contrast to that, there are virtually no cases of analytically tractable exact Bayesian analysis of a model with nonconjugate prior. If we use hyperpriors conjugate w.r.t. the conditional prior distributions over latent variables, we can compute the posterior of the hyperparameter given the value of the latent variables, but the posterior of the hyperparameter given the data usually remains intractable. There is an ongoing discussion between advocates of the evidence framework and statisticians advising the use of conjugate hyperpriors and early marginalization over the hyperparameters, see [Mac93]. This might seem strange: Why should we approximate an integration crudely if we can do it analytically? The reason why intuition might

go wrong here is that the true priors and likelihoods, obtained by integrating the hyperparameters out, are skew and usually give rise to *skew-peaked* posteriors. Applying a Gaussian approximation to such a posterior might introduce a larger error than approximating the (usually) more symmetric conditional posterior and plug in the MAP hyperparameter values.

Even though the choice of hyperpriors remains a subjective one and depends on the concrete situation, there are some broad and very useful classes that can be considered. Gaussian distributions are suitable as hyperpriors for location parameters. Their use can always be motivated using the maximum entropy principle (see subsection 1.3.3). Furthermore Gaussian distributions $P(\theta)$ are conjugate w.r.t. conditional Gaussian distributions of the form $N(\theta\mathbf{1}, \sigma^2\boldsymbol{I})$.

Scale parameters can be given Gaussian hyperpriors on the log scale, i.e. $P(\log\rho)$ is Gaussian. An alternative is to use a Gamma prior on $\tau = \rho^{-2}$. If ρ^2 is a variance parameter, for example of a Gaussian prior, τ is called *precision* parameter. The $\Gamma(\mu, \alpha)$ distribution is defined as

$$P(\tau|\mu,\alpha) = \frac{(\alpha/2\mu)^{\alpha/2}}{\Gamma(\alpha/2)}\tau^{\alpha/2-1}\exp\left(-\frac{\alpha}{2\mu}\tau\right)I_{\{\tau>0\}}. \tag{3.33}$$

It has mean μ and variance $2\mu^2/\alpha$. Gamma hyperpriors can be motivated using a conjugacy argument, see [Nea96], chapter 3.2.1. If some hidden variables are conditionally i.i.d. zero-mean Gaussian with precision τ, and the latter is given a Gamma hyperprior, the effective prior (obtained by integrating τ out) is a t distribution, see [Nea96], chapter 4.1.

We finally remark that the parameters of hyperpriors have to be given concrete values when using a one-level design. This can be done by looking at graphs of the hyperprior density for different values. Unless we have specific prior information, hyperpriors should be chosen such that their mass is broadly uniformly distributed over a broad range of plausible values.

3.2.4 Evidence versus Cross Validation

This subsection broadly follows a discussion on David MacKay's *Bayesian FAQ* page at `http://wol.ra.phy.cam.ac.uk/mackay/Bayes_FAQ.html`.

A central issue when analyzing a statistical pattern recognition model is to gain a good estimate of the generalization error. Let $(\boldsymbol{x}, t)$ be distributed according to an unknown data distribution. Let D be an i.i.d. sample of size n, drawn according to the data distribution. $g_n(\boldsymbol{x}, D)$ denotes a classifier

trained on the sample D. The *error* of g_n is defined as $L_n = P(g_n(\boldsymbol{x}, D) \neq t)$. L_n is a random variable depending on D. The *generalization error* is defined as $\mathrm{E}L_n$, where the expectation is over the ensemble of samples of size n. The generalization error depends on the sample size n.

An estimator of this quantity is a function[7] depending on the training sample D and the model description, but *not* on the unknown, data-generating distribution. An estimator is called *unbiased* if its expected value (over the ensemble of samples) is equal to the generalization error. The *cross validation (CV) estimator*, also called *rotation estimator* (see [DGL96], chapters 24 and 31), is an unbiased estimator of the generalization error. It works by partitioning the sample into $K > 1$ chunks D_k. For simplicity, let n/K be an integer and all the chunks have the same size. For any k, we train the machine on the sample $D \setminus D_k$ and compute the test error on D_k. These quantities are unbiased estimators of $\mathrm{E}L_{n-n/K}$, as is their arithmetic average, the rotation estimate. In the extreme case $K = n$, the estimator is called *leave-one-out cross validation estimator* or *deleted estimator*. Note that the estimator sacrifices no training points for validation purposes only (as does for example the *holdout* or *validation set estimator*, see [DGL96]). However, the computational cost is in general much larger than what is needed for the holdout estimator, therefore the latter is often preferred in practice.

The CV estimator can be used for general loss functions. If $y_n(\boldsymbol{x}, D)$ is a predictor of y and g a loss function, we define the sample risk $L_n = \mathrm{E}[g(t, y_n(\boldsymbol{x}, D))]$ where the expectation is over $(\boldsymbol{x}, t)$. The risk is $\mathrm{E}_D L_n$ and depends on n. The leave-one-out CV estimator is

$$\hat{L}_n = \frac{1}{n} \sum_{i=1}^{n} g(t_i, y_{n-1}(\boldsymbol{x}_i, D \setminus \{(\boldsymbol{x}_i, t_i)\})), \tag{3.34}$$

and clearly $\mathrm{E}\hat{L}_n = \mathrm{E}L_{n-1}$. For reasonably large n, this should be close to $\mathrm{E}L_n$ which is the risk we are really interested in. We will concentrate on leave-one-out CV here.

Now let $\mathcal{H}$ be a model and $P(D|\mathcal{H})$ its evidence. By factorization we obtain

$$-\log P(D|\mathcal{H}) = \sum_{i=1}^{n} -\log P(t_i|t_1, \ldots, t_{i-1}, \mathcal{H}). \tag{3.35}$$

Thereby, we have $n!$ possibilities to order the sample points. One can plot i versus $-\log P(t_i|t_1, \ldots, t_{i-1}, \mathcal{H})$ which will typically be a decreasing curve

[7] Or, more generally, a sequence of functions, one for each sample size n.

(usually not monotonically) if the model is any worth. The curve is flat only for models that don't adapt to new data at all. The negative log evidence is the "integral" under the curve, while leave-one-out CV averages the *last* term in the sum over different orderings. Therefore, MacKay concludes that negative log evidence and leave-one-out CV are fundamentally different measures, and the latter is expected to render a better estimate of $\mathrm{E}L_{n-1}$.

However, the evidence measure has some apparent advantages over the CV score. The latter is subject to noise, and in general gradients w.r.t. model parameters cannot be computed. If we have a model which performs poorly on unseen data, CV might indicate this fact, but gives no suggestion for *how* the model should be changed to achieve better performance. Indeed, optimization of the CV score is usually done by trial-and-error, using grid searches, and this is clearly infeasible for more than a very small number of model parameters. Second, the naive way to compute the CV score is demanding, rendering it impractical on large datasets. For special loss functions or linear architectures, CV scores can be computed efficiently, see for example [Wah90]. In the general case, computationally cheap upper bounds on the CV score can sometimes be used, see [JH99], [Wah98].

MacKay concludes that in practice it is advisable (if it proves feasible) to use both scores, namely to use the CV score to "validate" the evidence. Suppose we variate certain parameters of the model and compute both scores for all configurations. If we assume that the CV score is a good estimate of the risk, and we observe a poor correlation between evidence and CV score, we can conclude that evidence and risk are in disagreement. This might indicate a *structural deficiency* of the model. Unless we have a good idea about the Bayes risk of the data distribution[8], information of this kind is difficult to obtain using one of the methods alone. MacKay [Mac91] observed this behaviour in experiments on real-world data.

[8]The Bayes risk (or error) is the infimum of $\mathrm{E}[L_n]$ over all (possibly randomized) predictors.

Chapter 4

Bayesian model selection for kernel regression classifiers

This chapter introduces a new variational Bayesian method for model selection among kernel regression classifiers which is applied to Gaussian process and Support Vector classifiers. The previous chapter paves the way for the techniques used here, but can be used in an encyclopedical fashion.

We compare the new method with several existing techniques for model selection. We finally present results of experiments done on a number of commonly used real-world tasks and analyze these results.

4.1 Model selection techniques

Given a set of models $\mathcal{H}$, we would like to select one that is supported most by given data D, in the sense that it is able to predict future data from the same source as D with smallest error or loss. Model selection is usually applied if we are constrained to use *one* model out of the hypothesis space. If this constraint is not given, mixtures over a subset or all of the hypothesis space should be preferred (see [Bis95], section 9.6). The ideal weights in such a mixture are posterior probabilities of the models, but many other weighting schemes also do a good job. A particularly interesting technique to build up a mixture is AdaBoost (and relatives), as discussed in subsection 2.2.1.

Model selection for kernel regression classifiers focusses on adapting free kernel hyperparameters to given data. While for Gaussian process classifiers, there are Bayesian approaches that usually work well in practice, model selection is an important current issue in Support Vector learning. For a ker-

nel with a very small number of free parameters (e.g. a Radial Basis Function (RBF) kernel) techniques like cross validation (see subsection 3.2.4 and [Wah98], [WLZ99]) can be applied. Wahba [WLZ99] shows how certain proxies to the *generalized approximative cross validation* score can be obtained without the need to actually run leave-one-out cross validation (which would be quite expensive for large training set sizes). Jaakkola and Haussler [JH99] use an upper bound one the leave-one-out score which can be evaluated efficiently. Campbell et al [CCST98] propose a method based on the *Kernel Adatron* algorithm to adapt the width parameter of a RBF kernel to the data, but they only consider the hard margin case which behaves very badly for noisy data. Opper and Winther [OW99] use mean field techniques to efficiently compute an approximation to the leave-one-out estimator.

Much recent interest is focussed on data-dependent PAC bounds (see discussion in subsection 1.2.4). [SSTSW99] Although these bounds are usually quite loose, they can be used for model selection, in much the same way as CV scores. The bounds are usually based either on Structural Risk Minimization and the luckiness framework (see subsection 1.2.4) or the idea of *skeleton estimates*. *Empirical covering* is an alternative to SRM with luckiness functions, but is not practically feasible at the time. See [DGL96] for details.

4.1.1 Bayesian model selection: Related work

Consider the evidence (as discussed in subsection 3.2.1) for a kernel regression model:

$$\begin{aligned} P(\boldsymbol{t}|\boldsymbol{\theta}) &= \int \exp\left(-\sum_{i=1}^{n} g(t_i, y_i)\right) P(\boldsymbol{y}|\boldsymbol{\theta})\, d\boldsymbol{y} \\ &= \int \exp\left(-\sum_{i=1}^{n} g(t_i, y_i)\right) N(\boldsymbol{y}|0, \boldsymbol{K}(\theta))\, d\boldsymbol{y}. \end{aligned} \tag{4.1}$$

We will restrict ourselves to normalized loss functions $g(t_i, y_i)$. If our loss function is unnormalized, we can find a normalized approximation. A possible way to find such an approximation is simply to normalize g, i.e. to replace it by $g(t_i, y_i) + \log Z(y_i)$ where $Z(y_i) = \exp(-g(+1, y_i)) + \exp(-g(-1, y_i))$. The reader might notice that we dropped the "noise variance" parameter C which appears at various places in this thesis as a prefactor of g. In the SVC case, we decided to fix $C = 1$ such that the SVC loss has the same behaviour for large $|y|$ as other common classification losses (see subsection

1.2.6). We leave the issue of a variable C (and therefore two different variance parameters, namely C and the prior variance parameter) for future work. For large or very small C, the normalized SVC loss (and all other normalized classification losses discussed in subsection 1.2.6) is far from the unnormalized one in the margin region. The latter one actually approaches an infinite step at 1 (being the "loss" used by the hard margin Support Vector machine) for large C. Intuitively we would expect to introduce more problems with degeneracies than benefits by using two seperate variance parameters, but practical exploration is required.

Bayesian model selection amounts to maximizing the log evidence plus a log prior for $\boldsymbol{\theta}$. This can be done using the EM algorithm, employing exact or approximate posterior distributions of the latent variables, as discussed in subsection 3.1.4.

One point is worth commenting on here. What happens if we just use the "evidence" (4.1) together with an unnormalized loss function? This will in general introduce a *systematic error* into the framework, because $\int P(D|\boldsymbol{\theta})\, dD$ depends on $\boldsymbol{\theta}$ and the training set size n. See [Sol00] where this effect is demonstrated on a toy example. Just normalizing (4.1) by this factor might help, but the resulting "evidence" is not consistent w.r.t. marginalization, as discussed in subsection 2.1.8. This inconsistency destroys the Bayesian justification of the "evidence" as model selection criterion.

Given the evidence criterion $P(D|\boldsymbol{\theta})$, how can we evaluate and maximize it? A straightforward Laplace approximation like in the Gaussian process classification case (as will be discussed in subsection 4.3.1, see also [WB98]) is the simplest option for most loss functions $g(t_i, y_i)$. However, for the SVC loss it is not applicable since the latter is not differentiable at $t_i y_i = 1$, and this fact cannot be ignored: With very high probability at least one of the patterns will lie on the margin in the final solution, i.e. $t_i \hat{y}_i = 1$[1]. Sollich [Sol99],[Sol00] suggests an approximation to $P(D|\boldsymbol{\theta})$ which works for the unnormalized SVC loss, but cannot be applied to the normalized one. The advantage of this formula is that it can be quickly evaluated given the SVC solution with a rather small number of Support Vectors since it requires the inversion of a matrix of size m only, where m is the number of essential Support Vectors. The approximation can be modified to result in a continuous expression (P. Sollich, personal communication), but without being able to compute a gradient the optimization of this formula remains difficult. Sollich presently uses sampling techniques like Monte Carlo chaining to deal with the normalization factors.

[1] We called the corresponding $\boldsymbol{x}_i$ *essential Support Vectors* above.

Another technique was proposed in [Kwo99]. The author replaces the step in the Support Vector loss by a differentiable proxy and then uses an Eigenvalue decomposition of the covariance matrix $\boldsymbol{K}$. This is closely related to the Laplace approximation discussed above. We (independently) spent some time to do a similar thing, namely replacing the SVC loss by a series of functions uniformly convergent to the loss. Performing the Laplace approximation and taking the limit leads (of course) to divergence. Progress can be made by subtracting off a term that only depends on the approximating series and (unfortunately) the number of patterns with $t_i\hat{y}_i \neq 1$. Having done this, the limit is finite and looks quite similar to the formula in [Kwo99]. However, the limit of the Laplace approximations using the functions of the series is a degenerate Gaussian whose variance is zero in the directions i with $t_i\hat{y}_i = 1$, and we finally considered the Laplace approximation not a valid technique in this case. Furthermore, Kwok [Kwo99] uses an unnormalized joint distribution, which might lead to systematically wrong results of the second-level inference, as discussed in subsection 4.1.1.

4.2 A variational technique for model selection

We suggest a *variational* method based on the idea of *ensemble learning* [HVC93]. The latter technique has been successfully applied to Radial Basis Function networks [BS97] and Multi-layer Perceptrons [BB97]. The basics of variational techniques are discussed thoroughly in the previous chapter, but the derivation here is (somewhat) self-contained.

4.2.1 Derivation of the algorithm

The log evidence cannot be efficiently evaluated, because we can neither normalize the posterior $P(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta})$ nor compute expectations over this distribution. However, approximating the posterior by a variational distribution $\tilde{P}$ from a suitably restricted model class Γ leads to a lower bound on the log evidence whose value and gradient w.r.t. $\boldsymbol{\theta}$ can be evaluated efficiently. Within the class, the models are ranked using the (differential) relative entropy (or *Kullback-Leibler divergence*): For fixed kernel parameters $\boldsymbol{\theta}$, we choose the variational distribution that minimizes the relative entropy to the true posterior. This minimization is equivalent to the maximization of the lower bound w.r.t. the variational distribution, and therefore feasible.

We referred to the negative log evidence as *free energy* and to the negative of the lower bound as *variational free energy* in the previous chapter, in accordance to the physical origins. The latter is, for the variational distribution $\tilde{P}$:

$$F(\tilde{P}, \boldsymbol{\theta}) = \mathrm{E}-\log P(\boldsymbol{t}, \boldsymbol{y}|\boldsymbol{\theta}) - H(\tilde{P}) \tag{4.2}$$

where $H(\tilde{P}) = \mathrm{E}-\log \tilde{P}(\boldsymbol{y})$ denotes the *differential entropy* of $\tilde{P}$ and the expectations are over $\tilde{P}$. The joint distribution is

$$P(\boldsymbol{y}, \boldsymbol{t}|\boldsymbol{\theta}) = \exp\left(-\sum_{i=1}^{n} g(t_i, y_i)\right) P(\boldsymbol{y}|\boldsymbol{\theta}), \tag{4.3}$$

where the loss function g is assumed to be normalized. F can be written as the free energy $-\log P(\boldsymbol{t}|\boldsymbol{\theta})$ plus the relative entropy between the posterior $P(\boldsymbol{y}|\boldsymbol{t}, \boldsymbol{\theta})$ and its model $\tilde{P}$:

$$\begin{aligned} F(\tilde{P}, \boldsymbol{\theta}) &= -\int \tilde{P}(\boldsymbol{y}) \log\left(\frac{P(\boldsymbol{y}|\boldsymbol{t}, \boldsymbol{\theta}) P(\boldsymbol{t}|\boldsymbol{\theta})}{\tilde{P}(\boldsymbol{y})}\right) d\boldsymbol{y} \\ &= -\log P(\boldsymbol{t}|\boldsymbol{\theta}) + \int \tilde{P}(\boldsymbol{y}) \log\left(\frac{\tilde{P}(\boldsymbol{y})}{P(\boldsymbol{y}|\boldsymbol{t}, \boldsymbol{\theta})}\right) d\boldsymbol{y}. \end{aligned} \tag{4.4}$$

Thus, changing $(\tilde{P}, \boldsymbol{\theta})$ such that F is decreased will enlarge the log evidence or decrease the KL divergence and fit the model $\tilde{P}$ tighter to the posterior. Note that, by the information inequality[2] for the relative entropy (see [CT91]), F is equal to the free energy iff $\tilde{P}$ is equal to the posterior almost everywhere. In general, Γ will not be broad enough to contain the posteriors for all the $\boldsymbol{\theta}$ considered during optimization.

The model class Γ we are using is the family of Gaussians with *factor-analyzed covariance matrices*:

$$\Sigma = \boldsymbol{D} + \sum_{j=1}^{M} \boldsymbol{c}_j \boldsymbol{c}_j^t, \tag{4.5}$$

where $\boldsymbol{D}$ is a diagonal matrix with positive entries and usually $M \ll n$. A factor-analyzed covariance matrix is able to track and represent the most important correlations between the component variables, as opposed to a diagonal matrix. This variational model class was suggested in [BB97], and we

[2]Which states that the relative entropy is nonnegative, and zero iff the compared distributions are equal almost everywhere.

will give an argument below why we think this class is suitable for approximating posteriors of kernel methods too. The main reason for employing a restricted class of Gaussians is that the reduced number of parameters (compared to Gaussians with full covariance matrices) leads to a simpler optimization problem which can be solved more efficiently. Gaussian distributions with factor-analyzed covariances occur in *factor analysis models*, as discussed in Appendix A.

To incorporate the distinction between normalized and unnormalized SVC loss, let $\tilde{g}(t_i, y_i)$ be any admissable loss function and $g(t_i, y_i) = \tilde{g}(t_i, y_i) + \log Z(y_i)$ be its normalized counterpart. Here, $Z(y_i) = \exp(-\tilde{g}(+1, y_i)) + \exp(-\tilde{g}(-1, y_i))$. This leads in an obvious way to the factorization $F = \tilde{F} + G$, where $\tilde{F}$ is the variational free energy based on the loss $\tilde{g}$, and

$$G = \mathrm{E}_{\tilde{P}} \left[\sum_i \log Z(y_i) \right]. \tag{4.6}$$

G and its gradients w.r.t. $\tilde{P}$ will in general be intractable, but we can employ any reasonably tight upper bound to retain the variational idea. If $\tilde{g}$ is already normalized, $Z \equiv 1$ and $G \equiv 0$ can be ignored. We will concentrate on $\tilde{F}$ first. For Gaussians $\tilde{P} = N(\boldsymbol{\mu}, \Sigma)$, $\tilde{F}$ can be computed relatively easy (at least approximately):

$$\begin{aligned} \tilde{F}(\tilde{P}, \boldsymbol{\theta}) &= \int \tilde{P}(\boldsymbol{y}) \left(\sum_{i=1}^{n} \tilde{g}(t_i, y_i) - \log N(\boldsymbol{y}|0, \boldsymbol{K}) \right) d\boldsymbol{y} - H(\tilde{P}) \\ &= \sum_{i=1}^{n} \nu_i + \frac{1}{2} \boldsymbol{\mu}^t \boldsymbol{K}^{-1} \boldsymbol{\mu} + \frac{1}{2} \operatorname{tr}\left(\Sigma \boldsymbol{K}^{-1}\right) + \frac{1}{2} \log |\boldsymbol{K}| \\ &\quad - \frac{1}{2} \log |\Sigma| - \frac{n}{2}, \end{aligned} \tag{4.7}$$

where we define

$$\nu_i = \mathrm{E}_{\tilde{P}} \left[\tilde{g}(t_i, y_i) \right] = \int \tilde{g}(t_i, y_i) N(y_i|\mu_i, \sigma_i^2)\, dy_i \tag{4.8}$$

and $\operatorname{diag} \Sigma = (\sigma_1^2, \ldots, \sigma_n^2)^t$. The calculation of the gradients w.r.t. the parameters of $\tilde{P}$ and w.r.t. $\boldsymbol{\theta}$ is demonstrated in Appendix C. Note that the loss function $\tilde{g}$ does not have to be differentiable for the gradients to exist. Although the algorithm is conceptually very simple, constructing an efficient implementation is challenging. The notoriously bad $O(n^3)$ scaling of other Bayesian Gaussian process techniques (see [WB98],[Gib97]) also affects the

variational algorithm, but further approximations might be applied to improve on this. These issues are also discussed in the Appendices B and C.

Having obtained feasible formulas for $\tilde{F}$ and its gradients, we can optimize $\tilde{F}$ using a nested loop algorithm as follows. In the inner loop we run an optimizer to minimize $\tilde{F}$ w.r.t. $\tilde{P}$ for fixed $\boldsymbol{\theta}$. We used a *conjugate gradients* optimizer since the number of parameters of $\tilde{P}$ is rather large. The outer loop is an optimizer minimizing $\tilde{F}$ w.r.t. $\boldsymbol{\theta}$, and we chose a *Quasi-Newton* method here since the dimension of Θ is usually rather small and gradients w.r.t. $\boldsymbol{\theta}$ are costly to evaluate. For questions about optimization, [Fle80] is an excellent reference (see also [Lue84]).

Finally, we have to deal with the criterion part G of (4.6). If $\tilde{g}$ is the unnormalized SVC loss, we can find a tight upper bound on G, as detailed in Appendix C.2.3. This upper bound and its gradients w.r.t. $\tilde{P}$ can be evaluated efficiently (in $O(n)$). Adding these to the corresponding terms for $\tilde{F}$ results in the complete criterion and gradient for inner loop optimization. Since G does not depend on $\boldsymbol{\theta}$, the outer loop optimization remains unchanged. We mention that we can also replace the quite complicated expressions of the bound on G by a simple approximation of G using numerical quadrature, although this will in general not be an upper bound.

In the sequel we will assume, if not otherwise stated, that the ν_i (as defined in (4.8)) are (tight bounds of) expectations over $g(t_i, y_i)$ instead of $\tilde{g}(t_i, y_i)$.

Technical remarks about optimization

Using the nested loop optimization technique detailed above, we quickly ran into some technical problems. Since problems of a similar kind occur very often in the context of variational Bayesian algorithms, we formulate the problem and a possible solution in general terms. Suppose we are given a criterion function of the form $f(\boldsymbol{x}) = \min_{\mathbf{y}} h(\boldsymbol{x}, \boldsymbol{y})$. We refer to $\boldsymbol{x}$ as *evaluation point*, to f as *true criterion*, to h as *upper bound* (for reasons detailed below) and to $\boldsymbol{y}$ as *(hidden) state*. These notations are used *exclusively* within this subsection, and we urge the reader not to confuse them with global notations. The general notation can be translated into the special case of the variational algorithm discussed above via $\boldsymbol{x} \to \boldsymbol{\theta}, \boldsymbol{y} \to \tilde{P}, h \to \tilde{F}$.

For every $\boldsymbol{x}$ we have $f(\boldsymbol{x}) = h(\boldsymbol{x}, \hat{\boldsymbol{y}})$ for $\hat{\boldsymbol{y}} = \operatorname{argmin}_{\mathbf{y}} h(\boldsymbol{x}, \boldsymbol{y})$. If the minimum point is not unique, we arbitrarily choose one of the candidates. We refer to $\hat{\boldsymbol{y}}$ as state *corresponding to* $\boldsymbol{x}$. Let $\boldsymbol{x}_0$ some evaluation point and $\hat{\boldsymbol{y}}_0$ the corresponding state. At first sight it seems to be hard to compute the gradient

of f at $\boldsymbol{x}_0$, but using the fact that $\nabla_{\mathbf{y}} h(\boldsymbol{x}_0, \hat{\boldsymbol{y}}_0) = \mathbf{0}$ we have

$$\nabla_{\mathbf{x}} f(\boldsymbol{x}_0) = \nabla_{\mathbf{x}} h(\boldsymbol{x}_0, \hat{\boldsymbol{y}}_0). \tag{4.9}$$

To compute $\nabla_{\mathbf{x}} f(\boldsymbol{x}_0)$, we first determine $\hat{\boldsymbol{y}}_0$, then fix this state and compute the gradient $\nabla_{\mathbf{x}} h(\boldsymbol{x}_0, \hat{\boldsymbol{y}}_0)$ while ignoring the dependency of $\hat{\boldsymbol{y}}_0$ on $\boldsymbol{x}_0$.

This method works fine in practice as long as the state $\hat{\boldsymbol{y}}_0$ corresponding to $\boldsymbol{x}_0$ is always unique and can be determined to high accuracy[3]. If this is not the case, we are in danger to run into the following problem. Suppose the gradient of f at $\boldsymbol{x}_0$ is rather small. For a unit vector $\boldsymbol{v}$ and a small $\varepsilon > 0$ we would expect the finite difference $(f(\boldsymbol{x}_0 + \varepsilon \boldsymbol{v}) - f(\boldsymbol{x}_0))/\varepsilon$ be close to the projected gradient $(\nabla_{\mathbf{x}} f(\boldsymbol{x}_0))^t \boldsymbol{v}$. However, if the value of f at $\boldsymbol{x}_0$ and $\boldsymbol{x}_0 + \varepsilon \boldsymbol{v}$ is "noisy" due to inaccuracies in the determination of the corresponding states at these points, the finite difference can be grossly wrong, and common line search algorithms cannot cope with this fact (see discussion in subsection 5.2).

We suggest a solution to this problem which is based on the ideas which led to a generalization of the EM algorithm (see [HN97]). This idea is discussed in great detail in subsection 3.1.4. If $\hat{\boldsymbol{y}}_0$ is the state corresponding to $\boldsymbol{x}_0$, and $\boldsymbol{x}$ is any point such that $h(\boldsymbol{x}, \hat{\boldsymbol{y}}_0) < h(\boldsymbol{x}_0, \hat{\boldsymbol{y}}_0)$, then we have

$$f(\boldsymbol{x}) \leq h(\boldsymbol{x}, \hat{\boldsymbol{y}}_0) < h(\boldsymbol{x}_0, \hat{\boldsymbol{y}}_0) = f(\boldsymbol{x}_0), \tag{4.10}$$

which means that decreasing the upper bound $h(\boldsymbol{x}, \hat{\boldsymbol{y}}_0)$ on $f(\boldsymbol{x})$ is guaranteed to decrease the true criterion too. If we therefore run into the problem stated above, we can proceed by computing the state $\hat{\boldsymbol{y}}_0$ corresponding to $\boldsymbol{x}_0$, use this to compute the gradient of f at $\boldsymbol{x}_0$ and to choose a search direction $\boldsymbol{s}$ based on this gradient and some history, then fix the state $\hat{\boldsymbol{y}}_0$ and perform the line search using the upper bound $h(\boldsymbol{x}, \hat{\boldsymbol{y}}_0)$ instead of the true criterion $f(\boldsymbol{x})$. This upper bound does not exhibit the "noise" problems detailed above and can also be evaluated more efficiently than $f(\boldsymbol{x})$.

The drawback of this replacement is of course that in general the line minimum point $\boldsymbol{x}_1$ of the upper bound will not be a line minimum point of the true criterion. This can be problematic when the modified line search is used with optimizers like conjugate gradients which require fairly accurate line searches on the true criterion. We therefore evaluate the gradient of f at $\boldsymbol{x}_1$

[3] An example seems to be the Laplace method for Gaussian process classification of [WB98]. In this case, the state is the posterior mode which can be determined as (unique) solution of a convex optimization problem. The problem can be solved by the Fisher Scoring (or Newton-Raphson) algorithm which is of second order and stable for convex settings.

and project it onto the search direction $\boldsymbol{s}$. If the absolute value of this derivative is still too large, we start another line search from $\boldsymbol{x}_1$ using the upper bound $h(\boldsymbol{x}, \hat{\boldsymbol{y}}_1)$ where $\hat{\boldsymbol{y}}_1$ corresponds to $\boldsymbol{x}_1$. The search direction for this line search is $-\sigma\boldsymbol{s}$ where σ is the sign of the projected gradient at $\boldsymbol{x}_1$. This is iterated until the absolute value of the projected gradient is sufficiently small.

To conclude, we now have two different options to perform a line search on $f(\boldsymbol{x})$ at any given time during the optimization. Either we base everything on the true criterion, or we use the upper bound in the way described above. The (optimal) scheduling between the two options depends strongly on the course of the optimization so far. The extreme strategies are to use the upper bound approximation only if the correct line search fails (which is what we currently do) and to use the approximation for all line searches.

We finally note that our implementation (see also Appendix D) supports criteria like $f(\boldsymbol{x})$ generically, using a sophisticated interface for communication between the criterion function object, the (generic) optimizer and the line search method.

The reader might wonder what happens if the criterion h is also noisy in the sense that finite differences of h are not reliable approximations of derivatives. This problem occurs if h has a certain relative error (usually unsystematic) which is *not* strongly correlated with the evaluation point. Approximations such as those suggested in Appendix B and C include *estimators* and fall into this category, so we are forced to address this problem. A possible solution would be to employ a line search method which is fairly robust against noisy criterion values. Such a method would focus mostly on gradient information, and it has to be designed with care since the basic task of bracketing a line minimum point is not trivial any more with noisy criteria. We are currently working on this problem.

4.2.2 Factor-analyzed variational distributions

We show in subsection 5.2.1 that the best Gaussian variational approximation to the posterior $P(\boldsymbol{y}|\boldsymbol{t}, \boldsymbol{\theta})$ has a covariance of the form $(\boldsymbol{K}^{-1} + \boldsymbol{S})^{-1}$ where $\boldsymbol{S}$ is a diagonal matrix. Since we restrict ourselves to Gaussians with factor-analyzed covariances, we cannot hope that the restricted class will contain the best Gaussian approximation. However, it is easy to show, by applying the *Woodbury formula* (see [PTVF92]), that if $\boldsymbol{K}$ is of the factor-analyzed form with M factors, so is the covariance of the best Gaussian approximation. Thus, by continuity, we might expect that if $\boldsymbol{K}$ is reasonable

well represented by a factor-analyzed matrix, tracking the prior variances and strongest correlations in the data, then a Gaussian with factor-analyzed covariance might do a good job in approximating the posterior, as compared to the best Gaussian approximation. This argument is not a firm one, and to derive a more precise result we would have to specialize to a particular kernel and loss function. We show in subsection 5.2.1 how the best Gaussian approximation might be computed using a sequential updating scheme, and this could be used to perform an empirical study, at least for not too large datasets.

4.2.3 MAP prediction

In this subsection, we assume for simplicity that the prior $P(\boldsymbol{\theta})$ is uninformative, i.e. that the log evidence (or marginal log likelihood) and the log posterior $\log P(\boldsymbol{\theta}|D)$ are identical up to constants not depending on $\boldsymbol{\theta}$. There is no loss in generality, since we can always add $\log P(\boldsymbol{\theta})$ to the log evidence which does not introduce any further complications.

Let $\hat{\boldsymbol{\theta}}$ denote the MAP value for the hyperparameter vector found by the algorithm described in this section. MAP prediction amounts to replacing the true posterior $P(\boldsymbol{\theta}|D)$ by the peak distribution $\delta(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})$ or, equivalently, to approximating the full posterior $P(y_*|\boldsymbol{x}_*, D)$ by $P(y_*|\boldsymbol{x}_*, D, \hat{\boldsymbol{\theta}})$. Recall the discussion of subsection 1.3.5. For notational convenience, we neglect conditioning on $\hat{\boldsymbol{\theta}}$ and $\boldsymbol{x}_*$. If $y_* \sim P(y_*|D)$ and $\boldsymbol{y} \sim P(\boldsymbol{y}|D)$, we have the linear relation $y_* = \boldsymbol{q}^t\boldsymbol{y} + r$ where $r \sim N(0, k_* - \boldsymbol{q}^t\boldsymbol{k})$ independent of $\boldsymbol{y}$. Note that $k_*, \boldsymbol{k}$ and $\boldsymbol{q} = \boldsymbol{K}^{-1}\boldsymbol{k}$ depend on $\boldsymbol{x}_*$ and $\hat{\boldsymbol{\theta}}$. Thus, if we approximate the intractable posterior $P(\boldsymbol{y}|D)$ by a Gaussian, y_* is Gaussian too, and the moments of the two distributions are linearly related.

Let $\tilde{P}$ denote the variational approximation to the final conditioned posterior $P(\boldsymbol{y}|D)$, and let $\boldsymbol{\mu}$ and Σ denote mean and covariance of the Gaussian distribution $\tilde{P}$. Within our variational framework it is reasonable to employ $\tilde{P}$ for prediction which, by the linear relation mentioned above, means that we approximate the predictive distribution $P(y_*|D)$ by

$$N(\boldsymbol{q}^t\boldsymbol{\mu}, \boldsymbol{q}^t\Sigma\boldsymbol{q} + k_* - \boldsymbol{q}^t\boldsymbol{k}). \tag{4.11}$$

The variance of the predictive distribution can be used to derive error bars for our predictions. The MAP discriminant uses the prediction vector $\hat{\boldsymbol{c}} = \boldsymbol{K}^{-1}\boldsymbol{\mu}$ and predicts $\hat{y}_* = \hat{\boldsymbol{c}}^t\boldsymbol{k}$. The cost of computing error bars is dominated by the need to evaluate $\boldsymbol{q} = \boldsymbol{K}^{-1}\boldsymbol{k}$ for every prediction point $\boldsymbol{x}_*$ which can be done in $O(n^2)$ using a conjugate gradients technique. If there are more test

points than training points, it pays to compute the Cholesky decomposition of the covariance matrix $\boldsymbol{K}$. If Σ has the factor-analyzed structure of (4.5), an error bar can be computed in $O(Mn)$ if $\boldsymbol{q}$ is given. This can be seen as additional advantage of the restricted variational model class. We will refer to the discriminant based on the approximation (4.11) as *prediction based on the model mean* $\boldsymbol{\mu}$.

Alternatively we can derive an approximate MAP discriminant by replacing $P(\boldsymbol{y}|D)$ by its Laplace approximation, as discussed in subsection 1.3.5. In this case, the mode (or mean) of the predictive distribution is $\boldsymbol{q}^t\hat{\boldsymbol{y}}$ where $\hat{\boldsymbol{y}}$ denotes the mode of $P(\boldsymbol{y}|D)$. The discriminant uses the prediction vector $\hat{\boldsymbol{c}} = \boldsymbol{K}^{-1}\hat{\boldsymbol{y}}$ and predicts $\hat{y}_* = \hat{\boldsymbol{c}}^t\boldsymbol{k}$. Error bars can be obtained using the variance in (1.17). The computational cost is roughly the same as in the "model mean" case, if some biconjugate gradients technique is employed (note that the matrix in (1.17) is not symmetric). Alternatively, a LU decomposition of $\boldsymbol{I} + \boldsymbol{W}\boldsymbol{K}$ might be considered. However, if the loss function of the model is unnormalized and therefore $\hat{\boldsymbol{y}}$ cannot be regarded as mode of a proper posterior, the computation of error bars is no longer possible. We will refer to the discriminant based on the Laplace approximation as *prediction based on the posterior mode*, even in the case of unnormalized loss functions.

Can we really justify prediction based on the posterior mode? It seems odd to neglect the posterior approximation $\tilde{P}$ we employed so far to determine $\hat{\boldsymbol{\theta}}$ and to replace it by the Laplace approximation. Indeed, there is a subtle danger we have to be aware of, as was discussed in subsection 3.1.4. If all possible variational distributions are far from the true posterior conditioned on the mode of the true evidence, $\hat{\boldsymbol{\theta}}$ might be far from this mode, since it might be the case that the true posterior conditioned on $\hat{\boldsymbol{\theta}}$ can be fitted more accurately by one of the variational distributions. In the sense of the minimum overall coding cost (see discussion in subsection 3.1.5), it is still reasonable to predict using $\hat{\boldsymbol{\theta}}$ *and* $\tilde{P}$, but plugging in $\hat{\boldsymbol{\theta}}$ as approximation to the mode of $P(\boldsymbol{\theta}|D)$ and employing a local approximation (such as Laplace) to $P(\boldsymbol{y}|D)$ can fail badly. There is no general answer to this problem. If we favour prediction based on the posterior mode, we usually have a good reason to do so, for example the sparsity of the discriminant in the SVC case. By choosing a broad model class for the variational distributions, we can decrease the likelihood that the conditioned posterior at the mode of $P(\boldsymbol{\theta}|D)$ is far from all these distributions. This gives some justification to the conjecture that $\hat{\boldsymbol{\theta}}$ is close to the mode of the true evidence. If we assume that the latter is sharply peaked around its mode, we can plug $\hat{\boldsymbol{\theta}}$ in. If the posterior $P(\boldsymbol{y}|D)$ conditioned on $\hat{\boldsymbol{\theta}}$ is well represented by its Laplace approximation, prediction based on the posterior mode is reasonable. If the

loss $\tilde{g}$ is unnormalized, we replace it by a normalized approximation g and use this approximation to determine $\hat{\boldsymbol{\theta}}$. After plugging $\hat{\boldsymbol{\theta}}$ in, we determine $\hat{\boldsymbol{y}}$ as maximizer of the criterion which mimics the log posterior of $\boldsymbol{y}$, but employing the unnormalized loss $\tilde{g}$. Under the further assumption that $\hat{\boldsymbol{y}}$ is a good approximation to the mode of the posterior $P(\boldsymbol{y}|D)$ incorporating g, we can conclude that prediction using $\hat{\boldsymbol{y}}$ should work reasonably well. An analysis of the accuracy of all these approximations to validate (or invalidate) the conjectures is likeli to prove very difficult and is far out of the scope of this work. The final justification is as always given by good performance on real-world tasks.

4.3 Comparison with related methods

By revealing relations and similarities to other methods for Bayesian model selection among kernel regression classifiers, we can gain some additional insight into the method proposed in the previous section, as well as into the role of variational methods among other approximation techniques in the toolbox of a Bayesian analyst. Let us briefly recapitulate that one basis of the variational technique is the use of families of bounds for convex functions. If a function of interest happens not to be convex, we might arrive at a convex expression by transforming the outputs or inputs of the function. An example is the logistic function $\sigma(u) = (1 + \exp(-u))^{-1}$ which is sigmoid-shaped, but whose negative logarithm is strictly convex (such functions are called *log-concave*). Jaakkola [Jaa97] gives a detailed discussion of such transformations. If a function is piecewise convex, we can get an overall bound by piecing together affine bounds. An example of this is the bound we use on the criterion part (4.6) in subsection 4.2.1, see also Appendix C.2.3.

As discussed in chapter 3, affine lower bounds of convex functions can be derived in a principled way, while (sensible) upper bounds usually depend on more properties than just convexity. Replacing the function by the bound, we trade loss of exactness for tractability. Whether the resulting approximation to the final criterion of interest (for example predictive probabilities) is also a bound, depends on the way the "inner" approximations (which *are* bounds) are combined. If the bounding property can be preserved, the variational method delivers precise confidence intervals for the wanted probabilities, thus gives considerably stronger results than many other approximating techniques. However, in most cases preservation is not possible or results in too wide gaps between upper and lower bounds to be of practical use. In this case, we can compare the variational technique directly with methods

that involve "inner approximations" by parameterized families which are *not* bounds.

4.3.1 The Laplace method

The Laplace method, as described in [WB98], implements evidence maximization approximately by replacing the log of the joint distribution $P(\boldsymbol{t}, \boldsymbol{y}|\boldsymbol{\theta})$ (as a function of $\boldsymbol{y}$) by the Taylor series around its mode $\hat{\boldsymbol{y}}$ and retaining only terms up to second order in $\delta\boldsymbol{y}$. In general this works only for differentiable loss functions and, as discussed in subsection 4.1.1, is not applicable to the SVC loss. To circumvent further difficulties, we will restrict ourselves to the Bernoulli loss $g(t_i, y_i) = -\log \sigma(t_i y_i)$, introduced in subsection 1.3.5.

This manipulation renders the evidence integral (4.1) tractable, and we end up with

$$-\log P(\boldsymbol{t}|\boldsymbol{\theta}) \approx \Psi(\hat{\boldsymbol{y}}) + \frac{1}{2}\log|\boldsymbol{K}^{-1} + \boldsymbol{W}| - \frac{n}{2}\log 2\pi, \tag{4.12}$$

where $\Psi(\boldsymbol{y}) = -\log P(\boldsymbol{t}, \boldsymbol{y}|\boldsymbol{\theta})$ is given by (1.15) and $\boldsymbol{K}^{-1} + \boldsymbol{W}$ is the Hessian of Ψ evaluated at $\hat{\boldsymbol{y}}$, being the mode of the posterior $P(\boldsymbol{y}|\boldsymbol{t}, \boldsymbol{\theta})$. Here, $\boldsymbol{W} = \nabla\nabla(-\log P(\boldsymbol{t}|\boldsymbol{y})) = \mathrm{diag}(\sigma(\hat{y}_i)(1 - \sigma(\hat{y}_i)))_i$, evaluated at $\hat{\boldsymbol{y}}$. The derivatives of (4.12) can easily be obtained and plugged into a standard optimizer. The algorithm contains the search for the mode $\hat{\boldsymbol{y}}$ as a subroutine. Since Ψ is strictly convex, $\hat{\boldsymbol{y}}$ is easily found using the Newton-Raphson (i.e. Fisher scoring) method. The computational burden of (4.12) is the log determinant term which seems to be $O(n^3)$. Using approximations proposed in [Ski89] (and discussed in Appendix B), one might be able to pull this down to expected $O(n^2)$.

It is easy to relate the Laplace method to the variational technique introduced above. Consider replacing the loss terms $g(t_i, y_i)$ in the variational free energy criterion (4.2) by second-order polynomials

$$\begin{aligned} g_{a,i}(t_i, y_i|\mu_i) &= g(t_i, \mu_i) + \frac{\partial g}{\partial y}(t_i, \mu_i)(y_i - \mu_i) + \frac{1}{2}\frac{\partial^2 g}{\partial y^2}(t_i, \hat{y}_i))(y_i - \mu_i)^2 \\ &= -\log \sigma(t_i\mu_i) - t_i\sigma(-t_i\mu_i)(y_i - \mu_i) + \frac{1}{2}w_i(y_i - \mu_i)^2. \end{aligned} \tag{4.13}$$

Note that μ_i is the only free parameters of this family of polynomials, the prefactor w_i of the quadratic part is fixed as a function of $\hat{y}_i$. This changes the criterion F to F_a, where the latter additionally depends on $\boldsymbol{\mu} = (\mu_i)_i$. Note

that F_a is not in general an upper bound on the free energy $-\log P(\boldsymbol{t}|\boldsymbol{\theta})$. Consider minimizing F_a w.r.t. $\boldsymbol{\mu}$ and an arbitrary variational distribution $\tilde{P}$ over $\boldsymbol{y}$. Let $\boldsymbol{a}$ and Σ be mean and covariance matrix of $\tilde{P}$. Then we have

$$F_a(\boldsymbol{\mu}) = \tau + \boldsymbol{\pi}^t(\boldsymbol{a}-\boldsymbol{\mu}) + \frac{1}{2}(\boldsymbol{a}-\boldsymbol{\mu})^t\boldsymbol{W}(\boldsymbol{a}-\boldsymbol{\mu}) + \text{const}, \tag{4.14}$$

where $\tau = \sum_i g(t_i, \mu_i)$, $\boldsymbol{\pi} = (\partial g(t_i,\mu_i)/\partial\mu_i)_i$ and "const" does not depend on $\boldsymbol{\mu}$. Setting the derivative equal to zero and noting that g is strictly convex, we get $\boldsymbol{\mu} = \boldsymbol{a}$. We identify these parameter vectors, arriving at

$$F_a(\boldsymbol{\mu}) = \tau + \frac{1}{2}\operatorname{tr}\left(\boldsymbol{W} + \boldsymbol{K}^{-1}\right)\Sigma + \frac{1}{2}\boldsymbol{\mu}^t\boldsymbol{K}^{-1}\boldsymbol{\mu} - H(\tilde{P}) + \text{const}, \tag{4.15}$$

where "const" is independent of $\tilde{P}$. Setting the derivative w.r.t. $\boldsymbol{\mu}$ equal to zero, we have $\boldsymbol{\pi} + \boldsymbol{K}^{-1}\boldsymbol{\mu} = 0$. A brief look at (1.15) shows that $\boldsymbol{\mu}$ must be the posterior mode $\hat{\boldsymbol{y}}$. Furthermore, (4.15) can be written (up to constants which only depends on the mean $\boldsymbol{\mu}$ of $\tilde{P}$) as relative entropy $D(\tilde{P}||N(\boldsymbol{\mu}, (\boldsymbol{K}^{-1}+\boldsymbol{W})^{-1}))$. Putting all together, we see that F_a is minimized over $\boldsymbol{\mu}$ and unrestricted $\tilde{P}$ by setting $\boldsymbol{\mu} = \hat{\boldsymbol{y}}$ and $\tilde{P} = N(\hat{\boldsymbol{y}}, (\boldsymbol{K}^{-1} + \boldsymbol{W})^{-1})$. $\tilde{P}$ is the approximation of the posterior employed by the Laplace method. Evaluating F_a at this minimum point gives

$$F_a(\boldsymbol{\theta}) = -\log P(\boldsymbol{t}, \hat{\boldsymbol{y}}|\boldsymbol{\theta}) + \frac{1}{2}\log\left|\boldsymbol{K}^{-1} + \boldsymbol{W}\right| - \frac{n}{2}\log(2\pi) + n, \tag{4.16}$$

which is equal to (4.12) up to a constant.

How does this compare to the variational technique? Both methods introduce parametric approximations to arrive at a criterion that can be maximized efficiently. Both maximize this criterion w.r.t. the parameters of the approximation in an inner and w.r.t. $\boldsymbol{\theta}$ in an outer loop. The variational method derives the approximation by replacing the true posterior in the free energy integral by a variational distribution, thus arriving at the variational free energy which is a true upper bound for *any* $\tilde{P}$. This has the advantage that in the inner loop the algorithm will always choose the best approximation within the tractable family[4]. On the other hand, the Laplace method modifies the log joint distribution within the free energy integral such that the posterior becomes Gaussian. While $\tilde{P}$ employed by the Laplace method is the exact posterior of this modified joint distribution, it is in general *not* the best Gaussian approximation to the posterior of the true joint distribution.

[4]In the sense of relative entropy distance. This is of course a weak argument for someone who doesn't accept this distance, but we have given sufficient motivation above.

Recall the discussion in subsection 3.1.5, where the EM algorithm and the variational technique are motivated using an MDL argument. Instead of description length we will use the term "information", and we will not be concerned about issues of quantization. We refer to [CT91] for an introduction to informatic-theoretic concepts like entropy (or (self) information), relative entropy and their joint and conditional versions. Consider the equation

$$-\log P(\boldsymbol{t}|\boldsymbol{\theta}) + H(P(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta})) = \mathrm{E}_{P(\mathbf{y}|\mathbf{t},\theta)}\left[-\log P(\boldsymbol{t},\boldsymbol{y}|\boldsymbol{\theta})\right]. \tag{4.17}$$

The right-hand side is the joint information of the event that the targets are $\boldsymbol{t}$ and of the random variable $\boldsymbol{y} \sim P(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta})$. This is the sum of the information of the $\boldsymbol{t}$ event and the uncertainty in $\boldsymbol{y}$, sampled from the posterior. The variational approximation is based on a modification of this equation:

$$-\log P(\boldsymbol{t}|\boldsymbol{\theta}) + H(\tilde{P}) + D(\tilde{P}||P(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta})) = \mathrm{E}_{\tilde{P}}\left[-\log P(\boldsymbol{t},\boldsymbol{y}|\boldsymbol{\theta})\right]. \tag{4.18}$$

This relation can be described using the following setting. Suppose the $\boldsymbol{t}$ event occurs and a variable $\boldsymbol{y}$ is sampled. Our knowledge about $\boldsymbol{y}$ is that it is sampled from $P(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta})$, but accidentally the true sampling distribution is $\tilde{P}$, a fact that is unknown to us. Under these circumstances, the right-hand side is the information of the $\boldsymbol{t}$ event and of the random variable $\boldsymbol{y}$, from our point of view.

The left-hand side is the information we get from achieving the knowledge that the targets have the value $\boldsymbol{t}$. The left-hand side decomposes this information into the information of the $\boldsymbol{t}$ event, the uncertainty in $\boldsymbol{y} \sim \tilde{P}$ (originating from the sampling) and a relative entropy term measuring the error of assuming $P(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta})$ as sampling distribution while the true one is $\tilde{P}$. This is the usual interpretation put forward in case of the relative entropy (see [CT91]).

On the other hand, the Laplace approximation of the log evidence is local around $\hat{\boldsymbol{y}}$, and the error has no information-theoretic interpretation. We can get an idea of the nature of this approximation if we compare it to the variational one with $\tilde{P}(\boldsymbol{y}) = N(\hat{\boldsymbol{y}}, (\boldsymbol{K}^{-1} + \boldsymbol{W})^{-1})$. If *error* denotes the error of the Laplace approximation, we have

$$-\log P(\boldsymbol{t}|\boldsymbol{\theta}) + H(\tilde{P}) + error = -\log P(\boldsymbol{t},\hat{\boldsymbol{y}}|\boldsymbol{\theta}) + \frac{n}{2}. \tag{4.19}$$

The right-hand side of this equation is an approximation to the right-hand side of (4.18), obtained by replacing the negative log joint distribution by its second-order Taylor expansion around $\hat{\boldsymbol{y}}$. If we continue this expansion by including higher-order terms, *error* tends to the relative entropy term in (4.18).

This analysis suggests using the variational method introduced above, but employing the variational distribution $\tilde{P} = N(\hat{\boldsymbol{y}}, (\boldsymbol{K}^{-1} + \boldsymbol{W})^{-1})$. It will be interesting to compare this method with the Laplace algorithm and the variational algorithm based on distributions with factor-analyzed covariances. The figures in subsection 4.4.2 give some intuition.

4.3.2 Another variational method

Gibbs [Gib97] introduced a variational method for inference and learning in Gaussian process classification models which is based on local approximations of the logistic function, being the Bernoulli loss function introduced above. He employs upper and lower bounds suggested in [Jaa97] to derive different approximations to the predictive distribution. Since we focus here on approximations of the evidence, we only need to apply an upper bound to the Bernoulli loss $g(t, y) = -\log \sigma(ty)$. We follow the derivation in [Jaa97], Appendix 3.B. Obviously, since $g(t, y)$ is convex, there is no affine upper bound, but we can nevertheless apply the convex duality principle as follows. If we manage to transform input and/or output of $-\log \sigma(y)$ linearly such that the resulting function $\tilde{f}(x)$ is symmetric in x and concave in x^2, we can use an affine upper bound on f, where $f(x^2) = \tilde{f}(x)$. Transforming back, this results in a quadratic upper bound for the original function. We have $-\log \sigma(y) = -y/2 + \tilde{f}(x)$ where $x = y$ and $\tilde{f}(x) = f(x^2) = \log(\exp(-x/2) + \exp(x/2))$. f is concave in x^2, thus by convex duality we have the tangents as upper bounds:

$$f(x^2) \leq (x^2 - \xi^2)\nabla_{\xi^2} f(\xi^2) + f(\xi^2) \tag{4.20}$$

Here, ξ is a variational parameter. The bound is exact iff $x^2 = \xi^2$. With $\lambda(\xi) = \nabla_{\xi^2} f(\xi^2) = \tanh(\xi/2)/4\xi = (\sigma(\xi) - 1/2)/2\xi$, we finally have the upper bound

$$-\log \sigma(ty) \leq -\log \sigma(\xi) - \frac{1}{2}(ty - \xi) + \lambda(\xi)(y^2 - \xi^2), \tag{4.21}$$

a quadratic which touches the loss at $\pm\xi$. Now, replacing the loss terms in the evidence integral by their bounds, we get an analytically tractable upper bound on the free energy $-\log P(\boldsymbol{t}|\boldsymbol{\theta})$ which is minimized in turn w.r.t. the variational parameters $\boldsymbol{\xi} = (\xi_i)_i$ and the hyperparameters $\boldsymbol{\theta}$. It is interesting to note that this algorithm can be modified to yield simultaneous upper *and* lower bounds on the free energy. The latter bound employs the convexity of the Bernoulli loss and is derived straightforwardly from the convex duality principle. However, the affine bounds used to construct the lower bound are

usually quite loose, and the free energy is usually closer to the upper than to the lower bound.

Note that the upper bound on the Bernoulli loss is a very special one. It is not obvious how to find a tractable upper bound on the normalized SVC loss which is not even differentiable everywhere. In this context "tractable" means that the Gaussian evidence integral becomes analytically tractable once the loss is replaced by its bound. This narrows down the class of possible functional forms for such bounds. For example, the idea of piecing together bounds to attack piecewise defined functions cannot be applied. The variational free energy framework breaks down the problem to simple one-dimensional expectations over loss terms which are almost trivial to bound or approximate.

The variational approach of [Gib97] differs from the generic variational free energy minimization algorithm discussed above. Very much like the Laplace method of the previous subsection, we introduce a parametric family of approximations to the loss function. Plugging these in for the true loss terms renders the posterior Gaussian and the evidence integral tractable. However, since the polynomial approximation employed here are also upper bounds, the final criterion is an upper bound of the free energy for *all* values of the variational parameters, which justifies the inner loop minimization w.r.t. $\boldsymbol{\xi}$. Note that the "posterior" derived from the normalized bound of the joint distribution is in general *not* the best Gaussian approximation of the true posterior in terms of relative entropy distance. The former distribution is a Gaussian with mean $(1/2)(\boldsymbol{K}^{-1}+2\Lambda)^{-1}\boldsymbol{t}$ and covariance $(\boldsymbol{K}^{-1}+2\Lambda)^{-1}$, where $\Lambda = \mathrm{diag}(\lambda(\xi_i))_i$, evaluated at the maximum point $\boldsymbol{\xi}$. A direct comparison between the algorithms follows along the lines of the discussion in the previous subsection. The algorithm of this section has no obvious information-theoretic motivation.

Jaakkola and Haussler [JH99] suggest a sequential version of the variational algorithm of [Gib97]. There, the posterior of $y(\cdot)$ is approximated by propagating Gaussian approximations to the distributions $P(y(\cdot)|D_i)$, $D_i = \{(\boldsymbol{x}_1, t_1), \ldots, (\boldsymbol{x}_i, t_i)\}$, starting from the Gaussian process prior. This is done by employing the quadratic upper bound on the Bernoulli loss discussed above. The resulting approximation to the posterior will in general be worse than that employed by the algorithm in [Gib97] and also depend on the ordering in which the training points are presented, but it can be computed extremely fast. A modification of the algorithm to implement evidence maximization is straightforward.

4.4 Experiments and results

The variational Bayesian model selection algorithm described above was evaluated on a number of datasets from the *UCI machine learning repository* and the *DELVE archive* of the University of Toronto[5], namely *Leptograpsus crabs (crabs)*, *Pima Indian diabetes (pima)*, *Wisconsin Breast Cancer (wdbc)*, *Ringnorm (ringnorm)*, *Twonorm (twonorm)* and *Waveform (waveform)* (class 1 against 2). We refer to these repositories for detailed documentations. In each case we normalized the whole set to zero mean, unit variance in all input columns, picked a training set of desired size at random and used the rest for testing. We chose (for $X = \mathbb{R}^d$) the *squared-exponential kernel* (see [Wil97])

$$K(\boldsymbol{x}, \boldsymbol{x}'|\boldsymbol{\theta}) = C \left(\exp\left(-\frac{1}{2d} \sum_{i=1}^{d} w_i (x_i - x_i')^2 \right) + v \right), \tag{4.22}$$

being a widely used "standard" kernel for Gaussian processes and Support Vector machines. All parameters are constrained to be positive, and we enforced this constraint by representing a parameter θ_i as ν_i^2 where ν_i was optimized without constraints. We refer to C as variance parameter, to v as bias parameter and to the w_i as inverse squared length scales, or simply *relevance factors*. The quantities $1/\sqrt{w_i}$ are known as *length scales* in the literature (see [Nea97],[Mac97]). The number of factors in the representation (4.5) of the covariance matrix Σ of the variational distribution was 3 in all cases. The kernel parameters were initialized by sampling from rather broad log uniform distributions, but our experiments indicated that the algorithm is fairly invariant w.r.t. this initialization. The mean of the variational distribution was initialized with the minimizer of the SVC criterion (1.20). Although this initialization seems reasonable, we observed in all cases a very fast decrease in the criterion value during the first few inner loop steps which indicates that the initial variational distribution is far from the best one. We conjecture that the inner loop optimization is also fairly invariant w.r.t. initialization and that time and space required for the SVC optimization can be saved by simply using a fixed or random initial variational distribution. We have not investigated this question in detail.

Although the optimization of the variational distribution for fixed hyperparameters is not a convex problem, our experience in all cases was that convergence to a (possibly local) minimum was independent from the starting point and seemed not to get stuck in shallow local minima. The $O(n^3)$

[5] See `http://www.cs.utoronto.ca/~delve` and `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Name	train size	test size	Var. GP $\hat{\boldsymbol{y}}$	Var. GP $\boldsymbol{\mu}$	GP Lapl.	Var. SVM $\hat{\boldsymbol{y}}$	Var. SVM $\boldsymbol{\mu}$	SVM 10-CV	Lin. discr.
crabs	80	120	3	4	4	4	4	4	3
pima	200	332	66	66	68	64	66	67	67
wdbc	300	269	11	11	8	10	10	9	19
twonorm	300	7100	233	224	297	260	223	163	207
ringnorm	400	7000	119	124	184	129	126	160	1763
waveform	800	2504	206	204	221	211	206	197	220

Table 4.1: Number of test errors for various methods. None of the methods employed a hyperprior.

running time scaling behaviour somewhat limits the scope of the algorithm[6], but we hope to be able to improve on that (see section 5.2).

The experiments were done using the normalized SVC loss and the Bernoulli loss. For the normalized SVC loss, we evaluated two different discriminants, namely prediction based on the model mean and prediction based on the posterior mode. The model mean discriminant is abbreviated as $\boldsymbol{\mu}$ discriminant, while we abbreviate the posterior mode discriminant as $\hat{\boldsymbol{y}}$ discriminant. These types of discriminants are discussed in subsection 4.2.3.

For comparison we trained a Gaussian Process classifier with the MAP Laplace method of [WB98] and a Support Vector machine using 10-fold cross validation to select the free parameters. In contrast to [WB98], the results in table 4.1 for the MAP Laplace method were obtained without employing a hyperprior. Experiments with a hyperprior are discussed in subsection 4.4.1. In the cross validation case we constrained the relevance factors w_i of the kernel (4.22) to be equal (it is infeasible to adapt $d+2$ hyperparameters to the data using cross validation) and dropped the v parameter while using a bias parameter for the SVM. Error bars were not computed. The baseline method was a linear discriminant trained to minimize the squared error. Table 4.1 shows the test errors the different methods attained on the datasets.

What conclusions can be drawn from these results as well as those which are presented further below? A proper frequentist statistical test for significant differences between the methods requires that these methods are trained and tested on many independently drawn datasets from the same source (see [Ras96] for a thorough discussion of test designs for this purpose). While this is possible for large or artificially created datasets, it is clearly unsuitable for

[6]The largest training set we used was an extract of 800 cases from *Waveform*.

small real-world datasets. The commonly used practice is to randomly select training and test set of fixed sizes from the given base set and compute a sample test error. This is repeated many times, and the final estimate of the test error is the average over all the sample errors. This procedure is more robust against noise introduced by the random selection of training and test sets as well as other random sources such as those used for initialization of the method. However, it is not clear at all how to base a rigorous frequentist statistical test on this estimator (as test statistic), i.e. how to compute confidence intervals or p values, even if very simple assumptions about the nature of the noise are made. Furthermore, many Bayesians criticize frequentist tests severely. For some it is doubtful if *any* conclusions can ever be drawn from the result of such a test (see [Ber85] for details). Anyway, the results presented here are based on single runs and therefore subject to noise. The major reason for this design was limited time[7]. We conclude that although differences in performance between the methods can be observed on each datasets, we cannot test these differences in a rigorous frequentist statistical manner without investing much more time and labour, and this labour may be wasted considering the fact that most of the real-world datasets available are so small that they cannot be a faithful representation of their underlying source. We finally remark that a Bayesian test might overcome this difficulty, but such issues would lead us far out of scope (see [Ras96]).

The absence of rigorous statistical tests should not stop us from commenting the results we obtained on single runs, since this is common practice in the literature up until today. The ranking of the new variational algorithm within the list of considered methods is diverse, from which we conclude that its performance is comparable to the best algorithms known so far. These results have of course to be regarded in combination with how much effort was necessary to produce them. It took us almost a whole day and a lot of user interactions to do the cross validation model selection. The rule-of-thumb that a lot of Support Vectors at the upper bound indicate too large a parameter C in (4.22) failed for at least two of these sets, so we had to start with very coarse grids and sweep through several stages of refinement. We finally selected the parameter vector giving rise to the best score in the last stage. If there were several best vectors, we selected the one resulting in the smallest number of Support Vectors.

[7]On the majority of the dataset-method combinations we actually did multiple runs after minor changes in the code of the optimizers. Significant differences between these runs were not observed. The presented results are all computed using the latest code version prior to submission of this thesis.

Let us have a closer look at the results for each dataset. On some of the sets, an effect called *automatic relevance determination* (see [Nea97]) can nicely be observed. If an input dimension of the training set contains very little information about the target, Bayesian model selection methods will tend to favour very small values for the corresponding relevance factor w_i in the squared-exponential kernel, so that this input dimension is effectively ignored. Apart from rendering additional structural information about the data distribution, this feature can also be used to speed up predictions on large test sets, since we can savely ignore terms corresponding to very small w_i when evaluating the kernel (4.22). This form of "input sparsity" should be seen in constrast to the sparsity in the dual variables of a SVM. In combination, they render quite efficient discriminants. Note that ARD can easily be generalized by preprocessing the data by a linear nondiagonal transform before feeding it into the squared-exponential kernel and learning the transform by means of Bayesian model selection. The transform could be unconstrained or of some particular form, see for example Appendix A. Work in this direction has been done by Vivarelli and Williams [VW99]. ARD was observed on *crabs* (focussed on dimensions 2,3), *pima* (ignored 3,4), *wdbc* (largest relevance factor on 24, many dimensions effectively ignored), *waveform* (largest relevance factors on 9,10,11,16,17, ignored 1,2,3,6). *ringnorm* and *twonorm* exhibited no ARD effects[8]. All Bayesian methods behaved somewhat similar w.r.t. ARD on the datasets, in the sense that the sets of strongly preferred and ignored dimensions were highly overlapping (the figures above are from the intersections). Table 4.2 shows the final variance parameter value C, and table 4.3 shows the Support Vector statistics for each dataset (in case of SVM discriminants). We discriminate between Support Vectors at the upper bound C and essential Support Vectors which lie in $(0, C)$. Some additional comments follow.

It is rather difficult to draw conclusions from the variance parameter values in table 4.2, since we are dealing with real-world data which might have a lot of good explanations with rather different variances. The values chosen by the Bayesian methods seem to be rather large in comparison with the values chosen by cross validation. There are several possible explanations for this fact. First, for most of the datasets there were always parameter vectors with quite large C which attained almost as good a CV score as the "best" vector, and the CV score is known to be a noisy quantity[9]. Second, since we tied the relevance factors in the squared-exponential kernel for the CV experiments,

[8]This is not surprising, since in both cases the class-conditioned densities are spherical Gaussians, and the class means are related by $\boldsymbol{\mu}_1 = -\boldsymbol{\mu}_{-1}$.

[9]It is strongly quantized for rather small datasets in the first place.

Name	Var. GP (No HP)	GP Lapl. (No HP)	Var. SVM (No HP)	SVM 10-CV
crabs	1324	1075	571	1000
pima	251	12.7	79.5	31
wdbc	1619	910.5	401.4	300
twonorm	49.2	532.7	24.1	9
ringnorm	33	112.8	19.6	1
waveform	606.9	60.3	249.7	1.5

Table 4.2: Variance parameter chosen by different methods. HP: Hyperprior.

Name	Var. SVM (No HP)	Var. SVM (LN HP)	SVM 10-CV
crabs	6,5	8,6	8,7
pima	100,7	102,6	101,8
wdbc	13,5	6,9	16,14
twonorm	0,83	0,71	180,11
ringnorm	0,117	0,107	73,48
waveform	133,17	122,25	35,425

Table 4.3: Support Vector statistics. The figures are the number of SV at the upper bound C, followed by the number of essential SV. HP: Hyperprior, LN: Lognormal (see text and subsection 4.4.1).

the model was not able to perform ARD, i.e. concentrate on some input dimensions and ignore others. Therefore, a very sharp decision boundary as chosen by the Bayesian methods is not optimal for this model. Finally, we did not use hyperpriors with the Bayesian algorithms, and large hyperparameter values were not penalized. We investigate the hyperprior issue in more detail in subsection 4.4.1 below.

The SV statistics for the datasets *twonorm*, *ringnorm* and *waveform* differ strongly between the variational method and the cross validation solution, as do the test errors. Again, a direct comparison is difficult since the kernels are different and the statistics noisy. On *ringnorm*, the variational method has found a better solution by employing a larger variance parameter ($C = 1$ found by CV seems to be too small). Consequently, the CV solution mistook important variations in the data as noise, which can also be seen in the SV statistics (many dual variables at the upper bound C indicate a high noise assumption). All in all, the model chosen by CV was too simple to give good generalization. On *twonorm* (two Gaussians with rather strong overlap), the situation is inverted. Again, the model chosen by CV is simpler than the one selected by the variational method, but this time the simpler model generalized better while the complex model exhibited overfitting[10]. The underlying simplicity can also be inferred from the very good performance of the simple linear discriminant. Rather surprisingly, the Laplace GPC method exhibited extreme overfitting on this dataset.

4.4.1 Hyperpriors

The experiments described above were done without hyperpriors. To investigate the effect of hyperpriors, we repeated the experiments employing the lognormal hyperprior used in [WB98]. This prior can be described as follows. If $\boldsymbol{\theta} = (w_1, \ldots, w_d, v, C)^t$ is the hyperparameter vector for the squared-exponential kernel (4.22), transform $\boldsymbol{\theta}$ via $\boldsymbol{\gamma} = \log \boldsymbol{\theta}$ and place a Gaussian prior with mean $-3 \cdot \mathbf{1}$ and covariance $9 \cdot \boldsymbol{I}$ on the variable $\boldsymbol{\gamma}$. The MAP value $\hat{\boldsymbol{\theta}}$ is then found by optimizing the posterior

$$P(\boldsymbol{\gamma}|D) \propto P(D|\boldsymbol{\theta} = e^{\boldsymbol{\gamma}})P(\boldsymbol{\gamma}). \tag{4.23}$$

If $\hat{\boldsymbol{\gamma}}$ is the maximizer, we choose $\hat{\boldsymbol{\theta}} = \exp(\hat{\boldsymbol{\gamma}})$. Note that this is in general *different* from directly maximizing $P(\boldsymbol{\theta}|D)$, since MAP is not invariant

[10]Note that *overfitting* is impossible within a full Bayesian analysis. However, here we effectively select a single model instead of averaging over all models, and this *can* result in overfitting.

Name	GP Lapl. (LN HP)	Var. GP (LN HP)		Var. SVM (LN HP)	
		$\hat{\boldsymbol{y}}$	$\boldsymbol{\mu}$	$\hat{\boldsymbol{y}}$	$\boldsymbol{\mu}$
crabs	4	4	4	6	4
pima	67	66	66	66	66
wdbc	10	10	10	9	10
twonorm	284	230	231	271	233
ringnorm	209	127	121	136	124
waveform	216	217	214	208	213

Table 4.4: Number of test errors for various methods. HP: Hyperprior, LN: Lognormal (see text).

Name	GP Lapl. (LN HP)	Var. GP (LN HP)	Var. SVM (LN HP)
crabs	255	482.8	203.1
pima	31.2	80.2	25.3
wdbc	272	757.1	343.6
twonorm	308.9	47.0	22.1
ringnorm	133.5	31.9	17.4
waveform	50.2	494.7	206.7

Table 4.5: Variance parameter chosen by different methods. HP: Hyperprior, LN: Lognormal (see text).

against reparameterization (see also subsection 3.2.1). We chose this *particular* parameterization simply to be able to compare our results with those in [WB98].

Although this prior is very broad, it penalizes large and very small values of the hyperparameters. We undertook a series of experiments with the variational and the Laplace GPC method using the lognormal hyperprior described above. The results are given in table 4.4. Table 4.5 shows the final values of the variance parameter C. The Support Vector statistics can be found in table 4.3.

Let us compare the results among the different GPC algorithms for each dataset and point out some effects of using a (broad) hyperprior. On *crabs*, all methods performed identically, and the hyperprior has the effect of avoiding very large values of C, although in this case, by the simplicity of the dataset, even C values larger than 1000 (the value chosen by CV) do not

result in overfitting. Furthermore, the relevance factors w_i of the kernel are prevented from becoming very small[11]. On *pima*, the variational method performs insignificantly better than the Laplace algorithm. The prior effects are similar to the *crabs* case.

wdbc is a very small set with a large number of input dimensions, and both the Laplace and the variational algorithm for GPC and SVC needed an unusually large amount of running time before reaching stable minima. The Laplace GPC algorithm without hyperprior found a good solution by using a rather large variance parameter and focussing on only 8 dimensions. Dimension 24 was considered by far as most important. In this case, using the lognormal prior prevented the algorithm from choosing this solution, since this prior severely penalizes very small relevance factor and large C values. The variational GPC algorithm found another solution focussing on 10 dimensions and considering dimension 21 by far as most important. Employing a lognormal hyperprior lead to a (insignificantly) better solution using a smaller C value and relevance factor values bounded away from 0. Note that if we consider both solutions (found by the Laplace and the variational method) as good approximations to the true situations, we hereby encounter a situation where two quite different hypotheses are ranked highly under the posterior and would *both* contribute significantly to the exact Bayesian solution.

twonorm consists of two overlapping spherical Gaussians in high-dimensional space. The Bayes classifier is a hyperplane, which partly explains the very good performance of the simple linear discriminant. Since the class centers are placed at $-a \cdot \mathbf{1}$ and $a \cdot \mathbf{1}$, we would not expect any ARD effects, and except for the ignorance of component 3 by the Laplace method without hyperprior we indeed observed none for any of the Bayesian methods. The Laplace method exhibited strong overfitting by choosing a much too large C value, this failure was somewhat alleviated by the hyperprior. For the variational method, prediction based on the model mean outperformed predictions based on the posterior mode. This might be an instance of the problem mentioned in subsection 3.1.4, namely that sometimes variational free energy techniques prefer a solution at a $\hat{\hat{\boldsymbol{\theta}}}$ rather far from a free energy minimum point, because the posterior conditioned on $\hat{\boldsymbol{\theta}}$ can be fitted much better by a variational distribution than the one at the true minimizer.

ringnorm consists of two Gaussian clusters, one inside the other. There is no hope to seperate the classes by a linear discriminant, but the variational GPC method does quite well. The Laplace GPC method is again affected

[11] This is a weakness of this prior, since we *want* unimportant dimensions to be switched off completely.

1: Var. SVM, No HP ($\hat{\boldsymbol{y}}$)	5: Var. GP, No HP ($\hat{\boldsymbol{y}}$)	9: GP Lapl., No HP
2: Var. SVM, No HP ($\boldsymbol{\mu}$)	6: Var. GP, No HP ($\boldsymbol{\mu}$)	10: GP Lapl., LN HP
3: Var. SVM, LN HP ($\hat{\boldsymbol{y}}$)	7: Var. GP, LN HP ($\hat{\boldsymbol{y}}$)	11: SVM, 10-CV
4: Var. SVM, LN HP ($\boldsymbol{\mu}$)	8: Var. GP, LN HP ($\boldsymbol{\mu}$)	12: Lin. discr.

Table 4.6: Legend for box plots of figure 4.1.

by overfitting, which is somewhat alleviated by using a hyperprior. No ARD effects were observed, except that the Laplace GPC algorithm switched off dimension 20. The $\hat{\boldsymbol{\theta}}$ values chosen by the variational methods with and without hyperprior are quite similar, and the difference in their performances is probably due to chance.

For the *waveform* dataset we used the largest training set of all experiments. Not only was the optimization process based on the exactly computed criterion slow, but also the number of variational parameters (describing the variational distribution $\tilde{P}$) is rather large. The variational algorithm outperforms the Laplace method, but again the differences in the test error are not significant. The variational method without hyperpriors employed a rather large variance parameter and exhibited ARD effects. It focussed on the dimensions 11,10,9,16,17,12 and switched off the dimensions 3,2,6. The same method with lognormal hyperpriors gave worse results, the relevance factors ranged from 0.0019 (dimension 2) to 0.1213 (dimension 11). The Laplace method used much smaller values of C. The usage of a lognormal hyperprior was slightly beneficial in this case.

The same kind of comparison can be done between the variational SVC algorithm and the solutions determined by 10-fold CV. The results are not significantly different except for the sets *twonorm* and *ringnorm*. The easy set *twonorm* was handled well by CV, but the variational SVC method did a surprisingly bad job. Note also the significant difference in test error between prediction based on the model mean and on the "posterior mode" $\hat{\boldsymbol{y}}$. This might indicate that we encountered the problem discussed in subsection 4.2.3. On *ringnorm*, the variational method outperformed the CV solution significantly.

The series of box plots in figure 4.1 allows a direct comparison of the runs of all considered methods on the datasets. The legend is given in table 4.6.

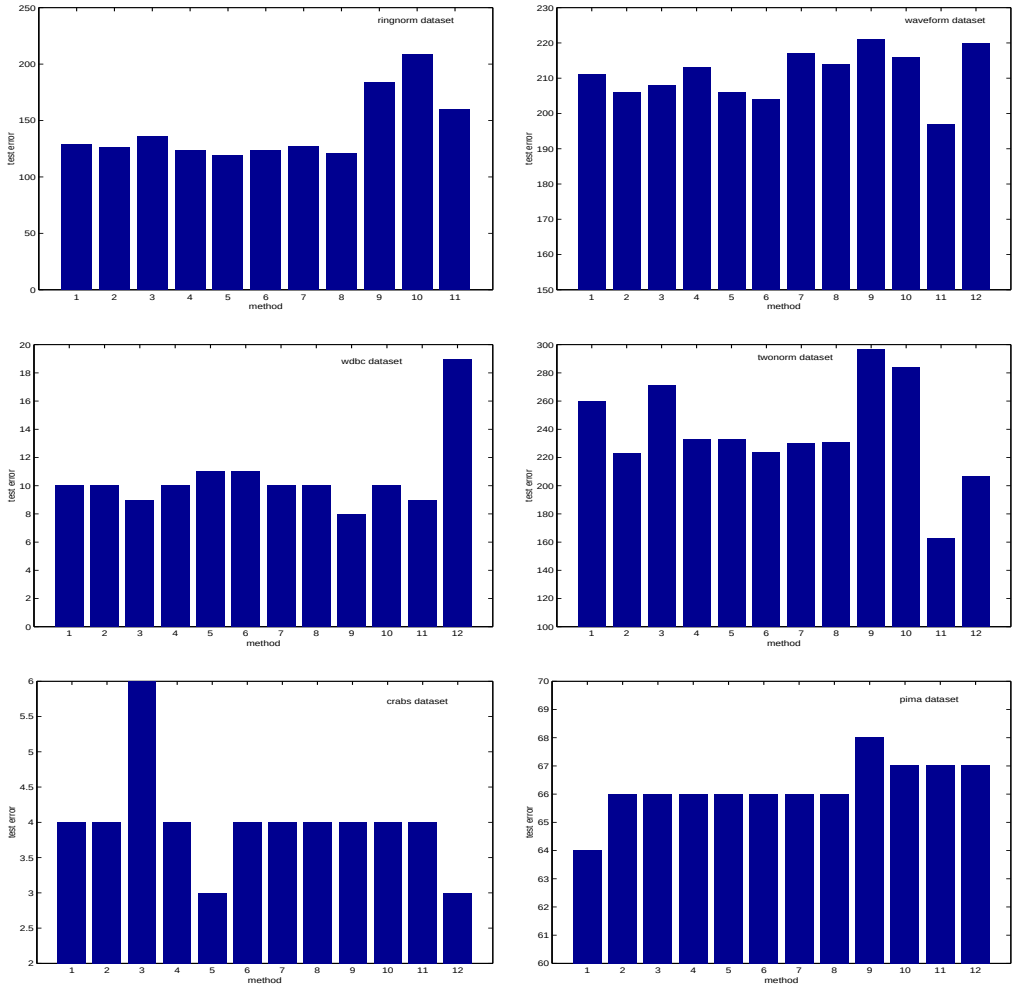

Figure 4.1: Comparison of test error of different methods. The legend is given in table 4.6.

4.4.2 Using the Gaussian from the Laplace method as variational density

At the end of subsection 4.3.1 we mentioned that it is natural to ask whether the Gaussian approximation to the posterior employed by the Laplace method (which depends only on local information at the posterior mode) would be a good *variational* approximation to the posterior in the sense of the relative entropy metric. In this subsection, we refer to this particular distribution as "Laplace Gaussian". The variational algorithm which employs the Laplace Gaussian as posterior approximation is called "variational Laplace"

method. Even though a local approximation seems to be very restrictive, the covariance matrix of the Laplace Gaussian is not of factor-analyzed form and might therefore outperform the best variational density with *restricted* covariance matrix.

We have not evaluated the variational Laplace algorithm explicitely, for the following reasons. First of all, there is no theoretical justification for using the Laplace Gaussian as variational approximation, since it is determined by criteria which are *not* related to the natural metric in this case, the relative entropy (see subsection 4.3.1). If we want to improve the variational algorithm, the correct thing to do is to broaden the variational model class (see also subsection 5.2.1). Second, the variational Laplace method is only applicable if the original Laplace method is, while the variational technique using factor-analyzed covariances is more general and works for nondifferentiable loss functions like the SVC loss. However, the question is of general interest, and the experiments described as follows might give some insights. We specialize to the GPC Bernoulli loss and do not employ hyperpriors. Following the $\boldsymbol{\theta}$ trajectory for the variational method with factor-analyzed covariances during the outer loop optimization, we obtained a sequence $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots$ where $\boldsymbol{\theta}_i$ is the hyperparameter vector value at which the i-th line search began. At each $\boldsymbol{\theta}_i$ we computed the Laplace Gaussian approximation to the conditioned posterior $P(\boldsymbol{y}|D, \boldsymbol{\theta}_i)$ and, using this approximation as variational density $\tilde{P}_L$, the variational free energy criterion $F(\tilde{P}_L, \boldsymbol{\theta}_i)$. This value can then be compared to the value of $F(\tilde{P}, \boldsymbol{\theta}_i)$ where $\tilde{P}$ is the best variational approximation to $P(\boldsymbol{y}|D, \boldsymbol{\theta}_i)$ with a factor-analyzed covariance matrix. The graphs of figure 4.2 show both criteria for various datasets.

Commenting on these figures is rather difficult, since the evaluations took part at points along learning trajectories which might not be representative. At least we can observe that on these trajectories the globally fitted (but restricted) variational density consistently outperforms the locally fitted Laplace Gaussian. A nice example is the figure 4.2 for the *twonorm* dataset where the fit of the Laplace Gaussian is poor compared to the global approximation. This might be an explanation for the bad performance of the Laplace algorithm on this dataset (see table 4.4). The same argumentation applies to the *ringnorm* dataset. On *crabs* and *pima* the fit of the Laplace Gaussian is as good as the global fit. We did not produce these protocols for *waveform* and *wdbc*.

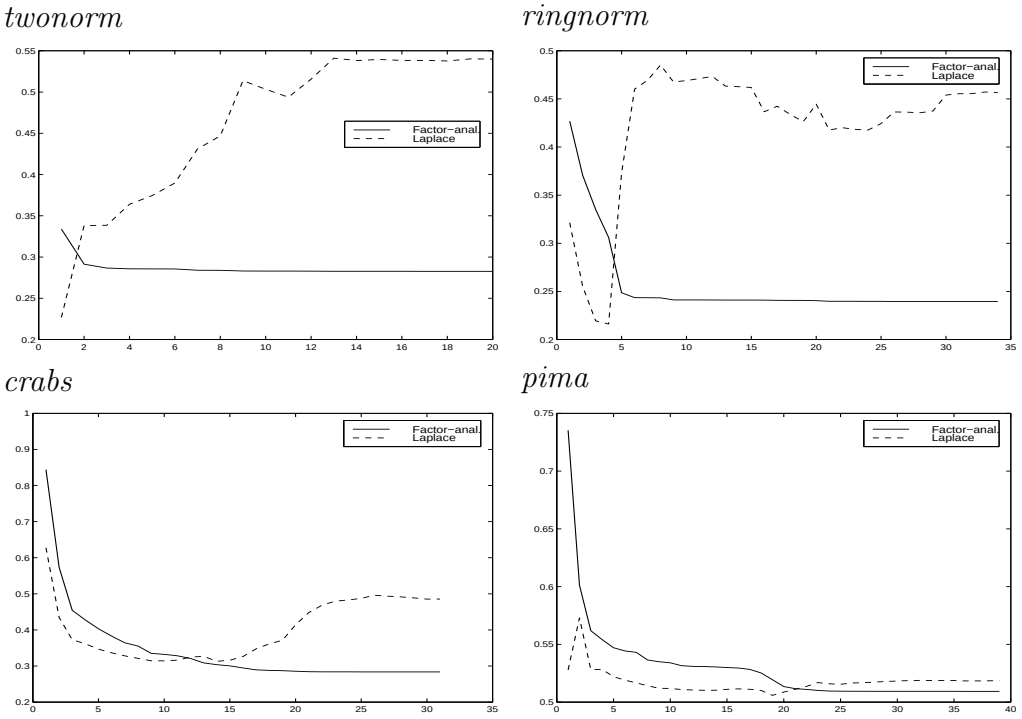

Figure 4.2: Criterion curves for several datasets. "Factor-anal." is the variational free energy criterion minimized during the optimization. "Laplace" is computed in the same way, but using the Laplace approximation to the posterior instead of the best variational approximation with restricted covariance.

4.4.3 Evidence approximation of Laplace method

It is interesting to compare the approximation (4.12) to the negative log evidence employed by the Laplace GPC method with the upper bound F used by the variational algorithm with Bernoulli loss. The approximation is clearly poor if it renders values larger than F for fixed $\boldsymbol{\theta}$. For real-world data, the true evidence cannot be computed, therefore we cannot decide for fixed $\boldsymbol{\theta}$ if the approximation or the bound is closer to the truth. A large gap between approximation and bound might indicate that the bound is poor, or that the approximation renders a much too small value, or both. Figure 4.3 collects plots of these two criteria which were again sampled along trajectories of the variational algorithm, as described in subsection 4.4.2. Note that these trajectories might not be a representative region for typical $\boldsymbol{\theta}$ values.

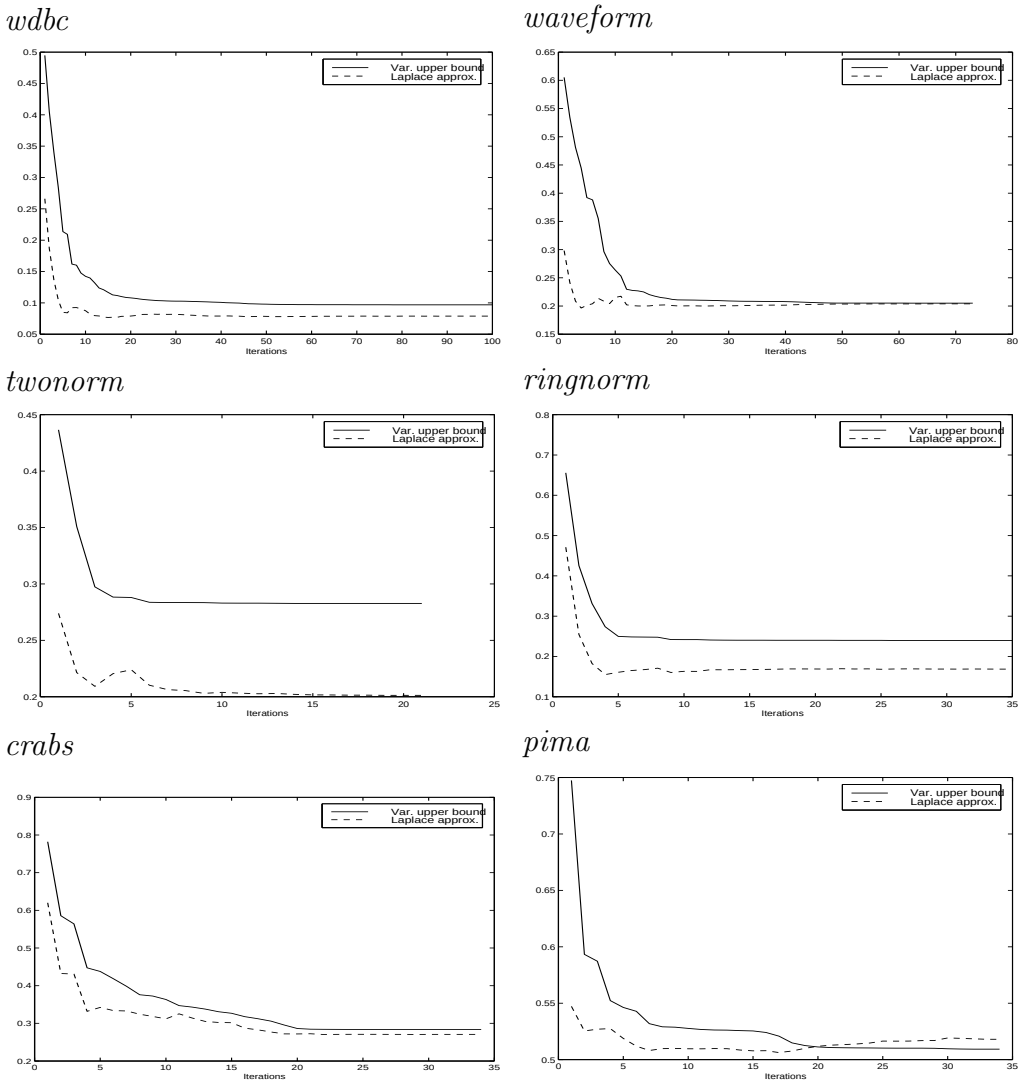

Figure 4.3: Comparison of Laplace approximation of the negative log evidence with the variational upper bound.

Chapter 5

Conclusions and future work

5.1 Conclusions

We have presented a common probabilistic framework for *kernel* or *spline smoothing* methods including popular architectures such as Gaussian processes and Support Vector machines. We identified the problem of unnormalized loss functions and suggested a general technique to overcome this problem at least approximately. We gave an intuitive interpretation of the effect an unnormalized loss function can induce, by comparing Support Vector classification to Gaussian process classification as a nonparametric generalization of logistic regression. This interpretation relates SVC to boosting techniques.

We proposed a variational Bayesian model selection algorithm for general normalized loss functions. This algorithm has a wider applicability than other Bayesian techniques based on the Laplace approximation and exhibits comparable performance in cases where both techniques can be used. Experiments on real-world datasets (see section 4.4) show that the variational algorithm is competitive with powerful state-of-the-art techniques. Like other Bayesian techniques, but unlike frequentist techniques such as cross validation, the variational method can simultaneously adapt a large number of free kernel parameters without any user interaction. Thus, Bayesian techniques like automatic relevance determination can be employed to gain additional knowledge about the data and to derive more efficient discriminants. In the present work, we restricted ourselves to model selection, i.e. employed a MAP approach, but MCMC techniques to build Bayesian committees will be considered in future work (see section 5.2).

The wider scope of this thesis is to provide a bridge between the fields of

probabilistic Bayesian techniques and Statistical Learning Theory as applied to kernel methods, although the emphasis is clearly on the former domain. We therefore chose to include some material of tutorial nature, and we hope that the text will prove useful as an introduction into modern Bayesian techniques to researchers in the field of Learning Theory. Furthermore, we have, neglecting any mathematical rigor, tried to shed some intuitive light on important SLT topics like the margin distribution, data-dependent error bounds and the field of boosting algorithms. The latter has an enormous practical impact without yet being completely understood in theoretical terms, and we hope that our comments are of interest to researchers in the Bayesian probabilistic modelling field.

Nonparametric discriminative techniques begin to transdencent their originally quite small field of applications. To date they have been mainly used for classification, regression estimation and smoothing problems with fixed-sized attribute vectors. However, recently complex problems like structure learning in graphical models (see [JMJ99]) or classification over spaces of variable-length sequences (see [JH98]) have been attacked. In the wider future, our new approach might also be applied to a broader range of problems. Obviously this requires an efficient, powerful and easy-to-use implementation which we hope to be able to provide and distribute in the public domain. Appendix D gives some preliminary details. In short, it is our opinion that a method can only achieve a widespread practical utility if it is powerful but also easy to use for non-experts in the field, i.e. if the method is as automatized as possible, without a large number of free parameters to be fiddled with "until it works"[1]. A further requirement, especially if the technique is quite complicated inside its black box, is free availability of code whose interface is generic enough to allow the method to be plugged into larger systems. The ultimate goal is a large toolbox with a common generic interface, which can be used by researchers to "plug-and-play", to compare methods and to build large systems using the methods that are suited best to solve their problems. Obviously, there is still a long way to go ...

5.2 Future work

In this section, we mention possible extensions to the work presented in this thesis. Some of them have already been worked out in considerable detail, as shown below.

[1] A striking counter-example seems to be the multi-layer perceptron, but automatic Bayesian techniques have found widespread application in this field too.

The running time behaviour of the variational algorithm proposed here and of Bayesian methods for Gaussian Processes in general is notoriously bad, namely $O(n^3)$ in general, where n is the training set size. Powerful approximations suggested by Skilling (see Appendix B) have been successfully applied to Bayesian Gaussian Processes in [Gib97], and we show in Appendix C that it should be possible to improve the scaling of our variational algorithm dramatically by employing these methods. We are currently working on providing code to test this conjecture. However, applying Skilling's techniques to the variational algorithm discussed above is, at least in our experience so far, not at all a straightforward task. The tridiagonal version of a Conjugate Gradients optimizer required to compute the Skilling bounds has to be designed very carefully to avoid numerical instabilities. Even a perfectly stable implementation of these approximations delivers only estimates of the required quantities whose relative errors cannot be decreased arbitrarily (see discussion in Appendix B). An optimizer based on such a "noisy" criterion has to be designed with care, since standard techniques like bracketing a minimum point only work up to a certain accuracy determined by the noise variance. In fact, we encounter one of these unusual situations where approximate gradient information about a criterion is much more reliable than approximate information about the criterion values at different points, and most standard optimizers do not work well in such situations. The recent paper [BWB99] deals with the same problem of a "noisy" criterion in the context of reinforcement learning. The authors suggest the relatively straightforward gradient-based line search function GSEARCH in combination with a conjugate gradients optimizer.

Another promising idea, namely dropping the factor-analyzed covariance matrix constraint when modeling the variational distribution in the algorithm discussed above and using a sequential approach to optimize free parameters, has already been worked out in some detail, but even though preliminary tests have been done, no conclusions can be drawn so far. We describe this approach in the following subsection.

In the present thesis we advocate using a proper generative model which is in a sense "close" to the improper SVC model and then viewing the SVC solution as efficient *approximation* of the solution for the proper model (see subsection 2.1.7). Given the proper model, we can employ Bayesian techniques, then use the SVC approximation whenever applicable. For example, we can grow a *Bayesian committee* of SV classifiers by replacing the MAP optimization by an MCMC method which produces samples $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ from the posterior $P(\boldsymbol{\theta}|D)$ or at least from a reasonable approximation. The committee discriminant is simply the arithmetic average of the discriminants

conditioned on $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$. Note that although the expert discriminants are computed as inner products, the mixture discriminant cannot be written as a single inner product. Therefore, a mixture discriminant is a nontrivial extension of the concept of a linear smoother such as a Gaussian process or a SVM. As mentioned above, we can base the expert discriminants either on the SVC solution (in this case, they are sparse) or on the mean of the final variational distribution. It would be interesting to compare such committees to MAP discriminants or committees trained by boosting (see subsection 2.2.1) and cross validation.

We close with some technical details which might or might not be improved upon in the future. The present implementation uses four $n \times n$ matrices of double precision floating point numbers. This can be reduced to two such matrices if some running time performance is sacrificed. For general covariance kernels, we currently see no way to improve upon the $O(n^2)$ storage space scaling. Even if we apply Skilling approximations, it is essential that we can multiply arbitrary vectors with kernel covariance matrices very efficiently which means that the actual kernel matrix must be kept in memory. One might consider an artificial sparsification using quantization ideas, but the quantification noise is likely to spoil the optimization. For special problems, a careful choice of the kernel function can render covariances of a special structure which can be stored and computed with very efficiently (see [Sto99]), but the range of applicability of such techniques is usually rather limited.

5.2.1 Sequential updating of the variational distribution

While choosing Gaussians as variational distributions seems to be essential for tractability[2], the further restriction of the covariances to factor-analyzed form is merely done to reduce the computational demands. As we show now, there is a family of covariances with even fewer degrees of freedom which nevertheless includes the covariance matrix of the best Gaussian variational approximation to the posterior[3]. Ignoring terms which do not depend on the

[2]We know of no other family of multivariate densities that would render the evaluation of the variational free energy and its gradients tractable.

[3]From a theoretical point of view, the number of degrees of freedom of the model class does not have to be restricted, since there is no danger of overfitting. But in general we expect the optimization of the variational parameters to be easier in practice if their number is small.

variational density $\tilde{P} = N(\boldsymbol{\mu}, \Sigma)$, the criterion to minimize is

$$F(\tilde{P}) = \sum_{i=1}^{n} \nu_i + \frac{1}{2}\boldsymbol{\mu}^t \boldsymbol{K}^{-1}\boldsymbol{\mu} + \frac{1}{2}\operatorname{tr}\left(\Sigma \boldsymbol{K}^{-1}\right) - \frac{1}{2}\log|\Sigma|. \tag{5.1}$$

Define $\boldsymbol{\pi} = (\partial\nu_i/\partial\mu_i)_i$ and $\boldsymbol{W} = \operatorname{diag}(2\partial\nu_i/\partial\sigma_i^2)_i$. The gradients are

$$\begin{aligned} \partial F/\partial\boldsymbol{\mu} &= \boldsymbol{\pi} + \boldsymbol{K}^{-1}\boldsymbol{\mu}, \\ \partial F/\partial\Sigma &= \frac{1}{2}\left(\boldsymbol{W} + \boldsymbol{K}^{-1} - \Sigma^{-1}\right). \end{aligned} \tag{5.2}$$

Setting these equal to zero, we arrive at the stationary conditions

$$\boldsymbol{\mu} = -\boldsymbol{K}\boldsymbol{\pi}, \qquad \Sigma = \left(\boldsymbol{K}^{-1} + \boldsymbol{W}\right)^{-1}. \tag{5.3}$$

This is a coupled nonlinear system of equations, but solving by iteration (as commonly done in variational techniques) is clearly infeasible because of the inversion required in every step. Note that at the minimum, the covariance has the form $(\boldsymbol{K}^{-1} + \boldsymbol{D})^{-1}$, where $\boldsymbol{D}$ is a diagonal matrix. Therefore, the family of Gaussians with unconstrained mean and covariance of this particular form contains the best variational approximation to the posterior. The family has only $2n$ degrees of freedom, as opposed to $(M+1)n$ for the factor-analyzed covariances class. For common g (e.g. the SVC loss), $\boldsymbol{W}$ contains only positive entries, and we can restrict $\boldsymbol{D}$ to have elements ≥ 0, e.g. by parameterizing $\boldsymbol{D} = \boldsymbol{E}^2$. This has the advantage of automatically keeping Σ positive definite.

The drawback of this parameterization is the complicated relation between $\boldsymbol{D}$ and Σ. Any optimization strategy for F must necessarily recompute $\operatorname{diag}\Sigma$ after each change of $\boldsymbol{D}$, because the ν_i and the corresponding gradients $\boldsymbol{\pi}$ and $\boldsymbol{W}$ depend on this diagonal. The computation is $O(n^3)$ if $\boldsymbol{D}$ is changed arbitrarily, which would be infeasible. However, a feasible optimization of F might be achieved using a *sequential* update strategy, i.e. changing only a small number of components of $\boldsymbol{D}$ in each iteration. Naive sequential updating would use a fixed schedule to run over the parameters of $\boldsymbol{D}$ and optimize F w.r.t. each of them seperately. This is a special case of a *working set* method which maintains a working set (or window) $I \subset \{1, \ldots, n\}$, $K = |I| \ll n$ and optimizes F w.r.t. to the parameters indexed by I. I is then moved over the complete index set, using either a fixed schedule or a working set selection heuristic. The *sequential minimal optimization (SMO)* algorithm for Support Vector classification [Pla98] is an example for the latter case. The rationale behind using a selection heuristic rather than a fixed selection schedule is that in many optimization tasks, given a particular dataset, not all of the

free model parameters are equally important when optimizing the criterion. Often, an algorithm can concentrate on a small set of parameters most of the time, without running less efficiently than an algorithm with a fixed fair selection schedule. A good selection heuristic should manage to find this particular subset, at least approximately. Of course, some care is required when implementing such a heuristic-based working set optimization, we have to avoid getting caught in spurious minima.

Given that a selection heuristic can be identified which can be expected to work well in the sense discussed above, sequential updating can be implemented very efficiently using an *implicit storage structure*. By this we mean a structure which supports *lazy evaluation* strategies. This is made precise below. The sequential updating algorithm employs iterations of the form:

1. Do several Newton steps to update the mean $\boldsymbol{\mu}$.
2. Select a *working set* $I \subset \{1, \ldots, n\}$, $K = |I| \ll n$, using a selection heuristic. The goal of the heuristic would be to select these components in $\boldsymbol{D}$ which affect F most if changed.
3. Optimize F w.r.t. D_I (the subscript I denotes a reduction of the corresponding entity to the components with indices in I). Thereby keep track of Σ, at least in an implicit form (see below). The diagonal of Σ is maintained explicitely.

The optimization of F w.r.t. $\boldsymbol{\mu}$ can be done using the Newton-Raphson algorithm. The Hessian of F w.r.t. $\boldsymbol{\mu}$ is $\boldsymbol{B} + \boldsymbol{K}^{-1}$ where $\boldsymbol{B} = \mathrm{diag}(\partial^2 \nu_i / \partial \mu_i^2)_i$, and we have the Newton step:

$$\boldsymbol{\mu}' = \boldsymbol{K}\boldsymbol{a}, \qquad \boldsymbol{a} = (\boldsymbol{I} + \boldsymbol{B}\boldsymbol{K})^{-1}(\boldsymbol{\pi} + \boldsymbol{B}\boldsymbol{\mu}). \tag{5.4}$$

Note that the gradient $\nabla_{\mu'} F$ is $\boldsymbol{\pi}' + \boldsymbol{a}$, and the gradient norm can be used in a stopping criterion. Note that optimization to a high accuracy is necessary only during later iterations.

We postpone the issue of a selection heuristic and concentrate on optimizing F w.r.t. $\boldsymbol{D}_I$ where the working set I is given. First experiments with a Newton optimization failed, so we switched to a simple line search along the negative gradient. The latter is given by

$$\partial F / \partial \boldsymbol{D}_I = \left(\frac{1}{2} \boldsymbol{v}_i^t (\boldsymbol{D} - \boldsymbol{W}) \boldsymbol{v}_i \right)_{i \in I}, \tag{5.5}$$

where $\boldsymbol{v}_i = \Sigma \boldsymbol{e}_i$ denote the columns of Σ. Note that the gradient depends only on those columns of Σ whose indices are in I. This "locality" characteristic

allows us to use an implicit storage structure for Σ, as discussed below. The criterion F itself does not have the same locality characteristic, i.e. we cannot update it using the columns I of Σ only. Therefore, the line search has to be based on gradient information only, which might prove to be quite unstable, since we cannot bracket the line minimum point.

Trading memory space for efficiency, we choose the following implicit storage structure for Σ:

$$\boldsymbol{v}_i = \hat{\boldsymbol{v}}_i + \Sigma \boldsymbol{\alpha}^{(i)}, \tag{5.6}$$

where $\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^n$ is usually sparse and $\alpha_i^{(i)} = 0$. $\boldsymbol{v}_i$ is called *explicit* iff $\boldsymbol{\alpha}^{(i)} = \mathbf{0}$, *implicit* otherwise. $\boldsymbol{v}_j$ is called *parent* of $\boldsymbol{v}_i$ if $\alpha_j^{(i)} \neq \mathbf{0}$. The representation induces a directed graph with the $\boldsymbol{v}_i$ being the vertices and an edge from $\boldsymbol{v}_j$ to $\boldsymbol{v}_i$ iff $\alpha_j^{(i)} \neq 0$. We need the graph to be acyclic, i.e. its transitive hull must be a semi-ordering. We start with an explicit Σ matrix, i.e. $\boldsymbol{\alpha}^{(i)} = \mathbf{0}$ for all i. We maintain at any time an ordering $o(\cdot)$ of $\{1, \ldots, n\}$ which is consistent with the transitive hull of the graph, i.e. $\boldsymbol{v}_j \to \cdots \to \boldsymbol{v}_i$ iff $o(j) < o(i)$.

We will see shortly that to optimize F w.r.t. $\boldsymbol{D}_I$ using a line search and to maintain the implicit representation of Σ, we only need the columns $\boldsymbol{v}_i,\ i \in I$ and the diagonal $\operatorname{diag} \Sigma$ of the covariance in explicit form. After having chosen a working set I, we therefore have to ensure that the columns corresponding to I are known explicitely. This can be done using a straightforward stack-based algorithm which marks the $\boldsymbol{v}_i,\ i \in I$ as well as all their ancestors. We then run over $\{1, \ldots, n\}$ following the ordering $o(\cdot)$ and compute each marked column $\boldsymbol{v}_i$ as a linear combination of $\hat{\boldsymbol{v}}_i$ and its parents (which, by definition of $o(\cdot)$, are explicit at this time).

We introduce the notation $\boldsymbol{A}_{J,K} = (\alpha_{j,k})_{j \in J, k \in K}$ where J, K are ordered subsets of $\{1, \ldots, n\}$. The complete index set is abbreviated with a dot, i.e. $\boldsymbol{A}_{J,\bullet} = (\alpha_{j,k})_{j \in J, k}$. If J or K contain only one element, we drop the set braces. Updating $\boldsymbol{D}_I$ to $\tilde{\boldsymbol{D}}_I = \boldsymbol{D}_I + \Delta_I$ can be done using the Woodbury formula (see [PTVF92]). If $\boldsymbol{U} = \boldsymbol{I}_{\bullet,I}$, then

$$\begin{aligned} \tilde{\Sigma} &= \left(\Sigma^{-1} + \boldsymbol{U}\Delta_I \boldsymbol{U}^t\right)^{-1} = \Sigma - \Sigma \boldsymbol{U} \Delta_I \left(\boldsymbol{I} + \boldsymbol{U}^t \Sigma \boldsymbol{U} \Delta_I\right)^{-1} \boldsymbol{U}^t \Sigma \\ &= \Sigma - \Sigma_{\bullet,I} \Delta_I \left(\boldsymbol{I} + \Sigma_{I,I} \Delta_I\right)^{-1} \Sigma_{\bullet,I}^t. \end{aligned} \tag{5.7}$$

Note that the update term only depends on $\boldsymbol{v}_i,\ i \in I$. We have

$$\tilde{\boldsymbol{v}}_j = \boldsymbol{v}_j - \Sigma_{\bullet,I} \Delta_I \left(\boldsymbol{I} + \Sigma_{I,I} \Delta_I\right)^{-1} \Sigma_{j,I}^t. \tag{5.8}$$

It follows that $\tilde{\Sigma}_{\bullet,I} = \Sigma_{\bullet,I} \boldsymbol{M}$, where $\boldsymbol{M} = \boldsymbol{I} - \Delta_I (\boldsymbol{I} + \Sigma_{I,I} \Delta_I)^{-1} \Sigma_{I,I}^t$. This allows an explicit update of the "working set" columns.

The columns $\boldsymbol{v}_i$, $i \notin I$ will only be updated implicitely, by modifying the weights $\boldsymbol{\alpha}^{(i)}$. Let

$$\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_i + \tilde{\Sigma}_{\bullet,I}\boldsymbol{\gamma}_i \tag{5.9}$$

and $\Gamma = (\gamma_{i,j})_{i,j} = (\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_n) \in \mathbb{R}^{K,n}$ (recall that $K = |I|$). Using (5.8) it is easy to see that $\Gamma = -\Delta_I \Sigma^t_{\bullet,I}$. Furthermore, we have to take into account that the parents of $\boldsymbol{v}_i$ have also been changed according to (5.9). Some algebra reveals the final update equation

$$\tilde{\alpha}_l^{(i)} = \alpha_l^{(i)} + \gamma_{l,i} - \sum_k \alpha_k^{(i)} \gamma_{l,k}, \qquad l \in I, \tag{5.10}$$

and $\tilde{\alpha}_l^{(i)} = \alpha_l^{(i)}$ for all $l \notin I$. Using (5.9), we also have the explicit update equation for $\operatorname{diag}\Sigma$:

$$\tilde{\sigma}_i^2 = \sigma_i^2 + \tilde{\Sigma}_{i,I}\boldsymbol{\gamma}_i \tag{5.11}$$

This completes the description of the optimization on the working set I. During the line search, we work on copies of $\Sigma_{\bullet,I}$ and $\operatorname{diag}\Sigma$ and only update these (explicitely). After the line search has terminated, resulting in the step Δ_I, we in addition perform an implicit update of columns not in the working set, as described above.

The issue of choosing a working set selection heuristic is sometimes supported by the nature of the optimization problem, for example in the Support Vector classification case (see [Pla98]). In the case of the variational sequential updating algorithm, such a support is not given, and we are only beginning to explore some ideas. A good selection heuristic for the working set would be of course to compute $\nabla_{\mathcal{D}} F$ and select the indices of the largest elements. However, to compute this gradient we need the complete covariance Σ in explicit form, and the idea of a implicit storage structure could not be applied in this case. In contrast to that, $\nabla_\Sigma F = (1/2)(\boldsymbol{W} - \boldsymbol{D})$, which can be computed using $\boldsymbol{\mu}$ and $\operatorname{diag}\Sigma$ only. From (5.5) we have

$$\frac{\partial F}{\partial d_i} = \sum_{k=1}^{n} \frac{\partial F}{\partial \sigma_k^2}(-v_{i,k}), \tag{5.12}$$

where $\boldsymbol{D} = \operatorname{diag}(d_1, \ldots, d_n)$ and $\boldsymbol{v}_i = (v_{i,1}, \ldots, v_{i,n})^t$. If we assume that Σ is strongly diagonal dominant, we can approximate this by

$$\frac{\partial F}{\partial d_i} \approx -\sigma_i^2 \frac{\partial F}{\partial \sigma_i^2}. \tag{5.13}$$

This suggests computing $-(\operatorname{diag}\Sigma)\nabla_\Sigma F$ and choosing the indices of the largest components. Although the assumption is too strong in reality, Σ exhibits diagonal dominance to a certain degree if $\boldsymbol{K}$ does and $\boldsymbol{D}$ contains only positive elements. Thus, the heuristic might prove useful in practice.

Bibliography

[Ant97] Martin Anthony. Probabilistic analysis of learning in artificial neural networks: The PAC model and its variants. *Neural Computing Surveys*, 1:1–47, 1997.

[Bar98] Peter Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 2(44):525–536, 1998.

[BB97] David Barber and Christopher Bishop. Ensemble learning for multi-layer networks. In *Advances in Neural Information Processing Systems 10*, pages 395–401. MIT Press, 1997.

[Ber85] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, 2nd edition, 1985.

[Bis95] Christopher Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[Bre96] Leo Breiman. Arcing classifiers. Technical report, University of California, Berkeley, 1996.

[Bre97] Leo Breiman. Prediction games and arcing algorithms. Technical Report 504, University of California, Berkeley, 1997.

[BS97] David Barber and Bernhard Schottky. Radial basis functions: A Bayesian treatment. In *Advances in Neural Information Processing Systems 10*, pages 402–408. MIT Press, 1997.

[BSW96] Christopher Bishop, Markus Svensen, and Christopher K.I. Williams. GTM: The generative topographic mapping. Technical Report 15, NCRG, Aston University, April 1996.

[Bur98a] Christopher Burges. Geometry and invariance in kernel based methods. In Schölkopf et al. [SBS98], pages 89–116.

[Bur98b] Christopher Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[BWB99] Jonathan Baxter, Lex Weaver, and Peter Bartlett. Direct gradient-based reinforcement learning: Ii. gradient ascent algorithms and experiments. Technical report, Australian National University, September 1999.

[CC96] M. Cowles and B. Carlin. Markov chain Monte Carlo convergence diagnostics – a comparative review. *Journal of the American Statistical Association*, 91(434):337–348, 1996.

[CCST98] Colin Campbell, Nello Cristianini, and John Shawe-Taylor. Dynamically adapting kernels in Support Vector machines. Technical Report 17, Royal Holloway College, London, 1998.

[Cre93] Noel Cressie. *Statistics for Spatial Data*. Wiley series in probability and mathematical statistics. Wiley, 2nd edition, 1993.

[CT91] Thomas Cover and Joy Thomas. *Elements of Information Theory*. Series in Telecommunications. John Wiley & Sons, 1st edition, 1991.

[Dev86] Luc Devroye. *Nonuniform Random Variate Generation*. Springer, New York, 1986.

[DGL96] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Applications of Mathematics: Stochastic Modelling and Applied Probability. Springer, 1st edition, 1996.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[Fel71] William H. Feller. *An Introduction to Probability Theory and its Applications*, volume 2. John Wiley & Sons, 2nd edition, 1971.

[FHT98] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University, 1998.

[Fle80] Roger Fletcher. *Practical Methods of Optimization: Unconstrained Optimization*, volume 1. John Wiley & Sons, 1980.

[Fri99] Jerome Friedman. Greedy function approximation: a gradient boosting machine. Technical report, CSIRO CMIS, 1999.

[FS96] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the 13th international conference*, 1996.

[Gib97] Mark N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997.

[GS92] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford Science Publications. Clarendon Press, Oxford, 2nd edition, 1992.

[GS94] P.J. Green and Bernhard Silverman. *Nonparametric Regression and Generalized Linear Models*. Monographs on Statistics and Probability. Chapman & Hall, 1994.

[Hal57] P. Halmos. *Introduction to Hilbert Space and the Theory of Spectral Multiplicity*. Chelsea, New York, 1957.

[HKS94] David Haussler, Michael Kearns, and Robert Schapire. Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. *Machine Learning*, 14:83–113, 1994.

[HN97] G. E. Hinton and R. M. Neal. A new view on the EM algorithm that justifies incremental and other variants. In Jordan [Jor97].

[HS52] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal Res. Nat. Bur. Standards*, 49:409, 1952.

[Hub81] P. Huber. *Robust Statistics*. Wiley, New York, 1981.

[HVC93] G. E. Hinton and D. Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Conference on Computational Learning Theory 6*, pages 5–13. Morgan Kaufmann, 1993.

[Jaa97] Tommi Jaakkola. *Variational methods for inference and estimation in graphical models*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1997.

[Jaa99] Tommi Jaakkola. Class notes 6.892 machine learning seminar, mit. 1999.

[Jay82] E. T. Jaynes. *Papers on Probability, Statistics and Statistical Physics*. Reidel, Dordrecht, 1982.

[JH98] Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 11*, 1998.

[JH99] Tommi Jaakkola and David Haussler. Probabilistic kernel regression models. In D. Heckerman and J. Whittaker, editors, *Workshop on Artificial Intelligence and Statistics 7*. Morgan Kaufmann, 1999.

[JHD99] Tommi Jaakkola, David Haussler, and M. Diekhans. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of ISMB*, 1999.

[JMJ99] T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. Technical Report MIT-AITR 1668, Massachusetts Institute of Technology, August 1999. See `http://www.ai.mit.edu/~tommi/papers.html`.

[Joa98] Thorsten Joachims. Making large-scale SVM learning practical. In Schölkopf et al. [SBS98], pages 169–184.

[Jor97] M. I. Jordan, editor. *Learning in Graphical Models*. Kluwer, 1997.

[KV94] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, 1994.

[Kwo99] James Tin-Tau Kwok. Integrating the evidence framework and the Support Vector machine. Submitted to ESANN 99, 1999.

[Lue84] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 2nd edition, 1984.

[Mac91] David MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1991.

[Mac93] David MacKay. Hyperparameters: optimize, or integrate out? In *Proceedings of Maxent*, 1993.

[Mac95] D. MacKay. Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks. *Network — Computation in Neural Systems*, 6(3):469–505, 1995.

[Mac97] D. MacKay. Introduction to Gaussian processes. Technical report, Cambridge University, UK, 1997. See `http://wol.ra.phy.cam.ac.uk/mackay/README.html`.

[Mac98] David J.C. MacKay. Choice of basis for Laplace approximation. *Machine Learning*, 33(1), October 1998.

[MBB98] Llew Mason, Peter Bartlett, and Jonathan Baxter. Improved generalization through explicit optimization of margins. Technical report, Australian National University, 1998. To appear in Machine Learning.

[McA99a] David McAllester. PAC-Bayesian model averaging. In *Conference on Computational Learning Theory 12*, pages 164–170, 1999.

[McA99b] David McAllester. Some PAC-Bayesian theorems. *Machine Learning*, 37(3):355–363, 1999.

[MN83] P. McCullach and J.A. Nelder. *Generalized Linear Models*. Number 37 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1st edition, 1983.

[MOR98] Klaus-Robert Müller, Takashi Onoda, and Gunnar Rätsch. Soft margins for AdaBoost. Technical Report NeuroCOLT2-TR-1998-021, GMD FIRST, Berlin, 1998.

[Nea93] R. M. Neal. Probabilistic inference using Markov chain monte carlo methods. Technical report, University of Toronto, 1993.

[Nea96] Radford M. Neal. *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. Springer New York, 1996.

[Nea97] Radford M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian classification and regression. Technical Report 9702, Department of Statistics, University of Toronto, January 1997.

[OW99] M. Opper and O. Winther. Gaussian process classification and SVM: Mean field results and leave-one-out estimator. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.

[OW00] Manfred Opper and Ole Winther. Gaussian processes for classification: Mean field algorithms. *Neural Computation*, 12(11):2655–2684, 2000.

[Pla98] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Schölkopf et al. [SBS98], pages 185–208.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[PW96] J. Propp and D. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1–2):223–252, 1996.

[Ras96] C. E. Rasmussen. *Evaluation of Gaussian Processes and Other Methods for Nonlinear Regression*. PhD thesis, University of Toronto, 1996.

[RG99] S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2), 1999.

[Rip96] Brian D. Ripley. *Pattern Recognition for Neural Networks*. Cambridge University Press, 1996.

[Ris86] J. Rissanen. Stochastic complexity and modeling. *Annals of Statistics*, 14(3):1080–1100, 1986.

[Roc70] R. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[Row98] S. Roweis. EM algorithms for PCA and SPCA. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 626–632. MIT Press, 1998.

[SBS98] B. Schölkopf, C. Burges, and A. Smola, editors. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1998.

[Sch64] I. J. Schönberg. Spline functions and the problem of graduation. In *Proc. Nat. Acad. Sci.*, volume 52, pages 947–950, 1964.

[SFBL98] R. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

[Ski89] John Skilling, editor. *Maximum Entropy and Bayesian Methods*. Cambridge University Press, 1989.

[Sol99] Peter Sollich. Probabilistic interpretations and Bayesian methods for Support Vector machines. In *Proceedings of ICANN 99*, 1999.

[Sol00] Peter Sollich. Probabilistic methods for support vector machines. In S. Solla, T. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 349–355. MIT Press, 2000.

[SS98] R. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the 11th annual conference on computational learning theory*, 1998.

[SSM98] A. Smola, B. Schölkopf, and K. Müller. General cost functions for Support Vector regression. In *Australian Congress on Neural Networks*, 1998.

[SSSV97] Bernhard Schölkopf, P. Simard, Alexander Smola, and Vladimir N. Vapnik. Prior knowledge in Support Vector kernels. In *Advances in Neural Information Processing Systems 10*, 1997.

[SSTSW99] B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Generalization bounds via the eigenvalues of the Gram matrix. Technical Report TR-1999-035, NeuroCOLT, Royal Holloway College, London, 1999.

[STBWA96] John Shawe-Taylor, Peter Bartlett, Robert Williamson, and Martin Anthony. A framework for Structural Risk Minimization. In *9th ACM Conference on Computational Learning Theory*, pages 68–76, 1996.

[Sto99] Amos Storkey. Truncated covariance matrices and Toeplitz methods in Gaussian processes. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 1999.

[TB98] M. Tipping and C. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1998.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1st edition, 1998.

[VW99] F. Vivarelli and C. K. I. Williams. Discovering hidden features with Gaussian process regression. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press, 1999.

[Wah90] Grace Wahba. *Spline Models for Observational Data*. CBMS-NSF Regional Conference Series. SIAM Society for Industrial and Applied Mathematics, 1990.

[Wah98] Grace Wahba. Support vector machines, reproducing kernel Hilbert spaces and the randomized GACV. In Schölkopf et al. [SBS98], pages 69–88.

[WB98] Christopher K. I. Williams and David Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

[Wil96] Christopher K. I. Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems 9*. MIT Press, 1996.

[Wil97] Christopher K. I. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In Jordan [Jor97].

[WLZ99] Grace Wahba, Yi Lin, and Hao Zhang. Generalized approximative cross validation for Support Vector machines, or, another way to look at margin-like quantities. Technical Report 1006, Department of Statistics, University of Wisconsin, 1999. Available at `http://www.stat.wisc.edu/~wahba/`.

[WR96] Christopher K. I. Williams and Carl E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*. MIT Press, 1996.

[WV00] C. K. I. Williams and F. Vivarelli. Upper and lower bounds on the learning curve for Gaussian processes. *Machine Learning*, 40(1):77–102, 2000.

Appendix A

Factor-analyzed covariances

A.1 Origins of factor-analyzed covariances

Standard *maximum likelihood factor analysis* models describe the distribution of observed continuous variables as Gaussian with factor-analyzed covariance matrix. We use this family of distributions in the variational algorithm described above, and although we gave some evidence to support this choice in our special case, we think it is useful to mention briefly the origins of this family here. Factor analysis is a special case of a *linear Gaussian model*, among a lot of other architectures developed in seperation, as discussed in the comprehensive review [RG99]. We broadly follow this work here. A linear Gaussian model consists of two discrete-time processes and can be described by the equations

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \phi\left(\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{w}_t\right), \\ \boldsymbol{y}_t &= \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{v}_t. \end{aligned} \tag{A.1}$$

$(\boldsymbol{w}_t)_t$ and $(\boldsymbol{v}_t)_t$ are temporally white Gaussian processes with $\boldsymbol{w}_t \sim N(\boldsymbol{0}, \boldsymbol{Q})$, $\boldsymbol{v}_t \sim N(\boldsymbol{0}, \boldsymbol{R})$, and independent of each other. Here, $\boldsymbol{x}_t \in \mathbb{R}^M$, $\boldsymbol{y}_t \in \mathbb{R}^n$, $M < n$. If ϕ is either the identity or the winner-takes-all mapping $\phi(\boldsymbol{x}) = \boldsymbol{e}_{\mathrm{argmax}_j\, x_j}$, inference (i.e. filtering, smoothing) is easy for these models, and the EM algorithm (as discussed in section 3.1.4) can be used for parameter learning (i.e. system identification).

Consider $\boldsymbol{A} = 0$ and ϕ as identity. Then $\boldsymbol{x}_t = \boldsymbol{w}_t$, i.e. a white Gaussian process, and $\boldsymbol{y}_t$ is also temporally white with $\boldsymbol{y}_t \sim N(\boldsymbol{0}, \boldsymbol{C}\boldsymbol{Q}\boldsymbol{C}^t + \boldsymbol{R})$. W.l.o.g. we can set $\boldsymbol{Q} = \boldsymbol{I}$, and *with* l.o.g., we will assume that $\boldsymbol{R}$ is diagonal. Then, $\boldsymbol{y}_t \sim N(\boldsymbol{0}, \boldsymbol{C}\boldsymbol{C}^t + \boldsymbol{R})$ with $\boldsymbol{C} \in \mathbb{R}^{n,M}$, which is the standard factor analysis

model. Note that $\boldsymbol{C}\boldsymbol{C}^t = \sum_j \boldsymbol{c}_j \boldsymbol{c}_j^t$ if $\boldsymbol{c}_j$ are the columns of $\boldsymbol{C}$, and compare this to (4.5). For $M \ll n$, the rationale behind this model is that the high-dimensional data is generated by latent "causes" in a much lower-dimensional space, but then corrupted by noise which obscures the simple structure. This is of course a form of capacity control (see [BSW96] for a generalization of this idea). The assumption of white Gaussian noise is artificial, but allows to solve this model analytically.

If we further restrict the model by assuming $\boldsymbol{R} = \sigma^2 \boldsymbol{I}$, we arrive at *sensible principal component analysis (SPCA)* or *probabilistic principal component analysis (PPCA)* (see [Row98],[TB98]). We note that narrowing down our variational model class in this way would allow us to apply Skilling approximation techniques to speed up the variational algorithm for large datasets, but we are not yet sure if this decrease in modeling power is acceptable for our purposes. This issue is discussed in section C.2.

Appendix B

Skilling approximations

A number of powerful approximation techniques applicable to Bayesian analysis have been suggested in [Ski89], and some of them have been applied to Gaussian processes in [Gib97]. These techniques should have a wide applicability in all sorts of optimization problems, and we give a short introduction here, as they can be applied in an efficient implementation of the variational model selection algorithm described in this thesis.

B.1 Conjugate gradients optimization

Let us first recall some well-known facts about *conjugate gradients (CG)* optimization of quadratic functions. For more details see [Lue84] or [Fle80]. Hestenes and Stiefel [HS52] suggested the use of conjugate gradients to efficiently solve large sparse linear systems. Much later the algorithm was generalized by Fletcher and Reeves (see [Fle80] for citations and more details on the history) to cope with nonlinear criteria and quickly became the method of choice to attack very large unconstrained nonlinear systems with linear storage requirements only.

We concentrate on the following problem: Given a symmetric, positive definite matrix $\boldsymbol{C} \in \mathbb{R}^{n,n}$ and a residue vector $\boldsymbol{b}$, maximize

$$Q(\boldsymbol{y}) = \boldsymbol{b}^t \boldsymbol{y} - \frac{1}{2} \boldsymbol{y}^t \boldsymbol{C} \boldsymbol{y}. \tag{B.1}$$

The unique maximum point is $\hat{\boldsymbol{y}} = \boldsymbol{C}^{-1}\boldsymbol{b}$. The basic idea of conjugate gradients is to construct the nested sequence of subspaces $S_r = [\boldsymbol{b}, \boldsymbol{C}\boldsymbol{b}, \ldots, \boldsymbol{C}^{r-1}\boldsymbol{b}]$[1]

[1] S_r is spanned by the vectors inside the brackets.

and a sequence $\boldsymbol{y}_r$ such that $\boldsymbol{y}_r$ is the maximum point of (B.1) within S_r. After $r \leq n$ steps, we have $\dim S_r < r$ for the first time, i.e. the exist a polynomial p such that $\boldsymbol{C}p(\boldsymbol{C})\boldsymbol{b} - \boldsymbol{b} = \boldsymbol{0}$, or $p(\boldsymbol{C})\boldsymbol{b} = \boldsymbol{C}^{-1}\boldsymbol{b}$. Therefore, $\hat{\boldsymbol{y}}$ must be in S_r and $\boldsymbol{y}_r = \hat{\boldsymbol{y}}$, since the latter is the unconstrained maximum point. The conjugate gradients algorithm is able to build the sequence $\boldsymbol{y}_r$ using one matrix-vector multiplication involving $\boldsymbol{C}$ and some $O(n)$ operations per iteration. We will not go into details of the algorithm, but only describe some properties we need for our purposes here. S_r is spanned by the gradients $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_r$ where $\boldsymbol{g}_i = \nabla Q(\boldsymbol{y}_i) = \boldsymbol{b} - \boldsymbol{C}\boldsymbol{y}_i$. By definition, $\boldsymbol{g}_j$ is orthogonal to S_{j-1}, therefore the gradient set is an orthogonal base of S_r. Let $\boldsymbol{e}_r = \boldsymbol{g}_r / \|\boldsymbol{g}_r\|$ and $\boldsymbol{E}_r = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_r) \in \mathbb{R}^{n,r}$. Then we have $\boldsymbol{E}_r^t \boldsymbol{E}_r = \boldsymbol{I}_r$, and it is easy to show from the CG update equations that $\boldsymbol{T}_r = \boldsymbol{E}_r^t \boldsymbol{C} \boldsymbol{E}_r$ is *tridiagonal*.

If $\boldsymbol{C}$ is not sparse, it seems that we don't gain a great deal by using the CG method. In general, we cannot hope that the $\boldsymbol{C}^i\boldsymbol{b}$ become linearly dependent for r significantly smaller than n. We therefore need $O(n)$ steps, each being $O(n^2)$, which is the same as an ordinary $O(n^3)$ decomposition method. However, we can stop the algorithm after $r \ll n$ iterations and use $\boldsymbol{y}_r$ as an *approximation* to $\hat{\boldsymbol{y}}$. But why use CG to derive an approximative solution? Consider an arbitrary optimization method for (B.1) which outputs $\boldsymbol{y}_r = P_{r-1}(\boldsymbol{C})\boldsymbol{b}$ after the r-th iteration, where P_{r-1} is a polynomial of degree $\leq r - 1$. This is a large subclass of the set of all optimization methods restricted to one matrix-vector multiplication and a constant number of $O(n)$ operations per iteration. The CG algorithm is optimal within this subclass, since it outputs the maximum point of (B.1) within S_r. If we write

$$E(\boldsymbol{y}) = \frac{1}{2}(\boldsymbol{y} - \hat{\boldsymbol{y}})^t \boldsymbol{C} (\boldsymbol{y} - \hat{\boldsymbol{y}}) = -Q(\boldsymbol{y}) + \frac{1}{2}\boldsymbol{b}^t \boldsymbol{C}^{-1} \boldsymbol{b}, \tag{B.2}$$

one can show (see [Lue84]) that

$$E(\boldsymbol{y}_{r+1}) \leq \min_{P_r} \max_{\lambda_i} \left(1 + \lambda_i P_r(\lambda_i)\right) E(\boldsymbol{y}_0), \tag{B.3}$$

where the minimum is over all polynomials of degree $\leq r$ and the maximum is over the spectrum of $\boldsymbol{C}$. Choosing special polynomials one can show that the typical behaviour of CG exhibits a very fast increase of Q in the first $r \ll n$ iterations.

B.2 Convergence bounds for CG

By exploiting the tridiagonal form of $\boldsymbol{C}$ in the base spanned by the normalized gradients and using a cleverly chosen parallel CG problem alongside with

(B.1), Skilling shows how to obtain rigorous bounds on $Q_{max} = Q(\hat{\boldsymbol{y}})$ if $\boldsymbol{C}$ exhibits a particular form which occurs often in the context of Bayesian analysis.

Let $\boldsymbol{A}$ be positive semidefinite. We observe that any CG problem using the matrix $\boldsymbol{C} = \alpha\boldsymbol{I} + \theta\boldsymbol{A},\ \alpha \geq 0, \theta > 0$ and the residue $\boldsymbol{b}$ gives rise to the same sequence of subspaces S_r. Note that in the case $\boldsymbol{C} \propto \boldsymbol{A}$ the maximum point might not be unique and the sequence $\boldsymbol{y}_r$ produced by CG is only one out of an infinite number of possibilities, but all these give rise to the unique sequence $Q_r = Q(\boldsymbol{y}_r)$ of consecutive maxima within S_r.

We consider the case $\boldsymbol{C} = \alpha\boldsymbol{I}+\boldsymbol{A}$. This situation arises naturally in Bayesian analysis if we have a Gaussian prior with covariance $\alpha\boldsymbol{I}$ on some latent variables $\boldsymbol{y}$ and search for the posterior mode. In this setting $\boldsymbol{A}$ corresponds to the Hessian of the likelihood for a certain fixed $\boldsymbol{y}$, and the Newton-Raphson step to update $\boldsymbol{y}$ involves solving a linear system with matrix $\boldsymbol{C}$. If we start with $\boldsymbol{y}_1 = \boldsymbol{0}$, the CG iterations render an increasing sequence of *lower bounds*

$$0 = Q_1 \leq Q_2 \leq \cdots \leq Q_{max} = \frac{1}{2}\boldsymbol{b}^t\boldsymbol{C}^{-1}\boldsymbol{b} \tag{B.4}$$

on the maximum Q_{max}. Furthermore, the orthogonal base is built up in $\boldsymbol{E}_r$. Now consider the related problem to maximize

$$Q^*(\boldsymbol{y}^*) = \boldsymbol{b}^t\boldsymbol{A}\boldsymbol{y}^* - \frac{1}{2}\boldsymbol{y}^{*t}\boldsymbol{C}\boldsymbol{A}\boldsymbol{y}^*. \tag{B.5}$$

CG can be generalized to compute a (non unique) sequence $\boldsymbol{y}_r^*$ such that $\boldsymbol{y}_r^*$ maximizes (B.5) over S_r, by transforming $\boldsymbol{x} = \boldsymbol{A}^{1/2}\boldsymbol{y}^*$ and $\boldsymbol{v} = \boldsymbol{A}^{1/2}\boldsymbol{b}$, thus

$$Q^*(\boldsymbol{y}^*) = \boldsymbol{v}^t\boldsymbol{x} - \frac{1}{2}\boldsymbol{x}^t\boldsymbol{C}\boldsymbol{x}, \tag{B.6}$$

writing down the CG equations for the latter problem and undoing the transform by dropping the leading $\boldsymbol{A}^{1/2}$ matrices in the equations. This is possible since, by the special form of $\boldsymbol{C}$, we have $\boldsymbol{C}\boldsymbol{A}^{1/2} = \boldsymbol{A}^{1/2}\boldsymbol{C}$. Starting with $\boldsymbol{y}_1^* = \boldsymbol{0}$, we have the sequence of lower bounds

$$0 = Q_1^* \leq Q_2^* \leq \cdots \leq Q_{max}^* = \frac{1}{2}\boldsymbol{b}^t\boldsymbol{C}^{-1}\boldsymbol{A}\boldsymbol{b}. \tag{B.7}$$

Now, here's the trick:

$$Q_r^* + \alpha Q_{max} \leq Q_{max}^* + \alpha Q_{max} = \frac{1}{2}\boldsymbol{b}^t\boldsymbol{b}, \tag{B.8}$$

which gives us a decreasing sequence of *upper bounds* on Q_{max}. All in all we have for any r

$$Q_r \leq Q_{max} \leq \alpha^{-1} \left(\frac{1}{2} \boldsymbol{b}^t \boldsymbol{b} - Q_r^* \right). \tag{B.9}$$

If G_r denotes the difference between upper and lower bound after iteration r, we know that $Q_{max} - Q_r \leq G_r$. Let $\delta \boldsymbol{y}_r = \boldsymbol{y}_r - \hat{\boldsymbol{y}}$. We can bound $\|\delta \boldsymbol{y}_r\|$ as follows: $Q_{max} - Q_r = (1/2)\delta \boldsymbol{y}_r^t \boldsymbol{C} \delta \boldsymbol{y}_r \geq (\alpha/2)\delta \boldsymbol{y}_r^t \delta \boldsymbol{y}_r$, since α is a lower bound on all eigenvalues of $\boldsymbol{C}$. Thus

$$\|\delta \boldsymbol{y}_r\|^2 \leq \frac{2G_r}{\alpha}. \tag{B.10}$$

However, this bound proved to be quite loose in our experiments.

A straightforward implementation of these ideas would run both CG algorithms in parallel, requiring 3 matrix-vector multiplications per iteration. However, by exploiting the simple structure of $\boldsymbol{C}$ *and* related matrices like $\boldsymbol{CA}$ in the gradient basis, we can compute the Q_r^* (and therefore the upper bound) with *negligible additional cost* alongside the CG optimization of (B.1). The details of an implementation are, however, tedious and will not be given here. Several manipulations of the basic scheme have to be done to ensure stability. For example, it is wise to compute the orthonormal bases of the S_r using CG on $\boldsymbol{A}$ instead of the original $\boldsymbol{C} = \alpha \boldsymbol{I} + \boldsymbol{A}$ (as mentioned above, the subspaces are the same, see [Gib97] for some details). We plan to compile our "practical experiences" with this algorithm in a seperate technical report.

Remark In the above derivation, we implicitely assumed the starting point $\boldsymbol{y}_1 = \boldsymbol{0}$. This is suboptimal if we have a good guess for $\hat{\boldsymbol{y}}$. Choosing $\boldsymbol{0}$ as starting point has the advantage that the original algorithm can be converted into a numerically more stable form (see [Gib97] for details). Some special applications of the algorithm, such as estimating $\log |\boldsymbol{C}|$ (as discussed in the next section) require the choice $\boldsymbol{y}_0 = \boldsymbol{0}$. In any other case, our implementation usually starts with $\boldsymbol{y}_0$ specified by the user. If the optimization fails, we try again, starting at $\boldsymbol{0}$ and employing the more stable version.

B.3 Rotationally invariant functions

Other terms frequently arising during Bayesian analysis are determinants and traces. These are special cases of functions $f(\boldsymbol{C})$ which are *rotationally invariant* in the sense that $f(\boldsymbol{C}) = f(\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^t)$ for any rotation $\boldsymbol{U}$. f can be

written as $f(\boldsymbol{C}) = g(\lambda_1, \ldots, \lambda_n)$ where $\{\lambda_i\}$ is the spectrum of $\boldsymbol{C}$ and g is invariant against permutations of its arguments. An important special case is given by choosing an arbitrary scalar function ϕ, defining $\phi(\boldsymbol{C}) = \boldsymbol{U}\phi(D)\boldsymbol{U}^t$ where $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^t$ is the spectral (or singular value) decomposition of $\boldsymbol{C}$, and setting $f(\boldsymbol{C}) = \operatorname{tr}\phi(\boldsymbol{C})$. Note some special cases:

- $\phi = (\cdot)^{-1}$ gives $f(\boldsymbol{C}) = \operatorname{tr}\boldsymbol{C}^{-1}$
- $\phi = \log$ gives $f(\boldsymbol{C}) = \log|\boldsymbol{C}|$

Consider an algorithm which chooses a seed $\boldsymbol{r} \sim P(\boldsymbol{r})$, computes $\boldsymbol{r}^t\phi(\boldsymbol{C})\boldsymbol{r}$ and estimates $f(\boldsymbol{C})$ on the basis of this information. If $\boldsymbol{C}$ is chosen uniformly, we have no reason to prefer a particular seed $\boldsymbol{r}$ over $\boldsymbol{U}\boldsymbol{r}$ for any rotation $\boldsymbol{U}$, so we can choose a spherical distribution $P(\boldsymbol{r})$. The simplest choice is $P(\boldsymbol{r}) = N(\boldsymbol{0}, \boldsymbol{I})$. The estimator $x = \boldsymbol{r}^t\phi(\boldsymbol{C})\boldsymbol{r}$ has mean $\mathrm{E}x = \operatorname{tr}\phi(\boldsymbol{C}) = f(\boldsymbol{C})$ and variance $2\operatorname{tr}(\phi(\boldsymbol{C})^2)$. For large n, both terms grow as $O(n)$ (they are traces), therefore the *relative error* is $O(1/\sqrt{n})$. The estimate gets better as the dimension grows, there is blessing instead of curse in the latter! If we draw L seeds $\boldsymbol{r}_i$ independently from $P(\boldsymbol{r})$ and take the sample average, the relative error is $O(1/\sqrt{Ln})$. Therefore, if the term $\boldsymbol{r}^t\phi(\boldsymbol{C})\boldsymbol{r}$ can be computed or approximated efficiently, we have an excellent estimator for $f(\boldsymbol{C})$.

If $\phi = (\cdot)^{-1}$, x is called *randomized trace estimator*. We have to compute terms $x = \boldsymbol{r}^t\boldsymbol{C}^{-1}\boldsymbol{r}$. This can be done approximately using the CG technique described in the previous section. Note that since $Q_{max} = (1/2)\boldsymbol{r}^t\boldsymbol{C}^{-1}\boldsymbol{r} = x/2$, the algorithm gives rigorous upper and lower bounds on the term of interest with negligible extra cost relative to the usual CG algorithm. The randomized trace estimator together with the CG method can also be used to estimate trace terms like $\operatorname{tr}\boldsymbol{C}^{-1}\boldsymbol{B}$ or more complicated combinations of $\boldsymbol{C}^{-1}$ and other matrices. However, in such cases usually no upper bounds on x can be given.

Skilling also shows how to estimate $f(\boldsymbol{C}) = \log|\boldsymbol{C}|$ which corresponds to $\phi = \log$, and the estimator averages $x = \boldsymbol{r}^t\log(\boldsymbol{C})\boldsymbol{r}$. This requires starting the CG optimization with $\boldsymbol{y}_0 = \boldsymbol{0}$ (see discussion above). After the r-th iteration we have $\boldsymbol{T}_r = \boldsymbol{E}_r^t\boldsymbol{C}\boldsymbol{E}_r$. We now approximate x by replacing $\boldsymbol{C}$ by $\boldsymbol{E}_r\boldsymbol{T}_r\boldsymbol{E}_r^t$. The latter matrix has rank r and is identical to $\boldsymbol{C}$ for $r = n$. Let $\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^t$ be the spectral decomposition of $\boldsymbol{T}_r$. We have

$$\tilde{x} = \boldsymbol{r}^t\boldsymbol{E}_r\boldsymbol{L}\log(\boldsymbol{D})\boldsymbol{L}^t\boldsymbol{E}_r^t\boldsymbol{r}. \tag{B.11}$$

The computation of $\boldsymbol{D}$ from $\boldsymbol{T}_r$ is $O(r^2)$ using for example the QL algorithm with implicit shifting (see [PTVF92]), and $\boldsymbol{v} = \boldsymbol{L}^t\boldsymbol{E}_r^t\boldsymbol{r}$ is a byproduct of this

method. Rigorous upper and lower bounds on x can also be computed with negligible extra cost, as described in [Ski89]. Especially it turns out that $\tilde{x}$ is a rigorous lower bound on x.

Appendix C

The variational free energy and its gradients

In this chapter, we compute the gradients of the variational free energy criterion of subsection 4.2.1. We show how to compute or approximate the gradients and the criterion efficiently, using techniques described in chapter B of the appendix.

C.1 The gradients

We will begin by deriving the expressions for F and its gradients and after that give some comments on how to efficiently approximate them. Recall the definition (4.2) of the *variational free energy* $F(\tilde{P}, \boldsymbol{\theta})$. From an implementational viewpoint it is advantageous to normalize F by dividing through n, we will do so in what follows. Recall the definition of $\boldsymbol{\nu} = (\nu_i)_i$ from (4.8). The one-dimensional Gaussian expection can, depending on the actual loss function g, either be computed analytically or approximated using numerical quadrature. We will return to this issue below, but will assume in the following that ν_i is differentiable w.r.t. μ_i and σ_i^2. This is clearly the case for differentiable g, but it is important to note that this is not a necessary condition. The family Γ of variational distributions $\tilde{P}$ was introduced in subsection 4.2.1. From (4.7) we have

$$\begin{aligned} F(\tilde{P}, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nu_i + \frac{1}{2n} \boldsymbol{\mu}^t \boldsymbol{K}^{-1} \boldsymbol{\mu} + \frac{1}{2n} \operatorname{tr}\left(\Sigma \boldsymbol{K}^{-1}\right) \\ + \frac{1}{2n} \log |\boldsymbol{K}| - \frac{1}{2n} \log |\Sigma| - \frac{1}{2}. \end{aligned} \tag{C.1}$$

Differentiating w.r.t. the parameters of $\tilde{P}$, we arrive at

$$\begin{aligned} \nabla_{\boldsymbol{\mu}} F &= \frac{1}{n}\left(\left(\frac{\partial \nu_i}{\partial \mu_i}\right)_i + \boldsymbol{K}^{-1}\boldsymbol{\mu}\right) \\ \nabla_{\Sigma} F &= \frac{1}{n}\left(\operatorname{diag}\left(\frac{\partial \nu_i}{\partial \sigma_i^2}\right)_i + \frac{1}{2}\boldsymbol{K}^{-1} - \frac{1}{2}\Sigma^{-1}\right). \end{aligned} \tag{C.2}$$

Recall the structure of Σ, defined by (4.5) and let $\boldsymbol{W} = \operatorname{diag}(\partial \nu_i / \partial \sigma_i^2)_i$. We end up with

$$\begin{aligned} \nabla_{\boldsymbol{D}} F &= \frac{1}{n}\left(\boldsymbol{W} + \frac{1}{2}\operatorname{diag}\boldsymbol{K}^{-1} - \frac{1}{2}\operatorname{diag}\Sigma^{-1}\right) \\ \nabla_{\boldsymbol{c}_j} F &= \frac{1}{n}\left(2\boldsymbol{W}\boldsymbol{c}_j + \boldsymbol{K}^{-1}\boldsymbol{c}_j - \Sigma^{-1}\boldsymbol{c}_j\right). \end{aligned} \tag{C.3}$$

Now consider the gradient of F w.r.t. $\boldsymbol{\theta}$ for fixed $\tilde{P}$. We have

$$\nabla_{\boldsymbol{K}} F = -\frac{1}{2n}\left(\left(\boldsymbol{K}^{-1}\boldsymbol{\mu}\right)\left(\boldsymbol{K}^{-1}\boldsymbol{\mu}\right)^t + \boldsymbol{K}^{-1}\Sigma\boldsymbol{K}^{-1} - \boldsymbol{K}^{-1}\right), \tag{C.4}$$

therefore

$$\nabla_{\theta_i} F = -\frac{1}{2n}\left(\left(\boldsymbol{K}^{-1}\boldsymbol{\mu}\right)^t \boldsymbol{G}_i \left(\boldsymbol{K}^{-1}\boldsymbol{\mu}\right) + \operatorname{tr}\left(\boldsymbol{K}^{-1}\left(\Sigma\boldsymbol{K}^{-1} - \boldsymbol{I}\right)\boldsymbol{G}_i\right)\right), \tag{C.5}$$

where $\boldsymbol{G}_i = \partial \boldsymbol{K} / \partial \theta_i$ and θ_i denotes the i-th component of $\boldsymbol{\theta}$.

C.2 Efficient computation or approximation

Because of the special structure of the covariance Σ, we can compute $\log|\Sigma|$, $\operatorname{diag}\Sigma^{-1}$ and $\Sigma^{-1}\boldsymbol{c}_j$, $j = 1, \ldots, M$ efficiently using the Sherman–Morrison formula (see [PTVF92], p.73). Let $\Sigma_0 = \boldsymbol{D}$ and $\Sigma_i = \Sigma_{i-1} + \boldsymbol{c}_i\boldsymbol{c}_i^t$, $\boldsymbol{D}_i = \operatorname{diag}\Sigma_i^{-1}$ and $\boldsymbol{v}_{ij} = \Sigma_i^{-1}\boldsymbol{c}_j$. Our aim is to calculate $\boldsymbol{D}_M$ and $\boldsymbol{v}_{Mj}$, $j = 1, \ldots, M$. This can easily be done using a dynamic programming approach in which we retain the entities $\boldsymbol{v}_{i-1,j}$ and $\boldsymbol{D}_{i-1}$ to compute $\boldsymbol{v}_{i,j}$ and $\boldsymbol{D}_i$. For $i = 0$, we have $\boldsymbol{D}_0 = \boldsymbol{D}^{-1}$ and $\boldsymbol{v}_{0j} = \boldsymbol{D}^{-1}\boldsymbol{c}_j$. In the i-th step, we first compute $\beta_i = (1 + \boldsymbol{c}_i^t\boldsymbol{v}_{i-1,i})^{-1}$, then

$$\boldsymbol{v}_{i,j} = \boldsymbol{v}_{i-1,j} - \beta_i\left(\boldsymbol{c}_j^t\boldsymbol{v}_{i-1,i}\right)\boldsymbol{v}_{i-1,i} \tag{C.6}$$

and

$$\boldsymbol{D}_i = \boldsymbol{D}_{i-1} - \beta_i \operatorname{diag}\left(\boldsymbol{v}_{i-1,i}\boldsymbol{v}_{i-1,i}^t\right). \tag{C.7}$$

The log determinant of Σ can be computed using the identity

$$\log|\Sigma_i| = \log|\Sigma_{i-1}| + \log|\boldsymbol{I} + \boldsymbol{v}_{i-1,i}\boldsymbol{c}_i^t| = \log|\Sigma_{i-1}| + \log(1 + \boldsymbol{v}_{i-1,i}^t\boldsymbol{c}_i) \tag{C.8}$$

and noting that the update term is just $-\log\beta_i$. Every step is $O(Mn)$, so we have an overall cost of $O(M^2n)$. We have to store the vectors $\boldsymbol{v}_{i,j}$, $j = 1,\ldots,M$ and some further $O(n)$ vectors, so the storage cost is $O(Mn)$.

If n is not too large, it might be feasible to recompute the inverse of $\boldsymbol{K}$ after each change of $\boldsymbol{\theta}$. In this case the computation of the criterion and the gradient w.r.t. the parameters of $\tilde{P}$ is straightforward. The gradient w.r.t. $\boldsymbol{\theta}$ (C.5) is more problematic since the right trace term is cubic in n, even if the inverse of the covariance is known.

In general, we can resort to approximations suggested in [Ski89]. We give a description of these techniques in Appendix B. Consider (C.1). $\boldsymbol{K}^{-1}\boldsymbol{\mu}$ is approximated using CG, the trace term by using the randomized trace method together with CG to compute $\boldsymbol{v}_i = \boldsymbol{K}^{-1}\boldsymbol{u}_i$ for $N(\boldsymbol{0},\boldsymbol{I})$ random vectors $\boldsymbol{u}_i$. The same vectors can be used during all iterations of the inner loop. $\log|\boldsymbol{K}|$ can be estimated using the technique described in section B.3. This is done alongside the computation of the $\boldsymbol{v}_i$, with virtually no additional cost. Now to (C.5). Using the same sample $\{\boldsymbol{u}_i\}$ as above, we compute $\boldsymbol{w}_i = \boldsymbol{K}^{-1}\Sigma\boldsymbol{u}_i$, using CG. Note that alongside the trace term in (C.1) can be estimated at no extra cost, using

$$\operatorname{tr}\Sigma\boldsymbol{K}^{-1} \approx \frac{1}{L}\sum_{j=1}^{L}\boldsymbol{u}_j^t\boldsymbol{w}_j. \tag{C.9}$$

The estimate for the trace in (C.5) is

$$\frac{1}{L}\sum_{j=1}^{L}\boldsymbol{w}_j^t\boldsymbol{G}_i\boldsymbol{v}_j. \tag{C.10}$$

$\nabla_{\boldsymbol{D}}F$ in (C.3) remains problematic because $\operatorname{diag}\boldsymbol{K}^{-1}$ cannot be estimated efficiently using Skilling's techniques. If an inversion (via Cholesky decomposition, see [PTVF92]) of $\boldsymbol{K}$ per outer loop iteration can be afforded, this should be considered. Doing so would render all applications of CG superfluous, and the only remaining approximation would be the trace term in (C.5)[1]. Actually, $\operatorname{diag}\boldsymbol{K}^{-1}$ can be computed from the Cholesky decomposition of $\boldsymbol{K}$

[1] Another matrix multiplication is needed to compute this gradient exactly.

in $O(n^2)$, and $\log|\boldsymbol{K}|$ is a byproduct. Computing only the decomposition requires roughly half of the time needed to invert $\boldsymbol{K}$. Another option is to narrow down the class of admissable covariances Σ. For example, imposing $\boldsymbol{D} = (\alpha/n)\boldsymbol{I}$, $\alpha > 0$[2] for the diagonal part of Σ reduces the degrees of freedom of Σ by $n-1$ only, while the problematic gradient changes to

$$\nabla_\alpha F = \frac{1}{n^2}\left(\operatorname{tr}\boldsymbol{W} + \frac{1}{2}\operatorname{tr}\left(\Sigma^{-1} - \boldsymbol{K}^{-1}\right)\right) \tag{C.11}$$

which can be approximated using the randomized trace method. In fact, the estimate of $\operatorname{tr}\boldsymbol{K}^{-1}$ is a byproduct of the computation of the $\boldsymbol{v}_i$ vectors described above. This manipulation also simplifies the computation of (C.1) since

$$\operatorname{tr}\left(\Sigma\boldsymbol{K}^{-1}\right) = \frac{\alpha}{n}\operatorname{tr}\boldsymbol{K}^{-1} + \sum_{j=1}^{M}\boldsymbol{c}_j^t\boldsymbol{K}^{-1}\boldsymbol{c}_j, \tag{C.12}$$

and all the quantities involved have to be computed for (C.3) anyway. However, more experiments have to be done to decide whether the reduced class of variational distributions can compete with the original factor-analyzed covariances class.

We now give detailed descriptions of the computations. To bring in some structure, we first identify some "building blocks" and then show how to combine them to compute the quantities we need. We describe the approximative computations using the reduced variational class (with $\boldsymbol{D} = \alpha\boldsymbol{I}$) since the exact computations are trivial. When giving running time estimates, we assume that the CG method described in chapter B requires a number of iterations *independent* of the dimension n until a sufficiently small gap between upper and lower bound of Q_{max} is achieved. This is surely wrong in a worst case sense and we don't know if it holds in the expected case, using some sensible distribution over the linear system matrices. We expect $\boldsymbol{K}$ not to be sparse, so we count $O(n^2)$ for matrix-vector multiplications involving the covariance. The building blocks are:

1. Draw L independent sample points $\boldsymbol{u}_i$ from $N(0, \boldsymbol{I})$ and approximate $\boldsymbol{v}_i = \boldsymbol{K}^{-1}\boldsymbol{u}_i$. Alongside the computation of the $\boldsymbol{v}_i$, the approximations to $\operatorname{tr}\boldsymbol{K}^{-1}$ and $\log|\boldsymbol{K}|$ can be accumulated at virtually no extra cost. The cost of this block is $O(Ln^2)$.

2. Approximate $\boldsymbol{w}_i = \boldsymbol{K}^{-1}\Sigma\boldsymbol{u}_i$, $i = 1, \ldots, L$. Alongside, an estimate (C.9) of $\operatorname{tr}\Sigma\boldsymbol{K}^{-1}$ can be accumulated at no extra cost. Note that, given the

[2]We choose the parameterization $\alpha = \gamma^2$ to fulfil this constraint automatically.

$\boldsymbol{d}_i$ vectors from block 4, the $\boldsymbol{w}_i$ can be calculated using no Skilling approximation at all:

$$\boldsymbol{w}_i = \frac{\alpha}{n}\boldsymbol{v}_i + \sum_{j=1}^{M} \left(\boldsymbol{u}_i^t \boldsymbol{c}_j\right) \boldsymbol{d}_j, \tag{C.13}$$

and $\operatorname{tr} \Sigma \boldsymbol{K}^{-1}$ can be computed via (C.12). In this case, the cost of the block is $O(LMn)$. If the $\boldsymbol{d}_i$ are not used, the cost is $O(Ln^2)$.

3. Compute the ν_i and its gradients. This can either be done exactly or using numerical quadrature, depending on the loss function g. The cost is $O(n)$.

4. Approximate $\boldsymbol{d}_i = \boldsymbol{K}^{-1}\boldsymbol{c}_i$, $i = 1, \ldots, M$. The cost is $O(Mn^2)$.

5. Approximate $\boldsymbol{m} = \boldsymbol{K}^{-1}\boldsymbol{\mu}$. The cost is $O(n^2)$.

6. Compute $\operatorname{tr} \Sigma^{-1}$, $\log|\Sigma|$ and $\Sigma^{-1}\boldsymbol{c}_j$ using the techniques described above. If only $\log|\Sigma|$ is required, there is a faster method. The cost of this block is $O(M^2 n)$.

An outer loop iteration of the variational free energy minimization method can now be described as follows:

1. Compute the covariance matrix $\boldsymbol{K}$.

2. Prepare for the inner loop by performing block 1.

3. Inner loop. Minimize F w.r.t. $\tilde{P}$. Every step requires the evaluation of the gradient $\nabla_{\tilde{P}} F$, i.e. blocks 3, 4,5 and full 6.

4. Update of $\boldsymbol{\theta}$. The covariance $\boldsymbol{K}$ and the $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ vectors can be recycled. We need blocks 2 and 5 to compute the gradients $\nabla_{\theta_i} F$. If $K < L$, it pays to precede block 2 by 4 and to use the $\boldsymbol{d}_i$ vectors. The computation of the relevant part of F is no big additional burden. Depending on the optimizer used in the inner loop, we might also be able to recycle $\boldsymbol{m}$ and the $\boldsymbol{d}_i$ vectors.

As for the choice of L, i.e. the number of terms in the randomized trace and log determinant estimates we observed in preliminary experiments that as long as n is reasonably large, L can be chosen very small without affecting the relative accuracy very much. However, we have to bear in mind that these estimates *have* a certain relative error which can spoil an optimizer

that expects the criterion values to be very accurate. An optimizer fed with the approximative quantities discussed above should use a stopping criterion based on gradient size, *not* on running differences of criterion values. Furthermore, the line search routine has to be designed with some care to be able to bracket a line minimum based on noisy criterion values (see discussion in section 5.2). The choice of M is a modeling decision, being a classical trade-off between model accuracy and computational efficiency.

C.2.1 The variance parameter

Note that if the variance parameter C of the kernel K becomes large, computations based on the covariance matrix $\boldsymbol{K}$ and its inverse can cause numerical instabilities. We found it necessary to separate C from the other kernel parameters in $\boldsymbol{\theta}$, i.e. substituting $K = CR$ into (C.1) which results in

$$\begin{aligned} F(\tilde{P}, \boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\nu_i + \frac{1}{2Cn}\boldsymbol{\mu}^t\boldsymbol{R}^{-1}\boldsymbol{\mu} + \frac{1}{2Cn}\operatorname{tr}\left(\Sigma\boldsymbol{R}^{-1}\right) \\ + \frac{1}{2n}\log|\boldsymbol{R}| + \frac{1}{2}\log C - \frac{1}{2n}\log|\Sigma| - \frac{1}{2}. \end{aligned} \tag{C.14}$$

C.2.2 Computation of loss-related terms

Some comments about the computation of ν_i and its derivatives are in order. For special choices of the loss function g, the integral of (4.8) can be solved analytically. The unnormalized SVC loss $g(t, y) = [1 - ty]_+$ is such a case, since

$$\begin{aligned} \nu_i &= \int [1 - t_i y]_+ N(y|\mu_i, \sigma_i^2)\, dy \\ &= -\int_{-\infty}^{0} uN(u|t_i\mu_i - 1, \sigma_i^2) \\ &= \int_{-\infty}^{0} \left(-u + (t_i\mu_i - 1) - (t_i\mu_i - 1)\right) N(u|t_i\mu_i - 1, \sigma_i^2)\, du \\ &= \int_{-\infty}^{0} \left(\sigma_i^2 \frac{d}{du} N(u|t_i\mu_i - 1, \sigma_i^2) + (1 - t_i\mu_i)N(u|t_i\mu_i - 1, \sigma_i^2)\right) du \\ &= \sigma_i^2 N(0|t_i\mu_i - 1, \sigma_i^2) + (1 - t_i\mu_i)P(\sigma_i U + t_i\mu_i - 1 \le 0) \\ &= \frac{\sigma_i}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{1 - t_i\mu_i}{\sigma_i}\right)^2\right) + (1 - t_i\mu_i)\Phi\left(\frac{1 - t_i\mu_i}{\sigma}\right), \end{aligned} \tag{C.15}$$

where $U \sim N(0,1)$. The derivatives in this case are

$$\frac{\partial \nu_i}{\partial \mu_i} = -t_i \Phi\left(\frac{1 - t_i \mu_i}{\sigma_i}\right) \tag{C.16}$$

and

$$\frac{\partial \nu_i}{\partial \sigma_i^2} = \frac{1}{\sqrt{8\pi}\sigma_i} \exp\left(-\frac{1}{2}\left(\frac{1 - t_i \mu_i}{\sigma_i}\right)^2\right). \tag{C.17}$$

We also have

$$\frac{\partial^2 \nu_i}{\partial \mu_i^2} = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{1}{2}\left(\frac{1 - t_i \mu_i}{\sigma_i}\right)^2\right) = 2\frac{\partial \nu_i}{\partial \sigma_i^2}. \tag{C.18}$$

If (4.8) is not analytically tractable, we may resort to numerical quadrature techniques like *Gaussian quadrature* (see [PTVF92], p.154). If g is well-behaved in the sense that we can interchange differentiation and integration, we have

$$\begin{aligned} \frac{\partial \nu_i}{\partial \mu_i} &= \frac{\partial}{\partial \mu_i} \int g(t_i, y_i) N(y_i | \mu_i, \sigma_i^2)\, dy_i \\ &= \int g(t_i, y_i) \frac{y_i - \mu_i}{\sigma_i^2} N(y_i | \mu_i, \sigma_i^2)\, dy_i \end{aligned} \tag{C.19}$$

and

$$\begin{aligned} \frac{\partial \nu_i}{\partial \sigma_i^2} &= \frac{\partial}{\partial \sigma_i^2} \int -g(t_i, y_i) N(y_i | \mu_i, \sigma_i^2)\, dy_i \\ &= \int -g(t_i, y_i) \frac{(y_i - \mu_i)^2}{\sigma_i^3} N(y_i | \mu_i, \sigma_i^2)\, dy_i, \end{aligned} \tag{C.20}$$

which are Gaussian expectations themselves. If g is differentiable and a linear quadrature scheme is used, the derivatives of the quadrature approximation are identical to the quadrature approximation of the expectation over the derivatives of g. This is the case for the Bernoulli loss $g(t, y) = \log(1 + \exp(-ty))$.

C.2.3 Upper bound on loss normalization factor

In the previous subsection we have been concerned with the unnormalized SVC loss $g(t, y) = [1 - ty]_+$ only. However, we decided to replace this by

its normalized counterpart, corresponding to the noise distribution (2.18), therefore we have to find an approximation to

$$\int \log Z(y) N(y|\mu, \sigma^2)\, dy \tag{C.21}$$

where

$$Z(y) = \exp(-C[1-y]_+) + \exp(-C[1+y]_+) \tag{C.22}$$

is the normalization factor. In the following we describe a tight upper bound to (C.21) which can easily be evaluated. Although we could also employ a simple quadrature scheme, we feel more comfortable with the bound since it matches the nondifferentiable form of the function almost perfectly. Figure C.1 plots $\log Z(y)$ against y. The plan is to use different tight upper bounds for the parts $y < -1,\ y \in [-1, 1]$ and $y > 1$. Each of the bounds is parameterized by a variational parameter, but it turns out that we can solve for the optimal values of these parameters analytically.

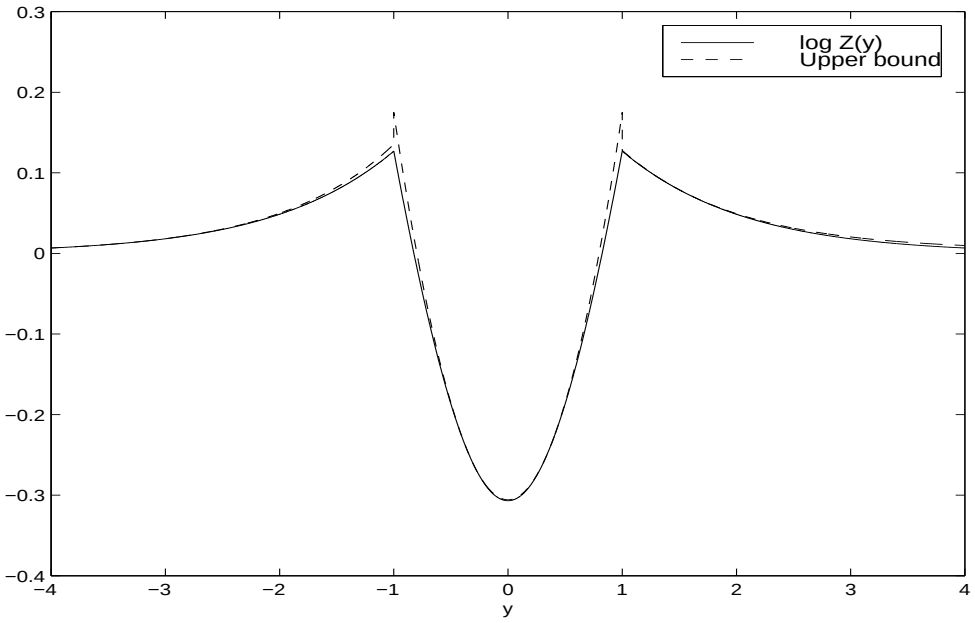

Figure C.1: Log normalization factor of SVC noise distribution for $C = 1$. Also shown is a member of the family of variational upper bounds we are using. The parameters are not optimized.

All three bounds are derived using the principle of convex duality, as described in subsection 3.1.1. First note that $\log x$ is concave and has the con-

cave dual $\log\lambda + 1$. For $|y| > 1$, we therefore have

$$\begin{aligned}\log Z(y) &= \log\left(1 + \exp(-C(1+|y|))\right) \\ &\leq \lambda\left(1 + \exp(-C(1+|y|))\right) - \log\lambda - 1.\end{aligned} \tag{C.23}$$

We choose different variational parameters for the two cases, λ_1 for $y < -1$ and λ_3 for $y > 1$. The bound (C.23) is exact for $\lambda = \sigma(C(1+|y|)) \in [1/2, 1)$. For $y \in [-1, 1]$ we employ the quadratic upper bound described in subsection 4.3.2. Following along these lines, we arrive at

$$\begin{aligned}\log Z(y) &= \log 2\cosh(Cy) - C \\ &\leq \lambda(\xi)(4C^2y^2 - \xi^2) + \log 2\cosh(\xi/2) - C.\end{aligned} \tag{C.24}$$

Here, $\lambda(\xi) = \tanh(\xi/2)/(4\xi)$. (C.24) is exact for $\xi = \pm 2Cy$. The parameter of this bound will be referred to as ξ_2, and $\lambda_2 = \lambda(\xi_2)$.

Denote the value of (C.21) by I. Plugging in the upper bound for $\log Z(y)$ and doing the integral over the Gaussian gives $B \geq I$. Note that B still depends on the variational parameters. We will do the latter integral in three steps, i.e. $B = B_1 + B_2 + B_3$. We abbreviate $N(y|\mu, \sigma^2)$ by $N(y)$. Also, in slight abuse of notation, let $\Phi(y)$ be the c.d.f. of $N(\mu, \sigma^2)$[3].

$$\begin{aligned}B_1 &= \int_{-\infty}^{-1} \left(\lambda_1\left(1 + \exp(-C(1-y))\right) - \log\lambda_1 - 1\right) N(y)\, dy \\ &= (\lambda_1 - \log\lambda_1 - 1)\Phi(-1) \\ &+ \lambda_1 \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{-1} \exp\left(-\frac{1}{2\sigma^2}(y - \mu - C\sigma^2)^2 + C(\mu - 1) + \frac{1}{2}\sigma^2C^2\right) dy \\ &= (\lambda_1 - \log\lambda_1 - 1)\Phi(-1) \\ &+ \lambda_1 \exp\left(C(\mu - 1) + \frac{1}{2}\sigma^2C^2\right)\Phi(-1 - C\sigma^2).\end{aligned} \tag{C.25}$$

B_3 is easy now,

$$\begin{aligned}B_3 &= \int_{1}^{\infty} \left(\lambda_3\left(1 + \exp(-C(1+y))\right) - \log\lambda_3 - 1\right) N(y)\, dy \\ &= \int_{-\infty}^{-1} \left(\lambda_3\left(1 + \exp(-C(1-y))\right) - \log\lambda_3 - 1\right) N(y|-\mu, \sigma^2)\, dy \\ &= (\lambda_3 - \log\lambda_3 - 1)(1 - \Phi(1)) \\ &+ \lambda_3 \exp\left(C(-\mu - 1) + \frac{1}{2}\sigma^2C^2\right)(1 - \Phi(1 + C\sigma^2)),\end{aligned} \tag{C.26}$$

[3]Everywhere else in this thesis, Φ denotes the c.d.f. of $N(0, 1)$.

using the derivation of B_1 with $\mu \to -\mu$ and $\lambda_1 \to \lambda_3$. The calculation of B_2 is the most tedious part. We begin by noting the fact

$$y^2 N(y) = \frac{\partial}{\partial y}\left(-\sigma^2(y+\mu)N(y)\right) + (\mu^2+\sigma^2)N(y). \tag{C.27}$$

Thus,

$$\begin{aligned}\int_{-1}^{1} y^2 N(y)\, dy = (\mu^2+\sigma^2)\left(\Phi(1)-\Phi(-1)\right) \\ + \sigma^2(\mu-1)N(-1) - \sigma^2(\mu+1)N(1).\end{aligned} \tag{C.28}$$

Throwing all together, we have

$$\begin{aligned}B_2 &= \int_{-1}^{1} \left(\lambda_2\left(4C^2y^2-\xi_2^2\right) + \log 2\cosh(\xi_2/2) - C\right) N(y)\, dy \\ &= \left(-\lambda_2\xi_2^2 + \log 2\cosh(\xi_2/2) - C + 4C^2\lambda_2(\mu^2+\sigma^2)\right)\left(\Phi(1)-\Phi(-1)\right) \\ &+ 4C^2\lambda_2\sigma^2\left((\mu-1)N(-1) - (\mu+1)N(1)\right).\end{aligned} \tag{C.29}$$

It turns out that the best values for the variational parameters of B_1, B_2 and B_3 can be determined analytically, and substituting them back gives the tightest possible bounds within the family. Writing (C.25) in the short notation

$$B_1 = (\lambda_1 - \log\lambda_1 - 1)\Phi(-1) + \lambda_1 \exp(-A_1) \tag{C.30}$$

where

$$A_1 = C(1-\mu) - \frac{1}{2}\sigma^2C^2 - \log\Phi(-1-C\sigma^2), \tag{C.31}$$

differentiating w.r.t. λ_1, equating to zero and solving for λ_1, we arrive at

$$\lambda_1 = \frac{\Phi(-1)}{\Phi(-1)+\exp(-A_1)} = \sigma\left(A_1 + \log\Phi(-1)\right). \tag{C.32}$$

Plugging this back into (C.30), we have

$$B_1 = \Phi(-1)\log\left(1 + \frac{\exp(-A_1)}{\Phi(-1)}\right). \tag{C.33}$$

Along the same lines, we arrive at

$$B_3 = (1-\Phi(1))\log\left(1 + \frac{\exp(-A_3)}{1-\Phi(1)}\right) \tag{C.34}$$

where

$$A_3 = C(\mu + 1) - \frac{1}{2}\sigma^2 C^2 - \log(1 - \Phi(1 + C\sigma^2)). \tag{C.35}$$

Furthermore, the derivative of (C.29) w.r.t. ξ_2 can be written as

$$\frac{\partial B_2}{\partial \xi_2} = \left(-\lambda_2' \xi_2^2 + \lambda_2' 4C^2(\mu^2 + \sigma^2)\right) A_{2,1} + \lambda_2' 4C^2 A_{2,2} \tag{C.36}$$

where

$$\begin{aligned} A_{2,1} &= \Phi(1) - \Phi(-1), \\ A_{2,2} &= \sigma^2 \left((\mu - 1)N(-1) - (\mu + 1)N(1)\right). \end{aligned} \tag{C.37}$$

Here we used $\partial \log(2\cosh(\xi_2/2))/\partial \xi_2 = 2\xi_2 \lambda_2$. Equating this to zero and noting that $\lambda_2' < 0$ for all $\xi_2 > 0$, we have

$$\xi_2 = 2C\sqrt{\mu^2 + \sigma^2 + A_{2,2}/A_{2,1}}. \tag{C.38}$$

Plugging this into (C.29) we arrive at

$$B_2 = A_{2,1}\left(\log 2\cosh(\xi_2/2) - C\right). \tag{C.39}$$

Finally, we have to compute the derivatives of B w.r.t. μ and σ^2. The derivation is tedious, and we only present the results here.

$$\frac{\partial B_1}{\partial \mu} = g_1(-1)B_1 + \lambda_1 e^{-A_1}\left(C + g_1(-1 - C\sigma^2) - g_1(-1)\right) \tag{C.40}$$

where $g_1(x) = \partial \log \Phi(x)/\partial \mu = -N(x)/\Phi(x)$.

$$\begin{aligned} \frac{\partial B_1}{\partial \sigma^2} = h_1(-1)B_1 + \lambda_1 e^{-A_1}\Big(&\frac{1}{2}C^2 + h_1(-1 - C\sigma^2) - h_1(-1) \\ &+ C g_1(-1 - C\sigma^2)\Big) \end{aligned} \tag{C.41}$$

where $h_1(x) = \partial \log \Phi(x)/\partial \sigma^2 = g_1(x)(x - \mu)/2\sigma^2$.

$$\frac{\partial B_3}{\partial \mu} = g_3(1)B_3 + \lambda_3 e^{-A_3}\left(-C + g_3(1 + C\sigma^2) - g_3(1)\right) \tag{C.42}$$

where $g_3(x) = \partial \log(1 - \Phi(x))/\partial \mu = N(x)/(1 - \Phi(x))$.

$$\begin{aligned} \frac{\partial B_3}{\partial \sigma^2} = h_3(1)B_3 + \lambda_3 e^{-A_1}\Big(&\frac{1}{2}C^2 + h_3(1 + C\sigma^2) - h_3(1) \\ &- C g_3(1 + C\sigma^2)\Big) \end{aligned} \tag{C.43}$$

where $h_3(x) = \partial \log(1 - \Phi(x))/\partial\sigma^2 = g_3(x)(x-\mu)/2\sigma^2$.

$$\frac{\partial B_2}{\partial \mu} = g_2(1)B_2 + 4C^2\lambda_2 A_{2,1}\left(2\mu + g_2(1)\left(\sigma^2 + 1 - \mu^2 - \frac{A_{2,2}}{A_{2,1}}\right)\right) \quad \text{(C.44)}$$

where $g_2(x) = \partial \log(\Phi(x) - \Phi(-x))/\partial\mu = (N(-x) - N(x))/(\Phi(x) - \Phi(-x))$.

$$\begin{aligned}\frac{\partial B_2}{\partial \sigma^2} = h_2(1)B_2 + 4C^2\lambda_2 A_{2,1}\Bigg(1 + \frac{A_{2,2}}{2\sigma^2 A_{2,1}} \\ + h_2(1)\left(1 - \mu^2 - \frac{A_{2,2}}{A_{2,1}}\right)\Bigg)\end{aligned} \quad \text{(C.45)}$$

where $h_2(x) = \partial \log(\Phi(x) - \Phi(-x))/\partial\sigma^2 = ((-x-\mu)N(-x) - (x-\mu)N(x))/(2\sigma^2(\Phi(x) - \Phi(-x)))$.

The value and derivatives of B_1 cannot be computed numerically stable if $(-1-\mu)/\sigma \gg 0$, since in this case both $\Phi(-1)$ and $\exp(-A_1)$ are extremely small. Noting that $\log Z(y) > 0$ for $y < -1$ and using (C.25) with $\lambda_1 = 1$, we have the trivial bounds $0 \leq B_1 \leq \exp(-A_1)$. Therefore, if $\exp(-A_1) < \varepsilon$ for a small constant ε, we can safely ignore B_1 and its derivatives. The same problem and solution applies to B_3 and $\exp(-A_3)$. The formulas for B_2 and its gradients are unstable if *both* $(\pm 1 - \mu)/\sigma$ are $\ll 0$, or both are $\gg 0$. In these cases, $A_{2,1}$ and $A_{2,2}$ are both very close to zero which causes the problem. Consider the case $(\pm 1 - \mu)/\sigma \ll 0$. Noting that $\log Z(y) \geq \log 2 - C$ for $y \in [-1, 1]$ and plugging $\xi_2 = 2C$ into (C.29), we see that

$$\begin{aligned}(\log 2 - C)A_{2,1} \leq B_2 \leq \left(\log 2 \cosh C - C + \frac{1}{2}C \tanh C \left(\mu^2 + \sigma^2 - 1\right)\right) A_{2,1} \\ + \frac{1}{2}C \tanh C A_{2,2},\end{aligned} \quad \text{(C.46)}$$

therefore B_2 is itself very close to zero. If the absolute values of these upper and lower bounds on B_2 are both smaller than ε, we can ignore B_2 and its derivatives. The case $(\pm 1 - \mu)/\sigma \gg 0$ is dealt with in a similar fashion.

Appendix D

The STATSIM system

In this chapter, we briefly outline the purpose, structure and features of the STATSIM system within which all the programs for the experiments of this thesis have been implemented. The system is not completely implemented at present, and large parts of the code are not tested yet, but it will hopefully grow to realize its design at some distant time ...

D.1 Purpose and goals

The principal goal of the STATSIM design is to offer an alternative to creeping slow Matlab code on the one hand, and dirty C implementations, user-controlled by command line arguments and cryptic parameter files[1] on the other hand. Matlab is great for prototyping, but is awfully slow in execution of iterative code. Even more important, Matlab's memory management is poor, and it simply cannot be used for problems that demand by their very nature a large amount of direct memory space. The algorithms developed in this thesis are of this kind, and the use of Matlab was out of question from the beginning. Matlab also lacks the structure of an object-oriented language, and it is very difficult to implement large systems in a clear way.

On the other hand, an interactive user interface is nice to play with ideas, to create complicated structures and to run a variety of algorithms on them, without the need of tediously complicated control files or steady recompilation of the system. Above all, this applies to object-oriented interfaces, and

[1] We must admit that the present version of the system is controlled by a cryptic parameter file.

therefore we designed such an interface for STATSIM, realized as extension of the well-known Tcl shell.

Another goal of STATSIM is that one should be able to incorporate foreign existing code easily into the system. A stub compiler is provided to automatically generate code to bind new classes into the user interface of the system. The clear object-oriented design of STATSIM makes it very easy to link in new code without risking interference with other parts of the system.

STATSIM was designed to be highly extensible. As an example, the optimization algorithms encoded within the system act on abstract criterion function classes and can therefore be used to optimize anything from simple benchmark criteria to neural networks or criteria which itself incorporate nested optimization problems[2]. On the other hand, we paid special attention from the beginning to retain high efficiency, especially in memory management. Almost all interfaces to algorithms allow the user optionally to pass buffers for the required memory. The user itself can decide what level of control he wants to keep over memory management. The technique of mask objects allows to impose a "virtual" object structure on parts of a simple buffer, thus reducing the need for copying objects to a minimum. For example, the complete set of weights of a neural network can be stored as a vector, allowing a generic optimizer to operate on, but at the same time the weights between two layers can be accessed as matrices, and methods from the matrix library can be applied as if the buffer chunk would be a real matrix object.

D.1.1 Programs we built on

It would be a waste of time to develop any large system from scratch. STATSIM is developed in C++ and requires, at least under Sun Solaris systems, a later version of the EGCS compiler to be built[3]. Many routines from the *Numerical Recipes* distribution [PTVF92] are incorporated. The excellent reference [Dev86] was extensively used to implement the random number generation module. The final user interface will be implemented as extension of the Tcl/Tk shell. The interface code between C++ and Tcl/Tk is generated using the SWIG interface generator. Both Tcl/Tk and SWIG are freely available in the public domain.

[2]Indeed, the algorithm presented in this thesis, together with Skilling approximations (as suggested above) requires the use of four levels of different nested optimizations.

[3]EGCS is the current label of the GNU C/C++/Fortran compiler family.

D.2 System structure

The coarse structure is a module hierarchy which is flat at present. Each module contains a number of classes which mostly interact with other classes in the same module. The basic modules are *interface* (code managing the user interface), *matrix* (matrix library), *rando* (random number generators for common distributions), *optimize* (selection of optimizers), *data* (management of large datasets), *mlp* (multi layer perceptrons), *mcmc* (Markov Chain Monte Carlo code), *gp* (Gaussian process code), *svm* (Support Vector code, including the algorithms proposed in this thesis).

D.2.1 Optimization: An example

To give an idea of how STATSIM combines generic code with efficiency we will have a look at the *optimize* module. All optimizers operate on criterion functions of the abstract base class *CritFunc*. Any CritFunc implementation maintains a reference to the actual evaluation point of the criterion and must provide methods to evaluate the criterion and its gradient there. Note that for many real-world criterion functions, the evaluation of the gradient can be done very efficiently by reusing quantities that were computed during the evaluation of the criterion itself, and/or vice versa. Examples are multi layer perceptrons or the F criterion of the variational algorithm proposed in this thesis. Therefore, CritFunc also provides methods for criterion and gradient evaluation given that the last evaluation of one of them took place at the same evaluation point. The generic optimizers use this facility whenever possible.

The usual optimizers return the optimal point, the criterion value there and some statistics. Sometimes, results of more complicated structure need to be returned and/or accumulated during optimization. The class *ResultSet* provides a generic interface for such cases. An implementation always stores a reference to the corresponding CritFunc object. Every time an element of the result set should be acquired or accumulated, the optimizer notifies the ResultSet object. The latter one communicates with the CritFunc which provides services to pass the required information to the ResultSet. Finally, the user can apply methods of the ResultSet to read the result out. Note that the ResultSet object is completely decoupled from the optimization process itself. A nice example of this interplay is the realization of a Markov Chain Monte Carlo method using the dynamics of a energy criterion. The CritFunc object maintains the details of the criterion and the current state of the chain. The ResultSet object accumulates statistics based on samples or simply stores them. It might also analyze the samples to determine the length

of the burnin phase (in which chain states are discarded) or the right time to stop. The algorithm itself can be coded independently of all these concerns and therefore used without modification in a great number of configurations.

D.2.2 Sketch of the user interface

The user interface is object-oriented. The system knows a number of classes. Most of these classes can be directly instantiated by the user, an instance is called object. Objects may refer to other objects to create an object hierarchy. Class hierarchies are currently not supported. Each object consists of methods, attributes and subobjects. Attributes have simple, unstructured datatypes like string, int or float. A subobject is an object itself (*single*), a vector of objects (*array*) or a list of objects organized as a hashtable (*hashmap*). Arrays and hashmaps are potentially heterogenous. An object is either *toplevel* or the *son* of another object (i.e. subobject or part of a subobject). An object explicitely created by the user is called *bound* (it is bound to a Tcl variable), as opposed to *free* objects generated implicitely.

A watchdog system takes care that no object is destroyed if there still exist other objects referring to it. A bound object can only be destroyed by the user itself. If the binding between a Tcl variable and its associated objects gets lost, the object is maintained in a *lost list* and can be recovered. It is also possible to keep references to a bound or free object in Tcl variables and use them like pointers. However, this is not considered as a binding, and the association is not supervised by the watchdog.

Linking existing code as new class into the user interface requires the specification of a quite large amount of C++ code to interface the SWIG generator which in turn creates code for the communication between Tcl/Tk and the C++ program. However, most of this work can be automatized using a stub compiler which assembles the C++ code given a description of the class, its attributes and methods, the parameters and options of the methods, and so on. Even though the present prototype version of the stub compiler is quite rudimentary, it allows to link new classes into the system requiring only very small efforts from the programmer.

D.3 Status Quo

We have given a sketch of the design of STATSIM and how it hopefully will look like when completed. The present version includes a large matrix library

(heavily borrowing from Numerical Recipes), a small random number library, a set of optimizers like conjugate gradients, Quasi-Newton, Levenberg-Marquardt, an implementation of the Skilling techniques discussed in the appendix above, code for Gaussian process classification using evidence maximization or the hybrid Monte Carlo algorithm, an SMO optimizer (see [Pla98]) for Support Vector classification, a generic class for cross validation and the code to compute criterion and gradients for the algorithm discussed in this thesis.

The system is at present controlled by a command file of a very simple structure. The user interface described above is almost completely implemented, and a rudimentary version featuring some matrix and vector classes can be executed. A prototype of the stub compiler was used to create the interface code for these classes.

The system will be put into the public domain as soon as it will have been developed so far that it can be handled without frustration by other people than the author. If the reader is interested in parts of the code, he is invited to contact us.