LUCERNE UNIVERSITY OF APPLIED SCIENCES AND ARTS

REPORT VM2

Gaussian Processes vs. Convolutional Networks on Medical Imaging

Author:
Christian VON GUNTEN

Co-Advisors:
Dr. Prof. Ruedi ARNOLD
Dr. Prof. Marc POULY

Expert:
Dr. Stanislas NANCHEN

A report submitted in fulfillment of the requirements
for the module VM2 as part of the Master of Science in Engineering

in the

Algorithmic Business Research Team ABIZ

December 18, 2019

Declaration of Authorship

I hereby declare that I have prepared the present work independently and have used nothing other than the specified aids. All text sections, citations or contents of other authors used have been explicitly marked as such.

Signed:

Date:

LUCERNE UNIVERSITY OF APPLIED SCIENCES AND ARTS

Abstract

Master of Science in Engineering

Gaussian Processes vs. Convolutional Networks on Medical Imaging

by Christian VON GUNTEN

The current success of modern machine learning techniques - neural networks (NN) in particular – is backed by unprecedented availability of data and computational power. In domains such as medical imaging, they fall short because high quality labeled datasets are expensive. Optimization of millions of parameters poses various challenges regarding regularization and interpretability. As an alternative, this report investigates Gaussian Processes (GP). They are a non-parametric, kernel-based and fully Bayesian approach to Machine Learning.

A comprehensive introduction to the topic is given in the first part. A step by step derivation of the predictive distribution for GP Regression is a core contribution of this work. Various examples demonstrate the application of GPs to binary and multiclass classification. Multiclass likelihoods are still supported by only few librarys and in rather naive form.

Specialized image-kernels and corresponding inference algorithms are still under active research. The `Jass` dataset served therefore as a surrogate for final experiments. The performance of various kernels was evaluated based on classical cross-validation accuracy and the marginal likelihood – a Bayesian model selection criterion. Conclusions vary between criteria. A further benchmark investigates generalization performance of (deep) neural networks versus Gaussian Processes. Multiple GP models outperformed the NNs in a six-class classification task on middle sized training sets. Though, there is no evidence that GPs are generally more data efficient than neural networks.

Acknowledgements

I would like to thank my project advisors Dr. Prof. Ruedi Arnold and Dr. Prof. Marc Pouly of the ABIZ research unit at HSLU. Marc, for pointing me to such an enlightening topic and helpful feedback regarding formal aspects of my derivations. Ruedi, for the continuous encouragement to invest in fundamentals and to write down insights in an instructive way.

I am also grateful to Ludovic Amruthalingam and Thomas Koller for providing me information and datasets for real world task experiments.

Finally, special thanks to Benjamin Haymond for the mentoring regarding technical and scientific writing.

Contents

List of Figures

List of Tables

List of Abbreviations

w.r.t.	**w**ith **r**espect **t**o
l.h.s.	**l**eft **h**and **s**ide (of equation)
r.h.s.	**r**ight **h**and **s**ide (of equation)
iff	**i**f and only **if**
ABIZ	**A**lgorithmic **B**usiness Research Team
EM	**E**xpectation **M**aximization
EP	**E**xpectation **P**ropagation
GLM	**G**eneralized **L**inear **M**odel
(D)GP	(**D**eep) **G**aussian **P**rocess
GPR	**G**aussian **P**rocess **R**egression
GPC	**G**aussian **P**rocess **C**lassification
KL	**K**ullback **L**eibler Divergence
MAP	**M**aximum **A**-**P**osteriori
MCMC	**M**arkov **C**hain **M**onte **C**arlo
ML	**M**achine **L**earning
MLE	**M**aximum **L**ikelihood **E**stimate
(R)MSE	(**R**oot) **M**ean **S**quared **E**rror
(D)NN	**D**eep **N**eural **N**etwork
SVM	**S**calable **V**ector **M**achine
VI	**V**ariational **I**nference

Notation

The notation used throughout this report is based on *Gaussian Processes for Machine Learning* by Rasmussen and Williams 2005.

$\doteq$	Equality relationship that is true by definition
$\propto$	Proportional up to constant factor
$\leftarrow$	Assignment
$\approx$	Approximately equal
$\ll, \gg$	Much less/greater than
$[a, b)$	The real interval between a and b including a but not including b
$\{0, 1, 2\}$	Discrete set
$\arg\max_a f(a)$	A value of a at which $f(a)$ achieves its maximum
$\cancel{ab} - \cancel{ab}$	Indicates that two terms are removed because they cancel out
$\cancel{ab}$	Indicates that a term is constant w.r.t. the function argument and is *moved* into a *collector variable*.
$\mathbb{R}$	Set of real numbers
$\mathbb{N}$	Set of integers
x	Lower-case letters denote scalars
$\mathbf{a}$	Bold lower-case letters denote vectors
a_i	Denotes the i^{th} component of vector $\mathbf{a}$ - a scalar
$\mathbf{A}$	Bold upper-case letters denote matrices
$\mathbf{a}_i$	Denotes the i^{th} row of matrix $\mathbf{A}$ - a row vector
$\mathbf{A}^T, \mathbf{a}^T$	Transpose of a matrix or vector
$\lvert A \rvert$	Determinant of a matrix
$\mathbf{w}$	Parameters of the model
$\boldsymbol{\theta}$	Hyperparameters of the model
$\mathcal{D}$	Dataset
$\mathbf{X}$	N sample inputs with D features - matrix with N rows and D columns
$\mathbf{x}_i$	Denotes the feature vector of the i^{th} sample in $\mathbf{X}$ - i.e. the i^{th} **row transposed**
$\mathbf{y}$	N scalar sample outputs - N-dimensional vector

X	Upper-case letters denote random variables
X_i	Denotes the i^{th} component of random vector X
$P(X)$	Probability function for random variable X
$P(y)$	Short notation for $P(Y = y)$. The probability of random variable Y taking the value y
$P(X, Y)$	Joint probability of random variables X and Y
$P(X\|Y)$	Conditional probability of random variables X given Y
$f_X(x)$	Probability mass/density function for random variable X
$F_X(x)$	Cumulative density function for cont. random variable X
$\mathbb{E}[X]$	Expectation of random variable X, i.e., $\mathbb{E}[x \in X] \doteq \sum_x p(x)x$
$\mathbb{E}_{X\|Y}[X]$	Expectation of random variable X given Y, i.e., $\mathbb{E}_{X\|Y}[X] \doteq \sum_{x \in X} p(x\|y)x$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate Gaussian distribution
$\mathrm{m}(\mathbf{x})$	Mean function for a single feature-vector $\mathbf{x}$
$\mathrm{M}(\mathbf{X})$	Vectorized mean function. Returns vector of means $\mathrm{m}(\mathbf{X}_i)$ for each row in $\mathbf{X}$
$\mathrm{k}(\mathbf{x}, \mathbf{x}')$	Kernel function. A symmetric, positive semi-definite function.
$\mathrm{K}(\mathbf{X}, \mathbf{X}')$	Vectorized Kernel function. Returns symmetric matrix of pairwise covariances $\mathrm{k}(\mathbf{x}_i, \mathbf{x}'_i)$ between rows of $\mathbf{X}$ and $\mathbf{X}'$.

Chapter 1

Introduction

1.1 Problem Description and Purpose

The success of deep neural networks (DNN) is backed by massive datasets and computational power. Deep neural networks are able to autonomously learn relevant features from raw input data. The first layers of a DNN are therefore often depicted as *hierarchical feature extractors*. Low-level features from the first layers are successively combined into more complex concepts.

In image analysis for example, edges and corners form basic shapes. Two circles might then represent eyes and eventually, eyes combined with a triangle (nose) and a rectangle (mouth) might represent the concept of a face. Based on these more abstract concepts, the last layers finally solve the original learning task (e.g. classification). This is a major advantage over manual feature engineering in traditional Machine Learning[1].

There is a trade-off however. DNNs require a lot of data to autonomously learn useful hierarchical features for a specific task. A neural network is a very generic form of a parametric model. The term *parametric* implies that the model is parameterized by a finite set of parameters[2]. The number of parameters controls the capacity of the model. In DNNs for example, the number of parameters is dependent on the number of layers (depth of hierarchy) and the number of neurons per layer. Usually this has to be defined a priori. If the model is too simple, it will not be able to model the data well. This problem is denoted **underfitting**. It can be imagined as trying to fit a curve by a straight line. If the capacity is too high, the model might learn even irrelevant noise in the data. Especially, if the number of parameters exceeds the number of datapoints, the model is able to learn the dataset by heart - similar to a simple lookup-table. This has bad impacts for unseen examples and is denoted as **overfitting** - the model is not able to *generalize* from data in the training set to unseen data.

[1] For example the manual design of filters to detect edges, corners etc. in computer vision

[2] We stick to the definition by (Russell and Norvig 2009, chapter 18.2). Hyperparameters, such as optimization learning-rate, number of neurons etc. are not considered as parameters in this sense.

Modern neural networks consist of millions of parameters. They are therefore very prone to overfitting if they are not trained on equally massive datasets. In domains such as medical imaging, the collection of datasets - and the labeling in particular - might be costly. For such cases, there are several types of measures to reduce overfitting. **Regularization** constrains the capacity of the model by e.g. limited training time (early-stopping), penalty-terms for large parameter values (L1-, L2-regularization) and many more. **Data-Augmentation** increases the size of the dataset by adding synthetic data such as slightly rotated versions of original images. **Unsupervised Pretraining** might be used when labels exist only for a small subset of the data and **Transfer Learning** is used to reuse common lower-level features learned in similar tasks or datasets. The problem with most of this measures is that they introduce additional hyperparameters (e.g. type and amount of regularization, degree of rotation etc.). Finding appropriate values for all these hyperparameters is itself a difficult combinatorial problem.

In this work, we investigate an alternative and still rather unknown approach - **Gaussian Processes** (GP). Models based on GPs are sometimes claimed to be less *data-hungry* or less prone to overfitting on small or moderately sized datasets[3]. As a **non-parametric** model, the capacity is controlled directly by the size of the training set instead of the number of parameters. Other well known and successful models in this family are Support Vector Machines (SVM), Decision Trees and K-Nearest-Neighbors.

This work introduces functioning and characteristics of GPs, contrasts them with other methods and evaluates their conceptual and practical applicability. In particular, the claim of being less data-hungry is examined. Therefore, a performance comparison between GPs and NNs on various sizes of training sets was conducted.

1.2 Goal

The original goals of this work are[4]:

The *Algorithmic Business Research Team* (ABIZ) successfully applied and evaluated various methods such as Deep Learning and SVMs to rather large datasets. One focus of project *SkinApp* is to classify healthy or diseased skin based on medical image data. Beside neural networks, Machine Learning using Gaussian Processes is an interesting approach to supervised learning. In particular, it is often claimed that GPs would require less data than (convolutional) neural networks. In medical image analysis, the manual labeling of training data is expensive. It cannot be supported by laity (crowd worker) because of its complexity. Furthermore, each image has to be judged by multiply experts to ensure good data quality.

[3] Though, an explicit formulation of this claim was not found during research
[4] Translated from *VM2 Aufgabenstellung*

This motivates the research for methods with similar performance but better data-efficiency. In this work the performance and data-efficiency of Gaussian Processes should be investigated. The ISIC[5] dataset and existing models based on convolutional neural networks serve as a baseline.

Main goals:

1. Introduction to Gaussian Processes (background, functioning, classification)
2. Performance evaluation - comparison to existing results in Skin-App project
3. Fine-tuning and improvements of Gaussian Processes
4. Studying the scalability of Gaussian Processes w.r.t. increasingly sized datasets
5. Interpretation of results, recommendations & outlook

Remark

In the meeting between M. Pouly and C. von Gunten on December 5, 2019, it was decided to stick to the `Jass` dataset for the scope of this work. Experiments on the `ISIC` task were postponed for future work.

The main reason was the additional complexity introduced by high-dimensional image data.

1.3 Structure

This report consists of two main parts. Chapter 2 summarizes underlying theory and functioning of Gaussian Processes. Chapter 3 keeps record about the various conducted experiments.

Sections annotated with an **asterisk*** introduce **advanced topics**. They are referenced in subsequent sections and are therefore introduced in suitable context. Nevertheless, their depth might disturb the flow of reading. It is suggested to only scan those sections in a first reading and consider them in detail upon reference.

Python **code** used in experiments and for illustrations is hosted on GitHub[6]. Jupyter notebooks contain main threads of execution of experiments. In this report, notebooks are referenced as "`ID_Name.ipynb`". The repository includes a docker image with all the necessary tools and dependencies for reproduction.

1. Start local docker container using: `docker-compose up`
2. Open JupyterLab at: `http://localhost:8888`

[5] International Skin Imaging Collaboration - https://www.isic-archive.com/
[6] https://github.com/vogi23/VM2-Code

1.4 Gaussian Process - A First Intuition

"A Gaussian Process is a generalization of a multivariate Gaussian distribution to an infinite number of dimensions." Anonymous

This widely used definition of a **Gaussian Process** (GP) is quite abstract. The goal of this section is to develop a first intuition for functioning and applications of GPs.

A main motivation behind Gaussian Processes is **non-linear regression**. Real valued outputs $\mathbf{y}$ are mapped to inputs $\mathbf{X}$ through a parameterized non-linear function. N denotes the number of training input-output pairs. The form of the function (polynomial, sinusoidal, a neural network etc.) implies a prior assumption of how the *true* underlying function might look like. Figure 1.1 illustrates an example using the *family* of second-order polynomial functions.

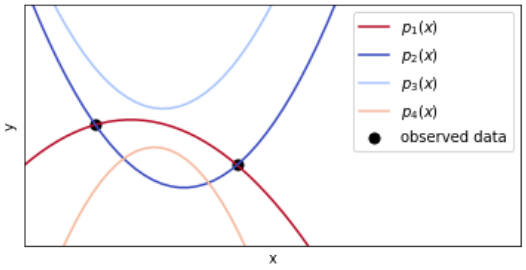

FIGURE 1.1: Four functions from the family of second-order polynomials. Only two of them explain the observed datapoints well.

Additionally, an error component ϵ is used to address small measuring errors of the observations. Errors are usually assumed to be independently Gaussian distributed. The full model is therefore:

$$\mathbf{y} = f(\mathbf{x}) + \epsilon \tag{1.1}$$

Instead of a parameterized function, **Gaussian Processes use a random vector** to map inputs to outputs. Imagine a 3-dimensional random vector X. A scatter plot is constructed with the components of X on the horizontal and the values of the components on the vertical axis (cf. figure 1.2.a).

Using this setting, a set of N datapoints can be interpreted as one possible realization of a random vector in $\mathbb{R}^N$. There is no restriction that the positions of the components along the horizontal axis are equidistant. Figure 1.2.b shows such a *dataset* sampled using a N-dimensional Gaussian.

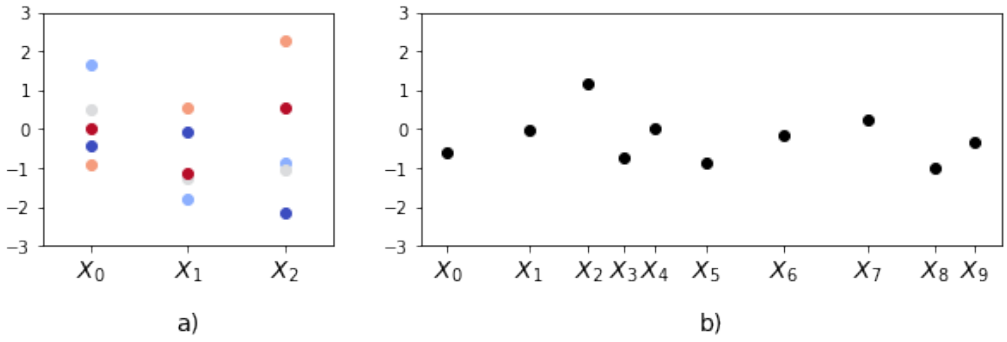

FIGURE 1.2: a) 5 samples of a 3-dimensional random vector X. Each color corresponds to one sample of X. b) One sample of a 10-dimensional random vector X. This could represent 10 observed datapoints in 2D. For both plots X, is distributed according to a multivariate Gaussian with zero mean vector and identity covariance matrix.

Connecting the datapoints of one sample results in a piece wise linear function. **A random vector does therefore also define a *family* of functions**. Each sample corresponds to one particular function in this family. The overall shape of the functions, produced by such a random vector X, is **characterized by its distribution**. For Gaussian Processes, the underlying distribution is always a **multivariate Gaussian**.

Multivariate Gaussian Distribution

Multivariate Gaussians will be introduced in detail in section 2.2.1. For the moment, the following knowledge is sufficient:

A Gaussian distributed random vector X is characterized by a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. The covariance matrix defines correlation between components of random vector X. Components with positive covariance have positive correlation and vice versa.

If two components have **high positive covariance**, they will have **very similar values**[a].

[a] Assuming the same mean and variance for every component.

Figure 1.3 and 1.4 show various *functions* resulting from Gaussians with different covariance matrices.

Gaussian Processes assume that the data is modeled by such a function. Because it is unclear which particular function is the *true* one, all of them are considered. Figure 1.5 shows multiple sampled functions which pass through the two *observed* points. All of them could therefore theoretically be the *true* underlying function. To **predict** the value for another component (e.g. X_{10}), **the mean** of all those function values

can be used. The variance of the values allows to specify confidence bounds around this mean.

The core idea of Gaussian Processes is to use a multivariate Gaussian distribution for modeling and prediction. Subsequent sections explain various details, important properties and extensions. Benefits and challenges in practical application are investigated and emphasized in experiments in chapter 3.

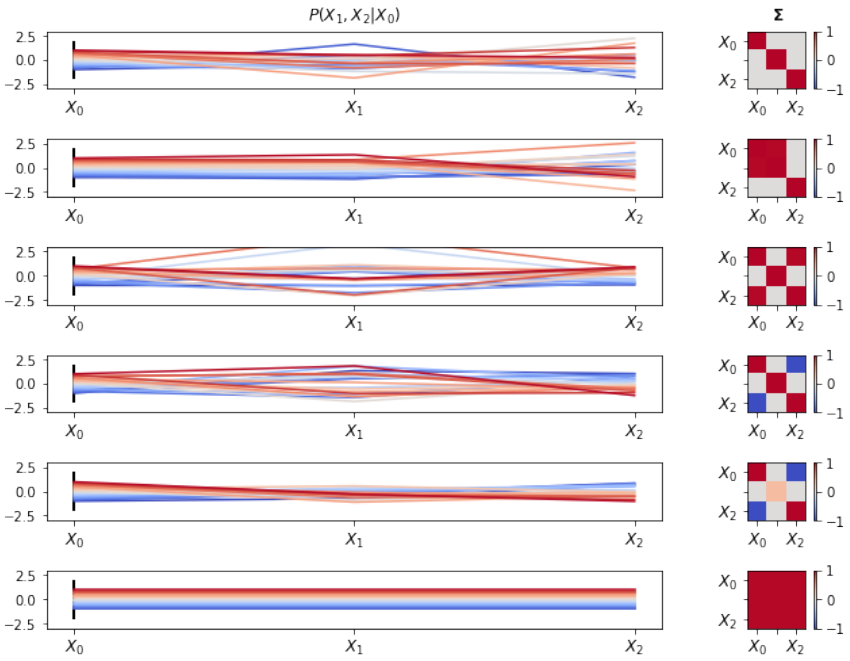

FIGURE 1.3: Effect of the covariance on correlation between components. The plots in the left columns show 20 *functions* sampled from a 3-dimensional Gaussian. Each color corresponds to one sampled function. The values of X_0 have been fixed to values between -1 and 1 two illustrate the effects of covariance. The vertical black lines indicate this *conditioning* on X_0. The right column shows the respective covariance matrices. $\boldsymbol{\mu} = \mathbf{0}$ in all examples.

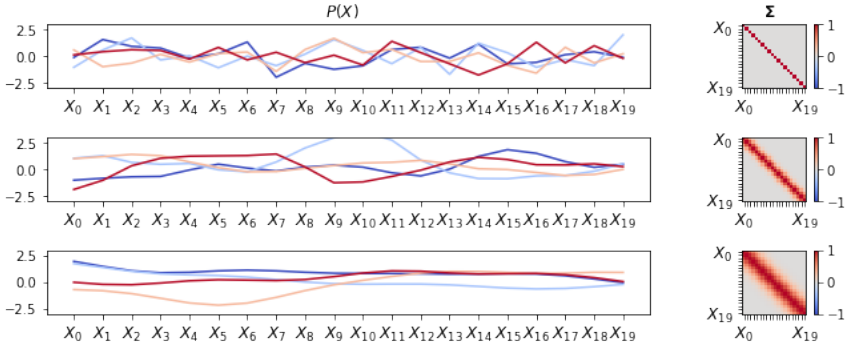

FIGURE 1.4: Effect of the covariance on the overall shape of sampled *functions*. In the top example, $\mathbf{\Sigma}$ is the identity-matrix which results in a *white-noise process*. For the further examples, the covariance decreases exponentially in the squared horizontal distance between components. The slower the covariance decreases, the smoother the sampled functions are. This is the most common form of covariance matrix used for GPs and is called *squared exponential* (cf. eq. (2.29)).

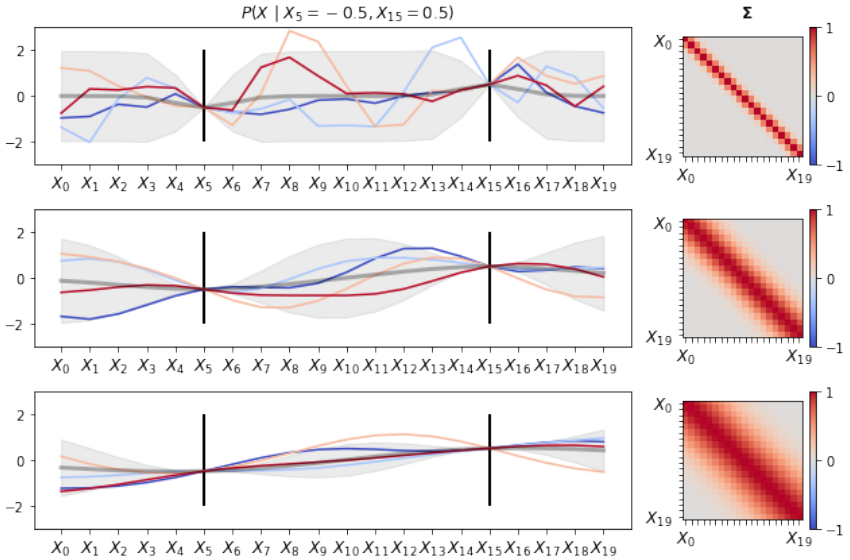

FIGURE 1.5: Sampled *functions* which are conditioned to pass through 2 points. Bold gray line represents the mean of the sampled functions. The light-gray region is a 95% confidence region. I.e. if an infinite number of functions were sampled, 95% of them would be inside this region. Note that for a *slower* decreasing covariance function, the confidence bounds also increase *slower* away from the fixed points.

1.5 Historical Remarks and Related Work

1.5.1 History of Gaussian Processes

One of the first known applications of Gaussian Processes is prediction and smoothing of time-series data by Kolmogoroff 1941 and Wiener 1949. Under the name *kriging*, Matheron 1973 and Journel and Huijbregts 1978 applied GPs in the field of geostatistics. Several other applications to spatial data followed - summarized by Cressie 1993. Neal 1995 derived a connection between GPs and infinite (wide) neural networks. This gave rise to increasing interest and various research on GPs[7]. Eventually, Rasmussen and Williams 2005 gave a comprehensive discussion of GPs in the context of Machine Learning. Their book titled *Gaussian Processes for Machine Learning* is still considered the most significant introduction and reference work on Gaussian Processes to date.

1.5.2 Performance Benchmarks

Most of recent publications around GPs report their performance on standardized tasks like MNIST or CIFAR10. However, current research is often still focused on scalability and more accurate approximate inference. A lot of effort is put on interpretable models, to get rid of unrealistic assumptions and the need to choose hyperparameters[8]. This seems typical for the *Bayesian Community*.

In contrast, the results of state of the art neural networks are often backed by huge datasets and massive computational resources. Furthermore, extensive hyperparameter- and neural architecture search systematically overestimate the general performance of such models. A fair comparison of GPs with other state-of-the-art algorithms is therefore hard.

Rasmussen 1997 proposed a systematic approach to evaluate different algorithms. He compared five common models[9] with a GP using MAP estimates of the hyperparameters and a fully Bayesian GP using Monte-Carlo sampling. Each algorithm was trained on differently sized training sets[10]. Regarding this work, the relevant conclusions of Rasmussen are:

- Performance[11] of GPs and NNs are often close irrespective of training set size.
- In contrast to NNs, GPs do well irrespective of degree of non-linearity, noise-level, input-dimensionality and data-family.
- A Bayesian treatment of NNs (using sampling) often outperforms conventional NNs (trained with back-propagation). This is true even if sampling is

[7] Rasmussen and Williams 2005, p. 30.
[8] which generally increases the risk of overfitting
[9] Linear Regression, K-Nearest-Neighbors, Regression Splines (MARS), a Neural Network with Early-Stopping and a Bayesian Neural Network using Monte-Carlo Sampling
[10] Training set sizes: 64, 128, 256, 512, 1024
[11] Squared Error, Absolute Error or Negative Log Density was used depending on the experiment

conducted for a short time[12]. This contrasts with the belief that Bayesian methods are too slow for practical importance.

- Which method to choose still depends mostly on its efficiency for the given training set size.

There is **no explicit statement or evidence** that GPs are generally **more data-efficient**.

1.5.3 Gaussian Processes on Image Data*

High-dimensional and/or spatial data, such as images, contain a lot of local structure. In (deep) neural networks the convolution operations[13] revealed to be very effective feature extractors.

Hassouna 2016 showed that spatial contextual information improves image classification performance of a GP. The task was to classify single pixels of a satellite image into classes such as *asphalt*, *water*, *sand* etc. For the baseline GP, the input was just the color of the single pixel x_i. For the spatial contextual GP, the input was enhanced by the pixels in an N×N spatial neighborhood of x_i. Adding such relevant information is likely to improve classification performance. However, it does not seem to have much in common with successful techniques to classify entire images. For example, it does not leverage important properties such as *translation invariance*.

The approach of Wilson et al. 2016 is to transform the raw inputs with a (convolutional) neural network. Then a common kernel is applied to ensure a valid covariance matrix. The (many) weights of the network have to be considered as kernel hyperparameters. This increases complexity of hyperparameter optimization.

van der Wilk, Rasmussen, and Hensman 2017 proposed a method using a *true* convolutional kernel. It is based on the concept of additive kernels which model *OR* relations between covariances[14]. The input image $\mathbf{X}$ is divided into P patches. The covariance between two pictures is then determined by comparing each patch from the first with each patch of the second image. Similar kernels have been proposed before[15][16]. However, the additional computational complexity of P^2 per element of the covariance matrix prevented practical applications so far. van der Wilk, Rasmussen, and Hensmans main contribution was a sparse approximation well-tailored to this convolutional kernel. This allows to benefit from the properties of the convolution operation while maintaining fast and accurate posterior inference.

[12] i.e. comparable run-time to conventional NN
[13] LeCun et al. 1999.
[14] Duvenaud 2014.
[15] Mairal et al. 2014.
[16] Pandey and Dukkipati 2014.

1.5.4 Need and Types of Approximations*

Application of GPs is generally limited by one of two reasons. For both, the key contributions to understand the underlying concepts and methods implemented in state of the art librarys are listed:

1. The computational time complexity of $O(N^3)$ limits the size of the training set N to some (ten) thousand samples at most. Here, the main thread of research focuses on **sparse approximations** of the covariance matrix (using only a small number of *pseudo* or *inducing* inputs): Cf. (Snelson and Ghahramani 2006), (Quiñonero-Candela, Ramussen, and Williams 2007) and (Titsias 2009).

2. If the likelihood is non-Gaussian (as in e.g. GP Classification), the posterior has no analytical solution and marginalizing the latent values becomes intractable. For this case, methods for **approximate inference** were proposed by (Williams and Barber 1998), (Minka 2001) and (James. Beal 2003).

According to Hensman, Matthews, and Ghahramani 2015, there has been little overlap of those fields. They claim that their combined approach allows to scale GP Classification to millions of data points.

1.5.5 Gaussian Processes in the Context of Deep Learning*

Inspired by the success of deep learning, there is active research under the name **Deep Gaussian Processes** (DGP). Despite the name, a DGP is not a GP overall. As in deep learning, inputs are mapped to outputs through a number of intermediate layers. In a DGP, the layers are GPs.

Damianou, Titsias, and Lawrence 2011 showed how the complete latent space between two stacked GPs can be marginalized variationally. Damianou and Lawrence 2013 extended this idea to an arbitrary number of layers. The latter approach is used for experiments described in section 3.8.

Both ideas use approximate inference which assumes strong independence between layers. Salimbeni and Deisenroth 2017 claims that the true posterior is likely to exhibit high correlation between layers. Their method removes those assumptions and simultaneously claims to make DGPs applicable to large datasets.

Chapter 2

Theory

This chapter describes characteristics, underlying theory and application of Gaussian Processes. Before diving into GPs itself, relevant terms and concepts accompanying GP literature are introduced. As a newcomer to the field - and to the Bayesian methodology in particular - literature research and understanding of these topics revealed crucial. Substantial effort in this fundamentals eventually allowed me to interpret literature on GPs and to evaluate them within the overall field of Machine Learning.

2.1 Statistical Inference

The goal of statistical inference is to learn something about a *process* which generated data at hand. The process is often described by a stochastic model i.e. a model which contains at least one source of uncertainty.

Consider the following example. We observe that the outcome of flipping a coin ten times is seven heads and three tails. This process of flipping a coin N times can be *modeled* by a binomial distribution. The binomial distribution has a parameter $p \in [0, 1]$. It describes the *fairness* of the coin. A value of $p = 0.5$ implies a fair coin. Here, the goal of inference is to learn if the coin was fair or biased. Formally, inference estimates the value of p given the observed data. The uncertainty about the value of p is decreased.

After estimating this *parameter* of the model, it can be used to *predict* the outcome of future flips. Inference assumes therefore, that the observed data is itself just a sample from an overall larger population. This can be contrasted with *descriptive statistics*, which describes and summarizes existing data without assuming a larger population.

2.1.1 Paradigms of Statistics

Different forms of inference have been established. Part III of the book *Philosophy of Statistics*[1] describes *Four Paradigms of Statistics*. They are however not mutually exclusive.

- Classical/Frequentist Statistics
- Bayesian Paradigm
- Likelihood Paradigm
- Akaikean Paradigm

A discussion on commonalities, relative advantages and fields of applications is still rather controversial and out of the scope of this work. This work focuses on the first two and most common paradigms. **Frequentist and Bayesian** statistics both use **Likelihood Functions**. They describe how likely it is that a model (e.g. a binomial distribution with $p = 0.23$) generated the observed data. Note that likelihood functions are different from the **Likelihood Paradigm**, which is considered a rather minor or less known paradigm. The **Akaikean Paradigm** is often used in the context of model selection (see chapter 2.6).

A major distinction of the Frequentist and Bayesian paradigm is the **interpretation of parameters and data**. In the Frequentist perspective, parameters are deemed *fixed but unknown*. Given a fixed underlying model, the observed data is interpreted as just one possible outcome. In the Bayesian perspective, the data is considered fixed. However, it is assumed that there are many possible models (e.g. each a binomial with different p) which could have generated this data. Each possible value of p is then assigned a probability. It corresponds to the likelihood of the data under this model.

Parameters are therefore described by a single value (**point-estimate**) in the Frequentist paradigm, but by a **probability distribution** over values in the Bayesian paradigm.

2.1.2 Maximum Likelihood Estimate

The Maximum Likelihood Estimate (MLE) is one popular method to estimate the parameters of the underlying process. The likelihood of a process is therefor formulated as a function $\mathcal{L}(\mathbf{w})$ of the parameters $\mathbf{w}$. For the binomial model, the parameter vector $\mathbf{w}$ is $[p]^T$. Finding parameters, which maximize this function, is then an optimization problem.

$$\text{MLE}_{\mathbf{w}} = \text{argmax}_{\mathbf{w}} \, \mathcal{L}(\mathbf{w}) \tag{2.1}$$

[1] Bandyopadhyay and Forster 2011.

For simple models such as the binomial model or linear regression, there are analytical solutions[2]. Otherwise, iterative approaches such as *Gradient Ascent* or *Newtons method*[3] may be used to find a (local) maxima.

The maximum likelihood estimation returns a single (most-likely) estimate for the parameters. It is therefore often assigned to the Frequentist paradigm. However, there is a very similar concept called MAP estimate in the Bayesian paradigm (see next section).

2.1.3 Inference in Latent Variable Models

Latent variables cannot be observed or measured directly. They can be divided in local and global latent variables. *Local* variables can be imagined by missing features of datapoints, i.e. each datapoint has a relevant feature which is however not observed or measured. Parameters of the underlying model (like p for the binomial) are referred to as *global* latent variables. Their values are not related to a single datapoint. Nevertheless, they are generally unknown and have to be estimated.

Latent variables are used primarily in two scenarios. If there are missing values for some specific datapoints[4], their value can be represented by latent variables. On the other hand, they are often used because they allow either a convenient or more adequate formulation of the model. A typical example is the task of modeling people's heights.

An imaginary dataset consists of the measured heights $[h_1, ..., h_N]^T$ of N people. A simple approach would be to assume the heights to be Gaussian distributed. Inference consists now of estimating the mean μ and variance σ^2 of this underlying distribution. Common knowledge indicates that women and men have distinct average heights. A more adequate model might therefore use two separate Gaussians with corresponding parameters $\mathbf{w} = [\mu_w, \sigma_w^2, \mu_m, \sigma_m^2]^T$.

Estimating these parameters directly is only possible if the gender of each observation is known. Because this information is missing, it can be represented by local latent binary[5] variables $\mathbf{f} = [f_1, ..., f_N]^T$. The likelihood of the model is then:

$$\mathcal{L}(\mathbf{w}, \mathbf{f}) = \prod_i^N P(h_i | f_i, \mathbf{w}) \tag{2.2}$$

$$= \prod_i^N \mathcal{N}(h_i | \mu_m, \sigma_m^2)^{f_i} * \mathcal{N}(h_i | \mu_w, \sigma_w^2)^{1-f_i} \tag{2.3}$$

[2] By setting the derivative to zero and solve for the parameters

[3] Assuming the function is (twice) differentiable w.r.t. the inputs

[4] Because of temporary sensor failures or because data was aggregated from various sources for example

[5] Where for example 0 = woman and 1 = man

where $\mathcal{N}(\cdot|\mu, \sigma^2)$ denotes the probability density function of the Gaussian distribution.

Unfortunately, the parameters of the Gaussians and the latent variables do influence each other. To estimate the parameters of the Gaussians, the assignment of each measurement to a gender($\mathbf{f}$) is necessary. Without further information, the gender could only be estimated in terms of shorter distance to the mean of women and men respectively. This results in a kind of *chicken-egg* problem. Fortunately, there is a solution.

"A general technique for finding maximum likelihood estimates in latent variable models is the expectation-maximization (EM) algorithm."(Bishop 2006, p. 424).

Expectation Maximization*

The EM algorithm[6] is an iterative procedure which performs an expectation and a maximization step alternately.

In the expectation step, the expected value of the likelihood $\mathcal{L}(\mathbf{w}, \mathbf{f})$ is formulated w.r.t. the posterior[7] of the latent values $\mathbf{f}$.

$$\mathbb{E}_{P(\mathbf{f}|\mathbf{h},\mathbf{w}_{\text{old}})}[\mathcal{L}(\mathbf{w}, \mathbf{f})] = \prod_{i}^{N} \sum_{f_i \in \{0,1\}} P(h_i|f_i, \mathbf{w}) P(f_i|h_i, \mathbf{w}_{\text{old}}) \tag{2.4}$$

Note that this is still a function of the model parameters $\mathbf{w}$. In the maximization step, this expected likelihood function is then maximized w.r.t. $\mathbf{w}$.

Algorithm 1: Expectation Maximization

Input: Observed data h_i for $i \in \{1..N\}$
Output: Optimal model parameters $\mathbf{w}^*$
Initialize model parameters $\mathbf{w}_{\text{old}} = [\mu_m, \sigma_m^2, \mu_w, \sigma_w^2]^T$ arbitrary;
repeat
  Expectation: Define $g(\mathbf{w}) \doteq \mathbb{E}_{P(f_i|h_i,\mathbf{w}_{\text{old}})}[\mathcal{L}(\mathbf{w}, \mathbf{f})]$
  Maximization: $\mathbf{w}_{\text{old}} \leftarrow \text{argmax}_{\mathbf{w}} g(\mathbf{w})$;
until *convergence of* $\mathbf{w}_{old}$;
$\mathbf{w}^* \leftarrow \mathbf{w}_{\text{old}}$

It is proven that this procedure converges to a local optima. The EM algorithm (cf. algorithm 1) is therefore a simple yet powerful tool when models involve latent variables.

[6] Moon 1996.

[7] For some fixed parameters $\mathbf{w}_{\text{old}}$, the posterior gives the probability that a person of height h_i is of gender f_i.

Similar to MLE however, it gives only a point-estimate of the optimal parameters $\mathbf{w}^*$. Another drawback is that it assumes the posterior to be tractable. This is often not the case for more complex models and limits applicability of EM. In the next sections, more general algorithms for latent variable models will be introduced.

2.1.4 Bayesian Inference

Bayesian inference uses *Bayes Theorem* to update an initial believe about $\mathbf{w}$ based on observed data.

$$P(\mathbf{w}|\mathbf{y},\mathbf{X}) = \frac{P(\mathbf{y}|\mathbf{X},\mathbf{w})P(\mathbf{w})}{P(\mathbf{y},\mathbf{X})}, \quad \text{for } P(\mathbf{y},\mathbf{X}) \neq 0 \tag{2.5}$$

The left term of equation (2.5) is called the **Posterior** - a probability distribution over the values of $\mathbf{w}$. This distribution is what contrasts the point estimate for $\mathbf{w}$ in the Frequentist perspective.

The posterior consists of three factors. $P(\mathbf{y}|\mathbf{X},\mathbf{w})$ represents the **Likelihood** to observe outputs $\mathbf{y}$, given inputs are $\mathbf{X}$ some fixed values of $\mathbf{w}$.

$P(\mathbf{w})$ is the **Prior** distribution which represents an initial believe about $\mathbf{w}$. Sometimes no reasonable believe can be formulated. In this case, a uniform distribution suggests that all values are equally likely *a-priori*.

Finally, $P(\mathbf{y},\mathbf{X})$ is called the **Marginal Likelihood** or the **Evidence**. It has several interpretations. One is the role of a normalization term, which ensures that the posterior remains a valid probability distribution (i.e. $\sum_{\mathbf{w}} P(\mathbf{w}|\mathbf{y},\mathbf{X}) = 1$). The name marginal likelihood stems from the fact that $\mathbf{w}$ has to be *marginalized out* from the joint $P(\mathbf{w},\mathbf{y},\mathbf{X})$ to arrive at $P(\mathbf{y},\mathbf{X})$. Marginalization means to average over all possible values of $\mathbf{w}$. For continuous parameters, this results in a D-dimensional integral, where D is the number of parameters in $\mathbf{w}$. The posterior can be rewritten to:

$$P(\mathbf{w}|\mathbf{y},\mathbf{X}) = \frac{P(\mathbf{y}|\mathbf{X},\mathbf{w})P(\mathbf{w})}{\int_{\mathbf{w}} P(\mathbf{y}|\mathbf{X},\mathbf{w})P(\mathbf{w})} \tag{2.6}$$

Unfortunately, such an integral is often intractable, rendering the whole posterior intractable. There are several approaches in the case of intractable posteriors:

1. The Maximum a Posteriori (MAP) estimate
2. The Laplace approximation of the posterior
3. Expectation Propagation (EP)
4. Variational Inference (VI)
5. Sampling using Markov-Chain Monte-Carlo (MCMC)

Maximum A-Posteriori Estimate

The **Maximum A-Posteriori** (MAP) estimate is defined as the following optimization problem.

$$\begin{aligned} \text{MAP}_{\mathbf{w}} &= \text{argmax}_{\mathbf{w}}\ P(\mathbf{w}|\mathbf{y},\mathbf{X}) \\ &= \text{argmax}_{\mathbf{w}}\ \frac{P(\mathbf{y}|\mathbf{X},\mathbf{w})P(\mathbf{w})}{P(\mathbf{y},\mathbf{X})} \\ &= \text{argmax}_{\mathbf{w}}\ P(\mathbf{y}|\mathbf{X},\mathbf{w})P(\mathbf{w}) \end{aligned} \tag{2.7}$$

Equality in the last step is retained because the marginal likelihood (the denominator) is independent of $\mathbf{w}$. It can therefore be ignored in the maximization problem of eq. (2.7).

The MAP estimate differs from the MLE in eq. (2.1) only by the *prior*. If a uniform prior is chosen, this prior becomes constant w.r.t. $\mathbf{w}$ as well. In this case, the MAP is equal to the MLE. Otherwise, the MAP can be interpreted as a **generalization of the MLE** which allows to incorporate prior believes about the parameters.

Using the MAP instead of the posterior is a trade-off between tractability and gain of information. While the whole posterior distribution over all possible parameter values is often intractable, the MAP is tractable but produces only a point-estimate.

The Laplace Approximation of the Posterior

The idea behind the **Laplace Approximation** is to approximate the true posterior by a Gaussian distribution. Intuitively, the peak of the Gaussian is placed at the MAP. The variance, which defines the curvature of the peak, is determined by the second derivative of the MAP w.r.t. the parameters (also called the Hessian).

This simple approximation works best if the true posterior is expected to have a single mode. If there are multiple modes or if the Hessian at the MAP does not describe the overall curvature of the peek good, the approximation will be poor (cf. figure 2.1).

Nevertheless, in contrast to the MAP estimate, this approximation preserves uncertainties about the parameters in the form of a distribution. It is therefore suited for a fully Bayesian treatment in subsequent steps like prediction.

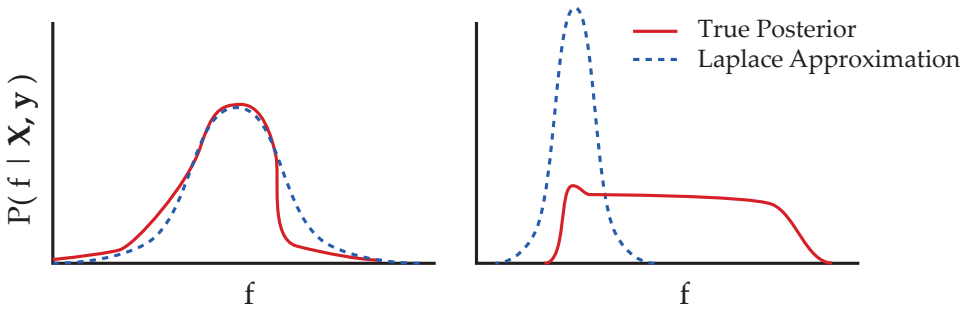

FIGURE 2.1: Laplace approximation of a 1D probability distribution. Approximation is fully controlled by the curvature at the mode of the true density.

Expectation Propagation*

Expectation Propagation (EP, Minka 2001) is a way to approximate intractable probability distributions based on a *message passing technique*. The original idea was to partition very large datasets, perform *local* inference on the pieces separately and then recombine the results into a global distribution.

The posterior usually factorizes into $K = N + 1$ factors. N factors for the likelihood and an additional factor for the prior. Ignoring the normalization term for the moment, the posterior density is proportional to:

$$\begin{aligned} P(\mathbf{f}|\mathbf{X},\mathbf{y}) = f(\mathbf{f}) &\propto \prod_{i=0}^{N-1} P(y_i|f_i) * P(\mathbf{f}|\mathbf{X}) \\ &= \prod_{k=1}^{K} f_k(\mathbf{f}) * f_0(\mathbf{f}) \\ &= \prod_{k=0}^{K} f_k(\mathbf{f}) \end{aligned} \tag{2.8}$$

Single factors are often depicted as *site* distributions. For GPs with non-Gaussian likelihoods, the factors $f_1(\mathbf{f})$ to $f_k(\mathbf{f})$ have arbitrary distributions. EP approximates the true posterior $f(\mathbf{f})$ with a distribution $g(\mathbf{f})$ factorized equivalently:

$$g(\mathbf{f}) \propto \prod_{i=0}^{K} g_k(\mathbf{f}) \tag{2.9}$$

The only restriction is that all the site distributions $g_k(\mathbf{f})$ are in the same exponential family. This assures that any product or division of site distributions remains in the same exponential family and has a simple analytic solutions[8].

[8]Natural parameters or distributions are added in the case of a product, and subtracted in the case of a division of densities

The (unnormalized) *cavity* density is then defined as:

$$g_{-k}(\mathbf{f}) \doteq \frac{g(\mathbf{f})}{g_k(\mathbf{f})}$$

The *cavity* density corresponds to the approximation $g(\mathbf{f})$ without factor $g_k(\mathbf{f})$.

EP iteratively optimizes site distributions $g_k(\mathbf{f})$, so that $g_k(\mathbf{f})g_{-k}(\mathbf{f})$ approximates $f_k(\mathbf{f})g_{-k}(\mathbf{f})$. Optimization is based on moment matching e.g. to assure equal mean and variance. This corresponds to minimizing the Kullback-Leibler (KL) divergence $\mathrm{KL}\big(f_k(\mathbf{f})g_{-k}(\mathbf{f})||g_k(\mathbf{f})g_{-k}(\mathbf{f})\big)$[9]. The crucial point is that site distributions $f_k(\mathbf{f})$ depend on only one component of $\mathbf{f}$. The 1-dimensional integral to calculate the required normalization factor for $f_k(\mathbf{f})g_{-k}(\mathbf{f})$ can therefore be evaluated efficiently using numerical approximations.

Algorithm 2: Expectation Propagation

Input: Observations $\mathbf{y}$ and $\mathbf{X}$ (used inside likelihood factors)
Output: Approximation $g(\mathbf{f})$ to the posterior $f(\mathbf{f})$
Initialize site approximations $g_k(\mathbf{f})$ arbitrary;
repeat
  for $k \in \{0..K\}$ **do**
    $g_{-k}(\mathbf{f}) \leftarrow g(\mathbf{f})/g_k(\mathbf{f})$;
    $g_k^{\text{new}} \leftarrow \text{argmin}_{g_k} KL\big(f_k(\mathbf{f})g_{-k}(\mathbf{f})||g_k(\mathbf{f})g_{-k}(\mathbf{f})\big)$;
    $g(\mathbf{f}) \leftarrow g_k^{\text{new}}(\mathbf{f})g_{-k}(\mathbf{f})$;
until *all* $g_k(\mathbf{f})$ *converge*;

Unfortunately, there is no general guarantee of convergence. Furthermore, even if local divergence between site distributions is minimized, this does not necessarily lead to global approximation of $g(\mathbf{f})$ and $f(\mathbf{f})$. Though, the algorithm has proven successful in many applications such as binary classification with Gaussian Processes.

Variational Inference*

Similar to the MAP estimate, **Variational Inference**[10][11] approximates the posterior by turning the inference problem into an optimization problem.

All unknown quantities of interest are treated as latent random variables[12]. A family of distributions $\mathcal{Q}$ is specified over these latent variables Z. In a simple case, $\mathcal{Q}$ might be a factorized Gaussian distribution. This is a type of the *mean-field variational family*.

[9] The KL divergence is a measure of how different two probability distributions are. It is based on relative Shannon entropy and is asymmetric, i.e. $\mathrm{KL}\big(P(X)||Q(X)\big) \neq \mathrm{KL}\big(Q(X)||P(X)\big)$.
[10] Peterson and Anderson 1987.
[11] Hinton and Camp 1993.
[12] Cf. gender assignments and means/variances per gender in section 2.1.3

It assumes that every latent variable Z_i is independently distributed - in this case each according to a Gaussian with mean μ_i and variance σ_i^2. Any configuration of μ's and σ^2's corresponds then to a specific member $Q(Z)$ of this family.

The goal of Variational Inference is to find a member $Q^*(Z)$ of family $\mathcal{Q}$ which is as similar as possible to the true posterior $P(Z|\mathbf{y}, \mathbf{X})$. Similarity is achieved by minimizing the KL divergence between the variational[13] distribution $Q(Z)$ and the true posterior $P(Z|\mathbf{y}, \mathbf{X})$.

$$Q^*(Z) = \text{argmin}_{Q(Z)\in\mathcal{Q}} \text{KL}\big(\, Q(Z) \,||\, P(Z|\mathbf{y}, \mathbf{X}\,)\big)$$

where

$$\begin{aligned}\text{KL}\big(\, Q(Z) \,||\, P(Z|\mathbf{y}, \mathbf{X})\,\big) &= \sum_{z\in Z} Q(z) \log \frac{Q(z)}{P(z|\mathbf{y}, \mathbf{X})} \\ &= \mathbb{E}_{Q(Z)}\left[\log \frac{Q(Z)}{P(Z|\mathbf{y}, \mathbf{X})}\right] \end{aligned} \tag{2.10}$$

The KL divergence still contains the log marginal likelihood (cf. $\log P(\mathbf{y}, \mathbf{X})$ in eq. (2.11)) and is therefore often intractable. However, because the marginal likelihood is constant w.r.t. the variational distribution, it can be ignored for optimization.

$$\begin{aligned}&= \mathbb{E}_{Q(Z)}[\log Q(Z)] - \mathbb{E}_{Q(Z)}[\log P(Z|\mathbf{y}, \mathbf{X})] \\ &= \mathbb{E}_{Q(Z)}[\log Q(Z)] - \mathbb{E}_{Q(Z)}\left[\log \frac{P(Z, \mathbf{y}, \mathbf{X})}{P(\mathbf{y}, \mathbf{X})}\right] \\ &= \mathbb{E}_{Q(Z)}[\log Q(Z)] - \mathbb{E}_{Q(Z)}[\log P(Z, \mathbf{y}, \mathbf{X}) - \log P(\mathbf{y}, \mathbf{X})] \\ &= \mathbb{E}_{Q(Z)}[\log Q(Z)] - \mathbb{E}_{Q(Z)}[\log P(Z, \mathbf{y}, \mathbf{X})] + \cancel{\log P(\mathbf{y}, \mathbf{X})} \qquad (2.11) \\ &\propto \mathbb{E}_{Q(Z)}[\log Q(Z)] - \mathbb{E}_{Q(Z)}[\log P(Z, \mathbf{y}, \mathbf{X})] \qquad (2.12)\end{aligned}$$

The r.h.s. of eq. (2.12) is called the negative **Evidence Lower Bound** (ELBO). Maximizing the ELBO is equivalent to minimizing the KL divergence.

$$\text{ELBO}(Q) \doteq \mathbb{E}_{Q(Z)}[\log P(Z, \mathbf{y}, \mathbf{X})] - \mathbb{E}_{Q(Z)}[\log Q(Z)] \tag{2.13}$$

The remaining challenge in VI is to find an appropriate family $\mathcal{Q}$ for the variational distribution. It should be flexible enough to approximate the true posterior, but at the same time in a form where the marginal likelihood stays tractable.

Besides Bayesian Inference, the ELBO is also a useful criterion for model selection[14]. The name ELBO stems from the fact that it is a lower bound of the (log) evidence.

[13] The name stems from the use of or from the similarity to the mathematical field of *Calculus of Variations*

[14] Jordan et al. 1999.

Using equations (2.11) and (2.13) gives:

$$KL(\ Q(Z)\ ||\ P(Z|\mathbf{y},\mathbf{X}\ )) = -\text{ELBO} + \log P(\mathbf{y},\mathbf{X}) \tag{2.14}$$

$$KL(\ Q(Z)\ ||\ P(Z|\mathbf{y},\mathbf{X}\ )) + \text{ELBO} = \log P(\mathbf{y},\mathbf{X}) \tag{2.15}$$

From $KL(\cdot) >= 0$, it follows that $\text{ELBO}(Q) <= \log P(\mathbf{y},\mathbf{X})$, i.e. the ELBO is a lower bound of the evidence. There are intuitive arguments to use the evidence (in this sense more commonly called the marginal likelihood) as a model selection criterion (cf. section 2.6.3). Using only a lower bound of the evidence is not justified in theory, but has been explored successfully in practice[15].

It can be shown that Variational Inference is a generalization of the EM algorithm. Therefore, it has some of the same important properties. Every step of optimization will monotonically increase the ELBO and therefore decrease the KL-divergence of the approximate and true posterior. This is an important distinction to EP.

Markov-Chain Monte-Carlo*

Markov-Chain Monte-Carlo is a method to sample from a probability distribution without using its *normalized* density. In the posterior distribution, the often intractable evidence term is only a normalization factor. MCMC can therefore be used to generate samples from the posterior without the computation of $P(\mathbf{y},\mathbf{X})$. These samples can subsequently be used as a representation of the true posterior.

Variational inference and MCMC are methods to solve the same problem - intractable probability distributions. The main difference is that MCMC uses a sampling approach, while VI solves an optimization problem. It is proven that MCMC produces asymptotically exact samples of the target distribution. Variational inference does not have such guarantees but is computational more efficient. The relative accuracy of VI and MCMC is not known yet.

[15] Ueda and Ghahramani 2002.

2.2 Gaussian Probability Distribution

2.2.1 Multivariate Gaussian Distribution

An N-dimensional multivariate Gaussian distribution is a generalization of the 1-dimensional Gaussian distribution. Instead of a scalar mean and variance parameter, it is characterized by a mean *vector* $\boldsymbol{\mu} \in \mathbb{R}^N$ and a covariance *matrix* $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N}$. Because the corresponding random variable is N-dimensional, it is also called a **random vector**. $\boldsymbol{\Sigma}$ describes pairwise covariances of the components of such a random vector. A valid covariance function is by definition symmetric and positive semi-definite.

The **probability density function** (PDF - $f_X(x)$) of a random variable $X \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given as:

$$f_X(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N |\boldsymbol{\Sigma}|}} \exp\left(\frac{(\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2}\right) \tag{2.16}$$

Figure 2.2 shows the PDF for a 2D Gaussian distribution. In the case of a single dimension, this reduces to the PDF of the 1D random variable $X \sim \mathcal{N}(\mu, \sigma^2)$:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.17}$$

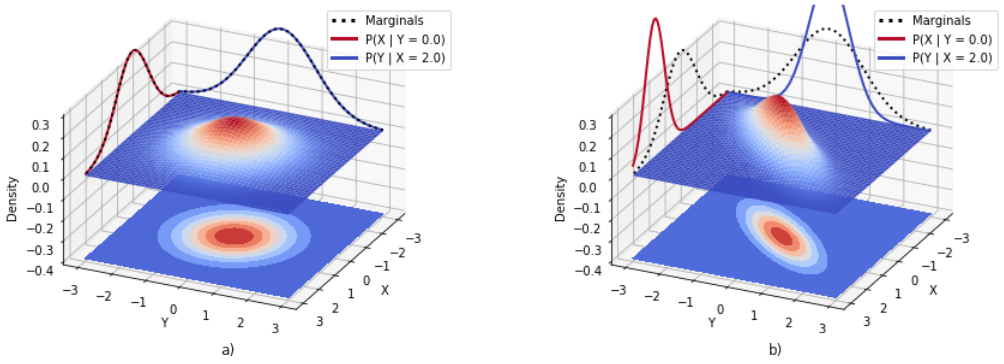

FIGURE 2.2: PDF for a 2D multivariate Gaussian distribution. At the bottom of each plot, a 2D contour-plot emphasizes the shape of the density. The marginal and one conditional distribution w.r.t to both components are indicated on the walls. The mean is 0 and the variance is 1 for X and Y in both plots. a) The covariance between X and Y is 0. This is also called a factorized Gaussian distribution. Note that for a factorized Gaussian the conditional and the marginal are equal. b) X and Y have a covariance of 0.95. This implies that high values of X correlate with high values of Y and vice-versa.

Marginal Distribution

The **marginalization property** of a multivariate Gaussian distribution states that any subset of components is again jointly Gaussian distributed. Furthermore, the corresponding mean vector and covariance matrix can be extracted directly from the rows and columns of the original parameters.

Imagine a random vector where the components are arbitrarily partitioned into two sets. For example, the 3-dimensional random vector $[X, Y, Z]^T$ can be partitioned as follows:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} A \\ B \end{bmatrix} \text{, for } A = \begin{bmatrix} X \\ Z \end{bmatrix} \text{ and } B = \begin{bmatrix} Y \end{bmatrix} \tag{2.18}$$

Now imagine that the corresponding $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are partitioned similarly:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{AA} & \boldsymbol{\Sigma}_{AB} \\ \boldsymbol{\Sigma}_{BA} & \boldsymbol{\Sigma}_{BB} \end{bmatrix} \tag{2.19}$$

where:

$$\boldsymbol{\mu}_A = \begin{bmatrix} \mu_x \\ \mu_z \end{bmatrix}, \boldsymbol{\mu}_B = \begin{bmatrix} \mu_y \end{bmatrix}, \boldsymbol{\Sigma}_{AA} = \begin{bmatrix} \sigma_{XX} & \sigma_{XZ} \\ \sigma_{ZX} & \sigma_{ZZ} \end{bmatrix}, \boldsymbol{\Sigma}_{AB} = \begin{bmatrix} \sigma_{XY} \\ \sigma_{ZY} \end{bmatrix}, ... \tag{2.20}$$

According tho the marginalization property, the random vector A is then Gaussian distributed with mean $\boldsymbol{\mu}_A$ and $\boldsymbol{\Sigma}_{AA}$:

$$A = \begin{bmatrix} X \\ Z \end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu}_A, \boldsymbol{\Sigma}_{AA}) \tag{2.21}$$

Conditional Distribution

Given a multivariate Gaussian distribution, the conditional of one set of components w.r.t. the other set of components is again Gaussian distributed. Following the example from the previous section:

$$f_{A|B}(\boldsymbol{a}|\boldsymbol{b}) \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) \tag{2.22}$$

where

$$\hat{\boldsymbol{\mu}} = \boldsymbol{\mu}_A + \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}(\boldsymbol{b} - \boldsymbol{\mu}_B) \tag{2.23}$$

$$\hat{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA} \tag{2.24}$$

Product of Two Gaussian PDFs

The product of two Gaussian PDFs is an unnormalized Gaussian PDF [16]. That is for $A \sim \mathcal{N}(\mu_A, \sigma_A^2)$, $B \sim \mathcal{N}(\mu_B, \sigma_B^2)$ and $C \sim \mathcal{N}(\mu_C, \sigma_C^2)$:

$$f_A(x) * f_B(x) \propto f_C(x) \tag{2.25}$$

where the corresponding μ_C and σ_C^2 are:

$$\mu_C = \frac{\mu_A \sigma_B^2 + \mu_B \sigma_A^2}{\sigma_A^2 \sigma_B^2}, \; \sigma_C^2 = \sqrt{\frac{\sigma_B^2 + \sigma_A^2}{\sigma_A^2 \sigma_B^2}} \tag{2.26}$$

This does not mean that $A * B$ is Gaussian distributed. The term $f_A(x) * f_B(x)$ represents the joint probability density of two *uncorrelated* random variables. I.e. A and B both taking on the value x simultaneously.

[16] Full proof see: Bromiley 2018

2.3 Gaussian Process - Formally

A multivariate Gaussian is characterized by a mean *vector* $\boldsymbol{\mu}$ and covariance *matrix* $\boldsymbol{\Sigma}$. A Gaussian Process in contrast, is characterized by a mean *function* $\mathrm{m}(\mathbf{x})$ and a covariance *function* $\mathrm{k}(\mathbf{x}, \mathbf{x}')$.

Both are functions of sample input(s) $\mathbf{x}$[17]. A function - in contrast to a vector or matrix - can have an infinite support. This corresponds to a $\boldsymbol{\mu}$ in $\mathbb{R}^{\infty}$ and covariance matrix in $\mathbb{R}^{\infty \times \infty}$. For this reason, a GP is often interpreted as a multivariate Gaussian with infinite dimensions.

On a computer with finite resources, the handling of a vector with infinite dimensions is intractable. In reality however, one is often interested in the values of a finite subset of components (dimensions) $[X_1, X_2, ..., X_K]^T$ only. According to the marginalization property (cf. section 2.2.1), this *marginal* distribution is again a (K-dimensional) multivariate Gaussian.

Multiple sample inputs are usually held in a matrix $\mathbf{X}$. The N rows represent sample-inputs and the D columns the specific features[18]. $\mathbf{x}_i$ denotes a sample input (feature vector) of the i^{th} sample. For a more compact notation, let:

$$\mathrm{M}(\mathbf{A}) \doteq \begin{bmatrix} \mathrm{m}(\mathbf{a}_1) \\ \mathrm{m}(\mathbf{a}_2) \\ \vdots \\ \mathrm{m}(\mathbf{a}_N) \end{bmatrix}, \tag{2.27}$$

$$\mathrm{K}(\mathbf{A}, \mathbf{B}) \doteq \begin{bmatrix} \mathrm{k}(\mathbf{a}_1, \mathbf{b}_1) & \mathrm{k}(\mathbf{a}_1, \mathbf{b}_2) & \cdots & \mathrm{k}(\mathbf{a}_1, \mathbf{b}_M) \\ \mathrm{k}(\mathbf{a}_2, \mathbf{b}_1) & \mathrm{k}(\mathbf{a}_2, \mathbf{b}_2) & \cdots & \mathrm{k}(\mathbf{a}_2, \mathbf{b}_M) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{k}(\mathbf{a}_N, \mathbf{b}_1) & \mathrm{k}(\mathbf{a}_N, \mathbf{b}_2) & \cdots & \mathrm{k}(\mathbf{a}_N, \mathbf{b}_M) \end{bmatrix}, \tag{2.28}$$

for any $\mathbf{A} \in \mathbb{R}^{NxD}$ and $\mathbf{B} \in \mathbb{R}^{MxD}$.

In statistical inference, the overall dataset $\mathcal{D}$ is usually partitioned in two subsets. The **training set** consists of N samples with observed inputs $\mathbf{X}$ and outputs $\mathbf{y}$. For the M datapoints in the **test set**, only the inputs $\mathbf{X}_*$ are available. The outputs $\mathbf{y}_*$ are then predicted using an inference method. Inference with Gaussian Processes heavily depends on the pairwise covariances between inputs in the training and test set.

[17] The vector notation emphasizes that the input can have any finite number of dimensions (features)
[18] Cf. Design-Matrix

The following short-notation reduces the notational overhead significantly.

$$\begin{aligned}
\boldsymbol{K} &\doteq \mathrm{K}(\mathbf{X}, \mathbf{X}) \text{ is the covariance matrix of the training inputs} \\
\boldsymbol{K}_* &\doteq \mathrm{K}(\mathbf{X}, \mathbf{X}_*) \text{ is the covariance matrix between training and test inputs} \\
\boldsymbol{K}_{**} &\doteq \mathrm{K}(\mathbf{X}_*, \mathbf{X}_*) \text{ is the covariance matrix of the test inputs}
\end{aligned}$$

Starting with the infinite random vector, there are two common (finite) marginal distributions. $P(Y)$ is the distribution of the outputs in the training set. $P(Y, Y_*)$ is the joint distribution over the outputs in training and test set. The respective (finite) mean vectors and covariance matrices are then given by:

$$\begin{aligned}
P(Y) &\sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t), \text{where: } \boldsymbol{\mu}_t = \mathrm{M}(\mathbf{X}), \boldsymbol{\Sigma}_t = \boldsymbol{K} \\
P(Y, Y_*) &\sim \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j), \text{where: } \boldsymbol{\mu}_j = \begin{bmatrix} \mathrm{M}(\mathbf{X}) \\ \mathrm{M}(\mathbf{X}_*) \end{bmatrix}, \boldsymbol{\Sigma}_j = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix}
\end{aligned}$$

In the context of Gaussian Processes, the covariance function $\mathrm{k}(\cdot, \cdot)$ is often called a **kernel**. A kernel has to be symmetric and positive semi-definite[19] in order to produce a valid covariance matrix. Common kernels are the *dot-product* and the *squared exponential* (`SE`) kernel. The `SE` kernel is also called *Radial Basis Function* (RBF) or just *Gaussian* kernel. It is defined as:

$$\mathrm{k}_{\mathrm{SE}}(x, x') = \exp\left(-\frac{(x - x')^2}{2l^2} \right)$$

or:

$$\mathrm{k}_{\mathrm{SE}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^T(\mathbf{x} - \mathbf{x}')}{2l^2} \right) \tag{2.29}$$

in case of vector inputs. The covariance decays exponentially in the squared euclidean distance between inputs.

The *length-scale* parameter l is a **kernel hyperparameter**. It controls the decay rate of the covariance as illustrated in figure 1.4. Kernels and their hyperparameters will be discussed in section 2.5.

[19] A positive-definite kernel is a generalization of a positive-definite function or positive-definite matrix.

2.4 Prediction in Gaussian Process Regression

In literature,[20] the predictive distribution for **Gaussian Process Regression** (GPR) is often given as:

$$P(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where:

$$\begin{aligned}\boldsymbol{\mu} &= \mathrm{K}(\mathbf{X}_*, \mathbf{X})[\mathrm{K}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \\ \boldsymbol{\Sigma} &= \mathrm{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathrm{K}(\mathbf{X}_*, \mathbf{X})[\mathrm{K}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}]^{-1}\mathrm{K}(\mathbf{X}, \mathbf{X}_*)\end{aligned}$$

or using our shorthand notation from page 26:

$$\begin{aligned}\boldsymbol{\mu} &= \boldsymbol{K}_*^T[\boldsymbol{K} + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \\ \boldsymbol{\Sigma} &= \boldsymbol{K}_{**} - \boldsymbol{K}_*^T[\boldsymbol{K} + \sigma^2\mathbf{I}]^{-1}\boldsymbol{K}_*\end{aligned} \tag{2.30}$$

The following sections provide a step by step derivation and intuition for (2.30). Care has been taken to state all underlying assumptions, used transformations and identitys explicitly. To our knowledge, such an in-detail derivation has not been published yet. Existing introductions are on a rather high-level of abstraction, rely heavily on Gaussian and matrix identitys and assume knowledge of concepts like *completing-the-square* or the *Woodbury, Sherman, Morrison formula*. While this is common and useful for compact and general proofs, the underlying details and the computational complexity is somehow obfuscated.

The main goal is to show where and what kind of computational bottlenecks Gaussian Processes often face. In particular, when they are used in a Regression or Classification task. On this basis, widely used claims and statements regarding GPs will be discussed:

- The computational complexity of the training of a GP is $O(N^3)$
- The computational complexity of prediction using a GP is $O(N^2M)$
- Non-Gaussian likelihoods render the predictive distribution intractable

[20] Rasmussen and Williams 2005, eq. 2.22, p. 16.

2.4.1 The Model

Gaussian Process Regression[21] assumes that the observed outputs $\mathbf{y} = [y_1, y_2, \cdots, y_N]^T$ are generated by a hidden (latent) function of the inputs $f(\mathbf{X})$ and some additive i.i.d. noise. Figure 2.3 illustrates the individual components.

$$\begin{aligned} \mathbf{y} &= f(\mathbf{X}) + \boldsymbol{\epsilon} \\ \mathbf{y} &= \mathbf{f} + \boldsymbol{\epsilon} \end{aligned} \tag{2.31}$$

N denotes the number of input-output pairs ($\mathbf{x}$, y). $\mathbf{f} \in \mathbb{R}^N$ denotes the vector of latent function values $[f(\mathbf{x}_1), f(\mathbf{x}_2), \cdots, f(\mathbf{x}_N)]^T$ and $\boldsymbol{\epsilon} \in \mathbb{R}^N$ the noise components. GPR assumes further that the latent function $f(\cdot)$ is a member of the family represented by a Gaussian Process. For GPR in particular, the distribution of the noise is assumed to have a *factorized* multivariate Gaussian distribution[22]. Formally:

$$\mathbf{f} \sim GP(\mathrm{m}(\cdot), \mathrm{k}(\cdot)) = \mathcal{N}(\mathrm{M}(\mathbf{X}), \mathrm{K}(\mathbf{X}, \mathbf{X})) \tag{2.32}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \tag{2.33}$$

where $\mathbf{I} \in \mathbb{R}^{NxN}$ is the identity matrix and σ^2 the variance of the noise.

The primary role of the latent variables, in this case, is to allow a convenient formulation of the model (cf. section 2.1.3). In reality, the true values of $\mathbf{f}$ can never be observed - only their noisy observations $\mathbf{y}$. The noise is either due to small errors in measurements or because not every relevant feature[23] is actually measured and contained in $\mathbf{X}$.

[21] As well as Gaussian Process Classification and further applications using Gaussian Processes.

[22] instead of a vector with N independent 1-dimensional Gaussians a N-dimensional Gaussian with zero covariance between the dimensions is used (i.e. all non-diagonal elements of the covariance matrix are 0). The term *factorized* comes from the fact that for such a covariance matrix the joint probability $P(\epsilon_1 = 5, \epsilon_1 = 2, \epsilon_1 = -3 | \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is equal to $P(\epsilon_1 = 5 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{1,1}) P(\epsilon_1 = 2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{2,2}) P(\epsilon_1 = -2 | \boldsymbol{\mu}_3, \boldsymbol{\Sigma}_{3,3})$

[23] Which might influence the outcome of $\mathbf{y}$

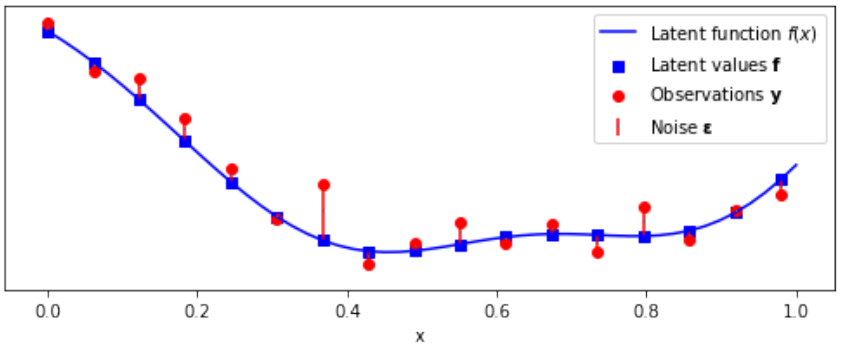

FIGURE 2.3: GP model for regression with 1 dimensional inputs. Observations are modeled by a latent function and additive i.i.d. noise.

For prediction, i.e. for inputs $\mathbf{X}_*$ with no associated outputs available, the goal is to predict the values of the (noise-free) latent variables $\mathbf{f}_*$. It would not be beneficial to add artificial noise to the prediction. This uncertainty will however be incorporated into the confidence of the prediction.

Error/Noise vs. Likelihood

The term *error* or *noise* is somewhat misleading. While it is a intuitive explanation in the regression case (cf. figure 2.3), it has a more general interpretation. In general, latent values are often mapped to observations with a *link* function. Then, a *likelihood* function evaluates the probability to observe $\mathbf{y}$ given the transformed latent values $\hat{\mathbf{f}}$. This concept is the core of *Generalized Linear Models*.

In the regression case, latent values and observations are mapped using the identity *link* (i.e. $\hat{\mathbf{f}} \doteq \mathbf{f}$). It is therefore often suppressed. The likelihood is a factorized Gaussian.

For classification, the *link* and *likelihood* functions will be different. The simple interpretation as a measurement-error or noise component ϵ will not hold anymore. To avoid this discontinuity, the term *likelihood* should be preferred over *noise* or *error*.

2.4.2 Composition and Use of the Predictive Distribution

The assumption of GPR is that for any finite set of inputs $\mathbf{X}$ the corresponding latent values $\mathbf{f} = f(\mathbf{X})$ are jointly Gaussian distributed with a mean vector of $M(\mathbf{X})$ and a covariance matrix given by $\mathrm{K}(\mathbf{X}, \mathbf{X})$. The relation between $\mathbf{f}$ and $\mathbf{f}_*$ is therefore as following.

Let $\mathbf{X} \in \mathbb{R}^{NxD}$ and $\mathbf{X}_* \in \mathbb{R}^{MxD}$ be the inputs for N training and M test points with D features per input. Further let $\mathbf{f} \in \mathbb{R}^N$ and $\mathbf{f}_* \in \mathbb{R}^M$ be the corresponding latent values. Assuming that $\mathbf{f}$ and $\mathbf{f}_*$ are generated by the same latent function $f(\cdot)$, they can be combined in one random vector with a joint Gaussian distribution[24].

$$\begin{bmatrix} f^1 \\ \vdots \\ f^N \\ f_*^1 \\ \vdots \\ f_*^M \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}^1) \\ \vdots \\ m(\mathbf{x}^N) \\ m(\mathbf{x}_*^1) \\ \vdots \\ m(\mathbf{x}_*^M) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}^1,\mathbf{x}^1) & \cdots & k(\mathbf{x}^1,\mathbf{x}^N) & k(\mathbf{x}^1,\mathbf{x}_*^1) & \cdots & k(\mathbf{x}^1,\mathbf{x}_*^M) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ k(\mathbf{x}^N,\mathbf{x}^1) & \cdots & k(\mathbf{x}^N,\mathbf{x}^N) & k(\mathbf{x}^N,\mathbf{x}_*^1) & \cdots & k(\mathbf{x}^N,\mathbf{x}_*^M) \\ k(\mathbf{x}_*^1,\mathbf{x}^1) & \cdots & k(\mathbf{x}_*^1,\mathbf{x}^N) & k(\mathbf{x}_*^1,\mathbf{x}_*^1) & \cdots & k(\mathbf{x}_*^1,\mathbf{x}_*^M) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ k(\mathbf{x}_*^M,\mathbf{x}^1) & \cdots & k(\mathbf{x}_*^M,\mathbf{x}^N) & k(\mathbf{x}_*^M,\mathbf{x}_*^1) & \cdots & k(\mathbf{x}_*^M,\mathbf{x}_*^M) \end{bmatrix} \right) \tag{2.34}$$

or more compact:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} M(\mathbf{X}) \\ M(\mathbf{X}_*) \end{bmatrix}, \begin{bmatrix} \mathrm{K}(\mathbf{X},\mathbf{X}) & \mathrm{K}(\mathbf{X},\mathbf{X}_*) \\ \mathrm{K}(\mathbf{X}_*,\mathbf{X}) & \mathrm{K}(\mathbf{X}_*,\mathbf{X}_*) \end{bmatrix} \right) \tag{2.35}$$

This **joint probability distribution** is the starting point for the predictive distribution in (2.30). While neither $\mathbf{f}$ nor $\mathbf{f}_*$ can ever be observed directly, there are at least noisy observations $\mathbf{y}$ corresponding to $\mathbf{f}$. This implies that prediction actually consists of two steps:

1. Given the observations $\mathbf{y}$, find probable values for $\mathbf{f}$.
2. Given these values of $\mathbf{f}$, find probable values for $\mathbf{f}_*$.

For the first step, there are two possible solutions. Finding only the most likely value $\mathbf{f}$ is called the maximum-a-posteriori (MAP) estimate of $\mathbf{f}$. For a fully Bayesian treating however, a probability distribution over $\mathbf{f}$ should be used to retain the uncertainty about its value. Every possible value of $\mathbf{f}$ should be considered and weighted according to its posterior probability. This corresponds to integrating over all possible realizations of the random vector $\mathbf{f}$. Because the N components of $\mathbf{f}$ are in $\mathbb{R}$, this will be an N-dimensional infinite integral.

[24] Note that superscripts instead of subscripts are used to denote rows of input matrices $\mathbf{X}$ and $\mathbf{X}_*$ exceptionally

In terms of probabilitys, this can be expressed as:

$$P(\mathbf{f}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y}) = \int_{\mathbf{f}} P(\mathbf{f}_*|\mathbf{f},\mathbf{X}_*,\mathbf{X}) * P(\mathbf{f}|\mathbf{X},\mathbf{y}) \tag{2.36}$$

The predictive distribution in equation (2.36) is **the fundamental basis of a Gaussian Process**. Its decomposition and detailed explanation is the main topic of this section and a core contribution of this work.

Subsequently, the first factor inside the integral is denoted as **the conditional** and the second as **the posterior**. Remember from section 2.1.4 that the posterior itself consists of three factors. Figure 2.4 shows a full decomposition with *labels* of individual components for subsequent references.

$$\underbrace{P(\mathbf{f}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y})}_{\text{Predictive}} = \int_{\mathbf{f}} \underbrace{P(\mathbf{f}_*|\mathbf{f},\mathbf{X}_*,\mathbf{X})}_{\text{Conditional}} * \underbrace{\frac{\overbrace{P(\mathbf{y}|\mathbf{f},\mathbf{X})}^{\text{Likelihood}} * \overbrace{P(\mathbf{f}|\mathbf{X})}^{\text{Prior}}}{\underbrace{\int_{\mathbf{f}} P(\mathbf{y}|\mathbf{f},\mathbf{X}) * P(\mathbf{f}|\mathbf{X})}_{\text{Marginal Likelihood}}}}_{\text{Posterior}}$$

FIGURE 2.4: Decomposition of the GP predictive distribution.

The predictive distribution does not return a value for $\mathbf{f}_*$ yet. It is only a density function for $\mathbf{f}_*$ *parametrized by* $\mathbf{X}_*$, $\mathbf{X}$ and $\mathbf{y}$. I.e. given the training set $(\mathbf{X},\mathbf{y})$, it returns the probability density for latent values $\mathbf{f}_*$ at test inputs $\mathbf{X}_*$. In reality, one is often not interested in the probability density of a specific value of $\mathbf{f}_*$. More useful for prediction is often the *expected* value of $\mathbf{f}_*$[25]. An expected value is calculated by another integral over *the predictive*:

$$\begin{aligned}\mathbb{E}_{P(\mathbf{f}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y})}[\mathbf{f}_*] &= \int_{\mathbf{f}_*} \mathbf{f}_* * P(\mathbf{f}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y}) \\ &= \int_{\mathbf{f}_*} \mathbf{f}_* * \int_{\mathbf{f}} P(\mathbf{f}_*|\mathbf{f},\mathbf{X}_*,\mathbf{X}) * \frac{P(\mathbf{y}|\mathbf{f},\mathbf{X}) * P(\mathbf{f}|\mathbf{X})}{\int_{\mathbf{f}} P(\mathbf{y}|\mathbf{f},\mathbf{X}) * P(\mathbf{f}|\mathbf{X})}\end{aligned} \tag{2.37}$$

Three multidimensional infinite integrals are most likely not tractable. However, equation (2.30) states that for GPR, the predictive distribution of $\mathbf{f}_*$ is Gaussian distributed with some $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Given that the expected value for a Gaussian distributed random variable is equal to its mean $\boldsymbol{\mu}$, it is not even necessary to calculate the outermost integral numerically.

[25] and some confidence intervals

2.4.3 Outline of the Proof

The next sections will show in detail why and how the integrals in *the predictive* in fact reduce to an overall Gaussian distribution of $\mathbf{f}_*$.

In general, an expression like the r.h.s of equation (2.38) is called an *unnormalized* Gaussian PDF for random variable A with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. c is any constant factor w.r.t. $\mathbf{a}$.

$$f_A(\mathbf{a}) \propto c * \exp\left(-\frac{(\mathbf{a}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{a}-\boldsymbol{\mu})}{2}\right) \tag{2.38}$$

Any such unnormalized Gaussian PDF can then be turned into a valid PDF by replacing c with $\sqrt{(2\pi)^N|\boldsymbol{\Sigma}|}^{-1}$:

$$f_A(\mathbf{a}) = \frac{1}{\sqrt{(2\pi)^N|\boldsymbol{\Sigma}|}} * \exp\left(-\frac{(\mathbf{a}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{a}-\boldsymbol{\mu})}{2}\right) \tag{2.39}$$

$\mathbf{f}_*$ is therefore Gaussian distributed iff *the predictive* density can be rearranged to an unnormalized Gaussian PDF.

Section 2.4.4 and 2.4.5 will show that *the conditional* and *the posterior* are both unnormalized Gaussian PDFs. Eventually section 2.4.6 shows how the product of these unnormalized Gaussian PDFs can be transformed into the overall predictive Gaussian distribution from equation 2.30.

2.4.4 The Conditional

Using the definition of conditional probability, *the conditional* in figure 2.4 is given by the joint divided by the marginal distribution:

$$\begin{aligned} P(A|B) * P(B) &= P(A,B) \\ P(A|B) &= \frac{P(A,B)}{P(B)}, \quad \text{for } P(B) \neq 0 \end{aligned} \tag{2.40}$$

applied to *the conditional*

$$\begin{aligned} P(\mathbf{f}_*|\mathbf{f},\mathbf{X}_*,\mathbf{X}) &= \frac{P(\mathbf{f}_*,\mathbf{f}|\mathbf{X}_*,\mathbf{X})}{P(\mathbf{f}|\mathbf{X}_*,\mathbf{X})} \\ &= \frac{P(\mathbf{f}_*,\mathbf{f}|\mathbf{X}_*,\mathbf{X})}{P(\mathbf{f}|\mathbf{X})} \end{aligned} \tag{2.41}$$

The equality of $P(\mathbf{f}|\mathbf{X}_*,\mathbf{X}) = P(\mathbf{f}|\mathbf{X})$ in the last step follows from the fact that $\mathbf{f}$ is assumed to be conditionally independent of $\mathbf{X}_*$ given $\mathbf{X}$.

The joint in the numerator is given by (2.35). According to the marginalization property (section 2.2.1), the marginal in the denumerator can be obtained by taking the corresponding sub-vector and sub-matrix from the joint.

In terms of explicit probability densitys, this means:

$$= \frac{\frac{1}{\sqrt{(2\pi)^{N+M}|\mathbf{\Sigma}_j|}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} - \begin{bmatrix} M(\mathbf{X}) \\ M(\mathbf{X}_*) \end{bmatrix} \right)^T \mathbf{\Sigma}_j^{-1} \left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} - \begin{bmatrix} M(\mathbf{X}) \\ M(\mathbf{X}_*) \end{bmatrix} \right) \right)}{\frac{1}{\sqrt{(2\pi)^{N}|\mathbf{\Sigma}_m|}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} \mathbf{f} \end{bmatrix} - \begin{bmatrix} M(\mathbf{X}) \end{bmatrix} \right)^T \mathbf{\Sigma}_m^{-1} \left(\begin{bmatrix} \mathbf{f} \end{bmatrix} - \begin{bmatrix} M(\mathbf{X}) \end{bmatrix} \right) \right)} \tag{2.42}$$

where:

$$\mathbf{\Sigma}_j \text{ (subscript } \mathbf{j} \text{ for } \mathbf{j}\text{oint)} = \begin{bmatrix} \mathrm{K}(\mathbf{X}, \mathbf{X}) & \mathrm{K}(\mathbf{X}, \mathbf{X}_*) \\ \mathrm{K}(\mathbf{X}_*, \mathbf{X}) & \mathrm{K}(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix} \tag{2.43}$$

$$\mathbf{\Sigma}_m \text{ (subscript } \mathbf{m} \text{ for } \mathbf{m}\text{arginal)} = \begin{bmatrix} \mathrm{K}(\mathbf{X}, \mathbf{X}) \end{bmatrix} \tag{2.44}$$

For simplicity, it is assumed that the mean function is defined as $M(\cdot) = \mathbf{0}$. This assumption can always be satisfied by mean-centering the outputs in a preprocessing step. The mean can therefore be omitted subsequently.

Appendix A.1.1 shows in detail how the full conditional in (2.42) can be rearranged to the form of an unnormalized Gaussian PDF (cf. eq. (2.38)). Constants w.r.t. $\mathbf{f}_*$ are continuously collected in the variable c_c (with subscript $_c$ for **c**onditional). The result of this transformation reveals an expression for $P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}_*, \mathbf{X})$ which is proportional to a Gaussian probability density:

$$P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}_*, \mathbf{X}) = c_c * \exp\left(-\frac{(\mathbf{f}_* - \boldsymbol{\mu}_c)^T \mathbf{\Sigma}_c^{-1} (\mathbf{f}_* - \boldsymbol{\mu}_c)}{2} \right)$$

where:

$$\boldsymbol{\mu}_c = \boldsymbol{K}_* \boldsymbol{K}^{-1} \mathbf{f}$$
$$\mathbf{\Sigma}_c = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T \boldsymbol{K}^{-1} \boldsymbol{K}_*$$

2.4.5 The Posterior

This section will show in detail why *the posterior* density in GPR is proportional to a Gaussian probability density. In contrast to *the conditional* in the previous section, the posterior represents the density for $\mathbf{f}$. It does not depend or use $\mathbf{X}_*$ or $\mathbf{f}_*$ in any way.

To prove *Gaussianity*, an expression as in (2.38) must be derived. Constants w.r.t. $\mathbf{f}$ are again collected in a variable c_p (with subscript $_p$ for posterior respectively).

The posterior is derived using *Bayes Rule*:

$$P(A|B)P(B) = P(B|A)P(A)$$
$$P(A|B) = \frac{P(B|A)}{P(B)}, \quad \text{for } P(B) \neq 0$$

applied to the posterior:

$$P(\mathbf{f}|\mathbf{y},\mathbf{X}) = \frac{P(\mathbf{y}|\mathbf{f},\mathbf{X})P(\mathbf{f}|\mathbf{X})}{P(\mathbf{y}|\mathbf{X})} \tag{2.45}$$

$P(\mathbf{y}|\mathbf{f},\mathbf{X})$ is called the **likelihood** and models the relation of latent values and observations. I.e. how likely it is to observe $\mathbf{y}$ when the latent values are $\mathbf{f}$. For GPR the likelihood is assumed to be i.i.d Gaussian distributed with a mean of $\boldsymbol{\mu} = \mathbf{f}$.

$P(\mathbf{f}|\mathbf{X})$ is called the GP **prior**. It reflects the prior assumption that the latent values are generated by a GP with (often) zero mean and a covariance matrix according to a kernel function[26].

$P(\mathbf{y}|\mathbf{X})$ is called the **evidence** or **marginal likelihood**. It is further analyzed in section 2.6. It is constant w.r.t $\mathbf{f}$ and its inverse is therefore immediately collected in c_p.

In terms of explicit probability densitys, the full posterior is:

$$P(\mathbf{f}|\mathbf{y},\mathbf{X}) = c_p * \prod_{i=1}^{N}\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i-f_i)^2}{2\sigma^2}\right)\right] * \frac{1}{\sqrt{(2\pi)^N|\boldsymbol{K}|}}\exp\left(-\frac{(\mathbf{f}-\mathbf{0})^T\boldsymbol{K}^{-1}(\mathbf{f}-\mathbf{0})}{2}\right)$$

The likelihood term (the product expression) can be reformulated using matrix/vector notation and the identity matrix $\mathbf{I}$. The constant normalization terms are collected in c_p

$$= c_p * \frac{1}{\sqrt{(2\pi)^N|\sigma^2\mathbf{I}|}}\exp\left(-\frac{(\mathbf{y}-\mathbf{f})^T(\sigma^2\mathbf{I})^{-1}(\mathbf{y}-\mathbf{f})}{2}\right) * \exp\left(-\frac{(\mathbf{f}-\mathbf{0})^T\boldsymbol{K}^{-1}(\mathbf{f}-\mathbf{0})}{2}\right)$$

[26] Informally: By a latent function which is *smooth*. Similar inputs will result in similar outputs and *similarity* is defined through a kernel.

The diagonal covariance matrix $\sigma^2\mathbf{I}$ satisfies the assumption that the noise is independent and identically distributed between observations.

Appendix A.1.2 shows how the posterior in (2.46) can be rearranged to reveal an expression which is proportional to a Gaussian probability density with:

$$\boldsymbol{\mu}_p = \boldsymbol{K}(\boldsymbol{K} + (\sigma^2\mathbf{I}))^{-1}\mathbf{y} \tag{2.46}$$

$$\boldsymbol{\Sigma}_p = ((\sigma^2\mathbf{I})^{-1} + \boldsymbol{K}^{-1})^{-1} \tag{2.47}$$

2.4.6 Putting It Together - The Predictive Distribution

Section 2.4.4 and 2.4.5 have shown that the densities of *the conditional* and *the posterior* are both proportional to a Gaussian PDF. According to (2.36), the predictive distribution for GPR consists of an infinite integral over the conditional times the posterior. This section shows how to calculate this integral analytically using the following steps:

1. Show that *the conditional* times *the posterior* is again proportional to a Gaussian PDF
2. Calculate the integral using the definition of a Gaussian PDF

For step 1, the focus is on the product of the two (unnormalized) Gaussian densities inside the integral. A product of two Gaussian PDFs is again an unnormalized Gaussian PDF (cf. section 2.2.1). This assumes however, that both PDFs are functions of the same argument, i.e. $f_A(x) * f_B(x)$ for random variables A and B and argument x. The conditional and posterior densities are functions of $\mathbf{f}_*$ and $\mathbf{f}$ respectively. This identity is therefore not directly applicable.

Fortunately, the special form in GPR allows to apply another *trick*. The mean of the conditional is a linear combination of $\mathbf{f}$ (cf. eq. (2.48)). Symmetry of the Gaussian PDF allows to interpret *the conditional* as a function of $\mathbf{f}$ with a mean of $\mathbf{f}_*$.

For the last part of the proof, the explicit and rearranged terms from section 2.4.4 and 2.4.5 are substituted for *the conditional* and *the posterior*. The following placeholders are reused:

$$\boldsymbol{\mu}_c = \boldsymbol{K}_* \boldsymbol{K}^{-1} \mathbf{f} \rightarrow \hat{\boldsymbol{\mu}}_c = \boldsymbol{K}_* \boldsymbol{K}^{-1} \tag{2.48}$$

$$\boldsymbol{\Sigma}_c = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T \boldsymbol{K}^{-1} \boldsymbol{K}_* \tag{2.49}$$

$$\boldsymbol{\mu}_p = \mathbf{y} \boldsymbol{K} (\sigma^2 \mathbf{I} + \boldsymbol{K})^{-1} \tag{2.50}$$

$$\boldsymbol{\Sigma}_p = \left((\sigma^2 \mathbf{I}) \boldsymbol{K} ((\sigma^2 \mathbf{I}) + \boldsymbol{K})^{-1} \right)^{-1} \tag{2.51}$$

$$c_{pred} \leftarrow c_c * c_p \tag{2.52}$$

As mentioned, the mean of the conditional $\boldsymbol{\mu}_c$ is dependent on $\mathbf{f}$. Therefore, $\boldsymbol{\mu}_c$ is replaced by $\hat{\boldsymbol{\mu}}_c \mathbf{f}$ to make this dependency explicit for further rearrangements. Furthermore, c_{pred} is initialized with the product of the collected constants from above.

$$P(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) = c_{pred} \int_{\mathbf{f}} P(\mathbf{f}_* | \mathbf{f}, \mathbf{X}_*, \mathbf{X}) * P(\mathbf{f} | \mathbf{X}, \mathbf{y})$$

$$= c_{pred} \int_{\mathbf{f}} \exp \left(\frac{-\left((\mathbf{f}_* - \hat{\boldsymbol{\mu}}_c \mathbf{f})^T \boldsymbol{\Sigma}_c^{-1} (\mathbf{f}_* - \hat{\boldsymbol{\mu}}_c \mathbf{f}) \right)}{2} \right) \exp \left(\frac{-\left((\mathbf{f} - \boldsymbol{\mu}_p)^T \boldsymbol{\Sigma}_p^{-1} (\mathbf{f} - \boldsymbol{\mu}_p) \right)}{2} \right)$$

Appendix A.1.3 shows the detailed arrangements to the following expression.

$$= c_{pred} \exp \left(\frac{{\mathbf{f}_*}^T \boldsymbol{\Sigma}_c^{-1} \mathbf{f}_*}{-2} \right) \exp \left(\frac{\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}}{2} \right) \int_{\mathbf{f}} \exp \left(\frac{\left(\mathbf{f} - \mathbf{A}^{-1} \mathbf{b} \right)^T \mathbf{A} \left(\mathbf{f} - \mathbf{A}^{-1} \mathbf{b} \right)}{-2} \right) \tag{2.53}$$

where $\mathbf{A}$ is independent of $\mathbf{f}$ and $\mathbf{f}_*$, and $\mathbf{b}$ is independent of $\mathbf{f}$ (details see appendix).

The remaining (multidimensional) integral can now be evaluated analytically. For a valid Gaussian PDF with a mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ the following must hold[27]:

$$\int_{\mathbf{x}} \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp \left(-\frac{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}{2} \right) = 1 \tag{2.54}$$

therefore:

$$\int_{\mathbf{x}} \exp \left(-\frac{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}{2} \right) = \sqrt{(2\pi)^k |\boldsymbol{\Sigma}|} \tag{2.55}$$

[27] From the definition of a probability density function

After replacing $\mathbf{x}$ with $\mathbf{f}$, $\boldsymbol{\mu}$ with $\mathbf{A}^{-1}\mathbf{b}$ and $\boldsymbol{\Sigma}^{-1}$ with $\mathbf{A}$, the l.h.s. is equal to the integral in (2.53). Both represent an integral over an unnormalized Gaussian PDF. Therefore, using (2.55) the predictive distributions reduces to:

$$
\begin{aligned}
&= c_{pred} \exp\left(\frac{\mathbf{f}_*{}^T \boldsymbol{\Sigma}_c^{-1} \mathbf{f}_*}{-2}\right) \exp\left(\frac{\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}}{2}\right) \int_{\mathbf{f}} \exp\left(\frac{\left(\mathbf{f} - \mathbf{A}^{-1}\mathbf{b}\right)^T \mathbf{A} \left(\mathbf{f} - \mathbf{A}^{-1}\mathbf{b}\right)}{-2}\right) \\
&= c_{pred} \exp\left(\frac{\mathbf{f}_*{}^T \boldsymbol{\Sigma}_c^{-1} \mathbf{f}_*}{-2}\right) \exp\left(\frac{\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}}{2}\right) \cancel{\sqrt{(2\pi)^k |\mathbf{A}^{-1}|}}
\end{aligned} \tag{2.56}
$$

Eventually, because $\mathbf{A}$ does not depend on $\mathbf{f}_*$, the analytical expression for the integral is collected immediately in c_{pred}. [28]

Appendix A.1.4 shows how (2.56) can be rearranged to an expression proportional to a Gaussian PDF:

$$
P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}_*, \mathbf{X}) = c_{pred} * \exp\left(-\frac{(\mathbf{f}_* - \boldsymbol{\mu}_{pred})^T \boldsymbol{\Sigma}_{pred}^{-1} (\mathbf{f}_* - \boldsymbol{\mu}_{pred})}{2}\right) \tag{2.57}
$$

where:

$$
\begin{aligned}
\boldsymbol{\mu}_{pred} &= (\boldsymbol{\Sigma}_c^{-1} - \mathbf{u}\mathbf{W}\mathbf{u}^T)^{-1} \mathbf{u}\mathbf{W}\mathbf{v}^T \\
\boldsymbol{\Sigma}_{pred} &= (\boldsymbol{\Sigma}_c^{-1} - \mathbf{u}\mathbf{W}\mathbf{u}^T)^{-1}
\end{aligned}
$$

We have shown that the PDF of the predictive distribution of GPR is proportional to a Gaussian PDF. The parameters $\boldsymbol{\mu}_{pred}$ and $\boldsymbol{\Sigma}_{pred}$ can be further simplified using the *Sherman-Morrison-Woodburry* formula. Finally substituting back in the original means and covariances gives:

$$
\boldsymbol{\mu}_{pred} = \boldsymbol{K}_*^T (\boldsymbol{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} \tag{2.58}
$$

$$
\boldsymbol{\Sigma}_{pred} = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T (\boldsymbol{K} + \sigma^2 \mathbf{I})^{-1} \boldsymbol{K}_* \tag{2.59}
$$

This is equivalent to (2.30) presented at the beginning of this chapter and often seen in literature.

$$
P(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\boldsymbol{K}_* (\sigma^2 \mathbf{I} + \boldsymbol{K})^{-1} \mathbf{y}, \boldsymbol{K}_{**} - \boldsymbol{K}_*^T (\sigma^2 \mathbf{I} + \boldsymbol{K})^{-1} \boldsymbol{K}_*) \tag{2.60}
$$

[28] **Remark**: the crucial part of the last step was to show that the remaining integral is constant w.r.t. $\mathbf{f}_*$. The fact that there is an analytical solution for GPR makes this more obvious. But even if it was intractable, as long as it is constant w.r.t. $\mathbf{f}_*$, it has only a scaling-effect on the predictive distribution. The fact that $\mathbf{f}_*$ only appears in the mean is sufficient to claim that the interval is constant w.r.t. $\mathbf{f}_*$. The mean of an (unnormalized) Gaussian has only an influence on the location, but not the shape of the function. This implies that the area under the function remains constant.

2.4.7 Conclusion

Complexity Considerations

After detailed derivation of the predictive distribution, the justification of the first two initial claims on page 27 are straight forward. It became clear that all the underlying infinite integrals (cf. Figure 2.4) had either analytical solutions or were not even relevant for *the predictive*.

Furthermore, the integral to calculate the *expected value* of the prediction has an analytical solution as well. This is however a special property of GPR, where the predictive is Gaussian. This does not necessary hold for other applications like GP-Classification. But even in such cases the outer 1-dimensional[29] integral can often be evaluated efficiently using numerical approximations.

The computational complexity of the *training* of a GP is $O(N^3)$[30]. Part of the *training* phase is usually everything which can be precomputed independent of $\mathbf{X}_*$. This is dominated by the factor $(\boldsymbol{K} + \sigma^2\mathbf{I})^{-1}$ to calculate the mean and covariance. The inversion of the covariance matrix of N training inputs results in the cubic time-complexity.

The *prediction phase* is then the dominated by the matrix multiplications. The time-complexity of $\boldsymbol{K}_*^T(\sigma^2\mathbf{I} + \boldsymbol{K})^{-1}$, which corresponds to $\mathbb{R}^{M\times N} \times \mathbb{R}^{N\times N}$ is $O(N^2M)$.

Non-Gaussian Likelihood - Gaussian Process Classification

Regarding the problem of non-Gaussian likelihoods, the findings gained through this derivation were surprising.

Similar as in **Generalized Linear Models** (GLM), the *likelihood*[31] is the only part which distinguishes regression from classification, discrete predictions etc. Binary classification for example uses a *Bernoulli* likelihood. The likelihood is part of the posterior. For regression, section 2.4.5 shows that the GP-prior (a Gaussian) and a factorized Gaussian likelihood results in a Gaussian posterior. Using another form of likelihood will often not result in a Gaussian posterior. This has several consequences.

A non-Gaussian posterior might be intractable because of the integral in the marginal likelihood term being intractable. However, even tractable non-Gaussian posteriors may lead to an overall intractable predictive distribution. Section 2.4.6 shows that *the conditional* times *the posterior* in GPR is still a Gaussian. This is holds only because

[29] 1-dimensional if only 1 test output is predicted. However, because multiple (e.g. M) test outputs $\mathbf{f}_*$ are assumed to be conditionally independent given $\mathbf{X}_*$, $\mathbf{X}$ and $\mathbf{y}$, this reduces to the approximations of M independent 1D-integrals.

[30] Assuming school-book algorithms for matrix algebra

[31] including and associated *link-function*

both are already (unnormalized) Gaussians. If this is not true, then the resulting density may be non-Gaussian. This again introduces the risk that either the density itself is intractable, or that integrating this density over **f** is intractable.

The conditional is independent of the likelihood and always a Gaussian. Inside the posterior, the prior (a GP) is also fixed to a Gaussian. One way to assure that likelihood times the prior results in Gaussian is the concept of **Conjugate Priors**. A distribution is called *conjugate prior* for a given likelihood, if the distribution of the posterior stays in the same family as the prior. For the Gaussian likelihood for example, a Gaussian (or GP) prior[32] is a conjugate prior.

One way to avoid intractabilitys in GPs is therefore to use only likelihoods for which the Gaussian distribution is a conjugate prior. Unfortunately, it turns out that the Gaussian likelihood is the only common one having a conjugate Gaussian prior. This is the reason why various methods (cf. 2.1.4) approximate the posterior by a Gaussian. In this case the predictive distribution stays Gaussian and tractable irrespective of the likelihood.

This relation to the problem of conjugate distributions, as well as the different sources of intractability, are the key insights of this chapter.

[32] for the mean of the likelihood

2.5 Kernels

Kernels are the key ingredients of Gaussian Processes. A kernel is what defines **similarity** of two inputs. Based on this similarity, a GP predicts similar outputs for similar inputs. If the kernel is not appropriate for the task, performance will be poor.

As an example, imagine to predict the top-speed of a car based on its color, horse-power and weight. A kernel which compares cars by their color only will most likely not be very successful. A better approach in this case might be to design a kernel which *calls* cars similar if they have equal horse-power and weight.

There are various forms of kernels applicable for any type of data (e.g. discrete, text, image, matrices etc.). Duvenaud 2014 gives a comprehensive discussion on various types. It is also possible to combine kernels through addition and multiplication. This section summarizes most common kernels, important properties and special forms used in the experiments.

2.5.1 Prior over Latent Functions

A GP is often called a *prior over functions*. The family, i.e. the characteristics of those functions, is determined by the kernel. For six common kernels, some sampled functions, the shape of the kernel function and corresponding covariance matrix are illustrated in figure 2.5. Table 2.1 shows the corresponding definitions.

The well-known **Squared Exponential** (`SE`) kernel defines similarity based on the *exponential decaying squared euclidean distance* of the inputs. Other common names are **Gaussian** kernel and **Radial Basis Function** (RBF). It is parameterized by a single hyperparameter - the *characteristic length-scale* l.

The **Matérn** (`Mat`) kernel is a generalization of `SE`. Its kernel function is slightly more peaked. This allows more local fluctuations because covariance decays faster at the small distances. Like `SE`, it retains the overall smooth shape of the resulting functions. The *peakedness* is controlled by an additional hyperparameter v. Common values for v are $\frac{3}{2}$ and $\frac{5}{2}$. For $v \to \infty$, the `Mat` kernel approaches `SE`.

The **Rational Quadratic** (`RQ`) is equivalent to adding together many SE kernels with different *length-scales*[33]. It represents functions which vary smoothly across multiple *length-scales*. Hyperparameters are the average *length-scale* l and α, which is a trade-off between the high and low *length-scales*.

The **Exponentiated Sine Squared** (`ExpSineSq`) kernel allows to model periodic functions. Its hyperparameters are the *length-scale* l and *periodicity* p.

[33] Duvenaud 2014, section Rational Quadratic Kernel.

Kernel	Kernel Function $\mathbf{k}(\mathbf{x}, \mathbf{x}')$
SquaredExponential	$\exp \frac{(\mathbf{x}-\mathbf{x}')^T(\mathbf{x}-\mathbf{x}')}{2l^2}$
Matérn	cf. literature[35]
RationalQuadratic	$(1 + \frac{(\mathbf{x}-\mathbf{x}')^T(\mathbf{x}-\mathbf{x}')}{2\alpha l^2})^{-\alpha}$
ExpSineSquared	$\exp \frac{2\sin^2(\lvert\mathbf{x}-\mathbf{x}'\rvert^2/p)}{l^2}$
DotProduct	$\mathbf{x}^T\mathbf{x}'$

TABLE 2.1: Kernel functions of common kernels.

The **DotProduct** (`Dot`) kernel is also called the `Linear` kernel. The resulting functions are straight lines crossing the origin. A GP with a pure `DotProduct` kernel is equivalent to Bayesian Linear Regression[34].

2.5.2 Signal Variance

Most kernels result in a fundamental **signal variance** of 1 (diagonal elements of covariance matrix). For the `SE` kernel for example, the distance between two equal inputs is zero. This leads to a variance $k_{SE}(\mathbf{x}, \mathbf{x}) = \exp(0) = 1$. If the observed values have a variance other than 1, a kernel (or combination of kernels) has to be multiplied by a scalar to allow more or less deviation from the mean in general. This factor is therefore a common hyperparameter of any kernel and is usually called the *signal variance* σ_s^2:

$$\sigma_s^2 * k(\mathbf{x}, \mathbf{x}') \tag{2.61}$$

Another approach is to assure a variance of 1 by transforming the observations in a preprocessing step. This is similar to a preprocessing step to assure $\mathbf{y}$ (and therefore $\mathbf{f}$ if the identity link is used) to have a zero mean (cf. section 2.4.4).

[34] When a Gaussian prior over the weights/slopes is used

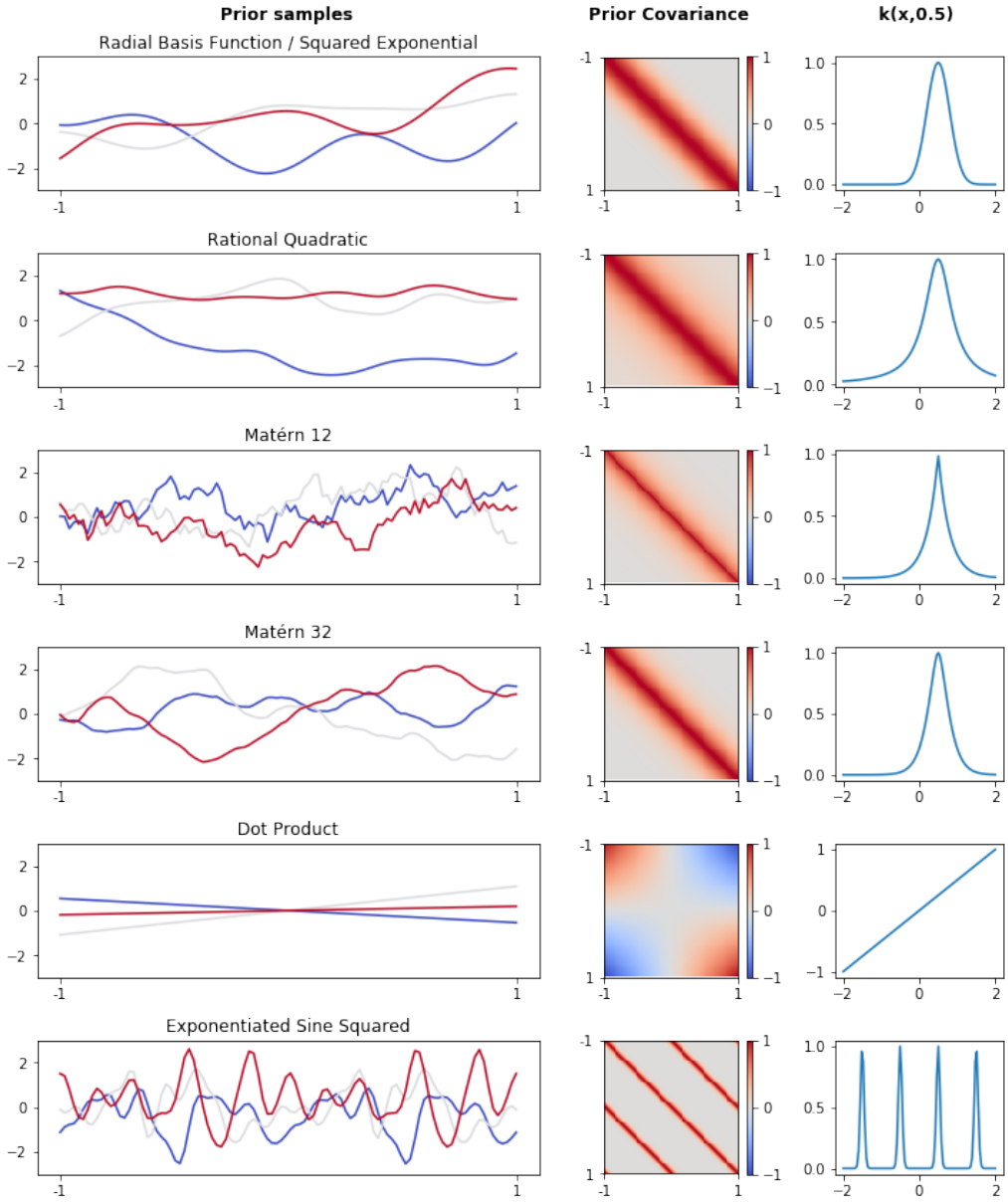

FIGURE 2.5: First column: Three samples from the GP prior. Second column: Covariance function of the GP prior. Third column: Corresponding kernel function. Covariance of scalar inputs between -2 and 2 and a fixed scalar of 0.5 is shown. Value 0.5 was chosen because a value of 0 results in a degenerate covariance for the `DotProduct` kernel. For all kernels with a *length-scale* parameter, l is 0.3 `Matérn12` indicates that $\alpha = \frac{1}{2}$. `Matérn32` indicates that $\alpha = \frac{3}{2}$. Periodicity for `ExpSineSq` is 1.0.

2.5.3 Stationary and Non-Stationary Kernels*

A kernel is stationary if only the relative position of two inputs is relevant for their covariance.

Using the `SE` kernel for example, the covariance $k_{SE}(5,6)$ is equal to $k_{SE}(8,9)$. I.e. only the distance between the two inputs, but not their absolute position, is relevant. A counter-example is the `DotProduct` kernel where $k_{Dot}(5,6) = 30$ but $k_{Dot}(8,9) = 72$.

2.5.4 Isotropic and Anisotropic Kernels*

Most kernels can be extended naturally to operate on multi-dimensional inputs (cf. vector-operations used in table 2.1). However, they can be generalized further. For example, instead of a global *length-scale* parameter, a *length-scale* per dimension can be defined. Such a kernel is called *anisotropic*. In the most basic example, it represents functions with a different degree of smoothness per dimension. Figure 2.6 shows an example for 2-dimensional inputs.

Formally, a kernel is anisotropic if a change in input dimension A and the same change in input dimension B has different impacts on the covariance between the original and changed input.

Anisotropic kernels are often used for **Automatic Relevance Determination**[36] (ARD). Kernel hyperparameters $[l_1, ...l_D]$ are optimized w.r.t. the marginal likelihood. A small *length-scale* for a dimension is then interpreted as indicator for a *relevant* feature. "If the length-scale has a very large value, the covariance will become almost independent of that input, effectively removing it from the inference" (Rasmussen and Williams 2005, p. 107).

[36] Neal 1995.

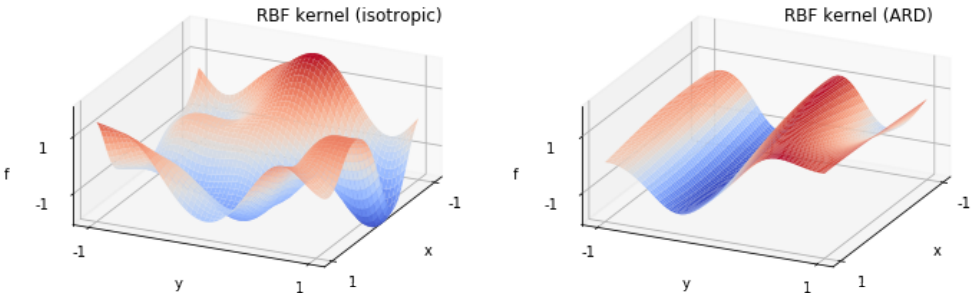

FIGURE 2.6: Prior sample of latent values for GP on 2-dimensional inputs. Left plot shows a GP with isotropic kernel, i.e. *length-scale* l is 0.3 for x and y axis. Right plot shows a GP with anisotropic kernel, where $l_x = 2.0$ and $l_y = 0.3$. Here, given the value of y, the value x is more or less irrelevant regarding the value of f.

2.6 Model Selection

Model selection refers to the process of searching a model which explains the data at hand. Selection occurs on **three levels of abstraction**[37].

On the first level, this is a discrete choice of a model $\mathcal{H}$ from **fundamental types of models**: Decision Trees, Neural Networks, Gaussian Processes etc.

On the second level, continuous and/or discrete values for the **hyperparameters** $\boldsymbol{\theta}$ of one model have to be chosen. This corresponds for example to the number of layers in a NN, the depth of decision trees or the kernel-hyperparameters in a GP. The choice is often based on heuristics or a mechanic grid-search in the hyperparameter space.

On the third level, the **parameters of the model** are often *learned* with a more sophisticated learning-algorithm. *Back propagation* is an example for parameter (weights) optimization in neural networks. Variational inference is the counterpart to estimate the latent values $\mathbf{f}$ in a Gaussian Process. Most of these learning algorithms use the posterior (or at least parts of, i.e. the likelihood) to find good values according to observed data.

2.6.1 Third Level Inference

The posterior is usually formulated as a probability distribution over the parameters, conditioned on the data $\mathcal{D}$[38]. The preceding decisions for a type of model and some values for the hyperparameters are often not indicated explicitly. This is reasonable regarding the notional overhead. In this section however, the explicit posterior is used to introduce a more sophisticated way for second-level inference:

$$P(\mathbf{f}|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta}, \mathcal{H}) = \frac{P(\mathbf{y}|\mathbf{f}, \mathbf{X}, \boldsymbol{\theta}, \mathcal{H}) P(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}, \mathcal{H})}{P(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \mathcal{H})} \tag{2.62}$$

This formulation is equivalent to the posterior described in section 2.4.5. The only (notational) difference is that the implicit conditioning on a fixed model $\mathcal{H}$ and corresponding fixed hyperparameter values $\boldsymbol{\theta}$ is now explicit.

[37] Rasmussen and Williams 2005, Chapter 5.2.

[38] Notation of *data* depends on task and algorithm. For unsupervised tasks just $\mathbf{X}$. For supervised task usually $\mathbf{X}, \mathbf{y}$.

2.6.2 Second Level Inference

For the second level, the *posterior over hyperparameters* $\boldsymbol{\theta}$ can be defined similar to equation (2.62):

$$P(\boldsymbol{\theta}|\mathbf{y},\mathbf{X},\mathcal{H}) = \frac{P(\mathbf{y}|\mathbf{X},\boldsymbol{\theta},\mathcal{H})P(\boldsymbol{\theta}|\mathcal{H})}{P(\mathbf{y}|\mathbf{X},\mathcal{H})} \tag{2.63}$$

The marginal likelihood from eq. (2.62) is now used as the *second-level-likelihood*. Remember that this term consists of a multidimensional infinite integral of the likelihood times the prior over all possible parameters values $\mathbf{f}$ (cf. eq. 2.6 on page 15). $P(\boldsymbol{\theta}|\mathcal{H})$ is an appropriate prior over the hyperparameters and also called **hyperprior**. $P(\mathbf{y}|\mathbf{X},\mathcal{H})$ is independent of $\boldsymbol{\theta}$.

A major advantage of Gaussian Process Regression is that the integral in the marginal likelihood and therefore the *second-level-likelihood* has an analytical solution. A maximum likelihood estimate of $\boldsymbol{\theta}$ can therefore be found by optimizing the *second-level-likelihood*[39] w.r.t. $\boldsymbol{\theta}$. This is called a **Type II Maximum Likelihood** (ML-II).

$$\begin{aligned}
\text{ML-II}_{\boldsymbol{\theta}} &= \text{argmax}_{\boldsymbol{\theta}} P(\boldsymbol{\theta}|\mathbf{y},\mathbf{X},\mathcal{H}) && (2.64)\\
&= \text{argmax}_{\boldsymbol{\theta}} \frac{P(\mathbf{y}|\mathbf{X},\boldsymbol{\theta},\mathcal{H})P(\boldsymbol{\theta}|\mathcal{H})}{P(\mathbf{y}|\mathbf{X},\mathcal{H})} && (2.65)\\
&\propto \text{argmax}_{\boldsymbol{\theta}} P(\mathbf{y}|\mathbf{X},\boldsymbol{\theta},\mathcal{H}), \quad \text{assuming flat hyperprior } P(\boldsymbol{\theta}|\mathcal{H}) && (2.66)\\
&= \text{argmax}_{\boldsymbol{\theta}} \int_{\mathbf{f}} P(\mathbf{y}|\mathbf{f},\mathbf{X},\boldsymbol{\theta},\mathcal{H})P(\mathbf{f}|\mathbf{X},\boldsymbol{\theta}\mathcal{H}) && (2.67)
\end{aligned}$$

This integral has an analytical solution for Gaussian Process Regression. Otherwise an approximation must be found.

2.6.3 Marginal Likelihood as Optimization Criterion

The previous section showed how a type II maximum likelihood estimate of the hyperparameters is obtained by maximization of the marginal likelihood. In general, any optimization introduces the risk of overfitting. This applies to the ML-II estimates, especially if there are many hyperparameters[40]. The marginal likelihood however distinguishes the Bayesian scheme of inference from other schemes based on optimization. "It is a property of the marginal likelihood that it automatically incorporates a trade-off between model fit and model complexity. This is the reason why the marginal likelihood is valuable in solving the model selection problem."(Rasmussen and Williams 2005, p. 110).

[39] The *second-level-prior* is often assumed to be flat and therefore irrelevant for optimization. If prior believes about the hyperparameters are available they can of course be incorporated, leading to a MAP-II estimate

[40] Convolutional kernels or deep kernels (based on neural networks) for example can have an arbitrary number of kernel hyperparameters.

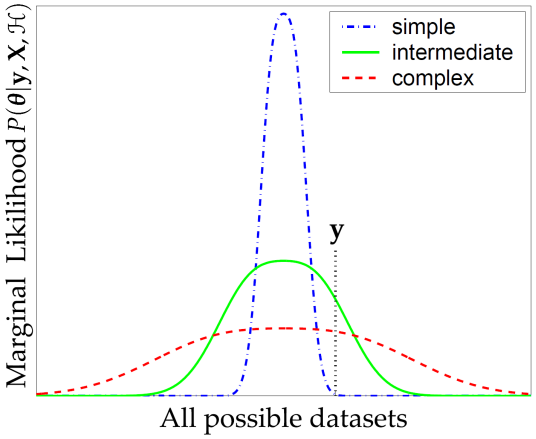

FIGURE 2.7: Source: Rasmussen and Williams 2005, Fig. 5.2. The horizontal axis is an idealized representation of all possible vectors of observations $\mathbf{y}$. For a particular observation, indicated by $\mathbf{y}$, the marginal likelihood prefers a model of intermediate complexity over too simple or too complex alternatives

The marginal likelihood $P(\mathbf{y}|\mathbf{X},\boldsymbol{\theta},\mathcal{H})$ is itself a probability distribution. For given inputs and model hyperparameters, it is the probability density (likelihood) of any possible observed output $\mathbf{y}$. The definition of a probability function constraints the area below this probability function to be 1.0. For a very complex model, which is able to fit every imaginable observations, some probability mass has to be assigned to every possible dataset - the shape of the function is therefore very broad. Together with the mentioned constraint, this implies that the function values are rather low. If the model can only fit very simple (e.g. linear) observations, zero probability is assigned to more complex datasets. This implies a more peaked shape of the function and higher function values in this region. Figure 2.7 gives an intuition for this claim.

The marginal likelihood allows to choose optimal hyperparameters while automatically including a trade-off between model-complexity and model fit. This effect is also called **Occam's Razor**[41].

2.6.4 Marginal Likelihood vs. Cross-Validation

Marginal Likelihood and Cross-Validation are both methods for model selection.

The main advantage of marginal likelihood is that optimal hyperparameters can be chosen automatically. This makes the very inefficient grid-search superfluous. A major disadvantage is that it assumes the posterior to be tractable. For most successful models, such as deep neural networks, this is not the case. Another assumption is that the *true* model is among the models under investigation. In simple Bayesian Linear Regression for example, the most likely value for the *noise* hyperparameter σ^2 is found regarding all possible linear models. However, if a linear model would be

[41] Sometimes paraphrased by "The simplest solution is most likely the right one."

appropriate in the first place, or if a more complex model might be necessary, cannot be determined by the marginal likelihood.

Cross-Validation has neither of these assumptions. It is based on the *Leave-p-Out Accuracy* - a measure of generalization. Computation is often easier and applicable for any type of model. Unfortunately, hyperparameters have do be defined in advance.

Finally, a recent paper by Fong and Holmes 2019 shows an interesting connection between the two. It claims that (Bayesian) Cross-Validation is in fact an approximation to the marginal likelihood.

If marginal likelihood, cross-validation or a combination of both is used, depends on the used models and goal. For hyperparameter optimization, marginal likelihood seems to be less prone to overfitting. Cross-Validation is the general purpose tool to report generalization performance.

Remark

By definition, a likelihood is in $[0, 1]$. This implies that a log-likelihood is in $[-\infty, 0]$. In various experiments, a positive value for the log marginal likelihood was observed (e.g. figure 2.15).

Because outputs $\mathbf{y}$ are often continuous, it is in fact the *density* of the log likelihood which is maximized. A density is not constrained to be in $[0, 1]$ but can have any positive value. It is therefore legitimate that the log-density of the marginal likelihood is positive.

The term **(log) marginal likelihood** is often used interchangeably. Depending on the context, it refers to the effective likelihood as a concept or its *density* used as objective for optimization.

2.7 Related Methods

Gaussian Processes were originally developed quite in isolation. Though, they are closely related to various common and well-studied methods in Machine Learning.

A GP with a linear (i.e. `DotProduct`) kernel corresponds to **Bayesian Linear Regression**[42] with a Gaussian prior over the weights. For an arbitrary kernel, GPR can be compared to **Kernel Ridge Regression**[43]. GPR has however two major advantages: Optimization of hyperparameters based on gradient descent (compared grid-search in KLR) and the prediction of confidence intervals. Figure 2.8 shows various paths from which GP models can be derived.

An important correspondence to **neural networks** was discovered by Neal 1995. For **neural networks** with a single infinite-wide hidden layer, the prior over functions tends to a Gaussian Process[44]. There are consequences in both fields. For Gaussian Processes, it implies that the **Universal Representation Theorem** holds[45]. However, this result might be less powerful than it appears[46]. For neural networks, Williams

[42] Or Bayesian Logistic Regression - depending on the likelihood used.
[43] For the same hyperparameters, the prediction of KLR and *the mean of GPR* are equivalent
[44] This can be shown for various activation functions and priors over the weights
[45] I.e. a GP can approximate any continuous function to any degree of precision.
[46] DeltaIV n.d.

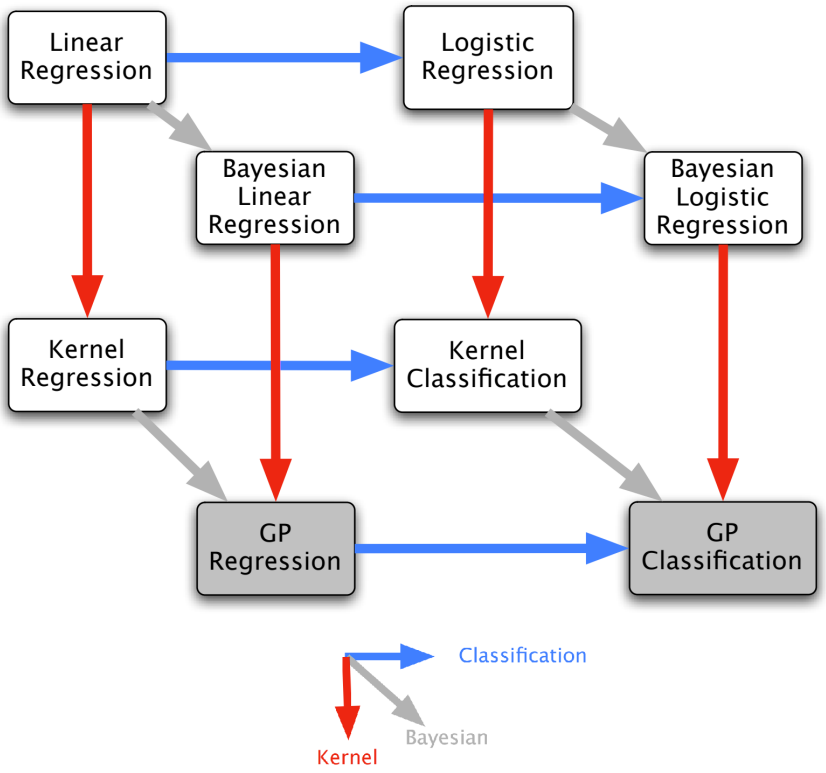

FIGURE 2.8: Source: Ghahramani 2011, Slide 18. Relation of Gaussian Processes and other methods.

1996 shows how **efficient and exact calculations in infinite-wide NNs** can be performed using GPs.

Finally, the non-parametric and kernel-based approach makes a relation to **Support Vector Machines** obvious. Seeger 2002 shows that both, under some conditions, are special forms of a *Smoothing Splines* model. From a practical perspective, Gaussian Processes seem to be a natural approach for regression. SVMs, as *maximum margin classifiers*, are mostly used for classification tasks. Both can however be adapted to the other scenario respectively. Advantages of GPs are again the ability to optimize hyperparameters and the full probabilistic predictions. From a computational perspective, SVMs might have slight advantages depending on the kernels used. Both involve a non-linear optimization problem[47]. In addition, SVMs are dominated by the calculation of N^2 dot-products, while the inversion of the covariance matrix for GPs involves N^3 operations.

[47] Loss minimization (SVM) vs. marginal likelihood maximization (GP)

2.8 Deep Gaussian Processes*

A Gaussian Process is a probabilistic mapping from inputs $\mathbf{X}$ to outputs $\mathbf{y}$. This can be illustrated with a graphical model, where nodes represent random variables and edges the GP-mapping between them. Figure 2.9(a) shows such models using a single GP.

Multiple layers can be stacked to form a **Deep Gaussian Process**. H layers correspond to H independent GP mappings. Except for the first mapping, all *latent inputs* $\{Z^{\{1\}}, ..., Z^{\{H-1\}}\}$ are now random variables, distributed according to *the predictive* of their preceding GP. Figure 2.10 shows the graphical model. Each layer is therefore effectively a GP-LVM as outlined in figure 2.9(b). Damianou and Lawrence 2013 show how the complete latent space of arbitrary deep GPs can be marginalized variationally.

The output of a GP is usually 1 dimensional. Latent spaces can however be extended to multiple dimensions. Latent random variable Z^h is then effectively a random vector of arbitrary dimensionality D_h. Each component of Z^h is usually distributed according to an independent GP inside layer h. This is similar as the output vectors in multiclass GP classification.

FIGURE 2.9: Single layer GP models. Shaded nodes indicate observed data, i.e. variables without uncertainty. a) A supervised learning task using a GP mapping. b) In a **GP Latent Variable Model**, inputs are not observed. A GP-LVM is useful for non-linear dimensionality reduction. Here, Y does not represent scalar outputs, but any observed D-dimensional dataset. A GP-LVM finds a lower-dimensional representation $\mathbf{X} \in \mathbb{R}^{N \times Q}$ where $Q < D$. In the simplest case[48], the marginal likelihood is maximized w.r.t. to $\mathbf{X}$ and the GP hyperparameters. If an SE kernel with ARD is used, the necessary dimensionality of X can then be determined by the number of the dimensions with small *length-scale*.

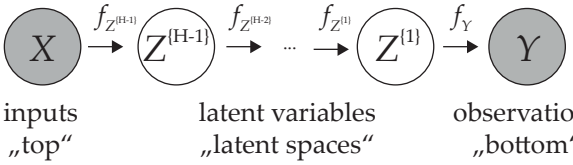

FIGURE 2.10: Stacked GP layers. Each intermediate node represents the latent random vector $Z^h = [f_h^{\{0\}}(Z^{h-1}), \cdots, f_h^{\{D_h\}}(Z^{h-1})]^T$.

2.8.1 Representational Capacity of Deep GPs

Deep GPs have a higher capacity than single layer GPs. This is similar to the capacity of Deep NNs being bigger than of Generalized Linear Models. A single GP is bounded by the expressiveness of the kernel. Deep GPs allow to model long-term correlations and non-stationary variance. Figures 2.11 to 2.13 show functions sampled by DGPs with different number of layers.

The **step function** is a typical example which is hard to model with a GP. Immediate jumps in the data contradict the assumption of a smooth underlying function. In general, the *length-scale* is often dominated by the smallest *wiggle* in the data. Figure 2.14 to 2.16 show the result of a single GP and Deep GPs with two and three layers fitted to such data.

The variational marginalization of the latent space gives a lower bound of the log marginal likelihood (cf. ELBO on p. 18). The width and number of layers can then be selected based on this criterion.

For the step data, the 3-layer model seems to explain the data best. The ELBO is significantly higher than for two layers. A visual inspection does not show great differences. The *length-scale* of the top layer is slightly higher for 3 layers. This implies less flexible functions which might be the reason for the higher ELBO. Figure 2.17 shows prior functions sampled by the two trained models on the step data. Even in this simple 1D example, it would be hard to tell which prior might be better suited for this particular dataset.

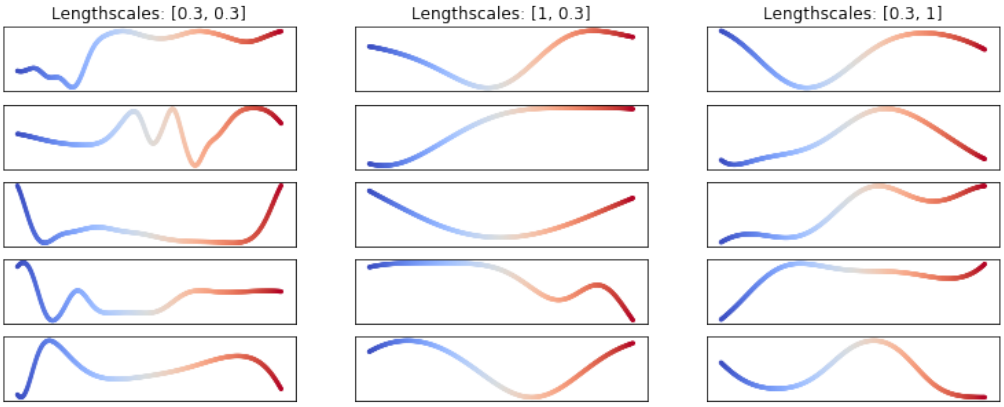

FIGURE 2.11: Functions generated by 2 stacked GPs with SE kernels.

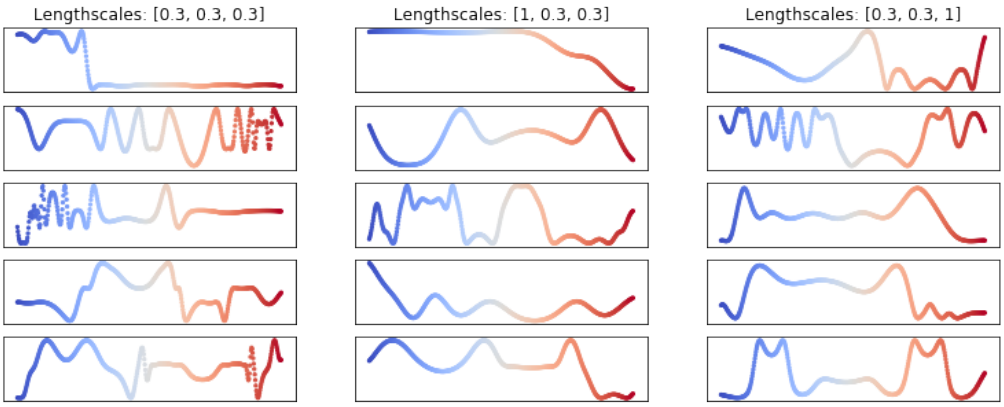

FIGURE 2.12: Functions generated by 3 stacked GPs with SE kernels.

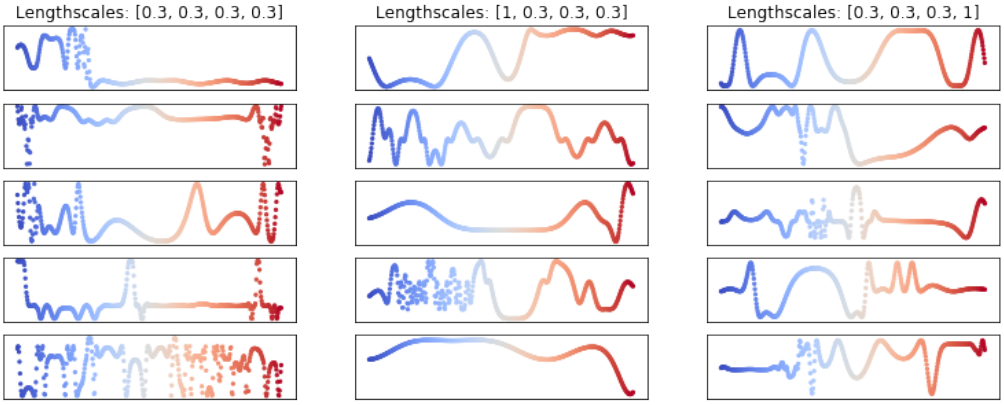

FIGURE 2.13: Functions generated by 4 stacked GPs with SE kernels.

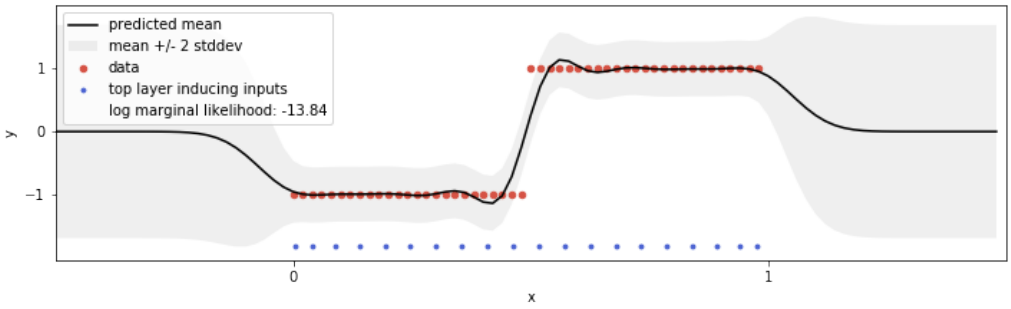

FIGURE 2.14: Single GP with SE kernel. Optimized *length-scale* is 0.07.

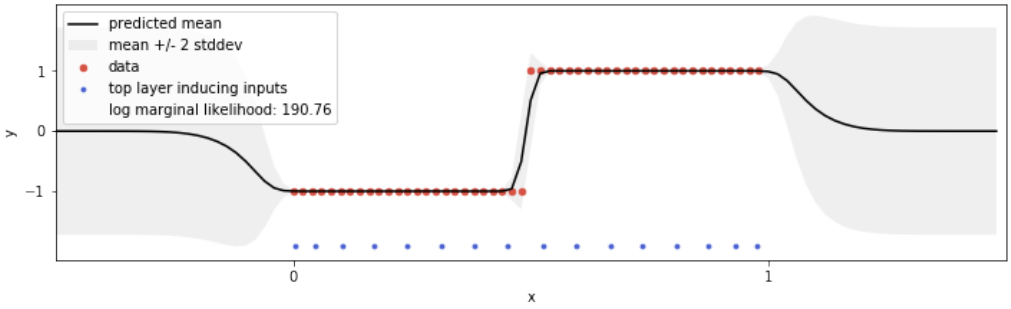

FIGURE 2.15: 2 stacked GPs with SE kernel. Optimized *length-scales* are $[0.09, 1.3]$

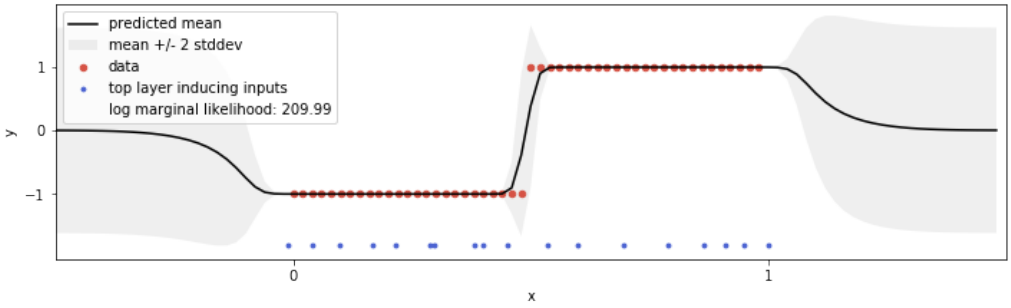

FIGURE 2.16: 3 stacked GPs with SE kernel. Optimized *length-scales* are $[0.14, 0.4, 2.57]$. Note that in terms of ELBO, the 3-layer GP explains the data best.

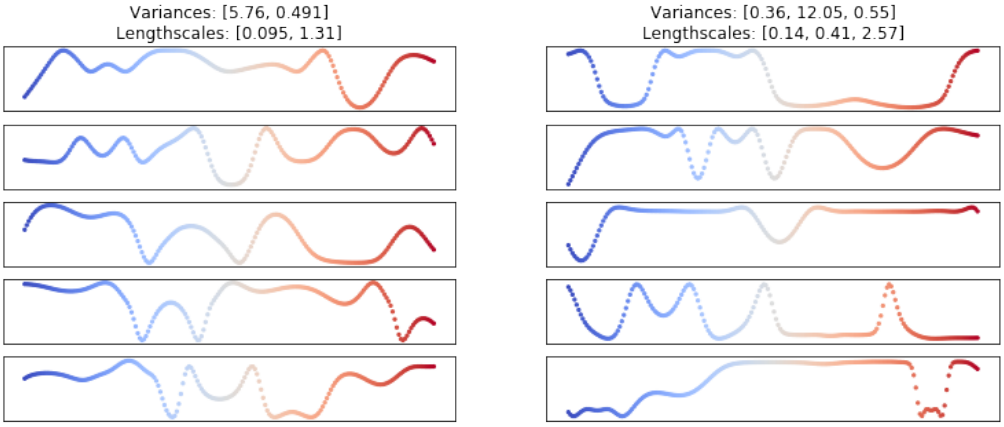

FIGURE 2.17: Prior functions sampled by the 2 and 3-layer GPs with hyperparameters optimized on step function data.

Chapter 3

Experiments

3.1 Visualizing Posterior Mean and Variance

The goal of the first experiment was to implement a vanilla Gaussian Process from scratch. Using `numpy` and `sklearn` for linear algebra operations, the implementation of the predictive distribution was straight forward. The following features are implemented in `01_Simple_GP.ipynb`.

- Calculation of posterior mean and covariance for GP Regression[1]
- Sampling from the posterior
- Plot posterior mean, confidence region and samples
- Compare prior and posterior covariance visually

Figure 3.1 illustrates the result for a Gaussian Process using the *Squared Exponential* (`SE`) kernel.

3.1.1 Conclusion

An interesting effect of the conditioning can be observed. As expected, the posterior is equal to the prior if there is no observed data (cf. first row). After conditioning on some training samples, the corresponding variance for that inputs becomes smaller (cf. diagonal elements and confidence region at X_3, X_{14}, X_{17} in row 2). Surprisingly, there are also blue elements in the posterior covariance matrix. They indicate that covariance between close outputs on the and left an right side of observed datapoints have negative covariance. I.e. if the left output is below its posterior mean, then the right element is deemed to be above its posterior mean.

Intuition for the `SE` kernel suggests that it introduces positive covariance between close[2] inputs and zero covariance otherwise. This is supported by the plot of the prior covariance matrices. It is further comprehensible that the posterior mean around observed inputs is close to the observed outputs. This effect already introduces

[1] I.e. assuming identity link function and Gaussian noise/likelihood
[2] squared distance in euclidean space

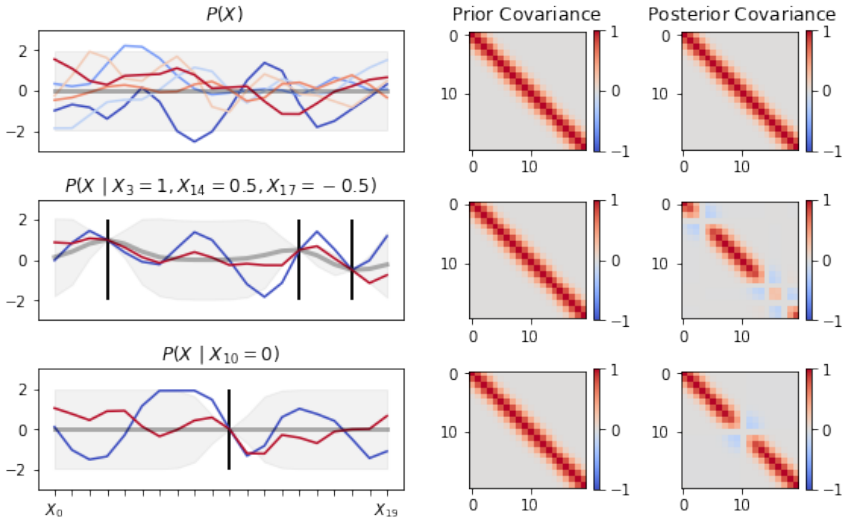

FIGURE 3.1: Plots in the left column show some posterior samples (colored lines), the posterior mean (bold gray line) and the 95% confidence region (light-gray area). The prior mean $\boldsymbol{\mu} = \mathbf{0}$ for all three rows. SE kernel hyperparameters are $\sigma_s^2 = 1$ and *length-scale* $l = 1.5$

smoothness into the resulting prediction. It was not obvious however, that the posterior covariance introduces another type of smoothness.

Consider the condition that $X_{14} = 0.5$ in row 2 of figure 3.1. Here, X_{13} and X_{15} have negative covariance. In both samples X_{13} is below and (therefore) X_{15} is above the posterior mean. In other words, the slope[3] of the sampled function is similar on both sides of X_{14}. This can be loosely interpreted as a smoothing-constraint on the first derivative of the a sampled function. In the experiment, this effect can only be observed close to observed datapoints. Otherwise posterior covariance is strictly greater or equal to zero.

A first guess was that this effect is somehow dependent on the position or slope of the posterior mean or other observed points. Therefore, the experiment was repeated with only one observed datapoint X_{10} with a neutral value of 0. This value corresponds to the prior mean. The resulting posterior shows the same characteristic (cf. third row of figure 3.1).

To our knowledge, this effect was not mentioned or justified in any literature considered during this work. It might be interesting to see if our interpretation of a smooth first order derivative is justified in theory and if it holds for other kernels.

[3] Relative to the posterior mean

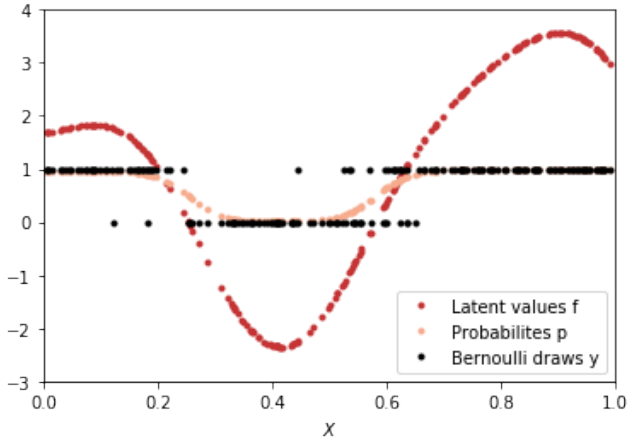

FIGURE 3.2: Generation of synthetic dataset. 1) Sampled latent function values $\mathbf{f}$ from a GP with SE kernel ($\sigma_s^2 = 7.0$ and $l = 0.2$). 2) Map f_i to a valid probability p_i using *sigmoid* link. 3) Sample observations $\mathbf{y}$ using corresponding $\mathbf{p}$'s

3.2 Binary Gaussian Process Classification on Synthetic Data

Gaussian Processes are very suitable for regression problems. The Gaussian likelihood and the conjugate GP prior allow analytic solutions. Classification of medical images requires several adaptions of the overall setup and approximate inference methods. The reason is the non-Gaussian likelihood.

Experiments in notebook `05_Gpy_Classification.ipynb` show how to apply GPs to classification. For easier visualization it is based on a synthetic binary classification task with one-dimensional inputs.

The dataset $\mathcal{D}$ consists of N datapoints $(\mathbf{x}, y)$. y is binary and represents the outcome of flipping a coin (1 = head and 0 = tail). The feature vector $\mathbf{x}$ contains only one single real valued feature - for example the temperature at the time of flipping. The assumption is that any y_i is independently Bernoulli distributed. Furthermore, it is assumed that the parameter $p_i \in [0, 1]$, of the Bernoulli distribution of y_i, is dependent on some latent function of $\mathbf{x}_i$. I.e. the temperature has an influence on how biased the coin is.

3.2.1 Generation of Synthetic Dataset

A dataset which conforms to this assumptions was generated as following. N inputs (corresponding to the temperatures) were sampled uniformly between 0 and 1. A single latent function was then sampled using a Gaussian Process with zero mean and SE kernel. Then, the *sigmoid* link-function was applied on $\mathbf{f}$ piece-wise to ensure valid values for $\mathbf{p}$. Finally, the observations $\mathbf{y}$ are N independent Bernoulli samples using the corresponding parameter value in $\mathbf{p}$. Figure 3.2 shows the resulting $\mathbf{f}, \mathbf{p}$ and $\mathbf{y}$.

3.2.2 Evaluating Resulting Posteriors

The dataset was generated using an `SE` kernel with $\sigma_s^2 = 7.0$ and $l = 0.2$. The values of those kernel hyperparameters are usually not known. If possible, they should be defined based on domain-knowledge. Otherwise they have to be estimated or learned from the data (cf. section 2.6).

Figure 3.3 - 3.6 show the resulting posterior for several values of σ_s^2 and l. It is surprising, that there is no perfect fit even for the true hyperparameters. The reason is that the *true* $\mathbf{f}$ are itself only one possible sample from this GP. It is possible that this particular sample might be more likely for slightly different hyperparameter values[4].

Figure 3.7 shows the result of **automatically learned hyperparameters**. In this example, a common iterative scheme was used[5]. Expectation-Propagation is used to approximate the posterior. Then, hyperparameters are optimized based on gradients w.r.t. the marginal likelihood. This procedure is iterated until convergence.

The learned *optimal* hyperparameters are $\sigma_s^2 = 7.24$ and $l = 0.24$. They are therefore quite close to the ground truth. From a subjective perspective however, the fit seems to be even a bit worse compared to the true values. Especially for inputs between 0 and 0.1 the values of $\mathbf{p}$ seem to be overestimated. The reason behind this optimum is the optimization criterion - the **(Log) Marginal Likelihood**.

[4]For the same reason that a sampled value 2.9, from a 1D Gaussian $\mathcal{N}(3.5, 1.0)$, is more likely under the model $\mathcal{N}(2.9, 0.5)$

[5]Default implementation of `GPy` for the `GPClassification` model

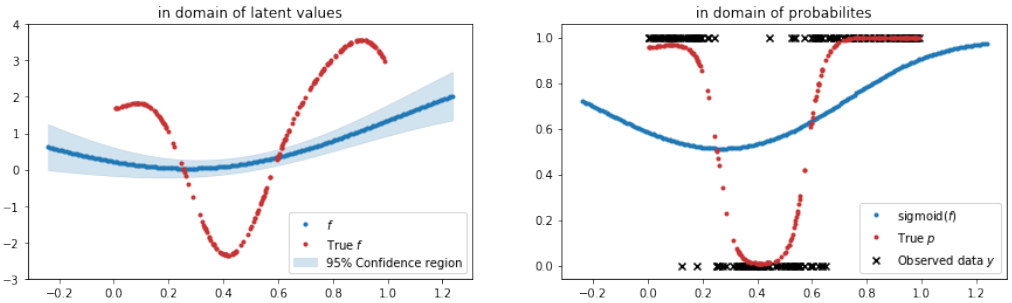

FIGURE 3.3: Resulting posterior (using EP approximation) for $\sigma_s^2 = 1.0$ and $l = 1.0$. Compared to the true values, σ_s^2 is too low and the characteristic *length-scale* l is too high. The resulting posterior is not flexible enough to model the true latent values. Note that confidence levels/regions have a special interpretation for binary outputs. They are omitted in the domain of probabilities for simplicity.

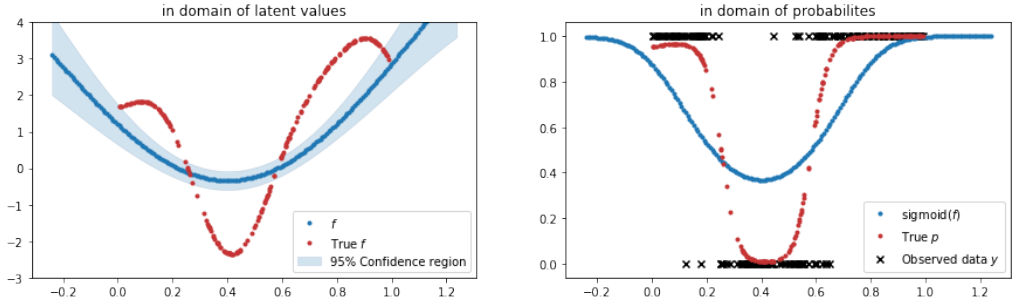

FIGURE 3.4: Resulting posterior (using EP) for $\sigma_s^2 = 7.0$ (true) and $l = 1.0$. The increased variance allows $\mathbf{f}$ to deviate more from zero. Due to the high *length-scale*, still not enough curvature is allowed.

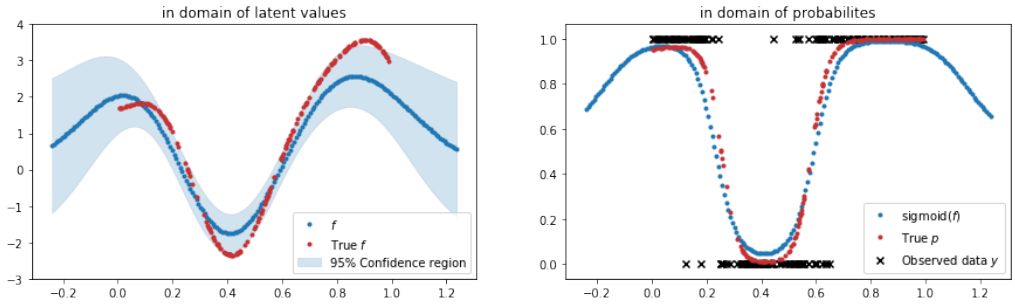

FIGURE 3.5: Resulting posterior (using EP) for $\sigma_s^2 = 1.0$ and $l = 0.2$ (true). Using shorter *length-scale* allows to model the overall shape of underlying $\mathbf{f}$. At the peaks (far from zero), the posterior mean still underestimates the truth due to the low σ_s^2.

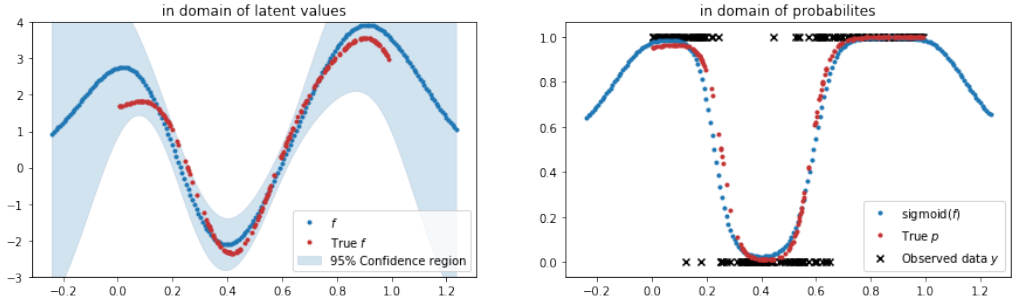

FIGURE 3.6: Resulting posterior (using EP) for true values of σ_s^2 and l. The best fit so far.

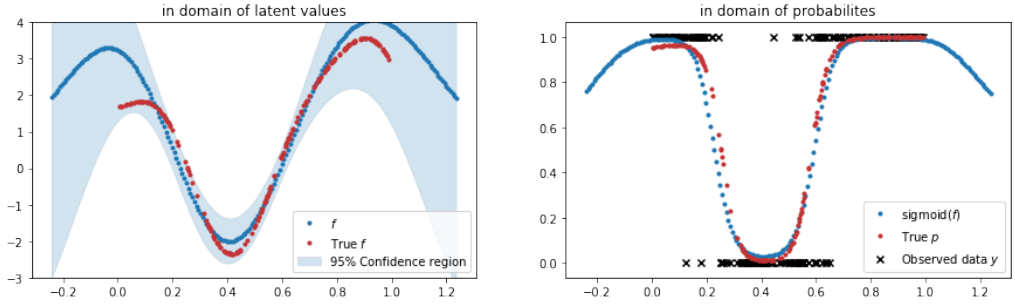

FIGURE 3.7: Resulting posterior (using EP and optimization of hyperparameters).

As described in section 2.6, the marginal likelihood marginalizes the latent values with an integral over the likelihood times the prior. It does not focus on one particular latent function, but takes all possible ones into account simultaneously. It can be shown that maximizing the log marginal likelihood automatically incorporates a regularization-term. GPs are therefore often said to be less prone to overfitting. Nevertheless, we observe multiple forms of overfitting in this experiment. The deviation of the optimal hyperparameters from their true values is the combined effect from:

1. The sampled *true* $\mathbf{f}$ during data generation does not represent a GP with SE(7.0, 0.2) perfectly. This mismatch is *generated* artificially in this experiment. The consequences are however equivalent if other model assumptions were violated. For example, if the smoothness of the underlying function is not stationary, which is assumed by the SE-kernel[6].

2. Another source of uncertainty is the sampling of observations from true and fixed latent probabilities $\mathbf{p}$. Such samples represent the underlying probabilities *perfectly* only in the limit of infinite samples. Even if the true model[7] is known, inferring the true latent function is only possible with infinite observations. This is not limited to GPs but inherent in any inference problem.

3. The *sigmoid* link and Bernoulli likelihood favor latent values with high magnitude. In regions with positive observations only, the most likely value of the latent function $f(\cdot)$ would be ∞. This explains why overestimation for temperatures between 0 and 0.1 increases the overall likelihood.

Table 3.1 shows the marginal likelihood of three models. According to this criterion, the model with optimized parameters explains the observed data best. This indicates at least, that the overall procedure worked and converged to a local optima.

3.2.3 Conclusion

Classification using Gaussian Processes is very similar to Logistic Regression or **Generalized Linear Models** in general. Especially regarding the use of a link function and adequate likelihood. It is therefore not surprising that it suffers from similar problems and biases.

Regarding **overfitting**, it is important to distinguish between the various sources. Gaussian Processes marginalize out the latent function values. I.e. they do not focus on only one most-likely latent function. In this sense, there is one less decision based on observations which probably do not represent the truth well. Kernel hyperparameters however remain a problem. As long as they are optimized - even w.r.t. a regularized criterion - they introduce the risk of overfitting. One approach

[6] Length-scale parameter is constant w.r.t. inputs
[7] I.e. the data generation process, including hyperparameters, is exactly known

Model (σ_s^2, l)	**Log-Marginal-Likelihood**
Baseline (1.0, 1.0)	-115.50
True (7.0, 0.2)	-49.97
Optimal (7.24,0.24)	**-49.87**

TABLE 3.1: Model comparison for binary classification.

to mitigate this risk is to use **Variational Inference**. In this setting, the kernel hyperparameters are itself treated as latent values with associated prior distributions[8]. Unfortunately the price is a harder inference problem. And eventually, it does only lead to more abstract decisions (family of priors vs effective values of hyperparameters).

To conclude with, Gaussian Processes relieve from the challenge of hyperparameter choice. They are of course not the answer to the *No-free-Lunch Theorem*.

[8]Which might be flat if no educated guess is possible

3.3 Jass Dataset

This section introduces the `Jass` dataset used in subsequent experiments.

During literature research, it became clear that high-dimensional input data causes additional challenges for GPs. We emphasize that this is not an inherent flaw or problem for Gaussian Processes. There are several propositions for kernels which leverage local structures in images for example. However, these approaches are quite young. Librarys are still experimental and examples and experiences in practical application are scarce.

The `Jass` dataset is well suited as surrogate for introductory experiments and benchmarks. Similar to `ISIC`, it is a multiclass classification task. The seven classes are not linearly separable[9], which justifies the use of a non-linear model like a GP. Compared to `ISIC`, another advantage was that no extensive preprocessing and cleaning of the dataset was necessary.

The goal of the task is to predict the first move of the beginning player in the swiss card game *Schieber*. Initially, a deck of 36 cards is evenly distributed to four players. Players sitting opposite form a team. Based on the own hand, the beginning player can decide for one of six modes of game play The 7^{th} option is to delegate the decision to the team member. This options is called *Schieben*.

The dataset was constructed from about 160k real online tournaments[10]. The **input** of one example consists of **37 binary features**. The first 36 represent the hand of the beginning player, i.e. always 9 out of 36 features are '1' and the rest is '0'. The last feature indicates if the first player already delegated the decision. In this case, the inputs are actually the hand of its team member and option *Schieben* is not allowed anymore.

[9] Cf. linear vs deep performance on 80k datasets in section 3.7.1
[10] Kindly provided by www.swisslos.ch

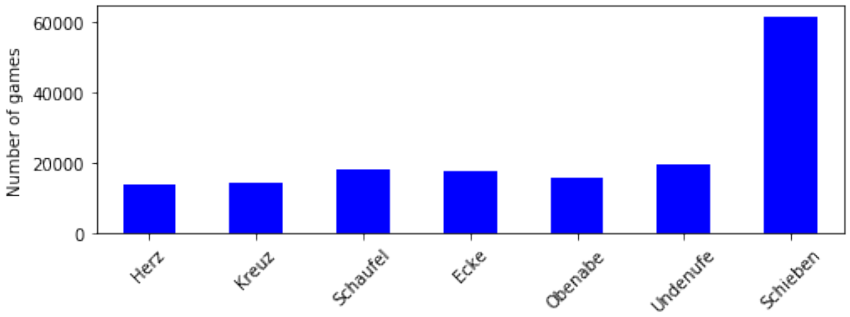

FIGURE 3.8: Distribution of target classes in `Jass` dataset

The **output** y is either a discrete number in $\{0, 1, 2, 3, 4, 5, 6\}$ or a **seven-dimensional binary one-hot vector**. Both forms represent the same information, i.e. the one out of seven actions chosen. The used form is an implementation detail.

Because *Schieber* is a game with partial information, the chosen actions are not necessarily the best choices. An accuracy of 100% is therefore not expected. No information on *human-level* performance (e.g. inter-expert agreement) is available. Figure 3.8 shows that classes are **severely imbalanced**. Almost 40% of the time, the chosen action was *Schieben*. Because imbalanced datasets often pose separate challenges in Machine Learning tasks, class *Schieben* was **excluded** for the experiments. It might be interesting for future work to investigate in the consequences of imbalanced datasets in GP models.

3.4 Multiclass Classification via Transformation to Binary

The machine learning framework `scikit-learn`[11] provides models for basic regression and classification using Gaussian Processes.

The implementation is based on algorithms 2.1 and 3.1 of Rasmussen and Williams 2005. The Laplace approximation to the posterior is used for classification. Unfortunately, multiclass classification is supported only via transformation to multiple binary problems. Common strategies for such multiclass classification are called *One-vs-All* and *One-vs-One*. They are applied widely and successful for models which do not support *true* multiclass classification naturally (e.g. vanilla SVM).

Gaussian Processes do support true multiclass classification. Rasmussen and Williams 2005 for example describe a multiclass Laplace approximation in chapter 3.5. The consequence is a harder inference problem. The following experiments, using `scikit-learn`'s `GaussianProcessClassifier`, serve as a baseline of this work.

Each model was trained with 1000 and 2000 training samples and validated on 10'000 test samples. Internally the hyperparameter optimization was restarted 5 times with randomly initialized values. **Random Restarting** is a trivial technique to escape local optima in gradient based optimization. Table 3.2 shows results for various training sets and tasks. For details consider notebook `04_Jass_sklearn.ipynb`.

[11] https://scikit-learn.org/stable/modules/gaussian_process.html

Task/Strategy	**ACC** N=1k	**ACC** N=2k	**ACC** N=4k	**CPU** N=1k	**CPU** N=2k	**CPU** N=4k
Binary (class *Herz* vs *Kreuz*)	0.961	0.962	0.964	2.2min	14.4min	58min
One-vs-Rest (incl. *Schieben*)	0.578	0.600	-	25min	1h 27min	-
One-vs-Rest (excl. *Schieben*)	0.736	0.751	-	21min	1h 17min	-
One-vs-One (excl. *Schieben*)	0.591	**0.752**	-	6min	21min	-

TABLE 3.2: Baseline models for variants of the *Schieber* task. All models use the SE-kernel. Accuracy (ACC) is calculated on 10k test samples. CPU is the total CPU time from parallel execution on 6 cores.

3.4.1 Conclusion

The dominant class *Schieben* has a significant effect on the performance. A reason might be that this class introduces inherent additional noise to the problem. In this case, the theoretical maximum performance would be lower for all models, including human experts. On the other hand, the imbalanced dataset might in fact be a problem for GPs (similar as for neural networks for example).

The performance of *One-vs-All* and *One-vs-One* does not show big differences. In theory, training time should be higher for *One-vs-One* because internally $\frac{6*5}{2} = 15$ models are trained, compared to 6 models in the *One-vs-All* strategy. Given the same dataset however, the single models in *One-vs-One* only operate on one third (2 out of 6 classes) of the training data. This is also an explanation for the bad performance of the *One-vs-One* model on 1000 samples.

Finally, the binary task shows the cubic increase of the training time in the number of training samples. With two times the amount of samples, we expect the time to increase by a factor $2^3 = 8$. Note that the best known algorithms[12] for matrix multiplication and inversion have a time complexity of about $O(N^{2.4})$ only. Therefore, a factor of $2^{2.4} \approx 5$ should be possible. This can be observed roughly by the 2.2min vs. the 14.4min, which is an increase by factor 6.5. Further, from 1k to 4k samples the expected increase in training time is $4^{2.4} \approx 28$. This corresponds roughly to the 58min vs. 2.2min, which is a factor of 26. The remaining differences are most probably due to varying convergence speed of internal hyperparameter optimization (i.e. more or less luckily initialized values).

[12] Cf. Coppersmith–Winograd algorithm

3.5 Evaluation of GPs and NNs for Binary Classification

A systematic benchmark of GPs and deep neural networks for binary classification has been conducted. Details can be found in notebooks `06_Jass_GPy_binary.ipynb` and `06_Jass_NN_binary.ipynb`.

The following **three neural networks** were trained and evaluated using 5-fold cross-validation. All hidden layers use the *ReLu* activation. For the output layer, the *Softmax* activation and *categorical-crossentropy* loss[13] is used.

- `Linear`: No hidden layer - direct mapping from 37 input to 2 output neurons
- `MLP4`: One hidden layer with 4 neurons
- `MLP16`: One hidden layer with 16 neurons

Figure 3.9 shows performance for several training set sizes. For 10k training samples, the performance of the `Linear` and deep `MLP16` model is comparable. The confidence interval for the validation accuracy is even slightly wider for the deep model. This indicates overfitting. It seems that the **two classes** are quite good **linearly separable**. More complex models are not able to increase validation accuracy.

Similarly, **three GP models** have been trained and evaluated using 5-fold cross-validation. All models use the `SE` kernel.

- `Laplace`: `scikit-learn` implementation of GPC using Laplace approximation
- `EP`: `GPy` implementation of GPC using Expectation Propagation to approximate the posterior
- `Sparse`: Sparse GP using only 150 inducing inputs (based on EP and DTC[14])

The performance for several training set sizes is also illustrated in figure 3.9. Figure 3.10 shows the optimized kernel hyperparameters.

[13] *Sigmoid* activation and binary-crossentropy might be appropriate as well for the binary classification task. Softmax was chosen to be consistent with subsequent multiclass experiments.

[14] One of the first proposed sparse approximations methods - summarized by Quiñonero-Candela, Ramussen, and Williams 2007

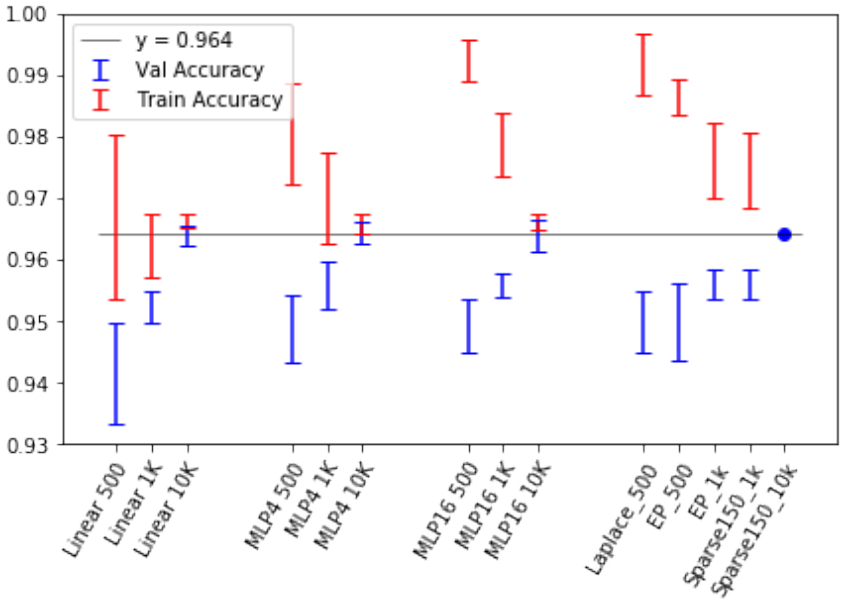

FIGURE 3.9: Training and Validation accuracy for binary classification (classes 0 and 1) on the `Jass` dataset. Error bars indicate 80% confidence intervals. Grey horizontal line serves for easier visual inspection only. For `Sparse150_10k` only one fold was trained due to long computation, i.e. no confidence interval is available.

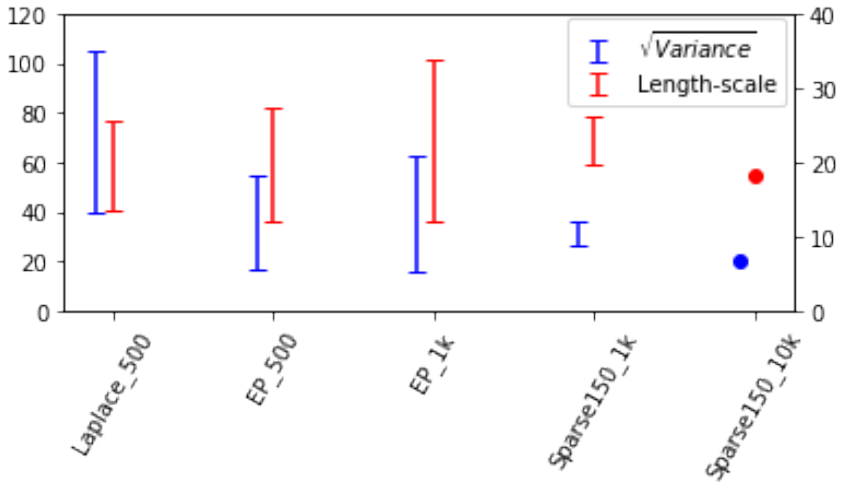

FIGURE 3.10: 80% confidence intervals for $\sqrt{\sigma_s^2}$ and length-scale l. The square root of the variance has been used for easier interpretation. Rather high values for the variance are due to the *sigmoid*-link function, which approximates 1 or 0 only at its fare tails. The *length-scale* seems to be constant w.r.t. to the number of samples. A decreasing *length-scale* would imply a model misspecification (cf. section *Pitfalls of the SE and RQ kernels* of Duvenaud 2014). This seems not to be the case here.

3.5.1 Conclusion

There are only few differences regarding raw validation set accuracy. For 10k training samples, all models show equal performance. This might be because the two classes seem to be quite linearly separable. The `MLP4` models seems to be a good trade-off between performance and overfitting on smaller datasets.

The `Sparse` GP models show some interesting results. Using only 150 inducing points, they achieve the same performance as other models on 1k and even 10k training samples. This is not only true for the accuracy, but also confirmed by the fact that the *marginal likelihood* is equal for `EP_1k` and `Sparse150_1k`[15]. Note that this does not mean that the GP models need less samples to achieve equal performance. They still leverage the information contained in the whole training set. But they are able to reduce the 10k samples to 150 points which still represent the whole training set well. This implies that there are **only about 150 relevant clusters** in the 37-dimensional input space.

Another main **advantage of the sparse models** is the overall training time. Time complexity is reduced from $O(N^3)$ to $O(M^2N)$ where M is the number of inducing points and $M \ll N$. While `EP_1k` already took about 1h 5mins for the training of one fold, `Sparse150_1k` took less than 20 minutes. Training of a full GP model with 10k samples would have been impossible. This is not only because of long training time, but also due to the memory requirements to hold the 10k $\times$ 10k matrices. Using the sparse approximation, `Sparse150_10k` was trained in about 1h 45mins.

[15] -167 +/-14 and -169 +/-14 respectively. Details see notebook

3.6 Multiclass Classification on Synthetic Data

`GPflow` is a Gaussian Process library based on TensorFlow[16]. Because it was launched by contributors of `GPy`, the interface is still quite similar. The main features and advantages are[17]:

- Most computations are delegated to TensorFlow - strong GPU support
- The focus is on Variational Inference and MCMC sampling
- Implementation of state-of-the-art sparse[18] and approx. inference[19] algorithms
- Support for **true multiclass classification**

Notebook `07_GPFlow_multi.ipynb` contains examples of multiclass classification using Gaussian Processes. Similar to the binary classification example in section 3.2, a synthetic dataset was generated.

The first experiment operates on 1D inputs for easier visualization. It is slightly adapted from the official introduction to GPflow. The second part is an extension to 2D inputs.

3.6.1 Data Generation

For each of the $C = 6$ classes, N latent values ($\mathbf{f}^{\{c\}} \in \mathbb{R}^N$ for $c \in \{1, ..., C\}$) for inputs between 0 and 1 using a common GP. Observations $\mathbf{y}$ at inputs $\mathbf{X}$ are then generated by taking the class with highest corresponding latent value:

$$y_i = \text{argmax}_c \; f_i^{\{c\}} \;, \quad \text{for } \; i \in [0, N) \cap \mathbb{N} \tag{3.1}$$

Alternatively, the *Softmax* function could be used to calculate discrete probability-distributions for every y^i. Observations could then be sampled from a *Multinoulli* distribution. The reason for the *argmax*-approach is that GPflow supports the `RobustMax` inverse link function only for the multiclass likelihood only.

The `RobustMax` maps the C latent function values for input i to a valid probability distribution as follows:

$$\text{RobustMax}(f_i^{\{1\}}, f_i^{\{2\}}, \cdots, f_i^{\{C\}}) = [p_1, p_2, ..., p_C]^T \tag{3.2}$$

$$\text{with:} \quad p_j = \begin{cases} 1 - \epsilon & \text{argmax}_c \; f_i^{\{c\}} == j \\ \epsilon / C & \text{otherwise} \end{cases} \tag{3.3}$$

[16] https://www.tensorflow.org
[17] Cf. https://gpflow.readthedocs.io/en/develop/intro.html
[18] Titsias 2009.
[19] Hensman, Matthews, and Ghahramani 2015.

The hyperparameter ϵ allows to model inherent *noise* in the observations. I.e. how likely is it to observe a class other than the one with the highest latent value. To make sure that the data satisfies the assumption implied by this *link*, the argmax was used.

Figure 3.11 shows the 6 latent functions and the derived dataset. The training set is a random subset of those *generated* observations.

3.6.2 Prediction for 1D Inputs

Figure 3.12 shows the posterior functions and resulting predictions after the training of the GP. Test samples are also plotted for visual evaluation. The overall test accuracy is 93%. Note that at x = 0.85 the model predicts class *green* instead of *purple*. Note further, that there are only few purple training points at this position in the training set (cf figure 3.11). The reason is the hyperparameter ϵ of the *RobustMax* likelihood. ϵ defines the likelihood of observing a class other than the class with the highest latent value. It can be interpreted as a trade-off between smooth functions and a perfect fit. Therefore, ϵ should be increased if a lot of noise is expected in the observations. Otherwise overfitting in the sense of very wiggly latent functions might occur. Figure 3.13 shows the effect of decreasing ϵ by some orders of magnitude.

3.6.3 Prediction for 2D Inputs

The experiment was repeated for 2D Inputs. Data generation and training was conducted as for the 1D case. The only difference was that feature vectors $\mathbf{x}$ are now 2-dimensional with values between 0 and 1. Figure 3.14

Figure 3.15 shows the posterior functions and resulting predictions after the training of the GP in 2D. Test and training samples are also plotted for visual evaluation. The overall test accuracy is 88%. Figure 3.16 shows regions with *high confidence* only.

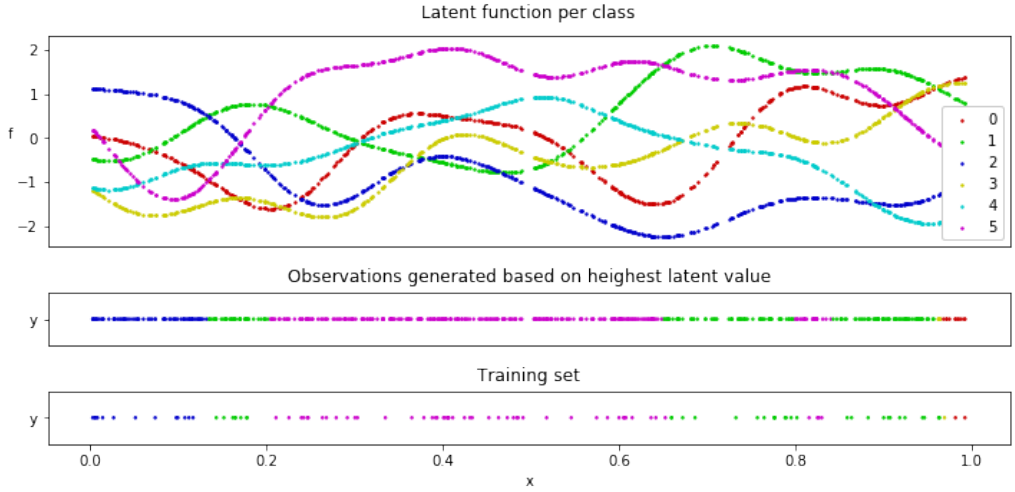

FIGURE 3.11: Generation of synthetic dataset for 6-class classification. $C = 6$, $N = 500$, Training set size = 100

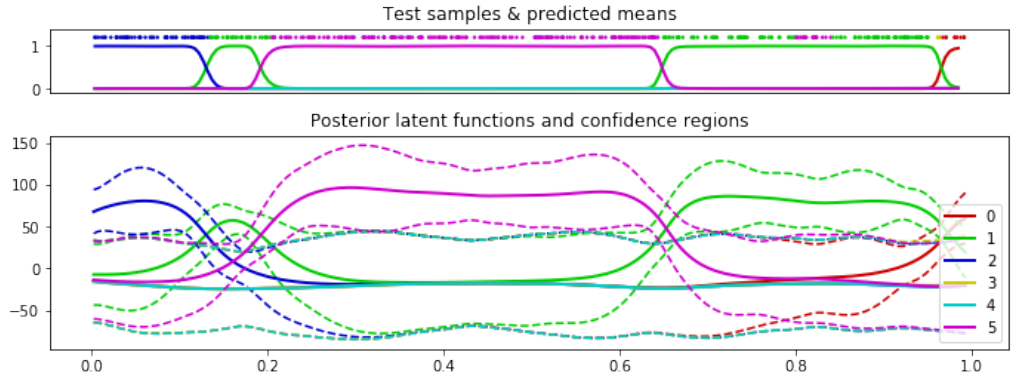

FIGURE 3.12: Posterior latent functions for *RobustMax* likelihood. Dotted lines are 95% confidence regions. Test accuracy is 93%.

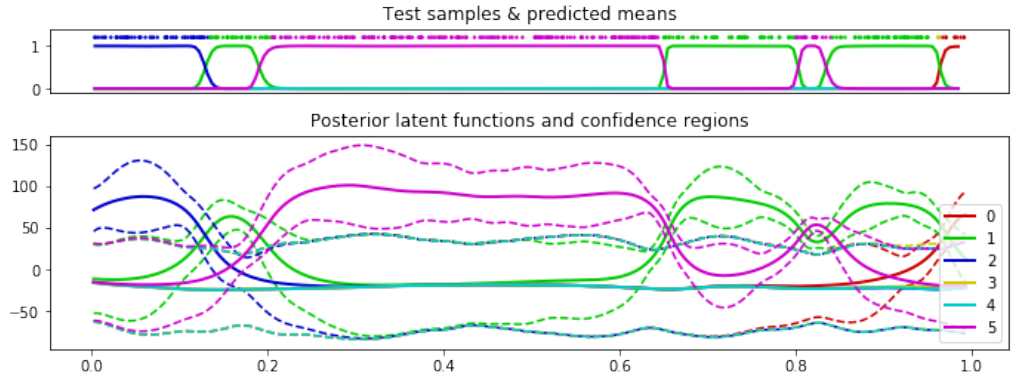

FIGURE 3.13: Posterior latent functions for *RobustMax* likelihood with a reduced ϵ value of 10^{-9} (default is 10^{-3}). Overall test accuracy is now 96%. The posterior latent functions are now less smooth to allow the better fit at $x = 0.85$.

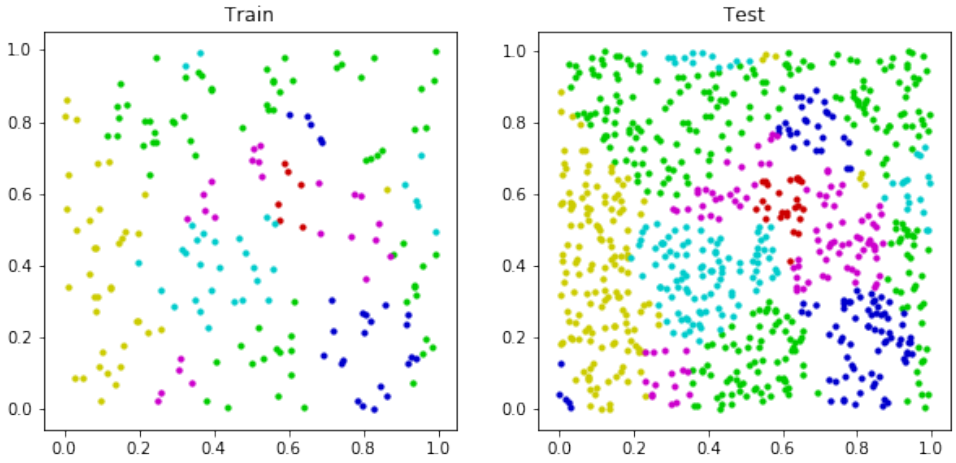

FIGURE 3.14: Generation of synthetic dataset for 6-class classification. $C = 6$, $N = 1000$, Training set size = 200

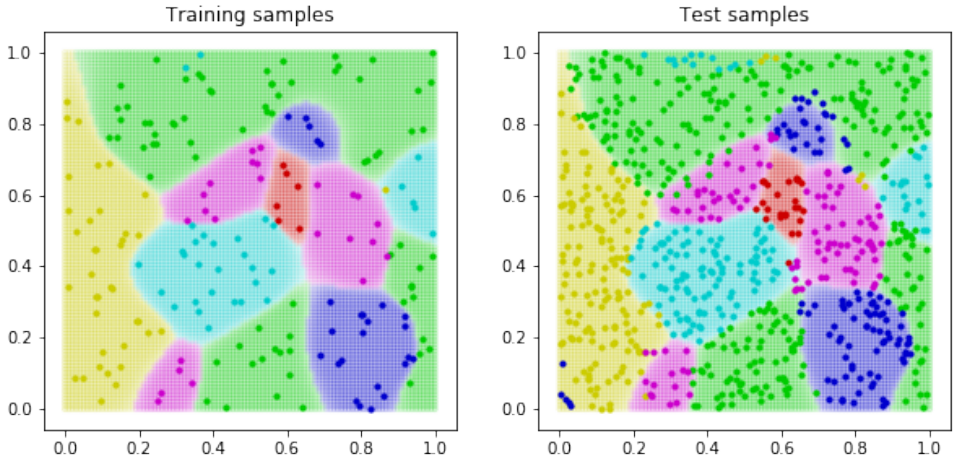

FIGURE 3.15: Posterior latent functions for *RobustMax* likelihood. Test accuracy is 88%.

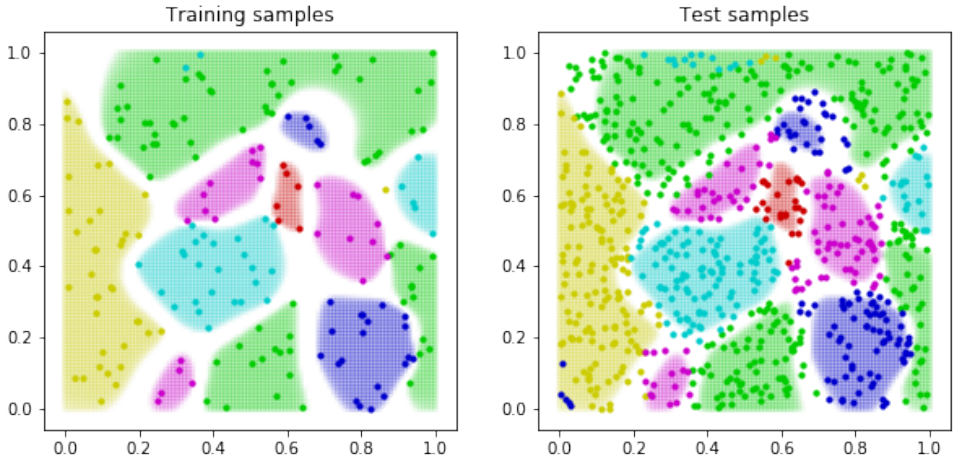

FIGURE 3.16: Posterior latent functions for *RobustMax* likelihood. White areas indicate that no class has a latent value above the threshold of 0.9. Considering only test-points with a confident prediction, the test accuracy is 96%.

3.7 Evaluation of GP and NN for Multiclass Classification

In notebooks `07_Jass_NN_multi.ipynb` and `07_Jass_GPFlow_multi.ipynb`, a systematic benchmark of GPs and deep neural networks for multiclass classification has been conducted.

3.7.1 Deep Neural Networks

Similar to section 3.5 the following neural networks were trained and evaluated using 5-fold cross-validation. All hidden layers use the *ReLu* activation. For the output layer, the *Softmax* activation and *categorical-crossentropy* loss is used. The `Jass` dataset without class *Schieben* was used. There are 6 classes in total.

- `Koller`: Baseline model by T. Koller - 3 hidden layers with 592 neurons each.
- `Linear`: No hidden layer - direct mapping from 37 input to 6 output neurons
- `MLP4`: One hidden layer with 4 neurons
- `MLP16`: One hidden layer with 16 neurons

Figure 3.17 illustrates performance for several training set sizes. The `Koller` models shows tremendous overfitting even for 80k training samples. `Linear` models achieves surprisingly good performance for small and middle sized training sets. For middle sized training sets, the 6 classes seem to be linearly separable for the most part. For the 80k datasets however, the deep `MLP16` model outperforms the linear model. The performance of the `MLP4` models is significantly worse than all others. The reason is that the 4 neuron hidden layer introduces a bottleneck in front of the 6 outputs. `MLP16` achieves highest performance and appears well suited for the largest training sets in particular.

3.7.2 Gaussian Process Models

In a first step, GPs using various kernels have been trained on a tiny subset (500 samples) of the data. The goal was to get a quick impression about the relative performance of various kernels. Figure 3.18 shows the performance of 8 common kernels.

It is interesting to observe that the validation accuracy and the log marginal likelihood may lead to different choices. While `rbf`, `rbf_ard` and `rq` perform best according to validation accuracy, the log marginal likelihood has strong preference for `linear_ard`.

For the `rbf_ard` and `linear_ard` kernel, the training set size and number of inducing points were increased to 1000 and 333 respectively. Figure 3.19 shows that the performance of the linear kernel does not scale as good as of the RBF kernel (cf. two left most models). Therefore, we focused on the `rbf_ard` kernel for further experiments. Figure 3.19 shows the performance for various setups.

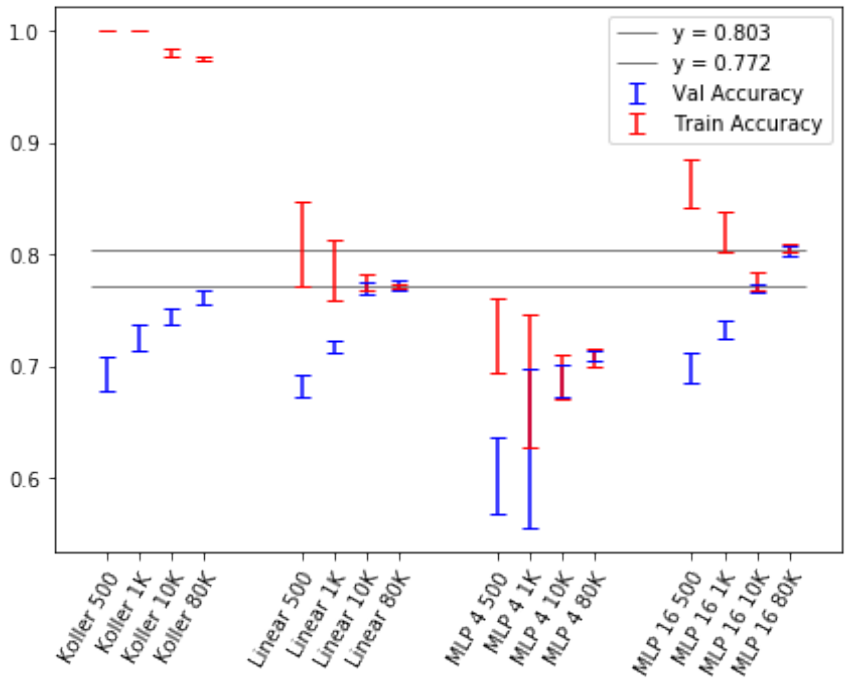

FIGURE 3.17: Training and Validation accuracy for 6-class classification on the Jass dataset. Each model is trained and evaluated on 500, 1000, 10'000 and 80'000 training samples. Error bars indicate 95% confidence intervals.

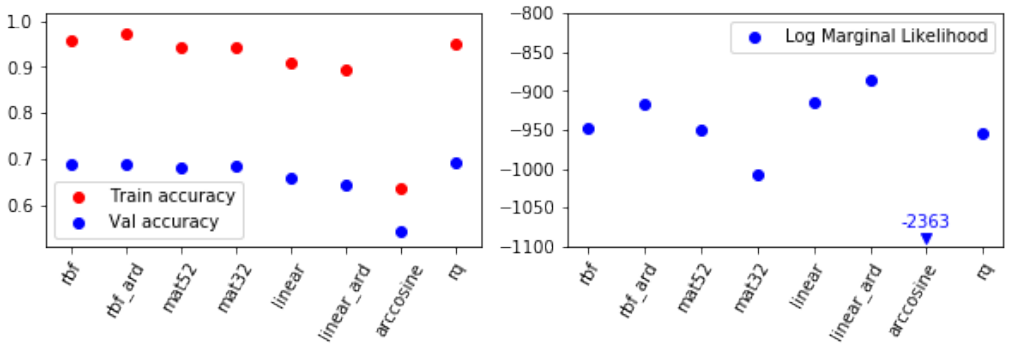

FIGURE 3.18: Performance of 8 common kernels on a tiny subset of Jass dataset. 500 training samples and a sparse GP model with 250 inducing inputs was used.

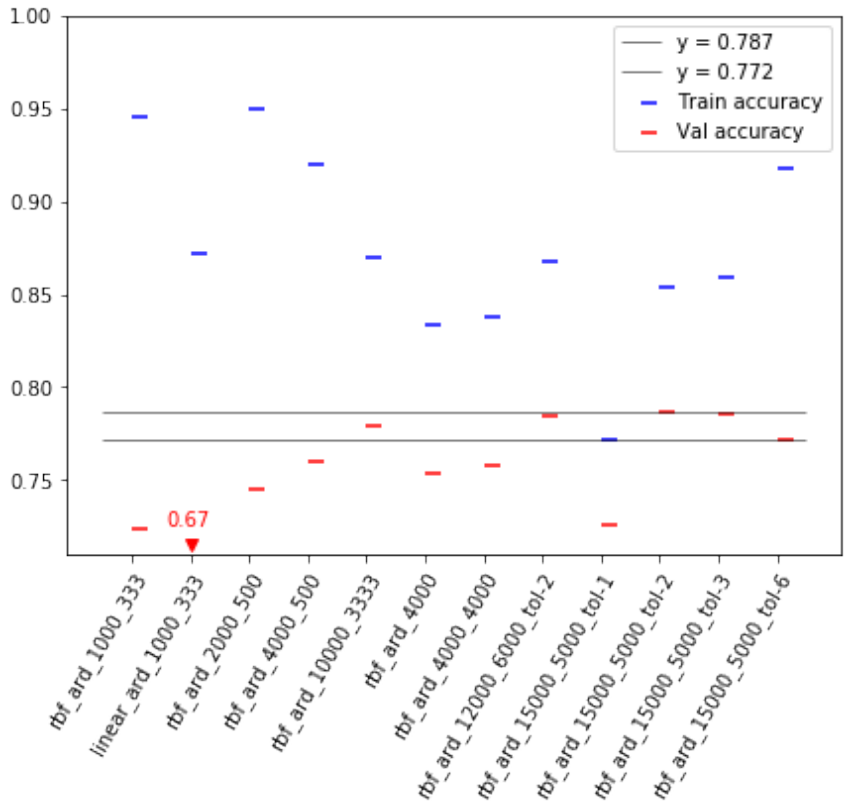

FIGURE 3.19: Train and validation set accuracy for various setups. The first part of the model name depicts the used kernel. Followed by the number of training samples and inducing points (no inducing points means trained on full training set). The suffix `_tol-6` indicates that the convergence tolerance for the log marginal likelihood is 10^{-6} (default is 10^{-10}.). $y = 0.787$ is overall best GP performance. $y = 0.772$ is best NN performance on 10k training data.

3.7.3 Conclusion

Figure 3.19 shows some interesting properties of the performance of the GP models.

First of all, note that the best performance of 78.7% validation accuracy is achieved by three GP models on 12k and 15k training samples. This outperforms the NN on 10K training samples by 1.6%. This is clearly outside the 80% (and even 95%) confidence interval of the NN performance[20]. This might be partly due to the bigger training datasets. However, even `rbf_ard_1000_3000` outperforms the NN by 0.7% validation accuracy[21].

In general, using more training samples will increase validation- and decrease training accuracy, i.e. it reduces overfitting. This can be observed for the third and fourth model from left in figure 3.19. It is even true if the number of inducing points remains the same. However, for 4000 training samples it does not matter if 500 or 4000 inducing points are used. I.e. the 4k training set seem to be represented adequately with 500 samples. This does however not hold for bigger training sets (cf. e.g. `rbf_ard_1000_3000`). This in turn implies that the 4k training samples in `rbf_ard_4000_xxx` do not represent the overall complexity of the task.

We observed that the convergence criterion[22] *tol* has a significant influence on the validation accuracy. A high value of *tol* corresponds to a form of *early-stopping*. For the models using 15k training samples, the best performance was achieved when $tol = 10^{-2}$. A smaller value did not only increase training time tremendously (time to convergence), but resulted in a worse validation accuracy. This is a clear sign of overfitting. In contrast, the marginal likelihood, as well as the training accuracy was monotonically decreasing over training time. The reason might be a model misspecification such as a too small ϵ value of the *RobustMax* likelihood. Currently, the model might favor *wiggly* functions to fit the training samples near perfectly. A next step might therefore be to increase ϵ and allow some false classifications in favor of more smooth functions. This might have positive effect on generalization and therefore validation accuracy.

[20] For the `Linear_10k` and `MPL16_10k` the validation accuracy is 76.9% with a standard deviation of 0.2%

[21] Which is still more than 1.96 standard deviations above the mean of the NN performance and therefore outside the 95% confidence interval

[22] If the increase of the log marginal likelihood relative to the last iteration is below the value of *tol* (for tolerance), optimization is stopped.

3.8 Deep Gaussian Processes

3.8.1 Source of Long Term Correlations

In a first experiment on Deep Gaussian Processes, a vanilla 3 layer Deep GP prior was implemented. A single function was then sampled through the cascade of multivariate Gaussians. Initially the 200 inputs $\mathbf{X}$ were equally distributed between 0 and 1 and the sampled points were colored linearly according to their input value.

Inputs and outputs of each layer are plotted in figure 3.20. Each point has the same color in every plot, so their *path* can be traced through the layers. Points hover on roughly three levels in the final plot. For one level ($y \approx 1.7$), horizontal and vertical lines indicate which and why points correlate.

3.8.2 Deep Gaussian Processes on Synthetic Data

Figure 3.21 shows a synthetic dataset, sampled from a 3 layer GP.

The training of a single layer GP serves as a baseline (figure 3.22). A staged optimization scheme was necessary to avoid poor local optima of hyperparameters. In

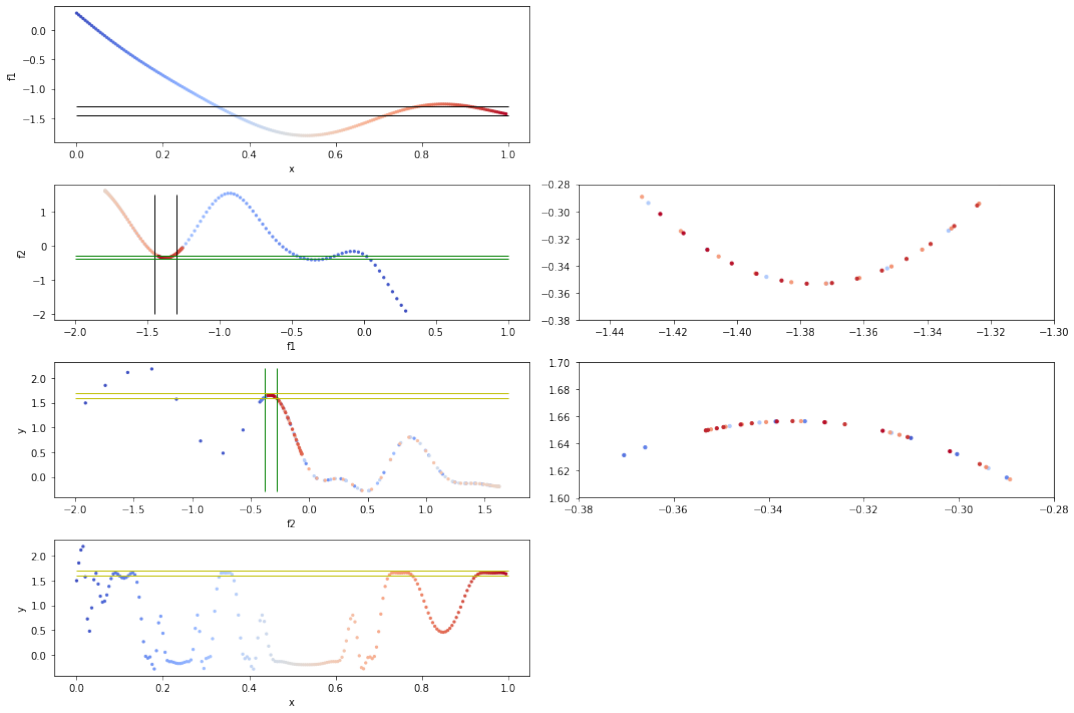

FIGURE 3.20: Left columns show scatter plots of inputs and outputs of each layer. The last row shows inputs vs outputs end-to-end. Each point has the same color in all plots. Horizontal and vertical lines serve for orientation, i.e. they map a region in the output to the corresponding region in the input of the following layer/plot. Plots in the right column are zoomed to the region bounded by the orientation lines.

a *warm-up* phase, the variance of the likelihood (cf. $\sigma^2\mathbf{I}$) was initialized and fixed to a small value. This enforces very flexible functions and a posterior mean which fits the data close to perfectly (optimal *length-scale* needs to be very small). Then, the constraint is removed and optimization continues for all hyperparameters. The marginal likelihood then automatically assures a trade-off between simple functions and a good fit. This scheme is a trivial heuristic and an alternative to *Random Restarts*.

A 3 layer GP was then trained using default values for the hyperparameters. Figure 3.24 shows the poor local optima from a naive optimization scheme (no warm-up phase). Unfortunately, the trivial staged optimization was not successful either. Training of the Deep GP turned out to be hard. After various tries with multiple stages of optimization (and different constraints on hyperparameters), the trained model explained the data quite well. The marginal likelihood did however not justify the deep model.

Finally, a model with only 2 layers was evaluated. Surprisingly, training succeeded again out of the box using the trivial scheme described above. The marginal likelihood was however worse than for 3 layers. It is unclear if the deep models perform worse only due to remaining local maxima or if the sampled data does effectively not justify a deep model.

From a practical perspective, the 3 layer GP seemed to explain the data equally well and has even tighter confidence bounds (cf. figure 3.22 vs. 3.24). In contrast, the marginal likelihood still suggests to use the single layer GP.

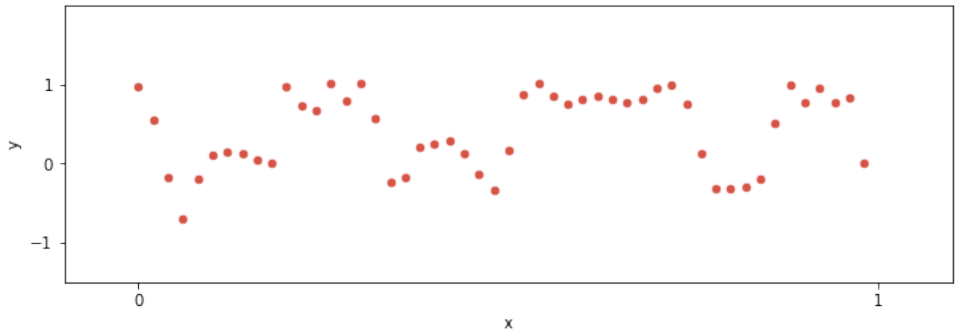

FIGURE 3.21: Synthetic dataset, sampled by a 3 layer GP with SE kernels and *length-scales* of $[0.2, 0.2, 0.5]$.

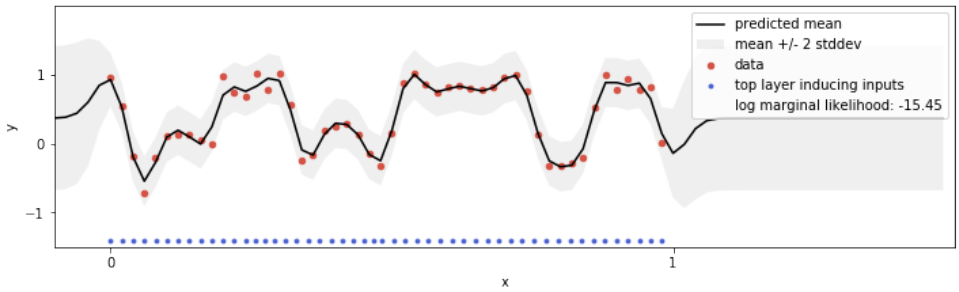

FIGURE 3.22: Single layer GP from staged optimization.

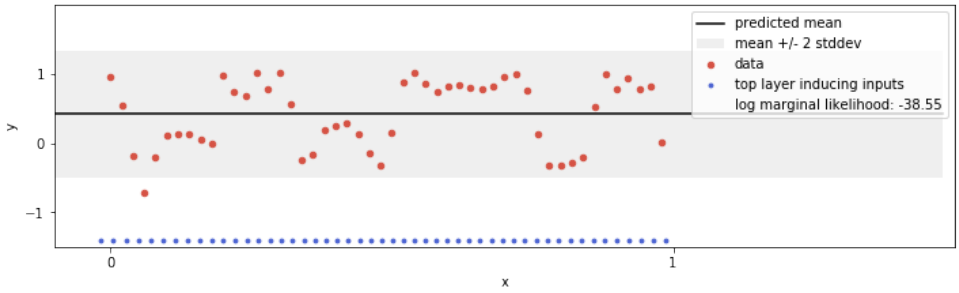

FIGURE 3.23: 3 layer GP from staged optimization. Results in poor local maximum.

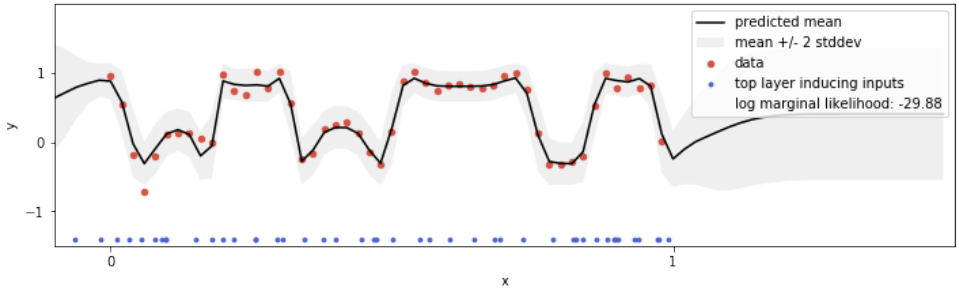

FIGURE 3.24: 3 layer GP with more sophisticated staged optimization (details cf. notebook `08_DeepGP.ipynb`). Compared to the single layer GP, the marginal likelihood does not justify the Deep GP.

3.8.3 More Data

The experiment from the previous section was repeated with more training data. The sampled data in figure 3.25 stems from the exact same Deep GP.

Surprisingly, even for the 2 and 3 layer GPs, a *naive* optimization (no warm-up phase or other staged optimization scheme) led to quite good performance. For the 3 layer GP, a staged optimization again improved performance, while the 2 layer GP achieved best performance with the naive scheme. We suppose that this behavior is very specific to the particular dataset and models used. Compared to the previous experiment, either the overall shape of the data or the higher density of training points seem to remove some local maxima.

Marginal likelihood suggests to use the 3 layer GP in this case. 2 layers outperform the single layer GP as well. The main reason might be that the single layer GP overestimates the variance for the long constant region between $x = 0.4$ and $x = 0.8$. The ability to model non-stationary (co)variance allows the Deep GPs to adapt to this region.

Visual inspection of the 2 and 3 layer GPs shows some differences in the *wiggly* region on the left side of the plots. The 3 layer GP seemed to explain the data better and also has an overall higher marginal likelihood. In this case, the subjective fit of the model and the marginal likelihood are consistent.

3.8.4 Conclusion

Training of Deep GPs revealed challenging and unstable. For 1 or 2 dimensional inputs, a staged optimization scheme can be found by visual inspections of intermediate states of the model (cf. notebook `08_DeepGP.ipynb`). This does not scale to tasks with more than 2 features however.

Another source of uncertainty is due to the variational marginalization of the latent space. The current implementation of `PyDeepGP`[23] optimizes only a lower bound of the log marginal likelihood (the ELBO). The term *marginal likelihood* in the interface of `PyDeepGP` is therefore misleading. Maximizing the ELBO assures that the KL divergence between the true and the approximate posterior is minimized. As model selection criterion however, the ELBO is not justified in theory. This might be part of the explanation for inconsistent suggestions by the *marginal likelihood* and the subjective fit of the model. The true marginal likelihood might effectively be higher than suggested by the ELBO.

`PyDeepGP` does not support non-Gaussian likelihoods yet. This prevents application to the `Jass` data. An interesting alternative is `pyro.ai`, a "probabilistic programming language"[24]. Unfortunately, time ran out for further evaluation within this work.

[23] Damianou and Lawrence 2013.
[24] https://pyro.ai/

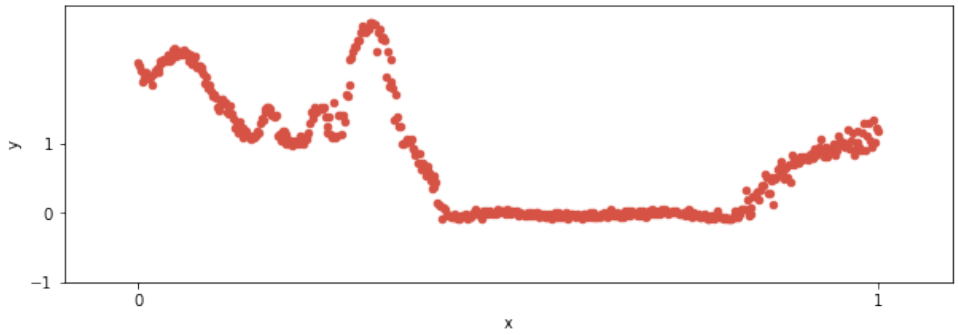

FIGURE 3.25: Synthetic dataset, sampled by a 3 layer GP with SE kernels and *length-scales* of $[0.2, 0.2, 0.5]$.

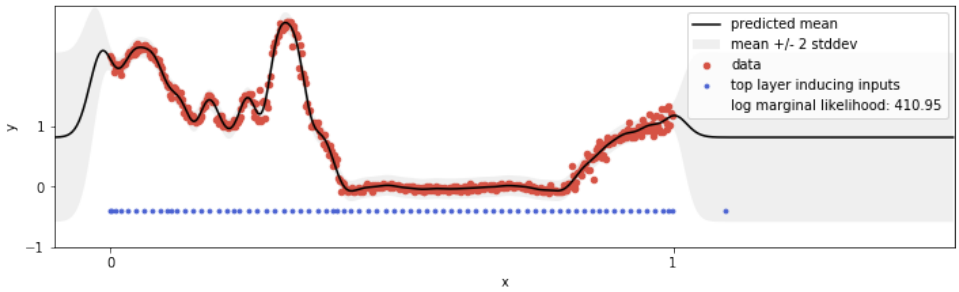

FIGURE 3.26: Single layer GP trained on synthetic data.

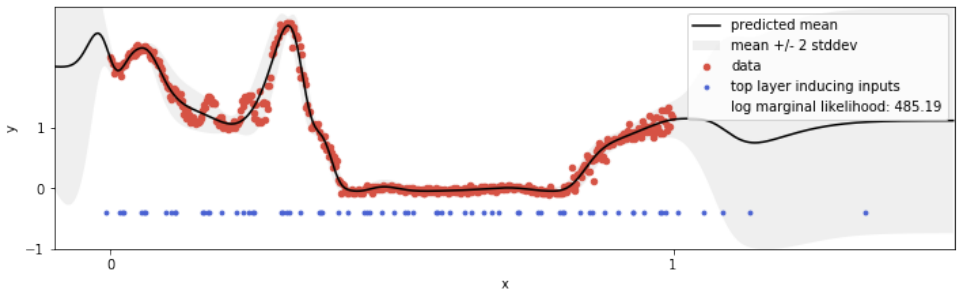

FIGURE 3.27: 2 layer GP from simultaneous optimization. Staged optimization led to lower marginal likelihood. Details cf. notebook `08_DeepGP.ipynb`

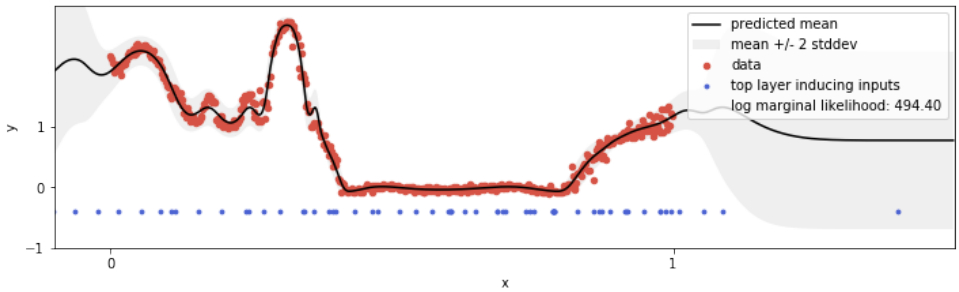

FIGURE 3.28: 3 layer GP from staged optimization optimization.

Chapter 4

Conclusion

4.1 Conclusion

In spite of - or maybe even precisely because of - their conceptual simplicity, Gaussian Processes are a powerful method for statistical inference. Prediction of similar outputs for similar inputs is an **intuitive** and **interpretable** concept. Unfortunately, rather abstract concepts and daunting necessary inference algorithms pose a quite **high entry barrier** to the field. The scattered and often specialized literature is not helpful in this context. Nevertheless, the tools and advantages provided by this approach justify this effort.

It seems negligent to claim that Gaussian Processes would be more **data efficient**. Successful inference is always a trade-off between available prior knowledge and data. To exaggerate, deep neural networks are just a way to compensate the often missing knowledge with an even greater amount of available data. GPs **require prior knowledge** in form of a suitable kernel. Though, general purpose kernels such as the *squared exponential* do surprisingly well on a wide range of tasks. Furthermore, sparse approximations allow conclusions about the amount of relevant clusters in the data and anisotropic kernels implement **automatic feature selection**.

Similarly, Gaussian Processes are not less prone to **overfitting** by design. However, the ability to perform *second-level inference* based on the **marginal likelihood** clearly distinguishes GPs from methods based on pure parameter optimization. On a finite dataset, every decision based on optimization includes the risk of overfitting. Marginalization of most parameters reduces the number of required decisions drastically. The ability to optimize remaining hyperparameters is another major advantage over naive grid-search. Furthermore, an **inherent regularization** is provided through the marginal likelihood. In practice, even some of the hyperparameters can be marginalized out by the use of *hyperpriors* and variational inference.

The proof that GPs correspond to infinitely wide (single hidden layer) neural networks raised awareness for their potential. The implied, or at least anticipated, consequences of this discovery were however **doubted early**. This is probably best reflected by the quote from MacKay in the beginning of this work. He argued that the

task of **feature discovery** is **not** the one Gaussian Processes solve. This finally led to research on Deep Gaussian Processes, which is an ongoing research topic to date. Unchanged by this discussion is the fact that many tasks are solved surprisingly well by *simple smoothing devices* in the form of GPs.

Practical application of GPs is usually limited by the size of the dataset and intractable posteriors. Both constraints have been mitigated drastically by recent advances in research. In my opinion, the **missing awareness** or interest in GPs might be at least an equivalent cause for the lack of broad practical application today.

Based on results and insights obtained in this work, **it is recommended to further investigate** in practical application of Gaussian Processes and Bayesian modeling.

4.2 Retrospective

A substantial part of this work dealt with fundamentals behind Gaussian Processes. Principles of **Bayesian modeling**, **probability theory** and various algorithms for **approximate inference** revealed crucial to evaluate GPs within the overall field of Machine Learning. The insights during literature research also led to a more extensive understanding of this field in general.

The amount of work spent on kernels - the key ingredients of GPs - may not reflect their practical importance sufficiently. In practice, even naive **combinations of kernels** seem to have a positive effect on the performance. In general, maybe affected by *typical* research in the Bayesian community, this work mostly focused on qualitative statements and experiments. Additional **practical advice** or efforts on **fine-tuning measures** might have been useful for future application on practical tasks.

4.3 Recommendations For Future Work

Hensman, Matthews, and Ghahramani 2015 claim to scale Gaussian Process Classification to **millions of training samples**. Practical feasibility and the remaining effect of computational resources on the performance should be examined in more detail on custom tasks.

The application of **Deep GPs** to the `Jass` or other custom tasks is an obvious next step. Experiments showed that local optima constitute a challenge and a suitable custom optimization scheme may be crucial. In this case, heuristics for high dimensional inputs remain to be found.

Gaussian Processes Latent Variable Models (GP-LVM cf. page 51) are an interesting application of GPs as generative models. For example, they are successful in non-linear dimensionality reduction. Lawrence 2004 demonstrates their ability to separate MNIST digits 0-4 in 2D latent space. This approach might be further used to impute missing values in partially corrupted datasets.

From a more technical perspective, `Pyro`[1] appears as an interesting framework to explore. It claims to be a **universal probabilistic programming language** and to unify the best of modern deep learning and Bayesian modeling[2].

Experiment 3.6 revealed some gaps in the support for multiclass classification in `GPy`. It might be useful to investigate in the implementation of a **multinoulli likelihood** with softmax link function. Performance on synthetic and real datasets should then be evaluated w.r.t. the existing (rather naive) `RobustMax` implementation.

[1] https://pyro.ai/
[2] https://eng.uber.com/pyro/

Appendix A

Derivations

A.1 Predictive Distribution for Gaussian Process Regression

A.1.1 The Conditional

This section shows how the full conditional in eq (2.42) can be rearranged into the form of an unnormalized Gaussian (cf. eq. (2.38)):

Constants w.r.t $\mathbf{f}_*$ are continuously collected in the variable c_c (with subscript $_c$ for **c**onditional). This is indicated by blue slashed terms.

Starting with (2.42), separate all constants w.r.t. $\mathbf{f}_*$ and use the exponential identity $e^a / e^b = e^{a-b} = e^a e^{-b}$. Then collect constants in c_c.

$$
\begin{aligned}
&= \frac{\frac{1}{\sqrt{(2\pi)^{N+M}|\boldsymbol{\Sigma}_j|}}}{\frac{1}{\sqrt{(2\pi)^{N}|\boldsymbol{\Sigma}_m|}}} \exp\left(- \frac{\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}^T \boldsymbol{\Sigma}_j^{-1} \begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}}{2} \right) \exp\left(+ \frac{\begin{bmatrix}\mathbf{f}\end{bmatrix}^T \boldsymbol{\Sigma}_m^{-1} \begin{bmatrix}\mathbf{f}\end{bmatrix}}{2} \right) \\
&= c_c \exp\left(- \frac{\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}^T \boldsymbol{\Sigma}_j^{-1} \begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix}}{2} \right)
\end{aligned}
\tag{A.1}
$$

For the next steps consider only the numerator in the exponent of (A.1).

The inverse of $\boldsymbol{\Sigma}_j$ is derived as follows. $\boldsymbol{\Sigma}_j$ can be written as a partitioned matrix with $\boldsymbol{K}, \boldsymbol{K}_*, \boldsymbol{K}_*^T$ and $\boldsymbol{K}_{**}$ being the submatrices.

$$
\boldsymbol{\Sigma}_j = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix} \tag{A.2}
$$

The inverse of this partitioned matrix is then given by:

$$\boldsymbol{\Sigma}_j^{-1} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{K}_{**} \end{bmatrix}^{-1} = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \tag{A.3}$$

where:[1]

$$\begin{aligned} \mathbf{A} &= \boldsymbol{K}^{-1} + \boldsymbol{K}^{-1}\boldsymbol{K}_*(\boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*)^{-1}\boldsymbol{K}_*^T\boldsymbol{K}^{-1} \\ \mathbf{B} &= -\boldsymbol{K}^{-1}\boldsymbol{K}_*(\boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*)^{-1} \\ \mathbf{B}^T &= -(\boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*)^{-1}\boldsymbol{K}_*^T\boldsymbol{K}^{-1} \\ \mathbf{C} &= (\boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*)^{-1} \end{aligned} \tag{A.4}$$

Substitute into the numerator of (A.1) and multiply out

$$\begin{aligned} \begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix}^T \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} &= \begin{bmatrix} \mathbf{f}^T\mathbf{A} + {\mathbf{f}_*}^T\mathbf{B}^T & \mathbf{f}^T\mathbf{B} + {\mathbf{f}_*}^T\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{f}^T\mathbf{A}\mathbf{f} + {\mathbf{f}_*}^T\mathbf{B}^T\mathbf{f} + \mathbf{f}^T\mathbf{B}\mathbf{f}_* + {\mathbf{f}_*}^T\mathbf{C}\mathbf{f}_* \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{f}^T\mathbf{A}\mathbf{f} + 2{\mathbf{f}_*}^T\mathbf{B}^T\mathbf{f} + {\mathbf{f}_*}^T\mathbf{C}\mathbf{f}_* \end{bmatrix} \end{aligned} \tag{A.5}$$

This is a quadratic expression in $\mathbf{f}_*$. Using a technique called *completing the square* any quadratic expression $ax^2 + bx + c$ can be reformulated to $a\left(x + \frac{b}{2a}\right)^2 - \frac{b^2}{4a} + c$.

Proof:

$$\begin{aligned} a\left(x + \frac{b}{2a}\right)^2 - \frac{b^2}{4a} + c &= a\left(x^2 + x\frac{2b}{2a} + \frac{b^2}{4a^2}\right) - \frac{b^2}{4a} + c \\ &= a\left(x^2 + x\frac{b}{a} + \frac{b^2}{4a^2}\right) - \frac{b^2}{4a} + c \\ &= ax^2 + \cancel{a}x\frac{b}{\cancel{a}} + \cancel{a}\frac{b^2}{4\cancel{a}^2} - \frac{b^2}{4a} + c \\ &= ax^2 + bx + \cancel{\frac{b^2}{4a}} - \cancel{\frac{b^2}{4a}} + c \\ &= ax^2 + bx + c \end{aligned} \tag{A.6}$$

[1] Cf. https://en.wikipedia.org/wiki/Block_matrix#Block_matrix_inversion

Completing the square using vectors $\mathbf{x}, \mathbf{b} \in \mathbb{R}^d$ and a symmetric invertible matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ and scalar constant c works similar:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{x} + c = (\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}) - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} + c \tag{A.7}$$

Proof:

$$(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}) - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} + c \tag{A.8}$$

$$= \mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\cancel{\mathbf{A}^{-1}\mathbf{A}}\mathbf{x} + \mathbf{b}^T\mathbf{A}^{-1}\cancel{\mathbf{A}\mathbf{A}^{-1}}\mathbf{b} - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} + c \tag{A.9}$$

$$= \mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{x} + \cancel{\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}} + c \tag{A.10}$$

$$= \mathbf{x}^T\mathbf{A}\mathbf{x} - 2\mathbf{b}^T\mathbf{x} + c \tag{A.11}$$

Completing the square in (A.5) with $\mathbf{x} = \mathbf{f}_*, \mathbf{A} = \mathbf{C}, \mathbf{b} = -B^T\mathbf{f}, c = \mathbf{f}^T A\mathbf{f}$:

$$= (\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}) - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} + c$$

Substituting back into the numerator of (A.1), separate and collect constants in c_c:

$$P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}_*, \mathbf{X}) = c_c \exp\left(-\frac{(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}) - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b} + c}{2}\right) \tag{A.12}$$

$$= c_c \exp\left(-\frac{(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})}{2} + \frac{\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}}{2} - \frac{c}{2}\right) \tag{A.13}$$

$$= c_c \exp\left(-\frac{(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x} - \mathbf{A}^{-1}\mathbf{b})}{2}\right) \tag{A.14}$$

Substituting back in for $\mathbf{x}$, $\mathbf{A}$, $\mathbf{b}$ and c

$$= c_c \exp\left(-\frac{(\mathbf{f}_* + \mathbf{C}^{-1}\mathbf{B}^T\mathbf{f})^T\mathbf{C}(\mathbf{f}_* + \mathbf{C}^{-1}\mathbf{B}^T\mathbf{f})}{2}\right) \tag{A.15}$$

Finally, an unnormalized Gaussian probability density with $\boldsymbol{\mu}_c = -\mathbf{C}^{-1}\mathbf{B}^T\mathbf{f}$ and $\boldsymbol{\Sigma}_c = \mathbf{C}^{-1}$ is revealed. Substituting back the covariance matrices from (A.4) we have:

$$\boldsymbol{\mu}_c = -\cancel{(\boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*)}\cancel{(\boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*)^{-1}}\boldsymbol{K}_*^T\boldsymbol{K}^{-1}\mathbf{f}$$

$$= \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\mathbf{f} \tag{A.16}$$

$$\boldsymbol{\Sigma}_c = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_* \tag{A.17}$$

We have shown that the conditional probability density $P(\mathbf{f}_*|\mathbf{f}, \mathbf{X}_*, \mathbf{X})$ is proportional to a Gaussian probability density with $\boldsymbol{\mu}_c = \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\mathbf{f}$ and $\boldsymbol{\Sigma}_c = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T\boldsymbol{K}^{-1}\boldsymbol{K}_*$.

A.1.2 The Posterior

Combine exponents, factor out the numerator and collect the constant terms in c_p:

$$
\begin{aligned}
&= c_p * \exp\left(-\frac{(\mathbf{f}-\mathbf{y})^T(\sigma^2\mathbf{I})^{-1}(\mathbf{f}-\mathbf{y})}{2} - \frac{(\mathbf{f}-\cancel{\mathbf{0}})^T\mathbf{K}^{-1}(\mathbf{f}-\cancel{\mathbf{0}})}{2} \right) \\
&= c_p * \exp\left(-\frac{(\mathbf{f}-\mathbf{y})^T(\sigma^2\mathbf{I})^{-1}(\mathbf{f}-\mathbf{y}) + \mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}}{2} \right) \\
&= c_p * \exp\left(-\frac{\mathbf{f}^T(\sigma^2\mathbf{I})^{-1}\mathbf{f} - 2\mathbf{f}^T(\sigma^2\mathbf{I})^{-1}\mathbf{y} + \cancel{\mathbf{y}^T(\sigma^2\mathbf{I})^{-1}\mathbf{y}} + \mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}}{2} \right) \\
&= c_p * \exp\left(-\frac{\mathbf{f}^T(\sigma^2\mathbf{I})^{-1}\mathbf{f} - 2\mathbf{f}^T(\sigma^2\mathbf{I})^{-1}\mathbf{y} + \mathbf{f}^T\mathbf{K}^{-1}\mathbf{f}}{2} \right)
\end{aligned}
$$

factor out $\mathbf{f}^T\mathbf{f}$

$$
= c_p * \exp\left(-\frac{\mathbf{f}^T(\mathbf{K}^{-1} + (\sigma^2\mathbf{I})^{-1})\mathbf{f} - 2\mathbf{f}^T(\sigma^2\mathbf{I})^{-1}\mathbf{y}}{2} \right)
$$

Again a quadratic expression in $\mathbf{f}$ in the numerator. Therefore completing the square with $\mathbf{x} = \mathbf{f}, \mathbf{A} = (\mathbf{K}^{-1} + (\sigma^2\mathbf{I})^{-1}), \mathbf{b} = (\sigma^2\mathbf{I})^{-1}\mathbf{y}$, and $c = 0$

$$
= c_p * \exp\left(-\frac{(\mathbf{x}-\mathbf{A}^{-1}\mathbf{b})^T\mathbf{A}(\mathbf{x}-\mathbf{A}^{-1}\mathbf{b}) - \cancel{\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}} + \cancel{c}}{2} \right) \tag{A.18}
$$

We already recognize the posterior as an unnormalized Gaussian PDF in $\mathbf{x}$ (i.e. in $\mathbf{f}$). Before we substitute back in to get $\boldsymbol{\mu}_p$ and $\boldsymbol{\Sigma}_p$ lets rewrite $\mathbf{A}^{-1}$. Using the following identity for two invertible square matrices $\mathbf{U}, \mathbf{V}$:

$$
\begin{aligned}
(\mathbf{U}^{-1} + \mathbf{V}^{-1})^{-1} &= (\mathbf{V}^{-1}\mathbf{V}\mathbf{U}^{-1} + \mathbf{V}^{-1}\mathbf{U}\mathbf{U}^{-1})^{-1} \\
&= (\mathbf{V}^{-1}(\mathbf{U} + \mathbf{V})\mathbf{U}^{-1})^{-1} \\
&= \mathbf{U}(\mathbf{U} + \mathbf{V})^{-1}\mathbf{V}
\end{aligned}
$$

applied to $\mathbf{A}^{-1}$:

$$
\begin{aligned}
\mathbf{A}^{-1} &= (\mathbf{K}^{-1} + (\sigma^2\mathbf{I})^{-1})^{-1} \\
&= \mathbf{K}(\mathbf{K} + (\sigma^2\mathbf{I}))^{-1}(\sigma^2\mathbf{I})
\end{aligned}
$$

Now substituting back in for $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{x}$ in eq. A.18.

$$= c_p * \exp\left(- \frac{\left(\mathbf{f} - \boldsymbol{K}(\boldsymbol{K} + (\sigma^2\mathbf{I}))^{-1}(\sigma^2\mathbf{I})(\sigma^2\mathbf{I})^{-1}\mathbf{y}\right)^T \mathbf{A} \left(\mathbf{f} - \boldsymbol{K}(\boldsymbol{K} + (\sigma^2\mathbf{I}))^{-1}(\sigma^2\mathbf{I})(\sigma^2\mathbf{I})^{-1}\mathbf{y}\right)}{2} \right)$$

$$= c_p * \exp\left(- \frac{\left(\mathbf{f} - \boldsymbol{K}(\boldsymbol{K} + (\sigma^2\mathbf{I}))^{-1}\mathbf{y}\right)^T (\boldsymbol{K}^{-1} + (\sigma^2\mathbf{I})^{-1}) \left(\mathbf{f} - \boldsymbol{K}(\boldsymbol{K} + (\sigma^2\mathbf{I}))^{-1}\mathbf{y}\right)}{2} \right)$$

Which is proportional to a Gaussian PDF with:

$$\boldsymbol{\mu}_p = \boldsymbol{K}(\boldsymbol{K} + (\sigma^2\mathbf{I}))^{-1}\mathbf{y}$$
$$\boldsymbol{\Sigma}_p = ((\sigma^2\mathbf{I})^{-1} + \boldsymbol{K}^{-1})^{-1}$$

A.1.3 The Predictive Distribution

$$= c_{pred} \int_{\mathbf{f}} \exp\left(\frac{\left((\mathbf{f}_* - \hat{\boldsymbol{\mu}}_c\mathbf{f})^T\boldsymbol{\Sigma}_c^{-1}(\mathbf{f}_* - \hat{\boldsymbol{\mu}}_c\mathbf{f})\right)}{2}\right) \exp\left(-\frac{\left((\mathbf{f} - \boldsymbol{\mu}_p)^T\boldsymbol{\Sigma}_p^{-1}(\mathbf{f} - \boldsymbol{\mu}_p)\right)}{-2}\right)$$

Combine exponents and multiply out numerators

$$= c_{pred} \int_{\mathbf{f}} \exp\left(\frac{\left((\mathbf{f}_* - \hat{\boldsymbol{\mu}}_c\mathbf{f})^T\boldsymbol{\Sigma}_c^{-1}(\mathbf{f}_* - \hat{\boldsymbol{\mu}}_c\mathbf{f})\right) + \left((\mathbf{f} - \boldsymbol{\mu}_p)^T\boldsymbol{\Sigma}_p^{-1}(\mathbf{f} - \boldsymbol{\mu}_p)\right)}{-2}\right)$$

$$= c_{pred} \int_{\mathbf{f}} \exp\left(\frac{\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\mathbf{f}_* - 2\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c\mathbf{f} + \mathbf{f}^T\hat{\boldsymbol{\mu}}_c^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c\mathbf{f} + \mathbf{f}^T\boldsymbol{\Sigma}_p^{-1}\mathbf{f} - 2\boldsymbol{\mu}_p^T\boldsymbol{\Sigma}_p^{-1}\mathbf{f} + \boldsymbol{\mu}_p^T\boldsymbol{\Sigma}_p^{-1}\boldsymbol{\mu}_p}{-2}\right)$$

Collect constants w.r.t. $\mathbf{f}$ and $\mathbf{f}_*$ in c_{pred}. Move constants w.r.t. $\mathbf{f}$ outside the integral.

$$= c_{pred} \exp\left(\frac{\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\mathbf{f}_*}{-2}\right) \int_{\mathbf{f}} \exp\left(\frac{-2\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c\mathbf{f} + \mathbf{f}^T\hat{\boldsymbol{\mu}}_c^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c\mathbf{f} + \mathbf{f}^T\boldsymbol{\Sigma}_p^{-1}\mathbf{f} - 2\boldsymbol{\mu}_p^T\boldsymbol{\Sigma}_p^{-1}\mathbf{f}}{-2}\right)$$

Factor out $\mathbf{f}^T\mathbf{f}$ from the two middle summands and $-2\mathbf{f}$ from the left- and rightmost summand inside the second exponent.

$$= c_{pred} \exp\left(\frac{\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\mathbf{f}_*}{-2}\right) \int_{\mathbf{f}} \exp\left(\frac{\mathbf{f}^T(\hat{\boldsymbol{\mu}}_c^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c + \boldsymbol{\Sigma}_p^{-1})\mathbf{f} - 2(\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c + \boldsymbol{\mu}_p^T\boldsymbol{\Sigma}_p^{-1})\mathbf{f}}{-2}\right)$$

Now completing the square using: $\mathbf{x} = \mathbf{f}, \mathbf{A} = (\hat{\boldsymbol{\mu}}_c^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c + \boldsymbol{\Sigma}_p^{-1}), \mathbf{b} = (\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\hat{\boldsymbol{\mu}}_c + \boldsymbol{\mu}_p^T\boldsymbol{\Sigma}_p^{-1})^T$ and $c = 0$:

$$= c_{pred} \exp\left(\frac{\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\mathbf{f}_*}{-2}\right) \int_{\mathbf{f}} \exp\left(\frac{\left(\mathbf{f} - \mathbf{A}^{-1}\mathbf{b}\right)^T\mathbf{A}\left(\mathbf{f} - \mathbf{A}^{-1}\mathbf{b}\right) - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}}{-2}\right)$$

because neither $\mathbf{A}$ nor $\mathbf{b}$ depend on $\mathbf{f}$ we can move the last summand in the second exponent outside of the integral. However, it cannot be collected in c_{pred} because $\mathbf{b}$ still contains $\mathbf{f}_*$

$$= c_{pred} \exp\left(\frac{\mathbf{f}_*{}^T\boldsymbol{\Sigma}_c^{-1}\mathbf{f}_*}{-2}\right) \exp\left(\frac{\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}}{2}\right) \int_{\mathbf{f}} \exp\left(\frac{\left(\mathbf{f} - \mathbf{A}^{-1}\mathbf{b}\right)^T\mathbf{A}\left(\mathbf{f} - \mathbf{A}^{-1}\mathbf{b}\right)}{-2}\right)$$

A.1.4 The Predictive Distribution - Simplifications

Starting from equation (2.56):

$$= c_{pred} \exp\left(\frac{\mathbf{f_*}^T \boldsymbol{\Sigma}_c^{-1} \mathbf{f_*}}{-2}\right) \exp\left(\frac{\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}}{2}\right) \tag{A.19}$$

First lets substitute in for $\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}$ and simplify:

$$\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b} = (\mathbf{f_*}^T \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c + \boldsymbol{\mu}_p^T \boldsymbol{\Sigma}_p^{-1})(\hat{\boldsymbol{\mu}}_c^T \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c + \boldsymbol{\Sigma}_p^{-1})^{-1} (\mathbf{f_*}^T \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c + \boldsymbol{\mu}_p^T \boldsymbol{\Sigma}_p^{-1})^T$$

For simplicity let:

$$\begin{aligned}
\mathbf{u} &= \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c \\
\mathbf{v} &= \boldsymbol{\mu}_p^T \boldsymbol{\Sigma}_p^{-1} \\
\mathbf{W} &= (\hat{\boldsymbol{\mu}}_c^T \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c + \boldsymbol{\Sigma}_p^{-1})^{-1} = (\hat{\boldsymbol{\mu}}_c^T \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c)^{-1} ((\hat{\boldsymbol{\mu}}_c^T \boldsymbol{\Sigma}_c^{-1} \hat{\boldsymbol{\mu}}_c)^{-1} + \boldsymbol{\Sigma}_p)^{-1} \boldsymbol{\Sigma}_p
\end{aligned}$$

Therefore:

$$\begin{aligned}
\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b} &= (\mathbf{f_*}^T \mathbf{u} + \mathbf{v}) \mathbf{W} (\mathbf{f_*}^T \mathbf{u} + \mathbf{v})^T \\
&= \mathbf{f_*}^T \mathbf{u} \mathbf{W} (\mathbf{f_*}^T \mathbf{u})^T + 2 \mathbf{f_*}^T \mathbf{u} \mathbf{W} \mathbf{v}^T + \cancel{\mathbf{v} \mathbf{W} \mathbf{v}^T} \\
&= \mathbf{f_*}^T \mathbf{u} \mathbf{W} \mathbf{u}^T \mathbf{f_*} + 2 \mathbf{f_*}^T \mathbf{u} \mathbf{W} \mathbf{v}^T
\end{aligned}$$

Substituting back for $\mathbf{b}^T \mathbf{A}^{-1} \mathbf{b}$ in (A.19):

$$\begin{aligned}
&= c_{pred} \exp\left(\frac{\mathbf{f_*}^T \boldsymbol{\Sigma}_c^{-1} \mathbf{f_*} - \mathbf{f_*}^T \mathbf{u} \mathbf{W} \mathbf{u}^T \mathbf{f_*} - 2 \mathbf{f_*}^T \mathbf{u} \mathbf{W} \mathbf{v}^T}{-2}\right) \\
&= c_{pred} \exp\left(\frac{\mathbf{f_*}^T (\boldsymbol{\Sigma}_c^{-1} - \mathbf{u} \mathbf{W} \mathbf{u}^T) \mathbf{f_*} - 2 \mathbf{f_*}^T \mathbf{u} \mathbf{W} \mathbf{v}^T}{-2}\right)
\end{aligned}$$

Completing the square with $\mathbf{A} = (\boldsymbol{\Sigma}_c^{-1} - \mathbf{u} \mathbf{W} \mathbf{u}^T)$ and $\mathbf{b} = \mathbf{u} \mathbf{W} \mathbf{v}^T$:

$$= c_{pred} \exp\left(\frac{(\mathbf{f_*} - \mathbf{A}^{-1} \mathbf{b})^T \mathbf{A} (\mathbf{f_*} - \mathbf{A}^{-1} \mathbf{b}) - \cancel{(\cdots)}}{-2}\right)$$

Reveals the final expression of the predictive as proportional to a Gaussian PDF in $\mathbf{f_*}$ with:

$$\begin{aligned}
\boldsymbol{\mu}_{pred} = \mathbf{A}^{-1} \mathbf{b} &= (\boldsymbol{\Sigma}_c^{-1} - \mathbf{u} \mathbf{W} \mathbf{u}^T)^{-1} \mathbf{u} \mathbf{W} \mathbf{v}^T \\
\boldsymbol{\Sigma}_{pred} = \mathbf{A}^{-1} &= (\boldsymbol{\Sigma}_c^{-1} - \mathbf{u} \mathbf{W} \mathbf{u}^T)^{-1}
\end{aligned}$$

After substituting back the original derived expressions for $\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p, \boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ and simplifications (using the *Sherman-Morrison-Woodburry* formula), this reduces to:

$$\boldsymbol{\mu}_{pred} = \boldsymbol{K}_*^T(\boldsymbol{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$$
$$\boldsymbol{\Sigma}_{pred} = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T(\boldsymbol{K} + \sigma^2\mathbf{I})^{-1}\boldsymbol{K}_*$$

Bibliography

Bandyopadhyay, Prasanta S. and Malcolm R. Forster (May 2011). *Philosophy of Statistics (Handbook of the Philosophy of Science 7)*. North Holland. ISBN: 0444518622. URL: https://www.xarg.org/ref/a/B0055SBULA/.

Bishop, Christopher M. (Apr. 2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer. ISBN: 9780387310732. URL: https://www.xarg.org/ref/a/0387310738/.

Bromiley, P. A. (June 2018). *Products and Convolutions of Gaussian Probability Density Functions*. http://www.tina-vision.net/docs/memos/2003-003.pdf.

Cressie, Noel A. C. (Sept. 1993). *Statistics for Spatial Data*. John Wiley & Sons, Inc. DOI: 10.1002/9781119115151.

Damianou, Andreas C. and Neil Lawrence (Apr. 2013). "Deep Gaussian Processes". In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Carlos M. Carvalho and Pradeep Ravikumar. Vol. 31. Proceedings of Machine Learning Research. Scottsdale, Arizona, USA: PMLR, pp. 207–215. URL: http://proceedings.mlr.press/v31/damianou13a.html.

Damianou, Andreas C., Michalis K. Titsias, and Neil Lawrence (2011). "Variational Gaussian Process Dynamical Systems". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., pp. 2510–2518. ISBN: 978-1-61839-599-3. URL: http://dl.acm.org/citation.cfm?id=2986459.2986739.

DeltaIV (n.d.). *Do Gaussian process (regression) have the universal approximation property?* Cross Validated. URL:https://stats.stackexchange.com/q/268884 (version: 2017-03-21). eprint: https://stats.stackexchange.com/q/268884. URL: https://stats.stackexchange.com/q/268884.

Duvenaud, David (2014). *The Kernel Cookbook*. URL: http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm (visited on 11/21/2019).

Fong, Edwin and Chris Holmes (May 21, 2019). "On the marginal likelihood and cross-validation". In: *To appear in Biometrika*. arXiv: http://arxiv.org/abs/1905.08737v2 [stat.ME].

Ghahramani, Zoubin (2011). *A Tutorial on Gaussian Processes (or why I don't use SVMs)*. https://mlss2011.comp.nus.edu.sg/uploads/Site/lect1gp.pdf.

Hassouna, H (Feb. 2016). "Gaussian Process for Image ClassiÖcation". PhD thesis. Peopleís Democratic Republic of Algeria. URL: http://thesis.univ-biskra.dz/2607/1/Th%C3%A8se_lmd_51_2016.pdf.

Hensman, James, Alexander Matthews, and Zoubin Ghahramani (Sept. 2015). "Scalable Variational Gaussian Process Classification". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, pp. 351–360. URL: http://proceedings.mlr.press/v38/hensman15.html.

Hinton, Geoffrey E. and Drew van Camp (1993). "Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights". In: *Proceedings of the Sixth Annual Conference on Computational Learning Theory*. COLT '93. Santa Cruz, California, USA: ACM, pp. 5–13. ISBN: 0-89791-611-5. DOI: 10.1145/168304.168306. URL: http://doi.acm.org/10.1145/168304.168306.

James. Beal, Matthew (Jan. 2003). "Variational algorithms for approximate Bayesian inference /". In: *PhD thesis*.

Jordan, Michael I. et al. (Nov. 1999). "An Introduction to Variational Methods for Graphical Models". In: *Mach. Learn.* 37.2, pp. 183–233. ISSN: 0885-6125. DOI: 10.1023/A:1007665907178. URL: https://doi.org/10.1023/A:1007665907178.

Journel, A. G. and C. J. Huijbregts (1978). "Mining Geostatistics". In: *Academic Press*.

Kolmogoroff, A. (1941). "Interpolation und Extrapolation von stationären zufälligen Folgen". In: *Izv. Akad. Nauk SSSR Ser. Mat.*

Lawrence, Neil (2004). "Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data". In: *Advances in Neural Information Processing Systems 16*. Ed. by S. Thrun, L. K. Saul, and B. Schölkopf. MIT Press, pp. 329–336. URL: http://papers.nips.cc/paper/2540-gaussian-process-latent-variable-models-for-visualisation-of-high-dimensional-data.pdf.

LeCun, Yann et al. (1999). "Object Recognition with Gradient-Based Learning". In: *Shape, Contour and Grouping in Computer Vision*. London, UK, UK: Springer-Verlag, pp. 319–. ISBN: 3-540-66722-9. URL: http://dl.acm.org/citation.cfm?id=646469.691875.

MacKay, David J. C. (1998). "Introduction to Gaussian processes". In: *NATO ASI Series F Computer and Systems Sciences*. Vol. 168, pp. 133–166.

Mairal, Julien et al. (2014). "Convolutional Kernel Networks". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2627–2635. URL: http://papers.nips.cc/paper/5348-convolutional-kernel-networks.pdf.

Matheron, G. (1973). "The Intrinsic Random Functions and Their Applications". In: *Advances in Applied Probability*.

Minka, Thomas P. (2001). "Expectation Propagation for Approximate Bayesian Inference". In: *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*. UAI'01. Seattle, Washington: Morgan Kaufmann Publishers Inc., pp. 362–

369. ISBN: 1-55860-800-1. URL: http://dl.acm.org/citation.cfm?id=2074022.2074067.

Moon, T.K. (1996). "The expectation-maximization algorithm". In: *IEEE Signal Processing Magazine* 13.6, pp. 47–60. DOI: 10.1109/79.543975.

Neal, Radford M. (1995). "Bayesian Learning for Neural Networks". AAINN02676. PhD thesis. Toronto, Ont., Canada, Canada: University of Toronto. ISBN: 0-612-02676-0.

Pandey, Gaurav and Ambedkar Dukkipati (June 2014). "Learning by Stretching Deep Networks". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, pp. 1719–1727. URL: http://proceedings.mlr.press/v32/pandey14.html.

Peterson, C. and J. R. Anderson (1987). "A Mean Field Theory Learning Algorithm for Neural Networks". In: *Complex Systems* 1, pp. 995–1019.

Quiñonero-Candela, Joaquin, Carl Edward Ramussen, and Christopher Williams (Apr. 2007). "Approximation methods for Gaussian process regression". In:

Rasmussen, Carl Edward (1997). "Evaluation of Gaussian Processes and Other Methods for Non-linear Regression". AAINQ28300. PhD thesis. Toronto, Ont., Canada, Canada: University of Toronto. ISBN: 0-612-28300-3.

Rasmussen, Carl Edward and Christopher Williams (Nov. 23, 2005). *Gaussian Processes for Machine Learning*. MIT Press Ltd. 272 pp. ISBN: 026218253X. URL: https://www.ebook.de/de/product/5192092/carl_edward_university_of_cambridge_rasmussen_christopher_k_i_university_of_edinburgh_williams_gaussian_processes_for_machine_learning.html.

Russell, Stuart and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press. ISBN: 0136042597, 9780136042594.

Salimbeni, Hugh and Marc Peter Deisenroth (2017). "Doubly Stochastic Variational Inference for Deep Gaussian Processes". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., pp. 4591–4602. ISBN: 978-1-5108-6096-4. URL: http://dl.acm.org/citation.cfm?id=3294996.3295212.

Seeger, Matthias (2002). *Relationships between Gaussian processes, Support Vector machines and Smoothing Splines*.

Snelson, Edward and Zoubin Ghahramani (2006). "Sparse Gaussian Processes using Pseudo-inputs". In: *Advances in Neural Information Processing Systems 18*. Ed. by Y. Weiss, B. Schölkopf, and J. C. Platt. MIT Press, pp. 1257–1264. URL: http://papers.nips.cc/paper/2857-sparse-gaussian-processes-using-pseudo-inputs.pdf.

Titsias, Michalis K. (2009). "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April*

16-18, 2009, pp. 567–574. URL: http://proceedings.mlr.press/v5/titsias09a.html.

Ueda, Naonori and Zoubin Ghahramani (Dec. 2002). "Bayesian Model Search for Mixture Models Based on Optimizing Variational Bounds". In: *Neural Netw.* 15.10, pp. 1223–1241. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(02)00040-0. URL: http://dx.doi.org/10.1016/S0893-6080(02)00040-0.

van der Wilk, Mark, Carl Edward Rasmussen, and James Hensman (Sept. 6, 2017). "Convolutional Gaussian Processes". In: *31st Conference on Neural Information Processing Systems (NIPS 2017)*. arXiv: http://arxiv.org/abs/1709.01894v1 [stat.ML].

Wiener, Norbert (1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. MIT Press, Cambridge, Mass.

Williams, Christopher (1996). "Computing with Infinite Networks". In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. NIPS'96. Denver, Colorado: MIT Press, pp. 295–301. URL: http://dl.acm.org/citation.cfm?id=2998981.2999023.

Williams, Christopher and D. Barber (Dec. 1998). "Bayesian classification with Gaussian processes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.12, pp. 1342–1351. ISSN: 1939-3539. DOI: 10.1109/34.735807.

Wilson, Andrew Gordon et al. (2016). "Stochastic Variational Deep Kernel Learning". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., pp. 2594–2602. ISBN: 978-1-5108-3881-9. URL: http://dl.acm.org/citation.cfm?id=3157382.3157388.