The dings that will become a thesis at some point

by

Ingrid Holm

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

February 2018

Abstract

This is an abstract text.

To someone

This is a dedication to my cat.

Acknowledgements

I acknowledge my acknowledgements.

Contents

Chapter 1

Introduction

- why nlo?
    - why gp?
    - why dgp?

Chapter 2

Gaussian Processes

In this chapter Gaussian process regression is introduced. First, some concepts and expressions in Bayesian statistics are reviewed. The following section introduces the mathematical framework needed for Gaussian processes, before selected covariance functions are discussed. Concepts in Bayesian model selection are used as a basis to quantify and improve the quality of predictions. Finally, distributed Gaussian processes are introduced as a way of scaling Gaussian processes to larger datasets.

2.1 Introduction to Bayesian Statistics

There are two general philosophies in statistics, namely *frequentist* and *Bayesian* statistics. To understand where they differ, consider a statement statisticians from both branches consider to be true

Statisticians use probability to describe uncertainty.

The difference between Bayesian and frequentist statistics is at the definition of the *uncertain*. Since uncertainty is described by probability this must also vary, and one distinguishes between *objective* and *subjective* probability. Consider an example in which a statistician throws a dice. Before throwing, he is uncertain about the outcome of the dice toss. This uncertainty related to the outcome is *objective*: no one can know if he will throw a 1 or a 4. On the other hand, he might also be uncertain about the underlying probability distribution of the dice toss. Is the dice loaded? Is one of the edges sharper than the others? This uncertainty is *subjective*, as it may vary depending on how much information is available about the system. One of the main critiques of subjective probability posed by frequentists is that the final probability depends on who you ask.

2.1.1 Bayes' Theorem

To further illustrate the difference between frequentist and Bayesian statistics *Bayes' theorem* is introduced. Bayes' theorem can be derived from the familiar rules of probability

$$P(X|I) + P(\bar{X}|I) = 1, \tag{2.1}$$

$$P(X, Y|I) = P(X|Y, I) \times P(Y|I), \tag{2.2}$$

commonly known as the *sum rule* and *product rule*, respectively. $P(X|I)$ is the probability of outcome X given the information I, and $P(X|Y, I)$ is the probability of outcome X given the information I *and* outcome Y. The bar over $\bar{X}$ means that the outcome X does *not* happen. The sum rule states that the total probability of the outcomes X and $\bar{X}$ is equal to 1. This is rather untuitive, considering an event either takes place not. The second rule, the product rule, states that the probability of both outcomes X and Y is equal to the probability of Y times the probability of X given that Y has ocurred. These expressions combine to give Bayes' theorem, first formulated by reverend Thomas Bayes in 1763,

$$P(X|Y, I) = \frac{P(Y|X, I) \times P(X|I)}{P(Y|I)}. \tag{2.3}$$

This theorem states that the probability of X given Y equals the probability of Y given X times the probability of X, divided by the probability of Y. Surprisingly, there is nothing Bayesian about Bayes' theorem. It merely reformulates the rules of logical consistent reasoning stated by Richard Cox in 1946 [6]. Laplace was the one to make Bayes' theorem Bayesian, when he used the theorem to perform inference about distribution parameters. These are, for example, the mean and variance of a Gaussian distribution. The resulting expression is

$$P(\Theta = \theta|X = x) = \frac{P(X = x|\Theta = \theta)P(\Theta = \theta)}{P(X = x)}, \tag{2.4}$$

where Θ are the possible probability distribution parameters, X are the possible outcomes, $P(X = x)$ is a normalization constant called the *marginal likelihood*, and $P(X = x|\Theta = \theta)$ and $P(\Theta = \theta)$ are the *likelihood* and *prior*, respectively. In other words, Eq. (4.4) states the probability of the parameters θ given the knowledge of outcomes x.

A crucial parting of Bayesian statistics from frequentist statistics is at the introduction of the *prior*, which expresses a probability distribution on the *parameters* of the probability distribution.

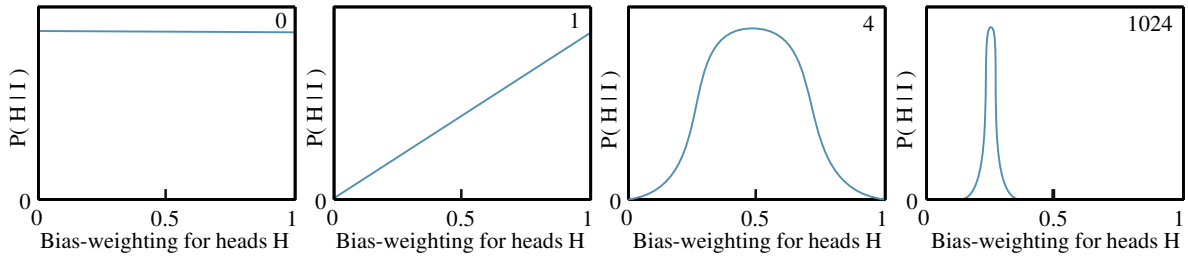


Figure 2.1: The posterior probability distribution of the bias-weighting of a coin for the uniform prior, $P(H|I)$. The first panel from the left is before the coin is tossed, the second panel is after 1 toss, the third is after 4 tosses, and the fourth is after 1024 tosses. The posterior converges towards a narrow peak at 0.25, so the coin is biased.

2.1.2 Priors and Likelihood

The likelihood $P(X = x|\Theta = \theta)$ is simply the probability of the observations x given the parameters of the probability distribution θ. Conversely, the prior expresses a prior belief or assumption of the data, and has to be determined beforehand. The measure $P(\Theta = \theta|X = x)$ from Eq. (4.4) is called the *posterior distribution*. This can be thought of as the prior belief, modified by how well this belief fits the data,

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{marginal likelihood}}.$$

Consider an example. The statistician from before now sets about tossing a coin. Before tossing he assumes the probability of all outcomes is equal, and so adopts a flat prior probability distribution. After one throw he gets heads, and the posterior changes to a function with high probability for heads, and low for tails. After 4 throws where two were heads and two were tails the posterior shows an equal probabilty for heads and tails, with a wide distribution centered at 0.5. After several throws the distribution converges to a narrow peak around 0.25, illustrated in Fig. (4.1). This indicates an unfair coin that is biased towards tails.

2.1.3 Best Estimate and Reliability

Given a posterior distribution it is important to decide how well it fits the data. For that purpose the *best estimate* and *reliability* are defined. The best estimate X_0 is the outcome with the highest probability. In other words, it is the maximum

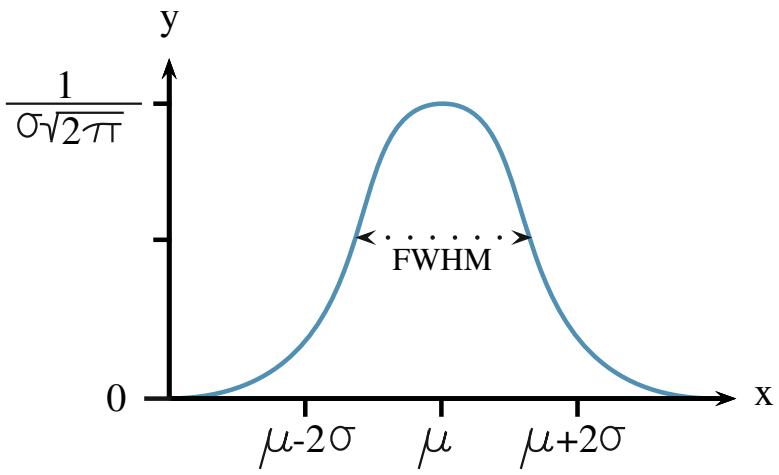


Figure 2.2: A Gaussian probability distribution. The maximum is at the mean value μ, with a full width at half maximum (FWHM) at around 2.35σ. Figure from [6].

of the posterior

$$\frac{dP}{dX}\Big|_{X_0} = 0, \qquad \frac{d^2P}{dX^2}\Big|_{X_0} < 0. \tag{2.5}$$

The second derivative must be negative to insure that X_0 is, in fact, a maximum. Equally important to finding a best estimate is knowing how reliable the estimate is. Reliability is connected the width of the distribution, or how much the data is smeared out. A narrow distribution has low uncertainty, while a wide distribution has large uncertainty. The width is found by Taylor expanding the posterior, and taking the logarithm [1]

$$L = L(X_0) + \frac{1}{2}\frac{d^2}{dx^2}L\Big|_{X_0}(X - X_0)^2 + ..., \quad L = \log_e\Big[\text{prob}(x|\{data\}, I)\Big] \tag{2.6}$$

This gives an approximate posterior in the shape of a *Gaussian distribution*, with mean and variance given by

$$\mathcal{N}(\mu, \sigma) \text{ where } \mu = X_0,\ \sigma = \Big(-\frac{d^2L}{dx^2}\Big)^{-1/2}. \tag{2.7}$$

The Gaussian distribution is symmetric with respect to the maximum at $x = \mu$, and has a full width at half maximum (FWHM) at around 2.35σ, as shown in Fig. (4.2).

[1] L is a monotonic function of P, so the maximum of L is at the maximum of P.

2.1.4 Covariance

Before embarking on Gaussian processes the concept of *covariance* is needed. A more detailed description is found in [6]. In many cases the equations are not as simple to solve as in Eq. (4.6), as the probability distribution might have several quantities of interest $\{X_i\}$. In that case, a set of *simultaneous equations* must be solved to get the best estimate

$$\frac{dP}{dX_i}\Big|_{X_{0j}} = 0. \tag{2.8}$$

To simplify expressions consider the problem in two dimensions, so that $\{X_i\} = (X, Y)$. The Taylor expansion of L is then

$$\begin{aligned} L =& L(X_0, Y_0) + \frac{1}{2}\Big[\frac{d^2L}{dX^2}\Big|_{X_0,Y_0}(X - X_0)^2 \\ &+ \frac{d^2L}{dY^2}\Big|_{X_0,Y_0}(Y - Y_0)^2 + 2\frac{d^2L}{dXdY}\Big|_{X_0,Y_0}(X - X_0)(Y - Y_0)\Big] + ... \end{aligned} \tag{2.9}$$

There are now four partial derivatives, reduced to three using the rules for mixed partial derivatives $\frac{\partial^2}{\partial X \partial Y} = \frac{\partial^2}{\partial Y \partial X}$. Writing the quadratic term of Eq. 4.9 in matrix form gives

$$Q = \begin{pmatrix} X - X_0 & Y - Y_0 \end{pmatrix} \begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} X - X_0 \\ Y - Y_0 \end{pmatrix}, \tag{2.10}$$

where the matrix elements are

$$A = \frac{\partial^2 L}{\partial X^2}\Big|_{X_0,Y_0}, \qquad B = \frac{\partial^2 L}{\partial Y^2}\Big|_{X_0,Y_0}, \qquad C = \frac{\partial^2 L}{\partial X \partial Y}\Big|_{X_0,Y_0}. \tag{2.11}$$

The *variance* is defined as the expectation value of the square of deviations from the mean. In the two-dimensional case this becomes [6]

$$\mathrm{Var}(X) = \sigma_x^2 = \big\langle (X - X_0)^2 \big\rangle = \int\int (X - X_0)^2 P(X, Y|\{\mathrm{data}\}, I) dX dY. \tag{2.12}$$

This is the variance σ_X^2 for X, and its square root is the standard deviation σ_X. A similar expression can be found for Y, by switching X and Y. It is also possible to find the simultaneous deviations of the parameters X and Y, or the correlation between the inferred parameters. This is called the *covariance* σ_{XY}^2, and is in two dimensions given by

$$\sigma_{XY}^2 = \langle (X - X_0)(Y - Y_0) \rangle = \int\int (X - X_0)(Y - Y_0) P(X, Y|\{\mathrm{data}\}, I) dX dY. \tag{2.13}$$

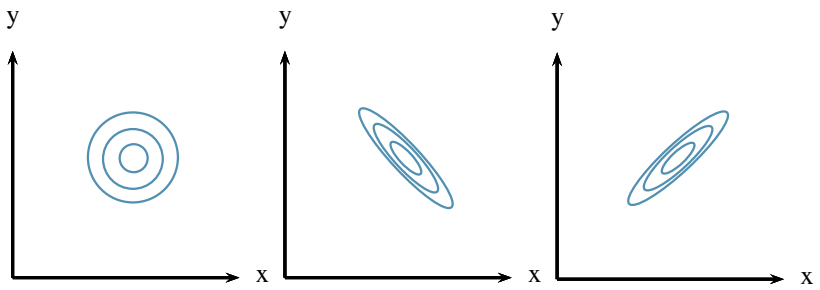


Figure 2.3: A schematic illustration of covariance and correlation. (a) The contours of a posterior pdf with zero covariance, where the inferred values of X and Y are uncorrelated. (b) The corresponding plot when the covariance is large and negative; $Y + mX =$ constant along the dotted line (where $m > 0$), emphasizing that only this sum of the two parameters can be inferred reliably. (c) The case of positive correlation, where we learn most about the difference YmX; this is constant along the dotted line.

The covariance indicates how an over- or underestimation of one parameter affects another parameter. If, for example, an overestimation of X leads to an overestimation of Y, the covariance is positive. If the overestimation of X has little or no effect on the estimation of Y, the covariance is negligible or zero $|\sigma_{XY}| \ll \sqrt{\sigma_X^2 \sigma_Y^2}$. These effects are illustrated in Fig. (4.3). It can be shown that [6]

$$\text{cov} = \begin{pmatrix} \sigma_X^2 & \sigma_{XY}^2 \\ \sigma_{XY}^2 & \sigma_Y^2 \end{pmatrix} = - \begin{pmatrix} A & C \\ C & B \end{pmatrix}^{-1}. \tag{2.14}$$

This is called the *covariance matrix*.

2.2 Gaussian Process Regression

Gaussian processes (GP) is a supervised machine learning method, designed to solve regression and probabilistic classification problems. In this thesis only regression will be used.

Consider a set of points $\mathcal{D} = \{\mathbf{x}_i, y_i\}$, where y is some (possibly noisy) function of $\mathbf{x}$, $y = f(\mathbf{x}) + \varepsilon$. This is illustrated by the black dots in in Fig. (4.4). In machine learning $\mathcal{D}$ is the *training data*, as it is used to train the model. It consists of *features*, which are the input vectors $\mathbf{x}_i$, and *targets*, which are the function values y_i. The set of points is discrete, so there is some $\mathbf{x}^*$ for which the target y^* is unknown. Gaussian Processes (GP) predict a Gaussian distribution *over function values* at this point $\mathbf{x}^*$, with a corresponding mean $m(\mathbf{x}^*)$ and variance σ^2. The GP prediction for the target value y^* is the mean $m(\mathbf{x}^*)$, with uncertainty σ^2.

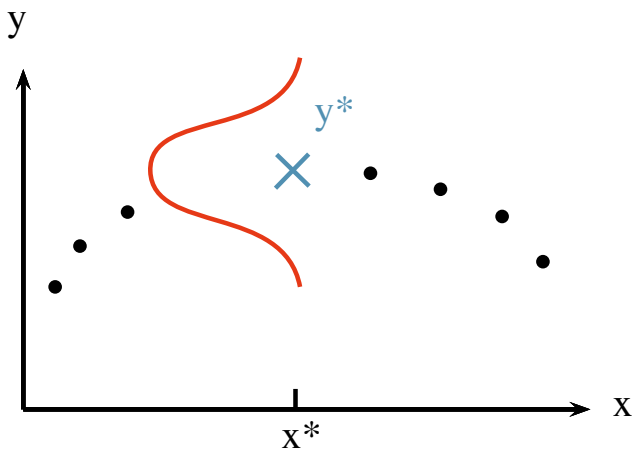


Figure 2.4: An illustration of a GP prediction of the target value y^* (blue cross), given the known set of points $\{x_i, y_i\}$ (black dots). The prediction is a Gaussian distribution in y with mean y^* and variance σ^2. The Gaussian distribution is drawn in red with y on the vertical axis, with uncertainty in the y-direction.

The predicted target y^* can be viewed as a linear combination of the known targets y_i, where the weights are controlled by the covariances between $\mathbf{x}_i$ and the test point $\mathbf{x}^*$.

Function Space View

Since Gaussian processes provide distributions over functions, it is useful to consider the problem in the function space view introduced in [5]. For a real process $f(\mathbf{x})$ the mean $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ are defined as

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{2.15}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \tag{2.16}$$

where $\mathbb{E}[a]$ is the expectation value of some quantity a. Given the mean and covariance, the Gaussian distribution for $f(\mathbf{x})$ is completely specified

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.17}$$

The covariance between two points $k(\mathbf{x}_i, \mathbf{x}_j)$ is decided by the *kernel function*. In this text the running example will be the squared exponential (SE) kernel, given by

$$k(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(-\frac{1}{2}|\mathbf{x}_i - \mathbf{x}_j|^2\Big). \tag{2.18}$$

Note that the covariance of the function values $f(\mathbf{x}_i)$, $f(\mathbf{x}_j)$ only depends on the input parameters $\mathbf{x}_i$, $\mathbf{x}_j$.

Specifying the covariance function specifies a distribution over functions [5]. This is because allowed functions must obey the correlation decided by $k(\mathbf{x}_i, \mathbf{x}_j)$. Using the kernel on a set of input vectors contained in the matrix X^*, gives the *covariance matrix*. Combined with an initial mean of zero [2] one obtains the *prior* distribution

$$f(x) \sim \mathcal{N}(0, K(X^*, X^*)). \tag{2.19}$$

This distribution encodes the prior knowledge about the function values $f(x)$, and so the choice of kernel is one of the most important steps in learning with GPs. The prior is modified by the training data to provide a posterior distribution. In Fig. (4.5) samples are drawn from both the prior and posterior distribution. Samples are here taken to mean functions that obey the covariance function. In the case of the prior, the samples are drawn from a distribution of functions with mean zero and constant variance, meaning that if you drew enough functions the mean value of all functions at every x would be zero. For the posterior, the mean values and uncertainties have been modified by the training data. In a point where there is training data the uncertainty is zero [3], and so all samples drawn from this distribution must pass through this point. Far away from training points the uncertainty is large.

Consider a simple example of a noise-free set of training points $\{\mathbf{x}_i, y_i\}$, so that $y = f(\mathbf{x})$. The joint distribution of training outputs $\mathbf{f}$ and test outputs $\mathbf{f}^*$ according to the prior is then

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X^*) \\ K(X,X^*) & K(X^*,X^*) \end{bmatrix} \right) \tag{2.20}$$

For n training points and n^* test points, $K(X,X)$ is the $n \times n$ matrix containing the covariance of training points, $K(X,X^*)$ is the $n \times n^*$ matrix of covariance between the test and training points, and $K(X^*,X^*)$ is the $n^* \times n^*$ matrix containing the covariance of test points. By conditioning the distribution of $\mathbf{f}^*$ on the observations, the posterior distribution over $\mathbf{f}^*$ is obtained[4] [5]

$$\mathbf{f}_* \big| X_*, X, \mathbf{f} \sim \mathcal{N}(K(X_*,X)K(X,X)^{-1}\mathbf{f}, \tag{2.21}$$

$$K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)). \tag{2.22}$$

Gaussian Noise Model

Noise-free observations are rare. In most cases targets will contain some noise $y = f(\mathbf{x}) + \varepsilon$, where the noise ε is assumed to follow a Gaussian distribution

[2] The mean does not have to be zero, it could for example be the mean of the training data.
[3] Assuming there is no noise in the data.
[4] For more details, see Appendix A.2 in [5].

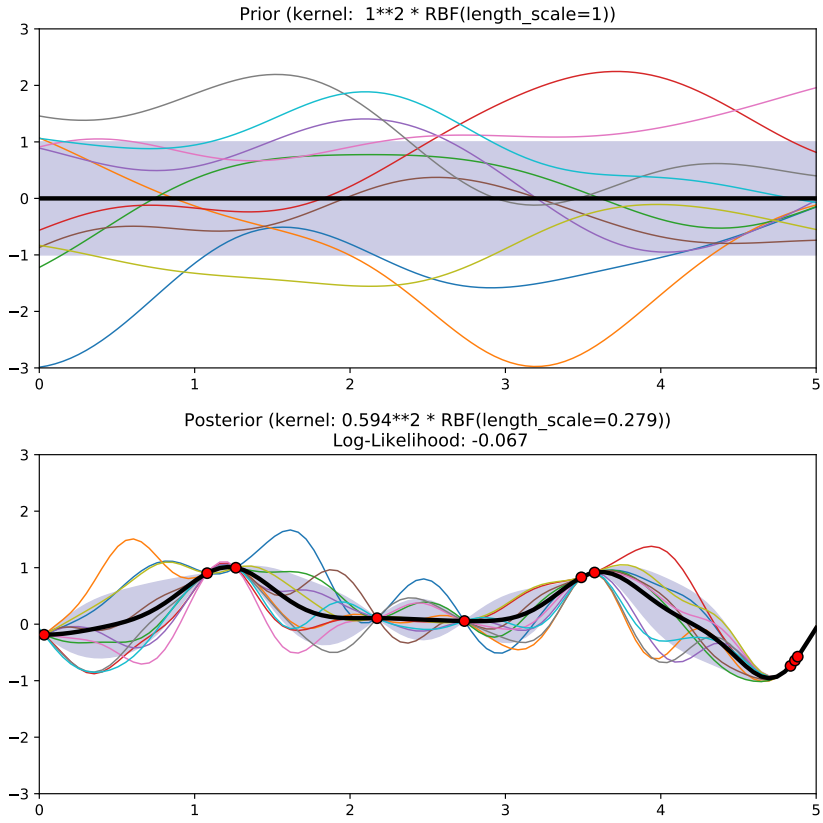


Figure 2.5: Drawing functions from the prior (top) and posterior (bottom) distributions. The thick, black line represents the mean of the distribution, while the shaded, blue area is the uncertainty. The multiple colored lines are functions drawn randomly from the prior and posterior distributions, whose correlation are dictated by the covariance function. The prior has mean 0 and covariance given by the squared exponential function. The posterior has been modified by training points (red dots), giving rise to zero uncertainty at the points where training data exists, and an altered mean value for the distribution. Figure generated using scikit-learn.

$\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$. This is the *Gaussian noise model*. The covariance can then be expressed as

$$\text{cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij} \tag{2.23}$$

which gives for the prior distribution

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 \mathbb{I} & K(X,X^*) \\ K(X,X^*) & K(X^*,X^*) \end{bmatrix}\right). \tag{2.24}$$

The conditioned distribution is then

$$\mathbf{f}^* \big| X^*, X, \mathbf{f} \sim \mathcal{N}(\bar{\mathbf{f}}^*, \text{cov}(\mathbf{f}^*)), \text{ where} \tag{2.25}$$

$$\bar{\mathbf{f}}^* = K(X^*,X)[K(X,X) + \sigma_n^2 \mathbb{I}]^{-1}\mathbf{y}, \tag{2.26}$$

$$\text{cov}(\mathbf{f}^*) = K(X^*,X^*) - K(X^*,X)[K(X,X) + \sigma_n^2 \mathbb{I}]^{-1} K(X,X^*) \tag{2.27}$$

For the sake of tidying up the expression define the matrix $K \equiv K(X,X)$ and the matrix $K^* \equiv K(X,X^*)$. In the case of a single test point $\mathbf{x}^*$ the matrix K^* is written as a vector $\mathbf{k}(\mathbf{x}^*) = \mathbf{k}^*$. Using this compact notation the GP prediction for a single test point $\mathbf{x}^*$ is

$$\bar{f}^* = \mathbf{k}^{*T}(K + \sigma_n^2 \mathbb{I})^{-1}\mathbf{y}, \tag{2.28}$$

$$\mathbb{V}[f^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^{*T}(K + \sigma_n^2 \mathbb{I})^{-1}\mathbf{k}^*. \tag{2.29}$$

Note that the predicted mean value $\bar{f}^*$ can be viewed as a linear combination of y_i of the form $\alpha \mathbf{y}$, where $\alpha = \mathbf{k}^{*T}(K + \sigma_n^2 \mathbb{I})^{-1}$. α then only contains the covariance between features.

Eqs. (4.28)-(4.29) form the basis for GP prediction in `scikit-learn` [**?**]. The algorithm is shown in Algorithm (3), and uses the Cholesky decomposition of the covariance matrix.

Data: X (inputs), $\mathbf{y}$ (targets), k (covariance function/kernel), σ_n^2 (noise level), $\mathbf{x}_*$ (test input).
L = Cholesky decomposition $(K + \sigma_n^2 I)$;
$\boldsymbol{\alpha} = (L^T)^{-1}(L^{-1}\mathbf{y})$;
$\bar{f}_* = \mathbf{k}_*^T \boldsymbol{\alpha}$;
$\mathbf{v} = L^{-1}\mathbf{k}_*$;
$\mathbb{V}[f_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T\mathbf{v}$;
$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$;
Result: f_* (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood).

Algorithm 1: Algorithm 2.1 from [5].

2.3 Covariance Functions

Covariance functions and kernels have been touched upon, but will be investigated further as they are central to Gaussian processes. A function that only depends on the difference between two points, $\mathbf{x} - \mathbf{x}'$, is called *stationary*. This implies that the function is invariant to translations in input space. If, in addition, it only depends on the length $r = |\mathbf{x} - \mathbf{x}'|$, the function is *isotropic* [5]. Isotropic functions are commonly referred to as *radial basis functions* (RBFs). The covariance function can also depend on the dot product, $\mathbf{x} \cdot \mathbf{x}'$, and is then called a *dot product* covariance function.

A function which maps two arguments $\mathbf{x} \in \mathcal{X}$, $\mathbf{x}' \in \mathcal{X}$ into $\mathbb{R}$ is generally called a *kernel* k. Covariance functions are symmetric kernels, meaning that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. As previously mentioned, the matrix containing all the covariance elements is called the *covariance matrix*, or the Gram matrix K, whose elements are given by

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{2.30}$$

The kernel should contain information about the noise in the data, represented by a constant term added to the diagonal

$$k(\mathbf{x}_i, \mathbf{x}_j)_{noise} = C\delta_{ij}, \tag{2.31}$$

where $C \in \mathbb{R}$ is a real, constant number, and δ_{ij} is the Dirac delta function. In `scikit-learn` this can be implemented either by giving a fixed noise level α to the regressor function, or by using the `WhiteKernel`, which estimates the noise level from the data. This kernel is implemented in `scikit-learn` in the following way for noise level 0.001 with bounds $[10^{-10}, 1]$

```
from sklearn.gaussian_processes.kernels import WhiteKernel
whitenoise = WhiteKernel(noise_level=0.001,
    noise_level_bounds=(1e-10,1))
```

2.3.1 The Squared Exponential Covariance Function

The *squared exponential covariance function* (SE) has the form

$$k_{SE}(r) = \exp\Big(-\frac{r^2}{2\ell^2}\Big), \tag{2.32}$$

where ℓ is the *characteristic length scale*. The length scale determines the smoothness of the function. For a large length scale one should expect a very slowly

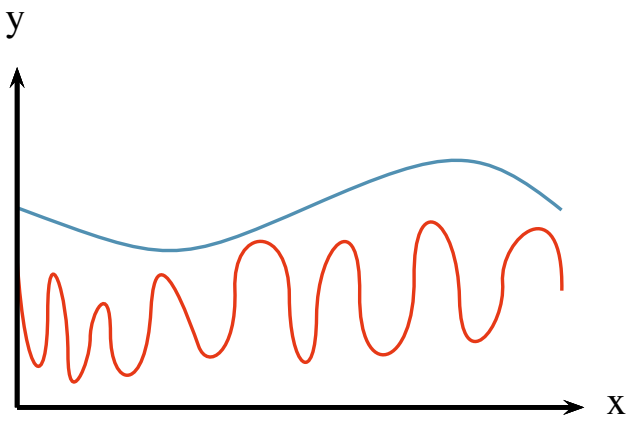


Figure 2.6: The effect of varying the length scale ℓ. A long length scale (blue) gives a smooth, slowly varying function, while a short length scale (red) gives a more staccato, quickly varying function.

varying function, while a shorter length scale means a more rapidly varying function, see the illustration in Fig. (4.6). The SE is infinitely differentiable and therefore very smooth.

The SE is implemented in `scikit-learn` under the name radial basis function, and may be called in the following way for length scale 10, with bounds on the length scale $[0.01, 100]$

```
from sklearn.gaussian_process.kernels import RBF
rbf = RBF(length_scale=10, length_scale_bounds=(1e-2, 1e2))
```

2.3.2 The Matérn Class

The *Matérn class of covariance functions* is given by

$$k_{Matérn}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\Big(\frac{\sqrt{2\nu}r}{\ell}\Big)^{\nu} K_{\nu}\Big(\frac{\sqrt{2\nu}r}{\ell}\Big), \tag{2.33}$$

where $\nu, \ell > 0$, and K_{ν} is a modified Bessel function. The hyperparameter ν controls the smoothness of the function, as opposed to the SE kernel which is by definition very smooth. For $\nu \to \infty$ this becomes the SE kernel. In the case of ν being half integer, $\nu = p + \frac{1}{2}$, the covariance function is simply the product of an

[5]Invariant to rigid rotations in input space.

exponential and a polynomial

$$k_{\nu=p+\frac{1}{2}} = \exp\Big(-\frac{\sqrt{2\nu}r}{\ell}\Big)\frac{\Gamma(p+1)}{\Gamma(2p+1)}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}\Big(\frac{\sqrt{8\nu}r}{\ell}\Big)^{p-i}. \tag{2.34}$$

In machine learning the two most common cases are for $\nu = 3/2$ and $\nu = 5/2$

$$k_{\nu=3/2}(r) = \Big(1+\frac{\sqrt{3}r}{\ell}\Big)\exp\Big(-\frac{\sqrt{3}r}{\ell}\Big), \tag{2.35}$$

$$k_{\nu=5/2}(r) = \Big(1+\frac{\sqrt{5}r}{\ell}+\frac{5r^2}{3\ell^2}\Big)\exp\Big(-\frac{\sqrt{5}r}{\ell}\Big). \tag{2.36}$$

In `scikit-learn` the hyperparameter ν is fixed, and so not optimized during training. The Matérn kernel is considered more appropriate for physical processes [5], and may be called in `scikit-learn` in the following way for length scale 10, length scale bounds $[0.01, 100]$ and $\nu = 3/2$

```
from sklearn.gaussian_process.kernels import Matern
matern = Matern(length_scale=10, length_scale_bounds=(1e-2,
    1e2), nu=1.5)
```

For values not in $\nu = [0.5, 1.5, 2.5, \infty]$ `scikit-learn` evaluates Bessel functions explicitly, which increases the computational cost by a factor as high as 10.

Other Kernels

There are several kernels which are not discussed here. Kernels can be multiplied and summed to form new kernels, making the space of possible kernels infinite. For further details see chapter 4 in [5].

Hyperparameters

Each kernel has a vector of hyperparameters, *e.g.* $\boldsymbol{\theta} = (\{M\}, \sigma_f^2, \sigma_n^2)$ for the radial basis function (RBF)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j))^T M(\mathbf{x}_i - \mathbf{x}_j) + \sigma_n^2\delta_{ij}. \tag{2.37}$$

The matrix M can have several forms, amongst them

$$M_1 = \ell^{-2}\mathbb{I}, \qquad M_2 = \mathrm{diag}(\boldsymbol{\ell})^{-2}. \tag{2.38}$$

Choosing $\boldsymbol{\ell}$ to be a vector in stead of a scalar is in many cases useful, especially if the vector of features contain values of different scales, *e.g.* $\mathbf{x} = (x_1, x_2)$ where $x_1 \in [0, 1]$ and $x_2 \in [200, 3000]$. The length scale can be set to a vector in `scikit-learn` by giving the `length_scale` parameter as a `numpy` array of the same dimension as the feature vector $\mathbf{x}$.

2.4 Model Selection

The choice of kernel and hyperparameters is important for the quality of the GP prediction. Model selection means finding the kernel and corresponding hyperparameters that best fit the data. This is referred to as *training* in machine learning. In this section Bayesian model selection is quickly overviewed, and the log marginal likelihood and cross validation are considered.

2.4.1 Bayesian Model Selection

A model has a set of model structures $\mathcal{H}_i$, hyperparameters $\boldsymbol{\theta}$ and parameters $\mathbf{w}$. Feature selection is done at all levels in a hierarchical way, by finding the posterior over *parameters*, the posterior over *hyperparameters* and the posterior for the *model*. Here only the posterior over parameters is considered [5],

$$p(\mathbf{w}|\mathbf{y}, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)}{p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)}, \tag{2.39}$$

as it gives rise to the *marginal likelihood* $p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)$, given by

$$p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)d\mathbf{w} \quad \text{(marginal likelihood)}. \tag{2.40}$$

Because of the complexity of integrals involved in model selection, it is common to maximize the marginal likelihood with respect to hyperparameters $\boldsymbol{\theta}$. This maximization is what distinguishes Bayesian model selection from other model selection schemes, as it incorporates a trade-off between model complexity and model fit. This means that a complex model will allow for several different kinds of models, but each of them will get a low probability. Meanwhile, simple models will only have a few possibilities, but each of these will have a large probability, see Fig. (4.7).

2.4.2 Log Marginal Likelihood

Gaussian process regression models with Gaussian noise have the wonderful trait of analytically tractable integrals for the marginal likelihood. The exact expression for the log marginal likelihood [6] can be shown to be [5]

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi. \tag{2.41}$$

Each of the terms has an interpretation: $-\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y}$ is the only term involving the data, and is therefore the data-fit; $-\frac{1}{2}\log|K_y|$ is the complexity penalty

[6] The logarithm is used as the marginal likelihood varies rapidly.

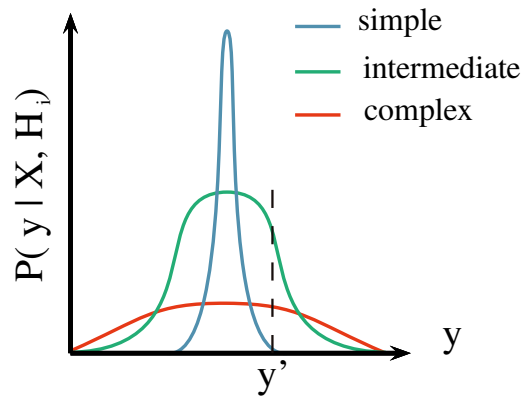


Figure 2.7: The marginal likelihood is the probability of the data, given the model. The number of data points n and inputs X are fixed. The horizontal axis represents an idealized set of all possible vectors of targets $\mathbf{y}$. Since the marginal likelihood is a probability distribution it must normalize to unity. For a particular set $\mathbf{y}'$, indicated by the dashed line, the intermediate model is preferred to the simple and complex ones.

depending only on the covariance function and the inputs; and $-\frac{n}{2}\log 2\pi$ is a normalization term. The marginal likelihood is conditioned on the hyperparameters of the covariance function $\boldsymbol{\theta}$, and the optimal parameters are found by maximizing. This requires the partial derivatives of the log marginal likelihood (LML)

$$\frac{\partial}{\partial \theta_j}\log p(\mathbf{y}|X,\boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial \theta_j}K^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}(K^{-1}\frac{\partial K}{\partial \theta_j}). \tag{2.42}$$

Computing the inverse of a matrix, K^{-1}, is computationally complex, and for n training points goes as $\mathcal{O}(n^3)$. Once this is done, however, finding the partial derivatives only requires complexity $\mathcal{O}(n^2)$, and so gradient based optimizers are advantageous.

The LML can have several local optima, as seen in Fig. (4.8). These correspond to different interpretations of the data. The rightmost optima in Fig. (4.8) for example, favors a small length scale and smaller noise level. This means that it considers little of the data to be noise. The rightmost optimum has a higher noise level, and allows for several large length scales, as it considers most of the data to be noise. Features with very large length scales are considered superfluous, as the function value depends little on them. Because of this type of complication, it might be wise to restart the optimizer a few times during learning.

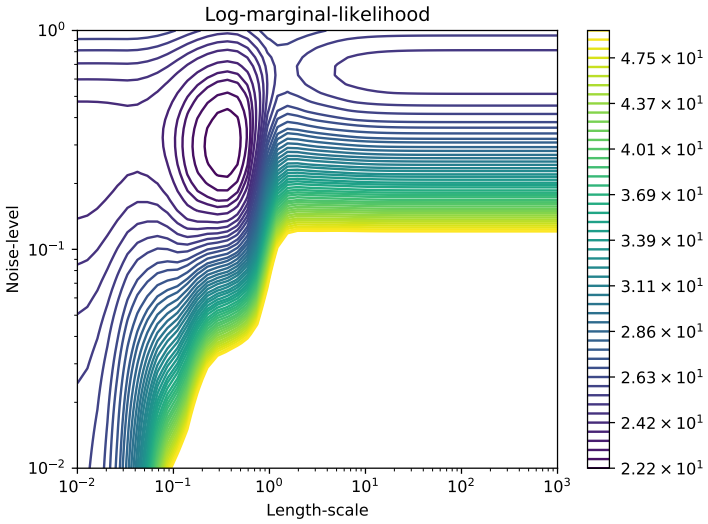


Figure 2.8: A contour plot of the log marginal likelihood with two local optima. The rightmost optima favours a short length scale and low noise, while the leftmost favors a high noise level and therefore several large length scales. Plot generated using scikit-learn.

2.4.3 Cross Validation

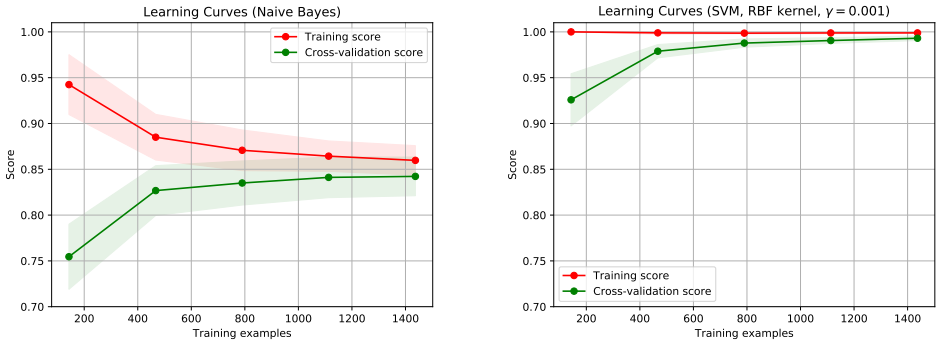
Cross validation is a means of monitoring the performance of a model. In k-fold validation this is done by dividing the data into k subsets and using $k-1$ folds to train the model, and a single fold to validate it. This is repeated k times. Cross-validation requires a loss function, such as the mean relative deviance or the R^2 score. The latter is given by

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1}(y_i - \bar{y})^2}, \tag{2.43}$$

where $\hat{y}_i$ is the predicted value of the ith sample, y_i is the true value and $\bar{y} = \frac{1}{N}\sum_{i=0}^{N-1} y_i$ for N samples. This is the score used for cross validation in this thesis.

Cross-validation can be used to plot learning curves, which is a tool to find out whether the model benefits from adding more data. The learning curve plots the training score and validation score used to find out if the model is *overfitting* or *underfitting*. *Overfitting* means that the model is a perfect fit to the training data, but predicts poorly for test data because it is not general. *Underfitting* occurs when the model is not able to capture the underlying structure of the data.

Examples of learning curves are shown in Fig. (4.9) for Naive Bayes and SVM



(a) Underfitting. **(b)** Well fit.

Figure 2.9: Learning curves for two different estimators.

estimators [7]. In a) both the training score and cross-validation score tend to a value below 1, which indicates underfitting. This model will not benefit from more data. The example in b) shows a training score of approximately 1, and a cross validation score that converges towards 1. This model could benefit from more data.

2.4.4 Relative Deviance

In this project the main loss function used for comparing predictions is the relative deviance. For true values y_i and values predicted by the estimator $\hat{y}_i$ this is given by

$$\varepsilon_i = \frac{y_i - \hat{y}_i}{y_i}. \tag{2.44}$$

The relative deviance is used because of the large span of the target values, ranging from about 10^{-30} to 10^{9}. The data is therefore divided into decades, meaning one set contains $\sigma \in [10^i, 10^{i+1}]$. Then a distribution over the relative deviances within each decade is found, with a mean value and variance. These are plotted as a function of i.

2.5 Distributed Gaussian Processes

Limitations of Gaussian Processes

The biggest weakness of Gaussian processes is that they scale poorly with the size of the data set n. The training and predicting scale as $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively, giving GP a practical limit of $\mathcal{O}(10^4)$.

[7] Methods in Machine Learning.

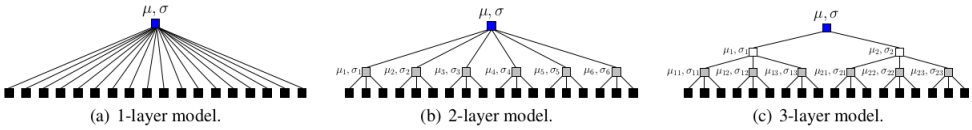


Figure 2.10: From [2].

In [2] a way of scaling GPs to large data sets is proposed, in the form of a robust Bayesian Comittee Machine (rBCM). This method is based on the product-of-experts and Bayesian Comittee Machine, and has the advantage of providing an uncertainty for the prediction.

2.5.1 Product-of-Experts

Product-of-expert (PoE) models are a way of parallelising large computations. They combine several independent computations on subsets of the total data, called 'experts'. In the case of distributed Gaussian processes each expert performs GP on a subset of the training data, and the predictions on a common test set are combined.

Consider the training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, which is partitioned into M subsets $\mathcal{D}^{(k)} = \{\mathbf{X}^{(k)}, \mathbf{y}^{(k)}\}$, $k = 1, ..., M$. Each GP expert does learning on its training data set $\mathcal{D}^{(k)}$, then predictions are combined at the parent node. This node could also be considered an expert for a PoE with several layers, see Fig. (4.10).

2.5.2 Algorithm

The marginal likelihood factorizes into the product of M individual terms because of the independence assumption [2]. The LML is then

$$\log p(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^{(k)}(\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I})^{-1}\mathbf{y}^{(k)} - \frac{1}{2}\log|\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I}|, \qquad (2.45)$$

where $a^{(k)}$ is the quantity corresponding to the kth expert. Computing the LML now entails inverting the $n_k \times n_k$ matrix $(\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I})$, which requires time $\mathcal{O}(n_k^3)$ and memory consumption $\mathcal{O}(n_k^2 + n_k D)$ for $\mathbf{x} \in \mathbb{R}^D$. For $n_k \ll N$, this reduces the computation time and memory use considerably, and allows for parallel computing.

Several methods for prediction are discussed in [2], but here only the robust Bayesian Comittee Machine is introduced. The PoE predicts a function value f^* at a corresponding test input $\mathbf{x}^*$ according to the predictive distribution

$$p(f^*|\mathbf{x}^*, \mathcal{D}) = \frac{\prod_{k=1}^{M} p_k^{\beta_k}(f^*|\mathbf{x}^*, \mathcal{D}^{(k)})}{p^{-1+\sum_k \beta_k}(f^*|\mathbf{x}^*)}. \qquad (2.46)$$

This prediction scheme allows for much flexibility, as it can vary the importance of an expert. The combined predictive mean and variance are

$$\mu_*^{rbcm} = (\sigma_*^{rbcm})^2 \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*)\mu_k(\mathbf{x}_*), \tag{2.47}$$

$$(\sigma_*^{rbcm})^{-2} = \sum_{k=1}^{M} \beta_k \sigma_k^{-2}(\mathbf{x}_*) + \big(1 - \sum_{k=1}^{M} \beta_k\big)\sigma_{**}^{-2}, \tag{2.48}$$

where the parameters β_k control the importance of the individual experts, but also the how strong the influence of the prior is. In the article, these are chosen according to the predictive power of each expert at $\mathbf{x}^*$. More specifically, β_k is the change in differential entropy between the prior $p(f^*|\mathbf{x}^*)$ and the posterior $p(f^*|\mathbf{x}^*, \mathcal{D}^{(k)})$, which can be calculated as

$$\beta_k = \frac{1}{2}(\log \sigma_{**}^2 - \log \sigma_k^2(\mathbf{x}^*)), \tag{2.49}$$

where σ_{**}^2 is the prior variance, and $\sigma_k^2(\mathbf{x}^*)$ is the predictive variance of the kth expert.

2.5.3 Implementing the Algorithm

The mean and variance in Eq. (4.47)-(4.48) were implemented in `Python` using the `scikit-learn` library's existing framework for regular Gaussian processes. The algorithm was parallelised, so that each expert can learn and predict in parallel, before being combined to the final prediction. Pseudocode for the implementation is found in Alg. (4).

For parallelisation the `scikit-learn` function `Parallel` from `joblib` was used, which runs Python functions as pipeline jobs. It uses the Python function `multiprocessing` as a backend. An example of usage with 3 parallel jobs is

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt
>>> Parallel(n_jobs=3)(delayed(sqrt)(i**2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

where `delayed` is a simple trick to be able to create a tuple with a function-call syntax.

2.5.4 Benchmark

The benchmark function for parallelised distributed Gaussian processes is

$$f(x_1, x_2) = 4x_1 x_2,$$

Data: $N_{experts}$ (number of experts), X (inputs), $\mathbf{y}$ (targets), k (initial kernel), σ_n^2 (noise level), $\mathbf{x}^*$ (test input)
Split training data into N subsets: $X_k, \mathbf{y}_k$;
for *each expert* **do**
  Fit GP to training data $X_k, \mathbf{y}_k$;
  Predict μ_*, σ_*^2 for $\mathbf{x}^*$ using GP ;
  $\sigma_{**}^2 = k(x^*, x^*)$;
end
for *each expert* **do**
  $\beta = \frac{1}{2}(\log(\sigma_{**}^2) - \log(\sigma_*^2))$;
  $(\sigma_*^{rbcm})^{-2} += \beta\sigma^{-2} + \left(\frac{1}{n_{experts}} - \beta\right)\sigma_{**}^{-2}$
end
for *each expert* **do**
  $\mu_*^{rbcm} += (\sigma_*^{rbcm})^2 \beta \sigma_*^{-2} \mu_*$
end
Result: Approximative distribution of $f_* = f(\mathbf{x}_*)$ with mean μ_*^{rbcm} and variance $(\sigma_*^{rbcm})^2$.

Algorithm 2: Pseudocode for using rBCM on a single test point $\mathbf{x}_*$. For the fit and prediction of each expert GP Algorithm (3) is used.

where the vectors $\mathbf{x} = (x_1, x_2)$ were drawn from a random normal distribution using the `numpy` function `random.randn`. Gaussian processes implemented by `scikit-learn` in the function `GaussianProcessRegressor` were compared to distributed Gaussian processes with 4 experts. 2000 training points and 1000 test points were used, and the resulting times for the GP and DGP were

$$\text{Gaussian processes time: } 154.12 \text{ s} \tag{2.50}$$
$$\text{Distributed Gaussian processes time: } 5.61 \text{ s} \tag{2.51}$$

Histograms of the relative deviances for Gaussian processes (GP) and Distributed Gaussian processes (DGP) are found in Fig. (4.11).

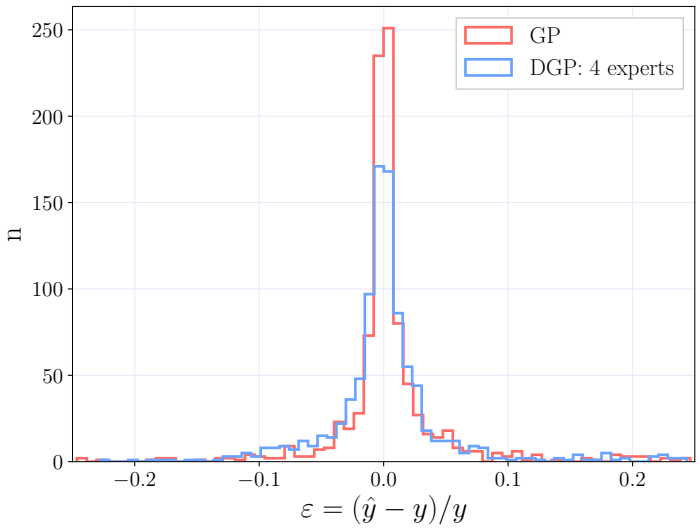


Figure 2.11: Histogram of the relative deviance between true value y and predicted value $\hat{y}$ for Gaussian process regression (GP) and Distributed gaussian process regression (DGP) for the function $f(x_1, x_2) = 4x_1x_2$.

Chapter 3

Supersymmetry at Hadron Colliders

At the moment of writing, the Large Hadron Collider (LHC) at CERN is one of the largest and most important particle physics experiments in the world. In this chapter, some of the advantages and challenges of using hadron colliders are discussed, along with some techniques for moving from theory to physically detectable signals. A short description of supersymmetric phenomenology follows, along with current bounds on some supersymmetric particles. Finally, the squark production cross section is calculated to leading order, and next-to-leading order terms are investigated.

3.1 Hadron Colliders

Colliding hadrons makes it possible to reach very high center-of-mass energies. This is owed to the heavy mass scale of hadrons compared to leptons, resulting in low *synchrotron radiation* which makes the use of circular accelerators beneficial. The circular form means the accelerating structures can be reused as many times as one desires, thus putting 'no limits' on the energies obtained (there are, of course, limits). Synchrotron radiation is the radiation of energy from a particle being accelerated, which goes as $\sim m^{-4}$. Thus for light particles, such as the electron, a lot of energy is wasted. The power radiated by a relativistic electron forced to move in circular motion with radius R is given by Schwinger's formula [**?**]

$$P_e = \frac{2}{3}\frac{e^2 c}{R^2}\left(\frac{E}{m_e c^2}\right)^4, \tag{3.1}$$

where E is the electron energy, m_e is the electron mass and c is the speed of light in vacuum. Because the proton mass is much larger than the electron mass, this

effect is relatively small, allowing the LHC to operate at energies currently as high as 13 TeV.

Amidst this praise of the hadron one question arises: if hadronic colliders are able to reach such high energies, why do bother colliding leptons at all? Alas, the heavy masses come at a kinematical price. The way hadrons are made up of valence and sea quarks and gluons make the kinematics of collisions very difficult to calculate. These quarks and gluons, the so-called *partons*, have the annoying habit of distributing the hadron momentum somewhat randomly amongst themselves. Since we do not know how the energy and momentum is distributed, the momenta of the ingoing particles is unknown. This is why the transverse momentum and energy, not to mention the *missing* transverse energy, are important when analysing data from colliders. However, there is a way to approximate the parton momenta, namely using *parton distribution functions*.

3.1.1 Parton Distribution Functions

Cross sections are calculated for two colliding partons, for example two quarks $q_1 q_2$. The cross section is a function of the center-of-mass energy, $\hat{s}$, which again can be written as a fraction of the center of mass of the colliding protons

$$\hat{s} = s x_1 x_2, \tag{3.2}$$

where s is the center-of-mass energy of the colliding protons, and x_i is the momentum fraction of the quark q_i. The fractions of momenta are then integrated over, using *parton distribution functions* $f(x_i)$, which differ for the different partons. The fraction x_u for an up-type quark in a proton e.g., would be much larger than that for a top-type squark. This yields the total cross section

$$\sigma_{q_1 q_2} = \int f(x_1) f(x_2) \sigma(\hat{s}) dx_1 dx_2, \tag{3.3}$$

where σ is the partonic cross section. In this project the CTEQ6 parton distribution functions from the LHAPDF Fortran library [**?**] are used.

3.1.2 Luminosity

Another important concept to be introduced is *luminosity*. The luminosity $\mathcal{L}$ is a measure on how many collisions happen at an accelerator per time, and is here given in inverse femto-barn $\text{fb}^{-1} = 10^{43}\ \text{m}^{-2}$. This can be used to set limits on the size of cross sections, by considering the following relation for a process where a particle A is produced

$$n_A = \mathcal{L} \sigma_A, \tag{3.4}$$

where $\mathcal{L}$ is the luminosity, σ_A is the cross section for the production of A, and n_A is the number of produced particles. Setting $n = 1$ as the limit for a single produced particle, and using $\mathcal{L} = 20.3\ \text{fb}^{-1}$, which is the luminosity for the dataset used here, the lower limit on cross sections is

$$\sigma = \frac{1}{20.3\ \text{fb}^{-1}} \approx 0.05\ \text{fb}. \tag{3.5}$$

Therefore, cross sections below $\sigma \sim \mathcal{O}(10^{-3})$, which correspond 0.02 produced particles, will be less important in this thesis.

3.2 Phenomenology

In order to know what to look for at the LHC it is necessary to consider the phenomenology of supersymmetry. This is done by first revisiting the soft supersymmetry breaking, in order to consider models with more constraints than the 124 parameters of the MSSM. *Hidden sector* (HS) scalar superfields X have very small or no interactions with the MSSM fields. Their couplings are usually supressed by some mass scale so that $\mathcal{L} \sim M^{-1}$. They have an effective (non-renormalizable) interaction with the scalar superfields of the form

$$\mathcal{L}_{HS} = -\frac{1}{M}(\bar{\theta}\bar{\theta})X\Phi_i\Phi_j\Phi_k. \tag{3.6}$$

If these sectors develope a vacuum expectation value $\langle X \rangle = \theta\theta \langle F_X \rangle$ for the auxilliary field F (auxilliary meaning that it can be removed using the Euler-Lagrange equations), supersymmetry is broken. The two most commonly known hidden sectors are Planck-scale mediated supersymmetry breaking (PMSB) and Gauge mediated supersymmetry breaking (GMSB), the first of which is obtained by blaming some gravity mechanism for mediating the breaking of SUSY from the hidden sector. The breaking scale is then the Plank scale, $M = M_P = 2.4 \cdot 10^{18}$ GeV. The vev is restricted to $\sqrt{\langle F \rangle} \sim 10^{10} - 10^{11}$ GeV in order not to reintroduce the hierarchy problem. The complete soft terms can be shown to be [1]

$$\begin{aligned}\mathcal{L}_{soft} = &-\frac{\langle F_X \rangle}{M_P}\Big(\frac{1}{2}f_a\lambda^a\lambda^a + \frac{1}{6}y'_{ijk}A_iA_jA_k + \frac{1}{2}\mu'_{ij}A_iA_j + \frac{\langle F_X \rangle^*}{M_P^2}x_{ijk}A_i^*A_jA_k + c.c.\Big) \\ &- \frac{|\langle F_X \rangle|^2}{M_P}k_{ij}A_iA_j^*.\end{aligned}$$

By simplifying as much as possible, all soft terms are fixed by just four parameters. This model is called minimal supergravity (mSUGRA) or the constrained minimal supersymmetric Standard Model (CMSSM). The CMSSM parameters are

$$m_{1/2} = f\frac{\langle F_X \rangle}{M_P}, \qquad m_0^2 = k\frac{|\langle F_X \rangle|}{M_P^2}, \qquad A_0 = \alpha\frac{\langle F_X \rangle}{M_P}, \qquad B_0 = \beta\frac{\langle F_X \rangle}{M_P}, \tag{3.7}$$

or $m_{1/2}$, m_0, A_0, $\tan\beta$ and sgnμ.

The second hidden sector is the aforementioned Gauge mediated supersymmetry breaking. In this case the soft terms come from loop diagrams with *messenger superfields* that get their own mass from coupling to the HS SUSY breaking vev, and have SM interactions. This model is parameterized by

$$\Lambda = \frac{\langle F \rangle}{M_{messenger}} \ , \ M_{messenger} \ , \ N_5 \ , \ \tan\beta, \tag{3.8}$$

where $M_{messenger}$ is the mass scale and N_5 (FYLL INN HER).

3.2.1 Searches For Supersymmetry

Supersymmetry at hadron colliders will be in the form of QCD processes with relatively small masses, as the incoming particles are quarks and gluons and QCD cross sections will therefore be the largest. The traces left by such processes will be closely related to the conservation of R-parity, in that all sparticles are produced in pairs and eventually decay to the LSP. If the LSP is indeed only weakly interacting, this makes it very difficult to detect. An indirect way of detecting it, however, is by looking for *missing transverse energy* $\not{E}_T$. The energy is only considered in the transverse plane, as the longitudinal momentum is difficult to predict in the hadronic case (see the above discussion of hadron colliders). To look for $\not{E}_T$ the *effective mass* is defined as

$$M_{eff} = \sum p_T^{jet} + \not{E}_T, \tag{3.9}$$

and used to search for deviations from the SM expectations. Here p_T^{jet} is the transverse momentum of jets. The hadronic jets produced from supersymmetric processes can be very complicated, such as the squark-squark production shown in Fig. (3.1). If all final state particles have low momenta, these are called soft particles. The very large background from SM processes provides another difficulty in searching for supersymmetry. An example generated with ATLAS Open Data is shown in Fig. (3.2), where the supersymmetric signals (pink lines) are all below the SM background, and so it is impossible to detect any deviation from SM background.

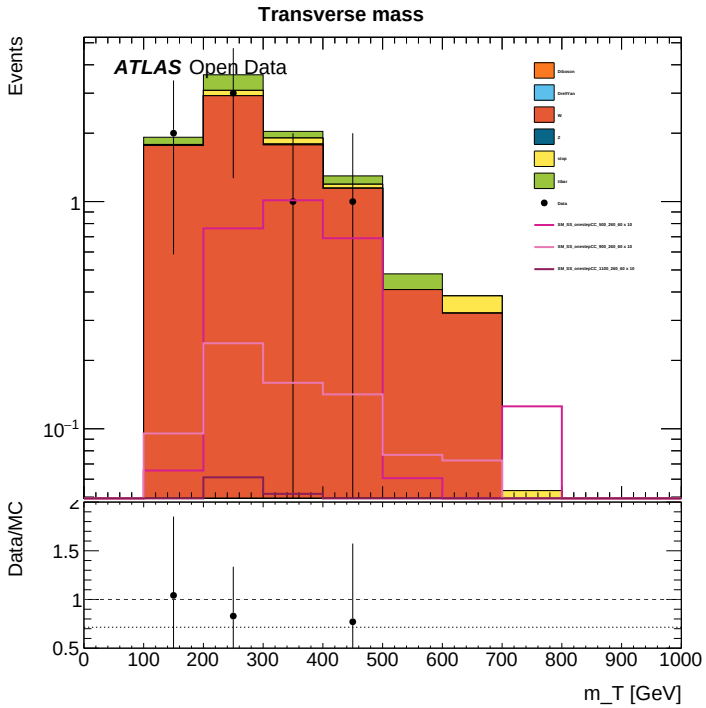


Figure 3.2: Analysis of ATLAS data, where the pink lines are supersymmetric signals for $m_{\tilde{q}} = 500$ (magenta), $m_{\tilde{q}} = 900$ (light pink) and $m_{\tilde{q}} = 1100$ (purple). The signal never exceeds the SM background, in this case Diboson (orange), Drell Yan (light blue), W (dark orange), Z (dark blue), $\tilde{t}$ (yellow) and $t\bar{t}$ (green), making it impossible to distinguish events from background. Analysis using 1 fb^{-1}.

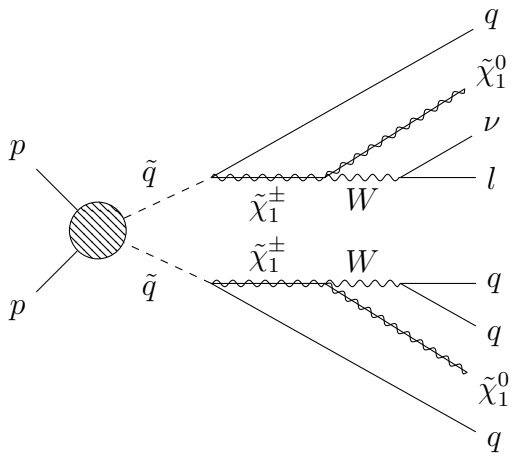


Figure 3.1: Possible signature of a supersymmetric QCD process, with four quark jets, a lepton and large missing transverse energy in the final state.

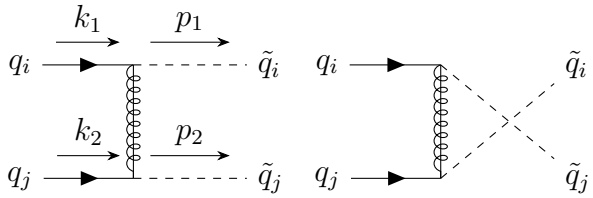


Figure 3.3: Feynman diagrams for squark pair production in quark quark collisions, both t and u diagram. Note that the u-channel (right diagram) is only possible for $i = j$.

Allowing the violation of R-parity opens up for more possibilities. Then the LSP can decay, and sparticles can be produced one at a time. It is possible to have *massive metastable charged particles* (MMCPs), which are typical for scenarios with a gravitino LSP [1].

3.2.2 Current Bounds on Sparticles

Searches and limits are constantly being updated, but here are a few results from the LHC Run I at 8 TeV, with up to 20 fb^{-1} luminosity. The bounds from the missing energy channel at 8 TeV for mSUGRA from ATLAS are $m_{\tilde{q}} > 1600$ GeV and $m_{\tilde{g}} > 1100$ GeV [1].

3.3 Squark-Squark Cross Section

In this thesis the relevant QCD process will be the production of squark pairs in quark-quark collisions,

$$q_i q_j \to \tilde{q}_i \tilde{q}_j, \tag{3.10}$$

where i, j are the 4 light quark flavours u, d, s, c which can be equal or different. Top and bottom are not considered here because of their large masses. Feynman diagrams of the leading order for this process can be found in Fig. (3.3). For equal flavour quarks there are two diagrams contributing: the t- and u-channel, while for different flavours only the t-channel contributes.

3.3.1 Calculation to LO

The cross section for squark production is here calculated to leading order. The momentum of the gluino is denoted p in the calculations, and defined as $p = k_2 - p_2$ for the t-channel and $p = k_2 - p_1$ for the u-channel. The chiralitites are decided later, and are simply denoted as P and P' for now. The Feynman

gauge is used, and all quark masses are set to zero. The expression for the matrix element of the pure t-channel becomes (reading direction is from q_j to q_i)

$$\begin{aligned} i\mathcal{M} &= \bar{v}(k_1)\Big(-i\sqrt{2}gP(t_a)^{ij}\Big) \times \delta^{ab}\frac{i}{\not{p}-m_{\tilde{g}}} \times \Big(-i\sqrt{2}gP'(t_b)^{lk}\Big) \times u(k_2) \\ &= -(t_a)^{ij}(t^a)^{lk} \times \frac{i2g^2}{t_g^2} \times \bar{v}(k_1)P(\not{p}+m_{\tilde{g}})P'u(k_2), \end{aligned}$$

where the color factor has been factored out. Some Mandelstam variables have been used to clean up the expresson; $t = p^2 = (k_2 - p_2)^2$, $t_g = t - m_{\tilde{g}}^2$. The matrix element squared is then

$$|\mathcal{M}_t|^2 = (t_a)^{ij}(t^a)^{lk}(t_b)^{mn}(t^b)^{op} \times \frac{4g^4}{t_g^2}\big(\bar{v}(k_1)P(\not{p}+m_{\tilde{g}})P'u(k_2)\big)\big(\bar{u}(k_2)P(\not{p}+m_{\tilde{g}})P'v(k_1)\big)$$

Averageing over spin

$$\sum |\mathcal{M}|^2 = A_{color,t} \times \frac{4g^4}{t_g^2}\mathrm{tr}\big[\not{k}_1 P(\not{p}+m_{\tilde{g}})P'\not{k}_2 P(\not{p}+m_{\tilde{g}})P'\big],$$

where $A_{color,t} = (t_a)^{ij}(t^a)^{lk}(t_b)^{mn}(t^b)^{op}$. The quark masses are set to zero, as they are small compared to $m_{\tilde{q}}$ and $m_{\tilde{g}}$. Inserting the different combinations of chiralities yield the traces

Different chiralities $P = P_{R/L}$, $P' = P_{L/R}$

$$\mathrm{tr}\big[\not{k}_1 P_{R/L}(\not{p}+m_{\tilde{g}})P_{L/R}\not{k}_2 P_{R/L}(\not{p}+m_{\tilde{g}})P_{L/R}\big] = 2\big(2(p\cdot k_2)(k_1\cdot p) - p^2(k_1\cdot k_2)\big). \tag{3.11}$$

Equal chiralities $P = P_{R/L}$, $P' = P_{R/L}$

$$\mathrm{tr}\big[\not{k}_1 P_{R/L}(\not{p}+m_{\tilde{g}})P_{R/L}\not{k}_2 P_{R/L}(\not{p}+m_{\tilde{g}})P_{R/L}\big] = 2m_{\tilde{g}}^2(k_1\cdot k_2)$$

where $P_{R/L}P_{R/L} = P_{R/L}$, $(\gamma^5)^2 = 1$ and $\mathrm{tr}[\gamma^5\gamma^\mu\gamma^\nu] = 0$. The expression in (3.11) is simplified to

$$2(2(p\cdot k_2)(k_1\cdot p) - p^2(k_1\cdot k_2)) + 2m_{\tilde{g}}^2(k_1\cdot k_2) = 2\big[t_1u_1 - s(m_{\tilde{q}}^2 - m_{\tilde{g}}^2)\big],$$

where

$$\begin{aligned} &s = (k_1+k_2)^2 = 2k_1\cdot k_2, && t_1 = (k_2-p_2)^2 - m_{\tilde{q}}^2, \quad t_g = (k_2-p_2)^2 - m_{\tilde{g}}^2, \\ &t = (k_2-p_2)^2 = m_{\tilde{q}}^2 - 2(k_2\cdot p_2), && u_1 = (k_1-p_2)^2 - m_{\tilde{q}}^2, \quad u_g = (k_1-p_2)^2 - m_{\tilde{g}}^2, \\ &u = (k_1-p_2)^2 = m_{\tilde{q}}^2 - 2(k_1\cdot p_2), \end{aligned}$$

Note that for equal quark flavors, the two possibilities for different chiralities are in fact equal, and so there should be a factor of 1/2 there, which yields

$$\sum |\mathcal{M}|^2 = (t_a)^{ij}(t^a)^{lk}(t_b)^{mn}(t^b)^{op} \times \frac{4g^4}{t_g^2}\Big[\delta_{ij}\big(\frac{1}{2}(t_1u_1 - sm_{\tilde{q}}^2) + sm_{\tilde{g}}^2\big) + (1-\delta_{ij})\big(t_1u_1 - s(m_{\tilde{q}}^2 - m_{\tilde{g}}^2)\big)\Big].$$

The color factor is

$$(t^a)^{ij}(t_a)^{kl}(t^b)_{ij}(t_b)^{kl} = \frac{1}{2}NC_F,$$

where $C_F = \frac{N^2-1}{2}$. Adding a factor 2 from the sum over different chiralities, the full expression for the t-channel is

$$\sum |\mathcal{M}_t|^2 = NC_F\frac{4g^4}{t_g^2}\Big[\delta_{ij}\big(\frac{1}{2}(t_1u_1 - sm_{\tilde{q}}^2) + sm_{\tilde{g}}^2\big) + (1-\delta_{ij})\big(t_1u_s - s(m_{\tilde{q}}^2 - m_{\tilde{g}}^2)\big)\Big]$$

The expression for the u-channel diagram is identical, but for the exchange of $t_g^2 \to u_g^2$. The u-channel only contributes in the case where $i = j$, so the matrix element is

$$\sum |\mathcal{M}_u|^2 = \delta_{ij}NC_F\frac{4g^4}{u_g^2}\Big[\frac{1}{2}(t_1u_1 - sm_{\tilde{q}}^2) + sm_{\tilde{g}}^2\Big].$$

Cross term

The ut cross term has the matrix element

$$i\mathcal{M}_{ut} = A_{color,ut}\frac{-i2g^2}{t_g}\big(\bar{v}(k_1)P(\not{p} + m_{\tilde{g}})P'u(k_2)\big)\frac{i2g^2}{u_g}\big(\bar{u}(k_2)P(\not{p} + m_{\tilde{g}})P'v(k_1)\big)$$

$$\sum \mathcal{M}_{ut} = A_{color,ut}\frac{4g^4}{u_gt_g}\big(\not{k}_1P(\not{k}_2 - \not{p}_2 + m_{\tilde{g}})P'\not{k}_2P(\not{k}_2 - \not{p}_1 + m_{\tilde{g}})P'\big)$$

Average over spins

$$\sum \mathcal{M}_{ut} = \delta_{ij}A_{color,ut}\frac{4g^4}{u_gt_g}\big(-t_1u_1 + m_{\tilde{q}}^2s + m_{\tilde{g}}^2s\big)$$

The other cross term – tu – is similar, but yields a slightly different combination

$$\sum i\mathcal{M}_{tu} = \delta_{ij}A_{color,ut}\frac{4g^4}{u_gt_g}\big(u_1t_1 - m_{\tilde{q}}^2s + m_{\tilde{g}}^2s\big).$$

Adding the cross terms and summing over the possible chiralities $P_R P_R$ and $P_L P_L$ then gives

$$\sum |\mathcal{M}_{ut+tu}| = \delta_{ij} A_{color,ut} \frac{8g^4}{u_g t_g} \big(2m_{\tilde{g}}^2 s\big).$$

The color factor is

$$\sum_{a,b} (t^a)^{ij} (t_a)^{kl} (t^b)_{ik} (t_b)_{jl} = -\frac{1}{2} C_F.$$

Combination of the matrix elements gives

$$\begin{aligned}\sum |\mathcal{M}|^2 =& \delta_{ij} \left[2g^4 N C_F (u_1 t_1 - s m_{\tilde{q}}^2) \big(\frac{1}{t_g^2} + \frac{1}{u_g^2}\big) + 4g^4 s m_{\tilde{g}}^2 \Big(N C_F \big(\frac{1}{t_g^2} + \frac{1}{u_g^2}\big) - 2 C_F \frac{1}{u_g t_g} \Big) \right] \\ &+ (1 - \delta_{ij}) \left[4g^4 N C_F \frac{u_1 t_1 - s(m_{\tilde{g}}^2 - m_{\tilde{g}}^2)}{t_g^2} \right].\end{aligned}$$

Cross section

The cross section for the process $q_i q_j \to \tilde{q}_i \tilde{q}_j$ is [**?**]

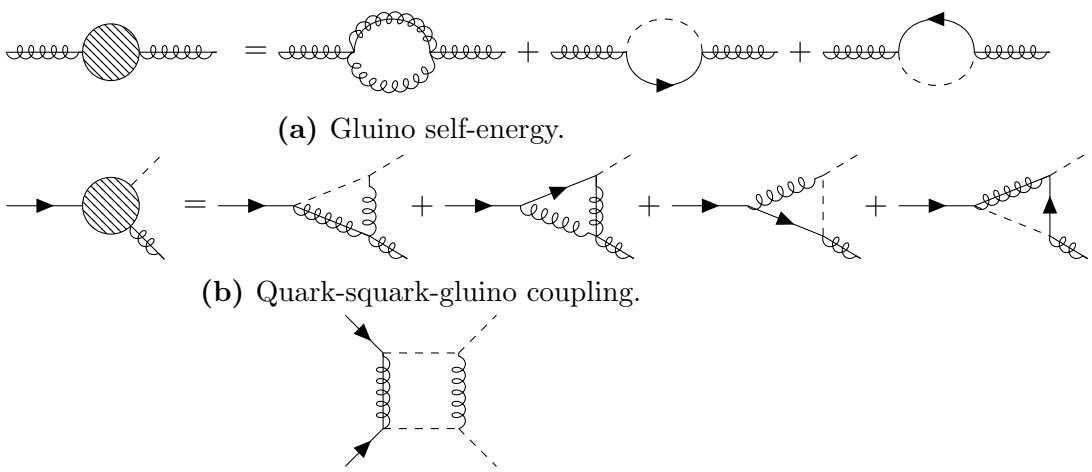
$$\sigma^B = \frac{\pi \alpha_s^2}{s} \left[\beta_{\tilde{q}} \Big(-\frac{4}{9} - \frac{4m_-^4}{9(m_{\tilde{g}}^2 s + m_-^4)} \Big) + \Big(-\frac{4}{9} - \frac{8m_-^2}{9s} \Big) L_1 \right] + \delta_{ij} \frac{\pi \alpha_s^2}{s} \left[\frac{8 m_{\tilde{g}}^2}{27(s + 2m_-^2)} L_1 \right],$$

where

$$L_1 = \ln\Big(\frac{s + 2m_-^2 - s\beta_{\tilde{q}}}{s + 2m_-^2 + s\beta_{\tilde{q}}}\Big), \quad \beta_{\tilde{q}} = \sqrt{1 - \frac{4m_{\tilde{q}}^2}{s}}, \quad m_-^2 = m_{\tilde{g}}^2 - m_{\tilde{q}}^2, \quad \alpha_s = \frac{g_s^2}{4\pi}.$$

3.4 NLO Corrections

The next-to-leading order terms contain virtual corrections to the Born level (tree-level) Feynman digrams. By allowing more complicated feynman diagrams, virtual particles can appear in the process. These never make it to the final state and become real particles, but are instead absorbed by the process. Since they are never real, or on-shell, they can have any momentum and mass. Thus, it is necessary to integrate over all possible momenta, which leads to divergences and infinities. There are three main categories of divergences, namely ultraviolet, infrared and collinear. Ultraviolet divergences appear when the integral is over an energy without bounds, thereby adding infinities. When the inverse problem occurs, that low momentum gives infinities, this is called an infrared divergence.



(a) Gluino self-energy.

(b) Quark-squark-gluino coupling.

(c) Box diagram.

Figure 3.4: A selected set of Feynman diagrams for the virtual corrections of $q_i q_j \to \tilde{q}_i \tilde{q}_j$ [**?**].

Collinear divergences are also called mass singularities, and stem from exactly that – masses going to zero.

In order to include the corrections and still get physical cross sections, it is necessary to integrate out the divergences using dimensional regularization and renormalise parameters. Renormalising a parameter means giving it a scale dependence, and baking the infinities into this expression. Physically, this can be interpreted as giving the electromagnetic coupling constant a length scale (energy scale^{-1}) dependence. Then the electromagnetic coupling becomes stronger the closer to a charged particle you get. These calulations are often very complicated and tedious. Some of the virtual corrections for $q_i q_j \to \tilde{q}_i \tilde{q}_j$ are shown in Fig. (3.4).

So, why should we calculate these cumbersome NLO terms? Firstly, the leading order terms are highly dependent on the a priori unknown renormalization scale, leading to a large uncertainty in theoretical predictions (up to a factor of two). Adding higher order terms reduces the scale dependency. An example is shown in Fig. (3.5), where the LO and NLO cross sections for $qq \to \tilde{q}\tilde{q}$ are plotted as a function of the renormalization/factorization scale [**?**]. The NLO terms are clearly less dependent on the scale. Secondly, the NLO contributions are expected to be large and positive, thereby raising the cross sections significantly and allowing for higher bounds on the gluino and squark masses [**?**].

The calculations from [**?**] assume degenerate squark masses $m_{\tilde{q}}$, set the 5 lightest quark masses to zero as they are much lighter than the squarks, and the top quark mass to $m_t = 175$ GeV. The two free parameters left are then $m_{\tilde{g}}$ and $m_{\tilde{q}}$. This indicates that these might be good features for the learning. The renormalization scheme used is the $\bar{M}S$ scheme. Masses and couplings are

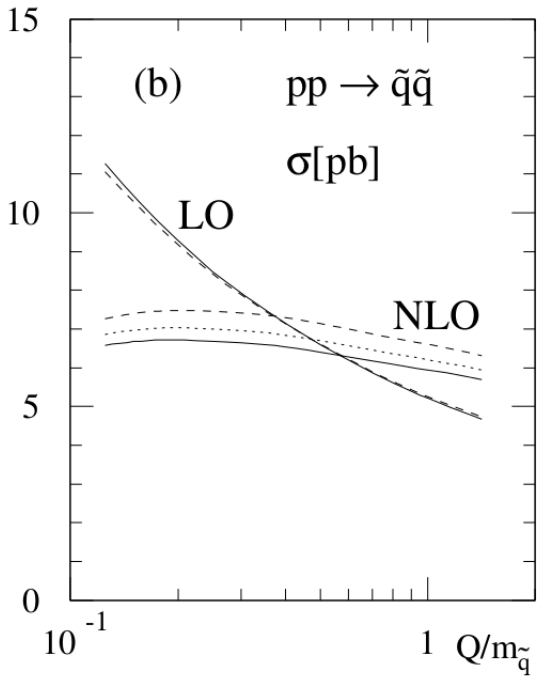


Figure 3.5: The dependence on the renormalization/factorization scale Q of the LO and NLO cross sections for squark-squark production at the LHC ($\sqrt{s} = 14$ TeV). Parton densities are GRV94 (solid), CTEQ3 (dashed) and MRS(A') (dotted). Mass parameters are $m_{\tilde{q}} = 600$ GeV, $m_{\tilde{g}} = 500$ GeV and $m_t = 175$ GeV. Figure from [**?**].

renormalized, and the resulting parameters can be found in [**?**].

Partonic Cross-Section

As previously discussed, calculations for hadron colliders require first finding the hadronic cross sections. In order to analyze these, scaling functions are introduced [**?**]

$$\hat{\sigma}_{ij} = \frac{\alpha_s^2(Q^2)}{m^2}\Big\{ f_{ij}^B(\eta, r) + 4\pi\alpha_s(Q^2)\left[f_{ij}^{V+S}(\eta, r, r_t) + f_{ij}^H(\eta, r) + \bar{f}_{ij}(\eta, r)\log\left(\frac{Q^2}{m^2}\right)\right]\Big\}, \tag{3.12}$$

where Q^2 is the mass scale, often set to $Q^2 = m^2$, and $m^2 = (\sqrt{p_1^2} + \sqrt{p_2^2})/2$ is the mass of the final particles. The scaling functions f are as follows: the Born term f^B, the sum of virtual and soft-gluon corrections f^{V+S}, the hard gluon corrections f^H, and the scale-dependent contributions $\bar{f}$. The partonic cross section depends on the parameters

$$\eta = \frac{s}{4m^2} - 1, \qquad r = \frac{m_{\tilde{g}}^2}{m_{\tilde{q}}^2}, \qquad r_t = \frac{m_t^2}{m^2}, \tag{3.13}$$

which may be used in deciding the target and features for the learning process.

Hadronic Cross-Section

As per the previous discussion, the partonic cross-sections must be integrated over in order to obtain the total hadronic cross section. A convolution integral over the parton distribution functions yields the expression

$$\sigma(S, Q^2) = \sum_{i,j=q,q,\bar{q}} \int_\tau^1 dx_1 \int_{\tau/x_1}^1 dx_2 f_i^{h_1}(x_1, Q^2) f_j^{h_2}(x_2, Q^2)\hat{\sigma}_{ij}(x_! x_2 S, Q^2)\Big|_{\tau=4m^2/S}. \tag{3.14}$$

The integrals are calculated numerically using the VEGAS integration routine [**?**] in [**?**]. The uncertainty due to different parameterizations of the parton densities for the LHC calculations of the NLO terms amount to $\lesssim 13\%$ at the central scale.

K-Factor

To quantify the improvement on the cross section found by adding NLO terms, the *K-factor* is introduced. It is the ratio between the cross sections

$$K = \sigma_{NLO}/\sigma_{LO}. \tag{3.15}$$

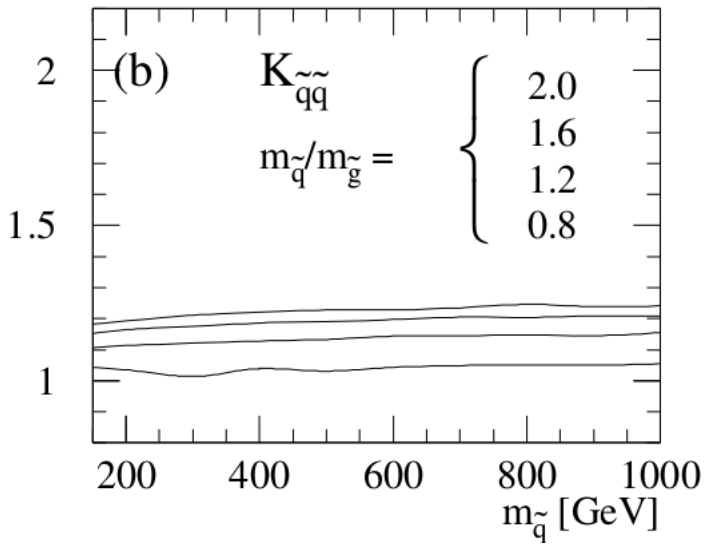


Figure 3.6: The K-factors for the LHC ($\sqrt{s} = 14$ TeV). Parton densities are GRV94, with scale $Q = m$, and the top squark mass is $m_t = 175$ GeV. Figure from [**?**].

The K-factor for squark production for varying mass ratios $m_{\tilde{q}}/m_{\tilde{g}} = 2.0, 1.6, 1.2, 0.8$ at the LHC is shown in Fig. (3.6). As seen from the figure, the K-factor is larger than 1 and quite stable as a function of the squark mass. This indicates a shift in the mass spectrum, and provides stronger bounds on particle masses.

3.4.1 State-of-the-art Tools

There are several numerical tools available for the calculation of NLO SUSY cross sections. Two of them are here considered, namely Prospino [**?**] and NLL-fast [**?**].

Prospino

A tool that uses the K-factor to calculate NLO cross sections is `Prospino 2.1` [**?**]. `Prospino 2.1` has the advantage of calculating cross sections with *non-degenerate* squark masses. This is done by first computing LO cross sections for the 5 or 4 lightest squark flavors, saved in the LO_ms parameter. The LO and NLO rates are then calculated for a mean value of the squark mass, and the corresponding K-factor is calculated. These values are given as LO and NLO. The LO_ms values are then multiplied by the K-factor, giving an approximation to the NLO terms for non-degenerate masses, given as NLO_ms. The calculations are outlined in the section above, and are described in more detail in [**?**]. This does mean, however, that calculations take very long. Evaluating the strong process cross sections for just a single CMSSM benchmark point takes about 15

minutes of CPU time on a modern processor [?]. In addition, as was discovered during the project, `Prospino 2.1` has a weakness in the heavy squark mass space. Consider the cross section for the production of $\tilde{c}_L\tilde{c}_L$. When all squark masses are heavy, but $m_{\tilde{c}_L}$ is slightly lighter than the others, the K-factor is set to zero, possibly due to a numerical error. This means that $LO \neq 0$ and $NLO = 0$, which gives problematic outlier points that cause trouble during learning.

NLL-fast

`NLL-fast 2.1` [?, ?, ?, ?, ?] computes the hadronic cross sections of gluino and squark pair production including NLO supersymmetric QCD corrections, and the resummation of soft gluon emission at next-to-leading-logarithmic (NLL) accuracy. It also provides a theoretical error estimation, based on the errors from scale dependence and the parton distribution functions. The calculation is done by reading in LO and NLO+NLL results, the scale uncertainty and the pdf and alphas error, and uses fast interpolation to calculate cross sections for any squark and gluino mass approximately within the range $[200, 1500]$ GeV. This does mean, however, that `NLL-fast 2.1` only calculates cross sections for mass-degenerate squarks.

Accuracy

Including the NLO cross sections has reduced the theoretical error to be as low as 10% for a wide range of processes and masses, see the discussion in [?]. There are other uncertainties, however, such as the ones from the PDFs and α_s. These must also be included in the total error estimate. PDFs are based on experimental data, and so the uncertainty increases with the sparticle masses because the PDFs are most poorly constrained at large scales and at large partons x.

To illustrate, consider both the gluino and (degenerate) squark mass to be 1.5 TeV, which is at the edge of what the LHC is able to produce at 8 TeV. `NLL-fast 2.1` then gives errors of $(+24.3\%, -22.2\%)$ for the PDFs and $(+8.3\%, -7.3\%)$ for α_s, when using the MSTW2008 NLO PDF set [?]. In this case the total error will not be much lower than 25%.

Thus, the relative errors obtained in this thesis should not exceed 10%, in order to improve on the existing methods.

Bibliography

[1] Paul Batzing and Are Raklev. Lecture notes for fys5190/fys9190. 2017.

[2] Marc Peter Deisenroth and Jun Wei Ng. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.

[3] Anders Kvellestad. Chasing susy through parameter space. 2015.

[4] C. Patrignani et al. Review of Particle Physics. *Chin. Phys.*, C40(10):100001, 2016.

[5] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[6] Devinderjit Sivia and John Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.

[7] Steven Weinberg. *The Quantum Theory of Fields*, volume 1. Cambridge University Press, 1995.

Bibliography

[1] Paul Batzing and Are Raklev. Lecture notes for fys5190/fys9190. 2017.

[2] Marc Peter Deisenroth and Jun Wei Ng. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.

[3] Anders Kvellestad. Chasing susy through parameter space. 2015.

[4] C. Patrignani et al. Review of Particle Physics. *Chin. Phys.*, C40(10):100001, 2016.

[5] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[6] Devinderjit Sivia and John Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.

[7] Steven Weinberg. *The Quantum Theory of Fields*, volume 1. Cambridge University Press, 1995.

Appendix A: The Gaussian Distribution

Bibliography

[1] Paul Batzing and Are Raklev. Lecture notes for fys5190/fys9190. 2017.

[2] Marc Peter Deisenroth and Jun Wei Ng. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.

[3] Anders Kvellestad. Chasing susy through parameter space. 2015.

[4] C. Patrignani et al. Review of Particle Physics. *Chin. Phys.*, C40(10):100001, 2016.

[5] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[6] Devinderjit Sivia and John Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.

[7] Steven Weinberg. *The Quantum Theory of Fields*, volume 1. Cambridge University Press, 1995.

Appendix B: Benchmarks

3.5 When the Error is Known

Most observations are noisy versions of the true function values. In [5] there is an example using additive independent identically distributed Gaussian noise ε with variance σ_n^2 (and standard deviation σ_n). The benchmark function attempts to convert the noisy function at hand to the form

$$y = f(x) + \varepsilon \ , \ \varepsilon \sim \mathcal{N}(0, \sigma_n). \tag{3.16}$$

This may, according to 5.10, be done. Assume a relatively simple function which spans over several orders of magnitude, with a shape that can easily be modelled by the RBF kernel. The function used here is

$$f(x) = 1000 \exp\Big(- \frac{x^2}{10} \Big) \ , \ x \in [0, 30], \tag{3.17}$$

with a corresponding noisy function

$$y = f(x) + \varepsilon f(x), \tag{3.18}$$

where $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$.

A plot of the true function and noisy function, along with their logarithms, are shown in Fig. (3.7 -3.8). A large standard deviation for the error is used for illustrational purposes. The error is relatively much smaller for the log of functions, and more evenly distributed, as can be seen from the zoom in in Fig (3.9).

For the actual training and prediction, a standard deviation of $\sigma_{std} = 0.001$ was used (to get a similar number to the real case of the cross sections). A single Gaussian process was performed with 500 training points and 4000 test points, using the RBF kernel. The Gaussian process was first tested on the true function, and the error measure was the relative error given by

$$\epsilon_{GP} = \frac{y_{true} - y_{predicted}}{y_{true}}, \tag{3.19}$$

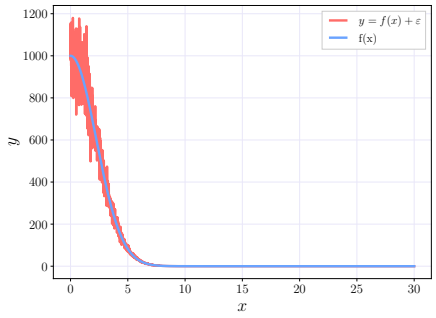


Figure 3.7: The pure and noisy function with $\varepsilon \sim \mathcal{N}(0, \sigma_{std} = 0.1)$.

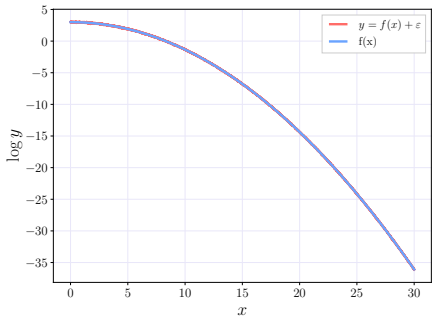


Figure 3.8: The logarithm of the pure and noisy function with $\varepsilon \sim \mathcal{N}(0, \sigma_{std} = 0.1)$.

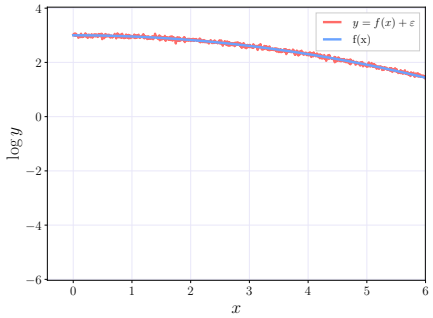


Figure 3.9: The logartihm of the pure and noisy function with $\varepsilon \sim \mathcal{N}(0, \sigma_{std} = 0.1)$.

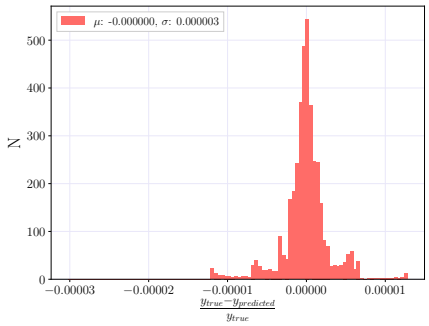


Figure 3.10: Error histogram of the Gaussian process prediction for the true function $f(x)$ using a noiseless RBF-kernel.

where $y_{predicted} = 10^{\log_{10} f_{predicted}}$, and $f_{predicted}$ is what the GP predicts. The optimal hyperparameters for the RBF kernel on the true function were

$$\text{kernel}_{true} = 3.16^2 \exp\Big(-\frac{x^2}{5.6^2}\Big),$$

with no noise. A histogram of the errors is shown in Fig. (3.10).

The prediction for the error is

$$\Big(\frac{\varepsilon}{\log 10}\Big)^2 = \frac{(1 \cdot 10^{-3})^2}{(\log 10)^2} = 1.886 \cdot 10^{-7}.$$

Setting $\alpha = 1.886 \cdot 10^{-7}$ gives the best prediction, with a kernel with very similar hyperparameters to the noise-free case

$$\text{kernel}_{noisy} = 3.16^2 \exp\Big(-\frac{x^2}{5.82^2}\Big). \tag{3.20}$$

It also improves the prediction significantly from the case where no noise is fed to the estimator. A plot of the case where $\alpha = 10^{-10}$ and $\alpha = 1.886 \cdot 10^{-7}$ are shown in Fig. (3.11) and Fig. (3.12), respectively.

The best value for α is in fact very close to our estimate, as can be seen from Fig. (). Note that $\alpha \propto 10^{-8}$ gives a smaller mean than $\alpha \propto 10^{-7}$, but it also has a much larger variance.

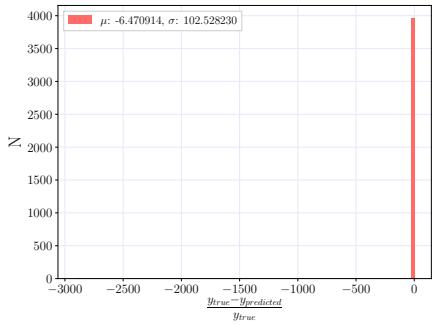


Figure 3.11: Error histogram of the Gaussian process prediction for the noisy function $y(x)$ using a noiseless RBF-kernel ($\alpha = 10^{-10}$).

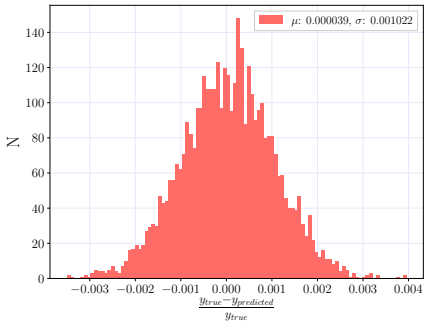


Figure 3.12: Error histogram of the Gaussian process prediction for the noisy function $y(x)$ using an RBF-kernel with $\alpha = 1.886 \cdot 10^{-6}$.

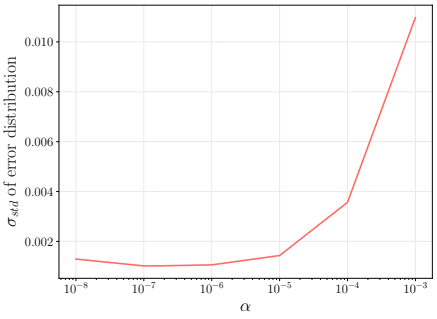


Figure 3.13: Standard deviation of error distribution of the Gaussian process prediction for a varying noise level variance α.

Chapter 4

Gaussian Processes

In this chapter Gaussian process regression is introduced. First, some concepts and expressions in Bayesian statistics are reviewed. The following section introduces the mathematical framework needed for Gaussian processes, before selected covariance functions are discussed. Concepts in Bayesian model selection are used as a basis to quantify and improve the quality of predictions. Finally, distributed Gaussian processes are introduced as a way of scaling Gaussian processes to larger datasets.

4.1 Introduction to Bayesian Statistics

There are two general philosophies in statistics, namely *frequentist* and *Bayesian* statistics. To understand where they differ, consider a statement statisticians from both branches consider to be true

Statisticians use probability to describe uncertainty.

The difference between Bayesian and frequentist statistics is at the definition of the *uncertain*. Since uncertainty is described by probability this must also vary, and one distinguishes between *objective* and *subjective* probability. Consider an example in which a statistician throws a dice. Before throwing, he is uncertain about the outcome of the dice toss. This uncertainty related to the outcome is *objective*: no one can know if he will throw a 1 or a 4. On the other hand, he might also be uncertain about the underlying probability distribution of the dice toss. Is the dice loaded? Is one of the edges sharper than the others? This uncertainty is *subjective*, as it may vary depending on how much information is available about the system. One of the main critiques of subjective probability posed by frequentists is that the final probability depends on who you ask.

4.1.1 Bayes' Theorem

To further illustrate the difference between frequentist and Bayesian statistics *Bayes' theorem* is introduced. Bayes' theorem can be derived from the familiar rules of probability

$$P(X|I) + P(\bar{X}|I) = 1, \tag{4.1}$$

$$P(X, Y|I) = P(X|Y, I) \times P(Y|I), \tag{4.2}$$

commonly known as the *sum rule* and *product rule*, respectively. $P(X|I)$ is the probability of outcome X given the information I, and $P(X|Y, I)$ is the probability of outcome X given the information I *and* outcome Y. The bar over $\bar{X}$ means that the outcome X does *not* happen. The sum rule states that the total probability of the outcomes X and $\bar{X}$ is equal to 1. This is rather untuitive, considering an event either takes place not. The second rule, the product rule, states that the probability of both outcomes X and Y is equal to the probability of Y times the probability of X given that Y has ocurred. These expressions combine to give Bayes' theorem, first formulated by reverend Thomas Bayes in 1763,

$$P(X|Y, I) = \frac{P(Y|X, I) \times P(X|I)}{P(Y|I)}. \tag{4.3}$$

This theorem states that the probability of X given Y equals the probability of Y given X times the probability of X, divided by the probability of Y. Surprisingly, there is nothing Bayesian about Bayes' theorem. It merely reformulates the rules of logical consistent reasoning stated by Richard Cox in 1946 [6]. Laplace was the one to make Bayes' theorem Bayesian, when he used the theorem to perform inference about distribution parameters. These are, for example, the mean and variance of a Gaussian distribution. The resulting expression is

$$P(\Theta = \theta|X = x) = \frac{P(X = x|\Theta = \theta)P(\Theta = \theta)}{P(X = x)}, \tag{4.4}$$

where Θ are the possible probability distribution parameters, X are the possible outcomes, $P(X = x)$ is a normalization constant called the *marginal likelihood*, and $P(X = x|\Theta = \theta)$ and $P(\Theta = \theta)$ are the *likelihood* and *prior*, respectively. In other words, Eq. (4.4) states the probability of the parameters θ given the knowledge of outcomes x.

A crucial parting of Bayesian statistics from frequentist statistics is at the introduction of the *prior*, which expresses a probability distribution on the *parameters* of the probability distribution.

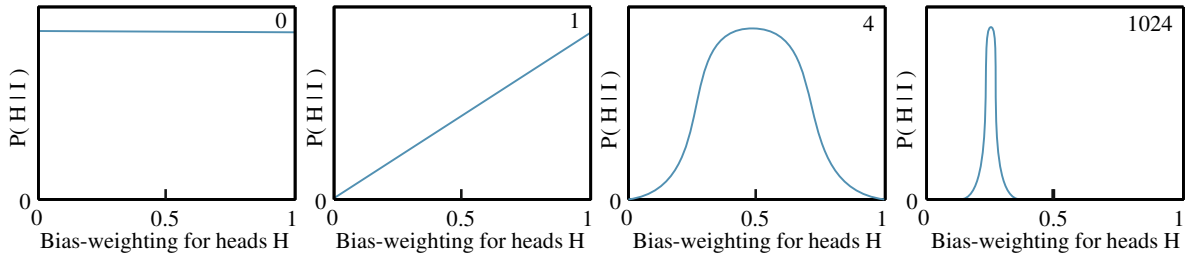


Figure 4.1: The posterior probability distribution of the bias-weighting of a coin for the uniform prior, $P(H|I)$. The first panel from the left is before the coin is tossed, the second panel is after 1 toss, the third is after 4 tosses, and the fourth is after 1024 tosses. The posterior converges towards a narrow peak at 0.25, so the coin is biased.

4.1.2 Priors and Likelihood

The likelihood $P(X = x|\Theta = \theta)$ is simply the probability of the observations x given the parameters of the probability distribution θ. Conversely, the prior expresses a prior belief or assumption of the data, and has to be determined beforehand. The measure $P(\Theta = \theta|X = x)$ from Eq. (4.4) is called the *posterior distribution*. This can be thought of as the prior belief, modified by how well this belief fits the data,

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{marginal likelihood}}.$$

Consider an example. The statistician from before now sets about tossing a coin. Before tossing he assumes the probability of all outcomes is equal, and so adopts a flat prior probability distribution. After one throw he gets heads, and the posterior changes to a function with high probability for heads, and low for tails. After 4 throws where two were heads and two were tails the posterior shows an equal probabilty for heads and tails, with a wide distribution centered at 0.5. After several throws the distribution converges to a narrow peak around 0.25, illustrated in Fig. (4.1). This indicates an unfair coin that is biased towards tails.

4.1.3 Best Estimate and Reliability

Given a posterior distribution it is important to decide how well it fits the data. For that purpose the *best estimate* and *reliability* are defined. The best estimate X_0 is the outcome with the highest probability. In other words, it is the maximum

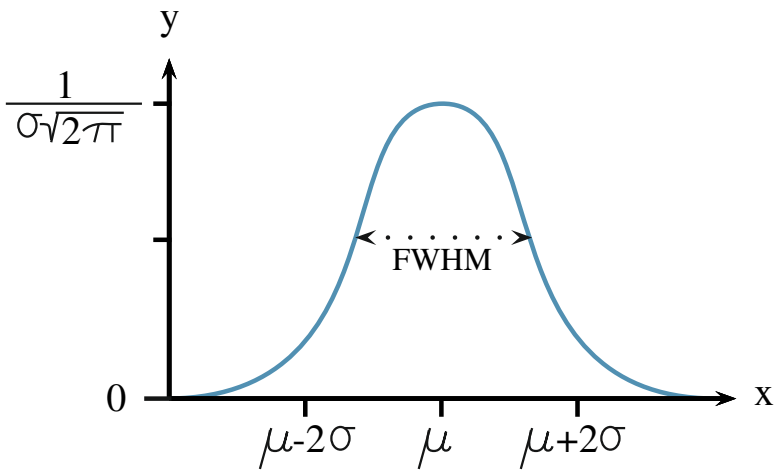


Figure 4.2: A Gaussian probability distribution. The maximum is at the mean value μ, with a full width at half maximum (FWHM) at around 2.35σ. Figure from [6].

of the posterior

$$\frac{dP}{dX}\Big|_{X_0} = 0, \qquad \frac{d^2P}{dX^2}\Big|_{X_0} < 0. \tag{4.5}$$

The second derivative must be negative to insure that X_0 is, in fact, a maximum. Equally important to finding a best estimate is knowing how reliable the estimate is. Reliability is connected the width of the distribution, or how much the data is smeared out. A narrow distribution has low uncertainty, while a wide distribution has large uncertainty. The width is found by Taylor expanding the posterior, and taking the logarithm [1]

$$L = L(X_0) + \frac{1}{2}\frac{d^2}{dx^2}L\Big|_{X_0}(X - X_0)^2 + ..., \quad L = \log_e\Big[\text{prob}(x|\{data\}, I)\Big] \tag{4.6}$$

This gives an approximate posterior in the shape of a *Gaussian distribution*, with mean and variance given by

$$\mathcal{N}(\mu, \sigma) \text{ where } \mu = X_0,\ \sigma = \Big(-\frac{d^2L}{dx^2}\Big)^{-1/2}. \tag{4.7}$$

The Gaussian distribution is symmetric with respect to the maximum at $x = \mu$, and has a full width at half maximum (FWHM) at around 2.35σ, as shown in Fig. (4.2).

[1] L is a monotonic function of P, so the maximum of L is at the maximum of P.

4.1.4 Covariance

Before embarking on Gaussian processes the concept of *covariance* is needed. A more detailed description is found in [6]. In many cases the equations are not as simple to solve as in Eq. (4.6), as the probability distribution might have several quantities of interest $\{X_i\}$. In that case, a set of *simultaneous equations* must be solved to get the best estimate

$$\frac{dP}{dX_i}\Big|_{X_{0j}} = 0. \tag{4.8}$$

To simplify expressions consider the problem in two dimensions, so that $\{X_i\} = (X, Y)$. The Taylor expansion of L is then

$$\begin{aligned} L =& L(X_0, Y_0) + \frac{1}{2}\Big[\frac{d^2L}{dX^2}\Big|_{X_0,Y_0}(X - X_0)^2 \\ &+ \frac{d^2L}{dY^2}\Big|_{X_0,Y_0}(Y - Y_0)^2 + 2\frac{d^2L}{dXdY}\Big|_{X_0,Y_0}(X - X_0)(Y - Y_0)\Big] + ... \end{aligned} \tag{4.9}$$

There are now four partial derivatives, reduced to three using the rules for mixed partial derivatives $\frac{\partial^2}{\partial X \partial Y} = \frac{\partial^2}{\partial Y \partial X}$. Writing the quadratic term of Eq. 4.9 in matrix form gives

$$Q = \begin{pmatrix} X - X_0 & Y - Y_0 \end{pmatrix} \begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} X - X_0 \\ Y - Y_0 \end{pmatrix}, \tag{4.10}$$

where the matrix elements are

$$A = \frac{\partial^2 L}{\partial X^2}\Big|_{X_0,Y_0}, \qquad B = \frac{\partial^2 L}{\partial Y^2}\Big|_{X_0,Y_0}, \qquad C = \frac{\partial^2 L}{\partial X \partial Y}\Big|_{X_0,Y_0}. \tag{4.11}$$

The *variance* is defined as the expectation value of the square of deviations from the mean. In the two-dimensional case this becomes [6]

$$\mathrm{Var}(X) = \sigma_x^2 = \big\langle (X - X_0)^2 \big\rangle = \int\int (X - X_0)^2 P(X, Y|\{\mathrm{data}\}, I) dX dY. \tag{4.12}$$

This is the variance σ_X^2 for X, and its square root is the standard deviation σ_X. A similar expression can be found for Y, by switching X and Y. It is also possible to find the simultaneous deviations of the parameters X and Y, or the correlation between the inferred parameters. This is called the *covariance* σ_{XY}^2, and is in two dimensions given by

$$\sigma_{XY}^2 = \langle (X - X_0)(Y - Y_0) \rangle = \int\int (X - X_0)(Y - Y_0) P(X, Y|\{\mathrm{data}\}, I) dX dY. \tag{4.13}$$

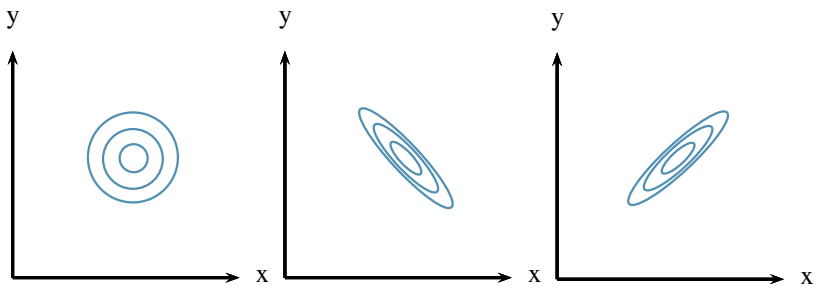


Figure 4.3: A schematic illustration of covariance and correlation. (a) The contours of a posterior pdf with zero covariance, where the inferred values of X and Y are uncorrelated. (b) The corresponding plot when the covariance is large and negative; $Y + mX =$ constant along the dotted line (where $m > 0$), emphasizing that only this sum of the two parameters can be inferred reliably. (c) The case of positive correlation, where we learn most about the difference YmX; this is constant along the dotted line.

The covariance indicates how an over- or underestimation of one parameter affects another parameter. If, for example, an overestimation of X leads to an overestimation of Y, the covariance is positive. If the overestimation of X has little or no effect on the estimation of Y, the covariance is negligible or zero $|\sigma_{XY}| \ll \sqrt{\sigma_X^2 \sigma_Y^2}$. These effects are illustrated in Fig. (4.3). It can be shown that [6]

$$\text{cov} = \begin{pmatrix} \sigma_X^2 & \sigma_{XY}^2 \\ \sigma_{XY}^2 & \sigma_Y^2 \end{pmatrix} = - \begin{pmatrix} A & C \\ C & B \end{pmatrix}^{-1}. \tag{4.14}$$

This is called the *covariance matrix*.

4.2 Gaussian Process Regression

Gaussian processes (GP) is a supervised machine learning method, designed to solve regression and probabilistic classification problems. In this thesis only regression will be used.

Consider a set of points $\mathcal{D} = \{\mathbf{x}_i, y_i\}$, where y is some (possibly noisy) function of $\mathbf{x}$, $y = f(\mathbf{x}) + \varepsilon$. This is illustrated by the black dots in in Fig. (4.4). In machine learning $\mathcal{D}$ is the *training data*, as it is used to train the model. It consists of *features*, which are the input vectors $\mathbf{x}_i$, and *targets*, which are the function values y_i. The set of points is discrete, so there is some $\mathbf{x}^*$ for which the target y^* is unknown. Gaussian Processes (GP) predict a Gaussian distribution *over function values* at this point $\mathbf{x}^*$, with a corresponding mean $m(\mathbf{x}^*)$ and variance σ^2. The GP prediction for the target value y^* is the mean $m(\mathbf{x}^*)$, with uncertainty σ^2.

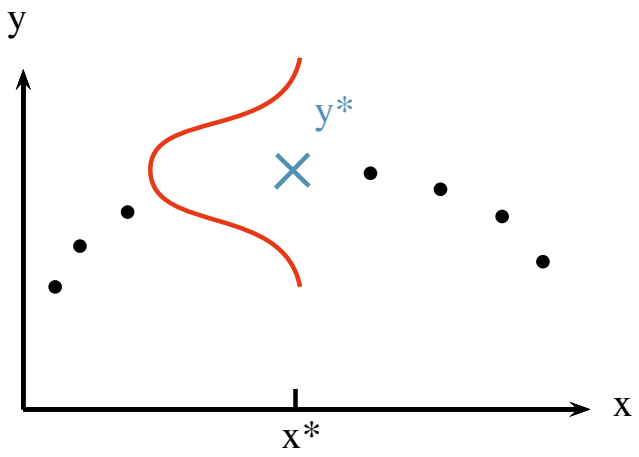


Figure 4.4: An illustration of a GP prediction of the target value y^* (blue cross), given the known set of points $\{x_i, y_i\}$ (black dots). The prediction is a Gaussian distribution in y with mean y^* and variance σ^2. The Gaussian distribution is drawn in red with y on the vertical axis, with uncertainty in the y-direction.

The predicted target y^* can be viewed as a linear combination of the known targets y_i, where the weights are controlled by the covariances between $\mathbf{x}_i$ and the test point $\mathbf{x}^*$.

Function Space View

Since Gaussian processes provide distributions over functions, it is useful to consider the problem in the function space view introduced in [5]. For a real process $f(\mathbf{x})$ the mean $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ are defined as

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{4.15}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \tag{4.16}$$

where $\mathbb{E}[a]$ is the expectation value of some quantity a. Given the mean and covariance, the Gaussian distribution for $f(\mathbf{x})$ is completely specified

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{4.17}$$

The covariance between two points $k(\mathbf{x}_i, \mathbf{x}_j)$ is decided by the *kernel function*. In this text the running example will be the squared exponential (SE) kernel, given by

$$k(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(-\frac{1}{2}|\mathbf{x}_i - \mathbf{x}_j|^2\Big). \tag{4.18}$$

Note that the covariance of the function values $f(\mathbf{x}_i)$, $f(\mathbf{x}_j)$ only depends on the input parameters $\mathbf{x}_i$, $\mathbf{x}_j$.

Specifying the covariance function specifies a distribution over functions [5]. This is because allowed functions must obey the correlation decided by $k(\mathbf{x}_i, \mathbf{x}_j)$. Using the kernel on a set of input vectors contained in the matrix X^*, gives the *covariance matrix*. Combined with an initial mean of zero [2] one obtains the *prior* distribution

$$f(x) \sim \mathcal{N}(0, K(X^*, X^*)). \tag{4.19}$$

This distribution encodes the prior knowledge about the function values $f(x)$, and so the choice of kernel is one of the most important steps in learning with GPs. The prior is modified by the training data to provide a posterior distribution. In Fig. (4.5) samples are drawn from both the prior and posterior distribution. Samples are here taken to mean functions that obey the covariance function. In the case of the prior, the samples are drawn from a distribution of functions with mean zero and constant variance, meaning that if you drew enough functions the mean value of all functions at every x would be zero. For the posterior, the mean values and uncertainties have been modified by the training data. In a point where there is training data the uncertainty is zero [3], and so all samples drawn from this distribution must pass through this point. Far away from training points the uncertainty is large.

Consider a simple example of a noise-free set of training points $\{\mathbf{x}_i, y_i\}$, so that $y = f(\mathbf{x})$. The joint distribution of training outputs $\mathbf{f}$ and test outputs $\mathbf{f}^*$ according to the prior is then

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X^*) \\ K(X,X^*) & K(X^*,X^*) \end{bmatrix} \right) \tag{4.20}$$

For n training points and n^* test points, $K(X,X)$ is the $n \times n$ matrix containing the covariance of training points, $K(X,X^*)$ is the $n \times n^*$ matrix of covariance between the test and training points, and $K(X^*,X^*)$ is the $n^* \times n^*$ matrix containing the covariance of test points. By conditioning the distribution of $\mathbf{f}^*$ on the observations, the posterior distribution over $\mathbf{f}^*$ is obtained[4] [5]

$$\mathbf{f}_* \big| X_*, X, \mathbf{f} \sim \mathcal{N}(K(X_*,X)K(X,X)^{-1}\mathbf{f}, \tag{4.21}$$

$$K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)). \tag{4.22}$$

Gaussian Noise Model

Noise-free observations are rare. In most cases targets will contain some noise $y = f(\mathbf{x}) + \varepsilon$, where the noise ε is assumed to follow a Gaussian distribution

[2] The mean does not have to be zero, it could for example be the mean of the training data.
[3] Assuming there is no noise in the data.
[4] For more details, see Appendix A.2 in [5].

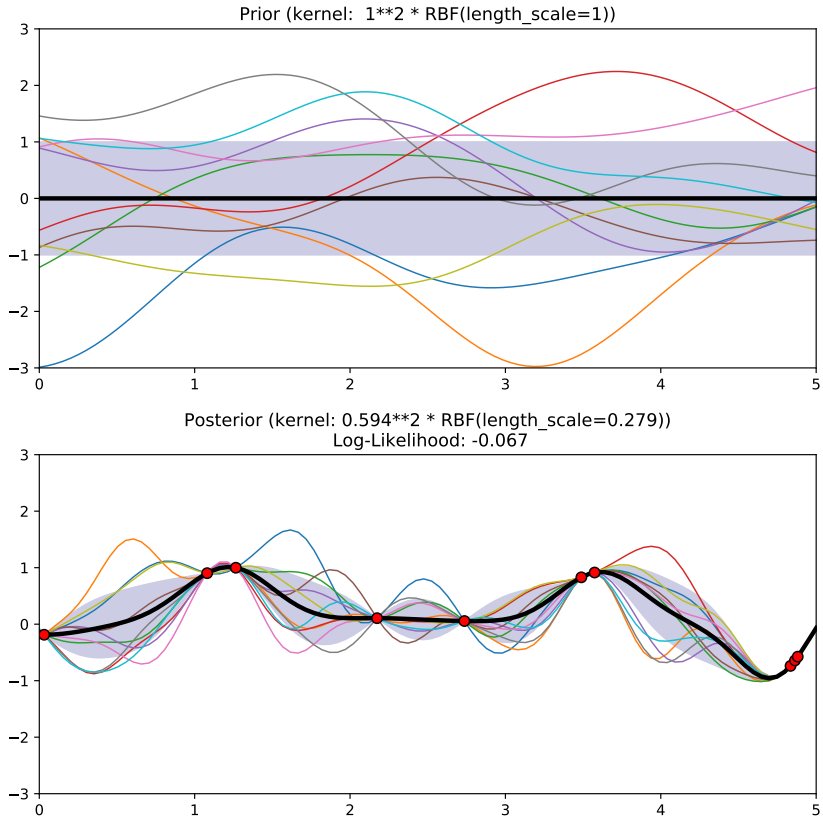


Figure 4.5: Drawing functions from the prior (top) and posterior (bottom) distributions. The thick, black line represents the mean of the distribution, while the shaded, blue area is the uncertainty. The multiple colored lines are functions drawn randomly from the prior and posterior distributions, whose correlation are dictated by the covariance function. The prior has mean 0 and covariance given by the squared exponential function. The posterior has been modified by training points (red dots), giving rise to zero uncertainty at the points where training data exists, and an altered mean value for the distribution. Figure generated using scikit-learn.

$\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$. This is the *Gaussian noise model*. The covariance can then be expressed as

$$\text{cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij} \tag{4.23}$$

which gives for the prior distribution

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 \mathbb{I} & K(X,X^*) \\ K(X,X^*) & K(X^*,X^*) \end{bmatrix}\right). \tag{4.24}$$

The conditioned distribution is then

$$\mathbf{f}^* \big| X^*, X, \mathbf{f} \sim \mathcal{N}(\bar{\mathbf{f}}^*, \text{cov}(\mathbf{f}^*)), \text{ where} \tag{4.25}$$

$$\bar{\mathbf{f}}^* = K(X^*,X)[K(X,X) + \sigma_n^2 \mathbb{I}]^{-1}\mathbf{y}, \tag{4.26}$$

$$\text{cov}(\mathbf{f}^*) = K(X^*,X^*) - K(X^*,X)[K(X,X) + \sigma_n^2 \mathbb{I}]^{-1} K(X,X^*) \tag{4.27}$$

For the sake of tidying up the expression define the matrix $K \equiv K(X,X)$ and the matrix $K^* \equiv K(X,X^*)$. In the case of a single test point $\mathbf{x}^*$ the matrix K^* is written as a vector $\mathbf{k}(\mathbf{x}^*) = \mathbf{k}^*$. Using this compact notation the GP prediction for a single test point $\mathbf{x}^*$ is

$$\bar{f}^* = \mathbf{k}^{*T}(K + \sigma_n^2 \mathbb{I})^{-1}\mathbf{y}, \tag{4.28}$$

$$\mathbb{V}[f^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^{*T}(K + \sigma_n^2 \mathbb{I})^{-1}\mathbf{k}^*. \tag{4.29}$$

Note that the predicted mean value $\bar{f}^*$ can be viewed as a linear combination of y_i of the form $\alpha \mathbf{y}$, where $\alpha = \mathbf{k}^{*T}(K + \sigma_n^2 \mathbb{I})^{-1}$. α then only contains the covariance between features.

Eqs. (4.28)-(4.29) form the basis for GP prediction in `scikit-learn` [**?**]. The algorithm is shown in Algorithm (3), and uses the Cholesky decomposition of the covariance matrix.

Data: X (inputs), $\mathbf{y}$ (targets), k (covariance function/kernel), σ_n^2 (noise level), $\mathbf{x}_*$ (test input).
L = Cholesky decomposition $(K + \sigma_n^2 I)$;
$\boldsymbol{\alpha} = (L^T)^{-1}(L^{-1}\mathbf{y})$;
$\bar{f}_* = \mathbf{k}_*^T \boldsymbol{\alpha}$;
$\mathbf{v} = L^{-1}\mathbf{k}_*$;
$\mathbb{V}[f_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T\mathbf{v}$;
$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$;
Result: f_* (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood).

Algorithm 3: Algorithm 2.1 from [5].

4.3 Covariance Functions

Covariance functions and kernels have been touched upon, but will be investigated further as they are central to Gaussian processes. A function that only depends on the difference between two points, $\mathbf{x} - \mathbf{x}'$, is called *stationary*. This implies that the function is invariant to translations in input space. If, in addition, it only depends on the length $r = |\mathbf{x} - \mathbf{x}'|$, the function is *isotropic* [5]. Isotropic functions are commonly referred to as *radial basis functions* (RBFs). The covariance function can also depend on the dot product, $\mathbf{x} \cdot \mathbf{x}'$, and is then called a *dot product* covariance function.

A function which maps two arguments $\mathbf{x} \in \mathcal{X}$, $\mathbf{x}' \in \mathcal{X}$ into $\mathbb{R}$ is generally called a *kernel* k. Covariance functions are symmetric kernels, meaning that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. As previously mentioned, the matrix containing all the covariance elements is called the *covariance matrix*, or the Gram matrix K, whose elements are given by

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{4.30}$$

The kernel should contain information about the noise in the data, represented by a constant term added to the diagonal

$$k(\mathbf{x}_i, \mathbf{x}_j)_{noise} = C\delta_{ij}, \tag{4.31}$$

where $C \in \mathbb{R}$ is a real, constant number, and δ_{ij} is the Dirac delta function. In `scikit-learn` this can be implemented either by giving a fixed noise level α to the regressor function, or by using the `WhiteKernel`, which estimates the noise level from the data. This kernel is implemented in `scikit-learn` in the following way for noise level 0.001 with bounds $[10^{-10}, 1]$

```
from sklearn.gaussian_processes.kernels import WhiteKernel
whitenoise = WhiteKernel(noise_level=0.001,
    noise_level_bounds=(1e-10,1))
```

4.3.1 The Squared Exponential Covariance Function

The *squared exponential covariance function* (SE) has the form

$$k_{SE}(r) = \exp\Big(-\frac{r^2}{2\ell^2}\Big), \tag{4.32}$$

where ℓ is the *characteristic length scale*. The length scale determines the smoothness of the function. For a large length scale one should expect a very slowly

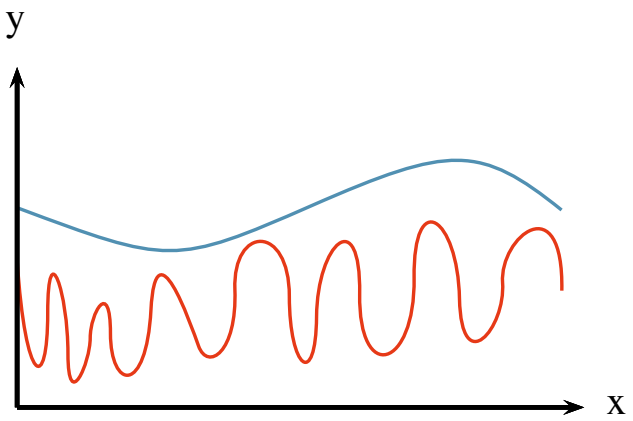


Figure 4.6: The effect of varying the length scale ℓ. A long length scale (blue) gives a smooth, slowly varying function, while a short length scale (red) gives a more staccato, quickly varying function.

varying function, while a shorter length scale means a more rapidly varying function, see the illustration in Fig. (4.6). The SE is infinitely differentiable and therefore very smooth.

The SE is implemented in `scikit-learn` under the name radial basis function, and may be called in the following way for length scale 10, with bounds on the length scale $[0.01, 100]$

```
from sklearn.gaussian_process.kernels import RBF
rbf = RBF(length_scale=10, length_scale_bounds=(1e-2, 1e2))
```

4.3.2 The Matérn Class

The *Matérn class of covariance functions* is given by

$$k_{Matérn}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}r}{\ell}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu}r}{\ell}\right), \tag{4.33}$$

where $\nu, \ell > 0$, and K_{ν} is a modified Bessel function. The hyperparameter ν controls the smoothness of the function, as opposed to the SE kernel which is by definition very smooth. For $\nu \to \infty$ this becomes the SE kernel. In the case of ν being half integer, $\nu = p + \frac{1}{2}$, the covariance function is simply the product of an

[5]Invariant to rigid rotations in input space.

exponential and a polynomial

$$k_{\nu=p+\frac{1}{2}} = \exp\Big(-\frac{\sqrt{2\nu}r}{\ell}\Big)\frac{\Gamma(p+1)}{\Gamma(2p+1)}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}\Big(\frac{\sqrt{8\nu}r}{\ell}\Big)^{p-i}. \tag{4.34}$$

In machine learning the two most common cases are for $\nu = 3/2$ and $\nu = 5/2$

$$k_{\nu=3/2}(r) = \Big(1+\frac{\sqrt{3}r}{\ell}\Big)\exp\Big(-\frac{\sqrt{3}r}{\ell}\Big), \tag{4.35}$$

$$k_{\nu=5/2}(r) = \Big(1+\frac{\sqrt{5}r}{\ell}+\frac{5r^2}{3\ell^2}\Big)\exp\Big(-\frac{\sqrt{5}r}{\ell}\Big). \tag{4.36}$$

In `scikit-learn` the hyperparameter ν is fixed, and so not optimized during training. The Matérn kernel is considered more appropriate for physical processes [5], and may be called in `scikit-learn` in the following way for length scale 10, length scale bounds $[0.01, 100]$ and $\nu = 3/2$

```
from sklearn.gaussian_process.kernels import Matern
matern = Matern(length_scale=10, length_scale_bounds=(1e-2,
    1e2), nu=1.5)
```

For values not in $\nu = [0.5, 1.5, 2.5, \infty]$ `scikit-learn` evaluates Bessel functions explicitly, which increases the computational cost by a factor as high as 10.

Other Kernels

There are several kernels which are not discussed here. Kernels can be multiplied and summed to form new kernels, making the space of possible kernels infinite. For further details see chapter 4 in [5].

Hyperparameters

Each kernel has a vector of hyperparameters, *e.g.* $\boldsymbol{\theta} = (\{M\}, \sigma_f^2, \sigma_n^2)$ for the radial basis function (RBF)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j))^T M(\mathbf{x}_i - \mathbf{x}_j) + \sigma_n^2\delta_{ij}. \tag{4.37}$$

The matrix M can have several forms, amongst them

$$M_1 = \ell^{-2}\mathbb{I}, \qquad M_2 = \mathrm{diag}(\boldsymbol{\ell})^{-2}. \tag{4.38}$$

Choosing $\boldsymbol{\ell}$ to be a vector in stead of a scalar is in many cases useful, especially if the vector of features contain values of different scales, *e.g.* $\mathbf{x} = (x_1, x_2)$ where $x_1 \in [0, 1]$ and $x_2 \in [200, 3000]$. The length scale can be set to a vector in `scikit-learn` by giving the `length_scale` parameter as a `numpy` array of the same dimension as the feature vector $\mathbf{x}$.

4.4 Model Selection

The choice of kernel and hyperparameters is important for the quality of the GP prediction. Model selection means finding the kernel and corresponding hyperparameters that best fit the data. This is referred to as *training* in machine learning. In this section Bayesian model selection is quickly overviewed, and the log marginal likelihood and cross validation are considered.

4.4.1 Bayesian Model Selection

A model has a set of model structures $\mathcal{H}_i$, hyperparameters $\boldsymbol{\theta}$ and parameters $\mathbf{w}$. Feature selection is done at all levels in a hierarchical way, by finding the posterior over *parameters*, the posterior over *hyperparameters* and the posterior for the *model*. Here only the posterior over parameters is considered [5],

$$p(\mathbf{w}|\mathbf{y}, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)}{p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)}, \tag{4.39}$$

as it gives rise to the *marginal likelihood* $p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)$, given by

$$p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)d\mathbf{w} \quad \text{(marginal likelihood)}. \tag{4.40}$$

Because of the complexity of integrals involved in model selection, it is common to maximize the marginal likelihood with respect to hyperparameters $\boldsymbol{\theta}$. This maximization is what distinguishes Bayesian model selection from other model selection schemes, as it incorporates a trade-off between model complexity and model fit. This means that a complex model will allow for several different kinds of models, but each of them will get a low probability. Meanwhile, simple models will only have a few possibilities, but each of these will have a large probability, see Fig. (4.7).

4.4.2 Log Marginal Likelihood

Gaussian process regression models with Gaussian noise have the wonderful trait of analytically tractable integrals for the marginal likelihood. The exact expression for the log marginal likelihood [6] can be shown to be [5]

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi. \tag{4.41}$$

Each of the terms has an interpretation: $-\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y}$ is the only term involving the data, and is therefore the data-fit; $-\frac{1}{2}\log|K_y|$ is the complexity penalty

[6] The logarithm is used as the marginal likelihood varies rapidly.

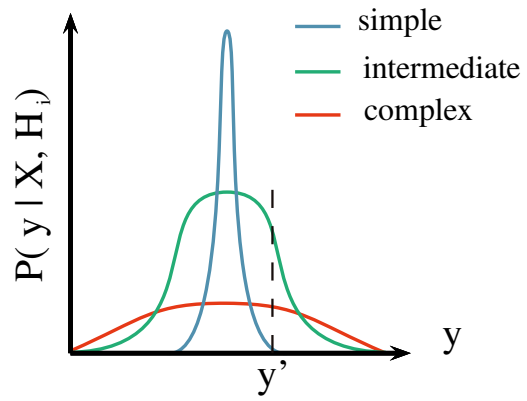


Figure 4.7: The marginal likelihood is the probability of the data, given the model. The number of data points n and inputs X are fixed. The horizontal axis represents an idealized set of all possible vectors of targets $\mathbf{y}$. Since the marginal likelihood is a probability distribution it must normalize to unity. For a particular set $\mathbf{y}'$, indicated by the dashed line, the intermediate model is preferred to the simple and complex ones.

depending only on the covariance function and the inputs; and $-\frac{n}{2}\log 2\pi$ is a normalization term. The marginal likelihood is conditioned on the hyperparameters of the covariance function $\boldsymbol{\theta}$, and the optimal parameters are found by maximizing. This requires the partial derivatives of the log marginal likelihood (LML)

$$\frac{\partial}{\partial \theta_j}\log p(\mathbf{y}|X,\boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial \theta_j}K^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}(K^{-1}\frac{\partial K}{\partial \theta_j}). \tag{4.42}$$

Computing the inverse of a matrix, K^{-1}, is computationally complex, and for n training points goes as $\mathcal{O}(n^3)$. Once this is done, however, finding the partial derivatives only requires complexity $\mathcal{O}(n^2)$, and so gradient based optimizers are advantageous.

The LML can have several local optima, as seen in Fig. (4.8). These correspond to different interpretations of the data. The rightmost optima in Fig. (4.8) for example, favors a small length scale and smaller noise level. This means that it considers little of the data to be noise. The rightmost optimum has a higher noise level, and allows for several large length scales, as it considers most of the data to be noise. Features with very large length scales are considered superfluous, as the function value depends little on them. Because of this type of complication, it might be wise to restart the optimizer a few times during learning.

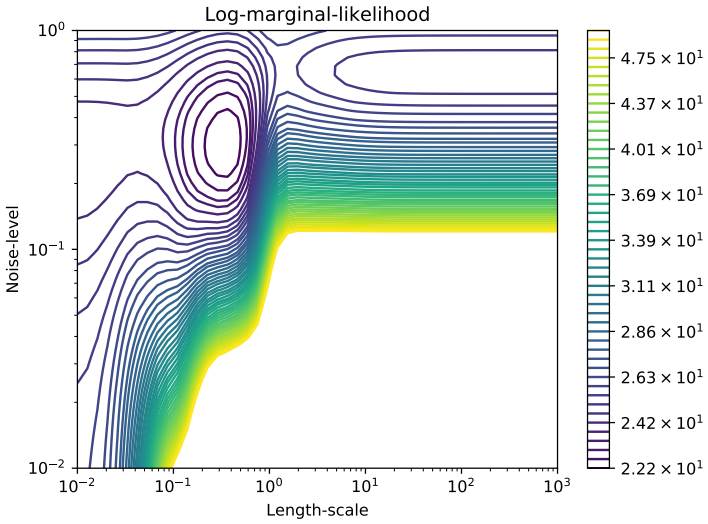


Figure 4.8: A contour plot of the log marginal likelihood with two local optima. The rightmost optima favours a short length scale and low noise, while the leftmost favors a high noise level and therefore several large length scales. Plot generated using scikit-learn.

4.4.3 Cross Validation

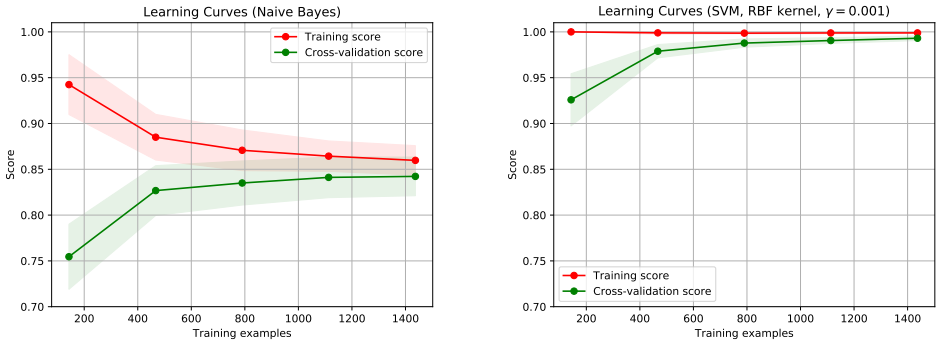
Cross validation is a means of monitoring the performance of a model. In k-fold validation this is done by dividing the data into k subsets and using $k-1$ folds to train the model, and a single fold to validate it. This is repeated k times. Cross-validation requires a loss function, such as the mean relative deviance or the R^2 score. The latter is given by

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1}(y_i - \bar{y})^2}, \tag{4.43}$$

where $\hat{y}_i$ is the predicted value of the ith sample, y_i is the true value and $\bar{y} = \frac{1}{N}\sum_{i=0}^{N-1} y_i$ for N samples. This is the score used for cross validation in this thesis.

Cross-validation can be used to plot learning curves, which is a tool to find out whether the model benefits from adding more data. The learning curve plots the training score and validation score used to find out if the model is *overfitting* or *underfitting*. *Overfitting* means that the model is a perfect fit to the training data, but predicts poorly for test data because it is not general. *Underfitting* occurs when the model is not able to capture the underlying structure of the data.

Examples of learning curves are shown in Fig. (4.9) for Naive Bayes and SVM



(a) Underfitting. **(b)** Well fit.

Figure 4.9: Learning curves for two different estimators.

estimators [7]. In a) both the training score and cross-validation score tend to a value below 1, which indicates underfitting. This model will not benefit from more data. The example in b) shows a training score of approximately 1, and a cross validation score that converges towards 1. This model could benefit from more data.

4.4.4 Relative Deviance

In this project the main loss function used for comparing predictions is the relative deviance. For true values y_i and values predicted by the estimator $\hat{y}_i$ this is given by

$$\varepsilon_i = \frac{y_i - \hat{y}_i}{y_i}. \tag{4.44}$$

The relative deviance is used because of the large span of the target values, ranging from about 10^{-30} to 10^{9}. The data is therefore divided into decades, meaning one set contains $\sigma \in [10^i, 10^{i+1}]$. Then a distribution over the relative deviances within each decade is found, with a mean value and variance. These are plotted as a function of i.

4.5 Distributed Gaussian Processes

Limitations of Gaussian Processes

The biggest weakness of Gaussian processes is that they scale poorly with the size of the data set n. The training and predicting scale as $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively, giving GP a practical limit of $\mathcal{O}(10^4)$.

[7] Methods in Machine Learning.

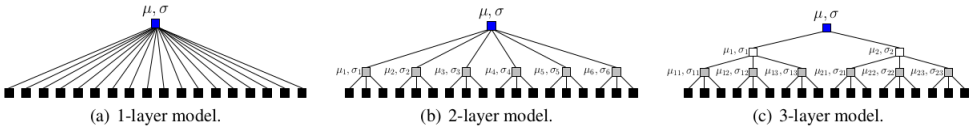


Figure 4.10: From [2].

In [2] a way of scaling GPs to large data sets is proposed, in the form of a robust Bayesian Comittee Machine (rBCM). This method is based on the product-of-experts and Bayesian Comittee Machine, and has the advantage of providing an uncertainty for the prediction.

4.5.1 Product-of-Experts

Product-of-expert (PoE) models are a way of parallelising large computations. They combine several independent computations on subsets of the total data, called 'experts'. In the case of distributed Gaussian processes each expert performs GP on a subset of the training data, and the predictions on a common test set are combined.

Consider the training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, which is partitioned into M subsets $\mathcal{D}^{(k)} = \{\mathbf{X}^{(k)}, \mathbf{y}^{(k)}\}$, $k = 1, ..., M$. Each GP expert does learning on its training data set $\mathcal{D}^{(k)}$, then predictions are combined at the parent node. This node could also be considered an expert for a PoE with several layers, see Fig. (4.10).

4.5.2 Algorithm

The marginal likelihood factorizes into the product of M individual terms because of the independence assumption [2]. The LML is then

$$\log p(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^{(k)}(\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I})^{-1}\mathbf{y}^{(k)} - \frac{1}{2}\log|\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I}|, \qquad (4.45)$$

where $a^{(k)}$ is the quantity corresponding to the kth expert. Computing the LML now entails inverting the $n_k \times n_k$ matrix $(\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2\mathbf{I})$, which requires time $\mathcal{O}(n_k^3)$ and memory consumption $\mathcal{O}(n_k^2 + n_k D)$ for $\mathbf{x} \in \mathbb{R}^D$. For $n_k \ll N$, this reduces the computation time and memory use considerably, and allows for parallel computing.

Several methods for prediction are discussed in [2], but here only the robust Bayesian Comittee Machine is introduced. The PoE predicts a function value f^* at a corresponding test input $\mathbf{x}^*$ according to the predictive distribution

$$p(f^*|\mathbf{x}^*, \mathcal{D}) = \frac{\prod_{k=1}^{M} p_k^{\beta_k}(f^*|\mathbf{x}^*, \mathcal{D}^{(k)})}{p^{-1+\sum_k \beta_k}(f^*|\mathbf{x}^*)}. \qquad (4.46)$$

This prediction scheme allows for much flexibility, as it can vary the importance of an expert. The combined predictive mean and variance are

$$\mu_*^{rbcm} = (\sigma_*^{rbcm})^2 \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*)\mu_k(\mathbf{x}_*), \tag{4.47}$$

$$(\sigma_*^{rbcm})^{-2} = \sum_{k=1}^{M} \beta_k \sigma_k^{-2}(\mathbf{x}_*) + \big(1 - \sum_{k=1}^{M} \beta_k\big)\sigma_{**}^{-2}, \tag{4.48}$$

where the parameters β_k control the importance of the individual experts, but also the how strong the influence of the prior is. In the article, these are chosen according to the predictive power of each expert at $\mathbf{x}^*$. More specifically, β_k is the change in differential entropy between the prior $p(f^*|\mathbf{x}^*)$ and the posterior $p(f^*|\mathbf{x}^*, \mathcal{D}^{(k)})$, which can be calculated as

$$\beta_k = \frac{1}{2}(\log \sigma_{**}^2 - \log \sigma_k^2(\mathbf{x}^*)), \tag{4.49}$$

where σ_{**}^2 is the prior variance, and $\sigma_k^2(\mathbf{x}^*)$ is the predictive variance of the kth expert.

4.5.3 Implementing the Algorithm

The mean and variance in Eq. (4.47)-(4.48) were implemented in `Python` using the `scikit-learn` library's existing framework for regular Gaussian processes. The algorithm was parallelised, so that each expert can learn and predict in parallel, before being combined to the final prediction. Pseudocode for the implementation is found in Alg. (4).

For parallelisation the `scikit-learn` function `Parallel` from `joblib` was used, which runs Python functions as pipeline jobs. It uses the Python function `multiprocessing` as a backend. An example of usage with 3 parallel jobs is

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt
>>> Parallel(n_jobs=3)(delayed(sqrt)(i**2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

where `delayed` is a simple trick to be able to create a tuple with a function-call syntax.

4.5.4 Benchmark

The benchmark function for parallelised distributed Gaussian processes is

$$f(x_1, x_2) = 4x_1 x_2,$$

Data: $N_{experts}$ (number of experts), X (inputs), $\mathbf{y}$ (targets), k (initial kernel), σ_n^2 (noise level), $\mathbf{x}^*$ (test input)
Split training data into N subsets: $X_k, \mathbf{y}_k$;
for *each expert* **do**
  Fit GP to training data $X_k, \mathbf{y}_k$;
  Predict μ_*, σ_*^2 for $\mathbf{x}^*$ using GP ;
  $\sigma_{**}^2 = k(x^*, x^*)$;
end
for *each expert* **do**
  $\beta = \frac{1}{2}(\log(\sigma_{**}^2) - \log(\sigma_*^2))$;
  $(\sigma_*^{rbcm})^{-2} += \beta\sigma^{-2} + \big(\frac{1}{n_{experts}} - \beta\big)\sigma_{**}^{-2}$
end
for *each expert* **do**
  $\mu_*^{rbcm} += (\sigma_*^{rbcm})^2 \beta \sigma_*^{-2} \mu_*$
end
Result: Approximative distribution of $f_* = f(\mathbf{x}_*)$ with mean μ_*^{rbcm} and variance $(\sigma_*^{rbcm})^2$.

Algorithm 4: Pseudocode for using rBCM on a single test point $\mathbf{x}_*$. For the fit and prediction of each expert GP Algorithm (3) is used.

where the vectors $\mathbf{x} = (x_1, x_2)$ were drawn from a random normal distribution using the `numpy` function `random.randn`. Gaussian processes implemented by `scikit-learn` in the function `GaussianProcessRegressor` were compared to distributed Gaussian processes with 4 experts. 2000 training points and 1000 test points were used, and the resulting times for the GP and DGP were

$$\text{Gaussian processes time: } 154.12 \text{ s} \tag{4.50}$$

$$\text{Distributed Gaussian processes time: } 5.61 \text{ s} \tag{4.51}$$

Histograms of the relative deviances for Gaussian processes (GP) and Distributed Gaussian processes (DGP) are found in Fig. (4.11).

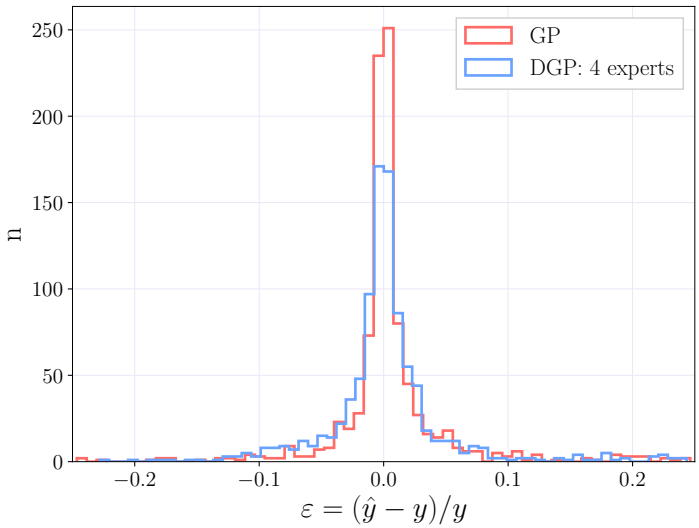


Figure 4.11: Histogram of the relative deviance between true value y and predicted value $\hat{y}$ for Gaussian process regression (GP) and Distributed gaussian process regression (DGP) for the function $f(x_1, x_2) = 4x_1x_2$.

Chapter 5

Evaluating Cross Sections with Gaussian Processes

This chapter is dedicated to evaluating cross sections for squark pair production with Gaussian processes. The learning of a benchmark process is considered, and compared to Gaussian processes with changes in the data set, the kernel and different features. One change to the benchmark settings is considered at a time. Then the effect of adding more data in the form of experts is investigated, and finally learning curves for the optimal parameters are calculated to decide whether the method benefits from adding more data.

5.1 Data Generation

In this section the generation of MSSM-24 training and test data is discussed, following closely the discussion in [**?**].

Sampling of Data

An MPI parallelized script generates a sample point in parameter space by drawing random values from the distributions in Tab. (5.1). The table contains log and flat priors, and a combination of these is used to cover more of the parameter space. When a parameter point has been sampled, it is run through the program `softpoint.x` which calculates its SUSY spectrum using the `Softsusy 3.6.2`-package [**?**]. The spectrum is then written to a `slha`-file and given as input to `Prospino 2.1`, which calculates the LO and NLO cross sections according to the method outlined in Section 3.4.1. The relevant features and NLO cross sections are harvested to `.dat`-files, which are used by the Gaussian processes.

Parameter	Log prior range	Flat prior range
M_1	[0,100,4000]	[0,4000]
M_2	[0,100,4000]	[0,4000]
M_3	[0,100,4000]	[0,4000]
A_t	[-4000, -100, 100, 4000]	[-4000, 4000]
A_b	[-4000, -100, 100, 4000]	[-4000, 4000]
A_τ	[-4000, -100, 100, 4000]	[-4000, 4000]
μ	[-4000, -100, 100, 4000]	[-4000, 4000]
m_A^{pole}	[0,100,4000]	[0,4000]
$\tan\beta$	[2, 60]	[2, 60]
m_{L_1}	[0, 100, 4000]	[0, 4000]
m_{L_2}	[0, 100, 4000]	[0, 4000]
m_{L_3}	[0, 100, 4000]	[0, 4000]
m_{e_1}	[0, 100, 4000]	[0, 4000]
m_{e_2}	[0, 100, 4000]	[0, 4000]
m_{e_3}	[0, 100, 4000]	[0, 4000]
m_{Q_1}	[0, 100, 4000]	[0, 4000]
m_{Q_2}	[0, 100, 4000]	[0, 4000]
m_{Q_3}	[0, 100, 4000]	[0, 4000]
m_{u_1}	[0, 100, 4000]	[0, 4000]
m_{u_2}	[0, 100, 4000]	[0, 4000]
m_{u_3}	[0, 100, 4000]	[0, 4000]
m_{d_1}	[0, 100, 4000]	[0, 4000]
m_{d_2}	[0, 100, 4000]	[0, 4000]
m_{d_3}	[0, 100, 4000]	[0, 4000]

Table 5.1: Table showing the sampling intervals used for the parameters when sampling the MSSM-24 model, where the soft breaking scale is set to $Q = 1$ TeV. The log priors have three and four limit values, which are of the form [start_flat, start_log, end_log] and [start_log, start_flat, start_log, end_log]. All values in GeV except $\tan\beta$ which is unitless. Table from [**?**].

Priors

To get a reasonable distribution in parameter space it is necessary to use an objective prior distribution of parameters. This means that priors are assigned according to a set of principles of how information should be encoded in a probability distribution. More details on objective priors can be found in [3].

The first of these principles is the *transformation group invariance*, which states that the probability distribution should be invariant under any transformation that is considered irrelevant to the problem. In other words, the *pdf* should satisfy

$$\pi(x|I)dx = \pi(x+a|I)d(x+a) \qquad \Rightarrow \qquad \pi(x|I) = \pi(x+a|I), \tag{5.1}$$

where a is some translation. This is often referred to as a uniform or *flat prior*. Invariance under scale transformations, which are transformations that introduce a scale to the problem $m \to m' = cm$, requires

$$\pi(m|I)dm = \pi(cm|I)cdm, \tag{5.2}$$

which is satisfied if $\pi(m|I) \propto 1/m$. Since this corresponds to $\pi(\log m|I)$ being flat, it is called the *log prior*.

The flat prior covers the edges of parameter space well, while a log prior covers the innermost points [1]. Therefore, a combination of the log and flat priors is used in order to properly cover parameter space. To avoid divergence of the log prior close to zero this region is covered by a flat prior. The limits on the priors are `[start, flat_start, flat_end, end]` for priors that include negative values, and `[flat_start, start, end]` for priors with only positive values. An illustration of a prior with positive values is shown in Fig. (5.1), where a flat distribution is used close to $x = 0$ to avoid divergences.

The weak scale MSSM model used in this project, MSSM-24, requires a soft breaking scale Q. This scale is set to $Q = 1$ TeV. It is also worth noting that the parameter space for the cross sections is significantly reduced from that of the MSSM-24. The cross sections depend on the values $m_{\tilde{g}}$, $m_{\tilde{q}}$, $\tilde{g}_s$, $\hat{g}_s$ and s. Since only first and second generation squarks are considered, this contributes with 8 masses [2] which combined with the 4 other parameters reduces the parameter space to 12 dimensions. The parameter space is thus better sampled than it would appear from the 24 parameters in MSSM-24.

Data Quality

To ensure the data is properly distributed in parameter space data quality plots are generated. In Fig. (5.2) distributions for $m_{\tilde{g}}$, $m_{\tilde{d}_L}$ and $m_{\tilde{u}_L}$ are shown, as

[1] The points close to 0.

[2] 4 quarks with a pair of lefthanded and righthanded squarks each.

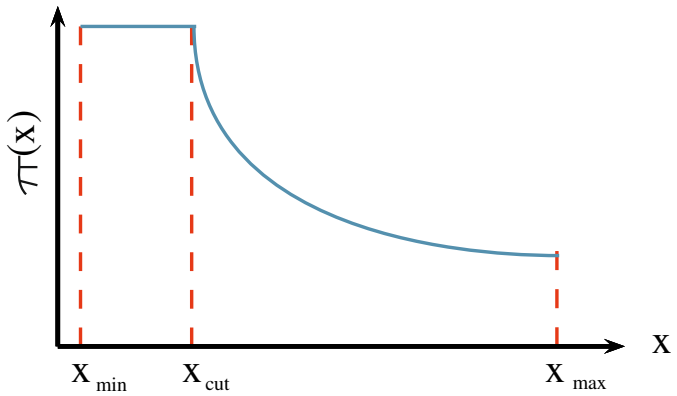


Figure 5.1: Illustration of the log prior distribution $\pi(x)$. Around $x = 0$ the prior would blow up, so at a limit x_{cut} a flat prior is used.

these are the most relevant for this thesis. The scatter plots use 2000 points, which is the number of training points used in the benchmark settings in the following sections. All parts of the parameter space seem to be covered, although the density of points is higher for small masses. This could affect predictions with few training points.

Noise

The observations contain some noise that originates in the `Prospino 2.1` calculation. In a parameter point chosen at random the relative error has a standard deviation of $\varepsilon = 0.002$. This error fluctuates little between parameter points, so it is considered a good approximation for the order of magnitude of errors in all points. The goal is now to incorporate this information in the Gaussian process. For that purpose the following relation is considered,

$$Y_i = y_i^{true} + \epsilon_i = y_i^{true}(1 + \varepsilon_i), \tag{5.3}$$

where cross sections provided by Prospino are denoted as Y_i, real cross sections as y_i^{true} and $\varepsilon_i \sim \mathcal{N}(0, \varepsilon)$. The distribution of Y_i can thus be written as

$$Y_i = \mathcal{N}(y_i^{true}, \varepsilon y_i^{true}), \tag{5.4}$$

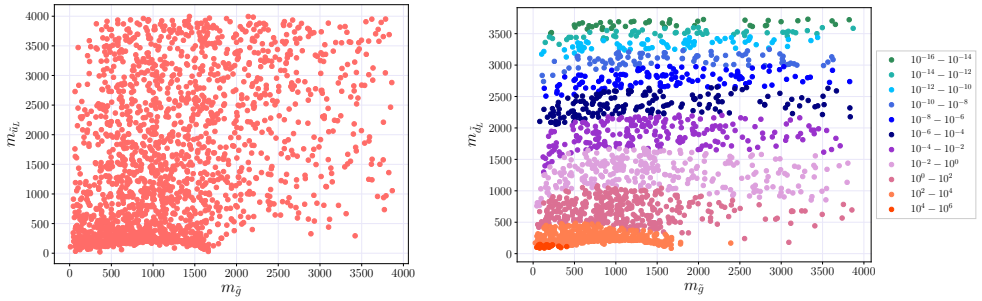
where the only random variable is ε. Changing variables to $\log_{10}$ gives

$$X_i = \log_{10} Y_i \to Y_i = 10^{X_i} \tag{5.5}$$

$$P_{X_i}(X_i) = P_{Y_i}(Y_i)\left|\frac{\partial Y_i}{\partial X_i}\right| \tag{5.6}$$

$$= P_{Y_i}(y_i)10^{X^i}\log 10 \tag{5.7}$$

$$= \mathcal{N}(10^{x_i^{true}}, \varepsilon 10^{x_i^{true}}) \cdot 10^{X_i} \cdot \log 10. \tag{5.8}$$



(a) $m_{\tilde{g}}$-$m_{\tilde{u}_L}$ **(b)** $m_{\tilde{g}}$-$m_{\tilde{d}_L}$

Figure 5.2: Data quality plots of the distribution of mass parameters $m_{\tilde{d}_L}$, $m_{\tilde{u}_L}$ and $m_{\tilde{g}}$ for 2000 points. In **(b)** different orders of magnitude of the cross sections for $\tilde{d}_L\tilde{d}_L$ are shown in different colors, indicating that there are few points in the very high cross sections $\sigma \propto 10^4 - 10^6$, and that lower cross sections are more spread across the gluino mass spectrum.

So the relevant distribution is in fact

$$X_i = \log_{10} Y_i = \log_{10} y_i^{true} + \log_{10}(1 + \mathcal{N}(0, \varepsilon)),$$

where the following expansion can be made

$$\log_{10}(1 + \mathcal{N}(0, \varepsilon)) \simeq \frac{\mathcal{N}(0, \varepsilon)}{\log 10} - \frac{\mathcal{N}(0, \varepsilon)^2}{\log 100} + ... \tag{5.9}$$

Since the leading order term is dominant for small ε the logarithm of the cross section may be approximated as

$$X_i \simeq \log_{10} y_i^{true} + \frac{1}{\log 10}\mathcal{N}(0, \varepsilon) \tag{5.10}$$

Eq. (**??**) is now on the form of the Gaussian noise model. The error distribution has a standard deviation $\varepsilon = 2 \cdot 10^{-3}$, so noise variance of the Gaussian process should be

$$\left(\frac{\varepsilon}{\log 10}\right)^2 = \frac{(2 \cdot 10^{-3})^2}{(\log 10)^2} = \frac{4 \cdot 10^{-6}}{5.301} \simeq 7.544 \cdot 10^{-7}.$$

The noise term predicted by the GP should therefore be of the order of $\mathcal{O}(10^{-7})$.

5.2 Dataset Transformations

The plot in Fig. (5.2b) indicates that cross sections, especially those of the order $\mathcal{O}(1$ fb) and lower, are very spread as a function of $m_{\tilde{g}}$. This is more evident

in the upper left panel of Fig. (5.3), where the cross section σ is plotted as a function of the gluino mass. The upper left panel shows σ as a function of the squark mass $m_{\tilde{q}}$, which is more defined but still has some spread. This section will be devoted to reducing the spread caused by the gluino mass.

Scaling Functions

As previously mentioned, the partonic cross sections can be written in terms of scaling functions f

$$\hat{\sigma}_{ij} = \frac{\alpha_s^2(Q^2)}{m^2}\left\{ f_{ij}^B(\eta, r) + 4\pi\alpha_s(Q^2)\left[f_{ij}^{V+S}(\eta, r, r_t) + f_{ij}^H(\eta, r) + \bar{f}_{ij}(\eta, r)\log\left(\frac{Q^2}{m^2}\right)\right]\right\}, \tag{5.11}$$

where

$$\eta = \frac{s}{m^2} - 1, \qquad r = \frac{m_{\tilde{g}}^2}{m_{\tilde{q}}^2}, \qquad r_t = \frac{m_t^2}{m^2},$$

where $m = (\sqrt{p_1^2} + \sqrt{p_2^2})/2$ is the average mass of the particles produced. The scaling functions are the different contributions to the cross section, as explained in Chapter 2. As the total cross section only differs from the partonic cross section by an integral over parton distribution functions, the mass dependencies in Eq. (5.11) are relevant for the total cross sections as well.

The energy near the threshold is the base for an important part of the contributions to the cross section [**?**]. In this region the scaling functions can be expanded in the low velocity of produced particles β, leading to the following expressions [**?**]

$$\begin{aligned}
f_{qq}^B &= \frac{8\pi\beta m_{\tilde{q}}^2 m_{\tilde{g}}^2}{27(m_{\tilde{q}}^2 + m_{\tilde{g}}^2)^2}, & f_{q'q}^B &= \frac{8\pi\beta m_{\tilde{q}}^2 m_{\tilde{g}}^2}{9(m_{\tilde{q}}^2 + m_{\tilde{g}}^2)^2} \\
f_{qq}^{V+S} &= f_{qq}^B \frac{1}{24\beta} & f_{q'q}^{V+S} &= f_{q'q}^B \frac{1}{24\beta} \\
f_{qq}^H &= f_{qq}^B\Big[\frac{2}{3\pi^2}\log^2(8\beta^2) - \frac{7}{2\pi^2}\log(8\beta^2)\Big] & f_{q'q}^H &= f_{q'q}^B\Big[\frac{2}{3\pi^2}\log^2(8\beta^2) - \frac{19}{6\pi^2}\log(8\beta^2)\Big] \\
\bar{f}_{qq} &= -f_{qq}^B\frac{2}{3\pi^2}\log(8\beta^2) & \bar{f}_{q'q} &= -f_{q'q}^B\frac{2}{3\pi^2}\log(8\beta^2).
\end{aligned} \tag{5.12}$$

As the main contributions come from this energy region, it may be possible to remove some of the complexity of the function using the expressions in Eq. 5.12. Note that all terms are proportional to f_{qq}^B ($f_{q'q}^B$), which is again proportional to $m_{\tilde{q}}^2 m_{\tilde{g}}^2$. Since the partonic cross section is proportional to $\sigma \propto 1/m^2$, and

$m^2 = m_{\tilde{q}}^2$ in the case of squark production this $m_{\tilde{q}}^2$-dependency is automatically cancelled. However, the following transformation can be made

$$\sigma \to \sigma_{m_{\tilde{g}}} = \frac{\sigma}{m_{\tilde{g}}^2}, \tag{5.13}$$

reducing the gluino mass dependency. The lower panels in Fig. (5.3) show $\sigma_{m_{\tilde{g}}}$ as a function of $m_{\tilde{g}}$ and $m_{\tilde{q}}$. The spread as a function of the squark mass is much smaller, and for high cross sections the shape of $\sigma_{m_{\tilde{g}}}$ as a function of squark and gluino mass are very similar, which may make it easier to find a kernel that fits well in both dimensions. Therefore, the target value in this thesis will be the logarithm of $\sigma_{m_{\tilde{g}}}$. The logarithm is used because of the large span in function values. In the threshold region, the partonic version of this expression is

$$\hat{\sigma}_{ij,m_{\tilde{g}}} = \frac{8\pi\beta\alpha_s^2(Q^2)}{27(m_{\tilde{q}}^2 + m_{\tilde{g}}^2)^2}\Bigg\{1 + 4\pi\alpha_s(Q^2)\Bigg[\frac{1}{24\beta} + \frac{2}{3\pi^2}\log^2(8\beta^2) \tag{5.14}$$

$$- \frac{7}{2\pi^2}\log(8\beta^2) - \frac{2}{3\pi^2}\log(8\beta^2)\log\left(\frac{Q^2}{m^2}\right)\Bigg]\Bigg\}. \tag{5.15}$$

Another possibility is to further reduce the mass dependecy, by defining σ_{fac} as

$$\sigma_{fac} = \sigma\frac{(m_{\tilde{g}}^2 + m_{\tilde{q}}^2)^2}{m_{\tilde{g}}^2}. \tag{5.16}$$

This transformation provides an even smoother function, but has been left as further work with GPs.

5.3 Learning the Gaussian Process

In this section a single Gaussian process is learned with benchmark settings, and then a selected set of improvements are introduced. The results are quantified using plots of the relative deviance defined in Chapter 4 (REFERER TIL LIKNINGEN).

5.3.1 The Benchmark

In this thesis the benchmark processes will be the production of $\tilde{d}_L\tilde{d}_L$ and $\tilde{d}_L\tilde{u}_L$. The benchmark settings are a single GP with 2000 training points and 20 000 test points that uses $m_{\tilde{g}}$ and $m_{\tilde{d}_L}$ as features (as well as $m_{\tilde{u}_L}$ for $\tilde{d}_L\tilde{u}_L$ production). The exponential squared kernel (RBF [3]) with a white noise term is used. The kernel is implemented in `scikit-learn` in the following way

[3]The exponential squared kernel is often called the radial basis function, hence the abbreviation.

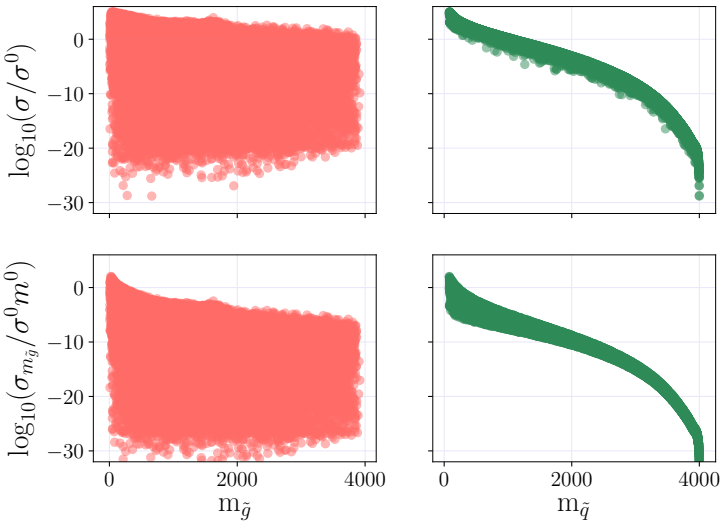


Figure 5.3: The quantities σ, $\sigma_{m_{\tilde{g}}}$ and $\sigma_{\tilde{q}}$ as functions of the gluino mass $m_{\tilde{q}}$ and squark mass $m_{\tilde{q}}$ for the production of $\tilde{d}_L\tilde{d}_L$. Here, $m_{\tilde{q}} = m_{\tilde{d}_L}$. The functions have less spread when some of the mass dependency is removed, which may make learning easier.

```
kernel_BM = C(constant_value=10,
    constant_value_bounds=(1e-3, 1e4)) * RBF(length_scale =
    np.array([1000, 1000]), length_scale_bounds=(1, 1e6)) +
    WhiteKernel(noise_level=1, noise_level_bounds=(2e-10,1e2))
```

where the features are `(m_gluino, m_dL)` for $\tilde{d}_L\tilde{d}_L$ and `(m_gluino, m_dL, m_uL)` for $\tilde{d}_L\tilde{u}_L$. Note that the length scale of the RBF is given a a vector of the same dimension as the feature vector $\mathbf{x}, \boldsymbol{\ell} \in \mathbb{R}^D$ [4]. This is in case the different masses have different covariances, as they appear to do from the lower panels in Fig. (5.3).

Because of the large span of $\sigma_{m_{\tilde{g}}}$-values (ranging from 10^{-32} to 10^6), values are divided into decades. This means that for each interval $10^i - 10^{i+1}$ an error is calculated. The error is the mean and variance of the distribution of relative deviations in each decade

$$\mu = \mathrm{m}\left(\frac{y-\hat{y}}{y}\right), \qquad \sigma^2 = \mathrm{var}\left(\frac{y-\hat{y}}{y}\right). \tag{5.17}$$

The means and variances of the logarithms are plotted for the BM settings in Fig. (5.4) for $\tilde{d}_L\tilde{d}_L$. The plot for $\tilde{d}_L\tilde{u}_L$ is very similar, and is therefore not shown here.

[4]This example is for $\tilde{d}_L\tilde{d}_L$ production, for $\tilde{d}_L\tilde{u}_L$ production $D = 3$, as $m_{\tilde{u}_L}$ is inluded as well.

	C	$\ell_{m_{\tilde{g}}}$	$\ell_{m_{\tilde{d}_L}}$	$\ell_{\bar{m}}$	α
BM	2981	5470	2190		0.47
No outliers	9702	5740	215		0.00372
$\sigma > 10^{-16}$ fb	515	1170	998		0.0036
$\bar{m}$	1095	1190	200	846	0.0000119
Matern	1000	30200	8600		0.462

Table 5.2: Optimal kernel parameters for different settings for $\tilde{d}_L\tilde{d}_L$.

	C	$\ell_{m_{\tilde{g}}}$	$\ell_{m_{\tilde{d}_L}}$	$\ell_{m_{\tilde{u}_L}}$	$\ell_{\bar{m}}$	α
BM	2490	5870	4000	2220		0.593
No outliers	6400	4420	3100	240		0.00348
$\sigma > 10^{-16}$ fb	10000	1540	2900	2150		0.0031
$\bar{m}$	3806	1340	3590	251	748	0.0000119
Matern	1000	31100	1000000	8260		0.585

Table 5.3: Optimal kernel parameters for different settings for $\tilde{d}_L\tilde{u}_L$.

The optimal kernel parameters found by the GP are found in Tab. (5.2)-(5.3), while computation times on a laptop are found in Tab. (5.4). The predicted noise levels α are very high for both processes, at $\alpha = 0.47$ for $\tilde{d}_L\tilde{d}_L$ and $\alpha = 0.593$ for $\tilde{d}_L\tilde{u}_L$, which is far from the expected value of $\sim 7.544 \cdot 10^{-7}$. In addition, the prediction is very unstable, with a over prediction for low cross sections.

5.3.2 Outliers

Calculations in `Prospino 2.1` set some NLO terms to zero because $K = 0$ for numerical reasons, as discussed in 3.4.1. This leads to outliers in the dataset, as shown in Fig. (5.5), where they can be seen as a cluster of points well below the other cross sections. These points have zero cross sections, which are set to $\epsilon = 10^{-32}$ fb in the calculation to avoid divergences. Removing the ouliers makes the prediction much better for small cross sections, but also stabilizes the

	Time $\tilde{d}_L\tilde{d}_L$	Time $\tilde{d}_L\tilde{u}_L$
BM	00:07:48	00:08:40
Outliers	00:08:42	00:11:20
Cut	00:07:24	00:11:07
Features	00:11:20	00:16:37
Kernel	00:07:28	00:13:05

Table 5.4: Computation times for GP with 2000 training points and 20 000 test points on a laptop with 4 cores.

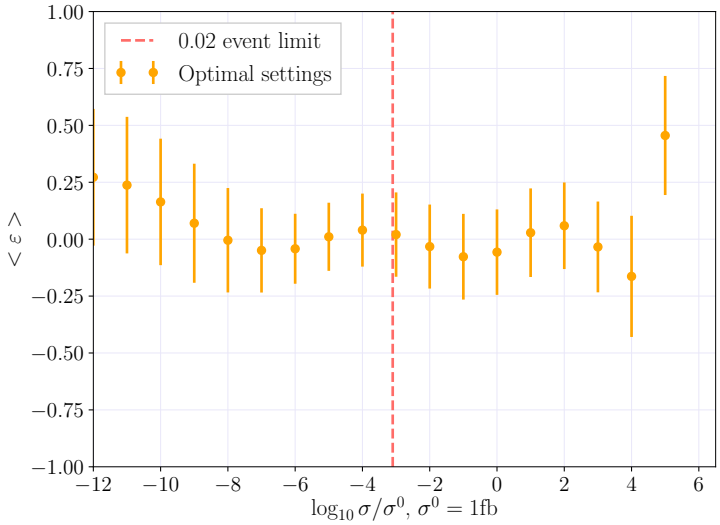


Figure 5.4: The relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with benchmark settings. 2000 training points and 20 000 test points were used on a regular GP, with the final kernel as described in the text. Features are the physical masses $m_{\tilde{g}}$ and $m_{\tilde{d}_L}$.

prediction for larger values, as can be seen in Fig. (5.7 a) where the prediction without outliers is compared to the BM. The predicted noise levels are significantly reduced with the removal of outliers for both processes, from $\alpha = 0.47$ to $\alpha = 0.00372$ for $\tilde{d}_L\tilde{d}_L$ and from $\alpha = 0.593$ to $\alpha = 0.00348$ for $\tilde{d}_L\tilde{u}_L$. This implies that including the outliers leads the GP to underfit, which means that much of the signal is considered noise.

5.3.3 Cuts on Cross Sections

Smooth functions are easier to fit with Gaussian processes, and the target values are very steep functions of large squark masses. This can be seen from the righthand panels in Fig. (5.3). In addition, small target values comprise the regions with the most spread as a function of the gluino mass. Since the limit for 0.02 events is at $\sigma = 10^{-3}$ fb [5], a lower cut is set at $\sigma_{cut} = 10^{-16}$ fb. The cut excludes all cross sections lower than σ_{cut} from both training and testing. The resulting error distributions for $\tilde{d}_L\tilde{d}_L$ are shown in Fig. (5.7 b), and the optimal kernel parameters are found in Tab. (5.2)-(5.3). Noise levels are further reduced from the case where outliers are removed, with the variance going from $\alpha = 0.00372$ to $\alpha = 0.00336$ for $\tilde{d}_L\tilde{d}_L$ and from $\alpha = 0.00348$ to $\alpha = 0.0031$ for

[5] Cross sections with lower values than this predict less than 0.02 events, and are thus less interesting in this thesis.

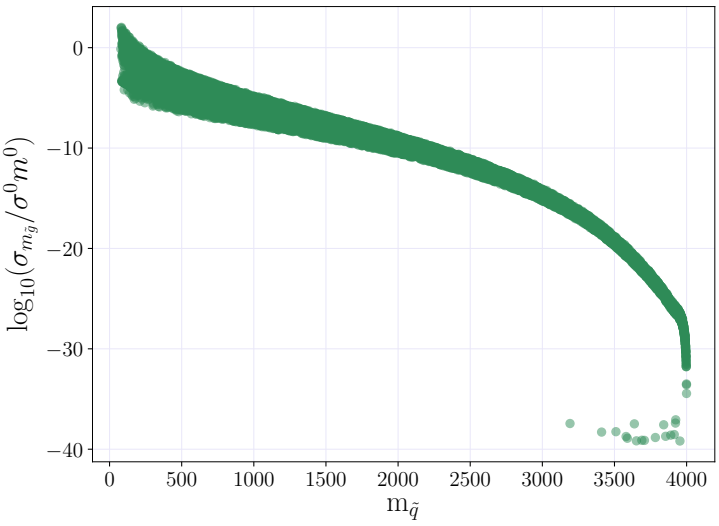


Figure 5.5: Plot of $\log(\sigma_{m_{\tilde{g}}})$ for $\tilde{d}_L\tilde{d}_L$ as a function of $m_{\tilde{d}_L}$, where outliers are included. The outliers are originally $\sigma = 0$ fb, but are set to $\sigma = 10^{-32}$ fb so as to avoid infinities.

$\tilde{d}_L\tilde{u}_L$. The estimated noise level seems to be approaching the expected theoretical value, indicating that the exclusion of low cross sections improves the prediction. The error in the prediction of the highest targets is significantly improved from the cases of the BM and the removed outliers. Training and testing only on high values renders the prediction very stable for all (included) orders of magnitude.

5.3.4 Features

`Prospino 2.1` calculates the NLO cross section for the mean of the squark masses, $\bar{m}$, and uses this to find the K-factor. The K-factor is then multplied with the LO cross sections for non-degenerate squark masses to find the individual NLO terms. There is therefore reason to believe that adding the mean squark mass as a feature will improve the prediction. The features used in this section are

```
features_dLdL = (m_gluino, m_dL, m_mean)
features_dLuL = (m_gluino, m_dL, m_uL, m_mean)
```

The optimal kernel values are found in Tab. (5.2)-(5.3). Adding the mean mass as a feature reduces the estimated noise level considerably, and estimates the same level for $\tilde{d}_L\tilde{d}_L$ and $\tilde{d}_L\tilde{d}_L$, at $\alpha = 1.19 \cdot 10^{-5}$. Since the two processes should have approximately the same level of noise this is an indication that the prediction is improved. Further, the length scale for $\bar{m}$ is smaller than for $m_{\tilde{g}}$.

This implies that there is a higher dependence on the mean mass than the gluino mass, since GPs attribute large length scales to irrelevant features. The resulting error distributions for $\tilde{d}_L\tilde{d}_L$ are shown in Fig. (5.7 c) and compared to the BM. With some exceptions where the variances are very large adding the new feature gives a mean of almost zero and very small variances for cross sections over the 0.02 event limit.

5.3.5 Kernel

While the exponential squared kernel is very smooth, the Matérn kernel has a hyperparameter ν to control its smoothness. It is sometimes argued that this makes Matérn a better kernel for physical processes. In this section the Matérn kernel with $\nu = 1.5$ is compared to the BM RBF kernel, and the resulting error distributions for $\tilde{d}_L\tilde{d}_L$ are found in the lower right panel of Fig. (5.4). The choice of hyperparameter is explained below. The predictions are somewhat more stable for high cross sections than with the RBF kernel. For low cross sections the error distributions have larger variances, but as this is currently not the region of interest the Matérn kernel is considered a better fit than the RBF. As can be seen from the optimal kernel values in Tab. (5.2)-(5.3) the Matérn kernel predicts a slightly lower noise level than the RBF. The noise variances go from $\alpha = 0.47$ with RBF to $\alpha = 0.462$ with Matérn for $\tilde{d}_L\tilde{d}_L$, and from $\alpha = 0.593$ with RBF to $\alpha = 0.585$ with Matérn for $\tilde{d}_L\tilde{u}_L$.

Hyperparameters

The Matérn kernel has the aforementioned hyperparameter ν, which controls the smoothness of the function. As $\nu \to \infty$ the function becomes very smooth, and approaches the exponential squared kernel. For half-integer values the Matérn kernel becomes the product of an exponential squared kernel and a polynomial – a simple form which is faster to compute [6]. In order to determine the best value of ν k-fold cross validation was performed as a function of ν for the BM settings. The validation curve uses the R^2-score and $k = 5$ for the cross validation, and is shown in Fig. (5.6). Based on the cross validation, the best value for the hyperparameter is $\nu = 1.5$.

5.3.6 Optimal Settings

A combination of the improved settings found in the foregoing sections is tested in this section. The cummulative settings are; a single GP with outlier points removed; a lower cut on cross sections $\sigma > 10^{-16}$ fb; the Matérn kernel with

[6] Scikit-learn takes around 10 times longer to compute values not in the interval $\nu = [0.5, 1.5, 2.5, \infty]$, because of the evaluation of Bessel functions.

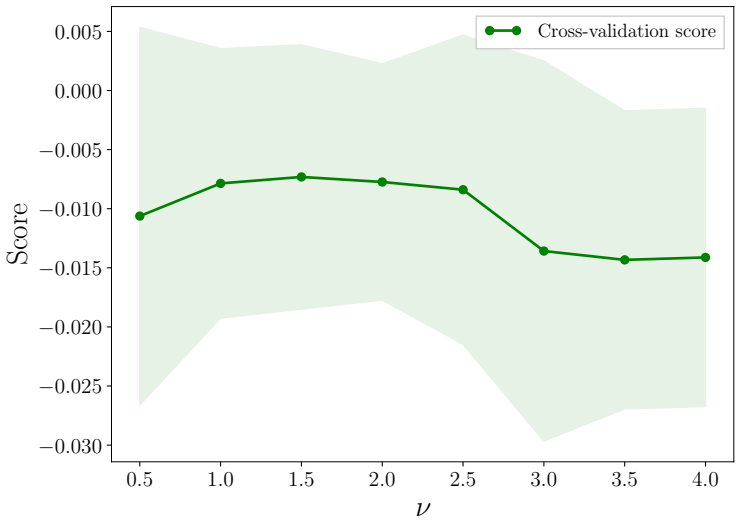


Figure 5.6: Validation curve as a function of the hyperparameter ν of the Matérn kernel, with k-fold validation with $k = 5$. The settings are the BM settings, except for the exchange of the RBF kernel for the Matérn kernel. The scoring parameter is the R^2-score, but here $R^2 - 1$ is plotted.

$\nu = 1.5$; and the gluino mass, the relevant squark mass(es) and the mean of all squark masses as features. The resulting error distributions are found in Fig. (5.8). With all improvements implemented the prediction is very good, as all error distribution variances are less than 5%. The computation with optimal settings takes 00:09:30 for $\tilde{d}_L\tilde{d}_L$ and 00:10:28 for $\tilde{d}_L\tilde{u}_L$ on a regular laptop with 4 cores.

Noise

Since the noise level is known to some degree, it is worth testing whether this knowledge be used in Gaussian processes. As previously shown, the variance of the Gaussian distribution of noise is around $\alpha_{fix} = 7.544 \cdot 10^{-7}$. In `scikit-learn` an option to letting the `WhiteKernel` estimate the level of noise is to specify the noise by passing it as `alpha` to the Gaussian process regressor function

```
gp = GaussianProcessRegressor(kernel=kernel, alpha=7.544e-7)
```

For the optimal settings the `WhiteKernel` predicts values close to α_{fix}, with $\alpha = 10^{-5}$ for $\tilde{d}_L\tilde{d}_L$ and $5.63 \cdot 10^{-6}$ for $\tilde{d}_L\tilde{u}_L$. The remaining kernel parameters therefore hardly change when α is fixed, as can be seen in Tab. (5.5). As expected, the prediction changes very little. A marginal improvement can be seen in the variance for α_{fix} in Fig. (5.9). For calculations with few training points the

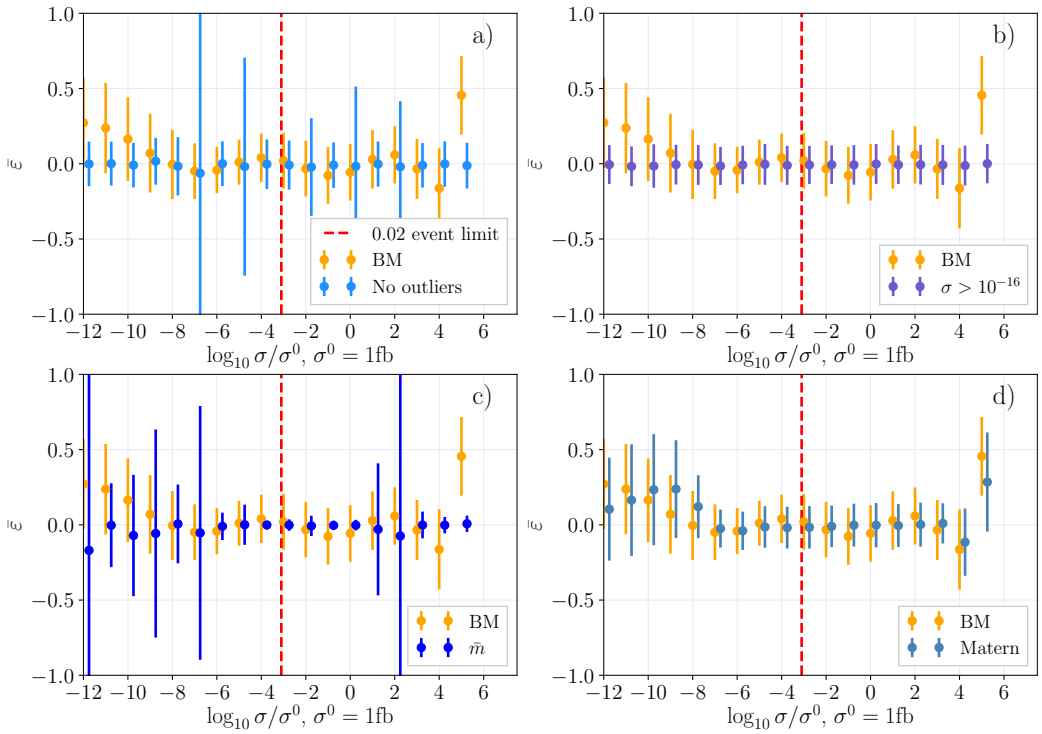


Figure 5.7: The distribution of the relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with a) benchmark settings (orange) and removed outliers (blue); b) benchmark settings (orange) and a lower cut on cross sections (blue); c) benchmark settings (orange) and the added feature $\bar{m}$ (blue); d) the benchmark settings (orange) and the Matérn kernel (blue). Benchmark settings are abbreviated BM.

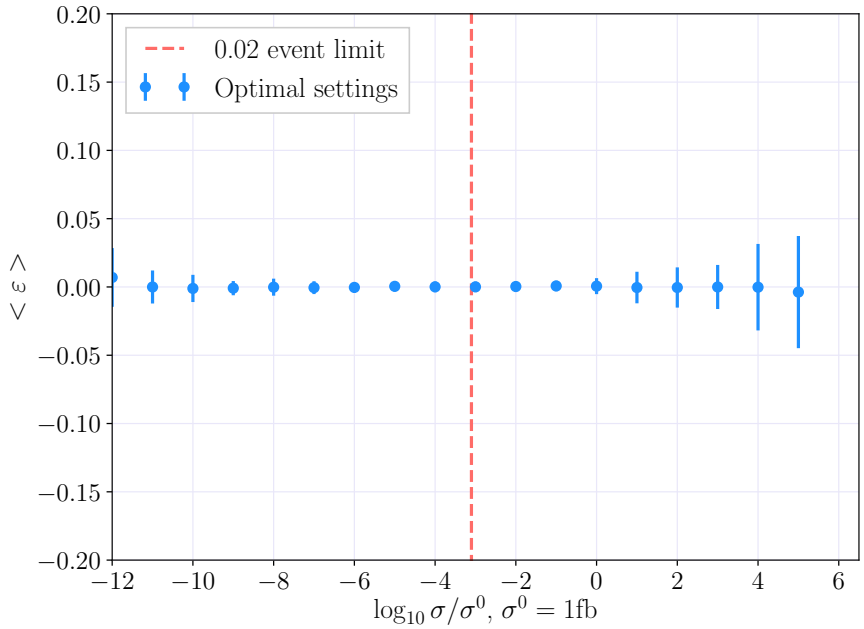


Figure 5.8: The distribution of the relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with the optimal settings; a single GP with 2000 training points that uses the Matérn kernel with $\nu = 1.5$. Outliers are removed and a lower cut at $\sigma = 10^{-16}$ fb is introduced. The features are $m_{\tilde{g}}$, $m_{\tilde{d}_L}$ and $\bar{m}$, and the target is $\sigma_{m_{\tilde{g}}}$. All error distibutions have σ_{std} smaller than 5%.

	C	$\ell_{m_{\tilde{g}}}$	$\ell_{m_{\tilde{d}_L}}$	$\ell_{m_{\tilde{u}_L}}$	$\ell_{\bar{m}}$	α
$\tilde{d}_L\tilde{d}_L$	750	30500	17400		74800	$1 \cdot 10^{-5}$
$\tilde{d}_L\tilde{d}_L$	1000	28300	18300		69900	$7.544 \cdot 10^{-7}$ (fixed)
$\tilde{d}_L\tilde{u}_L$	1000	30200	79600	18500	89000	$5.63 \cdot 10^{-6}$
$\tilde{d}_L\tilde{u}_L$	1000	29100	66200	18300	76000	$7.544 \cdot 10^{-7}$ (fixed)

Table 5.5: Kernel parameters for the optimal settings with the noise level estimated by the GP, and given as a constant $\alpha = 7.544 \cdot 10^{-7}$.

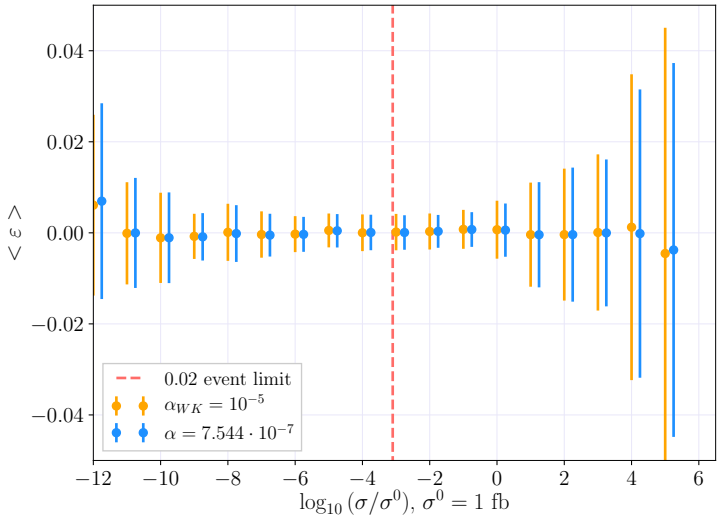


Figure 5.9: The distribution of the relative deviance ε as a function of the logarithm of the normalized cross section for $\tilde{d}_L\tilde{d}_L$ with the optimal settings. In one case the noise level is estimated by the GP (orange), and in the other it is given as a parameter α (blue).

computation time is not affected in any great way, but for larger datasets the removal of α from the kernel may affect the time.

5.4 Distributed Gaussian Processes

Adding experts improves the prediction with very little increase of the computational cost. In Fig. (5.10) a comparison of error distributions between one expert and four experts with 2000 training points each, and one expert with 8000 training points is shown. For comparison the settings are the benchmark settings. The improvement of the prediction with the addition of data is large, along with the stability of predictions. In terms of error distributions there seems to be little difference between using four experts with 2000 training points each and

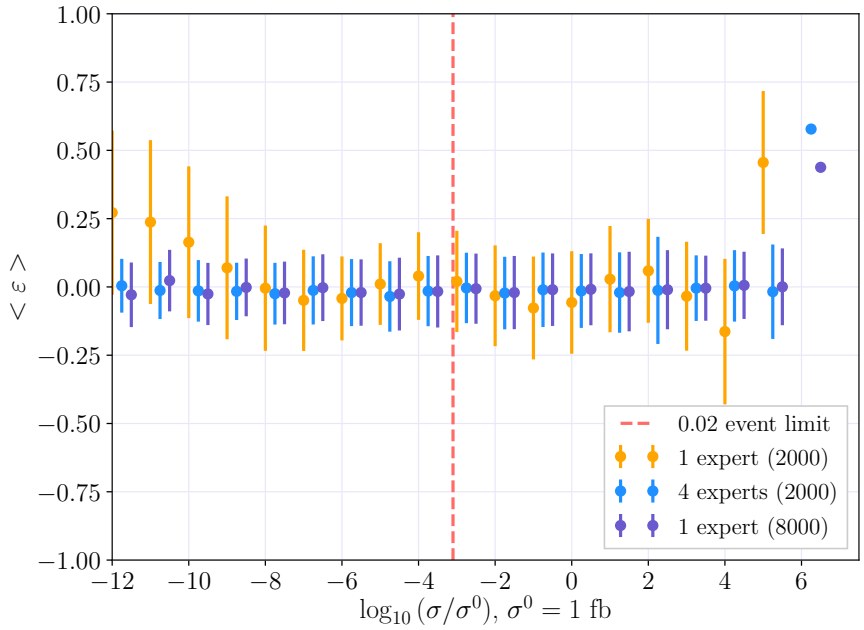


Figure 5.10: The error distributions of GP with the BM settings, using 1 (orange) and 4 (experts).

Number of experts	Points per expert	Time
1	2000	00:03:32
4	2000	00:05:46
1	8000	01:35:21

Table 5.6: Table of computation times for GP fit and prediction on Abel.

using a single expert with 8000 training points. The difference in computation times, however, is very large. Four experts with 2000 points take a little under six minutes to compute, while the single expert with 8000 points takes over an hour and a half. Computation times are shown in Tab. (5.6).

5.4.1 Cross validation for DGP

There is no `scikit-learn` function for distributed Gaussian processes, so an algorithm for k-fold cross validation as a function of experts was implemented. The algorithm uses the `scikit-learn` function `KFold` to find training and test indices for k splits of the data, and is found in the sub-library `model_selection`. For k folds it is implemented in the following way

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=k, random_state=42)
```

The loss function used in the CV is the R^2-score, introduced earlier, and pseudocode for the algorithm is found in Alg. (5). Learning curves for the optimal settings are found in Fig. (5.11) for $\tilde{d}_L\tilde{d}_L$ and Fig. (5.12) for $\tilde{d}_L\tilde{u}_L$, with 500, 1000 and 2000 points per expert.

Data: $N_{experts}$ (max number of experts), n (training points per expert), X (inputs), $\mathbf{y}$ (targets), k (number of folds for cross validation)
number of experts $\mathbf{n} = [1, ..., N_{experts}]$;
for *each number of experts* i **do**
  Training size $= n \cdot (i + 1)$;
  Total size = training size $\cdot \frac{k}{k-1}$;
  Split training data into subsets;
  Use KFold to create k-fold cross validation instance kf;
  for *training indices, test indices in* kf **do**
    Fit GP to $k - 1$ folds of training data;
    Use 1 fold as test data;
    Predict values $\hat{y}_{train}$ for training data;
    Predict values $\hat{y}_{test}$ for test data;
    $R^2_{\text{train}}(\hat{y}, y) = 1 - \frac{\sum_{i=0}^{\text{Training size}-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{\text{Training size}-1}(y_i - \bar{y})^2}$;
    $R^2_{\text{test}}(\hat{y}, y) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}$;
  end
  Find mean m and std σ_{std} of R^2_{test}-values and R^2_{train}-values
end
Result: $\mathbf{n}, \mathbf{m}(R^2_{\text{train}})$, $\boldsymbol{\sigma}_{std}(R^2_{\text{train}})$, $\mathbf{m}(R^2_{\text{test}})$, $\boldsymbol{\sigma}_{std}(R^2_{\text{test}})$.

Algorithm 5: Pseudocode for k-fold cross validation of distributed Gaussian processes, which calculates the R^2-scores for training and test data as a fucntion of the number of experts, to be used for *e.g.* learning curves. In the R^2-score calculation, y_i are true values, $\hat{y}_i$ are GP predicted values, and $\bar{y}$ is the mean of all y_i.

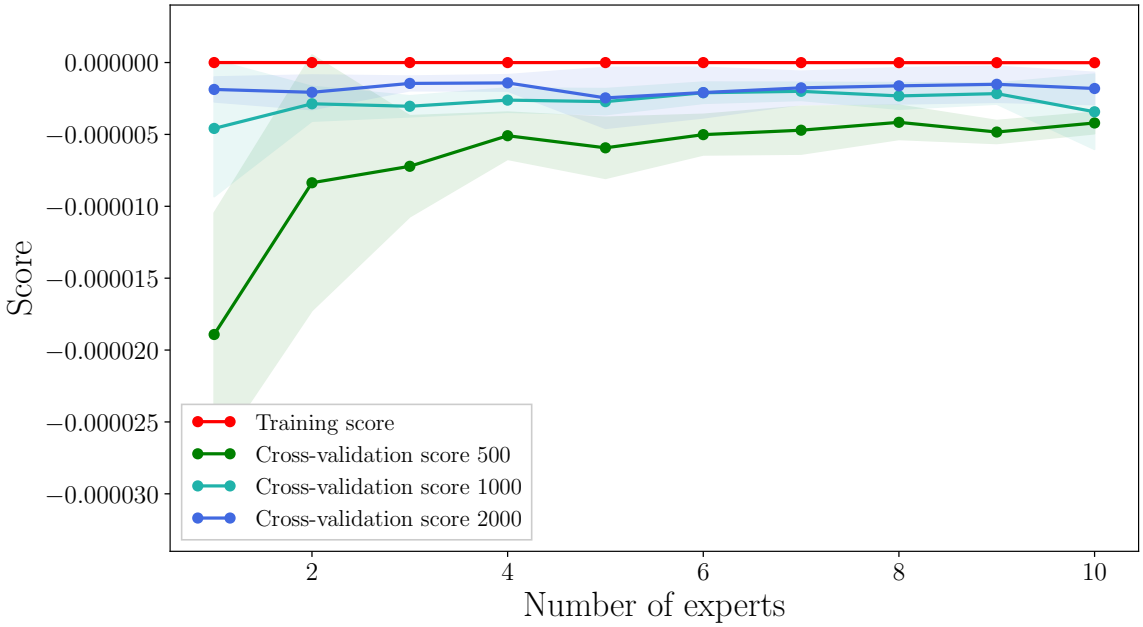


Figure 5.11: Learning curves as a function of number of experts, with 500 and 1000 training points per expert for $\tilde{d}_L\tilde{d}_L$. k-fold cross validation uses R^2-score, but here $R^2 - 1$ is plotted.

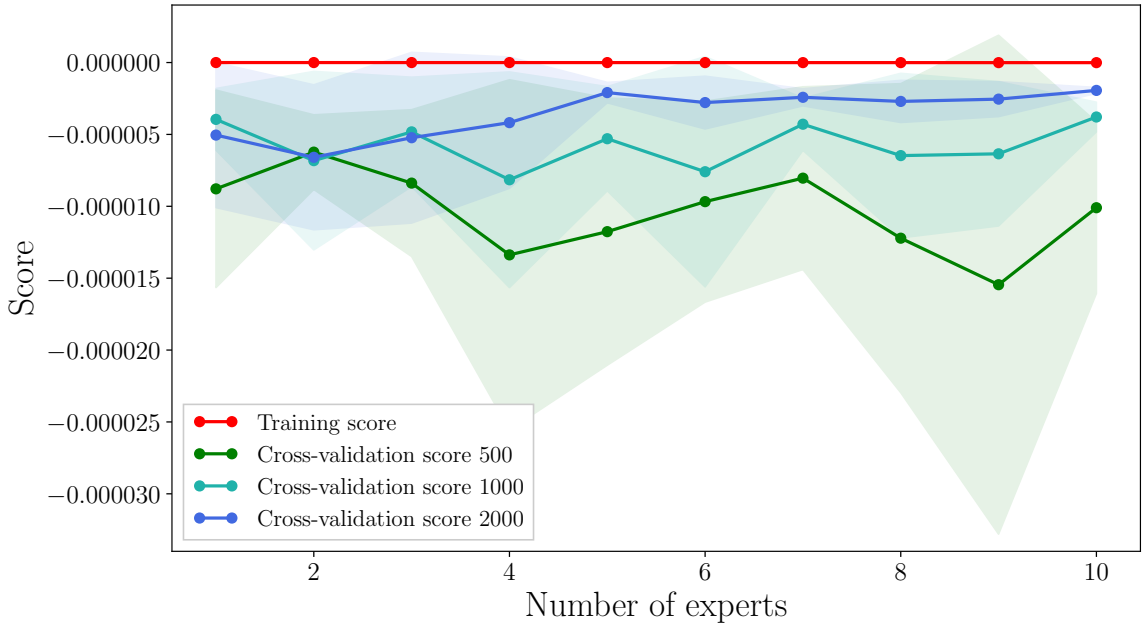


Figure 5.12: Learning curves as a function of number of experts, with 500 and 1000 training points per expert for $\tilde{d}_L\tilde{u}_L$. k-fold cross validation uses R^2-score, but here $R^2 - 1$ is plotted.

Bibliography

[1] Paul Batzing and Are Raklev. Lecture notes for fys5190/fys9190. 2017.

[2] Marc Peter Deisenroth and Jun Wei Ng. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.

[3] Anders Kvellestad. Chasing susy through parameter space. 2015.

[4] C. Patrignani et al. Review of Particle Physics. *Chin. Phys.*, C40(10):100001, 2016.

[5] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[6] Devinderjit Sivia and John Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.

[7] Steven Weinberg. *The Quantum Theory of Fields*, volume 1. Cambridge University Press, 1995.