LEARNING FROM DATA STREAMS WITH CONCEPT DRIFT
LEARNING FROM DATA STREAMS
WITH CONCEPT DRIFT

A thesis submitted for the degree of
Doctor of Philosophy

ROMAN GARNETT
Magdalen College
University of Oxford

Trinity Term 2010
SUMMARY

Increasing access to large, nonstationary datasets and corresponding demands to analyze these data has led to the development of new online algorithms for performing machine learning on data streams. An important feature of many real-world data streams is “concept drift,” whereby the characteristics of the data can change arbitrarily over time. The presence of concept drift in a data stream renders many classical data mining techniques unsuitable, and therefore new approaches must be developed in their place. In pursuit of this goal, we introduce a number of new algorithms and techniques.

First we discuss the dynamic logistic regressor (DLR), a sequential Bayesian approach for performing binary classification on nonstationary data streams. We proceed to show how the DLR method can be extended to cope with missing observations and missing and corrupted labels.

Next we introduce Gaussian processes, a Bayesian technique for performing inference about functions. Using this tool, we propose a new sequential algorithm for performing robust time-series prediction in the presence of changepoints and observation faults. We then extend this work to demonstrate how to perform effective active data selection in sensor networks that are prone to drastic changes in their behavior.

We proceed by presenting the Gaussian processes for global optimization (GPGo) algorithm, which provides a Bayesian decision-theoretical framework for global function optimization. The GPGo routine builds upon previously published work in several novel ways.

Finally, we describe how the GPGo algorithm can be used to optimize a drifting objective function defined on a power set. We apply this routine to the problem of selecting a small subset of sensors from a sensor network to maximize global predictive power over time.

Experimental evidence demonstrates that our proposed algorithms outperform previously developed solutions to these problems.
ACKNOWLEDGMENTS

The author would like to thank a number of people and organizations for their help throughout the pursuit of this degree.

Thank you to:

- Stephen Roberts, for providing me the ultimate in academic independence, being supportive without being dictatorial, shielding me from and promptly dealing with administrative nonsense, and always having a dataset lying around—these are the marks of a great supervisor;

- Michael Osborne, for not minding when I asked him to do matrix calculus for me, letting me store things in his attic and sleep on his futon, consuming more sides of beans than could possibly be healthy, and always paying his absurd debts in a timely fashion;

- Richard Mann (Richman?), for Gustavo,¹ “couscous is just pretentious rice: FACT,” the great nation of Sealand, warp 10, глонасс, megachlieb, bloomin’ onions, aggregated diamond nanorods, the truest epitaph I have ever encountered, hot mojitos, … tea rum?, and, of course, caipirinhas; is it 3.141592653 now?

- the remaining members of PARG: Rob and Maike for trying to inject some life into our woefully antisocial group and supporting The Great Rearrangement of 2009,² Nauman for not complaining about my constant nonsense and forgiving me for spreading lies about him, Steve Reece for bringing a fresh Kalmanesque viewpoint to conversations and inventing the career-ending paper writing machine, Mark for attending to the minutest of details for NIPS and pointing me towards the life-saving MIDAS dataset, and Ash, I hardly knew ye, but we should talk nerd at a pub again;

- and last but not least, my family, for not being usual, laughing at me every time this thesis came up, and for Robert the Bruce’s heart, may it someday reach the Holy Land.

The author would also like to thank the Bramblemet Support Group and Associated British Ports for providing access to the weather sensor-network data used in Chapters 6 and 7 and the British Atmospheric Data Centre and UK Meteorological Office for providing the MIDAS Land Surface Stations data used in Chapter 9.

Finally, the author would like to extend special thanks to the Ninety-fifth United States Congress for passing PL 95–105, amending the Foreign Gifts and Decorations Act of 1966 (5 USC § 7342(c)(1)(B))³.

¹N…n…n…n!! I hope you enjoy your Lickschnitz and the elusive second kind of lingonberry in Uppsala.
²Rob, I still have those incriminating photos of you.
³Yes, (B)!

STATEMENT OF ORIGINALITY AND DECLARATION OF PREVIOUS PUBLICATION

All original work in this thesis was undertaken by the author in collaboration with other researchers. In addition, all novel contributions presented here have previously appeared in peer-reviewed publications.

In particular, the author would like to address the nature of the collaboration between himself and Michael A. Osborne. All work related to Gaussian processes (Chapters 5–9) represents a joint effort with Mr. Osborne. The vast majority of the resulting material was generated from discussions at whiteboards, scribbles on napkins, and conversations (often debates) over email. Due to this close research relationship, it is difficult to assign sole credit to most of its output. However, some things can be attributed to one author alone, and when possible this is acknowledged below. The author is confident that none of the work described here could have been accomplished without the joint input of both researchers.

What follows is a statement outlining the original contributions of this thesis, with references to previous publications and acknowledgments of presented material that was the sole work of another collaborator.

CHAPTERS 2–3
This material is all review of well-known results and definitions.

CHAPTER 4
The work in this chapter has appeared in the following publication.


The DLR algorithm was first presented by Penny and Roberts (1999). Lowne provided the ability to handle missing labels. Garnett contributed the modifications related to label noise, including the online estimation of the $\rho$ parameter. The probabilistic presentation of the DLR algorithm is original work of Garnett. Except for Experiment 1, all experiments in Section 4.4 were designed and carried out by Garnett.

CHAPTER 5
This chapter reviews previously published materials. The presentation is mostly the author’s own. In a few instances, other works were followed closely; this is explicitly acknowledged in the text when it occurs.

CHAPTER 6
The work in this chapter has appeared in the following publications.

STATEMENT OF ORIGINALITY AND DECLARATION OF PREVIOUS PUBLICATION


Garnett supplied all changepoint covariance functions and related proofs, except the continuous conditionally independent covariance $K_{CCI}$, which was provided by Reece. Osborne completed work on observation faults and approximating hyperparameter posterior distributions. Roberts supplied guidance and several datasets. Garnett implemented all related software and gathered experimental results.

CHAPTER 7
The work in this chapter has appeared in the following publication.


The design of the loss function was suggested by Osborne. The ultimate details of the algorithm were jointly determined by Osborne and Garnett. Garnett gathered all experimental results. Initial problem motivation, general theoretical background, and some data were provided by Roberts.

CHAPTER 8
The work in this chapter has appeared in the following publication.


The foundation of the GPGO algorithm was joint work between Garnett and Osborne. Some extensions were joint work; others, especially noisy optimization, were provided by Osborne alone. Osborne supplied the bulk of the software implementation. Garnett completed all testing. Roberts contributed advice and served as a liaison with experts who offered assistance with choosing test functions.

CHAPTER 9
The work in this chapter has appeared in the following publication.


The germinative idea of placing an appropriately constructed Gaussian process model on a power set was due to Garnett. All remaining theoretical work and experimental design was jointly contributed by Garnett and Osborne. Garnett completed implementation and testing. Roberts provided advice and contextual information.
CONTENTS

SUMMARY v

ACKNOWLEDGMENTS vii

STATEMENT OF ORIGINALITY AND DECLARATION OF PREVIOUS PUBLICATION ix

1 INTRODUCTION 1

2 INTRODUCTION TO PROBABILITY THEORY 3
   2.1 What is Probability? 3
   2.2 Definitions and Basic Results of Probability Theory 4
   2.3 Bayesian Inference 10
   2.4 Justifications of Probability 12

3 PROBLEM DEFINITION AND PREVIOUS WORK 15
   3.1 The Online Supervised Learning Task 15
   3.2 Proposed Solutions for Online Classification with Concept Drift 17

4 THE DYNAMIC LOGISTIC REGRESSOR FOR ONLINE SEMI-SUPERVISED CLASSIFICATION 21
   4.1 Review of the Dynamic Logistic Regressor 21
   4.2 Notes Regarding the DLR 26
   4.3 Modifying the DLR to Handle Corrupted Labels 28
   4.4 Experimental Results 30

5 GAUSSIAN PROCESSES 39
   5.1 Definition and Prior Distribution 39
   5.2 The Posterior Distribution 41
   5.3 Discussion 42
   5.4 Example 46
   5.5 Hyperparameter Management 48

6 SEQUENTIAL BAYESIAN PREDICTION IN THE PRESENCE OF CHANGEPUNKTS AND FAULTS 63
   6.1 Changepoints in the Latent Function 64
   6.2 Changepoints in Observation Likelihood 72
   6.3 Expository Example 79
   6.4 Results 79

7 ACTIVE DATA SELECTION IN THE PRESENCE OF CHANGEPUNKTS AND FAULTS 93
   7.1 Active Data Selection for Robust Prediction 94
The amount of data transferred over communication networks is growing exponentially (Roberts, 2000a). With corresponding increases in data storage capacity and processing power, new methods are needed for extracting knowledge from fast-moving, quickly changing, and extremely large data sources. The growing field of data-stream mining addresses these problems.

Data streams pose several unique problems that inhibit the application of standard data-mining techniques. The dataset is continuously online and growing to include new measurements; therefore, effective algorithms for analyzing these data must be able to work within a constant memory footprint. In particular, the entire dataset cannot be stored in memory and historical data must eventually be forgotten. An additional problem is that characteristics of the data might change over time, a condition known as concept drift or dataset shift (Tsymbal, 2004; Quiñonero-Candela, et al., 2009). Any reasonable stream-learning algorithm must be able to recognize and cope with these situations.

Data-stream mining systems must also deal with missing and corrupted data—noisy communication lines, human error, experimental design, and failing sensors can all alter and interrupt data streams. In online learning systems, both observations and responses can be missing or corrupted at any time. Noisy and missing observations have been the subject of extensive research. Observation noise is often explicitly modeled by learning procedures, and various imputation techniques have been proposed for handling missing values (Schafer, 1997; Little and Rubin, 2002).

Several solutions have been proposed for performing online classification in the presence of concept drift. These solutions are generally based on the adaptive maintenance of some discriminating structure; those considered include sets of binary rules (Widmer and Kubat, 1996), decision trees (Hulten, et al., 2001), and ensembles of classifiers (Street and Kim, 2001; Kolter and Maloof, 2007, 2005). The streaming regression problem has been the subject of considerably less investigation, but proposed solutions do exist (Pankratz, 1991; Kolter and Maloof, 2005).

Considerably less research has focused on classification with missing and corrupted labels, especially in the online setting. The classification task with missing labels is called semi-supervised learning; see (Zhu, 2008) for an excellent survey. Nearly all proposed solutions for the semi-supervised learning problem assume stationarity and require multiple passes over the dataset; they are therefore not applicable to data streams that exhibit concept drift. Little research has investigated classification with corrupted labels, although some theoretical work has been published (Jackson, et al., 1999).

In this thesis, we will address these shortcomings by systematically applying techniques from Bayesian probability theory to these problems. We will begin with a short introduction to probability theory. After defining the problem and discussing previously proposed methods for learning from drifting data streams, we will introduce the dynamic logistic regressor (DLR) (Penny and Roberts, 1999; Lowne, et al., 2010), a method...
for performing semi-supervised binary classification on nonstationary data streams. We will show how to modify the DLR to cope with possibly corrupted labels. The result is a powerful online learning algorithm that can continue to operate even with noisy or missing observations and labels.

Next we will introduce Gaussian processes, a powerful technique for performing Bayesian inference about functions. Typical covariance functions used with Gaussian processes cannot perform effectively in the presence of sudden changepoints in the data stream. To address this issue, we will introduce nonstationary covariance functions to be used in Gaussian process prediction that model such changes, then proceed to demonstrate how to effectively manage the hyperparameters associated with these covariance functions. We will further introduce a flexible observation model to be used in situations where the nature of our observations undergoes changes, for example, when a sensor experiences a fault. By using Bayesian numerical integration, we can marginalize the hyperparameters associated with our model, allowing us to calculate the full marginal predictive distribution. Furthermore, if desired, the posterior distribution over putative changepoint and fault locations can be calculated as a natural byproduct of our prediction algorithm. We will then show how to use these methods to perform effective online active data selection in sensor networks that are prone to sudden drastic changes in their behavior.

We will proceed to discuss the Gaussian processes for global optimization (GP-GO) algorithm for Bayesian optimization of expensive-to-evaluate functions. We will frame optimization as a sequential decision problem and introduce solutions to this problem with varying degrees of sophistication. Our solutions can be tailored to exactly the degree of confidence required from them, allowing us to effectively optimize objective functions that are corrupted by observation noise. The use of Gaussian processes also permits us to benefit from the straightforward incorporation of prior knowledge about the objective function and from any available derivative observations. Using this ability to learn from derivative observations, we will introduce an innovative method to address potential numerical conditioning problems in the optimization process.

Finally, we will apply the GP-GO algorithm to an objective function that is subject to concept drift—the predictive quality of a sensor network through time. We will consider the problem of selecting a small set of sensors to maximize the predictive quality of the associated sensor network. This problem will require placing a Gaussian process prior distribution over a nontrivial discrete input space, specifically, the power set of available sensors. We will suggest a novel metric between sensor networks that can be used to construct effective covariance functions over this space and demonstrate the effectiveness of the resulting technique on a difficult weather prediction problem.

Finally, potential avenues for future research on these problems will be proposed and discussed.
Our approach to questions regarding data streams will be chiefly motivated by probabilistic reasoning. In this chapter, we will give a brief introduction to the basic definitions and results of probability theory. It is by no means complete; for a more in-depth treatment, see (Jaynes, 2003; Berger, 2006; Halpern, 2003). Before we begin the discussion, we will take a moment to consider what exactly we mean by "probability."

2.1 WHAT IS PROBABILITY?

The proper meaning of "probability" is the subject of some debate. The two chief interpretations have come to be known as frequency probability and Bayesian probability.

The frequency interpretation defines probability in terms of an idealized repeated experiment. The possible outcomes of an experiment are considered, and the probability of an event (which is simply a subset of the outcomes) is defined to be the relative frequency an outcome in that event would occur during a theoretical infinite series of experimental trials.

The chief inference procedure in frequency probability is known as a hypothesis test. Given the results of an experiment and the details of its design, a null hypothesis is chosen, which represents the conjecture that the experimental findings were not unusual. The purpose of the null hypothesis is to provide a convenient method for providing hypothetical alternative experimental results. A statistical test is constructed to calculate the probability that the results of a typical experiment would be "at least as unusual" as the dataset under consideration; when this probability is low, the null hypothesis is rejected with a confidence specified by the calculated probability.

The Bayesian interpretation of probability eschews the formalism of repeated experiments and instead regards a probability as measuring the "degree of belief" an actor has in a particular proposition. By convention, the value one represents absolute confidence in a proposition, zero represents absolute confidence in its negation, and the intervening interval represents a continuum of confidence between these extremes. Probability is established as a formal mechanism for working with these degrees of belief that obeys certain rationality requirements. Due to this broader meaning of "probability," the Bayesian interpretation can consider propositions that cannot be repeatedly tested by an experiment. For example, the proposition "a Democrat will win the next presidential election in the United States" is perfectly valid in Bayesian probability, but is ill posed and intractable in frequency probability.

A result central to Bayesian probability is Bayes’ theorem, which provides a method for updating one's belief in a proposition after new evidence is discovered. This will be particularly useful when dealing with concept-drifting data streams, because each newly observed datum can potentially provide valuable information about the proposition at hand.

Throughout this text, we will adopt the Bayesian interpretation of probability. We will provide the theoretical basis of this interpretation below.
2. INTRODUCTION TO PROBABILITY THEORY

2.2 DEFINITIONS AND BASIC RESULTS OF PROBABILITY THEORY

We begin by defining a probability space. It is useful to first define a measure space.

**Definition 2.2.1 (Measure, Probability Spaces)** A measure space is a tuple \((\Omega, \Sigma, \mu)\), where \(\Omega\) is a set, \(\Sigma \subseteq \mathcal{P}(\Omega)\) is a \(\sigma\)-algebra\(^1\) of subsets of \(\Omega\), and \(\mu : \Sigma \to [0, \infty]\) is a positive, extended real-valued function on \(\Sigma\), called a measure, that satisfies the following conditions.

- The measure of the empty set is zero:
  \[ \mu(\emptyset) = 0. \]

- The measure is countably additive: for a countable collection of pairwise-disjoint sets \(\{G_i\}_{i=1}^\infty \subseteq \Sigma\), we have
  \[ \mu \left( \bigcup_{i=1}^\infty G_i \right) = \sum_{i=1}^\infty \mu(G_i). \]

If, in addition, \(\mu(\Omega) = 1\), then the measure \(\mu\) is called a probability measure, and the measure space \((\Omega, \Sigma, \mu)\) is called a probability space.

When \((\Omega, \Sigma, \mu)\) is a probability space, the set \(\Omega\) is called the sample space or the set of possible worlds; each member of \(\Omega\) represents the potential outcome of some process. Typically, \(\Omega\) will be a discrete or countable set, a real vector space \(\mathbb{R}^d\), or a mixture (product) of the two. The members of \(\Sigma\) are called events and represent the sets of possible outcomes that we can assign a probability to. When \(\Omega\) is countable, it is usual (but not required) to set \(\Sigma\) equal to the power set of \(\Omega\), \(\mathcal{P}(\Omega)\). When \(\Omega = \mathbb{R}^d\), the power set is usually too large to be of use, and a restricted set (typically the Borel-measurable or Lebesgue-measurable sets) is assigned to \(\Sigma\) instead.

We will often denote probability measures with \(\text{Pr}\).

From Definition 2.2.1, we may immediately deduce several useful “sum rules.”

**Theorem 2.2.1 (Sum Rules of Probability)** If \((\Omega, \Sigma, \text{Pr})\) is a probability space and \(A, B \in \Sigma\) are two events, then the following statements are true.

- (the sum rule)
  \[ \text{Pr}(A) + \text{Pr}(\Omega \setminus A) = 1. \]

- (the extended sum rule)
  \[ \text{Pr}(A \cup B) = \text{Pr}(A) + \text{Pr}(B) - \text{Pr}(A \cap B). \]

---

\(^1\)For a set \(X\), we write \(\mathcal{P}(X)\) to represent the power set of \(X\)—the set of all subsets of \(X\).

\(^2\)A \(\sigma\)-algebra over a space \(\mathcal{X}\) is a nonempty set of subsets of \(\mathcal{X}\) that is closed under complementation and countable unions.
2.2. Definitions and Basic Results of Probability Theory

Probability density functions
Given a probability space \((\Omega, \Sigma, \Pr)\), we may write the probability of an event \(A \in \Sigma\) as a trivial integral with respect to the probability measure \(\Pr\):

\[
\Pr(A) = \int_A \, d\Pr.
\]

The measure \(\Pr\) will almost certainly not provide a simple method for direct integration with respect to it. For this reason, it is often useful to be able to represent probabilities in terms of integration with respect to a simpler, more easily managed measure.

If \(\Omega\) is a countable set and \(\Sigma = \mathcal{P}(\Omega)\), then the probability measure is completely determined by its value on singletons. For convenience, we will define a function \(p: \Omega \to [0, 1]\) trivially by

\[
p(\omega) = \Pr(\{\omega\}),
\]

where the \(\doteq\) symbol indicates an equality that is the result of a definition. For \(A \in \Sigma\), we may now write

\[
\Pr(A) = \sum_{a \in A} p(a).
\]

The function \(p\) is called the probability mass function of the probability space \((\Omega, \Sigma, \Pr)\).

If \(\Omega\) is the real numbers \(\mathbb{R}\), we may derive a similarly useful function. We begin by defining a function \(f: \mathbb{R} \to [0, 1]\) by

\[
f(x) = \Pr((\leq x)).
\]

The function \(f\) is called the cumulative distribution function (CDF) of the probability space. When \(f\) is an absolutely continuous function, \(f\) will be differentiable almost everywhere, and we can define a function \(p: \mathbb{R} \to \mathbb{R}\) by differentiation:

\[
p(x) = \frac{df(x)}{dx}.
\]

The function \(p\) is called the probability density function of the probability space and has the following useful property: given an event \(A \in \Sigma\), the probability of \(A\) can be calculated with

\[
\Pr(A) = \int_A p(x) \, dx,
\]

where \(dx\) represents integration with respect to the Lebesgue measure. It should be noted that the probability density function evaluated at \(x\) does not give the probability of the event \(\{x\}\) occurring; rather, we may loosely say that for an infinitesimal \(dx\), the interval \([x, x + dx]\) has probability \(p(x) \, dx\). Nonetheless, we may often manipulate probability density functions as if they were actual probabilities.

There is an intimate relationship between the cumulative distribution functions on \(\mathbb{R}\) and the probability measures on the Lebesgue-measurable sets in \(\mathbb{R}\); specifically, there is a one-to-one correspondence between them. The probability measure corresponding to a given CDF is said to be induced by that CDF.

If \(\Omega\) is a real vector space \(\mathbb{R}^d\), a probability density function can be found using a similar process as above, assuming the probability measure is absolutely continuous with respect to the Lebesgue measure.

---

\(\dagger\)If \(\mu\) and \(\nu\) are two measures on a space \((\Omega, \Sigma)\), we say \(\mu\) is absolutely continuous with respect to \(\nu\) if \(\mu(A) = 0\) for all sets \(A \in \Sigma\) with \(\nu(A) = 0\).
Finally, if $\Omega$ is a more complicated set, all is not lost. If the probability measure $\Pr$ is absolutely continuous with respect to a more useful $\sigma$-finite measure $\mu$ on the space, then the Radon–Nikodym theorem (Rudin, 1987) guarantees the existence of a non-negative function $p: \Omega \to [0, \infty)$ such that the probability of an event $A \in \Sigma$ is given by

$$\Pr(A) = \int_A p \, d\mu.$$ 

This $p$, when it exists, can also be called a probability density function. Of course, the probability mass and probability density functions defined above correspond to special cases of this theorem, where the more useful measures are the counting measure for countable sets and the Lebesgue measure for $\mathbb{R}^d$. In fact, the requirement that the cumulative distribution function of a measure on $\mathbb{R}^d$ be absolutely continuous is equivalent to requiring that the probability measure it induces be absolutely continuous with respect to the Lebesgue measure.

Probability measures on $\mathbb{R}^d$ that do not admit a probability density function are somewhat pathological, and for the remainder of the current work we will assume that probability density functions will always exist, with one special exception. If $\Sigma \subseteq \mathcal{P}(\mathbb{R})$ is a $\sigma$-algebra of subsets of the real line, we define the Dirac measure $\mu_\delta: \Sigma \to [0,1]$ on $\mathbb{R}$ by

$$\mu_\delta(A) = \begin{cases} 0 & 0 \notin A; \\ 1 & 0 \in A. \end{cases}$$

The measure $\mu_\delta$ is a trivially valid probability measure on $\mathbb{R}$ for any chosen $\sigma$-algebra $\Sigma$; however, it is not absolutely continuous with respect to the Lebesgue measure. Its cumulative distribution function is the Heaviside step function, which has a discontinuity at 0. Therefore, the Dirac measure does not have a corresponding probability density function. Nonetheless, we will allow an abuse of notation and assume the existence of a “function” $\delta: \mathbb{R} \to \mathbb{R}$ with the special property

$$\int_{-\infty}^{\infty} f(x) \delta(x) \, dx = f(0),$$

for any measurable function $f$. We call $\delta$ the Dirac delta function. This construction can be trivially extended to higher-dimensional Euclidean spaces.

**Random variables**

A fundamental concept in probability theory is that of a random variable.

**Definition 2.2.2 (Random variable)** Suppose $(\Omega, \Sigma, \Pr)$ is a probability space. A random variable is a measurable function $X: \Omega \to S$ from $\Omega$ into some set $S$. The set $S$ is called the state space.

We will often notate random variables with capital letters (e.g., $X$) and write arguments of functions related to a given random variable with the corresponding lowercase letter (e.g., $x$).

---

4 A measure $\mu$ on a space $\Omega$ is said to be $\sigma$-finite if $\Omega$ can be written as a countable union of sets with finite measure under $\mu$. All typical measures are $\sigma$-finite.

5 For a probability density function to not exist, at least one “small” set (that is, a set with Lebesgue measure zero) must have nonzero probability.
2.2. Definitions and Basic Results of Probability Theory

An important class of random variables is the class of real-valued random variables, corresponding to $S = \mathbb{R}$. Given a probability space $(\Omega, \Sigma, \Pr)$, a random variable $X: \Omega \to \mathbb{R}$ induces a probability measure on the real line in a straightforward way.

It will be useful to introduce a notational device here. If $x \in \mathbb{R}$ is a real number, we will write $X \leq x$ as a shorthand for the set

$$\{ \omega \in \Omega \mid X(\omega) \leq x \}.$$

Now if we define a function $f_X: \mathbb{R} \to \mathbb{R}$ by

$$f_X(x) \triangleq \Pr(X \leq x),$$

we will have created a valid cumulative distribution function, which corresponds to a probability measure on $\mathbb{R}$. Assuming this CDF is absolutely continuous, there will be a probability density function associated with this probability measure, which we will denote with $p_X(x)$. We will refer to $p_X$ simply as the distribution of $X$. Due to our notational convention of matching argument names with their associated random variables, we will often drop the subscript and simply write, for example, $p(x)$ for the distribution of the random variable $X$.

In a similar fashion, any two real-valued random variables, $X$ and $Y$, over a common probability space $(\Omega, \Sigma, \Pr)$ induce a probability measure on $\mathbb{R}^2$. This may be accomplished by defining the function $f_{X,Y}: \mathbb{R}^2 \to \mathbb{R}$ by

$$f_{X,Y}(x, y) \triangleq \Pr( X \leq x \cap Y \leq y ),$$

creating a two-dimensional analogue of the cumulative distribution function. When this function is absolutely continuous, the function

$$p_{X,Y}(x, y) \triangleq \frac{\partial^2 f_{X,Y}(x, y)}{\partial x \partial y}$$

will be a valid probability density function on $\mathbb{R}^2$. We will call $p_{X,Y}(x, y)$ the joint distribution of $X$ and $Y$. This procedure generalizes trivially for more than two random variables.

We will often encounter certain parametric families of distributions for the random variables we will consider, either because these distributions naturally arise from considering the situation at hand or because they have convenient mathematical properties. When specifying a parametric distribution $q$ for the random variable $X$, we will write $p(x) = q(x; a_1, \ldots)$ or simply $p(x) = q(a_1, \ldots)$, where the arguments $\{a_1, \ldots\}$ specify the parameters of the distribution family $q$.

Perhaps the most important family of parametric distributions that we will consider in this text is the multivariate normal or Gaussian distribution $\mathcal{N}(\mu, \Sigma)$. Appendix A briefly introduces this distribution and its properties.

**Expectation**

Given a real-valued random variable $X$ on a discrete space $\Omega$, we may consider the sum of the possible values of $X$ weighted by their probability of occurring,

$$\sum_{\omega \in \Omega} X(\omega) p(\omega).$$
This expression represents the value that can be “expected” to occur, on average, given a large number of observations of $X$. Note that this value itself may not be likely or even attainable by the random variable.

This notion of “expected value” generalizes trivially for any real-valued random variable.

**Definition 2.2.3 (Expected Value)** Suppose that $(\Omega, \Sigma, \Pr)$ is a probability space and $X: \Omega \to \mathbb{R}$ is a real-valued random variable on $\Omega$. The expected value of $X$, written $\mathbb{E}[X]$, is defined by

$$\mathbb{E}[X] = \int_{\Omega} X \, d\Pr.$$ 

If $X$ has a corresponding probability density function $p$, the expected value may be calculated with

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x p(x) \, dx.$$ 

If $f: \mathbb{R} \to \mathbb{R}$ is measurable, the expected value of $f(X)$ is given by

$$\mathbb{E}[f(X)] = \int_{\Omega} f(X) \, d\Pr = \int_{-\infty}^{\infty} f(x) p(x) \, dx.$$ 

The latter expression is only valid when the probability density function $p$ exists.

The definition of expected value as an integral allows us to conclude some basic linearity properties.

**Theorem 2.2.2** If $X$ and $Y$ are random variables on a common probability space and $a \in \mathbb{R}$ is a constant, then the expected value function has the following properties.

- **(Homogeneity)**
  $$\mathbb{E}[aX] = a\mathbb{E}[X].$$

- **(Additivity)**
  $$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y].$$

**Conditional probability and Bayes’ theorem**

It will often be useful to consider the probability of an event after discovering that the true state of the world lies in a restricted subset of the sample space. This type of probability is called conditional probability.

**Definition 2.2.4 (Conditional Probability)** Let $(\Omega, \Sigma, \Pr)$ be a probability space and let $A, B \in \Sigma$ be two events with $\Pr(B) > 0$. The conditional probability of $A$ given that $B$ is true, written $\Pr(A \mid B)$, is defined by

$$\Pr(A \mid B) = \frac{\Pr(A \cap B)}{\Pr(B)}. \tag{2.2.1}$$

If $\Pr(B) = 0$, then $\Pr(A \mid B)$ is undefined.

Conveniently, probabilities conditional on a given event provide valid probability measures.
If \((\Omega, \Sigma, \Pr)\) is a probability space and \(B \in \Sigma\) is an event with \(\Pr(B) > 0\), then the function \(\Pr(\cdot \mid B): \Sigma \to [0, 1]\) is a probability measure on \(\Omega\).

There is a construction analogous to conditional probability for real-valued random variables over a common probability space.

**Definition 2.2.5 (Conditional Probability Density Function)** Suppose \((\Omega, \Sigma, \Pr)\) is a probability space and \(X, Y: \Omega \to \mathbb{R}\) are two real-valued random variables with probability density functions \(p_X(x)\) and \(p_Y(y)\), respectively, and with joint distribution \(p_{X,Y}(x, y)\). The function

\[
p_{X|Y}(x \mid y) = \begin{cases} 
p_{X,Y}(x, y) / p_Y(y) & \text{if } p_Y(y) > 0; \\
0 & \text{otherwise,}
\end{cases}
\]

is called the conditional probability distribution of \(X\) given \(Y\).

Again, the conditional probability distribution provides a valid probability density function on \(\mathbb{R}\).

**Theorem 2.2.4** Suppose \(X\) and \(Y\) are real-valued random variables on a probability space \((\Omega, \Sigma, \Pr)\) and \(p_{X|Y}\) is the conditional probability distribution of \(X\) given \(Y\). Let \(y \in \mathbb{R}\) be fixed with \(p_Y(y) > 0\). The function \(p_{X|Y}(\cdot \mid y): \mathbb{R} \to \mathbb{R}\) is a valid probability density function.

We now derive a useful result from the definition of conditional probability distributions.

**Theorem 2.2.5 (Marginalization)** If \((\Omega, \Sigma, \Pr)\) is a probability space and \(X\) and \(Y\) are two real-valued random variables on \(\Omega\) with associated probability density functions \(p(x)\) and \(p(y)\), respectively, then

\[
p(x) = \int_{-\infty}^{\infty} p(x \mid y) p(y) \, dy.
\]

The process of deriving \(p(x)\) by integrating out the “nuisance parameter” \(y\) in \(p(x \mid y)\) is called marginalization, and the distribution \(p(x)\) in this context is called the marginal distribution of \(X\).

Written in a slightly different form, (2.2.1) is commonly referred to as the product rule of probabilities.

**Theorem 2.2.6 (Product Rule of Probabilities)** If \((\Omega, \Sigma, \Pr)\) is a probability space and \(A, B \in \Sigma\) are two events, then

\[
\Pr(A \mid B) \Pr(B) = \Pr(A \cap B) = \Pr(B \mid A) \Pr(A).
\]

Rearranging the product rule of probabilities produces one of the most important results in Bayesian probability, known as Bayes’ theorem.
2. Introduction to Probability Theory

Theorem 2.2.7 (Bayes’) If \((\Omega, \Sigma, \Pr)\) is a probability space and \(A, B \in \Sigma\) are two events with \(\Pr(B) > 0\), then

\[
\Pr(A \mid B) = \frac{\Pr(B \mid A) \Pr(A)}{\Pr(B)}.
\]

If \(X\) and \(Y\) are two real-valued random variables on \(\Omega\) with \(p(y) > 0\), then

\[
p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}.
\]

The significance of this result can be realized by considering a typical application. Suppose we are interested in the probability of some event \(A\) and learn during the course of our investigation that the true state of the world lies in some other event \(B\). Before we discovered this new evidence, we considered the probability of \(A\) to be \(\Pr(A)\). In this context, \(\Pr(A)\) is called the prior probability of \(A\), as it represents our outlook before learning the information contained in \(B\). Similarly, the probability \(\Pr(A \mid B)\) is called the posterior probability of \(A\) given \(B\). Bayes’ theorem provides a mechanism for updating our beliefs about a proposition after discovering new evidence.

In the context of processing streaming data, we will often be interested in the probability of a certain proposition \(A\) given the data observed up to a specified time. Suppose that during the course of observation, we had observed data \(D \triangleq \{x_1, x_2, \ldots, x_N\}\). The prior probability \(\Pr(A \mid D)\) would represent our belief about \(A\) after observing \(D\). If we then observed a new datum \(x_{N+1}\), we could use Bayes’ theorem to derive the posterior probability

\[
\Pr(A \mid D \cup \{x_{N+1}\}).
\]

We would then use this posterior probability to reason about \(A\). If we observed a new datum \(x_{N+2}\), the posterior probability just derived would become the prior probability in the next application of Bayes’ theorem. In this manner, we could progress through the data stream, at each step substituting the most-recently calculated posterior probability for the current prior probability. The process of continually updating our probabilities in light of new observations in this manner is called recursive Bayesian estimation.

2.3 Bayesian Inference

With the machinery of probability theory in place, we briefly discuss the typical procedures of Bayesian inference. As an expository example, we will consider a Bayesian formulation of simple linear regression.

In the Bayesian framework, we typically have observed data \(D\) that we wish to reason about. This inevitably entails selecting a model for the data, that is, an (often parametrized) family of probability distributions that might reasonably explain the observed data.

In the case of linear regression, our data are of the form

\[
D \triangleq \{(x_i, y_i)\}_{i=1}^N,
\]

where the \(x_i \in \mathbb{R}^d\) are real \(d\)-dimensional vectors of predictor or independent variables and the \(y_i \in \mathbb{R}\) are real response or dependent variables. Let \(X\) represent the \(d \times N\) matrix generated by collecting the \(\{x_i\}\), and let \(y\) represent the \(d \times 1\) vector of response variables. \(^*\)

*\(X\) is the transpose of what is typically called the design matrix.
generated by collecting the \( \{ y_i \} \). We select a model that represents an underlying linear relationship between our variables that is corrupted by observation noise:

\[
y_i \doteq x_i^T w + \epsilon_i,
\]

where \( w \in \mathbb{R}^d \) is a vector of weights and \( \epsilon_i \) is an additive noise term allowing the linear fit to be inexact. We assume the noise is independent and identically distributed, and choose for it a zero-mean Gaussian distribution:

\[
p(\epsilon_i \mid \sigma) \doteq \mathcal{N}(0, \sigma^2),
\]

where \( \sigma^2 \) is the (homoskedastic) noise variance. Of course, we do not know the value of the model parameters \( w \) beforehand, so our aim is to calculate the posterior belief \( p(w \mid X, y, \sigma) \). We begin by writing down the likelihood implied by our model:

\[
p(y \mid X, w, \sigma) = \prod_{i=1}^N p(y_i \mid x_i, w, \sigma) = \prod_{i=1}^N \mathcal{N}(x_i^T w, \sigma^2) = \mathcal{N}(X^T w, \sigma^2 I),
\]

where \( I \) represents the identity matrix. To proceed, we must choose a prior for the weight vector \( w \); a reasonable choice might be a zero-mean normal distribution with covariance \( \Sigma \):

\[
p(w \mid \Sigma) \doteq \mathcal{N}(w; 0, \Sigma).
\]

We may now write down our posterior for \( w \) directly using Bayes’ theorem:

\[
p(w \mid X, y, \Sigma, \sigma) = \frac{p(y \mid X, w, \sigma) p(w \mid \Sigma)}{p(y \mid X, \Sigma, \sigma)}.
\]

An expression for the denominator may be obtained by marginalization:

\[
p(y \mid X, \Sigma, \sigma) = \int p(y \mid X, w, \sigma) p(w \mid \Sigma) dw.
\]

The posterior can now be calculated using simple Gaussian identities (Rasmussen and Williams, 2006):

\[
p(w \mid X, y, \Sigma, \sigma) = \mathcal{N}(w; \sigma^{-2} S^{-1} X y, S^{-1}),
\]

where

\[
S = \sigma^{-2} XX^T + \Sigma^{-1}.
\]

The posterior is Gaussian distributed because both the likelihood and the prior were Gaussian. The denominator serves as a normalizing factor to ensure that the resulting probability density function has unit integral.

Note that in the above discussion, we assumed foreknowledge of various components of our model; for example, the noise variance \( \sigma^2 \) and the prior weight covariance \( \Sigma \) were presumed to be given. It is rare that we can actually be certain about the value of these
quantities \emph{a priori}. If that is the case, a rigorous Bayesian approach would specify priors \(p(\sigma)\) and \(p(\Sigma)\) and calculate the full posterior

\[
p(w \mid X, y) = \frac{\iiint p(y \mid X, w, \sigma) p(w \mid \Sigma) p(\sigma) \, d\sigma \, d\Sigma}{\iint p(y \mid X, w, \sigma) p(w \mid \Sigma) p(\sigma) \, dw \, d\sigma}
\]

Unfortunately, depending on the form of the chosen priors, this posterior might not have a closed-form solution.

As a practical example, we might want to use a particular shape \(\Sigma'\) for the prior covariance \(\Sigma\), multiplied by an unknown scaling factor \(\lambda^2\):

\[p(\Sigma \mid \Sigma', \lambda) = \delta(\Sigma - \lambda^2 \Sigma').\]

As we are again uncertain \emph{a priori} about the value of \(\lambda\), we would again have to choose a prior \(p(\lambda)\) and integrate out the uncertainty in \(\lambda\) when deriving the posterior for \(w\). Parameters of prior distributions (like \(\lambda\)) are called \emph{hyperparameters}, and their priors (like \(p(\lambda)\)) are called \emph{hyperpriors}.

As our model becomes increasingly complex, our chances of rendering a closed-form solution for the posterior of \(w\) become increasingly unlikely. For this reason, methods for successfully approximating nonanalytic posterior distributions are an important focus of Bayesian research.

### 2.4 Justifications of Probability

The measure-theoretic formulation of probability theory presented in the previous section is due to Kolmogorov (1933). Although the formal system presented for representing uncertainty in propositions is certainly consistent, it is unclear why we should choose it over any other. Numerous arguments have been proposed to justify probability as a natural—and in some sense epistemically correct—method for representing uncertainty. Most of these arguments present a series of “common-sense” rationality axioms that any seemingly reasonable belief function should obey and conclude that any plausibility function satisfying these axioms must be isomorphic to a probability measure.

One commonly presented argument is known as Cox’s theorem (Cox, 1946); here we will briefly follow the presentation presented by Halpern (1999). See (Jaynes, 2003) for a very in-depth and detailed presentation of this material.

Suppose we have a sample space \(\Omega\) and some domain \(W \subseteq \mathcal{P}(\Omega)\) on which we wish to measure plausibilities. Cox’s theorem supposes the existence of an arbitrary real-valued “belief” function \(\text{Bel} : W^2 \to \mathbb{R}\) that for an ordered pair \((A, B) \in W^2\) assigns a number commensurate with the plausibility of \(B\) given \(A\). For this reason, we write the arguments of \(\text{Bel}\) as \(\text{Bel}(B \mid A)\). Cox makes the following assumptions.

\((A1)\) \hspace{1em} (Halpern, 1999) The plausibility of a proposition is completely determined by the plausibility of its negation. We therefore assume the existence of a function \(S : \mathbb{R} \to \mathbb{R}\) such that for \(A, B \in W\) with \(B \neq \varnothing\),

\[
\text{Bel}(\Omega \setminus B \mid A) = S(\text{Bel}(B \mid A)).
\]
The plausibility of a proposition \((A \cap B) \mid C\) is completely determined by the plausibility of \(A \mid (B \cap C)\) and the plausibility of \((B \mid C)\). We therefore assume the existence of a function \(F: \mathbb{R}^2 \to \mathbb{R}\) such that for \(A, B, C \in W\) with \(B \cap C \neq \emptyset\),

\[
\text{Bel}(A \cap B \mid C) = F \left( \text{Bel}(A \mid B \cap C), \text{Bel}(B \mid C) \right). 
\]

Under a few additional reasonable assumptions, we may conclude the main result.

**Theorem 2.4.1 (Cox (Halpern, 1999))** Suppose \(\text{Bel}\) is a real-valued belief function that satisfies (A1) and (A2) and that

- the range of \(\text{Bel}\) is \([0, 1]\);
- \(\text{Bel}(\emptyset \mid A) = 0\), and \(\text{Bel}(A \mid A) = 1\) for \(A \neq \emptyset\);
- the function \(S\) in (2.4.1) is decreasing;
- the function \(F\) in (2.4.2) is strictly increasing in each coordinate in \((0,1]^2\) and continuous; and
- for all \(0 \leq \alpha, \beta, \gamma \leq 1\) and \(\varepsilon > 0\), there are sets \(U_4 \subseteq U_3 \subseteq U_2 \subseteq U_1\) such that \(U_3 \neq \emptyset\) and each of \(|\text{Bel}(U_4 \mid U_3) - \alpha|, |\text{Bel}(U_3 \mid U_2) - \beta|, \text{and } |\text{Bel}(U_2 \mid U_1) - \gamma|\) is less than \(\varepsilon\).

In this case, \(\text{Bel}\) is isomorphic to a probability measure.

See (Paris, 1994; Cox, 1946; Halpern, 1999) for justifications of these assumptions. Most of the assumptions are quite straightforward; the cryptic last requirement implies that the range of \(\text{Bel}\) must be dense in \([0, 1]\). The conclusion we come to is that if we want a model of plausibility to meet certain "common-sense" axioms, then it must correspond to a probability measure.

Other justifications for probability exist; see (Halpern, 2003) for an excellent discussion.
Let us begin by defining our problem and establishing some notation.

3.1 The Online Supervised Learning Task

The problem we will consider for the majority of this text is the *supervised learning* task. We observe two random variables, a *predictor* (or *independent*) variable \( X \) and a *response* (or *dependent*) variable \( Y \), and presume a functional relationship between the two. Let \( \mathcal{X} \) and \( \mathcal{Y} \) represent the ranges of \( X \) and \( Y \), respectively. We suppose that there is an unknown function \( f: \mathcal{X} \rightarrow \mathcal{Y} \) such that \( Y = f(X) \). We seek to find a good approximation to \( f \) that will allow us to predict the value of \( Y \) that will accompany a particular value of \( X \).

When \( \mathcal{Y} \) is a discrete set \( \{ c_1, c_2, \ldots, c_k \} \) of \( k \) class labels, this task is called *classification*; when \( \mathcal{Y} \) is the set of real numbers \( \mathbb{R} \), the task is called *regression*. In general, the random variable \( X \) takes the form of an \( d \)-dimensional vector \( x = [x_1, x_2, \ldots, x_d]^T \). Each entry in this vector is called a *feature* and may be of any type: real, integral, ordinal, discrete, etc.

In the classification case, the set \( \mathcal{Y} \) can in general be any discrete set, but in this work we will restrict the discussion to the important *binary classification* problem, corresponding to \( \mathcal{Y} = \{ 0, 1 \} \). Classification problems with more than two classes can often be effectively solved with multiple binary classifiers; for this reason, the impact of this restriction is not great. We will discuss multiple methods for reducing multiclass problems to the binary case below.

In the absence of other information, we will approximate the function \( f \) from a set of known values \( \{(x_i, y_i)\} \); this set is called the *training set*. In the data-stream case, the training set is presented as a sequence indexed by time. For convenience, this sequence is sometimes assumed to be a simple time series, in which case the series is indexed by the natural numbers, \( \mathbb{N} \).

The pursuit of this goal may be hindered by the presence of missing predictors or responses. In many problems, the cost of determining the response associated with a measurement can be quite high (for example, when human intervention is required) compared to the cost of making the measurement itself. In this case, we might have many observations of predictor variables without associated responses. When the distribution of the predictor observations is deemed informative (even in the absence of a full set of responses), the problem of using this information to predict undetermined response values is called *semi-supervised learning* (Zhu, 2008).

In many interesting real-world cases, some of the probability distributions associated with the learning problem will change over time. In general we will allow each of the distributions \( p(x) \), \( p(y) \), and \( p(y \mid x) \) to change in any way at any moment. We call such changes *concept drifts* (Tsybakov, 2004). In this case, we must reformulate the (semi-)supervised learning task slightly. We assume that there is a sequence of functions \( \{ f_t: \mathcal{X} \rightarrow \mathcal{Y} \} \) such that at a given time \( t \), \( Y = f_t(X) \) and seek to approximate this
sequence. This is equivalent to simply incorporating a temporal dimension into the space $\mathcal{X}$.

Solving multiclass classification problems with binary classifiers

The simplicity afforded by restricting the classification problem to the binary case has promoted a great deal of research devoted to solving this particular problem. Some of these methods (such as decision trees) can be easily extended to solve problems with more than two classes; however, in many cases no natural extension is apparent (Allwein, et al., 2000). As a result, several methods have been proposed for solving multiclass classification problems with a collection of arbitrary binary classifiers. We give a brief review below; see (Allwein, et al., 2000) for an excellent discussion.

A naïve solution for a $k$-class problem is to construct $k$ binary classification problems, one for each class. For class $c_i$, a classifier is constructed that attempts to classify observations into the groups “class $c_i$” or “everything else.” The outputs of these classifiers need to be combined in some fashion; assuming that each classifier provides an estimate of the posterior class probability, $Pr(y = c_i \mid x)$, a natural approach would be to output

$$\arg\max_i Pr(y = c_i \mid x),$$

that is, the class with the highest posterior probability estimated by its corresponding classifier. This method is very simple, but is far from ideal. A great deal of important information is ignored, for example, the correlations between the posterior probabilities estimated by each classifier and the true class labels.

A number of more complicated schemes have been proposed. One suggested method builds a classifier for every pair of classes (Hastie and Tibshirani, 1998); although this often works well in practice, it requires the construction of $\binom{k}{2}$ classifiers and therefore significantly increases the amount of computation required.

An interesting compromise between these two extremes was proposed by Dietterich and Bakiri (1995). In their framework, a collection of binary classifiers are trained for different binary partitions of the available classes. By choosing these partitions carefully, an error-correcting output code can be constructed, whereby the outputs of the constructed classifiers can be viewed as a codeword that may be decoded according to the selected class partitions. Allwein, et al. (2000) introduced an extension to this scheme that considers the confidence of each classifier’s output in the decoding step (rather than the predicted class alone), increasing performance.

In the special case where the output classes have a natural ordering, still another approach is useful (Frank and Hall, 2001). Suppose that the classes may be ordered in such a way that

$$c_1 < c_2 < \cdots < c_{k-1} < c_k.$$  

(3.1.2)

An example application with such an ordering is a problem requiring classification into one of a variety of arbitrarily defined size classes, for example,

$$\mathcal{Y}' \doteq \{ \text{“small,” “medium,” “large”} \}.$$  

(3.1.3)

In this case, we may efficiently solve the problem by constructing $k - 1$ two-class models that successively evaluate the probability that the true response is less than label index $i$. We therefore evaluate for $2 \leq i \leq k$

$$a_i \doteq Pr(y < c_i \mid x),$$  

(3.1.4)
which infers the cumulative density over the class posteriors. We may then form the class posteriors using

$$\Pr(y = c_i | x) = \begin{cases} a_2 & i = 1; \\ a_{i+1} - a_i & 2 \leq i \leq k - 1; \\ 1 - a_k & i = k. \end{cases} \quad (3.1.5)$$

There are still other approaches to this problem; for example, Lee and Oh (2003) suggested constructing a tournament of binary classifiers to solve the $k$-class problem. In any case, the restriction of our attention to the binary classification problem should not be regarded with apprehension.

3.2 PROPOSED SOLUTIONS FOR ONLINE CLASSIFICATION WITH CONCEPT DRIFT

A number of previous solutions have been proposed for the task of performing online classification in the presence of concept drift. Less work has focused on the online regression task with concept drift. We will provide a brief survey of existing methods. In general, each of these proposed methods defines a particular discriminating structure for performing the classification task, then specifies a method whereby this structure may be updated to adapt to changes in the system.

Maintaining a set of binary rules

An early approach to this problem, the FLORA family of algorithms, adopted a simple framework where collections of binary rules, or "descriptors" (of the form "feature = value" and conjunctions of these atoms), are maintained using a straightforward algorithm (Widmer and Kubat, 1996). Three sets of rules are constructed: 

$\text{ADes}$, for "accepted descriptors;" $\text{NDES}$, for "negative descriptors;" and $\text{PDES}$, for "potential descriptors.

Every time a new observation is made, rules are added, removed, and moved between these sets as appropriate to maintain consistency with current conditions. Widmer and Kubat (1996) introduced several extensions to this algorithm that allow for adaptively forgetting rules, storing learned sets of rules for future reuse, and more.

The FLORA set of algorithms works well on very simple constructed problems; however, it is applicable only when all input variables are discrete with a small range, a rather severe limitation. The work nonetheless remains an important early approach for the online classification problem with concept drift.

Maintaining a decision tree

Decision trees (Breiman, et al., 1984; Quinlan, 1993) have enjoyed enormous popularity in the classical classification setting, because they are simple to implement and evaluate, easy to interpret, and generalize fairly well. Nearly all methods for constructing decision trees require multiple passes over the entire dataset and are therefore inappropriate for the data-stream setting. Additionally, most decision-tree methods assume that the training data represent a random sample from a stationary distribution and therefore cannot efficiently handle concept drifts.

Domingos and Hulten (2000) introduced the very-fast decision tree (VFDT) algorithm for efficiently building a decision tree from a data stream. The general technique employed is to use the data streaming in to determine the next needed split point in
the decision tree. The first data points seen are used to determine the split point for the root node; after determining this split, the root remains fixed and the next data points are used to grow the decision tree from there. VFDT continues recursively building the tree in this manner. The algorithm employs a technique known as Hoeffding bounds (Hoeffding, 1963) to ensure probabilistic bounds on the quality of the chosen split points. These bounds can guarantee that the chosen splits are the best possible (with a specified confidence), as long as enough data have been observed.

The VFDT algorithm is attractive for a number of theoretical reasons; most importantly, it can be shown that the decision tree grown by VFDT will asymptotically approach the decision tree that would have been grown from a traditional multipass algorithm (Domingos and Hulten, 2000). Unfortunately, VFDT assumes that the data stream is free from concept drift. For this reason, the concept-adapting very-fast decision tree (CvFDT) algorithm was introduced (Hulten, et al., 2001). The CvFDT algorithm represents a simple extension to the VFDT framework, where the purities of internal nodes are monitored for quality. If a particular node in the decision tree moves outside specified required quality bounds, alternate subtrees are grown from that node; if advantageous at a particular point in time, the subtree rooted at that node is replaced with an alternate, and learning continues from that point. This simple mechanism allows for quickly constructing a good decision tree from a data stream, even when the data undergo concept drift.

As originally introduced, CvFDT could only handle discrete inputs; this restriction was later removed by its creators (Hulten and Domingos, 2003).

Maintaining an ensemble of classifiers

Ensemble learning\(^1\) is an increasingly popular approach to supervised learning problems. Instead of forming a single model to explain the data, multiple models are constructed, and their outputs are combined to form a single prediction. Empirical studies have shown that ensembles of learners often perform better than single learners (Dietterich, 2000). Under certain conditions, ensembles perform provably better than a single learner, with the degree of improvement dependent on the diversity of the ensemble (Tumer and Ghosh, 1996). Numerous methods for combining learners have been proposed for the classical setting, including bagging (Breiman, 1996), boosting (Freund and Schapire, 1997), and stacking (Wolpert, 1992). Unfortunately, many of these solutions are impractical in a streaming environment with concept drift. Several alternative methods for constructing ensembles in the data stream setting have been proposed (Street and Kim, 2001; Koltner and Maloof, 2007, 2005); however, many classical techniques remain unexplored in the streaming case.

**THE STREAMING ENSEMBLE ALGORITHM** Street and Kim (2001) provided a simple meta-algorithm (the streaming ensemble algorithm, or SEA) for constructing and maintaining a set of arbitrary classifiers to perform classification on drifting data streams. The authors did not assume the existence of adaptive instance-based learners and instead provided a general method for constructing an ensemble using any available base learning algorithm, including classical techniques for offline learning.

---

\(^1\)Not to be confused with the similarly named “ensemble learning” of variational-Bayesian inference.
To accomplish this task, the authors suggested segmenting the data stream into a series of successive fixed-size blocks. Each time a new block of data is observed, that block is used as training data for a new candidate classifier. If the ensemble has not reached a predefined maximum capacity, the candidate is added to the ensemble. Otherwise, the next block of data is then used to test the previously constructed candidate, as well as all the classifiers present in the ensemble at that time. The authors suggested an ad hoc scoring algorithm to measure the quality of each of these learners; this score was designed to both reward accuracy and encourage diversity among the ensemble members. The classifiers with the highest scores are retained until the next block of data is observed.

When the SEA algorithm must make a classification decision, the authors suggested simply determining the outputs of each of the learners in the ensemble and reporting the output class picked by the majority.

Although SEA is simple and efficient, it suffers from a number of problems. The classifiers comprising the ensemble are generated from a single block of data and never updated. If a drastic concept drift occurred, at least half the members of the ensemble would need to be replaced before the ensemble could be expected to report reasonable results. Given a large block size (which must be chosen to avoid overfitting the member classifiers of the ensemble), this might take a very long time. Additionally, if concept drifts were to occur more frequently than the chosen block size, the candidate classifiers would be built from disparate data and have trouble latching on to a single concept. For these reasons, most block-based approaches to the online learning task fail except under favorable conditions.

**The Dynamic Weighted Majority Algorithm** Kolter and Maloof (2007) introduced a mechanism for maintaining an ensemble of classifiers called the dynamic weighted majority (DWM) algorithm. The method is similar to methods proposed earlier, including the weighted majority (WM) (Littlestone and Warmuth, 1994) and Winnow algorithms (Littlestone, 1988). The DWM algorithm is quite simple. Each member of the ensemble is assigned an associated weight. When a classification must be made, the outputs of each of the ensemble members are combined using a simple weighted majority vote determined by these weights. When an ensemble member makes the wrong prediction, its weight is decreased by a multiplicative constant. Finally, when the ensemble at large makes an incorrect prediction, a new ensemble member is added with an initial unit weight. The creators of DWM also provided a simple mechanism for removing poorly performing ensemble members, although the suggested method cannot guarantee an upper bound on the number of member classifiers and is therefore not suitable for the general data-stream case.

Through a series of experiments, the DWM algorithm was shown to work well in practice, although its ad hoc weighting scheme could not be well motivated.

**The Additive Expert Family of Algorithms** Although DWM was shown to work well in empirical tests, no theoretical results could guarantee its efficiency. Indeed, it was later discovered that DWM can perform very poorly in the worst case (Kolter and Maloof, 2005). Nonetheless, the algorithm inspired the interesting additive-expert (AddExp) series of algorithms. By making small adjustments to the DWM algorithm,
3. PROBLEM DEFINITION AND PREVIOUS WORK

Kolter and Maloof (2005) introduced the AddExp.D ("D" for discrete classes) algorithm for classification and were able to prove theoretical bounds on its performance. In addition, the authors introduced the very similar AddExp.C ("C" for continuous classes) algorithm for performing online regression in the presence of concept drift. Finally, the authors suggested methods for pruning the ensemble when it grows too large, allowing for the algorithm to maintain a constant memory footprint.
Penny and Roberts (1999) introduced the dynamic logistic regressor (DLR) for online binary classification in the presence of concept drift. Recently, Lowne, et al. (2010) have extended this framework for tackling the online semi-supervised learning problem. In this chapter, we will demonstrate how to further extend the DLR algorithm to cope with a corrupted label stream. The ultimate result will be an online method for performing nonstationary semi-supervised learning with missing data and potentially contaminated labels. We will begin with an introduction of the DLR algorithm.

4.1 REVIEW OF THE DYNAMIC LOGISTIC REGRESSOR

We consider the problem of performing streaming binary classification. In this problem, our data are of the form

$$D = \{ (x_i, y_i) \} \subseteq (X, y),$$

where each $x_i \in \mathbb{R}^d$ is an observation of a $d$-dimensional real independent variable and each $y_i \in \{0, 1\}$ is an observation of an associated binary dependent variable. Logistic regression provides a classical way to approach this problem. The logistic-regression model takes the form of a generalized linear model, wherein a linear transformation in feature space is “squashed” through a sigmoid function to approximate the binary response.

Specially, the model takes the form

$$\Pr(y_i = 1 \mid x_i, w) = \sigma(x_i^T w), \quad (4.1.1)$$

where $w \in \mathbb{R}^d$ is a vector of weights and

$$\sigma(a) = \frac{\exp(a)}{1 + \exp(a)} \quad (4.1.2)$$

is the logistic function. The argument $a$ to the logistic function is called the activation. We assume that the labels are generated from independent Bernoulli trials with parameters given by our predictions in (4.1.1):

$$\Pr(y_i \mid x_i, w) = \text{Bernoulli}(y_i; \sigma(x_i^T w)). \quad (4.1.3)$$

The predictor variables $x_i$ are typically augmented with a constant term to allow the model to account for class bias.

This model is equivalent to modeling the log odds (or logits) as a linear transformation of $x_i$; defining

$$z_i = \Pr(y_i = 1 \mid x_i, w),$$

we have

$$\text{logit}(z_i) = \log \left( \frac{z_i}{1 - z_i} \right) = x_i^T w.$$
The logit function is simply the inverse of the logistic function \( \sigma \). In logistic regression, the likelihood factorizes as

\[
\Pr(y \mid X, w) = \prod_i \Pr(y_i \mid x_i, w) = \prod_i z_i^{y_i}(1 - z_i)^{(1-y_i)}. \tag{4.1.4}
\]

In typical applications, the unknown weight vector \( w \) is selected by maximizing this likelihood with respect to its value. The Bayesian approach is also possible; although the posterior \( p(w \mid D) \) cannot usually be calculated analytically, various approximations are possible (Rasmussen and Williams, 2006).

In a streaming environment, new observations \((x_i, y_i)\) will be continually arriving. Re-estimating \( w \) by naive sequential optimization of (4.1.4) will fail, because both \( X \) and \( y \) will be growing ever larger—eventually, both the computational and storage costs will become unbearable. This problem can be easily overcome, however, by adopting a sliding-window technique, where we only keep a fixed number of the most recent observations. Additionally, the sequential optimization problem is made easier by the fact that the maximum-likelihood estimate for \( w \) cannot change very quickly, assuming a window of reasonable width. Therefore, recently optimal points will tend to remain very good initial guesses for the chosen optimization routine, and when the weight vector \( w \) is expected to be stationary,\(^1\) it will work well. However, if the dependence of the classes on the input variables were to change over time (as a result of concept drift), this procedure could perform very poorly.

The DLR addresses this problem by adjusting the model to allow the weights to vary with time. The belief about the current weight distribution is maintained using recursive Bayesian estimation. The DLR assumes that the hidden weights propagate according to known dynamics, and the evolution of their distribution is tracked and predicted using the extended–Kalman filter (EKF) framework (Penny and Roberts, 1999; Lowne, et al., 2010).

**The DLR equations**

In the following discussion, we will adopt standard Kalman-filter notation and use the subscript \( t_2 \mid t_1 \) to indicate estimates for a variable at time \( t_2 \) after observing data up to and including time \( t_1 \). It will also be useful to define the symbol

\[
\mathcal{D}_t \triangleq \{(x_i, y_i)\}_{i=t_1}^t.
\]

In the DLR model, the weights \( \{w_t\} \) are assumed to evolve under a hidden-Markov process with transition function \( f \):

\[
w_t \triangleq f(w_{t-1}) + \eta_t, \tag{4.1.5}
\]

where \( \eta_t \) represents zero-mean additive Gaussian noise with covariance \( Q_t \):

\[
p(\eta_t \mid Q_t) \triangleq \mathcal{N}(0, Q_t).
\]

\(^1\)That is, the conditional distribution \( p(y \mid x) \) is assumed to be time-invariant.
The responses are modeled as the output of a nonlinear function $g$ of the observed predictor variables and the hidden weight:

$$y_t = g(x_t, w_t) + v_t,$$

where $v_t$ represents zero-mean additive Gaussian noise with variance $r_t^2$:

$$p(v_t | r_t) = \mathcal{N}(0, r_t^2).$$

The specific function assigned to $g$ in the DLR is the marginal probability that $y_t = 1$ given $x_t$ and $D_{t-1}$:

$$g(x_t, w_t) = \int \Pr(y_t = 1 | x_t, w_t) p(w_t | D_{t-1}) \, dw_t = \Pr(y_t = 1 | x_t, D_{t-1}).$$

As before we define

$$\Pr(y_t = 1 | x_t, w_t) = \sigma(x_t^T w_t).$$

We will calculate an approximation for the marginal probability below.

Combined, this dynamical system realizes a nonstationary model for logistic regression. A graphical model of the DLR is displayed in Figure 4.1.

We suppose the following simple observation paradigm. At time $t$, we observe the independent variable $x_t$. We are then compelled to predict the corresponding label $y_t$ before learning its value. At that point, the true value of the label is revealed to us, and we update our beliefs accordingly before proceeding to time $t + 1$.

We follow Lowne, et al. (2010) and set $r_t^2$ equal to the variance of the Bernoulli distribution $\Pr(y_t | x_t, D_{t-1})$:

$$r_t^2 = \text{var}(\Pr(y_t | x_t, D_{t-1})).$$
which we will derive presently. We will leave discussion about the choice of $Q_t$ until the next section.

In the EKF, our belief about the value of the hidden state is multivariate Gaussian. Suppose that before the data point $(x_t, y_t)$ arrives, the weight distribution conditioned on the observations up to time $t-1$ is given by

$$p(w_{t-1} | D_{t-1}) \triangleq \mathcal{N}(\bar{w}_{t-1|t-1}, P_{t-1|t-1}),$$

with mean $\bar{w}_{t-1|t-1}$ and covariance $P_{t-1|t-1}$. We will show how to process the observation $(x_t, y_t)$ and derive the posterior weight distribution

$$p(w_t | D_t) \triangleq \mathcal{N}(w_t; \bar{w}_{t|t}, P_{t|t}).$$

After observing $x_t$, we must make a prediction about $y_t$. This requires having a consistent belief about the value $w_t$ given our belief about $w_{t-1}$, which is complicated by the presence of the possibly nonlinear function $f$ in (4.1.5). For simplicity, Penny and Roberts (1999) and Lowne, et al. (2010) assumed no systematic dynamics for the weights, reducing (4.1.5) to a simple diffusion:

$$w_t = w_{t-1} + \eta_t. \quad (4.1.6)$$

For the present discussion we will also make this assumption.\(^2\) Our belief about $w_t$ given observations up to time $t-1$ can now be derived easily. Because $w_t$ is the sum of two independent Gaussian-distributed random variables (4.1.6), we may apply (4.3) to calculate

$$p(w_t | D_{t-1}) = \mathcal{N}(w_t; \bar{w}_{t|t-1}, P_{t|t-1}),$$

with

$$\bar{w}_{t|t-1} = \bar{w}_{t-1|t-1}, \quad (4.1.7)$$
$$P_{t|t-1} = P_{t-1|t-1} + Q_t. \quad (4.1.8)$$

Given our belief about $w_t$ and the observation $x_t$, we may now predict the (still unobserved) label $y_t$. Because we do not know the value of $w_t$ exactly, we cannot directly apply (4.1.1). Instead, we must marginalize out the unknown parameter $w_t$ under our belief:

$$z_{t|t-1} \triangleq \Pr(y_t = 1 | x_t, D_{t-1})$$
$$= \int \Pr(y_t = 1 | x_t, w_t) p(w_t | D_{t-1}) \, dw_t$$
$$= \int \sigma(x_t^T w_t) \mathcal{N}(w_t; \bar{w}_{t|t-1}, P_{t|t-1}) \, dw_t. \quad (4.1.9)$$

The activation at time $t$,

$$a_t \triangleq x_t^T w_t,$$
4.1. Review of the Dynamic Logistic Regressor

is a linear transformation of the Gaussian-distributed \( w_t \). We may therefore apply (A.6) to see that \( a_t \) is also Gaussian distributed according to

\[
p(a_t \mid D_{t-1}) = \mathcal{N}(a_t; \tilde{a}_{t|t-1}, \tilde{s}_{t|t-1}^2),
\]

with

\[
\tilde{a}_{t|t-1} = x_t^T \tilde{w}_{t|t-1},
\]

\[
\tilde{s}_{t|t-1}^2 = x_t^T P_{t|t-1} x_t.
\]  

(4.1.10)

(4.1.11)

This observation allows us to rewrite (4.1.9) as a one-dimensional integral:

\[
z_{t|t-1} = \int \sigma(x_t^T w_t) \mathcal{N}(w_t; \tilde{w}_{t|t-1}, \tilde{P}_{t|t-1}) \, dw_t
\]

\[
= \int \sigma(a_t) \mathcal{N}(a_t; \tilde{a}_{t|t-1}, \tilde{s}_{t|t-1}^2) \, da_t.
\]  

(4.1.12)

The integral in (4.1.12) cannot be evaluated analytically. Fortunately, MACKAY (1992b) has shown that it can be approximated effectively by

\[
z_{t|t-1} \approx \sigma(\tilde{a}_{t|t-1} \kappa(s_{t|t-1})),
\]  

(4.1.13)

where

\[
\kappa(s) = \left(1 + \frac{\pi s^2}{8}\right)^{-\frac{1}{2}}.
\]

Using this approximation, we may complete the specification of the dynamical equations for the label \( y_t \):

\[
y_t \approx \sigma(a_t \kappa(s_t)) + v_t,
\]  

(4.1.14)

and

\[
r_t^2 \approx z_{t|t-1}(1 - z_{t|t-1}).
\]

The overall effect of marginalizing out \( w_t \) is to make the class-conditional probability \( \Pr(y_t = 1 \mid x_t, D_{t-1}) \) more conservative (that is, nearer to \( \frac{1}{2} \)) by an amount that is dictated by the uncertainty in the activation.

After observing the true label \( y_t \), we need to update our belief about the hidden state \( w_t \) given this information. In the DLR, this is complicated by the nonlinear link function \( \sigma \) in (4.1.14). If \( \sigma \) were linear, \( w_t \) and \( y_t \) would be jointly Gaussian distributed, and we would be able to apply simple Gaussian identities to derive the desired distribution. The EKF solution is to linearize \( \sigma \) at the point \( \tilde{w}_{t|t-1} \). We define

\[
S_t = \frac{\partial \sigma(a_t \kappa(s_t))}{\partial w} \bigg|_{w=\tilde{w}_{t|t-1}}
\]

\[
= \sigma(\tilde{a}_{t|t-1} \kappa(s_{t|t-1})) \left(1 - \sigma(\tilde{a}_{t|t-1} \kappa(s_{t|t-1}))\right) \kappa(s_{t|t-1}) x_t^T
\]

\[
= z_{t|t-1}(1 - z_{t|t-1}) \kappa(s_{t|t-1}) x_t^T.
\]

We use this derivative to make the natural linear approximation to \( \sigma \):

\[
\sigma(a_t \kappa(s_t)) \approx z_{t|t-1} + S_t(w_t - \tilde{w}_{t|t-1}).
\]
The distribution of $y_t$ given $w_t$ is now approximately Gaussian:

$$p(y_t \mid x_t, w_t, D_{t-1}) \approx \mathcal{N}(y_t; z_{t|t-1} + S_t(w_t - w_{t|t-1}), r_t^2).$$

Because the conditional distribution of $y_t$ given $w_t$ has a mean that is an affine transformation of $w_t$, we may apply (A.4) to calculate the joint distribution

$$p(w_t, y_t \mid x_t, D_{t-1}) = \mathcal{N}\left(\begin{bmatrix} w_t \\ y_t \\ z_{t|t-1} \end{bmatrix}; \begin{bmatrix} \bar{w}_{t|t-1} \\ \bar{y}_{t|t-1} \\ \bar{z}_{t|t-1} \end{bmatrix}, \begin{bmatrix} P_{t|t-1} & P_{t|t-1} S_t^T \\ S_t P_{t|t-1} & S_t P_{t|t-1} S_t^T + r_t^2 \end{bmatrix}\right).$$

Finally, we may apply (A.2) to condition the above distribution on the observation $y_t$, giving the posterior distribution of $w_t$ given $D_t$:

$$p(w_t \mid D_t) = \mathcal{N}(w_t; \bar{w}_{t|t}, \bar{P}_{t|t}),$$

where

$$\bar{w}_{t|t} = w_{t|t-1} + K_t (y_t - z_{t|t-1}), \quad \bar{P}_{t|t} = P_{t|t-1} - r_t^2 K_t S_t^T P_{t|t-1}.$$

The common term

$$K_t = P_{t|t-1} S_t^T (S_t P_{t|t-1} S_t^T + r_t^2)^{-1} = \left(\frac{\kappa(z_{t|t-1})}{1 + (r_t s_{t|t-1} \kappa(z_{t|t-1}))}\right) P_{t|t-1} x_t$$

(4.1.17)

is known as the Kalman gain.

Given $p(w_t \mid D_t)$, we are now prepared to process the next data point $(x_{t+1}, y_{t+1})$—the newly found posterior simply becomes the prior for the next time step. In this way, the DLR recursively progresses through the data.

Pseudocode for the DLR algorithm is shown in Algorithm 4.1. The management of the process-noise parameter $Q_t$ referenced in the listing will be described in the next section.

4.2 Notes Regarding the DLR

Let us briefly discuss various issues related to the DLR, including how to estimate the process-noise parameter $Q_t$, and how to handle missing observations and labels.

**Estimating the process-noise parameter $Q_t$**

The discussion in the previous section does not specify how the noise parameter $Q_t$ is derived. In practice, this parameter will not be known exactly and instead must be inferred from the data. A method for deriving a suitable $Q_t$ from the data stream was discussed at length by LOWE, et al. (2010). The suggested approach was to use a time-variant, isotropic diagonal covariance matrix

$$Q_t = q_t I.$$
Handling missing observations

Shumway and Stoffer (2005) have provided a very simple method for addressing missing observations in the Kalman-filter framework. The solution was to simply replace
any missing features in \( \mathbf{x}_t \) with zeros. The effect of this action is that the corresponding entries of \( \mathbf{w}_{it} \) and \( \mathbf{P}_{it} \) will not be modified by the update equations (4.1.15–4.1.16).

If the fact that a particular feature is missing might be indicative of the class label,\(^3\) it might be helpful to augment the data \( \mathbf{x}_t \) with a vector of binary variables indicating whether each possibly missing variable was observed at time \( t \).

### Handling missing labels

LOWNE, et al. (2010) extended the DLR algorithm to address the semi-supervised learning problem. The suggested solution was simple. If the true class label \( y_t \) is missing, we replace the missing label with the “quasitarget”

\[
\hat{y}_t = \chi \left( \Pr(y_t = 1 \mid \mathbf{x}_t, \mathcal{D}_{t-1}) > \frac{1}{2} \right) \\
= \chi(z_{t|t-1} > \frac{1}{2}),
\]

where \( \chi \) is the indicator function. The DLR then proceeds as before with \( \hat{y}_t \) in place of the missing true value. To offset potential errors in the imputed quasitargets, LOWNE, et al. (2010) suggested increasing the process-noise parameter \( q_{t+1} \) by our uncertainty about the missing true value of \( y_t \):

\[
q_{t+1} = \max(u_t, 0) + z_{t|t-1}(1 - z_{t|t-1}).
\]

### 4.3 Modifying the DLR to Handle Corrupted Labels

In many applications, the labels cannot be expected to always represent the ground truth. The labeling process could be corrupted by a number of factors, including errors during communication and data entry. In a situation with many errors in the label stream, the DLR method described above can suffer from overcompensation and overly drastic changes to the distribution of the weight parameter \( \mathbf{w} \).

To illustrate, assume the DLR has developed the ability to make reasonably confident decisions, that is, decisions with \( \Pr(y_t = 1 \mid \mathbf{x}_t, \mathcal{D}_{t-1}) \) near 0 or 1. In this case, given an incorrect label \( y_t \), the innovation residual

\[
(y_t - z_{t|t-1})
\]

in (4.1.15) would be near \( \pm 1 \). Correspondingly, the weight distribution would experience a very large shift, and the decision boundary would move drastically in response. This behavior should be expected when the labels can be guaranteed to be correct—after all, if we were very confident about a decision that turned out to be incorrect, something must be wrong with our classifier. When the truth of the labels cannot be assured, however, we should tread more carefully and temper our decisions by our uncertainty in the labels. We may modify the DLR model accordingly.

### A simple noise model and its consequences

Let us adopt a simple model for the noise present in the labeling process. For the discussion below, let \( \tilde{y}_t \) represent the (possibly incorrect) observed label, and let \( \hat{y}_t \) represent the true, hidden label. Our observed data, therefore, will now be written

\[
\mathcal{D} \triangleq \{ (\mathbf{x}_t, \tilde{y}_t) \} \triangleq \{ (\mathbf{X}, \tilde{y}) \}.
\]

\(^3\)That is, the features are not missing at random (MAR); see (LITTLE and RUBIN, 2002; SCHAFER, 1997) for details.
4.3. Modifying the DLR to Handle Corrupted Labels

We assume the probability that an observed label will be incorrect is independent of the data observed and fixed throughout time:

\[ \Pr(\tilde{y}_t \neq y_t \mid x_t, t) = \Pr(\tilde{y}_t \neq y_t) \equiv \rho. \]

This assumption may not always be true, but in many important examples (such as errors introduced by a noisy communication channel), it will be valid. In many other cases, it can serve as a useful substitute for a more complicated model. To simplify the present discussion, let us assume that the probability \( \rho \) is known \textit{a priori}. Of course, in almost any situation this will not be true. We will propose an online method for estimating \( \rho \) from the data stream below.

In our description of the DLR, we assumed that the label \( y_t \) was determined from a Bernoulli distribution with parameter \( z_{t|t-1} \):

\[ \Pr(y_t \mid z_{t|t-1}) = z_{t|t-1}^{y_t}(1 - z_{t|t-1})^{(1-y_t)}. \]

Under our label-noise model, this assumption is violated. Instead, we may derive the distribution of \( \tilde{y}_t \) given \( z_{t|t-1} \) and \( \rho \):

\[ \Pr(\tilde{y}_t \mid z_{t|t-1}, \rho) = \rho + (1 - 2\rho) \left( z_{t|t-1}^{\tilde{y}_t}(1 - z_{t|t-1})^{(1-\tilde{y}_t)} \right). \]

This discrepant distribution may reconciled by replacing \( z_{t|t-1} \) with an appropriately modified output \( \tilde{z}_{t|t-1} \), defined by

\[ \tilde{z}_{t|t-1} \equiv \Pr(\tilde{y}_t = 1 \mid x_t, D_{t-1}) = (1 - 2\rho)z_{t|t-1} + \rho. \]

We now have

\[ \Pr(\tilde{y}_t \mid \tilde{z}_{t|t-1}) = \tilde{z}_{t|t-1}^{\tilde{y}_t}(1 - \tilde{z}_{t|t-1})^{(1-\tilde{y}_t)}, \]

that is, the original relation between the output of the DLR and the observed label is restored. Under the assumed noise model, the only required modification to the DLR framework is to replace the original output of the model, \( z_{t|t-1} \) (which assumes no noise in the labels), with the value \( \tilde{z}_{t|t-1} \) given above. The definition of \( \tilde{z}_{t|t-1} \) serves to moderate the certainty of our classifications according to our uncertainty in the labels.

**Estimating \( \rho \)**

As mentioned above, the true value of \( \rho \) will almost certainly not be known in typical applications. Fortunately, we may estimate the true value of \( \rho \) online from the data. To simplify the following expressions, define

\[ \pi_t \equiv z_{t|t-1}^{y_t}(1 - z_{t|t-1})^{(1-y_t)}. \]

Given our modified model, the likelihood factorizes as

\[ \mathcal{L}(\rho \mid D_t) \equiv \Pr(\tilde{y} \mid X, \rho) = \prod_{i=1}^{t} \Pr(\tilde{y}_i \mid x_i, D_{i-1}, \rho) = \prod_{i=1}^{t} \rho + (1 - 2\rho)\pi_i. \]

\[ (4.3.1) \]
We may estimate $\rho$ using a maximum-likelihood estimation scheme, where we seek to maximize (4.3.1) as a function of $\rho$. Because the logarithmic function is monotonically increasing, the maximum of $\mathcal{L}$ will not be affected by taking the logarithm:

$$
\log \mathcal{L}(\rho \mid D_t) = \sum_{i=1}^{t} \log \left( \rho + (1 - 2\rho) \pi_i \right).
$$

(4.3.2)

Furthermore, the derivative of (4.3.2) with respect to $\rho$ may be easily calculated:

$$
\frac{d \log \mathcal{L}(\rho)}{d \rho} = \sum_{i=1}^{t} \frac{1 - 2\pi_i}{\rho + (1 - 2\rho) \pi_i}.
$$

(4.3.3)

Given this derivative, optimizing $\log \mathcal{L}(\rho)$ for $\rho \in [0, 1]$ is a trivial task. As in our discussion of classical logistic regression, storing all the previous observations may be impossible; however, this can again be mitigated by using a moving-window approach. Moreover, the assumption that $\rho$ is fixed through time implies that previously estimated values will continue to work well in the future, so $\rho$ only needs to be periodically re-estimated. Finally, even when $\rho$ is not a fixed constant throughout time, sequential re-estimation using a sliding window should work fairly well.

The above discussion provides a simple modification to the DLR framework that effectively handles label noise with no additional parameters and very little additional computational overhead. Even when the labels are noise free, the above formulation may still be used without issue—the estimated parameter $\rho$ will quickly assume a small value accordingly.

4.4 EXPERIMENTAL RESULTS

We have carried out several experiments to evaluate the effectiveness of the DLR for nonstationary dynamic classification and the method described above for handling corrupted label streams.

Implementation and initialization

The DLR was implemented as described in Algorithm 4.1. A bias term was incorporated into the model by augmenting the predictor variables $\{x_i\}$ with a constant. Additionally, the label-noise modification described in Section 4.3 was implemented, and the $\rho$ parameter was estimated online using the technique described in Section 4.3. The `fmincon` function in the MATLAB Optimization Toolbox (The MathWorks, Inc., 2010a) was used to optimize the log-likelihood function (4.3.2) using the derivative calculated in (4.3.3). The $\rho$ parameter was re-estimated every 100 data points.

All components of the observed data were normalized to have zero mean and unit variance.

In all the experiments reported here, we initialized all components of $\tilde{w}_{0|0}$ to zero and both $P_{0|0}$ and $Q_{0}$ to the identity matrix.

Experiment 1: An illustrative example

We first discuss an expository example first described bylowne, et al. (2010).
**DESCRIPTION OF DATASET** To illustrate the DLR’s ability to perform effective classification in a nonstationary system, we considered a simple problem with constant concept drift. The task is a synthetic binary classification problem with a two-dimensional input space. The class-conditional distributions are multivariate Gaussian with equal spherical covariance. The class-conditional means oppose each other an equal distance from the origin and rotate around that point at a constant angular velocity.

We considered two such datasets—one with little overlap between the classes (Bayes error⁴ 4.5%) and a harder problem with more overlap (Bayes error 22%). Each dataset contained 1,000 points at discrete time steps from \( t = 1 \) to \( t = 1,000 \) and completed one full revolution around the origin in that interval. To construct the dataset, at each time step, a class was chosen randomly with equal probability, then the associated Gaussian at that time was sampled randomly to produce the corresponding input.

**TESTING PERFORMANCE WITH FULL LABEL SET** By construction, the optimal decision boundary for this problem constantly rotates around the origin along with the data. As a result, classical methods for stationary binary classification will fail. A standard logistic regression classifier trained on the entire dataset performed very poorly, correctly classifying only 52% of the data for the 4.5%–Bayes error dataset and only 49.5% of the data for the 22%–Bayes error dataset.

By contrast, the DLR trained online on the same datasets achieved very close to the best possible performance, correctly classifying 94.3% of the data for the 4.5%–Bayes error dataset and 77.3% of the data for the 22%–Bayes error dataset. Figure 4.2 shows the DLR algorithm in action. At two different times (\( t = 100 \) and \( t = 300 \)), the figure shows the last seventy-five data points observed by the DLR, their associated classes, and the learned decision boundary at that time. The learned boundaries are all excellent discriminators, even on the problem with large class overlap.

**TESTING PERFORMANCE WITH MISSING LABELS** After observing that the DLR worked very well on the rotating datasets with a full set of class labels, we proceeded to test the DLR’s ability to cope with missing labels. We selected a discretized range of label-observation probabilities in the interval \([0, 1]\). For each point in that range, we synthesized twenty-five random maskings of the labels for each of the rotating datasets and measured the accuracy of the DLR using these masked label sets. Figure 4.3 shows the results. Even when missing half the labels, the DLR continued to show near-optimal performance, correctly classifying an average of 94.4% of the points in the 4.5%–Bayes error dataset and an average of 75.8% of the points in the 22%–Bayes error dataset. When given 25% of the labels, the DLR still achieved an average of 86.9% and 70.4% accuracy on these datasets, respectively. The DLR can clearly perform very well even when a large portion of the labels are missing.

*Experiment 2: The sea datasets*

To continue testing the DLR’s ability to perform classification on nonstationary data streams, we subjected it to a simulated problem based on one previously presented for this purpose that we modified to be more difficult.

---

⁴The Bayes error is the error rate that a theoretically optimal classifier would achieve on a problem.
Figure 4.2: The seventy-five most recently observed data points and the associated decision boundaries learned by the DLK at two different times in the rotating datasets.

DESCRIPTION OF DATASET

The dataset, which was first presented by Street and Kim (2001) to test the streaming ensemble algorithm (SEA), is a binary classification problem with 60,000 data points. A similar dataset was used by Koller and Maloof (2005) to test the AddExp family of algorithms. Each feature is independent and identically distributed with uniform distribution $U([0, 10])$. The original dataset contained three features, but we used up to two. For a selected input $x$, the corresponding output depends only on a subset $\{x_i\}_{i \in I}$ of its features. The class label is completely determined by the mean value of the selected features according to the function

$$
\chi\left(\frac{1}{|I|} \sum_{i \in I} x_i < \xi\right),
$$

for some value $\xi$. In the original dataset, four concept drifts were introduced by periodically modifying the value of $\xi$. To increase the problem’s difficulty, we introduced many more. Every 500 data points, we randomly selected a new value for $\xi$ from the uniform distribution $U([3.5, 6.5])$. We further increased the problem’s difficulty by
4.4. Experimental Results

Figure 4.3: Mean accuracy as a function of label-observation probability for the rotating Gaussian datasets. Twenty-five random labelings were used at each point; error bars depict one standard deviation. The dotted lines indicate the best possible generalization error for each of the datasets.

also choosing a new relevant feature subset every 6,000 data points. Finally, label noise was introduced by replacing 10% of the responses with values drawn independently from a Bernoulli distribution with parameter \( p = \frac{1}{2} \). We will identify these datasets using the notation \( \text{sea}(d, r) \), where \( d \) signifies the dimension of the input space and \( r \) signifies the number of relevant features. For our experiments, we considered the datasets \( \text{sea}(d, r) \) for all combinations with \( d \in \{50, 100, 150, 200\} \) and \( r \in \{2, 10, 25\} \).

**EXPERIMENTAL SETUP AND RESULTS** We compared the performance of the DLR on these datasets with the performance of a standard logistic regression classifier built from the entire dataset (LR), a standard logistic regression classifier using a sliding-window approach as described in Section 4.1 (SWLR), and the concept-varying very-fast decision tree (CVFDT) algorithm described in Section 3.2.

The specific standard logistic regression classifier used was the implementation provided by the \texttt{glmfit} and \texttt{glmval} methods in the \texttt{MATLAB} Statistics Toolbox (MATLAB 	extsc{Inc.}, 2010b). The logit link function was specified. The \texttt{MATLAB} implementation automatically incorporates a constant bias term into the model. The sliding-window logistic regression classifier was built in the same manner, but used only the previous 2,000 data points to predict the current label. The model parameters were re-estimated every 50 data points.

The Very Fast Machine Learning (VFML) toolkit (Hulten and Domingos, 2003) supplied the code for the CVFDT algorithm. The CVFDT implementation provided in the toolkit has been extended by the VFML creators to handle continuous data, a feature
Table 4.1: Mean accuracy of various methods on the sea datasets. The best performance on each dataset is highlighted in bold. For the results marked with a dagger, the reference cvFDT implementation ended prematurely with a segmentation fault for some or all of the corresponding datasets.

<table>
<thead>
<tr>
<th>SEA dataset</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>r</td>
</tr>
<tr>
<td>2</td>
<td>0.605</td>
</tr>
<tr>
<td>50</td>
<td>0.609</td>
</tr>
<tr>
<td>25</td>
<td>0.606</td>
</tr>
<tr>
<td>2</td>
<td>0.595</td>
</tr>
<tr>
<td>100</td>
<td>0.591</td>
</tr>
<tr>
<td>25</td>
<td>0.586</td>
</tr>
<tr>
<td>2</td>
<td>0.594</td>
</tr>
<tr>
<td>150</td>
<td>0.585</td>
</tr>
<tr>
<td>25</td>
<td>0.577</td>
</tr>
<tr>
<td>2</td>
<td>0.595</td>
</tr>
<tr>
<td>200</td>
<td>0.582</td>
</tr>
<tr>
<td>25</td>
<td>0.570</td>
</tr>
</tbody>
</table>

not provided by the original published specification for the cvFDT algorithm. Because of this ability, the dataset did not need to be discretized for cvFDT. The vFml default parameters were used. The default parameters for cvFDT were used, except the allowed chance for error in each decision was increased from 1% to 20%, which improved its performance slightly.

To decrease potential bias caused by random fluctuations in the generated datasets, twenty-five sea datasets were created for each \((d, r)\) pair, and the performance of each algorithm was determined from its aggregate performance on these datasets.

Table 4.1 shows the results. On each of the datasets tested, the DLR algorithm outperformed the others by a significant margin.

Experiment 3: Identifying label noise

Finally, we carried out an experiment to test the ability of the DLR to identify and react to label noise as described in Section 4.3. Specifically, we wanted to ensure that the procedure for estimating the \(\rho\) parameter described in Section 4.3 is accurate and that the associated modified DLR equations provide a benefit to performance in the presence of corrupted labels.

TESTING ABILITY TO ESTIMATE LABEL-NOISE LEVEL. We first addressed the capacity of the DLR to accurately predict the amount of label noise present in a data stream using the method outlined in Section 4.3. We applied the modified DLR to the sea(100, 2) dataset with different values of known added label noise from 5% to 15% and observed the estimated value of \(\rho\) through time. Figure 4.4 shows the results. At the 5% and 10% noise levels, the DLR estimated the \(\rho\) parameter remarkably accurately. At the 15% noise level, the DLR slightly overestimated the \(\rho\) parameter by about 1%. This difference might
be explained by the difficulty of the problem—with high label noise, it is challenging to determine whether a misclassification was caused by ignorance in the model or by corruption in the label stream. Nonetheless, the modified DLR can clearly estimate added label noise with reasonable accuracy.

**TESTING IMPACT OF LABEL-NOISE MODIFICATION**

We also wanted to determine whether there is a benefit to classifier accuracy when using the modified DLR on a data stream with noisy labels. We applied both the unmodified and the modified DLR to the same SEA datasets used to compile Table 4.1, which by construction had labels flipped with probability $\rho \approx 5\%$. To test our hypothesis, we applied a Bayesian version of the two-sample paired $t$-test to the results of these two algorithms.

The two-sample paired $t$-test makes the assumption that for a particular value of $(d, r)$, the paired results of the two algorithms on each of the twenty-five $SEA(d, r)$ datasets will be identically and independently Gaussian distributed as

$$p(r_u | \mu_u, \sigma) \sim \mathcal{N}(\mu_u, \sigma^2),$$

$$p(r_m | \mu_m, \sigma) \sim \mathcal{N}(\mu_m, \sigma^2),$$

where $r_u$ and $r_m$ represent the results of the unmodified and modified DLR algorithms on the shared dataset, respectively, $\mu_u$ and $\mu_m$ represent their mean performances on the $SEA(d, r)$ datasets, and $\sigma^2$ represents a common variance around those means. With this assumption, we may apply (A.3) to see that the difference in performance will also

![Figure 4.4: The mean value of the label-noise parameter $\rho$ inferred by the DLR through time on the $SEA(100, 2)$ dataset with three different amounts of added label noise. Twenty-five random datasets were created for each noise level; error bars indicate one standard deviation.](image)
Table 4.2: Mean accuracy of the unmodified (UDLR) and modified (MDLR) DLR algorithms on the sea datasets and the probability that the difference Δ (4.4.2) is positive. The best performance on each dataset is highlighted in bold.

| SEA dataset | Algorithm | d | r | Pr(Δ > 0 | D) |
|-------------|-----------|---|---|-----------|
|             | UDLR      | 2 | 10| 0.769     | 0.783     | 1 − 6.8 × 10⁻¹⁹ |
|             |           | 25| 10| 0.782     | 0.786     | 1 − 1.6 × 10⁻⁵  |
| 100         | UDLR      | 2 | 10| 0.773     | 0.733     | 1 − 2.1 × 10⁻¹⁹ |
|             |           | 25| 10| 0.717     | 0.720     | 1 − 1.4 × 10⁻²  |
| 150         | UDLR      | 2 | 10| 0.737     | 0.771     | 1 − 1.2 × 10⁻²² |
|             |           | 25| 10| 0.696     | 0.709     | 1 − 1.5 × 10⁻¹² |
| 200         | UDLR      | 2 | 10| 0.729     | 0.757     | 1 − 5.8 × 10⁻²³ |
|             |           | 25| 10| 0.686     | 0.696     | 1 − 3.4 × 10⁻⁸  |
|             |           | 25| 10| 0.661     | 0.662     | 0.95            |

be normally distributed:

\[ p(r_m − r_u | \mu_u, \mu_m, \sigma) = \mathcal{N}(\mu_m − \mu_u, 2\sigma^2). \]

We wish to calculate the posterior probability that the difference

\[ \Delta \equiv \mu_m − \mu_u \] (4.4.2)

is positive, that is, that the modified algorithm had better average performance for the dataset under consideration. A simple calculation shows

\[
\begin{align*}
    p(\Delta | D) &= \int p(\Delta | D, \sigma) p(\sigma) \, d\sigma \\
    &= \int \int p(D | \Delta, \sigma) p(\Delta) p(\sigma) \, d\sigma \\
    &= \int \int p(D | \Delta, \sigma) p(\Delta) \, d\Delta \, d\sigma.
\end{align*}
\]

The probability \( p(D | \Delta, \sigma) \) can be calculated easily due to the independence assumption. Given this distribution, we can calculate our desired probability:

\[
\Pr(\Delta > 0 | D) = \int_0^\infty p(\Delta | D) \, d\Delta. \tag{4.4.3}
\]

Because we cannot know the value of the difference \( \Delta \) or the shared variance \( \sigma^2 \) a priori, the Bayesian version of this test insists that we must select priors and marginalize. From the nature of the measurements, we can surmise that the difference \( \Delta \) must lie in the interval \([-1, 1]\). Additionally, the shared variance \( \sigma^2 \) must lie in the interval \([0, 1/4]\). In light of these observations, we selected uniform priors over these ranges. The evaluation of the integral in (4.4.3) was then performed using the \texttt{dblquad} function in MATLAB, which approximates the integral of a function over a rectangular region in \( \mathbb{R}^2 \) using recursive adaptive Simpson quadrature (THE MATHWORKS, INC., 2010c).
Table 4.2 shows the results. For every dataset, the mean accuracy of the label noise-aware DLR was higher than the unmodified DLR. Additionally, the Bayesian paired test results firmly establish that the modified DLR performed better than the unmodified DLR; even for the closest result (for the sea(200, 25) dataset), we can conclude with 95% probability that the modified DLR was superior. There is clear utility to using our proposed modification to the DLR on data streams suspected of containing label noise.
The chief inference procedure that we will use throughout the remainder of this work is the Gaussian process \((\text{GP})\). Gaussian processes offer a powerful method for performing Bayesian inference about functions (Rasmussen and Williams, 2006).

Many common machine-learning problems, including regression, classification, optimization, and time-series prediction, may be naturally approached by modeling observed data as noisy observations of an underlying, unobserved latent function. For example, in Chapter 2, we considered a Bayesian treatment of linear regression, which models the dependent variable as a linear latent function of the inputs corrupted by Gaussian noise, and in Chapter 4, we considered logistic regression, which models the expectation of the dependent variable as a linear latent function of the inputs that is “squashed” through a sigmoid function.

Although the inference for these models was straightforward, these simple linear models are inherently inflexible. Any deviation from the rigid assumptions made (for example, one that might be caused by the data undergoing a drift in concept) can quickly render the model essentially useless. This is not an insurmountable problem—the DLR, for example, copes by placing a dynamic structure on the model parameters, then performing inference about their evolution as more data is received. Perhaps a more worrying complaint about these standard linear models is that they cannot easily incorporate further \(a \text{ priori}\) knowledge about the underlying latent function, for example the presence of periodic components, a known degree of differentiability, or (of course) that the latent function is nonlinear. Some of these complaints can again be addressed with further modifications to the model, for example, by using a basis expansion or by augmenting the state variables in a Kalman-style model. Gaussian processes, on the other hand, provide a simple, effective, and very adaptive model framework that can easily incorporate a wide array of prior knowledge in a straightforward manner.

We will briefly cover the simplest application of Gaussian processes—performing regression on an arbitrary function given noisy observations of it. We will then discuss various topics related to Gaussian processes relevant for this work, and finally, we will consider the problem of hyperparameter management in \(\text{GP}\) models.

### 5.1 Definition and Prior Distribution

In a sense, a Gaussian process is simply the natural extension of a Gaussian distribution to a real-valued function space.

**Definition 5.1.1 (Gaussian Process)** A Gaussian process on a space \(\mathcal{X}\) is a distribution over the functions \(\mathcal{X} \to \mathbb{R}\) such that the distribution of the possible function values at any finite set \(F \subseteq \mathcal{X}\) is multivariate Gaussian.

Many results about the multivariate Gaussian distribution (for example, the properties listed in Appendix A) have natural analogs for Gaussian processes. Consider a
function \( y(x) : \mathcal{X} \to \mathbb{R} \). Just as a multivariate Gaussian distribution is completely specified by its first two moments—a mean vector and a symmetric, positive-semidefinite covariance matrix—a Gaussian process is completely specified by its first two moments (now functions)—a mean function \( \mu : \mathcal{X} \to \mathbb{R} \),

\[
\mu(x) \doteq \mathbb{E}[y(x)],
\]

and a symmetric, positive-semidefinite covariance function \( K : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \),

\[
K(x, x') \doteq \mathbb{E}[ (y(x) - \mu(x))(y(x') - \mu(x'))].
\]

For simplicity, we often work with parametrized families of mean and covariance functions, rather than constructing entirely new, ad hoc functions for each situation. For example, a covariance function modeling a periodic component of the function \( y \) would naturally include the period and amplitude of that component as parameters.

Because these parameters specify our prior distribution, they serve as model hyperparameters. We will use the notation \( \theta \) to refer to the vector specifying the values of all the hyperparameters required by our choice of model. Later in this chapter, we will discuss how our hyperparameters can be effectively managed via estimation or marginalization.

Given valid mean and covariance functions and the values of any associated hyperparameters, for a finite set of inputs \( X_* \subseteq \mathcal{X} \), the prior distribution of the associated function values \( y_* \doteq y(X_*) \) is simply

\[
p(y_* | X_*, \theta) \doteq \mathcal{N}(y_*; \mu(X_*; \theta), K(X_*, X_*; \theta)).
\]

We will denote the prior Gaussian process distribution on \( y \) with

\[
p(y | \theta) = \mathcal{GP}(y; \mu, K).
\]

The prior mean function is chosen as appropriate for the problem at hand (often a constant for convenience), and the prior covariance function is chosen to reflect any prior knowledge about the structure of the function of interest, for example periodicity or a specific amount of differentiability. A large number of off-the-shelf covariance functions are available, and they can be easily combined in various ways to produce customized solutions for a given problem (Rasmussen and Williams, 2006). We will discuss the choice and construction of covariance functions later.

The space \( \mathcal{X} \) can be any set of interest (we can always define trivially valid mean and covariance functions). In most cases, we will consider functions on \( \mathbb{R}^d \), but we will occasionally consider other spaces in later chapters, for example the Cartesian product of a finite set with \( \mathbb{R}^d \) (when performing time-series prediction of correlated variables using input from multiple discrete sensors), and the symmetric product of copies of the sphere \( S^2 \) (when optimizing functions defined on point sets on the earth’s surface). Note that a Gaussian “process” on a finite space \( \mathcal{X} \) is simply a standard multivariate Gaussian distribution.

By allowing continuous input spaces, the GP approach can handle arbitrarily located observations. In particular, when performing inference about temporal data, we have no need to discretize our observations into time steps, as required by some algorithms. This ability is beneficial when approaching problems involving data streams—the real world does not always provide data at a predictable rate.
5.2 THE POSTERIOR DISTRIBUTION

So far we have only discussed how to calculate the prior distribution

\[ p(y_\ast \mid X_\ast, \theta), \]

and not how to derive the posterior distribution over \( y \) given observations of the function. We will now discuss how to condition our prior on given data.

In practice, we rarely observe exact observations of the latent function \( y \). Instead, in real-world regression situations, our observations are typically corrupted by noise. Consider the measurement of our variable of interest at a point \( x \in \mathcal{X} \). We define the observation function

\[ z(x) = y(x) + \epsilon(x), \quad (5.2.1) \]

where \( \epsilon \) is a random variable defining the error in our measurement. A simple but often useful case to consider is the addition of zero-mean, fixed-variance independent Gaussian noise to our measurements:

\[ p(\epsilon(x) \mid x, \eta) \sim \mathcal{N}(0, \eta^2). \]

In some scenarios, exact observations of the latent function are available; in this case, we simply have \( z(x) = y(x) \) and \( \eta = 0 \). The observation-noise parameter \( \eta \), as a hyperparameter of our model, will be incorporated into the hyperparameter vector \( \theta \).

Having made this choice, the distribution of \( z \equiv z(X) \) given the values of the latent function \( y \equiv y(X) \) is therefore

\[ p(z \mid y, X, \theta) = \mathcal{N}(z; y, \eta^2 I). \quad (5.2.2) \]

Suppose now that we have noisy training observations of the latent function \( y \) of the form

\[ D = \{(x, z(x))\}_{x \in X} \subseteq (X, z). \]

Given these observations and a set of test inputs \( X_\ast \subseteq \mathcal{X} \), our goal is to calculate the posterior distribution of the latent function values at \( X_\ast \), namely

\[ p(y_\ast \mid X_\ast, D, \theta). \]

Fortunately, due to the properties of the Gaussian distribution, we may calculate this posterior analytically. Before we continue, we will define compact symbols for two frequently occurring expressions:

\[ K \equiv K(X, X; \theta), \]
\[ V \equiv K(X, X; \theta) + \eta^2 I = K + \eta^2 I. \]

We may calculate the prior distribution of our observations, \( p(z \mid X, \theta) \), by marginalizing out the unknown values of the latent function at \( X \):

\[
 p(z \mid X, \theta) = \int p(z \mid y, X, \theta) p(y \mid X, \theta) \, dy \\
 = \int \mathcal{N}(z; y, \eta^2 I) \mathcal{N}(y; \mu(X, \theta), K) \, dy \\
 = \mathcal{N}(z; \mu(X, \theta), K + \eta^2 I) \\
 = \mathcal{N}(z; \mu(X, \theta), V),
\]

41
where we applied (A.5) to evaluate the integral.

The prior joint distribution of \( z \) and \( y_\ast \) is therefore multivariate Gaussian:

\[
p(z, y_\ast | X, X_\ast, \theta) = \mathcal{N}\begin{pmatrix}
    z \\
    y_\ast
\end{pmatrix}
\begin{bmatrix}
    \mu(X; \theta) \\
    \mu(X_\ast; \theta)
\end{bmatrix}
\begin{bmatrix}
    V & K(X, X_\ast; \theta) \\
    K(X_\ast, X; \theta) & K(X_\ast, X_\ast; \theta)
\end{bmatrix}
\]

We may now simply condition this distribution on the known values \( z \) using (A.2) to derive the posterior:

\[
p(y_\ast | X_\ast, D, \theta) = \mathcal{N}(y_\ast; m_\ast(\theta), C_\ast(\theta)),
\]

where

\[
m_\ast(\theta) = \mu(X_\ast; \theta) + K(X_\ast, X; \theta)V^{-1}(z - \mu(X; \theta)),
\]

\[
C_\ast(\theta) = K(X_\ast, X_\ast; \theta) - K(X_\ast, X; \theta)V^{-1}K(X_\ast, X_\ast; \theta),
\]

are the posterior mean and covariance, respectively.

Notice that for any finite set of test points \( X_\ast \subseteq \mathcal{X} \), the posterior \( p(y_\ast | X_\ast, D, \theta) \) is multivariate Gaussian. Therefore, the posterior of \( p(y | \theta) \) conditioned on \( D \) meets the requirements of Definition 5.1.1 and is therefore a Gaussian process. It is easy to generalize the above to see that

\[
p(y | D, \theta) = \mathcal{GP}(y; m_{y|D}(x; \theta), C_{y|D}(x, x'; \theta)),
\]

where

\[
m_{y|D}(x; \theta) = \mu(x; \theta) + K(x, X; \theta)V^{-1}(z - \mu(X; \theta)), \quad (5.2.3)
\]

\[
C_{y|D}(x, x'; \theta) = K(x, x'; \theta) - K(x, X; \theta)V^{-1}K(X, x'; \theta), \quad (5.2.4)
\]

are the posterior mean and covariance functions, respectively.

### 5.3 Discussion

The theory related to Gaussian processes is extensive, and the reader is invited to investigate further in a reference such as (Rasmussen and Williams, 2006). Below we will briefly discuss various topics related to the remainder of this work.

**Sampling from a Gaussian process distribution**

If desired, we may easily (approximately) sample from a Gaussian process distribution. Of course, completely specifying a function on all of \( \mathcal{X} \) is impossible for infinite spaces, and therefore we cannot typically hope to sample complete functions. Instead, we select a finite set of test points \( X_\ast \subseteq \mathcal{X} \) as dense as desired. Now the problem of sampling from \( p(y | D, \theta) \) is reduced to sampling from the multivariate Gaussian distribution

\[
p(y_\ast | X_\ast, D, \theta) = \mathcal{N}(y_\ast; m_{y|D}(X_\ast; \theta), C_{y|D}(X_\ast, X_\ast; \theta)),
\]

a problem for which optimized algorithms are known and widely available (Rasmussen and Williams, 2006).
Marginal likelihood

Given data $D = (X, z)$, we may analytically calculate the marginal likelihood of our GP model, $p(z | X, \theta)$. In fact, we have already performed the required work in Section 5.2:

$$p(z | X, \theta) = N(z; \mu(X; \theta), V).$$ (5.3.1)

The marginal likelihood is very important in Gaussian process inference (and Bayesian inference in general), because it represents the probability of our observations given our model; in other words, it reflects how well our model fits the observed data.

Covariance functions

Numerous covariance functions have been introduced for various applications. In fact, the Bayesian treatment of linear regression we presented in Section 2.3 can be understood as a special case of a Gaussian process, where the covariance function is an appropriately chosen dot-product covariance (Rasmussen and Williams, 2006).

Initially, we will primarily work with stationary covariance functions. Stationary covariance functions are not appropriate for concept-drift situations; however, in the next chapter we will describe how they can be successfully modified to model various changepoints that might occur in nonstationary data streams. A common goal of stationary covariance function is to encode the prior belief that “close” points in the input space $\mathcal{X}$ will have highly correlated function values, and that the strength of this correlation will decrease as points become further separated. For particular problems and applications, a frequent task will be finding or constructing a metric on $\mathcal{X}$ that will appropriately encode this notion of “closeness.” For now we will consider the flexible and often useful Mahalanobis distance on $\mathbb{R}^d$, defined as

$$d_M(x, x'; A) \triangleq \sqrt{\sum_{i=1}^d (x_i - x'_i)^2 A^{-1}(x_i - x'_i)^T},$$ (5.3.2)

where the scaling matrix $A$ is a symmetric, positive-definite matrix. Two common choices for $A$ are the isotropic scaling matrix

$$A \triangleq \sigma^2 I$$ (5.3.3)

and the anisotropic diagonal scaling matrix

$$A \triangleq \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2).$$

In these cases, the Mahalanobis distance becomes the scaled Euclidean distance

$$d_M(x, x'; A) = \sqrt{\sum_{i=1}^d \frac{(x_i - x'_i)^2}{\sigma_i^2}}.$$

We will consider the parametrized family of stationary covariance functions

$$K(x, x'; \lambda, \sigma) \triangleq \lambda^2 k(d_M(x, x'; \text{diag} \sigma^2)),$$ (5.3.4)

A covariance function $K(x, x')$ is said to be stationary when it is a function of $x - x'$ and therefore translation invariant.
for an appropriate function \( \kappa \), which is typically a monotonically decreasing positive function of its input. The hyperparameters \( \lambda \) and \( \sigma \) serve as the characteristic output scale and characteristic input scales of our covariance, respectively. The output scale defines the overall process variance of the values of the function \( y \). The input scales act to scale the inputs such that the covariance reacts appropriately with respect to changes in each dimension. We list a few examples of stationary covariance functions below.

**SQUARED EXPONENTIAL COVARIANCE FUNCTION** The prototypical covariance function is the squared exponential covariance function, given by

\[
K_{SE}(x, x'; \lambda, \sigma) = \exp\left(-\frac{d_M^2}{2}\right).
\]

The squared exponential covariance function is infinitely differentiable, so a Gaussian process incorporating it is infinitely mean-square differentiable; see (Rasmussen and Williams, 2006) for more information. \( K_{SE} \) therefore models \( y \) as an element of \( \mathcal{C}^\infty \). In many real-world applications, this smoothness assumption might be unreasonable (Rasmussen and Williams, 2006); however, the squared exponential covariance function is nonetheless widely used.

**MATÉRN COVARIANCE FUNCTION** When the smoothness assumption of the squared exponential covariance function is unreasonable, the Matérn covariance function provides a useful alternative (Stein, 1999). The function is given by

\[
K_M(x, x'; \lambda, \sigma, \nu) = \lambda^2 \left( \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu d_M^2})^\nu K_{\nu}(\sqrt{2\nu d_M^2}) \right),
\]

where \( \Gamma \) is the standard mathematical Gamma function and \( K_{\nu} \) is the modified Bessel function of the second kind with parameter \( \nu \). The \( \nu \) parameter controls the smoothness of the process—for an integer \( k \in \mathbb{N} \), the corresponding process is \( k \)-times mean-square differentiable if and only if \( \nu > k \) (Stein, 1999). Therefore, the Matérn covariance function models \( y \) as an element of \( \mathcal{C}^{[\nu]-1} \).

For half-integral \( \nu \), the Matérn covariance function has a simple form. In particular, for \( \nu = \frac{1}{2} \) we have

\[
K_M(x, x'; \lambda, \sigma, \nu = \frac{1}{2}) = \lambda^2 (1 + \sqrt{3} d_M) \exp\left(-\sqrt{3} d_M \right),
\]

which models continuous functions with noncontinuous derivative.

**RATIONAL QUADRATIC COVARIANCE FUNCTION** The rational quadratic covariance function is widely used in geostatistics when modeling spatial fields (see, for example, (Gundogdu and Gneey, 2007)). The function is given by

\[
K_{RQ}(x, x'; \lambda, \sigma, \alpha) = \lambda^2 \left( 1 + \frac{d_M^2}{2\alpha} \right)^{-\alpha}.
\]

The (positive) \( \alpha \) parameter controls the long-scale correlation of function values—smaller values of \( \alpha \) result in more long-scale smoothness. As \( \alpha \) approaches infinity, the
rational quadratic covariance converges to the squared exponential covariance function with equivalent parameters:

$$\lim_{\alpha \to \infty} K_{RQ}(x, x'; \lambda, \sigma, \alpha) = K_{SE}(x, x'; \lambda, \sigma).$$

The rational quadratic covariance function results in an infinitely mean-square differentiable process for all values of $\alpha$ (Rasmussen and Williams, 2006).

When appropriately parametrized, the rational quadratic covariance function can be seen as an infinite mixture of squared exponential covariance functions with different input scales (Rasmussen and Williams, 2006).

Combining and altering covariance functions

Covariance functions can be manipulated in various useful ways (Rasmussen and Williams, 2006). For example, both the sum and pointwise product of two arbitrary covariance functions are valid covariance functions; this fact can be applied to model a function that is believed to be the sum or pointwise product of two distinct components with different behavior.

Consider a space $X$ that can be decomposed into the Cartesian product $X_1 \times X_2$. Given arbitrary covariance functions on these smaller spaces,

$$K_1: X_1 \times X_1 \to \mathbb{R},$$
$$K_2: X_2 \times X_2 \to \mathbb{R},$$

the tensor product

$$K(x, x') = K_1(x_1, x_1')K_2(x_2, x_2')$$

is a valid covariance function on $X \times X$. This construction will often be useful.

Suppose we wish to place a covariance function on a space $X$. Rather than apply a covariance on $X$ directly, it is sometimes useful to first map the points to a different space that has desirable properties and apply a covariance function there instead. Suppose that $t: X \to \mathcal{T}$ is an arbitrary function into some space $\mathcal{T}$. We can now choose a covariance $K: \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ and apply it to the transformed points:

$$K(x, x') = K(t(x), t(x'));$$

this is also a valid covariance on $X \times X$. A useful application of this construction is to create a covariance function for modeling a periodic function. Let $X = \mathbb{R}$ and consider the map $t: X \to S^1$ onto the unit circle defined by

$$t(x; \gamma) \triangleq x \mapsto \left[ \cos \left( \frac{2\pi x}{\gamma} \right), \sin \left( \frac{2\pi x}{\gamma} \right) \right]^T.$$

We apply the squared exponential covariance function $K_{SE}$ (5.3.5) in $\mathcal{T} = S^1$ using the isotropic scaling matrix (5.3.3):

$$K_p(x, x'; \lambda, \sigma, \gamma) \triangleq K_{SE}(t(x; \gamma), t(x'; \gamma); \lambda, \sigma)$$

$$= \lambda^2 \exp \left( -\frac{2}{\sigma^2} \sin^2 \left( \frac{\pi|y - y'|}{\gamma} \right) \right).$$

(5.3.9)
where the result above may be derived using trigonometric identities. In the periodic covariance function $K_P$, the parameter $\lambda$ serves as the amplitude, the parameter $\gamma$ serves as the period, and the input scale $\sigma$ parametrizes the roughness of the periodic function.

Finally, another useful property is the following. Suppose that $y : \mathcal{X} \to \mathbb{R}$ has a Gaussian process distribution

$$p(y \mid \theta) = \mathcal{GP}(y; \mu, K),$$

and that $a, b : \mathcal{X} \to \mathbb{R}$ are two arbitrary given functions. Then the affine transformation $y'(x) = a(x)y(x) + b(x)$ also has a Gaussian process distribution, given by

$$p(y' \mid \theta) = \mathcal{GP}(y'; \mu', K'),$$

where

$$\mu'(x; \theta) = a(x)\mu(x; \theta) + b(x), \quad (5.3.10)$$

$$K'(x, x'; \theta) = a(x)K(x, x'; \theta)a(x'), \quad (5.3.11)$$

d this result is analogous to (5.6).

**Computational considerations**

For any proposed machine-learning algorithm, we must consider the associated computational cost. For Gaussian processes, this analysis is straightforward.

Suppose that we have $N$ total observations in our data $\mathcal{D}$. When calculating the posterior distribution $p(y \mid \mathcal{D}, \theta)$, the most computationally demanding step of the inference is the inversion of the matrix $V$. This calculation is usually performed using the Cholesky decomposition, which offers the best numerical stability and takes time $O(N^3)$. Once the Cholesky decomposition has been found, the posterior mean and variance at a given test point can be found in time $O(N)$. 

In a data-stream or time-series prediction setting, alterations to our data $\mathcal{D}$ usually take the form of the addition of a small number of new observations—previously recorded observations do not typically change. In this case, we do not need to completely recalculate the Cholesky decomposition of $V$ with each new datum. Instead, we can incorporate each arriving observation using a rank-one update to the Cholesky decomposition of this matrix (Seeger, 2008). This update takes time $O(N^2)$. In a similar manner, if an observation is removed from $\mathcal{D}$, we may perform a rank-one “downdate” to the Cholesky decomposition in time $O(N^2)$. These procedures allow us to perform online time-series prediction from $N$ observations in total time approximately equal to that required by performing the equivalent offline regression once.

Finally, we note that the Cholesky decomposition of the observation covariance matrix can be efficiently performed on a modern graphical processing unit (GPU). The nature of the Cholesky decomposition algorithm allows for an effective implementation that harnesses the highly parallel nature of GPUs. This implementation is made even simpler using new application–programmer interfaces provided by contemporary general-purpose GPU libraries such as the Compute Unified Device Architecture (CUDA) from NVIDIA (NVIDIA CORPORATION, 2010) and the cross-platform Open Computing Language (OpenCL) from Apple Inc. and the Khronos group (Khronos OPENCL...
5.4. Example

We will briefly demonstrate a concrete example of standard Gaussian process regression. A similar example was explored by Rasmussen and Williams (2006).

We will consider a one-dimensional function \( y : \mathbb{R} \rightarrow \mathbb{R} \) and take a simple Gaussian process prior over \( y \),

\[
p(y | \theta) \sim \mathcal{GP}(y; \mu, K).
\]

For the mean function \( \mu \), we choose the trivial constant function \( \mu(x) \equiv 0 \), and for the covariance function \( K \), we choose the squared exponential covariance function \( K_{\text{SE}}(5.3.5) \) with \( (\lambda, \sigma) \equiv (1, 1) \). We also select the independent Gaussian-noise observation model given in (5.2.2) with \( \eta \equiv 1/5 \).

Figure 5.1 shows our prior distribution, along with five random draws from the prior. We now condition the prior distribution on the five observations shown in Figure 5.2, which also shows the resulting posterior distribution

\[
p(y | D, \theta)
\]

and five random draws from the posterior. Notice that our uncertainty about \( y \) decreases with increasing proximity to our observations. In contrast, moving away from our
observations, the posterior becomes increasingly similar to the prior; by the point $x = 10$, any difference is trivial. This behavior agrees with our Bayesian intuition—we have considerable evidence about the behavior of $y$ near our observations and little evidence about its behavior far from our observations. This intuition is expressed formally by our choice of covariance function. Finally, notice that our uncertainty never completely decreases to zero due to our choice of observation model. The uncertainty does behave appropriately near our observations, however. For example, it becomes considerably smaller near $x = 6.5$, where we have two very close observations, than it does near $x = 2.5$, where we have only one. The two nearby observations effectively serve to solidify our belief about the value of $y$ in the surrounding region.

We now consider how our chosen Gaussian process model changes with respect to its hyperparameters, $\theta$. Figure 5.3(a) shows the GP posterior conditioned on twenty observations using the same model and hyperparameter values described above. The fit appears to be quite good. Figure 5.3(b) shows the posterior for the same model using different hyperparameters: a shorter input scale, a larger output scale, and a smaller noise variance. The fit appears to be much worse, and the marginal likelihood of the model is correspondingly much smaller—our data are approximately $2.64 \times 10^4$ times less likely under this model than the one in Figure 5.3(a). It is clear that the selection of hyperparameter values in Gaussian process models can be very important.

In the next section we will consider the problem of effectively dealing with our hyperparameters using the marginal likelihood function as our guide.
5.4. Example

Figure 5.3: The posterior Gaussian process mean with pointwise ±2 standard-deviation bounds for the example situation, conditioned on the twenty observations displayed, using the indicated hyperparameters. The marginal log-likelihood of each model is also given.

(a) $\theta = (\lambda, \sigma, \eta) = (1, 1, 1/5)$, $\log p(z | X, \theta) = -14.85$

(b) $\theta = (\lambda, \sigma, \eta) = (1/3, 1/3, 1/3)$, $\log p(z | X, \theta) = -25.03$
5. GAUSSIAN PROCESSES

5.5 HYPERPARAMETER MANAGEMENT

So far, we have only considered how to proceed with Gaussian process inference when the values of the hyperparameters are given. To perform inference in full Bayesian generality, we might want to instead specify a prior distribution \( p(\theta) \) for our hyperparameters and marginalize out our uncertainty in \( \theta \) when making predictions by calculating

\[
p(y_* \mid x_*, D) = \frac{\int p(y_* \mid x_*, D, \theta) p(\theta \mid X, \theta) \, d\theta}{\int p(z \mid X, \theta) p(\theta) \, d\theta}.
\] (5.5.1)

Unfortunately, the required integrals in (5.5.1) cannot be evaluated analytically due to the nature of the the Gaussian likelihood function \( p(z \mid X, \theta) \) (5.3.1). We will consider this problem in this section.

Occasionally, the value of certain hyperparameters can be fixed with significant prior knowledge; for example, we can be fairly certain that the period of the Earth’s rotation with respect to the Sun is approximately 86,400 seconds. In this case, we can safely simply fix an associated hyperparameter specifying this period to the appropriate value. This is equivalent to specifying a Dirac delta distribution for the prior of the hyperparameters fixed in this manner, rendering the integral in (5.5.1) trivial.

Another common situation occurs when the value of our hyperparameters are unknown a priori but assumed to be stationary (that is, their values do not vary in either time or space). When sufficient training observations are available, we can proceed in a similar manner even without strong prior beliefs. The assumption we make is that with increasing amounts of data, the likelihood will become increasingly peaked and localized, with a single mode that will occur at the proper values for our hyperparameters. Given our stationarity assumption, this is not unreasonable. In probabilistic terms, we assume that given sufficient data \( D \), our posterior belief about \( \theta \),

\[
p(\theta \mid D) = \frac{p(z \mid X, \theta) p(\theta)}{\int p(z \mid X, \theta) p(\theta) \, d\theta},
\]

will be dominated by the marginal likelihood:

\[
p(\theta \mid D) = Z^{-1} p(z \mid X, \theta),
\]

where \( Z^{-1} \) is a normalization term. With these assumptions, we proceed by simply locating this peak and setting our hyperparameters accordingly. Fortunately, finding a local maximum for the likelihood is facilitated by an analytic expression for its gradient. The marginal log-likelihood is

\[
\log p(z \mid X, \theta) = -\frac{z^T V^{-1} z + \log \det V + N \log 2\pi}{2},
\] (5.5.2)

where \( N \) is the number of observations. From this, we may calculate the derivative with respect to the \( i \)th hyperparameter, \( \theta_i \):

\[
\frac{\partial \log p(z \mid X, \theta)}{\partial \theta_i} = \frac{1}{2} \left( z^T V^{-1} \frac{\partial V}{\partial \theta_i} V^{-1} z - \text{tr} \left( V^{-1} \frac{\partial V}{\partial \theta_i} \right) \right).
\]

As discussed in (Rasmussen and Williams, 2006), this gradient can be calculated without prohibitive computational cost. In practice, gradient-based local optimizers
can usually maximize (5.5.2) with respect to $\theta$ reasonably quickly, and the location of this maximum can then be used in our further inference. The procedure of setting hyperparameter values in this manner is called *maximum-likelihood-II estimation*. The “II” refers to the fact that the optimization occurs at the second level of inference; that is, we are optimizing the marginal likelihood of our model with respect to parameters of the prior distribution, rather than optimizing the underlying data likelihood with respect to model parameters. In standard Gaussian process inference, the complete marginalization of all model parameters\(^7\) is analytic and is performed automatically.

In the situation described above, we are again implicitly selecting a Dirac delta distribution for the prior over the hyperparameters set in this manner. However, in this case, we choose their fixed values based on maximizing the posterior marginal likelihood of our Gaussian process model given many observations, rather than setting them *a priori* with domain-specific knowledge. Essentially, we rely on strong posterior beliefs, rather than strong prior beliefs.

In many cases, however, we will have neither strong prior nor strong posterior beliefs about the value of a hyperparameter. This will especially be true when considering problems involving concept drift, where the appropriate values of hyperparameters might change over time. If the rate of change is significant with respect to the frequency of observed data, we will never have enough appropriate training data to form a very certain posterior belief about the value of our hyperparameters at a given point in time. In the next chapter, for example, we will consider the problem of performing time-series prediction in the presence of sudden changes to the data. Our approach will be to construct Gaussian process covariance functions to model these changes, which will contain changepoint locations as hyperparameters. When performing online inference, we will often have no idea when a changepoint might occur before it does; the best we can hope for is to quickly recognize and adapt to them. For this reason, we will want to place a broad prior over changepoint location and, when making predictions, accordingly marginalize out our uncertainty in the changepoint location hyperparameter.

In the next two subsections, we will discuss the Bayesian Monte Carlo approach to numerical integration and how we can apply the technique for approximating the integral in (5.5.1) without resorting to trivial approximations.

---

Bayesian Monte Carlo for hyperparameter marginalization

Before we describe the approximation of the somewhat complicated integral in (5.5.1), we will first introduce the Bayesian Monte Carlo method in generality. We will closely follow the discussion in (Rasmussen and Ghahramani, 2003).

**Bayesian Monte Carlo** Consider the problem of evaluating the expectation

$$
\mathbb{E}_x[f(x)] := \int f(x)p(x) \, dx, \quad (5.5.3)
$$

where $f$ is an arbitrary function of $x$ and $p$ is a probability density function. If this integral cannot be performed analytically, we are forced to perform some form of

\[^7\]Although we don’t typically describe them as such, the parameters of a Gaussian process model are the unknown values of the latent function $y$ at our observed inputs $X$.\]
numerical integration (also called quadrature). Numerical integration entails evaluating the integrand in (5.5.3) at a set of sample points

\[ S = [s_1, s_2, \ldots], \]

then combining these observed values into a final estimate of the integral.

The simplest and most widely used numerical procedure for integrals of this type is the classical Monte Carlo algorithm, which generates \( S \) by drawing random samples from \( p(x) \) and approximates the integral in (5.5.3) with

\[ E_x[f(x)] = \frac{1}{|S|} \sum_{s \in S} f(s). \]

This approximation is guaranteed to converge to the right answer with an increasing number of samples (Rasmussen and Ghahramani, 2003). If the distribution \( p(x) \) is difficult to sample from, an importance-reweighting trick can be applied. A simpler sampling distribution \( q(x) \) is chosen, and the expectation of the function

\[ g(x) = \frac{f(x)p(x)}{q(x)} \]

is taken instead with respect to the new sampling distribution \( q \). Of course,

\[ g(x) q(x) = f(x)p(x), \]

so we evaluate the same integral while avoiding sampling from \( p \).

Although this procedure is guaranteed to converge to the correct answer, O’Hagan (1987) and Rasmussen and Ghahramani (2003) have pointed out that classical Monte Carlo is inefficient because the information contained in the location of the chosen samples \( S \) is completely ignored. When two samples are very close to each other (or in the extreme case, as discussed at length by Rasmussen and Ghahramani (2003), they lie on completely top of each other), for well-behaved functions, we should expect that the associated values of \( f \) will be highly correlated. That is, when two samples lie close to each other in space, there is some amount of redundancy in the information that they provide about the value of \( E_x[f(x)] \). In standard Monte Carlo integration, however, this nuance is not accounted for—all samples are blindly weighted equally. In many cases that we will consider later, the cost of evaluating the integrand will be considerable, and we will want to use as much available information as possible when inferring the value of our integral. For this reason, we seek a more sophisticated numerical-integration technique.

Bayesian Monte Carlo (BMC) (Rasmussen and Ghahramani, 2003), also called Bayes–Hermite quadrature (O’Hagan, 1987), is a Bayesian method for numerical integration that attempts to address these shortcomings of standard Monte Carlo integration. In BMC, we treat the unknown value of the integral in (5.5.3) as a random variable and perform standard Bayesian inference about this value.

To account for the spatial correlation between the values of \( f \) mentioned above, we place a Gaussian process prior over \( f \). As usual, we construct this model using our prior beliefs about the function, selecting appropriate mean and covariance functions. Now, given data of the form

\[ D = \{(s, f(s))\}_{s \in S} \triangleq (S, f), \]
we may calculate the posterior distribution of $f$ given $D$ as described in Section 5.2:

$$p(f \mid D, \theta) = \mathcal{GP}(f; m_{f|D}(x; \theta), C_{f|D}(x, x'; \theta)),$$

where

$$m_{f|D}(x; \theta) \triangleq K(x, S; \theta)K^{-1}f,$$

$$C_{f|D}(x, x'; \theta) \triangleq K(x, x'; \theta) - K(x, S; \theta)K^{-1}K(S, x'; \theta).$$

To ease exposition, for the remainder of this section, we will drop the explicit parametrization of the model by $\theta$. Notice that these expressions involve the latent covariance $\mathbf{K}$ where the observation covariance $\mathbf{V}$ is usually found. This is because we typically assume that our evaluations of $f$ are noise free.

Because integration is a linear operator, the value of the integral in (5.5.3) is jointly Gaussian distributed with $p(f \mid D)$, and its distribution is therefore completely described by its mean and variance (Rasmussen and Ghahramani, 2003). We calculate its mean directly with

$$\mathbb{E}_{f|D}[\mathbb{E}_x[f(x)]] = \int \mathbb{E}_x[f(x)] p(f \mid D) df$$

$$= \int \int f(x)p(x)p(f \mid D) dx df$$

$$= \int \left( \int f(x)p(f \mid D) df \right) p(x) dx$$

$$= \int m_{f|D}(x)p(x) dx. \quad (5.5.4)$$

By choosing convenient forms for the prior $p(x)$ and the covariance function $K$ over $f$, we can perform the integral in (5.5.4) analytically. Specifically, we choose a Gaussian prior for $p(x)$:

$$p(x) \triangleq p(x \mid \mu, \Sigma) \triangleq \mathcal{N}(x; \mu, \Sigma),$$

and a Gaussian covariance function for $f$:

$$K(x, x') \triangleq K(x, x'; \lambda, \mathbf{P}) \triangleq \lambda^2 \mathcal{N}(x - x'; 0, \mathbf{P})$$

$$= \lambda^2 \mathcal{N}(x; x', \mathbf{P}).$$

Now our integral becomes

$$\mathbb{E}_{f|D}[\mathbb{E}_x[f(x)]] = \int m_{f|D}(x)p(x) dx$$

$$= \int (K(x, S)K^{-1}f)p(x) dx.$$

To proceed, we transpose the (scalar) quantity $(K(x, S)K^{-1}f)$, which gives

$$\mathbb{E}_{f|D}[\mathbb{E}_x[f(x)]] = \int (K(x, S)K^{-1}f)^T p(x) dx$$

$$= \int f^T K^{-1}K(S, x)p(x) dx$$

$$= f^T K^{-1} \int K(S, x)p(x) dx.$$
We now define the convenient function

\[ \omega(s) = \int K(s, x) p(x) \, dx = \int \lambda^2 \mathcal{N}(s; x, \mathbf{P}) \mathcal{N}(x; \mu, \Sigma) \, dx = \lambda^2 \mathcal{N}(s; \mu, \mathbf{P} + \Sigma). \]  

(5.5.5)

Finally, we can write

\[ \mathbb{E}_{f|\mathcal{D}}[\mathbb{E}_s[f(x)]] = \omega^T K^{-1} f, \]

(5.5.6)

where \( \omega \equiv \omega(S) \).

Notice that the \( \lambda^2 \) term in \( \omega^T \) cancels out the \( \lambda^{-2} \) term in \( K \), and therefore our mean prediction of the integral does not depend on the chosen output scale for \( K \).

Above we only calculate the mean of our belief over \( \mathbb{E}_s[f(x)] \), which for our purposes will be all that we require in the remainder of this work. The variance of \( \mathbb{E}_s[f(x)] \) can also be derived; the reader is referred to (RASMUSSEN and GHAIHARAMANI, 2003). This variance can be used, for example, to drive a Bayesian decision-theoretical mechanism for optimally selecting where to place our samples—by minimizing the variance as a function of the sample locations, we can select the most informative possible samples for our inference. This idea was explored by MINKA (2000) and RASMUSSEN and GHAIHARAMANI (2003).

Note that the form of (5.5.6) is a linear combination of the function values \( f(S) \), just as in classical Monte Carlo. The important difference, however, is that our weights \( \omega(S)^T K(S, S)^{-1} \) now appropriately account for the information in the spatial locations of the samples \( S \).

The BMC procedure was evaluated by RASMUSSEN and GHAIHARAMANI (2003), who performed thorough empirical comparisons of BMC with both classical Monte Carlo and annealed importance sampling (AIS) (NEAL, 2001). BMC almost uniformly outperformed both competing methods for estimating several integrals. The authors suggested that the increased performance of BMC could be ascribed to the fact that, in addition to using information pertaining to the location of samples, the BMC procedure also analytically marginalizes out the unknown value of \( f \) away from the observed samples. This ability to interpolate between samples becomes important when the number of available samples is small.

**Bayesian Monte Carlo for Hyperparameter Marginalization**  
We now return to the problem of approximating the integral

\[ p(y_* \mid x_*, \mathcal{D}) = \frac{\int p(y_* \mid x_*, \mathcal{D}, \theta) p(z \mid X, \theta) p(\theta) \, d\theta}{\int p(z \mid X, \theta) p(\theta) \, d\theta}. \]

We will apply the Bayesian Monte Carlo procedure described above to this problem and ultimately derive a simple analytic approximation for the mean of our estimate. We will closely follow the discussion first presented in (OSBORNE, et al., 2008) and later expanded in (GARNETT, et al., 2010).

For notational brevity, we define the condensed prediction function

\[ q(y_*; \theta) = p(y_* \mid x_*, \mathcal{D}, \theta) \]
and likelihood function
\[ r(\theta) \triangleq p(z \mid X, \theta). \]

The integral of interest can now be written more compactly:
\[
p(y_* \mid x_*, \mathcal{D}) = \frac{\int q(y_*, \theta) r(\theta) p(\theta) \, d\theta}{\int r(\theta) p(\theta) \, d\theta}.
\]

Suppose that we have evaluated the \( q \) and \( r \) functions on a set of hyperparameter samples
\[
\Psi \triangleq [\psi_1, \psi_2, \ldots],
\]
and that we have \( k \) total hyperparameters to marginalize, so that each hyperparameter sample may be written
\[
\psi_i = [\psi_{i,1}, \psi_{i,2}, \ldots, \psi_{i,k}]^T.
\]

Notice that for a given point \( \theta \) in hyperparameter space, calculating the prediction \( p(y_* \mid x_*, \mathcal{D}, \theta) \) effectively evaluates the \( q \) function along the entire infinite “slice” \( \mathbb{R} \times \theta \) simultaneously.

As before, we will choose a convenient prior \( p(\theta) \) for our hyperparameters. We select the \( k \)-dimensional independent product prior
\[
p(\theta) \triangleq \prod_{i=1}^{k} p(\theta_i), \quad (5.5.7)
\]
where the component priors are either Gaussian:
\[
p(\theta_i) \triangleq p(\theta_i \mid \mu_i, \nu_i) \triangleq \mathcal{N}(\theta_i; \mu_i, \nu_i^2), \quad (5.5.8)
\]
or uniform in a given interval:
\[
p(\theta_i) \triangleq p(\theta_i \mid l_i, u_i) \triangleq \mathcal{U}(\theta_i; [l_i, u_i]). \quad (5.5.9)
\]

These priors suffice for a wide range of hyperparameters. For a hyperparameter known to be strictly positive, we will instead place an appropriate prior over its logarithm.

We now place zero-mean Gaussian process priors over the unknown functions \( q \) and \( r \). For the covariance function over \( r \) we choose the independent product Gaussian covariance
\[
K(\theta, \theta') \triangleq \prod_{i=1}^{k} K_i(\theta_i, \theta'_i), \quad (5.5.10)
\]
where
\[
K_i(\theta, \theta') \triangleq K_i(\theta_i, \theta'_i; \lambda_i, \sigma_i) \triangleq \lambda_i^2 \mathcal{N}(\theta_i - \theta'_i; 0, \sigma_i^2)
\]
\[
= \lambda_i^2 \mathcal{N}(\theta_i; \theta'_i, \sigma_i^2). \quad (5.5.11)
\]

For the covariance function over \( q \), we select the trivial product covariance
\[
K_q((y, \theta), (y', \theta')) \triangleq K(y, y')K(\theta, \theta'), \quad (5.5.12)
\]
where \( K(\theta, \theta') \) is as defined in (5.5.10) and
\[
K(y, y') \triangleq \delta(y, y')
\]
is the Kronecker delta function:

\[
\delta(y, y') = \begin{cases} 
1 & y = y' \\
0 & y \neq y'.
\end{cases}
\] (5.5.13)

As before, we now make the two sets of observations

\[
Q \triangleq \left\{(\mathbb{R} \times \Psi, q(\mathbb{R}, \Psi))\right\}_{q \in \Psi} \triangleq (\mathbb{R} \times \Psi, q)
\]

and

\[
R \triangleq \left\{(\psi, r(\psi))\right\}_{\psi \in \Psi} \triangleq (\Psi, r).
\]

and use these observations to train our GP models over \(q\) and \(r\). The posterior equations for \(r\) are completely ordinary; however, we pause briefly to consider the posterior mean function for the GP over \(q\) conditioned on \(Q\), which has a convenient form. At first glance, the fact that \(Q\) contains uncountably many observations of \(q\) might seem to present a potentially insurmountable problem. Due to our choice of covariance, however, for a given point \((y_*, \theta)\), the covariance

\[
K_q((y_*, \theta), \mathbb{R} \times \Psi)
\]

will only be nonzero for a finite subset of these observations—specifically, the points \((y_*, \Psi)\). We have

\[
m_q|Q(y_*, \theta) = K_q((y_*, \theta), \mathbb{R} \times \Psi) K_q(\mathbb{R} \times \Psi, \mathbb{R} \times \Psi)^{-1} q = K(\theta, \Psi)K(\Psi, \Psi)^{-1}q(y_*) ,
\] (5.5.14)

where the \(K\) in (5.5.14) is the covariance over \(r\) given in (5.5.10) and

\[
q(y_*) \triangleq q(y_*, \Psi).
\]

Therefore our mean estimate of the prediction equation \(q(y_*, \theta)\) is simply a linear combination of our prediction equations, with weights dependent only on the location of \(\theta\) and the hyperparameter samples \(\Psi\).

Our mean estimate of \(p(y_* | x_*, \mathcal{D})\) conditioned on the observations \(Q\) and \(R\) now becomes

\[
E_{q|Q,R}[p(y_* | x_*, \mathcal{D})] = \int \int \frac{q(y_*, \theta) r(\theta) p(\theta) \, d\theta}{\int r(\theta) p(\theta) \, d\theta} p(q) p(r | \mathcal{R}) \, dq \, dr.
\] (5.5.15)

For notational brevity, we will drop the subscript on the expectation in (5.5.15) and also define the function

\[
Z(r) \triangleq \int r(\theta) p(\theta) \, d\theta.
\]
We now have
\[
\mathbb{E}[p(y_*, x_*, D)] = \int\int \frac{q_{y*, \theta} r(\theta) p(\theta) d\theta}{r(\theta) p(\theta) d\theta} p(q | Q) p(r | R) dq dr = \int\int Z(r)^{-1} q_{y*, \theta} r(\theta) p(\theta) p(q | Q) p(r | R) d\theta dq dr = \int\left( \int q_{y*, \theta} p(q | Q) dq \right) Z(r)^{-1} r(\theta) p(\theta) p(r | R) d\theta dr = \int m_{q|Q}(y_*, \theta) Z(r)^{-1} r(\theta) p(\theta) p(r | R) d\theta dr. \tag{5.5.16}
\]

Unfortunately, at this point, the integral in (5.5.16) remains nonanalytic due to the nontrivial dependence on the likelihood function \(r\) in the \(Z(r)\) quotient term. We follow the path outlined by Osborne, et al. (2008) and Garnett, et al. (2010) and use a maximum a posteriori (MAP) approximation, whereby we replace the posterior distribution \(p(r | R)\) with a Dirac delta distribution at its maximum point. Because a Gaussian process is symmetric around its mean, the MAP point is simply its mean. Therefore we replace \(p(r | R)\) with \(\delta(r - m_{r|R})\) and continue:
\[
\mathbb{E}[p(y_*, x_*, D)] = \int\int m_{q|Q}(y_*, \theta) Z(r)^{-1} r(\theta) p(\theta) p(r | R) d\theta dr = \int m_{q|Q}(y_*, \theta) Z(m_{r|R})^{-1} m_{r|R}(\theta) p(\theta) d\theta \approx \int (K(\theta, \Psi)K^{-1} q_{y*}(\cdot))(K(\theta, \Psi)K^{-1} r) p(\theta) d\theta.
\]

We transpose the scalar quantity \((K(\theta, \Psi)K^{-1} q_{y*}(\cdot))\) and pull constants out of the integral:
\[
\mathbb{E}[p(y_*, x_*, D)] = \int (K(\theta, \Psi)K^{-1} q_{y*}(\cdot))^\top (K(\theta, \Psi)K^{-1} r) p(\theta) d\theta = q_{y*}(\cdot)^\top K^{-1} \left( \int K(\Psi, \theta) K(\theta, \Psi) p(\theta) d\theta \right) K^{-1} r.
\]

We define the convenient function
\[
\Omega(\Psi, \Psi') = \int K(\Psi, \theta) K(\theta, \Psi') p(\theta) d\theta \tag{5.5.17}
\]
\[
= \int \cdots \prod_{i=1}^k (K_i(\psi_i, \theta_i) K_i(\theta_i, \psi_i') p(\theta_i)) d\theta_1 d\theta_2 \ldots d\theta_k = \prod_{i=1}^k \int K_i(\psi_i, \theta_i) K_i(\theta_i, \psi_i') p(\theta_i) d\theta_i. \tag{5.5.18}
\]

Each term of the product in (5.5.18) may be resolved analytically for the uniform and Gaussian hyperparameter priors \(p(\theta)\). When \(p(\theta_i)\) is the Gaussian in (5.5.8), we have
\[
\int K_i(\psi_i, \theta_i) K_i(\theta_i, \psi_i') p(\theta_i) d\theta_i = \lambda_i^2 N \left( \begin{bmatrix} \psi_i \\ \psi_i' \end{bmatrix}, \begin{bmatrix} m_i \\ m_i \\ v_i^2 + \sigma_i^2 \\ v_i^2 + \sigma_i^2 \end{bmatrix} \right).
\]

57
and when \( p(\theta_i) \) is the uniform in (5.5.9), we have

\[
\int K_i(\psi_i, \theta_i)K_i(\theta_i, \psi_i) p(\theta_i) d\theta_i = \\
\lambda_i^2 \mathcal{N}(\psi_i; \psi'_i, 2\sigma^2) \left( \Phi \left( \frac{\mu_i + \psi'_i}{2}; \Phi \left( \frac{\mu_i - \psi'_i}{2} \right) \right) - \Phi \left( \frac{\mu_i + \psi'_i}{2}; \Phi \left( \frac{\mu_i - \psi'_i}{2} \right) \right) \right),
\]

where \( \Phi \) denotes the cumulative distribution function of the Gaussian distribution with corresponding parameters:

\[
\Phi(x; \mu, \sigma^2) = \int_{-\infty}^{x} \mathcal{N}(x; \mu, \sigma^2) dx. \tag{5.5.19}
\]

After these calculations, we may write down our final estimate of the integral in (5.5.1):

\[
\mathbb{E}[p(y_* | x_*, D)] \propto q(y_*)^T K^{-1} \Omega K^{-1} r,
\]

where \( \Omega = \Omega(\Psi, \Psi) \). We see that our final mean prediction of \( p(y_* | x_*, D) \) is approximately proportional to a linear combination of our prediction samples \( q(y_*) \), with weights proportional to

\[
w \propto K^{-1} \Omega K^{-1} r, \tag{5.5.20}
\]

where \( I \) is a vector of all ones of the appropriate size. Because we know that our integral of interest, \( p(y_* | x_*, D) \), must have unit integral and that our samples \( q \) are probability density functions with unit integral, we simply retain these conveniently normalized weights. Our final estimate becomes

\[
\mathbb{E}[p(y_* | x_*, D)] = q(y_*)^T w
\]

\[
= \sum_{\psi \in \Psi} w_{\psi} p(y_* | x_*, D, \psi)
\]

\[
= \sum_{\psi \in \Psi} w_{\psi} \mathcal{N}(y_*; m_{y|D}(x_*; \psi), C_{y|D}(x_*; x_*; \psi)). \tag{5.5.21}
\]

The final estimate of (5.5.1) therefore takes the form of a Gaussian process mixture distribution, with elements equal to the predictions associated with our hyperparameter samples and weights determined both by their associated marginal likelihoods and their spatial locations in hyperparameter space. We also note that the weights \( w \) are again not dependent on the choice of output scales \( \{ \lambda_i \} \)—the inverse factors in the \( K^{-1} \) terms exactly cancel the factors in the \( \Omega \) term.

**Discussion**

The section above provides a basic presentation of using Bayesian numerical integration to estimate the integral in (5.5.1). This basic framework has been extended in various ways (Osborne, et al., 2010; Garnett, et al., 2010), which we will briefly describe below.

**Choice of prior and covariance**

The independent product prior function (5.5.7) and covariance functions (5.5.10, 5.5.12) used above were chosen for convenience in the presentation. Other choices may be used, as long as the integral in (5.5.17) can be evaluated. Garnett, et al. (2010), for example,
demonstrated how to appropriately treat a hyperparameter that assumes values on a finite set.

In particular, it might be advantageous to choose different input scales for the $q$ and $r$ covariance functions. Furthermore, when there is a prior belief that the values of certain hyperparameters are correlated, a nonindependent prior $p(\theta)$ could be substituted. When wanting to exploit the computational savings afforded by grid sampling that will be discussed below, however, the independent product forms of these functions are required.

Choice of quadrature hyperparameters

In the above, the various parameters of the Gaussian process models over $q$ and $r$ were treated as being given a priori. As discussed, our final estimate does not depend on the choice of output scales $\{\lambda_i\}$ in the covariance $K$. The estimate is, however, dependent on the choice of input scales $\{\sigma_i\}$. For this reason, an intelligent method for managing these quadrature hyperparameters would be beneficial.

This problem was addressed by Osborne, et al. (2010), where the authors discussed how these quadrature hyperparameters can be adjusted dynamically in a reasonable manner using maximum-likelihood–I estimation.

Hyperparameter posterior distributions

Using similar techniques as those described above, we may effectively estimate our posterior belief about the values of a subset of our hyperparameters indexed by the set

$$I \subseteq \{1, 2, \ldots, k\}$$

by marginalizing out the remaining hyperparameters $\theta \setminus I$:

$$p(\theta_I | D) = \frac{\int p(z | X, \theta) p(\theta) \, d\theta \setminus I}{\int p(z | X, \theta) p(\theta) \, d\theta}.$$ (5.5.22)

If desired, we may also estimate the posterior mean of the same subset of hyperparameters:

$$E[\theta_I] = \int \theta_I p(\theta_I | D) \, d\theta_I.$$

The procedures for calculating these estimates are very similar in nature to the method described in the last section for estimating (5.5.1), and for brevity we will not discuss them explicitly here. The interested reader is referred to (Garnett, et al., 2010) for details.

Sampling over likelihood functions

In the above section, we described a simple maximum a posteriori approximation to resolve the nontrivial integral over the unknown likelihood function $r$ (5.5.16). Osborne, et al. (2010) discussed an alternative approach wherein a second level of numerical integration is applied to the $r$ integral instead. The proposed idea was to intelligently generate well-separated samples in the likelihood-function space, then approximate the integral over $r$ using another round of Bayesian Monte Carlo integration. Reasonable likelihood-function samples can be generated using techniques from computational geometry (Osborne, et al., 2010).
Computational considerations

We will briefly consider the computational resources required to calculate the Bayesian Monte Carlo estimate of $\text{(5.5.1)}$. The discussion below refers to the general case when there is no special structure in the hyperparameter samples that can be exploited. In the next subsection, we will describe how sampling on a grid can speed up some of these computations.

Suppose that we have $N$ observations of the underlying function of interest and $M$ total hyperparameter samples. Evaluating the $q$ and $r$ functions at each point in $\Psi$ serially takes time $O(MN^3)$. Calculating the final weights on each of our prediction functions requires evaluating the product

$$K^{-1}\Omega K^{-1}.$$  \hfill (5.5.23)

If the samples in $\Psi$ are fixed, this term can be precomputed once and stored for future evaluations. Inverting the $K$ matrix using the Cholesky decomposition takes time $O(M^3)$, after which calculating the entire product (5.5.23) also takes time $O(M^3)$, using the naïve matrix-multiplication algorithm.

Once calculated, this product must be multiplied by the likelihood vector $r$ and normalized to derive the weights vector $w$. This operation takes time $O(M^2)$. This result can again be cached and reused until the likelihood observations $r$ or the hyperparameter samples $\Psi$ change.

Finally, we note that the evaluation of the $q$ and $r$ functions at our hyperparameter samples is “embarrassingly parallel”—each operation can be done completely independently. With $C$ total processing cores available, the evaluation of $q$ and $r$ can be performed in time approximately equal to

$$O\left(\min\left(N^3, \frac{MN^3}{C}\right)\right).$$

Grid sampling

When the hyperparameter samples $\Psi$ are in the form of a grid, we can exploit this structure to gain computational savings in our estimate. In particular, assume that for each of our hyperparameters $\theta_i$, we have selected a set of samples $\phi_i = \{\phi_{i,1}, \phi_{i,2}, \ldots\}$ of size $|\phi_i| = m_i$. Assume further that our hyperparameter samples are the Cartesian product of these sets:

$$\Psi = \prod_{i=1}^{k} \phi_i,$$

resulting in $M = \prod_{i=1}^{k} m_i$ total hyperparameter samples in $\Psi$. By choosing the independent product prior (5.5.7) and covariance functions (5.5.10, 5.5.12), our final estimate has a computationally convenient form. Both the covariance matrix $K$ and the $\Omega$ term in (5.5.20) can now be decomposed as Kronecker products of smaller matrices:

$$K = \bigotimes_{i=1}^{k} K_i(\phi_i, \phi_i),$$

$$\Omega = \bigotimes_{i=1}^{k} \Omega_i(\phi_i, \phi_i).$$
where the $K_i$ functions are defined in (5.5.11) and $\Omega_i$ refers to the $i$th integral term in the product (5.5.18).

The properties of the Kronecker product now makes inverting the $K$ matrix in particular much simpler:

$$K^{-1} = \bigotimes_{i=1}^{k} K_i(\phi_i, \phi_i)^{-1}.$$ 

We have therefore reduced the work required to invert $K$ from $O(M^2)$ to $O\left(\sum_{i=1}^{k} m_i^3\right)$, which can be a considerable speedup when $M$ is large. Furthermore, the cost of evaluating the weights in (5.5.20) is reduced as well. We have

$$w \propto K^{-1}\Omega K^{-1}r$$

$$= \left(\bigotimes_{i=1}^{k} K_i(\phi_i, \phi_i)^{-1}\Omega_i(\phi_i, \phi_i) K_i(\phi_i, \phi_i)^{-1}\right) r,$$

which is of the form of a matrix–vector multiplication. The structure in the Kronecker decomposition of the matrix allows us to perform this multiplication quickly. The naive algorithm requires $O(M^2)$ multiplications as well as $O(M^2)$ memory, which can be prohibitive with many hyperparameter samples. Fernandes, et al. (1998) presented an algorithm for performing this multiplication using only

$$O\left(M \sum_{i=1}^{k} m_i\right)$$

multiplications and even less memory—the product is derived iteratively, and the full matrix term never needs to be explicitly calculated. The weights can then be normalized in time $O(M)$.

These computational savings can allow us to use many more hyperparameter samples than we would be able to use generally, as long as we can tolerate the grid structure of our samples. In particular, the inversion and multiplication of large matrices is avoided; instead, we only need to work with potentially much smaller matrices over only the samples for each hyperparameter. Of course, these considerations do not save us from the computational burden of evaluating $q$ and $r$ functions at each of the points in $\Psi$. Nonetheless, grid sampling will allow us in later chapters to effectively use tens of thousands of hyperparameter samples without any serious computational hassle.
We will now consider the problem of performing time-series prediction in the face of abrupt changes to the properties of the variable of interest. For example, a data stream might undergo a sudden shift in its mean, variance, or characteristic input scale; a periodic signal might have a change in period, amplitude, or phase; or a signal might undergo a change so drastic that its behavior after a particular point in time is completely independent of what happened before. We will also consider cases in which our observations of the variable undergo such changes (even if the variable itself does not), as might occur during a sensor fault. A robust prediction algorithm must be able to make accurate predictions even under such unfavorable conditions.

The problem of detecting and locating sudden changes in data sequences has been studied for decades under the name change point detection. A large number of methods have been proposed for this problem; see (Basuville and Nikiforov, 1993; Brosky and Darkovsky, 1993; Csörgő and Horvath, 1998; Chen and Gupta, 2000) and the references therein for more information. Relatively few algorithms perform prediction simultaneously with change point detection, although sequential Bayesian methods do exist for this problem (Chernoff and Zacks, 1964; Adams and Mackay, 2007). However, these methods—and most methods for change point detection in general—make the assumption that the data stream can be segmented into disjoint sequences such that in each segment the data represent independent and identically distributed observations from an associated probability distribution. The problem of change points in dependent processes has received less attention. Both Bayesian (Carlin, et al., 1992; Ray and Tsay, 2002) and non-Bayesian (Müller, 1992; Horvath and Kokoszka, 1997) solutions do exist, although they focus on retrospective change point detection alone and their simple dependent models are not employed for the purposes of prediction. Sequential and dependent change point detection has been performed (Fearnhead and Liu, 2007), but only for a limited set of change point models.

Fault detection, diagnosis, and removal is an important application area for sequential time-series prediction in the presence of change points. Venkatasubramanian, et al. (2003) classified fault-recognition algorithms into three broad categories: quantitative model-based methods, qualitative methods, and process history-based methods.

Particularly related to the work we will present are the quantitative methods that employ recursive state estimators. The Kalman filter is commonly used to monitor innovation processes and prediction error (Willsky, 1976; Basseville, 1988). Banks of Kalman filters have also been applied to fault recognition, where each filter typically corresponds to a specific fault mode (Kobayashi and Simon, 2003; Aggarwal, et al., 2004; Reece, et al., 2009a). Recently, fault detection has also been studied using Gaussian processes (Garnett, et al., 2009; Reece, et al., 2009b).

In this chapter, we will propose a fully Bayesian framework for performing sequential time-series prediction in the presence of various types of drastic events. We will
introduce classes of nonstationary covariance functions to be used in Gaussian process inference when modeling functions that might undergo drastic changes in their behavior. We will also consider cases where the function determining our observations of the latent process instead undergoes a drastic change, which might occur due to a faulty sensing mechanism. In these contexts, the position of a changepoint or the onset of a fault will become a hyperparameter of the model. Given an appropriately constructed model, we will make predictions of the latent process by evaluating the full marginal predictive distribution \( \text{uniF648.} \). If the locations of changepoints in the data are of interest, we can estimate the full posterior distribution of the related hyperparameters given the data. The result will be a robust time-series prediction algorithm that makes well-informed predictions even in the presence of sudden changes in the data or our observations. If desired, the algorithm can additionally perform changepoint and fault detection as a natural byproduct of the prediction process.

### 6.1 Changepoints in the Latent Function

We will now describe how to construct appropriate covariance functions for functions that experience sudden changes in their characteristics. This section is meant to be expository; the covariance functions we introduce are intended as examples rather than an exhaustive list of possibilities. To simplify the presentation, we assume the input variable of interest, \( t \), is entirely temporal. If additional features are available, they may be readily incorporated into the derived covariances (see Section 5.3).

We consider the family of stationary covariance functions of the form (5.3.4); in the case of one-dimensional input, this reduces to the isotropic covariance family

\[
K(t, t'; \lambda, \sigma) \doteq \lambda^2 \kappa \left( \frac{|t - t'|}{\sigma} \right),
\]

where \( \kappa : \mathbb{R} \to \mathbb{R} \) is the kernel function.

For the below discussion, we will suppose that we have data of the form

\[
D \doteq \left\{ (t, z(t)) \right\}_{t \leq t'} \doteq (t, z),
\]

where \( z(t) \) is an observation of the latent function \( y(t) \) corrupted by independent Gaussian noise as described in Section 5.2. We will also make use of the notation \( t_{<t'} \), \( D_{<t'} \), etc.

**A drastic change in covariance**

Suppose a function of interest is well behaved except for a drastic change at the point \( t_c \), which separates the function into two regions with associated covariance functions \( K_1(t, t'; \theta_1) \) before \( t_c \) and \( K_2(t, t'; \theta_2) \) after, where \( \theta_1 \) and \( \theta_2 \) represent the values of any hyperparameters associated with \( K_1 \) and \( K_2 \), respectively. Suppose that the change is so drastic that the observations before \( t_c \) are completely uninformative about the observations after the changepoint. Probabilistically, we assume that for \( t_c \in \mathbb{R} \) we have

\[
p(y_\ast | t_c, D) = \begin{cases} p(y_\ast | t_c, D_{<t_c}) & t_c < t_c; \\ p(y_\ast | t_c, D_{>t_c}) & t_c \geq t_c. \end{cases}
\]
Figure 6.1: Illustration of the covariance function $K_{DC}$ modeling a drastic change at time $t = 5$. The component covariances $K_1$ and $K_2$ were both taken to be the squared exponential covariance function $K_{SE}$ with hyperparameters $(\lambda, \sigma) = (1, 1)$. Top: Prior distribution showing mean and pointwise ±2 standard-deviation bounds, along with five samples from the prior. Bottom: Prior covariance function $K_{DC}(4, t)$, along with $K_{SE}(4, t)$ for reference.
In this scenario, the appropriate covariance function is trivial. This function can be modeled using the covariance function $K_{DC}$ defined by

$$K_{DC}(t, t'; \theta_{DC}) = \begin{cases} 
K_1(t, t'; \theta_1) & t, t' < t_c; \\
K_2(t, t'; \theta_2) & t, t' \geq t_c; \\
0 & \text{otherwise.}
\end{cases} \quad (6.1.2)$$

The new set of hyperparameters,

$$\theta_{DC} = (\theta_1, \theta_2, t_c),$$

contains knowledge about the original hyperparameters of the component covariance functions and the location of the change point. We will call this covariance function the drastic-change covariance. The drastic-change covariance function is easily seen to be positive semidefinite and therefore admissible.

**Theorem 6.1.1** $K_{DC}$ is a valid covariance function.

**Proof** We show that any Gram matrix given by $K_{DC}$ is positive semidefinite. Consider an arbitrary set of input points $t \subset \mathbb{R}$. By appropriately ordering the points in $t$, we may write the Gram matrix $K_{DC}(t, t; \theta_{DC})$ as the block-diagonal matrix

$$
\begin{pmatrix}
K_1(t_{c1}, t_{c1}; \theta_1) & 0 \\
0 & K_2(t_{c2}, t_{c2}; \theta_2)
\end{pmatrix};
$$

the eigenvalues of $K_{DC}(t, t; \theta_{DC})$ are therefore the eigenvalues of the blocks along the diagonal. Because both $K_1$ and $K_2$ are valid covariance functions, their corresponding Gram matrices are positive semidefinite, and therefore the eigenvalues of $K_{DC}(t, t; \theta_{DC})$ are non-negative.

The covariance $K_{DC}$ is demonstrated in Figure 6.1.

A smooth drastic change in covariance

Suppose we have a function of interest known to be continuous and wish to model it with different covariance functions before and after a changepoint at time $t_c$. Suppose further that the function values after the changepoint are conditionally independent of the function values before, given the value at the changepoint itself. The Bayesian network for this probabilistic structure is depicted in Figure 6.2. This situation represents a refinement to the drastic-change covariance $K_{DC}$ described above—the two regions can still have dramatically different properties, but we now wish to enforce continuity (of the function, not necessarily any of its derivatives) across the boundary between them.
The changepoint separates the function into two regions with associated covariance functions $K_1(t, t'; \theta_1)$ before $t_c$ and $K_2(t, t'; \theta_2)$ after, where $\theta_1$ and $\theta_2$ represent the values of any hyperparameters associated with $K_1$ and $K_2$, respectively. For our constructed covariance to be positive semidefinite, we must assume that the chosen prior covariance functions are equal at the boundary $t_c$, and we will denote this common value $K_c$:

$$K_c = K_1(t_c, t_c; \theta_1) = K_2(t_c, t_c; \theta_2).$$

We may now model the function using the covariance function $K_{CCI}$ defined by

$$K_{CCI}(t, t'; \theta_{CCI}) = \begin{cases} K_1(t, t'; \theta_1) & t, t' < t_c; \\ K_2(t, t'; \theta_2) & t, t' > t_c; \\ K_1(t, t_c; \theta_1)K_c^{-1}K_2(t, t_c; \theta_2) & \text{otherwise;} \end{cases} \tag{6.1.3}$$

where

$$\theta_{CCI} = (\theta_1, \theta_2, t_c).$$

We call this covariance function the continuous conditionally independent covariance. This covariance function can be extended to handle multiple change points or boundaries in multiple dimensions. It can also be extended to cases where function derivatives are known to be continuous at the change point. For further details about this covariance function, the reader is invited to consult [Reece et al., 2009b].

The covariance $K_{CCI}$ is demonstrated in Figure 6.3.

**Temporary drift**

For the purposes of fault detection and removal, we will occasionally want to model a function that undergoes a smooth temporary deviation from an otherwise constant value of zero. Define the beginning of the drift as $t_{c1}$ and its end as $t_{c2}$, and write

$$t_c = [t_{c1}, t_{c2}]^T.$$

Suppose that we wish to model the deviation using the covariance function $K(t, t'; \theta)$. Because we insist the drift must begin and end with the function value zero, the proper covariance for this situation is straightforward. Inside the region $t_{c1} \leq t, t' \leq t_{c2}$, we simply use the covariance $K$ conditioned on the two observations

$$y(t_{c1}) = y(t_{c2}) = 0;$$

outside this region, the covariance is trivial. With this observation, we can now define an appropriate covariance function for this situation with

$$K_{drift}(t, t'; \theta_{drift}) = \begin{cases} K(t, t'; \theta) - K(t, t_c; \theta)K(t_c, t_c; \theta)^{-1}K(t_c, t'; \theta) & t_{c1} \leq t, t' \leq t_{c2}; \\ 0 & \text{otherwise,} \end{cases} \tag{6.1.4}$$

where

$$\theta_{drift} = (\theta, t_{c1}, t_{c2}).$$
Figure 6.3: Illustration of the covariance function $K_{CCI}$ modeling a continuous drastic change at time $t = 5$. The component covariances $K_1$ and $K_2$ were taken to be the squared exponential covariance function $K_{SE}(5,3.5)$ with hyperparameters $(\lambda, \sigma) = (1,1)$ before the changepoint and the Matérn covariance function $K_{M}(5,3,6,5.3,7)$ with hyperparameters $(\lambda, \sigma, \nu) = (1,1,\frac{3}{2})$ after. Top: Prior distribution showing mean and pointwise $\pm 2\sigma$ standard-deviation bounds, along with five samples from the prior. Bottom: Prior covariance functions $K_{CCI}(4, t)$ and $K_{CCI}(6, t)$, along with $K_{SE}(4, t)$ and $K_{M}(6, t)$ for reference.
6.1. Changepoints in the Latent Function

Although we only define $K_{\text{drift}}$ for one-dimensional input spaces, it can easily be extended to multiple dimensions. In higher dimensions, conditioning the covariance on a complex boundary might be problematic; however, this can be overcome with care. For example, using an independent product covariance formed of several one-dimensional $K_{\text{drift}}$ covariances could work well in many cases without adding too much complexity. The covariance $K_{\text{drift}}$ is demonstrated in Figure 6.4.
A sudden change in input scale

Suppose a function of interest is well behaved except for a drastic change in the input scale \( \sigma \) at time \( t_c \), which separates the function into two regions with different degrees of long-term dependence.

Let \( \sigma_1 \) and \( \sigma_2 \) represent the input scale of the function before and after the changepoint at \( t_c \), respectively. Suppose we wish to model the function with an isotropic covariance function \( K \) of the form (6.1.1) that would be appropriate except for the change in input scale. We may model the function using the covariance function \( K_{CI} \) defined by

\[
K_{CI}(t, t'; \lambda, \sigma_1, \sigma_2, t_c) = \begin{cases} 
K(t, t'; \lambda, \sigma_1) & t, t' < t_c; \\
K(t, t'; \lambda, \sigma_2) & t, t' \geq t_c; \\
\lambda^2 K \left( \frac{|t-t'|}{\sigma_1} + \frac{|t'-t|}{\sigma_2} \right) & \text{otherwise.}
\end{cases}
\]

(6.1.5)

**Theorem 6.1.1** \( K_{CI} \) is a valid covariance function.

**Proof** Consider the map defined by

\[
u(t; \sigma_1, \sigma_2, t_c) = \begin{cases}
\frac{t}{\sigma_1} & t \leq t_c; \\
\frac{t}{\sigma_1} + \frac{t-t_c}{\sigma_2} & t \geq t_c.
\end{cases}
\]

(6.1.6)

A simple check shows that

\[
K_{CI}(t, t'; \lambda, \sigma_1, \sigma_2, t_c) = K \left( \nu(t; \sigma_1, \sigma_2, t_c), \nu(t'; \sigma_1, \sigma_2, t_c); \lambda, \sigma = 1 \right),
\]

the original covariance function with equivalent output scale and unit input scale evaluated on the input points after transformation by \( \nu \). Because \( \nu \) is injective and \( K \) is a valid covariance function, the result follows.

The function \( \nu \) in the proof above motivates the definition of \( K_{CI} \)—by rescaling the input variable appropriately, the change in input scale is removed.

The covariance \( K_{CI} \) is demonstrated in Figure 6.5.

A sudden change in output scale

Suppose a function of interest is well behaved except for a drastic change in the output scale \( \lambda \) at time \( t_c \), which separates the function into two regions.

Let \( y(t) \) represent the latent function of interest and let \( \lambda_1 \) and \( \lambda_2 \) represent the output scale of \( y(t) \) before and after the changepoint at \( t_c \), respectively. Suppose we wish to model the function with an isotropic covariance function \( K \) of the form (6.1.1) that would be appropriate except for the change in output scale. To derive the appropriate covariance function, we model \( y(t) \) as the product of a function with unit output scale, \( g(t) \), and a piecewise-constant scaling function, \( a(t) \), defined by

\[
a(t; \lambda_1, \lambda_2, t_c) = \begin{cases}
\lambda_1 & t < t_c; \\
\lambda_2 & t \geq t_c.
\end{cases}
\]

(6.1.7)
Figure 6.5: Illustration of the covariance function $K_{CI}$ modeling a change in input scale at time $t = 5$. The component covariance $K$ was taken to be the squared exponential covariance function $K_{SE}(5, 3, 5)$ with output scale $\lambda = 1$. The input scale changes from $\sigma = 1$ before the changepoint to $\sigma = \frac{3}{2}$ after. Top: Prior distribution showing mean and pointwise ±2 standard-deviation bounds, along with five samples from the prior. Bottom: Prior covariance function $K_{CI}(4, t)$, along with $K_{SE}(4, t)$ for reference.
Given the model \( y(t) = a(t) g(t) \), the appropriate covariance function for \( y \) is immediate. We may use the covariance function \( K \) defined by

\[
K(t, t'; \lambda_1, \lambda_2, \sigma, t_c) = a(t; \lambda_1, \lambda_2, t_c) K(t, t'; \lambda = 1, \sigma) a(t'; \lambda_1, \lambda_2, t_c)
\]

\[
= \begin{cases} 
K(t, t'; \lambda_1, \sigma) & t, t' < t_c; \\
K(t, t'; \lambda_2, \sigma) & t, t' \geq t_c; \\
K\left( t, t'; \sqrt{\lambda_1 \lambda_2}, \sigma \right) & \text{otherwise.}
\end{cases}
\]

The form of \( K \) is an application of (5.3.10).

Notice that if the latent function is nonzero at the changepoint, our model will introduce a removable jump discontinuity at \( t_c \). If we have a reasonable belief about the value of the latent function at the changepoint boundary, we can remove this discontinuity with an appropriate modification to our mean function.

The covariance \( K \) is demonstrated in Figure 6.6.

### 6.2. Change Points in Observation Likelihood

The covariance functions introduced in the last section all model various types of drastic changes to the properties of the underlying latent function \( y(t) \). In this section, we consider the case where the behavior of the latent function does not change, but the properties of our observations of it do. Such changes are frequently encountered, for example, in sensor networks, where a failing sensor may provide systematically inaccurate observations. Changes of this type are commonly referred to as faults.

A common technique for responding to a suspected fault is simply to ignore any incoming data until a determination has been made that the fault has ended. We will refer to this procedure as fault removal. In situations where a model of the fault is known, however, the faulty observations do not necessarily need to be discarded—they may still contain valuable information about the latent function. For example, if we believed that a sensor fault was introducing bias into our observations of a fixed value, the data would provide perfectly usable information after a simple translation. We will refer to the task of performing inference about \( y \) from faulty observations as fault recovery, to distinguish it from simple fault removal.

We will begin by defining a new, more general observation model, then proceed to describe how various common fault types can be handled using this framework.

#### A more general observation model

Suppose that we have a latent function \( y: X \rightarrow \mathbb{R} \) of interest and have placed a Gaussian process prior over \( y \):

\[
p(y \mid \theta) \equiv \mathcal{GP}(y; \mu, K).
\]

As in Section 5.2, we again consider the measurement of our variable of interest at a point \( x \in X \) and the associated observation function \( z(x) \).

In every situation so far encountered, we have assumed that our observations are equal to the latent function corrupted by independent and identically distributed zero-mean Gaussian noise:

\[
z(x) = y(x) + \epsilon(x),
\]

with

\[
p(\epsilon(x) \mid x, \eta) = \mathcal{N}(0, \eta^2).
\]
6.2. Changepoints in Observation Likelihood

Figure 6.6: Illustration of the covariance function $K_{CO}$ modeling a change in output scale at time $t = 5$. The component covariance $K$ was taken to be the squared exponential covariance function $K_{SE}(5, 5)$ with input scale $\sigma = 1$. The output scale changes from $\lambda = 1$ before the changepoint to $\lambda = \gamma / 2$ after. Top: Prior distribution showing mean and pointwise $\pm 2\sigma$ standard-deviation bounds, along with five samples from the prior. The jump discontinuity at the changepoint has been removed for clarity. Bottom: Prior covariance function $K_{CO}(4, t)$, along with $K_{SE}(4, t)$ for reference.
This gave rise to the simple observation likelihood used thus far,

\[ p(z \mid y, X, \theta) = \mathcal{N}(z; y, \eta^2 I). \]

To model faults of various types, we will adopt a more general observation likelihood. We will model our observations as an affine transformation of \( y \) corrupted by now potentially correlated zero-mean Gaussian noise:

\[ z(x) \triangleq (a(x)y(x) + b(x)) + \varepsilon(x), \]

where \( a, b: \mathcal{X} \to \mathbb{R} \) are two chosen functions and \( \varepsilon \) is a random variable representing the error in our measurement of \( a(x)y(x) + b(x) \).

To allow for potentially correlated noise during a fault, we will place a zero-mean Gaussian process prior on \( \varepsilon \):

\[ p(\varepsilon \mid \theta) \equiv \mathcal{GP}(\varepsilon; 0, K_{\text{fault}}). \]

The covariance function \( K_{\text{fault}} \) associated with the fault model will likely be different from the covariance over the latent process \( y, K \).

With these definitions, our observation function \( z \) now has a Gaussian process distribution, as it is a sum of two Gaussian-process distributed functions: the affine transformation \( a(x)y(x) + b(x) \) of \( y \) (see Section 5.3) and the correlated noise term \( \varepsilon(x) \). The new observation likelihood for the vector \( z \equiv z(X) \) with \( X \subseteq \mathcal{X} \) is now easy to compute:

\[ p(z \mid y, X, \theta) = \mathcal{N}(z; A(X; \theta)y + b(X; \theta), K_{\text{fault}}(X, X; \theta)), \quad (6.2.1) \]

where

\[ A(X; \theta) \equiv \text{diag} a(X; \theta) \]

and we have explicitly parametrized the \( a, b, \) and \( K_{\text{fault}} \) functions.

With this model, our observation likelihood remains Gaussian, and conditioning the distribution over the latent function \( y \) given faulty observations remains simple. The procedure we follow is identical to that outlined in Section 5.2, and we will omit the details of the intermediate calculations. Suppose that we have faulty observations of the form

\[ D \equiv \{(x, z(x))\}_{x \in X} \subseteq (X, z). \]

We define the following condensed symbols for notational brevity:

\[ A \equiv A(X; \theta), \]
\[ b \equiv b(X; \theta), \]
\[ V_{\text{fault}} \equiv \text{AKA} + K_{\text{fault}}(X, X; \theta). \]

We now have

\[ p(y \mid D, \theta) = \mathcal{GP}(y; m_{y|D}, C_{y|D}), \quad (6.2.2) \]

where

\[ m_{y|D}(x; \theta) \triangleq \mu(x; \theta) + K(x, X; \theta) V_{\text{fault}}^{-1} \left( z - (A\mu(X; \theta) + b) \right), \]
\[ C_{y|D}(x, x'; \theta) \triangleq K(x, X; \theta) - K(x, X; \theta) A V_{\text{fault}}^{-1} A K(X, X; \theta). \]
are the posterior mean and covariance functions, respectively.

If required, we can also determine the posterior for the fault contribution, defined as \( f = z - y \). Because we have

\[
  f(x) = z(x) - y(x) = a(x)y(x) + b(x) + \epsilon(x) - y(x) = (a(x) - 1)y(x) + b(x) + \epsilon(x),
\]

the posterior \( p(f | D, \theta) \) has the same form as \( p(y | D, \theta) \), but with all instances of \( A \) replaced with \( (A - I) \). Define

\[
  \hat{A} = A - I, \\
  \hat{V}_{\text{fault}} = \hat{A}K\hat{A} + K_{\text{fault}}(X, X; \theta).
\]

Then

\[
  p(f | D, \theta) = \mathcal{GP}(f; m_{f|D}, C_{f|D}), \\
  \text{(6.2.3)}
\]

where

\[
  m_{f|D}(x; \theta) = \mu(x; \theta) + K(x, X; \theta)\hat{V}_{\text{fault}}^{-1}\left(z - \left(\hat{A}\mu(X; \theta) + b\right)\right), \\
  C_{f|D}(x, x'; \theta) = K(x, x'; \theta) - K(x, X; \theta)\hat{A}\hat{V}_{\text{fault}}^{-1}\hat{A}K(X, x'; \theta).
\]

The observation model considered above can represent a wide variety of fault types. We will now consider some illustrative examples that can be handled using this approach.

**Bias**

Perhaps the simplest fault mode is bias, in which faulty readings are simply offset from the true values by some constant amount (and then, potentially, further corrupted by additive Gaussian noise). Clearly, knowing the fault model in this case can allow us to extract useful information from the faulty readings; in other words, we are able to perform fault recovery. Suppose that we believe \( \mathcal{F} \subseteq \mathcal{X} \) to be the region of our input space that will give rise to biased observations. In this scenario, we take

\[
  a(x) = 1, \\
  K_{\text{fault}}(x, x'; \eta) = \eta^2 \delta(x, x'),
\]

where \( \delta \) is the Kronecker delta function (5.5.13), and

\[
  b(x; e, \mathcal{F}) = \begin{cases} 
  e & x \in \mathcal{F}; \\
  0 & x \notin \mathcal{F},
\end{cases}
\]

where \( e \) is the constant offset in the faulty region. The offset value \( e \), the observation-noise parameter \( \eta \), and the faulty region \( \mathcal{F} \) are the fault-model hyperparameters to be included in \( \theta \).
Stuck value

Another simple fault model is a stuck value, where our faulty readings return a constant value regardless of the value of the actual latent process. We will consider the slightly more general model in which those faulty observations may also include a Gaussian-noise component on top of the constant value. In this situation we can only hope to perform fault removal—the faulty readings are not at all informative for our inference about the underlying variable of interest. Suppose that we believe \( \mathcal{F} \subseteq \mathcal{X} \) to be the region of our input space that will give rise to stuck-value observations. In this situation, we take as before

\[
K_{\text{fault}}(\mathbf{x}, \mathbf{x}'; \eta) \triangleq \eta^2 \delta(\mathbf{x}, \mathbf{x}').
\]

The \( a \) and \( b \) functions appropriate for this situation are given by

\[
a(\mathbf{x}; \mathcal{F}) \triangleq \begin{cases} 0 & \mathbf{x} \in \mathcal{F}; \\ 1 & \mathbf{x} \notin \mathcal{F}; \end{cases}
\]

\[
b(\mathbf{x}; s, \mathcal{F}) \triangleq \begin{cases} s & \mathbf{x} \in \mathcal{F}; \\ 0 & \mathbf{x} \notin \mathcal{F}, \end{cases}
\]

where \( s \) is the stuck value. The stuck value \( s \), the observation-noise parameter \( \eta \), and the faulty region \( \mathcal{F} \) are the fault-model hyperparameters to be included in \( \theta \).

Notice that by defining \( a(\mathbf{x}) = 0 \) for \( \mathbf{x} \in \mathcal{F} \), our prediction equations (6.2.2) will automatically ignore any values suspected of being faulty.

Drift

The final fault type we consider is drift. In this case, our sensor readings undergo a smooth excursion from the latent process; that is, they gradually “drift” away from the real values, before eventually returning back to normality. In Section 6.1, we defined an appropriate covariance family to use in this situation—\( K_{\text{drift}} \). Suppose that we believe \( \mathcal{F} \subseteq \mathcal{X} \) to be the region of our input space that will give rise to drifted observations. In this situation, we take

\[
a(\mathbf{x}) \triangleq 1,
\]

\[
b(\mathbf{x}) \triangleq 0,
\]

\[
K_{\text{fault}}(\mathbf{x}, \mathbf{x}'; \eta, \theta_{\text{drift}}) \triangleq K_{\text{drift}}(\mathbf{x}, \mathbf{x}'; \theta_{\text{drift}}) + \eta^2 \delta(\mathbf{x}, \mathbf{x}').
\]

With knowledge of this model, fault recovery is possible. The observation-noise parameter \( \eta \) and the faulty region \( \mathcal{F} \) are the fault-model hyperparameters to be included in \( \theta \), along with any further hyperparameters required by the covariance function chosen for use in \( K_{\text{drift}}, \theta_{\text{drift}} \).

Discussion

To perform time-series prediction on data streams featuring changepoints and faults (which we will refer to in this discussion as events), we simply choose a GP model that can represent the various types of drastic events that we wish to consider. The key feature of our approach is the treatment of the location and characteristics of changepoints and faults as model hyperparameters. Because we will often not know the exact locations
and characteristics of events before they happen, we will choose prior distributions for these hyperparameters and, when making predictions, marginalize them using BMC hyperparameter marginalization (5.5.21). This procedure effectively averages over models corresponding to a range of events compatible with the data. If desired, the inferred nature of various events can also be directly monitored by finding their posterior distributions given our observations (Garnett, et al., 2010).

Using this framework, we are able to calculate the posterior distributions of any unknown quantity, such as the putative location of a changepoint or the probability that a fault of a particular type might have occurred. In some applications, it might be necessary to make a hard decision, that is, to commit to a event having occurred at a given point in time. This would be necessary, for example, if a system had correctional or responsive actions that it could take when such an event occurred. Fortunately, we can address this temporal-segmentation problem using simple Bayesian decision theory. Given our observations \( D \), we can determine the probability that there was an event at \( t_c \),

\[
Pr(\text{event}(t_c) \mid D);
\]

see (Garnett, et al., 2010) for details. Now after specifying the costs of false-positive and false-negative reports as \( c_I \) and \( c_{II} \), respectively (and defining the cost of true-positive and true-negative reports to be zero), we simply take the action that minimizes the expected loss. If

\[
c_I \left(1 - Pr(\text{event}(t_c) \mid D)\right) < c_{II} Pr(\text{event}(t_c) \mid D),
\]

we report an event at time \( t_c \); otherwise, we do not. Continuing in this manner, we can segment the entire data stream.

The covariance functions presented above can be extended in a number of ways. First, they can be extended to handle multiple changepoints. In this case, we simply need to introduce additional hyperparameters for their locations and the values of the appropriate covariance hyperparameters within each segment. Note, however, that in the case of time-series prediction, we will nearly always be limiting our model to observations that lie within a moving window. At any point in time, our model only needs to accommodate the region spanned by the current window. In practice, allowing for one or two changepoints is usually sufficient for the purposes of prediction, considering that the data prior to a changepoint are typically weakly correlated with data in the current regime of interest. Therefore when performing prediction we can circumvent the computationally onerous task of simultaneously marginalizing the hyperparameters associated with the entire data stream.

Additionally, if multiple events might occur simultaneously, an appropriate covariance function or observation model can be derived by combining the above results. For example, a function that experiences a change in both input scale and output scale could be readily modeled by

\[
K_{CB}(t, t'; \lambda_1, \lambda_2, \sigma_1, \sigma_2, \lambda_c) \pm a(t)K(u(t), u(t); \lambda = 1, \sigma = 1) a(t'),
\]

where \( u \) is as defined in (6.1.6) and \( a \) is as defined in (6.1.7). The covariance \( K_{CB} \) is demonstrated in Figure 6.7.

6.2. Changepoints in Observation Likelihood
Figure 6.7: Illustration of the covariance function $K_{\text{CB}}$ modeling changes in input and output scale at time $t = 5$. The component covariance $K$ was taken to be the squared exponential covariance function $K_{\text{SE}} (5, 3, 5)$. The hyperparameters change from $(\lambda, \sigma) = (1, 1)$ before the changepoint to $(\lambda, \sigma) = (\lambda/2, \sigma/2)$ after. Top: Prior distribution showing mean and pointwise ±2 standard-deviation bounds, along with five samples from the prior. The jump discontinuity at the changepoint has been removed for clarity. Bottom: Prior covariance function $K_{\text{CB}} (4, t)$, along with $K_{\text{SE}} (4, t)$ for reference.
If we are required to segment a data stream using a model allowing multiple simultaneous event types, we may be compelled to decide exactly which type of event to report at a particular point in time. As in our discussion on decisions above, this would require the specification of a function that would stipulate the loss associated with reporting an event of one type when there was actually an event of another type. Given this function, we again simply make the report that minimizes the expected loss.

Finally, our framework allows for incorporating a possible change in mean, although this does not involve the covariance or observation structure of the model. Suppose that the mean function associated with the data is suspected of undergoing possible changes. We may treat its parameters as hyperparameters of the model and select appropriate priors and samples corresponding to the behavior of the mean function before and after a putative changepoint. The different possible mean functions will then be properly marginalized for prediction, and the likelihoods associated with the samples can give support for the proposition that an event occurred at a particular time.

6.3 Expository Example

As an expository example, let us consider a function that undergoes a sudden change in both input scale and output scale. The function \( y(t) \) is displayed in Figure 6.8; it undergoes a sudden change in input scale (becoming smaller) and output scale (becoming larger) at the point \( t = 0.5 \). We consider the problem of performing one-step lookahead prediction on \( y(t) \) using a GP model with a moving window containing 25 samples.

The uppermost plot in Figure 6.8 shows the performance of a standard GP prediction model with the squared exponential covariance \( K_{SE} \) (5.3.5) using hyperparameters selected by maximum-likelihood–estimation on the data before the changepoint. The standard GP prediction model has clear problems coping with the changepoint—after the changepoint it makes predictions that are very certain (that is, have small predictive variance) that are nonetheless very inaccurate.

The central plot shows the performance of a GP prediction model using \( K_{SE} \) incorporated into the input- and output-scale changepoint covariance function \( K_{CB} \) (6.2.4). The predictions were calculated using BMC hyperparameter marginalization (5.5.21); three samples each were chosen for the hyperparameters \( \lambda_1, \lambda_2, \sigma_1, \) and \( \sigma_2 \), and 25 samples were chosen for the location of the changepoint. Our model easily copes with the changed parameters of the process, continuing to make accurate predictions immediately after the changepoint. Furthermore, by marginalizing the various hyperparameters associated with our model, the uncertainty associated with our predictions is conveyed honestly. The standard deviation becomes roughly an order of magnitude larger after the changepoint due to the similar increase in the output scale.

The lowest plot shows the posterior distribution of the distance to the most-recent changepoint corresponding to the predictions made by the changepoint GP predictor. The changepoint at \( t = 0.5 \) can be clearly identified.

6.4 Results

In this section, we will demonstrate the performance of our changepoint prediction algorithm using a number of real datasets.
Figure 6.8: Top: Posterior mean predictions and pointwise ±2 standard-deviation bounds made for the expository example function using the squared exponential covariance function $K_{SE}$ (5.3.5). Middle: Posterior mean predictions and pointwise ±2 standard-deviation bounds made for the expository example function using the input- and output-scale changepoint covariance $K_{CB}$ (6.2.4). Bottom: Corresponding changepoint location posterior produced by our algorithm. Each vertical "slice" of the figure at a particular point shows the posterior probability distribution of the distance to the most-recent changepoint at that point. Blue indicates higher probability mass.
Figure 6.9: Top: Posterior mean predictions and pointwise ±2 standard-deviation bounds made for the Nile dataset using the input-scale changepoint covariance $K_{CI}$ (6.1.5). Bottom: Corresponding changepoint location posterior produced by our algorithm. Blue indicates higher probability mass.

Nile data

We first consider a canonical changepoint dataset—the minimum water levels of the Nile river during the period AD 622–1284 (Whitcher, et al., 2002). Several authors have found evidence supporting a change in input scale for these data around the year AD 722 (Ray and Tsay, 2002). The conjectured reason for this changepoint is the construction in AD 715 of a new device (a "Nilometer") on the island of Roda, which affected the nature and accuracy of the measurements.

We performed one-step lookahead prediction on this dataset using a moving window of size 150. We chose the constant prior mean function $\mu(t)$ $\pm$ 1000 cm and selected the squared exponential covariance function $K_{SE}$ (5.3.5). We assumed that the output scale and observation-noise variance in the dataset were stationary and selected the values of the corresponding hyperparameters using maximum-likelihood estimation. We modeled a possible change in input scale by modifying $K_{SE}$ using the input-scale changepoint covariance $K_{CI}$ (6.1.5). The hyperparameters that we were therefore required to marginalize were $\sigma_1$ and $\sigma_2$, the input scales before and after a putative changepoint, and the changepoint location $t_c$. We selected a grid of samples containing 11, 11, and 150 samples, respectively, for each of these hyperparameters.$^1$

$^1$The choice of hyperparameter grid size for each experiment was typically chosen to maximize the number of available samples given memory and time constraints.
Figure 6.10: Posterior mean retrospective predictions and pointwise ±2 standard-deviation bounds for the well-log data using a hidden-Markov model (top) and a \( \alpha \beta \) with drastic-change covariance \( K_{DC} \) (6.1.2) (bottom).

The results can be seen in Figure 6.9. The upper plot shows our predictions for the dataset, including the mean and pointwise ±2 standard-deviation error bars. The lower plot shows the posterior distribution of the number of years since the last changepoint. A changepoint around AD 720–722 is clearly visible and agrees with previous results.

Well-log data

Also commonly considered in the context of changepoint detection is the “well-log” dataset, which comprises 4 050 measurements of nuclear magnetic response (NMR) recorded during the drilling of a well (Ruanaidh, et al., 1994). Changepoints in the dataset correspond to the transitions between different strata of rock.

We performed prediction on this dataset using a simple diagonal covariance that assumed all measurements were independent and identically distributed. The noise
variance for this covariance (equivalently, its output scale) was determined by maximum-likelihood estimation and assumed known \textit{a priori}. We modeled the jump discontinuities in the dataset using a piecewise-constant prior mean function. Considering the length of the dataset and that regions of data before and after a changepoint were assumed to be independent, we performed predictions at a particular point from a window of data centered on that point. That is, we performed sequential prediction for data points midway through the current window. In each window (spanning 50 observations), we allowed for a single changepoint in the mean. Our model was therefore required to marginalize over three hyperparameters: the mean before the changepoint, the mean after the changepoint, and the changepoint location. For these hyperparameters, we selected a grid containing 13, 13, and 40 samples, respectively, in each coordinate.

We compared our results against those produced by a variational-Bayesian hidden-Markov model with a mixture-of-Gaussians emission probability (Ji \textit{et al.}, 2006; Lee, 2009). This model gave a marginal log-likelihood of

\[
\log p(z \mid X) \approx -1.51 \times 10^5;
\]

our \textit{GP} model gave

\[
\log p(z \mid X) \approx -1.02 \times 10^4.
\]

The resulting predictions for both methods are depicted in Figure 6.10. According to this metric, the performance of our \textit{GP} was an order of magnitude better than the alternative method, largely due to the predictions made in the regions just before \( t = 1600 \) s and just after \( t = 2400 \) s.

\textit{1972--1975 Dow Jones industrial average}

A final canonical changepoint dataset is the series of daily returns of the Dow Jones industrial average between the 3rd of July, 1972 and the 30th of June, 1975 (Adams and Mackay, 2007). This period included a number of newsworthy events that had significant macroeconomic influence, which are reflected in the Dow Jones returns.

We performed sequential prediction on these data using a zero-mean \textit{GP} with a diagonal covariance that assumed all measurements were independent; however, the variance of these observations was assumed to undergo changes. We modeled this behavior using the output-scale changepoint covariance \( \mathcal{K}_{CO} \) (6.1.8). We used a prediction window 350 observations long and assumed that the data contained no more than a single changepoint in this window. We therefore had three hyperparameters to marginalize: the variance before the changepoint, the variance after the changepoint, and the changepoint location. We selected a grid of samples containing 17, 17, and 50 samples, respectively, for these hyperparameters.

Our results are plotted in Figure 6.11. Our model clearly identified the important changepoints that likely correspond to the commencement of the OPEC embargo on the 19th of October, 1973 and the resignation of Richard Nixon as President of the United States on the 9th of August, 1974. A weaker changepoint was identified early in 1973, which Adams and Mackay (2007) speculated might be in response to the beginning of the Watergate scandal.
6. SEQUENTIAL BAYESIAN PREDICTION WITH CHANGEPONITS

Figure 6.11: Top: Posterior mean predictions and pointwise ±2 standard-deviation bounds made for the Dow Jones dataset using the output-scale changepoint covariance \( K_{\text{CCI}} \) (6.1.8). Bottom: Corresponding changepoint location posterior produced by our algorithm. Blue indicates higher probability mass.

**EEG data with epileptic event**

The next dataset we considered was electroencephalography (EEG) data from an epileptic subject (Roberts, 2000b). Prediction in such datasets is typically performed with the aim of ultimately building models for EEG activity strong enough to forecast seizure events (Faul, et al., 2007). Our particular dataset is a single EEG channel recorded at 64 Hz with 12-bit resolution and is plotted in Figure 6.12. Halfway through the data, a single epileptic event begins of the classic “spike-and-wave” type.

We selected a Gaussian process prior for the EEG signal with a constant mean function \( \mu(t) = 120 \). We used the continuous conditionally independent covariance \( K_{\text{CCI}} \) (6.1.3) to model the data, accommodating the smooth transition of the data between drastically different regimes. We took \( K_1 \) to be the squared exponential covariance \( K_{\text{SE}} \) (5.3.5) and \( K_2 \) to be the product of two terms: the periodic covariance \( K_P \) (5.3.9) and another squared exponential covariance. \( K_2 \) was designed to model the EEG data during the course of seizure; \( K_1 \) was designed to model data from other regions. We also selected the independent and identically distributed Gaussian observation noise model (5.2.1). We assumed that we had sufficient EEG data unaffected by seizure to set the hyperparameters for \( K_1 \) and the observation model using maximum-likelihood–it estimation. We further assumed that the input scale of the nonperiodic squared exponential within \( K_2 \) was identical to that for \( K_1 \), representing a constant long-term smoothness for both seizure and non-seizure periods. Therefore the hyperparameters that we were required to marginalize were the period \( \sigma \), amplitude \( \lambda \), and smoothness \( y \) of \( K_P \) for \( K_2 \), along with the location of the changepoint and its type (either periodic to nonperiodic,
6.4. Results

Figure 6.12: Top: Posterior mean retrospective predictions and pointwise ±2 standard-deviation bounds made for the epileptic EEG dataset using the continuous conditionally independent changepoint covariance $K_{CCI}$ (6.1.3). Bottom: Corresponding changepoint location posterior produced by our algorithm. Blue indicates higher probability mass.

or nonperiodic to periodic). For these hyperparameters, we used a grid containing 7, 7, 5, 50, and 2 samples, respectively, in each coordinate.

This model was used to perform retrospective prediction over the dataset, as depicted in Figure 6.12. Our posterior distribution for the location of the changepoint accurately located the true onset of seizure.

Tide height with stuck sensor

To illustrate our approach for sensor fault detection, we tested on data from a network of weather sensors located off the southern coast of England.\footnote{The network is maintained by the Bramblemet Support Group and funded by organizations including the Royal National Lifeboat Institution, the Solent Cruising and Racing Association, and Associated British Ports.} We specifically considered the readings from the Sotonmet sensor, which makes measurements of a number of environmental variables (including wind speed and direction, air temperature, sea temperature, and tide height) and makes up-to-date sensor measurements available on the internet.\footnote{See http://www.sotonmet.co.uk/}. These sensors are subject to network outages and other faults, suggesting that the models described in Section 6.2 might be beneficial.

In particular, we performed online prediction on a sequence of tide-height data during which readings from the sensor became stuck at an incorrect value. The float...
Figure 6.13: Top: Posterior mean retrospective predictions and pointwise ±2 standard-deviation bounds made for the tide-height data. The fault was modeled as a change in observation likelihood of the form described in Section 6.2. Bottom: Corresponding changepoint location posterior produced by our algorithm. Blue indicates higher probability mass.

sensor measuring the tide became lodged on a post for almost two days and reported a constant value of 4 m throughout that period. To model the possibility of this type of observation fault, we used the observation-fault model for stuck values described in Section 6.2.

We placed a zero-mean GP prior on the tide-height signal. The covariance for the underlying latent process was taken to be the sum of a periodic and a nonperiodic component as described in ( Osborne, et al., 2008 ); the values for these hyperparameters were determined offline via maximum-likelihood–estimation using previously recorded, fault-free data. We therefore needed to marginalize only the hyperparameter corresponding to the location of a changepoint in the prediction window and a binary hyperparameter corresponding to the type of that changepoint (that is, either not-stuck to stuck, or stuck to not-stuck). Our belief about the stuck value can be heuristically determined for any appropriate region—we simply search for a repeated constant observed value. Using a heuristic of this type, we do not need to sample over the (a priori unknown) stuck value. We employed a window size of 450 data points and correspondingly used 450 samples for the changepoint location. The results are plotted in Figure 6.13. Our model correctly identified the beginning and end of the fault. By then performing fault removal using (6.2.2), the model was also able to perform effective prediction
for the tide process throughout the faulty region. The uncertainty grew throughout the region because our model automatically ignored the incoming faulty observations; therefore, our predictions were based on increasingly old data during that time period.

**EEG data with saccade event**

To further illustrate our approach to sensor fault recovery, we also tested the performance of our algorithm on a further EEG application. Electroencephalography provides a highly effective noninvasive measurement of brain activity. However, the EEG signal can often be corrupted by artifacts corresponding to rapid, involuntary eye movements called saccades. Any saccade event artifacts in a subject’s EEG signal will correspond with simultaneous activity in the subject’s electro-oculograph (EOG) signal. Figure 6.14 shows typical EOG activity during a saccade. In most biomedical applications, however, a measured EOG signal is rarely available, and we must instead rely on artifact-removal algorithms to offer an accurate assessment of the pure EEG signal (which we will refer to as $\text{EEG}^*$). This problem was treated as a blind source-separation problem in (Roberts, et al., 1999), and an independent component analysis solution was proposed to separate the artifact-free $\text{EEG}^*$ signal from the EOG signal.

Here we will demonstrate an alternative approach to saccade-artifact removal, which we first proposed in (Reece, et al., 2009b). Our approach allows the user to encode any available information about the component signals, including smoothness, continuity at changepoints, and even the shape of the artifact signal if sufficient training data are available. In our approach, both the $\text{EEG}^*$ and EOG signals are modeled using Gaussian processes and are inferred from the EEG signal data using the fault-recovery approach.
Figure 6.15: Analysis of the EEG with saccade dataset. Predictions are made both for the latent EEG* process using (6.2.2) and for the fault contribution due to saccade using (6.2.3). The Gaussian process assumed a zero prior mean during the saccade. Top: Posterior mean retrospective predictions and pointwise ±2 standard-deviation bounds for the underlying EEG* signal. Middle: Posterior retrospective predictions for the EOG artifact. Bottom: Corresponding artifact start-time posterior produced by our algorithm. Blue indicates higher probability mass.
outlined in Section 6.2. Although the application of GPs to artifact detection in EEG signals is not new (FAUL, et al., 2007), we believe the use of GPs to actively remove artifacts and thus recover the underlying pure EEG* signal is novel.

We modeled the EEG* signal as a smooth function using a zero-mean GP with the squared exponential covariance function $K_{SE}$ (5.3.5). The EOG signal is a function that undergoes a temporary excursion from an otherwise constant value of zero; therefore, we modeled it using the drift covariance $K_{drift}$ (6.1.4). We considered two variations of the drift model when modeling the EOG artifact. These variations differed only in the prior mean that was assigned to the EOG artifact model. The first variation assumed that no information about the shape of the EOG signal was known a priori and, in this case, the EOG artifact prior mean was set to zero throughout. For the second variation of the drift model, a prior mean was learned from samples of EOG signals, giving the shape depicted in Figure 6.14. In this case, the EOG covariance function therefore modeled the residual between the EOG prior mean and the inferred signal.

The presence of abundant uncorrupted EEG signal data allowed the input- and output-scale hyperparameters for the EEG* model to be learned a priori using maximum-likelihood-estimation. We modeled the dynamics of the EOG excursion using the squared exponential covariance function $K_{SE}$ (5.3.5) and assumed that its input scale was the same as that for the EEG* data. We were therefore required to marginalize three hyperparameters: the output scale $\lambda$ of the EOG covariance, the artifact start time, and its duration. For the zero-mean fault model we selected a grid containing 13, 13, and 150 samples, respectively, for these hyperparameters. For the nonzero-mean model we used 5, 7, and 75 samples, respectively. The nonzero-mean model also requires a vertical scaling factor for the prior mean shape, and for this hyperparameter we took 9 samples.

For the artifact start-time hyperparameter, we took a uniform prior over the extent of the dataset. We cannot be very certain a priori about the duration of a saccade, which will typically be dependent on many factors. However, JÜRGENS, et al. (1981) have suggested that a reasonable prior on the logarithm of saccade duration is a Gaussian with a mean of log 110 ms) and a standard deviation of 0.6 (therefore, saccade durations of 60 ms and 200 ms are both one standard deviation from the mean). We adopted this as our prior for the artifact-duration hyperparameter.

Figures 6.15 and 6.16 show the result of performing retrospective prediction over our EEG data. Figure 6.15 shows the posterior mean and pointwise ±2 standard-deviation bounds for the artifact-free EEG* signal (calculated using (6.2.2)) and the EOG artifact (calculated using (6.2.3)) obtained by our algorithm with the zero-mean EOG model. The figure also shows the retrospective posterior distribution over the artifact start time. Although our approach was able to determine generally when the artifact started, the belief about its start time was not tightly determined. This can be explained by observing that the EOG signal output scale at the artifact onset is similar to that for the pure EEG* signal. Nonetheless, the predictions made by our approach appear plausible throughout the EOG region given the behavior before the artifact onset.

We can use (5.5.21) to produce the full predictive posterior distribution for the EEG* signal during the saccade event, and the results for the zero-mean fault model are plotted in Figure 6.17. Note that we can distinguish two competing aggregate hypotheses in the full posterior plot: the models that simply follow the EEG signal and the models that indicate a saccade artifact may be occurring. The former is characterized by a tight
Figure 6.16: Analysis of the EEG with saccade dataset. Predictions are made both for the latent EEG+ process using (6.2.2) and for the fault contribution due to saccade using (6.2.3). The Gaussian process assumed a prior mean during the saccade of the common form for EEG activity during such an event. Top: Posterior mean retrospective predictions and pointwise ±2 standard-deviation bounds for the underlying EEG+ signal. Middle: Posterior retrospective predictions for the EEG artifact. Bottom: Corresponding artifact start-time posterior produced by our algorithm. Blue indicates higher probability mass.
6.4. Results

Figure 6.17: Full retrospective posterior of the latent EEG* process for the EEG with saccade dataset, derived using (6.2.2) and (5.5.21). The Gaussian process assumed a zero prior mean during the saccade.

Figure 6.18: Full retrospective posterior of the latent EEG* process for the EEG with saccade dataset, derived using (6.2.2) and (5.5.21). The Gaussian process assumed a prior mean during the saccade of the common form for EEG activity during such an event.
distribution around the observations; the latter is much more uncertain due to the fault assumption. Note that the first hypothesis gradually loses probability mass to the second until the first becomes completely implausible under any of our models given the data.

Figure 6.16 shows the recovered signals obtained using the nonzero-mean EOG artifact model. In this case our approach more accurately identified the start and finish times of the artifact and also accurately separated the pure EEG* and EOG signals. It is interesting to note that our approach imputed a bimodal distribution over the artifact start time. The most likely start times identified by our algorithm correspond with kinks in the data at $t = 115$ ms and $t = 120$ ms, resulting in a bimodal posterior estimate of the EEG* and EOG signals. The full posterior predictive distribution is shown in Figure 6.18, which clearly shows the two competing hypotheses corresponding to these start times.
In this chapter, we will consider an extension of the work presented in the previous chapter—performing active data selection for time-series prediction in sensor networks that may be prone to changepoints and faults.

Until now, we have always assumed that the observations upon which we form our posterior beliefs were simply provided by an unnamed source, without discussing how or why those data were collected. In many scenarios, we will have control over the observation mechanisms available to us and can consider the decision problem of choosing exactly which observations to make.

The decision task of selecting the most informative observations for a system to take is called active data selection. Specifically, the active data selection paradigm seeks to choose observations that result in accurate predictions while incurring the least cost. In this chapter, we will specifically consider performing active data selection for time-series prediction tasks in sensor networks, where each sensor is capable of making observations of various correlated variables, such as the air temperature at the different sensor locations.

In our sensor-network scenario, our goals are twofold. First, we want to select observations that minimize our uncertainty about our variables of interest. Of course, the policy of simply taking every available observation will be optimal for this purpose. In practice, however, making an observation is usually associated with some cost, such as the battery energy required to power a sensor or the computational cost associated with processing additional data. For this reason, when selecting observations, we will simultaneously seek to minimize the total cost of observations required to maintain a certain acceptable level of uncertainty.

Active data selection has been the topic of a great deal of previous research; see for example (Mackay, 1992a; Seo, et al., 2000). Little previous work has focused on active data selection for time-series prediction problems. This problem was addressed using Gaussian processes by Osborne, et al. (2008). The method proposed in that work was overly simplistic, however, and would fail dramatically if applied to sensors that were prone to changepoints or faults. Our goal here will be to extend that work to manage more problematic datasets.

We will set our problem within a fully Bayesian framework. As in the previous chapter, the locations, types, and characteristics of changepoints and faults will become hyperparameters of our model. These can then be marginalized using Bayesian numerical integration to give the posterior distribution for any variable of interest, including the values of our model hyperparameters. Given our probabilistic model, we will specify the active data selection task as decision-theoretical problem and design a loss function to address the two goals outlined above.
7. ACTIVE DATA SELECTION IN THE PRESENCE OF DRASTIC EVENTS

7.1 ACTIVE DATA SELECTION FOR ROBUST PREDICTION

Before we address the specifics of our algorithm, let us more precisely define the class of situations that we will address.

Problem description

We consider performing time-series prediction on $S$ separate correlated variables. In typical sensor-network applications, these variables would correspond to the measurement of a correlated spatial field (for example, air temperature or humidity) by a set of $S$ discrete sensors. The framework we propose, however, does not require that the different variables under consideration be associated with unique sensors in distinct spatial locations. In some cases, a particular sensor might have several different modes of observation available to it; for example, a sensor might have the option of measuring a variable of interest at a higher resolution, at the expense of an increase in cost. Such situations are readily handled by our method.

We assume that the time-series prediction for each of the $S$ variables is performed using a Gaussian process. Each point in the input space $X$ for this Gaussian process can be identified with the ordered pair $(i, t)$, where $1 \leq i \leq S$ is a label identifying one of the $S$ variables and $t \in \mathbb{R}$ represents time. Therefore

$$X = \{1 \ldots S\} \times \mathbb{R}.$$  

If desired, any auxiliary features can easily be incorporated into our model; however, they will not play any role in the inference required for performing active data selection, so for ease of exposition we will ignore this detail.

To guide our ultimate decision process, we assume that each of our $S$ variables of interest has an associated positive real-valued cost of measurement $C_i \in \mathbb{R}^+$ for $1 \leq i \leq S$.

For our prediction $\text{GP}$, we choose a covariance function that decomposes into a product of separate covariance functions over the variable label and time:

$$K((i, t), (i', t')) = K_{\text{label}}(i, i') K_{\text{time}}(t, t').$$

The covariance over our variables, $K_{\text{label}}$, can be constructed by considering, for example, a function of the spatial separation between sensors. Alternatively, if the number of sensors is not too large, the covariance can be arbitrarily parametrized (Bonilla, et al., 2008; Osborne, et al., 2008). This formulation is equivalent to considering $S$ correlated one-dimensional Gaussian processes performing inference on the latent functions

$$y_i(t) \triangleq y(i, t),$$

for $1 \leq i \leq S$, where our Gaussian processes are coupled by the covariance function $K_{\text{label}}$.

We will fix the “lookahead” parameter of our $\text{GP}$ prediction algorithm to $\epsilon$ units of time into the future. That is, at a given time $t$, our ultimate goal is to predict each of the values $y_i(t + \epsilon)$ as accurately as possible. We will also allow $\epsilon$ to be zero, which corresponds to the task of simply predicting the value of the latent function associated with each of our variables at the current time. This situation is commonly called tracking.

We assume that our sensor network can be monitored and controlled by some central agent that is charged with the task of deciding which variables should be measured
and when. This agent is assumed to always have information about all the observations taken by the network up to the current time, perhaps limited to a window extending a finite time into the past.

The resulting decision problem

We now turn to the problem at hand—deciding which observations should be taken by our sensor network. Consider the decision task facing the central agent controller of the network at an arbitrary time $t$. The controller is compelled to select a (possibly empty) subset of the $S$ variables to measure. As previously discussed, our goals when making this decision are twofold: we wish to both minimize our uncertainty about our variables of interest at time $t + \epsilon$ and also to minimize the total cost of observations required to be taken. For simplicity, we limit our consideration to the singleton subsets of our variables. This allows us to avoid having to always consider the entire power set of our variables.

We can create a simple loss function to capture the uncertainty about the variables of interest. Consider an arbitrary time $t$. As usual, we define $D_{t}$ to be the set of observations available at time $t$. To represent our criterion of being as certain as possible about our variables of interest, we define the loss $\ell$ associated with having data $D_{t}$ at time $t$ as

$$
\ell(t \mid D_{t}) = \sum_{i=1}^{S} \sqrt{\text{var}(p(y_{i}(t + \epsilon) \mid t, \epsilon, D_{t}))}.
$$

Our loss function is therefore simply the sum of the uncertainty associated with each of the variables at time $t + \epsilon$; our goal is to minimize this loss function. If the monitoring of some variables is deemed more critical than others, this sum can be weighted appropriately.

We can now consider the expected loss associated with deciding to observe variable $i$ at time $t$. This observation will add the observation $(t, z_{i}(t))$ to the set $D_{t}$; however, before we make this observation, we cannot be certain what the exact value of $z_{i}(t)$ will be. Fortunately, our Gaussian process over $y_{i}$ furnishes our belief about this unknown value, and we simply marginalize it out:

$$
\mathbb{E}[\ell(t \mid z_{i}(t), D_{t})] = 
\int \int \ell(t \mid z_{i}(t), D_{t}) p(z_{i}(t) \mid y_{i}(t), D_{t}) p(y_{i}(t) \mid D_{t}) \, dz_{i}(t) \, dy_{i}(t). \tag{7.1.1}
$$

Unfortunately, this integral is nonanalytic; however, it can be effectively approximated using Bayesian numerical integration. See Section 5.5 for details.

Finally, the risk associated with measuring variable $i$ at time $t$, which we will denote $R(i, t)$, is equal to the known cost $C_{i}$ of measuring variable $i$ and the expected loss after measuring it:

$$
R(i, t) = C_{i} + \mathbb{E}[\ell(t \mid z_{i}(t), D_{t})].
$$

In this context, we can understand the observation-cost term $C_{i}$ as representing a “conversion factor” between the cost associated with making an observation and our uncertainty in the variables of interest.

---

Footnote: Because we can operate in continuous time, this is not a serious burden—if it is to our advantage to measure more than one variable at a particular time, we will instead simply measure single variables in rapid succession.
We can also consider the risk associated with not taking an observation at all, which is simply our current loss:

\[ R(\emptyset, t) \triangleq \ell(t \mid D_t). \]

Finally, we follow standard Bayesian decision theory and take the action with the lowest expected risk. Define

\[ i^*(t) \triangleq \arg \min_{i \leq S} R(i, t). \]

If \( R(i^*(t), t) < R(\emptyset, t) \), we sample variable \( i^*(t) \); otherwise, we do not sample at all.

### 7.2 Discussion

Of course, it might not desirable to have to evaluate the policy derived in the previous section at every possible time we could take a sample. Instead, after taking an observation, we can easily estimate the next time an observation will be required. In the absence of future observations, we simply use our GP prediction model to calculate the uncertainty about each variable for a reasonable range into the future and perform a simple search for when the uncertainty grows to the point where sampling will again become advantageous.

If a variable of interest might undergo the type of drastic events considered in the previous chapter, we can easily incorporate the changepoint or observation-fault models described therein. We simply construct the appropriate model and marginalize the unknown parameters when required. Notice in particular that the flexible observation models introduced in Section 6.2 can be used for the \( p(z_i(t) \mid y_i(t), D_t) \) term in (7.1.1), allowing us to properly handle potentially faulty observations. By marginalizing over the appropriate changepoint and fault models, our loss function will automatically act appropriately in the presence of such features. For example, if our model determined that a particular variable was probably in a nonrecoverable fault mode, our expected loss associated with measuring that variable would not reflect a significant improvement to the status quo; we would strongly expect that any observation we might receive to be uninformative and therefore not to significantly reduce our overall uncertainty. Our observation selection mechanism would therefore automatically guide us towards sampling more informative variables.

Note also that although in the framework presented above we make a decision at time \( t \) solely with the objective of minimizing our uncertainty about the time \( t + \epsilon \), the smoothness of typical GP covariance functions implies that our selected observations will also provide low uncertainty for previous and subsequent times.

It is worth briefly noting the difference between the loss function described above and that used in previous approaches. Osborne, et al. (2008) proposed a loss function that was infinite if the variance associated with any variable of interest become greater than a prespecified threshold. This can be a problematic choice, for example, when we have sensors that are subject to faults. Should a sensor become faulty, the uncertainty of the associated variable could easily increase beyond the chosen threshold, regardless of the samples we take. If this scenario occurred, the algorithm would request potentially useless observations from the faulty sensor constantly, which is clearly undesirable behavior.
We have applied our active data selection framework to several real datasets. Specifically, we performed prediction for our variables using (5.5.21) and used those predictions to perform active data selection as described in Section 7.1. Finally, we produced a posterior over changepoint or fault locations by estimating (5.5.22), as required.

Although we did not have active control over the sensors in question, the considered datasets contained sufficiently dense readings that we could select from the prerecorded observations without being significantly constrained by the available times of observation. The full list of all available observations additionally served as a measure of “ground truth” to verify the accuracy of our predictions, except, of course, where the true recorded observations were faulty.

*Tide height with stuck sensor*

For the first test of our approach, we returned to the Sotonmet tide-height sensor data first presented in Section 6.4.

We chose a Gaussian process model that was identical to the one described in 6.4, but was augmented with an additional hyperparameter specifying fault length. This prior over this hyperparameter and its associated samples informed the model about typical fault length, allowing it to model the probability that a fault was ongoing and when it might end. We chose a Gaussian prior over the natural logarithm of fault length with a mean of \( \log(12 \text{ hr}) \) and standard deviation of 0.5. A grid of hyperparameter samples was used for the purpose of Bayesian Monte Carlo hyperparameter marginalization with 11 samples for fault length and 100 samples for fault onset time. The prior over fault onset time was uniform over the input range. The cost of observing the tide was fixed at 0.075 m.

Figure 7.1 demonstrates active data selection for this dataset through the faulty period. Our algorithm correctly detected the stuck-value fault beginning at around \( t = 0.8 \) days and reduced its sampling frequency as a result. More observations were selected as it began to suspect that the fault might have ended. Accurate predictions with realistic uncertainties were made throughout. The somewhat unusual shape of the posterior fault distribution is an artifact of both the window size and the very long prior over fault length— with little informative data, it is very hard to differentiate between the various fault models, many of which suggest a long fault duration. If a fault-length prior emphasizing shorter faults were to be used, a quick decay in the fault posterior would be observed; however, the prior used in this experiment was justifiable given the situation due to the natural period of tide-height fluctuations.

*Wannengrat weather sensor network*

Our second dataset was drawn from the Wannengrat Alpine Observatory being built above and around the town of Davos, Switzerland. The observatory is part of an environmental-monitoring project called Swiss Experiment.\(^2\) The observatory comprises a wireless sensor network deployed with the goal of understanding the complex meteorological processes occurring on the snowpack surrounding Davos. These processes result in multiple changepoints within the collected data, suggesting that the

\(^2\)See http://www.swiss-experiment.ch/.
Figure 7.1: Demonstration of active data selection for the stuck tide-height sensor example. Top: Posterior mean predictions and pointwise ±2 standard-deviation bounds for the tide height through time. The entire dataset is displayed in light gray, with observations selected by our algorithm shown in black. Bottom: Corresponding posterior distribution for the probability that a particular time belongs to a faulty region.

changepoint-related methods presented in the previous chapter might be useful. In addition, the deployed remote sensors have a limited battery life, suggesting that the network could benefit from active data selection. For testing purposes, we specifically performed active data sampling on a dataset containing ambient temperature measurements from 16 sensors over a period of time that featured a dramatic changepoint.

For our GP model over ambient temperature, we set the covariance over sensors, $K_{\text{label}}$, to be a function of the isotropic spatial distance between the known sensor locations. The Matérn covariance function $K_M$ with $\nu = \frac{3}{2}$ (5.3.7) was chosen for this purpose. To the covariance over time, $K_{\text{time}}$, we assigned another Matérn covariance with $\nu = \frac{3}{2}$ that was modified to allow for changepoints in output scale using the changepoint covariance $K_{\text{CO}}$ (6.1.8). Our hypersamples lay on a grid with 50 samples for changepoint location, 7 samples each for the output scale before and after a putative changepoint, 5 samples in the input scale, and 3 samples in the noise variance. The cost of observing any variable was fixed at 0.5°C. Although this cost is slightly higher than would be realistic given the actual batteries used in these sensors, this choice did allow us to produce visually intelligible plots.

Figure 7.2 demonstrates active data selection over this dataset. At the onset, our algorithm selected two initial samples from each sensor at nearly identical times, which
7.3. Results

Figure 7.2: Demonstration of active data selection for the Wannengrat sensor-network example. The top two plots show posterior mean predictions and pointwise ±2 standard-deviation bounds for the ambient temperature through time corresponding to sensors 13 and 16, respectively, from top to bottom. The entire dataset for each sensor is displayed in light gray, with observations selected by our algorithm shown in black. The bottom plot displays the temporal locations of the observations selected by our algorithm for every sensor in the network over the same time period.
served to educate the model about the correlations between the sensors. These correlations were subsequently exploited by taking a more dispersed pattern of observations—it is wasteful to take observations from two sensors at similar times if we believe that their readings are strongly correlated. There was a dramatic increase in sampling frequency coincident with the volatile fluctuations in temperature that began at about $t = 0.7$ days. When the fluctuations subsequently dissipated at about $t = 0.9$ days, the rate of sampling dropped again.

**EEG data with saccade event**

Finally, we return to the EEG with saccade dataset first presented in Section 6.4. We considered the problem of performing active data selection on this dataset while performing online prediction. The GP model used for this purpose was the same as the one used in Section 6.4. In contrast to the application described in that section, however, we performed online rather than offline prediction. The typical EOG drift function displayed in Figure 6.14 was chosen for the prior mean function for the drift component. The cost of observation was fixed at 0.1.

Figure 7.3 displays the results. Sampling was reduced during the fault, when observations became more (but not completely) uninformative.
We will now temporarily shift our focus from learning in the presence of concept drift to consider the problem of function optimization. In the next chapter, we will apply the ideas developed here for general optimization to an objective function that changes over time, resuming the focus on nonstationary data.

Optimization has been an important area of mathematical study since the creation of the digital computer. A great deal of research has been devoted to solving dozens of subproblems in this field (Nocedal and Wright, 2006), with application areas ranging from economics to mechanical design. In this chapter, we will approach global optimization from the viewpoint of Bayesian probability theory, framing the procedure as a sequential decision problem.

We will specifically consider finding the global optimum of an expensive-to-evaluate objective function in a given region. Many real-world optimization problems involve expensive objective functions; for example, evaluating the objective might require performing a lengthy, complex computer simulation or an irreversible, expensive physical action. The cost associated with evaluating this function naturally compels us to select the location of each new trial location very carefully. Our eventual task is to return a final point in the domain where we believe the objective function to be minimized. This ultimately straightforward goal of optimization will allow us to define a simple and natural loss function that captures the utility of the decisions made by a particular optimization method.

We will use a Gaussian process to model the objective function, which we will condition on the observations we receive throughout the optimization process. With this model, we can derive an analytic expression for the expected loss associated with evaluating the objective at a given candidate point, using a limited approximation. We can also find simple closed-form expressions for the gradient and the Hessian of the expected loss surface, rendering its minimization much simpler and less costly than the original objective.

The resulting algorithm is similar to a previously proposed optimization procedure employing Gaussian processes, the expensive global optimization (EGO) algorithm (Jones, et al., 1998). However, we will improve upon the EGO framework in several ways. First, we can increase our performance on particularly expensive objectives by considering multiple function evaluations into the future. Our clear Bayesian formalism will also permit us to benefit from specifying the required confidence in our returned value. Further, our use of Gaussian processes will allow us to incorporate useful prior information about the objective function (such as periodicity) and also to learn from observations of the derivative. The ability to learn from derivative observations will lead to an innovative mechanism for addressing numerical conditioning issues. Finally, we will extend our method to handle the optimization of functions from noisy observations.
8.1 Optimization Using Gaussian Processes

As discussed in Chapter 5, Gaussian processes offer a powerful method for performing Bayesian inference about functions. Such inference is at the heart of optimization, made explicit by techniques that employ response surfaces or surrogates (Jones, 2001). Our goal is to build a statistical picture of the objective function's overall form given our observations of it, then exploit the structure in this model to guide us through our task.

Suppose that we are trying to minimize an objective function \( y : \mathcal{X} \rightarrow \mathbb{R} \) over the domain \( \mathcal{X} \) and that we have chosen a Gaussian process prior for \( y \),

\[
p(y \mid \theta) \triangleq GP(y; \mu, K),
\]
as well as an appropriate prior \( p(\theta) \) for the hyperparameters of our model. For the time being, we will assume that we can evaluate \( y \) directly without contamination by observation noise. We will consider the optimization of noise-corrupted objective functions later.

We will consider the minimization of \( y \) in the context of Bayesian decision theory. In general, any optimization process has the same basic form. Points in the domain \( \mathcal{X} \) are chosen one after another according to some criterion, and the objective function is evaluated at those points. The procedure terminates when some stopping criterion is met, at which point it is compelled to return a single final point

\[
(x_f, y(x_f)) \in \mathcal{X} \times \mathbb{R}.
\]
The utility of the returned point is judged solely by the value \( y(x_f) \)—the lower, the better.

Suppose that an optimization routine terminates after having made \( N \) total observations of \( y \), which we will denote with

\[
\mathcal{D}_N \triangleq \{(x_i, y_i)\}_{i=1}^N \triangleq (X_N, y_N),
\]
where \( y_i \triangleq y(x_i) \). For \( M \leq N \), we will similarly write \( \mathcal{D}_M \) to represent just the first \( M \) of these observations. To simplify the present discussion, we will insist that the routine choose from only among the observed points \( \mathcal{D}_N \) when selecting which point to return. We will discuss a relaxation of this condition later. In light of our assertion that an optimization procedure is to be judged only by the lowest value of the objective found, we define the loss associated with the chosen observations \( \mathcal{D}_N \) to be

\[
\ell(\mathcal{D}_N) \triangleq \min_{i \leq N} y_i.
\]

(8.1.1)

For \( 1 \leq k \leq N \), we will abbreviate \( \ell(\mathcal{D}_k) \) with \( \ell_k \).

We will now demonstrate how to derive the appropriate Bayesian decision rule for the optimization process given our Gaussian process model. We will first consider a simple one-step lookahead procedure.

One-step lookahead

To begin, imagine that we are only allowed one more function evaluation before we must report our inferred function minimum. Suppose \( \mathcal{D}_{N-1} \) are the observations gathered thus far. We must choose a point \( x_N \in \mathcal{X} \) where we would like to evaluate \( y \) for the
To continue with our evaluation of \( \ell_N \), we substitute the approximation above for \( y \) where the weights \( \Lambda \) are defined in (8.1.2).

Of course, we cannot know \( y_N \) exactly before evaluating it. Given our \( \text{GP} \) model over \( y \), however, we can determine an analytic form for the expected loss after selecting a particular point \( x \) for \( x_N \), assuming that we have only one evaluation remaining. We will denote the expected loss associated with choosing \( x \) for \( x_N \), given the observations \( D_{N-1} \), with \( \Lambda_N(x | D_{N-1}) \). We can calculate this value by marginalizing out the unknown value \( y \equiv y(x) \):

\[
\Lambda_N(x | D_{N-1}) = \mathbb{E}[\ell_N] = \int \ell_N p(y | x, D_{N-1}) \, dy.
\]  

In Chapter 5, we discussed how to approximate the \( p(y | x, D_{N-1}) \) term in (8.1.2); the solution was to choose a set of hyperparameter samples \( \Psi \) and approximate the marginal distribution with a weighted mixture:

\[
p(y | x, D_{N-1}) = \frac{\int p(y, x, D_{N-1}, \theta) p(y | x, D_{N-1}, \theta) p(\theta) \, d\theta}{\int p(y | x, D_{N-1}, \theta) p(\theta) \, d\theta} 
\approx \sum_{\psi \in \Phi} w_{\psi} \mathcal{N}(y; m_{\psi}(x; \psi), C_{\psi}(x; \psi)),
\]

where the weights \( w \) are defined in (5.5.20). To condense the following abbreviations, we define the following abbreviations:

\[
m_{\psi} = m_{\psi}(x; \psi),
C_{\psi} = C_{\psi}(x; \psi).
\]

To continue with our evaluation of (8.1.2), we substitute the approximation above for \( p(y | x, D_{N-1}) \):

\[
\Lambda_N(x | D_{N-1}) = \int \ell_N p(y | x, D_{N-1}) \, dy 
\approx \int \ell_N \left( \sum_{\psi \in \Phi} w_{\psi} \mathcal{N}(y; m_{\psi}, C_{\psi}) \right) \, dy 
= \sum_{\psi \in \Phi} w_{\psi} \int \ell_N \mathcal{N}(y; m_{\psi}, C_{\psi}) \, dy.
\]  

---

8.1. Optimization Using Gaussian Processes

last time. After the point \( x_N \) has been selected, we will augment our data \( D \) with the observation \( (x_N, y_N) \). Given this observation, our final loss \( \ell_N \) can be evaluated trivially, as it is simply the new smallest observation seen:

\[
\ell(D_N) = \min(\ell_{N-1}, y_N) = \begin{cases} 
\ell_{N-1} & y_N \geq \ell_{N-1}; \\
y_N & y_N < \ell_{N-1}.
\end{cases}
\]

Of course, we cannot know \( y_N \) exactly before evaluating it. Given our \( \text{GP} \) model over \( y \), however, we can determine an analytic form for the expected loss after selecting a particular point \( x \) for \( x_N \), assuming that we have only one evaluation remaining. We will denote the expected loss associated with choosing \( x \) for \( x_N \), given the observations \( D_{N-1} \), with \( \Lambda_N(x \mid D_{N-1}) \). We can calculate this value by marginalizing out the unknown value \( y \equiv y(x) \):

\[
\Lambda_N(x \mid D_{N-1}) = \mathbb{E}[\ell_N] = \int \ell_N p(y \mid x, D_{N-1}) \, dy.
\]  

In Chapter 5, we discussed how to approximate the \( p(y \mid x, D_{N-1}) \) term in (8.1.2); the solution was to choose a set of hyperparameter samples \( \Psi \) and approximate the marginal distribution with a weighted mixture:

\[
p(y \mid x, D_{N-1}) = \frac{\int p(y, x, D_{N-1}, \theta) p(y \mid x, D_{N-1}, \theta) p(\theta) \, d\theta}{\int p(y \mid x, D_{N-1}, \theta) p(\theta) \, d\theta} 
\approx \sum_{\psi \in \Phi} w_{\psi} \mathcal{N}(y; m_{\psi}(x; \psi), C_{\psi}(x; \psi)),
\]

where the weights \( w \) are defined in (5.5.20). To condense the following abbreviations, we define the following abbreviations:

\[
m_{\psi} = m_{\psi}(x; \psi),
C_{\psi} = C_{\psi}(x; \psi).
\]

To continue with our evaluation of (8.1.2), we substitute the approximation above for \( p(y \mid x, D_{N-1}) \):

\[
\Lambda_N(x \mid D_{N-1}) = \int \ell_N p(y \mid x, D_{N-1}) \, dy 
\approx \int \ell_N \left( \sum_{\psi \in \Phi} w_{\psi} \mathcal{N}(y; m_{\psi}, C_{\psi}) \right) \, dy 
= \sum_{\psi \in \Phi} w_{\psi} \int \ell_N \mathcal{N}(y; m_{\psi}, C_{\psi}) \, dy.
\]  

---

103
Due to the simple nature of the loss function $\ell(D_N)$, we can calculate each term in (8.1.3)—which we will denote $\Lambda_1(x_\ast \mid D_{N-1}, \psi)$—analytically:

$$
\Lambda_1(x_\ast \mid D_{N-1}, \psi) \\
= \int \ell_N N(y_\ast; m_\psi, C_\psi) \, dy_\ast \\
= \int \min(\ell_{N-1}, y_\ast) N(y_\ast; m_\psi, C_\psi) \, dy_\ast \\
= \int_{-\infty}^{\ell_{N-1}} \ell_{N-1} N(y_\ast; m_\psi, C_\psi) \, dy_\ast + \int_{-\infty}^{\ell_{N-1}} y_\ast N(y_\ast; m_\psi, C_\psi) \, dy_\ast \\
= \ell_{N-1} + (m_\psi - \ell_{N-1}) \Phi(\ell_{N-1}; m_\psi, C_\psi) - C_\psi N(\ell_{N-1}; m_\psi, C_\psi),
$$

where $\Phi$ is the cumulative distribution function of the Gaussian distribution (5.5.19).

Our final expression for the expected loss function now becomes

$$
\Lambda_1(x_\ast \mid D_{N-1}) \approx \sum_{\psi \in \Psi} w_\psi \Lambda_1(x_\ast \mid D_{N-1}, \psi). \tag{8.1.4}
$$

Note that the “expected improvement” function defined by Jones, et al. (1998) is similar to but differs from this Bayesian expected loss criterion.

The location where our expected loss is lowest gives the optimal location for our next function evaluation. Note that (8.1.4) decreases as $m_\psi$ becomes lower than $\ell_{N-1}$ and also as $C_\psi$ increases. The first property ensures exploitation of the objective function; the second, exploration. The minimization of the expected loss therefore gives a natural balancing of these two concerns.

To select the final point $x_N$, we minimize the expected loss function $\Lambda_1(x_\ast \mid D_{N-1})$ over the domain $\mathcal{X}$ and take the action with the lowest expected loss:

$$
x_N \doteq \arg \min_{x_\ast \in \mathcal{X}} \Lambda_1(x_\ast \mid D_{N-1}). \tag{8.1.5}
$$

Of course, we have merely shifted the minimization problem from one over the objective function $y$ to one over the expected loss function $\Lambda_1$. Fortunately, the expected loss function is continuous and computationally inexpensive to evaluate. Analytic expressions for its gradient and Hessian can also be easily derived. There are a range of off-the-shelf optimization routines suitable for this minimization.

Note that the decision rule described in this section can also be used as an approximation in a situation where we actually have many evaluations remaining. At every step of the optimization process, we simply act as if we had only one evaluation remaining and use the one-step lookahead rule in (8.1.5) to select the next point to evaluate. Essentially, we assume that we are so uncertain about the potential effect of our choices on future decisions that we can simply pretend that those remaining evaluations did not exist. Fortunately, such an approximation typically provides good results in practice, which we will demonstrate later. Algorithm 8.1 provides pseudocode for $N$ steps of this one-step lookahead optimization procedure.

Figure 8.1 illustrates the application of this procedure to minimize a simple one-dimensional objective function. Notice in particular the automatic tradeoff between exploration and exploitation—evaluations 3–4 and 7–8 exploit discovered local minima; the others explore unknown space. Also notice how the weights on hyperparameter
Algorithm 8.1: Pseudocode for the one-step lookahead GP-GO algorithm.

\[ \text{Input:} \ \text{objective function } y, \ p(y \mid \theta), \ p(\theta); \ \text{input space } \mathcal{X}, \ \text{initial point } x_0, \text{ number of evaluations } N \]

\[ \text{Output:} \ \text{suspected optimum point } (x_f, y_f) \in \mathcal{X} \times \mathbb{R} \]

\[ D_0 \leftarrow \emptyset \]

\[ \text{for } i \leftarrow 1 \text{ to } N \text{ do} \]

\[ \text{ evaluate } y(x_i) \]

\[ D_i \leftarrow D_{i-1} \cup \{x_i, y(x_i)\} \]

\[ \text{ for } \psi \in \Psi \text{ do} \]

\[ p(y \mid D_i, \psi) \leftarrow \text{CONDITION-GP} \left( p(y \mid D_{i-1}, \psi), D_i \right) \]  

\[ \text{ end} \]

\[ x_{i+1} \leftarrow \text{MINIMIZE-EXPECTED-LOSS} \left( p(y \mid D_i), p(\theta) \right) \]  

\[ \text{ end} \]

\[ x_f \leftarrow \arg \min_{x \in D_N} y(x) \]

\[ \text{return } (x_f, y(x_f)) \]

---

8.1. Optimization Using Gaussian Processes

samples are updated automatically over time to reflect the information contained in our growing observation set. By the end of the algorithm, the routine learned that both the input and output scales were larger than their priors indicated. This is reflected, for example, in the correspondingly larger pointwise standard deviation seen in the objective-function posterior during later function evaluations.

Multiple-step lookahead

Although the one-step lookahead rule described above works well in practice, proper probabilistic reasoning allows us to relax this short-sighted assumption if required. Let us consider the situation where we have \( n \) observations remaining and must select the location of the next observation while taking into account all future observations. We must therefore simultaneously consider both the selection of the point \( x_{N-n+1} \) and also the effect of this decision on our future choices. As before, we will represent the \( n \)-step lookahead expected loss associated with choosing \( x_* \) for \( x_{N-n+1} \), given the observations \( D_{N-n} \), by \( \Lambda_n(x_* | D_{N-n}) \). We can express this function probabilistically in a straightforward manner—we simply take the expectation of \( \ell_N \) while marginalizing out the effects of remaining choices, as well as the unknown value \( y_* \):

\[
\Lambda_n(x_* | D_{N-n}) = \int \cdots \int \ell_N p(y_* | x_*, D_{N-n}) \times \left( \prod_{i=N-n+2}^{N} p(y_i | x_i, D_{i-1}) p(x_i | D_{i-1}) \right) \, dy_* \, dy_{N-n+2} \cdots dy_N \, dx_{N-n+2} \cdots dx_N. \tag{8.1.6}
\]

This integral is unfortunately both unwieldy and nonanalytic; however, we can make an approximation using a recursive sampling approach. To illustrate, consider the problem...
Figure 8.1: A demonstration of the gpgo algorithm. Left: The current belief over the objective function, $p(y \mid \mathcal{D}_i)$, the one-step lookahead expected loss surface, $\Lambda_1(x_i \mid \mathcal{D}_i)$, and the chosen point of the next evaluation, $x_{i+1}$. Right: Contour plot showing the evolution of relative hyperparameter sample weights for the input scale $\sigma$ and the output scale $\lambda$. 

8. GAUSSIAN PROCESSES FOR GLOBAL OPTIMIZATION

---

**Diagram Description:**
- **Objective Function:** Dashed line
- **Mean:** Solid line
- **1σ:** Blue shaded area
- **Expected Loss:** Red shaded area
- **Observations:** Black dots
- **Next Evaluation:** Red dot

**Axes:**
- **Vertical Axis:** $\lambda$
- **Horizontal Axis:** $\sigma$
8.1. Optimization Using Gaussian Processes

![Figure 8.1: Continued.](image-url)
Figure 8.1: Continued.
8.2. Other Considerations

of evaluating $\Lambda_2(x, D_{N-2})$. We begin by sampling $y_*$ from $p(y_* | x_*, D_{N-2})$; call the sampled value $\hat{y}_{N-1}$. We now augment $D_{N-2}$ with the observation $(x_*, \hat{y}_{N-1})$; call this new observation set $\tilde{D}_{N-1}$. Given this observation, we may choose $x_N$ using (8.1.5). Fortunately, we can calculate $\Lambda_1$ analytically using (8.1.4). After selecting $x_N$, the value $\Lambda_1(x_N | \tilde{D}_{N-1})$ represents the expected final loss given the data $\tilde{D}_{N-1}$. Of course, we are uncertain about the sampled value $\hat{y}_{N-1}$, so we continue this combined sampling and optimization procedure as long as we desire and finally approximate (8.1.6) using numerical integration. Now that we know how to evaluate $\Lambda_2$, we may evaluate $\Lambda_3$ in an analogous fashion and can continue recursively as deep as desired.

Note that each of the intermediate loss functions $\Lambda_{n-1}$ is also integrals over $\theta$. These can be approximated as weighted sums over hyperparameter samples using (5.5.20), with weights that will differ from each other due to the different sets of information $\tilde{D}_{N-n+1} \ldots \tilde{D}_{N-1}$.

Unfortunately, only for $\Lambda_1$ can the required minimization step benefit from analytic derivatives. Considering this fact and the degree of sampling required, it is clear that multiple-step lookahead demands significantly more computational resources than the one-step lookahead rule described in the previous section. However, this cost might be justified if the function $y$ we want to minimize is itself correspondingly expensive to evaluate. Given a particular problem of interest, we simply need to select an appropriate value for $n$ by weighing the potential increase in performance from using multiple-step lookahead against both the cost of increased computation time and the cost of evaluating the objective.

8.2 OTHER CONSIDERATIONS

The framework described above can be extended in several useful ways, which we will now discuss.

Addressing conditioning problems

A common problem that arises when using Gaussian processes is poor numerical conditioning in covariance matrices, which can occur when we have observations that are very strongly correlated. For example, when using a typical stationary covariance function (5.3.4), conditioning on highly proximate observations can lead to poorly conditioned covariance matrices. This is a particular concern in the context of optimization, where we commonly want to take several observations close to a minimum. This problem is ameliorated slightly by our proposed procedure, which by design recognizes that sampling close to an existing observation rarely produces much new information.

As discussed in Section 5.3, rather than inverting the covariance matrix directly, we find its Cholesky decomposition. This gives us an improved degree of tolerance to conditioning problems. The Moore–Penrose pseudoinverse (which sets small singular values in the singular value decomposition of the covariance matrix to zero) can also be used, but is prohibitively slow to compute. In our testing, we used a simple heuristic to guarantee good conditioning. Each new function observation was required to be sufficiently far from all existing observations, using a prespecified minimum separation $\delta$.

109
Derivative observations

It is trivial to incorporate derivative observations into a Gaussian process (Rasmussen and Williams, 2006). Because differentiation is a linear operator, a function over which we have a Gaussian process distribution is jointly Gaussian-process distributed with all of its derivatives. This allows us to treat observations of \( y', y'', \) etc., exactly like observations of \( y \) itself, using an appropriately modified covariance function. Suppose we have a function \( y: \mathbb{R}^d \rightarrow \mathbb{R} \) and have chosen the prior

\[
p(y | \theta) \equiv \mathcal{G}\mathcal{P}(y; \mu, K),
\]

where \( K \) is differentiable. In this case, we may calculate the covariance between a function value and a partial derivative value, as well as between two partial derivative values. If \( x, x' \in \mathbb{R}^d \) and \( 1 \leq i, j \leq d \), then:

\[
\begin{align*}
\text{cov}\left( \frac{\partial y(x)}{\partial x_i}, y(x') ; \theta \right) &= \frac{\partial K(x, x'; \theta)}{\partial x_i}, \\
\text{cov}\left( \frac{\partial y(x)}{\partial x_i}, \frac{\partial y(x')}{\partial x_j} ; \theta \right) &= \frac{\partial^2 K(x, x'; \theta)}{\partial x_i \partial x_j}.
\end{align*}
\]

Such observations, when available, can therefore be readily incorporated into our optimization procedure. This approach has been incorporated into the ego framework by Leary, et al. (2004).

However, we have another use for derivatives. Covariance matrices over derivative observations are typically better conditioned than those over observations of the function itself; see Figure 8.2 for reference. Note that a derivative value at a particular point is only weakly correlated with the latent function values nearby, but does provide useful information about the function at more remote locations. Furthermore, when compared to function observations, derivative observations are more weakly correlated with each other; therefore, derivatives can be observed closer to each other at the same conditioning "cost." Although the illustration in Figure 8.2 uses the squared exponential covariance function \( K_{SE} \) (5.3.5), similarly shaped covariance functions, such as the rational quadratic (5.3.8) and Matérn (5.3.6) classes, behave in the same manner.

Given these facts, we can conclude that our covariance matrix would be better conditioned if we had taken derivative observations in the place of function observations. We now return to the numerical conditioning issue discussed in the previous section and propose a useful mechanism to sidestep problems caused by very close observations. In the previous section, we proposed simply ignoring any observation closer than a threshold \( \delta \) to another existing observation. Rather than completely ignoring these measurements, however, we can replace them with approximate derivative observations. These observations will continue to improve our model of the objective function while avoiding conditioning problems in the covariance matrix.

Suppose that we take a function evaluation \( (x_i, y_i) \) that is closer than \( \delta \) to an existing observation at \( (x_j, y_j) \). As before, we do not include the observation \( (x_i, y_i) \) into our covariance matrix. In its place, we include partial derivative observations that force the gradient at the midpoint of these two points to equal the natural linear approximation suggested by the two observations. For \( 1 \leq k \leq d \), if \( |x_{j,k} - x_{i,k}| > 0 \), we include the
“pseudoderivative” observation

$$\frac{\partial}{\partial x_k} \gamma \left( \frac{x_i + x_j}{2} \right) \approx \frac{y_j - y_i}{|x_j,k - x_i,k|};$$

if $|x_j,k - x_i,k| = 0$, we ignore dimension $k$. Because $x_i$ and $x_j$ lie close to each other, we can expect to lose little information when making this linear approximation. This procedure is demonstrated in Figure 8.3. Of course, we should not discard the knowledge of $y_j$ altogether—it can be used to impute future pseudoderivative observations, as illustrated in Figure 8.3, where the knowledge of $y_4$ helped us determine the pseudoderivative that replaced evaluation 5. We can also continue to use the measured value $y_j$ to compute the loss $\ell$. The proposed use of pseudoderivative observations to replace poorly separated function observations gives rise to much better numerical conditioning while discarding very little information.

**Confidence in output**

Note that even if we do not sample exactly at a minimal value of $y$, its location will often be evident in light of other nearby observations. A natural possibility to consider is to allow the algorithm to report a suspected, rather than actually evaluated, minimum at termination.

We therefore relax the constraint that the final returned value $x_f$ be one of the observed locations $X$. Instead, we merely require that we be reasonably certain about the value of the objective function at the chosen point. Probabilistically, we require
Figure 8.3: An illustration of the use of pseudoderivative observations to replace the sometimes excessively close function observations required to locate a minimum. Both function evaluations 4 and 5 are replaced by appropriately calculated pseudoderivative observations.
that the standard deviation of our belief about \( y_f \pm y(x_f) \) be no larger than a selected threshold \( \varepsilon \):

\[
\sqrt{\text{var}(\Pr(y_f \mid x_f, D_N))} \leq \varepsilon.
\]

Essentially, \( \varepsilon \) quantifies exactly how confident we need to be in the returned value \( y_f \). If \( \varepsilon \) is sufficiently small, typically only points very close to the actually evaluated points \((X,y)\) will be eligible to be returned by the algorithm; as \( \varepsilon \) grows larger, points where we are increasingly uncertain about the objective function become eligible for return.

To realize this suggestion, we modify our loss function \( \ell \) slightly. We define

\[
\ell'(D_N, \psi) \triangleq \min_{\psi < z} (m_{\psi}).
\]

Notice this loss function is now dependent on the value of our hyperparameters. If desired, we can trivially modify the decision rules described above to utilize this redefined loss function.

This change can improve the performance of our algorithm by eliminating the potential cost of evaluating the objective simply to return a value that we are already very confident about. As an example, consider the minimization of a function known \textit{a priori} to be quadratic. Using the appropriate covariance function, any three observations will exactly determine where the global minimum lies. The standard loss function \( \ell \) would compel us to nonetheless evaluate the function at that point. The modified loss function \( \ell' \), on the other hand, would allow us to end the procedure immediately—no point in the domain could possibly improve the loss \( \ell' \), because we are already completely confident about the value of the objective everywhere. We can conclude that no further observations are necessary and simply return the known minimum.

### 8.3 Noisy Optimization

Many applications require an effective means of noisy optimization (Carter, et al., 2001). In such problems, rather than observing the objective function \( y(x) \) directly, we observe a noise-corrupted version \( z(x) \) instead. We described how to incorporate noise into the observation function \( z \) in Sections 5.2 and 6.2.

In this situation, the above discussion about confidence in returned values becomes important. If the noise is considerable, for example, we may in fact need to take several function evaluations at, or very near, the same point before we become sufficiently confident in the function value there.

When performing noisy optimization, our ultimate loss remains the lowest function value known with sufficient confidence—there is no need for it to be subjected to any heuristic modifications (Huang, et al., 2006). We may therefore successfully apply the modified loss function \( \ell' \) described above.

In the noisy case, our data are of the form

\[
D_N \triangleq \{(x_i,z_i)\}_{i=1}^N \triangleq (X_N,z_N),
\]

where \( z_i \triangleq z(x_i) \). The one-step lookahead expected loss becomes

\[
\Lambda'(x_\ast \mid D_{N-1}) \triangleq \mathbb{E}[\ell'(D_N, \theta)]
= \int \int \ell'(D_N, \theta) p(z_\ast \mid x_\ast, \theta, D_{N-1}) p(\theta) \, dz_\ast \, d\theta,
\]
and the multiple-step lookahead expected loss becomes

\[ \Lambda'_n(x_\star | D_{N-n}) = \int \int \cdots \int \ell(D_n, \theta) p(z_\star | x_\star, D_{N-1}, \theta) p(\theta) \times \]

\[ \times \left( \prod_{i=N-n+2}^N p(z_i | x_i, D_{i-1}, \theta) p(x_i | D_{i-1}, \theta) \right) \]

\[ dz_\star d\theta dz_{N-n+2} \cdots dx_N dx_{N-n+2} \cdots dx_N. \]

As before, we must resort to numerical methods to evaluate these integrals. Unfortunately, the analytic result in (8.1.2) does not hold in the noisy case, due to the nature of the loss function \( \ell' \). If the noise is not too large relative to the scale of variation of the objective function, however, (8.1.2) will continue to be effective. Under this assumption, the one-step lookahead expected loss becomes

\[ \Lambda'_n(x_\star | D_{N-1}) \approx \sum_{\psi \in \Psi} w_\psi \Lambda'_n(x_\star | D_{N-1}, \psi), \]

(8.3.1)

where

\[ \Lambda'_n(x_\star | D_{N-1}, \psi) \approx \ell'(D_{N-1}, \psi) \]

\[ + (m_\psi - \ell'(D_{N-1}, \psi)) \Phi(\ell'(D_{N-1}, \psi); m_\psi, C_\psi) \]

\[ - C_\psi \mathcal{N}(\ell'(D_{N-1}, \psi); m_\psi, C_\psi). \]

8.4 Results

We have compared the results of our method, which we will call Gaussian processes for global optimization (GPGO), with several proposed alternatives on an extensive set of standard test functions for global optimization.

Testing methodology

To compare the various chosen optimization methods, we selected a diverse list of 14 standard test functions for global optimization (Molga and Smutnicki, 1995), which are listed in Table 8.1. Each function has a commonly used “standard” box-shaped test region containing the (not necessarily unique) global minimum,\(^1\) which is also presented in the table. Using these functions, we compared the performance of our algorithm with other optimization methods both in the noiseless case (where the objective could be observed exactly) and the noisy case (where the objective was corrupted by noise).

For noiseless testing, to reduce the possibility of an algorithm accidentally “stumbling” upon an optimal value rather than systematically finding it, we adjusted the test functions slightly. For each problem, we created ten associated subproblems by randomly translating the standard test region, guaranteeing that all global minima remained within the box.\(^2\) This modification to the testing procedure allowed us to

\(^1\)Note that our algorithm is not inherently limited to box-bounded problems; we can impose any arbitrary constraint by setting the expected loss to \(+\infty\) in the region outside the desired domain. The problem of constrained optimization is then transferred to the routine that minimizes the expected loss function.

\(^2\)For multiple-step lookahead problems, time restricted us to testing two each rather than ten.
compare the average performance of each method on each problem. Translating the subproblems also increased the difficulty of some test functions, because several of them are less well-behaved outside their standard test domain. A similar tactic was used to reduce sample bias during noisy testing—for each test problem and noise level, three different realizations of noise were used, and the performance of each optimization method was determined from its aggregate performance.

We also tested the noiseless optimization routines on two- and three-dimensional functions generated from the GKLS random-function generator (GAVIANO, et al., 2003). The default parameters were used, except the number of local minima was increased from ten to twenty. The first ten generated functions for each set of parameters formed the test set.

Because we are specifically interested in optimizing expensive-to-evaluate objectives, we limited the number of allowed function evaluations for each optimization method. For noiseless problems, we allowed $10d$ observations of the objective function, where $d$ is the dimension of the input domain; for noisy problems, we allowed $20d$. This restriction also applied to any initial function evaluations required to infer hyperparameters, if prescribed by the procedure under consideration. For very expensive functions, limiting the number of function evaluations is a natural stopping criterion.$^5$

We used the “gap” performance measure to compare the methods (HUANG, et al., 2006):

$$ G = \frac{y(x_i) - y(x_f)}{y(x_i) - y(x_{opt})}, $$

where $x_i$ is the location of the first evaluation of $y$, $x_f$ is the location of the point returned by the optimization method, and $x_{opt}$ is the location of a global optimum. $G$ therefore represents the portion of the “gap” between the value of the objective function at the initial point and its optimal value that was overcome by the algorithm at its termination. For every method tested, the initial point $x_i$ was set to the midpoint of the test region under consideration.

---

$^5$Other stopping criteria are of course possible, such as stopping when the expected reduction in loss drops below a certain level, where this loss might additionally include the cost of evaluating the function.
Implementation details

For testing the GPGO algorithm, we took a Gaussian process prior over the objective function,

\[ p(y \mid \theta) \cong \mathcal{GP}(y; \mu, K), \]

where the prior mean \( \mu \) was the simple constant zero function \( \mu(x) = 0 \). The covariance
K was the sum of two components:

\[ K(x, x'; \theta) \cong K_{SE}(x, x'; \lambda_{SE}, \sigma_{SE}) + K_{P}(x, x'; \lambda_{P}, \sigma_{P}, \gamma_{P}), \]

where \( K_{SE} \) is the squared exponential covariance function (5.3.5) and \( K_{P} \) is the higher-
dimensional analog of the periodic covariance function given in (5.3.9) with \( |x - x'| \) replaced
with an appropriately chosen Mahalanobis distance \( d_{M}(x, x'; A) \) (5.3.2). This covariance was intended to allow us to model functions that are the superposition of a nonperiodic and a periodic component. For functions known to have periodic components, the full covariance function was used; otherwise, only the nonperiodic component was used by setting \( \lambda_{P} = 0 \).

Fortunately, there are a wide variety of other covariance functions, should a problem call for them; see Chapter 5 and (Rasmussen and Williams, 2006). Note also that we do not necessarily need to place our GP distribution on the objective function directly—an objective known to be strictly positive, for example, might benefit from placing a GP over its logarithm. Other transformations can always be considered to improve the performance on a particular function.

We also used the sequential formulation of a GP described in (Osborne, et al., 2008), a natural fit for our sequential decision problem. After each new function evaluation, we can efficiently update our predictions in light of the new information received.

The optimization of the expected-loss surface was performed using parallel independent runs of the MATLAB function \texttt{fmincon} from several different starting points (The MathWorks, Inc., 2010a). The expected-loss function is unfortunately usually multimodal, but fortunately even a local minimum of the expected loss is likely to represent a reasonably informative observation. Our results indeed proved largely insensitive to changes in the method used for this optimization.

We tested both the one-step lookahead and the two-step lookahead versions of our algorithm, only applying the latter to the most demanding problems. For noisy optimization, we limited ourselves to the one-step lookahead policy with the nonperiodic covariance. These restrictions were made to accelerate testing, allowing more functions to be included in the test.

The input scales \( \sigma_{SE} \) and \( \sigma_{P} \) and the output scales \( \lambda_{SE} \) and \( \lambda_{P} \) were marginalized using sequential Bayesian Monte Carlo as described in Section 5.5. Suppose the test region is specified by the box \( \Pi_{i}[a_i, b_i] \) and let

\[ m \cong \max_{i}(b_i - a_i) \]

be the largest dimension of this box. The samples for our hyperparameters were chosen to be uniform in the log-space; specifically, the samples for \( \log \sigma_{SE} \) and \( \log \sigma_{P} \) were chosen to be equally spaced in the interval

\[ [\log m/10 - \gamma/2, \log m/10 + \gamma/2], \]
and the samples for log $\lambda_{SE}$ and log $\lambda_P$ were chosen to be equally spaced in the interval $[-2, 14]$. Five samples were used for each input-scale parameter, and nine samples were used for each output-scale parameter.

For the periodic covariance function $K_P$, the periodic smoothness parameter $\gamma_P$ was fixed at 1.

For noisy optimization, the noise parameter $\eta$ was marginalized in a similar manner. The samples for log $\eta$ were chosen to be uniform in the interval $[\log 0.01, 0]$. Five samples for the noise parameter were used. The threshold $\epsilon$ used to define $\ell'$ was set to 1 in all noisy tests.

For noiseless testing, we compared our algorithm with the efficient global optimization (EGO) (Jones, et al., 1998), the radial basis function (RBF) (Gutmann, 2001), and the dividing rectangles (DIRECT) (Jones, et al., 1993) optimization algorithms. The implementations used for these were those in the TOMLAB/CGO toolbox for MATLAB, version 6 (Holmström, 1999). The default settings were used for each, except that the number of function evaluations used for initial samples (from which the methods learn hyperparameters) was reduced by 25%. Otherwise, these methods would have used all of their available function evaluations just to initialize their models. These modifications thereby improved the performance of the algorithms by allowing them to intelligently, rather than systematically, choose where to place at least a portion of their function evaluations.

For noisy testing, we compared our algorithm with the implicit filtering (IF) (Kelley, 2008), version 0.7, and stable noisy optimization by branch and fit (SNORFIT) (Huyer and Neumaier, 2008), version 2.1, routines. The MATLAB implementations used for these algorithms are freely available from the authors’ web pages. The default parameters were used for each.

Discussion

The results of noiseless testing are shown in Table 8.2. Our method performed (or tied with) the best of the four tested algorithms for 13 of the 16 noiseless problems. Additionally, the grand mean performance of our method (even for the simplest model and the least informative prior) surpassed the other methods by 16%. We conclude that GPGO can provide an improvement over other state-of-the-art solutions for global optimization of expensive functions.

Our impressive performance can be partially explained by our proper handling of hyperparameters. By sequentially marginalizing hyperparameters, we can intelligently select where to take function evaluations from the very beginning, learning simultaneously about both the objective function and the model hyperparameters with each new observation. This is in contrast to the naïve maximum likelihood or maximum a posteriori estimation approaches to hyperparameter management used by competing methods. An improvement in performance was also seen when using a more informative prior covariance function for periodic objective functions, illustrating the power of incorporating all available prior knowledge. Finally, using the two-step lookahead policy also resulted in increased performance. Combining our best results for each problem (using a periodic covariance or multiple-step lookahead, as appropriate), our grand mean performance was 0.755, a 24% improvement over the closest tested competitor.

The results of testing noisy objective functions are listed in Table 8.3. The GPGO algorithm significantly outperformed tested competitors, exceeding or equaling their
Table 8.2: Mean measure of performance (1 for various global optimization heuristics).
Table 8.3: Mean measure of performance $G$ for various noisy global optimization techniques on noisy objective functions. Objective function outputs were corrupted with Gaussian noise with the indicated standard deviation. Numbers highlighted in bold are the highest for the relevant problem.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$\sigma = 0.1$</th>
<th>$\sigma = 0.2$</th>
<th>$\sigma = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SNOB</td>
<td>FIT</td>
<td>IF</td>
</tr>
<tr>
<td>A2</td>
<td>0.738</td>
<td>0.051</td>
<td>0.768</td>
</tr>
<tr>
<td>A5</td>
<td>0.499</td>
<td>0.053</td>
<td>0.293</td>
</tr>
<tr>
<td>Br</td>
<td>0.980</td>
<td>0.999</td>
<td>0.996</td>
</tr>
<tr>
<td>C6</td>
<td>0.997</td>
<td>0.994</td>
<td>0.998</td>
</tr>
<tr>
<td>G–P</td>
<td>0.998</td>
<td>0.997</td>
<td>1.000</td>
</tr>
<tr>
<td>G2</td>
<td>1.000</td>
<td>0.320</td>
<td>1.000</td>
</tr>
<tr>
<td>G5</td>
<td>1.000</td>
<td>0.195</td>
<td>1.000</td>
</tr>
<tr>
<td>H3</td>
<td>0.956</td>
<td>0.954</td>
<td>0.977</td>
</tr>
<tr>
<td>H6</td>
<td>0.797</td>
<td>0.733</td>
<td>0.936</td>
</tr>
<tr>
<td>R</td>
<td>0.000</td>
<td>0.988</td>
<td>0.000</td>
</tr>
<tr>
<td>Sh5</td>
<td>0.066</td>
<td>0.001</td>
<td>0.134</td>
</tr>
<tr>
<td>Sh7</td>
<td>0.118</td>
<td>0.038</td>
<td>0.158</td>
</tr>
<tr>
<td>Sh10</td>
<td>0.133</td>
<td>0.284</td>
<td>0.288</td>
</tr>
<tr>
<td>Shu</td>
<td>0.667</td>
<td>0.417</td>
<td>0.669</td>
</tr>
<tr>
<td>mean</td>
<td>0.638</td>
<td>0.501</td>
<td>0.658</td>
</tr>
</tbody>
</table>
8. GAUSSIAN PROCESSES FOR GLOBAL OPTIMIZATION

performance in 29 of the 42 problems. Our method is clearly also effective at optimizing functions corrupted by noise. Our performance is remarkable given the myopic one-step lookahead approximation (8.3.1) used for testing. The use of a more sophisticated algorithm, for example one using a multiple-step lookahead decision rule, can be expected to even further improve our performance.
In this chapter, we will consider the problem of selecting the optimal locations for making measurements of some dynamic spatial process. This task, studied under the name spatial sampling design (Olea, 1984; Zhu and Stein, 2006), is encountered in a wide range of applications, including soil science, petroleum geology, and oceanography. In particular, we will consider the problem of placing a small number of sensors with the goal of making accurate global predictions about a nonstationary spatial field, such as air temperature.

We will approach this problem by treating the prediction quality of a set of sensors as a “black-box”—a measure of the quality of a particular sensor layout that is supplied to us after having tried it. This quality measure could be any arbitrarily complicated function; for example, it might be provided by an external agency, such as a human user of the sensor network. Prediction quality will serve as an objective to be optimized as a function of sensor locations. We will use measurements of the objective function to perform inference about its likely value for other sensor sets. By considering the decision problem of selecting which sensor placements to evaluate, we can derive the optimal Bayesian decision rule for this task.

Our inference will be enabled by the use of Gaussian processes, introduced in Chapter 5. In particular, we will place a GP prior over the quality function on the power set of available sensors, which we will condition on our observations of sensor subset performance. We will then apply the Gaussian processes for global optimization (GPGO) algorithm described in the previous chapter to optimize the prediction quality function over this space.

To enable the proposed inference and optimization procedures within a Gaussian process framework, however, we must first specify an appropriate covariance function for use with the large, discrete input space that a power set represents. The solution to this problem will depend on the application; here we will define a metric that encapsulates several intuitive notions of “closeness” for networks of isotropic sensors. This metric can then be used to build a reasonable covariance function for our particular problem.

Other proposed approaches for this task, such as the mutual information optimization procedure proposed by Krause, et al. (2008), have typically assumed a great deal of a priori knowledge about how different sensor sets will perform and are unable to completely exploit the information gained from experimental trials of sensor layouts. In contrast, our formulation will grant us great flexibility, permitting its application to problems of many different types and allowing the incorporation of as much or as little prior knowledge that might be available. In its full generality, our method aims to optimize any arbitrary function defined on a power set.

Let us first briefly describe the problem we will consider in more detail.

9.1 PROBLEM DESCRIPTION

As in Chapter 7, we consider performing time-series prediction on $S$ separate correlated variables. In typical sensor-network applications, these variables would correspond to
the measurement of a correlated spatial field by a set of $S$ discrete sensors. Rather than building our belief about these variables from measurements of all of them, however, we limit ourselves to having observations of only a subset of them. Our ultimate goal is to choose a subset of the variables of interest that maximizes the predictive performance for all $S$ variables, when restricted to observations from only the smaller set.

Of course, unless some of the available sensors make very poor or faulty observations, the prediction quality will most likely be optimized by simply measuring all available variables. In most applications, however, there are costs associated with maintaining a sensor, such as power and communication overhead. The sensor selection problem seeks to place a small number of sensors such that the resulting network makes the best possible predictions at all locations of interest, reducing cost with the least possible reduction in quality.

To perform this optimization, we will require an appropriate objective that measures the quality of our predictions. Suppose that we label our sensors with the index set $$I \doteq \{1, \ldots, S\}.$$ We assume the existence of a function $q$ that for a subset $s \in \mathcal{P}(I)$ is commensurate with the quality of the predictions for all $S$ variables made by a model limited to measurements of the variables in $s$. In many situations, the quality of a particular sensor subset can vary over time; for example, drift in the spatial field of interest or a sensor fault could radically alter the predictive power of a sensor network. For this reason, we will let the function $q$ be a function of both sensor subset and time. Therefore, we take an objective function $$q : \mathcal{P}(I) \times \mathbb{R} \to \mathbb{R}$$ as given and attempt to intelligently optimize this function over the space $\mathcal{P}(I)$ through time. We make no assumptions about either the exact nature of the function $q$ or how in particular our sensor network generates its predictions. We only require that $q$ return a consistent measure of performance for the particular application under consideration. Beyond this, we treat the evaluation of network prediction quality as a "black box."

### 9.2 Applying GPGO to an Objective on a Power Set

Before we describe the exact details of our proposed algorithm for sensor set selection, we will address how the GPGO routine could be used to optimize an arbitrary function defined on a power set. As described in Chapter 5, we may place a Gaussian process prior distribution over the real-valued functions on any arbitrary input space $\mathcal{X}$.

Suppose we have a set $I$ and a real-valued objective function $$y : \mathcal{P}(I) \to \mathbb{R}$$ defined on the power set of $I$. The GPGO algorithm requires that we place a Gaussian process prior over $y$, $$p(y \mid \theta) \doteq \mathcal{GP}(y; \mu, K),$$ (9.2.1) where $$\mu : \mathcal{P}(I) \to \mathbb{R},$$ $$K : \mathcal{P}(I) \times \mathcal{P}(I) \to \mathbb{R},$$
are the prior mean and covariance functions, respectively. The prior mean function \( \mu \) can be chosen as required for the problem at hand—we will often use a constant for convenience, but other considerations might lead to more informative mean functions. The choice of the prior covariance \( K \), however, is a more difficult problem. It is unclear \textit{a priori} how the function \( y \) might behave over the somewhat unusual, discrete space represented by a power set.

A simple mechanism for building a covariance function is to first define a metric on the input space that captures any prior belief about the “similarity” of input points and assume that this metric should imply some level of correlation between function values at points that are deemed “close” to each other. Once this metric has been defined, we can simply incorporate it into an off-the-shelf stationary covariance function of the form (5.3.4) and assign the resulting covariance to \( K \). For arbitrary functions, we could build a covariance from a simple set metric, for example the Hamming distance. In many cases, properties of the points in the underlying space can allow us to define a more informative metric than this. In the next section, we will describe a metric for the specific problem of sensor subset selection that captures many intuitive properties of how network predictive accuracy should vary as a function of sensor locations.

Suppose that we have selected an appropriate distance

\[
d: \mathcal{P}(I) \times \mathcal{P}(I) \rightarrow \mathbb{R}^+
\]

between subsets of \( I \) and have used \( d \) to build a covariance \( K \) of the form (5.3.4), fully specifying the GP prior distribution (9.2.1). Given this prior distribution, the application of GP-GO to our problem is straightforward. We first select the optimization lookahead parameter described in the previous chapter; for convenience, we will use the one-step lookahead rule (8.1.5). At each step of the optimization process, we evaluate \( f \) on a particular subset of \( I \) and update our GP over \( \mathcal{P}(I) \) with this observation. We then evaluate the expected loss function (8.1.4) on candidate subsets in \( \mathcal{P}(I) \); the minimum among these becomes the subset used for the next evaluation of \( y \) (8.1.5). We continue in this manner for as long as desired. Pseudocode for \( N \) steps of our algorithm is provided in Algorithm 9.1.

Of course, the size of \( \mathcal{P}(I) \) grows exponentially in the size of \( I \), and evaluating (8.1.4) at every point quickly becomes infeasible. However, the nature of the expected loss function suggests various search heuristics that can be adopted to guide the search. In particular, the expected loss function is likely to be minimized at either a point that is close to an observation we have already made (exploitation), or far away from every observation (exploration). We can speed up our search by only considering sets with these properties. Of course, evaluating the distance itself between all pairs of subsets can also become quickly infeasible, but we may apply further heuristics. For example, we can encourage exploitation by including subsets that differ from our current observations by one point (that is, have Hamming distance 1) and encourage exploration by including a random selection of the remaining subsets. The effect to Algorithm 9.1 is that the search over the candidate set \( S' \) is constrained to these heuristically chosen points; the

---

\(^1\text{The Hamming distance between two finite sets } A \text{ and } B \text{ is equal to the cardinality of their symmetric difference:}\)

\[
d_{\text{Hamming}}(A, B) = |(A \setminus B) \cup (B \setminus A)|.
\]
Algorithm 9.1: Pseudocode for optimizing a function over a power set.

**Input**: objective function \( y \), priors \( p(y \mid \theta), p(\theta) \); power set \( \mathcal{P}(I) \), initial set \( s_i \), number of evaluations \( N \), GPGO lookahead \( L \)

**Output**: suspected optimum point \( (s_f, f(s_f)) \in \mathcal{P}(I) \times \mathbb{R} \)

\[ D_0 \leftarrow \emptyset \]

for \( i \leftarrow 1 \) to \( N \) do

\[ \text{evaluate } y(s_i) \]

\[ D_i \leftarrow D_{i-1} \cup \{ s_i, y(s_i) \} \]

for \( \psi \in \Psi \) do

\[ p(y \mid D_i, \psi) \leftarrow \text{CONDITION-GP} \left( p(y \mid D_{i-1}, \psi), D_i \right) \]

end

\[ S' \leftarrow \text{DETERMINE-CANDIDATE-SUBSETS} \left( D_i \right) \]

\[ s_{i+1} \leftarrow \text{MINIMIZE-EXPECTED-LOSS} \left( p(y \mid D_i), p(\theta), L \right) \]

end

\[ s_f \leftarrow \arg \min_{s \in D_N} y(s) \]

return \( (s_f, f(s_f)) \)

undefined function \( \text{DETERMINE-CANDIDATE-SUBSETS} \) controls the exact behavior of this selection.

### 9.3 Metrics over Sensor Sets

We now consider the challenge of constructing a metric between sensor subsets to ultimately construct a covariance of the form (5.3.4) for performing inference about the unknown values of the objective sensor-network quality function \( q \). We assume that the objective function is smooth, in the sense that it will have similar values for "related" sets. For example, when considering the predictive performance of a sensor network, we should expect \( \text{a priori} \) that moving each sensor by a small amount will not typically greatly affect its performance.

To formalize the meaning of "related" in this context, we seek to define a parametrized metric \( d_{\text{rel}}(A, B; \theta) \) for \( A, B \in \mathcal{P}(I) \). We will notate these sets

\[ A \triangleq \{ a_i \mid i = 1 \ldots N \}, \]
\[ B \triangleq \{ b_j \mid j = 1 \ldots M \}. \]

The metric we will propose for this role would be appropriate for considering the predictive quality of a network of multiple isotropic sensors.

For convenience, we will define our suggested \( d_{\text{rel}} \) in terms of an underlying distance between singletons. Suppose that the features of our sensors lie in some space \( \mathcal{S} \) with an associated metric \( d'(s, s') \) defined on \( \mathcal{S} \times \mathcal{S} \). We assume that the chosen metric would be useful for performing inference about the objective function if we restricted the domain to singleton sets and extend this distance to arbitrary finite subsets of \( I \).
Choosing the pairwise distance $d'\prime$

An appropriate distance $d'\prime$ can be easily obtained for sensors defined by their spatial coordinates; for example, the usual Euclidean distance (or great-circle distance on the earth) usually suffices.

As mentioned above, our sensor selection procedure does not require any particular form for the underlying prediction algorithm. If, however, a Gaussian process over the measured field is used in this capacity (separate from the Gaussian process over $q$ used for optimization), we have several more options for the pairwise distance $d'\prime$. In this case, the underlying prediction Gaussian process will have a parametrized covariance $K'(s, s'; \theta)$ defined on $S \times S$. This covariance could contain significant information about the field of interest that could be exploited during the process of selecting $d'\prime$. For example, we could use the characteristic input scales in $K'$ to influence the choice of any scale parameters of $d'\prime$. If the covariance is more sophisticated, reflecting, for example, periodicity or changepoints in the field, we might also consider defining the pairwise distance with

$$d'(s, s'; \theta) = \sqrt{K'(s, s'; \theta) + K'(s, s'; \theta) - 2K'(s, s'; \theta)}.$$ 

The covariance $K'$ might also be used to directly define a distance over sets without requiring an intermediate pairwise distance.

Desired properties of the subset metric for sensor networks

Suppose an appropriate distance $d'\prime$ has been chosen. Below, we will use this pairwise distance to build a metric between sets that can be used when considering the predictive accuracy of a network of isotropic sensors. Before we introduce the details of the metric, let us first consider some simple example situations to motivate our choice; see Figure 9.1 for reference. In each of the depicted arrangements, we assume that $d'\prime$ is the Euclidean distance and that the vertical distance $\delta$ is very large.

Figure 9.1(a) depicts a sensor arrangement where $d'(a, b; \theta)$ is large for every pair $(a, b) \in A \times B$. There is no reason to expect that the predictions made by the resulting sensor networks should be correlated, considering their dramatically different spatial locations. Therefore $d_{set}(A, B; \theta)$ should be large in such situations.

Next, Figure 9.1(b) illustrates a situation where every sensor $b \in B$ has a nearby companion sensor in $A$, except for one exception. Despite many overlaps in coverage between the two networks, the far-away $B$ sensor might be in a location that drastically affects the quality of the resulting sensor network, and there is no sensor in $A$ nearby to compensate. Therefore $d_{set}(A, B; \theta)$ should be at least somewhat large for such arrangements.

The grouping in Figure 9.1(c) has, for every $a \in A$, at least one $b \in B$ such that $d'(a, b)$ is small—the sensors in $A$ and $B$, despite relabeling, have very similar locations overall according to $d'\prime$. Accordingly, the resulting networks have very similar coverage, and the distance $d_{set}(A, B; \theta)$ should be small. We can conclude from this example that our set distance should depend (at least mostly) on the proximity of each point in $A$ to the closest point in $B$. Furthermore, $d_{set}$ should be invariant to permutations of sensor labels.

Figure 9.1(d) shows the sets from Figure 9.1(c) with an additional sensor, $a_3$, added close to $a_2$, and another sensor, $b_3$, added close to $b_2$. It seems reasonable to assume...
Figure 9.1: Illustrative hypothetical sensor-set arrangements and the intuitively desirable distance for each. When the sensors depicted have isotropic sensing capabilities, and $\delta$ is sufficiently large, the distance $d(A, B; \theta)$ between sets $A$ and $B$ should be (a) large, (b) somewhat large, (c) small, (d) small, and (e) large.
that adding a sensor very close to an existing sensor should not dramatically alter
the performance of the resulting network. Consequently, we should expect the
distance between $A$ and $B$ in this situation to be similar to the distance between
the sets in Figure 9.1(c).

An analogous argument should apply to the arrangement depicted in Figure 9.1(e),
which is the first situation to be considered that has sets of different sizes. Despite
the different number of sensors in $A$ and $B$, however, the networks resulting from
the sets in Figure 9.1(e) should be very similar in performance to those in Figure 9.1(a) (which
differs only by the addition of the mostly redundant sensor $a_2$). We can conclude that
$d_{\text{set}}(A, B; \theta)$ should similar to the distance between the sets in Figure 9.1(a).

We now propose a metric that satisfies these desiderata.

**The earth mover’s distance**

The metric to which we appeal is the earth mover’s distance (EMD), which is well known
and widely used in image processing (Rubner, et al., 2000; Zhou, et al., 2005). The
earth mover’s distance is defined for two “signatures” of the form $\{ (x_i, w_i) \}$, where
the $\{ x_i \}$ are points in space (in our case, points on the sphere $S^2$) and the $w_i \in \mathbb{R}^+$ are
positive real weights. When the total weight of each signature sums to unity (that is, each
signature represents a discrete probability distribution), the EMD is equivalent to
the first Wasserstein or Mallows distance (Zhou, et al., 2005). We will assume that each
signature normalizes in this manner.

Given two signatures

$$A = \{(a_i, w_i)\}_{i=1}^N,$$
$$B = \{(b_j, v_j)\}_{j=1}^M,$$

the EMD is defined intuitively as the minimum amount of work required to move the
points in $A$ to be aligned with the points in $B$. The notion of “work” in this case coincides
with its normal physical definition—the work required to move a point is proportional
to its weight and the distance traveled. If each signature is regarded as a collection
of mounds of dirt, the EMD would represent the minimum amount of earth (times
distance) one would have to move to transform one signature into the other, giving rise
to its name.

We may define EMD more formally in terms of an optimization problem.

**DEFINITION 9.3.1 (Earth mover’s distance)** Let

$$A = \{(a_i, w_i)\}_{i=1}^N,$$
$$B = \{(b_j, v_j)\}_{j=1}^M,$$

be two signatures, and let $d’$ represent a pairwise metric between the points $\{a_i\}$ and $\{b_j\}$.
The earth mover’s distance $d_{\text{EMD}}$ is defined by the solution to the following linear program.

$$d_{\text{EMD}}(A, B) = \min \sum_i \sum_j f_{i,j}d’(a_i, b_j), \quad (9.3.1)$$
subject to the following constraints:

\[
\begin{align*}
  f_{i,j} & \geq 0 & (1 \leq i \leq N, 1 \leq j \leq M); \\
  \sum_j f_{i,j} & = w_i & (1 \leq i \leq N); \\
  \sum_i f_{i,j} & = v_j & (1 \leq j \leq M); \\
  \sum_i \sum_j f_{i,j} & = 1.
\end{align*}
\]

The solution to this optimization problem can be found in polynomial time (\(O(N^3)\)) when \(|A| = |B| = N\) using the well-known Hungarian algorithm.

In the case of sensor networks, corresponding sets of sensors with the signatures described above can be done in a straightforward manner except for the assignment of weights to each point. Naively we might try a simple uniform weighting over the points in each set. The resulting distance achieves nearly all the heuristic goals set forth in the previous subsection. In particular, the uniformly weighted earth mover’s distance has the desired behavior for Figures 9.1(a), 9.1(b), 9.1(c), and 9.1(e). Figure 9.1(d) presents a problem, however—using a uniform weighting, the optimal flow pairs the two closest opposite-colored points at the top, \(a_1\) and \(b_1\), and similarly pairs the two closest opposite-colored points at the bottom, \(a_2\) and \(b_2\). The remaining two points, \(a_3\) and \(b_3\), are then compelled to share an edge of length \(\delta\), and the overall distance will be approximately \(\delta\), assuming the vertical distance dominates the horizontal. This behavior is not desirable—by increasing the vertical distance, we can arbitrarily increase the distance between the two sets, but their performance should be expected to be similar for any sufficiently large distance \(\delta\).

The reason for this behavior, of course, is that not all sensors are created equal. Assuming sufficient range, the sensors \(a_2\) and \(a_3\) will have significant overlap, as will \(b_2\) and \(b_3\). In the limit where \(a_2\) and \(a_3\) lie completely on top of each other, they might as well be a single sensor—one has been made completely redundant. With this insight, we modify the weights for the two signatures such that the nearby sensors \(a_2\) and \(a_3\) share nearly half the total weight of the \(A\) set (as they, combined, act essentially as a single sensor), and \(b_2\) and \(b_3\) share nearly half the total weight of the \(B\) set. With this modification, the \(emd\) distance will now pair both \(a_2\) and \(a_3\) with \(b_1\), because both sets now have nearly equal weight; similarly, \(b_2\) and \(b_3\) will be paired with \(a_1\). The overall distance will now be small (and in the limit, not depend on \(\delta\)), as desired.

Deriving appropriate \(emd\) signature weights

Depending on the nature of the sensors and the prediction method used, any number of weighting schemes appropriately accounting for redundancy could be used; see, for example, (Dept, et al., 2003) for a discussion of redundancy in sensor networks with identical isotropic sensors. The chosen scheme might also incorporate information about the range of each sensor. For example, suppose sensors \(a_2\) and \(a_3\) were actually directional sensors pointing away from each other. It would then be inappropriate to reduce their weighting as discussed above, because they would have very little overlap, despite their proximity.

We offer an appropriate weighting scheme to use when the underlying prediction algorithm is implemented using a Gaussian process with covariance \(K'\). Suppose we
take an observation of the spatial field of interest from each of the sensors in the set \( s \in \mathcal{P}(I) \). Let \( X_s \) represent the points in the input space of the prediction GP associated with these sensors and let
\[
D \triangleq (X_s, z_s)
\]
represent the (potentially noisy) measured observations at \( X_s \). Consider using these observations to make a prediction about the value of the field at an arbitrary point \( x_\ast \).

The mean of our prediction, defined in (5.2.3), is a weighted linear combination of the observed values:
\[
m_{yp}(x_\ast) = \mu(x_\ast) + K'(x_\ast, X_s)V'(X_s, X_s)^{-1}(z_s - \mu(X_s)).
\]
We will notate the weights on the measurements \( z_s \) used to predict the latent value at \( x_\ast \) with the symbol
\[
\beta(x_\ast, s) \triangleq K'(x_\ast, X_s)V'(X_s, X_s)^{-1}.
\]
To construct appropriate weights for the EMD signature associated with the sensors in \( s \), we will consider the expected value of these weights over a range of points \( x_\ast \). This requires choosing a prior distribution for the points of interest, \( p(x_\ast) \). In the absence of other information, we will choose the uninformative prior \( p(x_\ast) \propto 1 \), but the nature of most applications should allow for a more informative prior to be chosen.

We now calculate
\[
\hat{\beta}(s) \triangleq E[\beta(x_\ast, s)]
\]
\[
= \int \beta(x_\ast, s)p(x_\ast)\, dx_\ast
\]
\[
\propto \int \beta(x_\ast, s)\, dx_\ast
\]
\[
= \int K'(x_\ast, X_s)V'(X_s, X_s)^{-1}\, dx_\ast
\]
\[
= \left(\int K'(x_\ast, X_s)\, dx_\ast\right)V'(X_s, X_s)^{-1}
\]
We define the convenient function
\[
\omega(s) \triangleq \int K'(x_\ast, x_s)\, dx_\ast,
\]
where \( s \in s \) is one of the chosen sensors and \( x_s \) is its location in input space. Our expectation of interest now becomes
\[
\hat{\beta}(s) \propto \omega(s)V'(X_s, X_s)^{-1}.
\]
These weights, once appropriately normalized,\(^2\) represent the average weights associated with each of the chosen sensors when making predictions at other locations. This notion naturally reflects the behavior we desire for EMD signature weights—elements far away from all others will receive a large weight, whereas tight clusters of elements will together share, approximately, the weight of a single element.

Figure 9.2 demonstrates the difference between the unweighted and weighted EMD distances. In the weighted version, clusters of nearby sensors receive appropriate weights. The overall distance for the weighted version is more natural than the unweighted version, which is forced to incorporate long, highly weighted, awkward edges.

\(^2\)Note that because EMD signatures were defined to sum to unity, the \( \hat{\beta}(s) \) vector must be normalized before use.
Figure 9.2: An example of the EMD between two sets, which are differentiated by their shading. The distance is the sum of the lengths of the depicted edges multiplied by their weights. The edge weights, indicated by line width, are in turn determined with respect to node weights, which are also indicated by line width. For (a), the total unit weight for each set is split evenly between all nodes. For (b), the weight for each node is assigned according to its importance using (9.3.2). Nodes that might be expected to be more informative when determining the overall performance of a sensor network—those that are far separated from other sensors in the network—are assigned greater weight.
Outline of our approach

We will now describe how we will combine the ideas presented above to form a complete algorithm for sensor subset selection.

We begin by choosing an appropriate quality function $q$, defining the desired prediction mechanism, enumerating the possible sensor locations to be considered, and deciding what portion of the associated power set should be explored.

We use the GP-GO algorithm described in Section 9.2 to optimize $q$ through space and time. The appropriate input space $\mathcal{X}$ for the sensor subset selection problem as described above is

$$\mathcal{X} \triangleq \mathcal{P}(I) \times \mathbb{R},$$

where the continuous last component represents time. We place a GP prior on $q: \mathcal{X} \rightarrow \mathbb{R}$:

$$p(q \mid \theta) \triangleq \mathcal{GP}(q; \mu, K),$$

where $\mu$ is an appropriate mean function. For the covariance function $K$, we select a tensor product that conveniently separates:

$$K((s, t), (s', t'); \theta) \triangleq K_{\text{set}}(s, s'; \theta)K_{\text{time}}(t, t'; \theta). \quad (9.3.3)$$

For the covariance over time, $K_{\text{time}}$, a stationary covariance function (5.3.4) should suffice for most scenarios, although one of the changepoint covariances from Chapter 6 could be used as well if sensor-network performance through time was expected to undergo sudden behavioral changes. We derive the covariance over sets, $K_{\text{set}}$, by incorporating the weighted MD distance described in the previous section into a stationary covariance function (5.3.4). Once this prior has been specified, we apply Algorithm 9.1 for as long as desired.

9.4 RUNNING TIME

Let us briefly consider the running time of our proposed algorithm.

Suppose that we have chosen $M$ hyperparameter samples. Incorporating the $N$th objective function evaluation into our optimization GP requires making rank-one updates to $M$ Cholesky factorizations of size $(N - 1)^2$, and each training step therefore runs in $O(MN^2)$ time. Evaluating the expected loss at $K$ points then takes time $O(KMN^2)$.

In practice, a somewhat inefficient implementation of the algorithm described above took approximately 120 seconds to train the GP-GO model and select the next subset when trained on 100 observations, using approximately 1 000 hyperparameter samples, and limiting the evaluation of the expected loss to 40 000 subsets. The machine was an Apple Mac Pro workstation with two 2.26 GHz quad-core Intel Xeon "Gainestown" processors (model B5220) and 16 GB of RAM, running the Linux kernel (version 2.6.31).

9.5 EXPERIMENTS

We tested our proposed algorithm on the UK Meteorological Office MIDAS land surface stations data (BRITISH ATMOSPHERIC DATA CENTRE, 2009). In particular, we selected a dispersed set of 50 sensors across the United Kingdom that recorded meteorological measurements for every day between 1999 and 2009, inclusive. The locations of these sensors are plotted in Figure 9.3. The field we chose to perform prediction over was the maximum daily temperature at these locations.
The particular performance measure \( q \) we chose for our experiments was the root mean squared error (RMSE) of the predictions made when using observations from the chosen sensors to predict the maximum temperature at every location in the UK. Let \( s \in \mathcal{P}(I) \) be a set of sensors, \( t_0 \) be the first day the associated sensor network made predictions, and \( \Delta \) represent the number of days the sensor network was online and making predictions. We specifically define the RMSE\((s, t_0, \Delta)\) function to be the root mean squared error of the maximum daily temperature predictions made by the network associated with \( s \) over the time frame \([t_0, t_0 + \Delta] \). If necessary, the prediction algorithm could be granted a “burn-in” period so that the RMSE function is not dominated by poor predictions made when very little data was available, but we did not explore that possibility here.

We considered the problem of selecting the best possible subset of five sensors to minimize this RMSE function. This particular subset size was chosen for two reasons to increase the problem’s difficulty: producing good predictions over the entire United Kingdom with so few sensors is challenging, and the search space is also quite large—\( \binom{950}{5} = 2118760 \).

Our predictions were made using a GP sequentially trained on the readings at all test times (Osborne, et al., 2008). For each day in the desired prediction window, we used all available readings from the selected sensors to predict the maximum daily temperature at the other locations of interest. These predictions took the form of zero-step lookahead (tracking); that is, we used the current readings from the selected sensors when making predictions about that day.

For the prediction GP, we selected a simple mean function that for each sensor was defined to be the mean maximum daily temperature recorded at that location over the entire period 1959–2009. For the covariance function, we selected a product of two rational quadratic covariance functions (5.3.8), one over space and the other over time. The covariance over space incorporated the great-circle distance between points on the surface of the earth, and the covariance over time used the trivial distance

\[
d_{\text{time}}(t, t') = |t - t'|.
\]  

(9.5.1)

The emphasis of our testing was not to make the most accurate possible predictions (for which we would recommend using more sophisticated non-stationary covariances), but rather to test our sensor selection algorithm given any arbitrary black-box objective function. The hyperparameters of the prediction GP were sequentially marginalized using the Bayesian Monte Carlo procedure outlined in Section 5.5.

The covariance function used for GPGO was of the form (9.3.3). Both \( K_{\text{set}} \) and \( K_{\text{time}} \) were assigned squared exponential covariance functions (5.3.5), the former using our weighted EMD distance, \( d_{\text{EMD}} \), and the latter using the trivial distance (9.5.1). The hyperparameters of this model were sequentially marginalized using BMC. We applied search heuristics as suggested in Section 9.2, minimizing the expected loss at approximately 40,000 points for each selection\(^3\). These included all subsets that differed in only a single point from the current observation set (to encourage exploitation), along with many other randomly chosen subsets (to encourage exploration).

\(^3\)The number of points for each round of optimization was chosen based on time constraints during testing.
9.5. Experiments

Figure 9.3: The locations of all 50 weather sensors for our experiments with the MIDAS data.

Figure 9.4: The locations of sensors selected at the years indicated by (a)–(d), the MI algorithm, and (e)–(h), the GPGO algorithm.

For the MIDAS data, we began a new trial at the beginning of each successive period of $\Delta \leq 28$ days. We selected a set of five sensors and then performed online prediction using our GP model of the temperatures at all 50 sensors for each of the subsequent 28 days. To test the effectiveness of our proposed algorithm, we identified the current putative optimal subset every five years and evaluated the predictive quality of the associated sensor network for the entire subsequent year.

To provide a comparison for our algorithm, we first tested against a simple random selection algorithm, which selected subsets randomly for each trial period of 28 days.
Table 9.1: The root mean squared error (in °C) of the GP prediction algorithm for tracking the air temperature across the UK in selected years, using subsets picked by a random selection algorithm, our subset optimization routine (GPGO), and the mutual information algorithm (MI).

<table>
<thead>
<tr>
<th>Year</th>
<th>Random</th>
<th>MI</th>
<th>GPGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1964</td>
<td>2.9416</td>
<td>3.0182</td>
<td>2.4408</td>
</tr>
<tr>
<td>1969</td>
<td>3.0295</td>
<td>3.1553</td>
<td>2.7702</td>
</tr>
<tr>
<td>1974</td>
<td>2.6945</td>
<td>2.8208</td>
<td>2.4310</td>
</tr>
<tr>
<td>1979</td>
<td>2.8359</td>
<td>2.8238</td>
<td>2.8143</td>
</tr>
<tr>
<td>1984</td>
<td>2.9156</td>
<td>3.0690</td>
<td>2.6249</td>
</tr>
<tr>
<td>1989</td>
<td>2.9538</td>
<td>3.0845</td>
<td>2.6438</td>
</tr>
<tr>
<td>1994</td>
<td>3.0370</td>
<td>2.9886</td>
<td>2.8252</td>
</tr>
<tr>
<td>1999</td>
<td>3.0223</td>
<td>3.5081</td>
<td>2.8561</td>
</tr>
<tr>
<td>2004</td>
<td>2.9548</td>
<td>3.0695</td>
<td>2.6886</td>
</tr>
<tr>
<td>2009</td>
<td>3.0111</td>
<td>2.8051</td>
<td>2.8981</td>
</tr>
<tr>
<td>mean</td>
<td>2.9396</td>
<td>2.8256</td>
<td>2.6993</td>
</tr>
</tbody>
</table>

For each test year, we selected the subset that yielded the best RMSE performance during the preceding five years.

We also implemented the method outlined in (Krause, et al., 2008), which we will refer to as the mutual information (MI) algorithm. The MI algorithm uses a greedy method to select sensors that approximately maximize the mutual information between the locations of the chosen sensors and all locations of interest. This mutual information is derived from a GP covariance function to be used for prediction. For each trial period, we selected a set of five sensors using the MI algorithm. The covariance function used to calculate mutual information was determined from the prediction GP described above; in particular, we used the covariance associated with the hyperparameter sample with the highest BMC posterior weight.

The results of our experiment are shown in Table 9.1. Overall, our algorithm was able to locate subsets with better predictive performance, beating the other two tested algorithms in all but one of the test years.

The subsets selected by both the MI method and our algorithm for selected test years are displayed in Figure 9.4. The subsets chosen by both methods seem to provide intuitively good overall coverage of the UK. Note that our algorithm selects a greater number of sensors on the west coast of the UK. It is possible that sensors in this region were determined to be more useful for predicting temperatures by having better performance on the prediction task—weather predominantly moves from west-to-east in the UK. This nuance would have been automatically captured by our algorithm.

Our algorithm was able to locate reasonable optima, despite few observations (about 13 per year for 50 years from a total number of \( \binom{50}{5} = 2118760 \) possibilities). Additionally, our routine was able to adapt to information obtained by observing the performance of various subsets through time. Finally, by using a covariance defined on both space and time, our algorithm can cope with changes to the performance of various subsets over
time (for example, if a sensor becomes faulty), allowing for us to react to concept drift in the prediction problem.
CONCLUSIONS AND FUTURE WORK

The rapid expansion of the internet and increasing ease with which extremely large datasets may be collected have inspired a great deal of research into performing data mining on fast-moving data streams. The unique issues posed by these problems often renders the use of classical techniques impossible. Among these issues, one of the most important and drastic is the prevalence of arbitrary and often unexpected concept drifts in typical data streams.

After a brief review of relevant research into the nonstationary online learning problem, we introduced the dynamic logistic regressor (DLR), a Bayesian approach to nonstationary semi-supervised binary classification. We then demonstrated how the DLR framework could be extended to effectively cope with possible uncertainty in the label stream. The result was a very capable classifier able to handle an extraordinary range of online classification situations. A series of experiments on several datasets demonstrated the effectiveness of the DLR on dynamic classification problems and also confirmed that our proposed label-noise extension provided a noticeable benefit to its performance.

We then introduced Gaussian processes and for the remainder of the text leveraged the power of this technique to approach various nonstationary prediction problems. We introduced a number of covariance functions for performing time-series prediction on data streams that undergo sudden changes in their behavior. We also introduced a flexible observation model that can represent various types of observation faults, allowing for reasonable fault removal, and in some cases, fault recovery. We then demonstrated how to use these methods to perform effective active data selection in sensor networks that are prone to sudden drastic events. These methods were subjected to a very large number of real-world datasets and were able to make reasonable predictions while also identifying known changepoints and faults in the data sources.

We then described the Gaussian processes for global optimization (GPGO) algorithm for optimizing expensive objective functions, which incorporates several novel extensions to previous work. We tested the GPGO algorithm against several other state-of-the-art global optimization routines on an extensive set of standard test functions and found GPPO to have superior performance overall. Finally, we applied the GPPO routine to a nontrivial objective that is subject to drift—the predictive accuracy of a sensor network. We introduced a number of innovative ideas that allowed us to approach this problem from the viewpoint of Bayesian optimization. The resulting method was tested against a similar recently proposed algorithm on a difficult real-world weather-prediction problem and outperformed the competing technique almost universally.

10.1 FUTURE WORK

Despite the convincing results of our experimental tests, a number of improvements could potentially improve or extend the work presented in this thesis. We will briefly describe possible future avenues of research below.
The dynamic logistic regression algorithm

The dlr algorithm has been the subject of extensive research and is quite mature (Penny and Roberts, 1999; Lowne, et al., 2010). Nonetheless, the dlr could be refined in a number of ways. Perhaps the most significant improvement would be the addition of a predictive model for the weight parameters \( w \), that is, choosing a more complicated form for the transition function \( f \) in the update equation

\[
\mathbf{w}_t \triangleq f(\mathbf{w}_{t-1}) + \eta_t.
\]

In Section 4.1, we made the simplifying assumption that the weights do not undergo any sort of systematic drift, prescribing instead the identity function to \( f \):

\[
\mathbf{w}_t \triangleq f(\mathbf{w}_{t-1}) + \eta_t \triangleq \mathbf{w}_{t-1} + \eta_t.
\]

Although this assumption worked well in practice, it is perhaps not the best we could do. In many situations, the weights will undergo predictable drift, at least over short time scales; if this is the case, we should take advantage of that information. A more informative transition function should allow us to better predict the estimated posterior class-label probabilities. Even a very simple linear predictor might lead to improved performance for some problems.

Perhaps more importantly, however, using a more effective \( f \) function would allow for a more natural method for handling missing labels. Instead of defining the "quasitarget" (4.2.5) by

\[
\hat{y}_t \triangleq \chi(z_{t|t-1} > \bar{y}_t),
\]

we could instead calculate \( z_{t|t-1} \) using the more informative distribution \( p(w_t | D_{t-1}) \). Then, if the label \( y_t \) were missing, we could simply marginalize out the unknown value \( y_t \) when we derived the updated weight parameters \( p(w_t | D_t) \), which would have the effect of keeping the current weight estimate for the next step of the algorithm. We would then not presume to learn anything from the observation \((x_t, \varnothing)\), unlike the false knowledge suggested by quasitargets. As a result, this procedure would be less likely to bring the system into an unlikely state in the presence of many missing labels. On the other hand, the ekf framework would be made more complicated due to the more complex transition function \( f \), requiring a possibly difficult derivative to be calculated.

Finally, our approach to label-noise estimation could be extended easily; for example, different probabilities could be assigned to errors that flip label "0" to label "1" and label "1" to label "0." Such models have been used in the past (Lawrence and Schölkopf, 2001). Enhancements of this nature would not be difficult to incorporate into the dlr algorithm.

Gaussian process models

One issue with the methods proposed in this thesis is that Gaussian process models scale cubically with the number of inputs. By using rank-one updates to the Cholesky decomposition of the covariance matrix, we can decrease the running time per observation to \( O(N^3) \); however, even this can be prohibitive for large window sizes or large-scale data streams. It would be interesting to investigate the application of sparse Gaussian process approximations to these problems. Various procedures can be applied to approximate the gp posterior from \( M \) surrogate points, where \( M \ll N \), and achieve training cost \( O(M^2N) \) and prediction cost in terms of only \( M \); see, for example, (Snelson and Ghahramani, 2006).
Prediction with changepoints and faults

Our work with changepoints and faults could be extended in several simple ways. In the experiments presented in Chapter 6, we restricted ourselves to considering only one particular type of changepoint or fault at a time. It is easy to imagine selecting a broad prior distribution over various types of changepoints and faults and performing marginalization over all of these models simultaneously. Additionally, the particular type of fault that might be encountered during a prediction problem is often difficult to specify \textit{a priori}. For this reason, it might be beneficial to investigate an observation model with a very broad noise variance, or several with different noise variances, which might be flexible enough to automatically recognize any type of fault when it occurs. Although potentially possible, such an uninformative fault observation model would render fault recovery impossible. Nonetheless, the flexibility afforded by such a model might be useful in highly uncertain environments.

Additionally, our work on changepoint and fault models only considered the time-series (regression) prediction problem. It would be interesting to apply the derived models to other problem domains, for example classification. Gaussian process classification can be effectively performed using various approximations to the posterior distribution \cite[]{Rasmussen:2006}. By placing one of the changepoint covariances derived in Chapter 6 on a classification latent function, we could create a new method for performing online nonstationary classification. The resulting algorithm would serve as a potentially interesting competitor to the DLR. Whereas the DLR will by design work best with streams experiencing slow drift, a GP classifier using one of our changepoint covariances would be best-suited for reacting to sudden drastic changes to the classification boundary. For this reason, it might perform even better than the DLR on the sea datasets investigated in Chapter 4. Furthermore, if we incorporated a label-noise likelihood function as described in Section 4.3, we could enable this GP nonstationary classifier to handle potentially corrupted label streams. By treating the label-noise parameter \( \rho \) as a hyperparameter of the model, we could then marginalize out its unknown value for prediction or derive the posterior distribution over its value, if desired. Semi-supervised classification has also been performed with Gaussian processes \cite[]{Lawrence:2005}.

The GPGO algorithm

A number of extensions to the work discussed in Chapter 8 could be investigated. First, some applications involve problems where the objective function can be observed in one of several different modes. For example, we might be able to make a more accurate measurement of the objective function for an increased cost, or we might be able to choose between taking an observation of the function itself or one of its derivatives. By building upon our approach to noisy optimization, we could allow our algorithm to handle this sort of \textit{multiple-fidelity} problem. We would simply have to create appropriate models for each of the various modes of observation available to us and take the action with the lowest risk, taking into account the various costs of each possible observation.

Additionally, we limited ourselves to problems of relatively low dimension to enable manageable computation. It would be interesting to investigate various approximation methods that might allow the GPGO algorithm to work with higher-dimensional objective functions.
Sensor set selection

The sensor set selection algorithm presented in Chapter 9 could be extended in multiple ways. To begin, we could consider incorporating various extended cost functions into our optimization procedure. For example, we could apply our routine to manage a set of mobile sensors. We could accomplish this by, say, incorporating a cost proportional to the total distance moved by all sensors into the GPGO loss function (8.1.1). With the loss modified appropriately, the GPGO algorithm would then automatically select the correct Bayesian optimal action at every stage of the optimization process.

We could also consider performing optimization over sensor sets of different sizes. Suppose we defined a cost associated with including an additional sensor into an existing network. Our algorithm could then automatically select the optimal set size by balancing the trade-off between the cost of larger networks against the potentially better predictive quality they might achieve.

Each of these modifications to the cost function would be very easy to incorporate into our method, due to our straightforward Bayesian decision-theoretical treatment of the problem.

Finally, we could consider metrics between sensor networks other than the earth mover’s distance that we suggested. Kondor and Jeffers (2009), for example, proposed fitting Gaussian distributions to each set and using the Bhattacharyya distance between these distributions as a metric between point sets. We could also consider building a metric directly from the prediction GP’s covariance function—for example, we could try using the Hellinger distance between the covariance matrices over the locations of interest that are associated with each of the networks under consideration.
The multivariate Gaussian or normal distribution has a number of useful properties, which we will briefly outline here. The $d$-dimensional multivariate Gaussian distribution is specified by two parameters (its first two moments): a $d \times 1$ mean vector $\mu$ and a $d \times d$ symmetric, positive-definite\footnote{The multivariate Gaussian distribution exists for positive-semidefinite covariance matrices as well; however, the probability density function does not exist.} covariance matrix $\Sigma$. Given these, the multivariate Gaussian probability density function is given by

$$p(x \mid \mu, \Sigma) = \mathcal{N}(x; \mu, \Sigma) = \frac{1}{\sqrt{\det(2\pi \Sigma)}} \exp \left( -\frac{(x - \mu)^T \Sigma^{-1} (x - \mu)}{2} \right).$$

An important property of the Gaussian distribution is that conditional and marginal distributions of Gaussians are Gaussian. To elaborate, suppose that $X$ and $Y$ are jointly Gaussian-distributed random variables on a common probability space:

$$p(x, y) = \mathcal{N}\left( \begin{bmatrix} x \\ y \end{bmatrix}; \begin{bmatrix} \mu \\ \nu \end{bmatrix}, \begin{bmatrix} \Sigma & P \\ P^T & \Sigma \end{bmatrix} \right).$$

The marginal distribution of $X$ is Gaussian:

$$p(x) = \mathcal{N}(x; \mu, \Sigma), \quad (A.1)$$

and the distribution of the conditional $X \mid Y$ is Gaussian:

$$p(x \mid y) = \mathcal{N}\left( x; \mu + P \Sigma^{-1} (y - \nu), \Sigma - P \Sigma^{-1} P^T \right). \quad (A.2)$$

If $X$ and $Y$ are two independent Gaussian-distributed random variables,

$$p(x) = \mathcal{N}(x; \mu_1, \Sigma_1),$$
$$p(y) = \mathcal{N}(y; \mu_2, \Sigma_2),$$

then their sum $Z \triangleq X + Y$ is also Gaussian, with distribution

$$p(z) = \mathcal{N}(z; \mu_1 + \mu_2, \Sigma_1 + \Sigma_2). \quad (A.3)$$

Another useful identity is the following. If $X$ has a multivariate Gaussian distribution

$$p(x) = \mathcal{N}(x; \mu, \Sigma),$$

and the conditional $Y \mid X$ has a multivariate Gaussian distribution with a mean that is an affine map of $x$:

$$p(y \mid x) = \mathcal{N}(y; Ax + b, T),$$
then the joint distribution of \((X, Y)\) is given by

\[
p(x, y) = \mathcal{N}
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
\begin{pmatrix}
  \mu \\
  \Lambda \mu + \mathbf{b}
\end{pmatrix}
\begin{pmatrix}
  \Sigma & \Sigma \Lambda^T \\
  \Lambda \Sigma & \Lambda \Sigma \Lambda^T + T
\end{pmatrix}.
\]

(\ref{eq:joint_distribution})

and, applying (\ref{eq:linear_combination}) to (\ref{eq:joint_distribution}), the marginal distribution of \(Y\) is given by

\[
p(y) = \mathcal{N}(y; \Lambda \mu + \mathbf{b}, \Lambda \Sigma \Lambda^T + T).
\]

(\ref{eq:marginal_distribution})

Finally, affine transformations of Gaussians are Gaussian. If \(X\) has a multivariate Gaussian distribution

\[
p(x) \equiv \mathcal{N}(x; \mu, \Sigma),
\]

then the distribution of the affine transformation \(Y \equiv \Lambda X + \mathbf{b}\) is

\[
p(y) = \mathcal{N}(y; \Lambda \mu + \mathbf{b}, \Lambda \Sigma \Lambda^T).
\]

(\ref{eq:affine_transformation})


BIBLIOGRAPHY


BIBLIOGRAPHY


NVIDIA CORPORATION (2010). NVIDIA CUDA programming guide 3.0. Available at:


http://www.robots.ox.ac.uk/~parg/pubs/SPQItechreport.pdf


BIBLIOGRAPHY


