

Contents

1	Gaussian Processes	1
1.1	Introduction to Bayesian Statistics	1
1.1.1	Bayes' Theorem	2
1.1.2	Priors and Likelihood	3
1.1.3	Best Estimate and Reliability	3
1.1.4	Covariance	5
1.2	Gaussian Process Regression	6
1.2.1	Gaussian Noise Model	11
1.3	Covariance Functions	11
1.3.1	The Squared Exponential Covariance Function	12
1.3.2	The Matérn Class	13
1.4	Model Selection	14
1.4.1	Bayesian Model Selection	15
1.4.2	Log Marginal Likelihood	16
1.4.3	Cross Validation	16
1.4.4	Loss Functions	18
1.5	Distributed Gaussian Processes	18
1.5.1	Limitations of Gaussian Processes	18
1.5.2	Product-of-Experts	18
1.5.3	Algorithm	19
1.5.4	Implementing the Algorithm	19
1.5.5	Benchmark	19

Chapter 1

Gaussian Processes

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

1.1 Introduction to Bayesian Statistics

As it turns out, statistics are not merely statistics. In fact, there is great disagreement between the *frequentist* statisticians and the *Bayesian* statisticians as to what statistics should be. In order to understand the dispute, first consider a statement they *do* agree on

Statisticians use probability to describe uncertainty.

This seems reasonable. Where statisticians reach disagreement is at the definition of the *uncertain*. Defining the uncertain defines probability, and one can distinguish between two branches, namely *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 from person to person depending on how much information one has about the system. One of the main critiques of subjective probability posed by frequentists is that the final probability depends on who you ask.

1.1.1 Bayes' Theorem

To further illustrate the difference a soundly Bayesian framework is introduced, namely *Bayes' theorem*. Bayes' theorem can be derived from the familiar rules of probability

$$P(X|I) + P(\bar{X}|I) = 1, \quad (1.1)$$

$$P(X, Y|I) = P(X|Y, I) \times P(Y|I), \quad (1.2)$$

commonly known as the *sum rule* and *product rule*, respectively. $P(X|I)$ means the probability of outcome X given the information I , and $P(X|Y, I)$ means 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 probability of the outcome X happening plus the probability of X not happening is equal to 1. This is rather untuitive, considering an event either takes place or it doesn't. 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 already happened. These expressions can easily be combined into Bayes' theorem, first formulated by reverend Thomas Bayes in 1763,

$$P(X|Y, I) = \frac{P(Y|X, I) \times P(X|I)}{P(Y|I)}. \quad (1.3)$$

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 . So what is Bayesian about Bayes' theorem? *Nothing*. It merely reformulates the rules of logical consistent reasoning stated by Richard Cox in 1946 [4]. 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

$$P(\Theta = \theta|X = x) = \frac{P(X = x|\Theta = \theta)P(\Theta = \theta)}{P(X = x)}, \quad (1.4)$$

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, both of which will be revisited later. In other words, Eq. (1.4) states the probability of the parameters θ , given the knowledge of outcomes x .

A crucial parting of the Bayesians from the frequentists is at the introduction of the *prior*, which expresses a probability distribution on the *parameters* of the probability distribution. Fret not, this will be further explained.

1.1.2 Priors and Likelihood

The likelihood $P(X = x|\Theta = \theta)$ is simply the probability of the observations given the parameters. The prior, on the other hand, expresses a prior belief or assumption of the data, and has to be determined beforehand. The measure $P(\Theta = \theta|X = x)$ from Eq. (1.4) 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 has now grown tired of throwing dice, and starts to throw a coin. Before throwing he assumes the probability of getting heads or tails is equal, and so adopts a flat prior. After one throw he gets heads, and the posterior changes to a function with high probability for heads, and low for tails. Another heads makes the posterior even narrower around 1, but after several throws the distribution converges to a narrow peak around 0.25, as can be seen in Fig. (1.1), along with other examples of priors. This indicates an unfair coin. Note that the different priors arrive at the same posterior after a large amount of throws.

1.1.3 Best Estimate and Reliability

Now that the posterior distribution has been defined, it would be useful to quantify how well it actually fits the data. For that purpose the *best estimate* and *reliability* are introduced. The best estimate X_0 is the outcome with the highest probability. In other words, it is the maximum of the posterior

$$\left. \frac{dP}{dX} \right|_{X_0} = 0, \quad \left. \frac{d^2P}{dX^2} \right|_{X_0} < 0. \quad (1.5)$$

The second derivative must be negative in order to insure that it is, in fact, a maximum. Once the best estimate has been found, one should ask oneself how reliable this estimate is. Reliability is represented by the width of the distribution, or how much the data is smeared out. A very narrow distribution has very low uncertainty, while a wide distribution has large uncertainty. The width is found using a Taylor expansion of the posterior, and taking the logarithm ¹

$$L = L(X_0) + \frac{1}{2} \left. \frac{d^2}{dx^2} L \right|_{X_0} (X - X_0)^2 + \dots, \quad L = \log_e [\text{prob}(x|\{\text{data}\}, I)] \quad (1.6)$$

This gives a proximate posterior which is a *Gaussian distribution*, with mean and variance given by

$$\mathcal{N}(\mu, \sigma) \text{ where } \mu = X_0, \sigma = \left(- \left. \frac{d^2L}{dx^2} \right|_{X_0} \right)^{-1/2}. \quad (1.7)$$

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

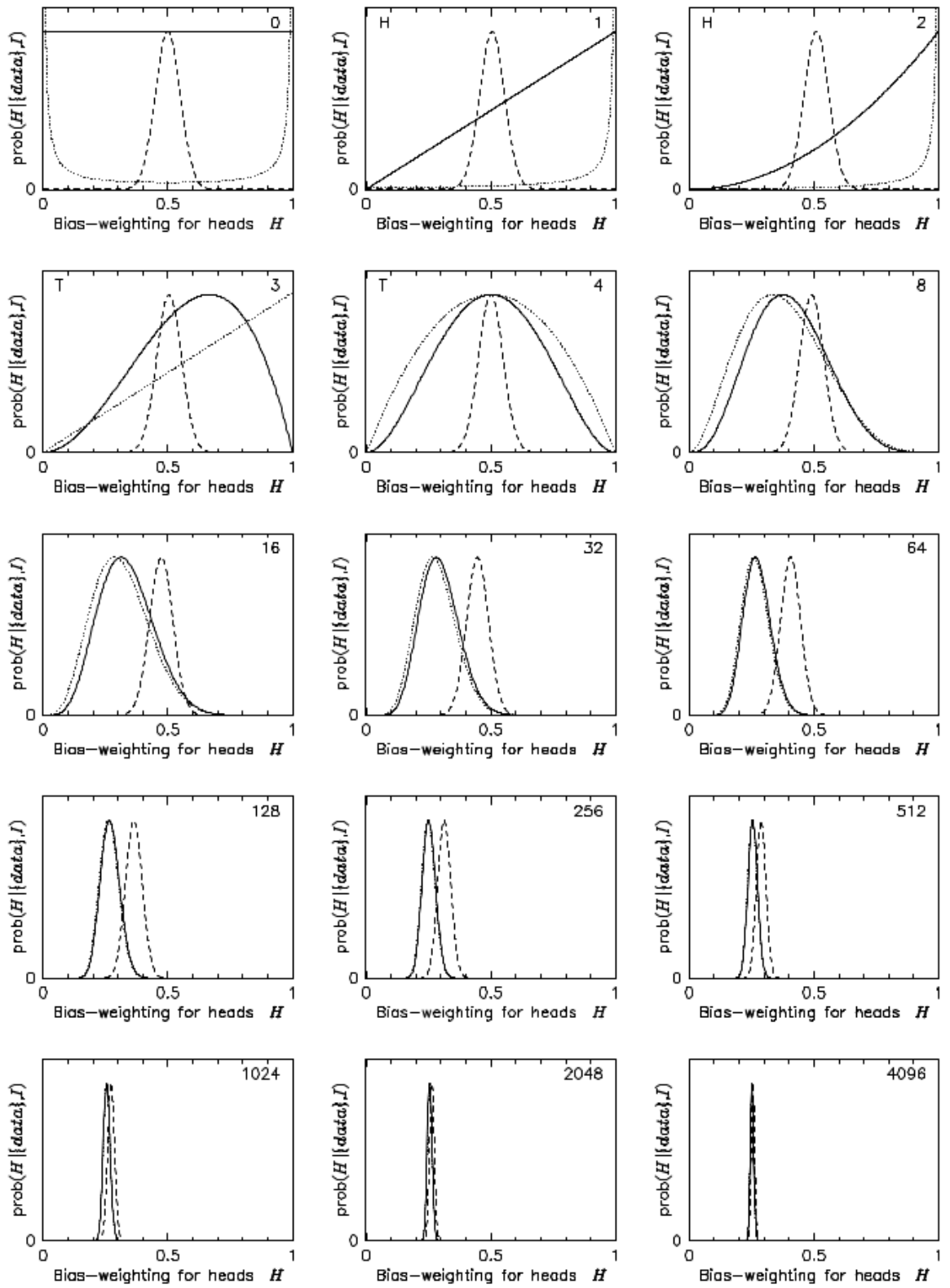


Figure 1.1: The effect of different priors, $P(H|I)$, on the posterior pdf for the bias-weighting of a coin. The solid line is a flat prior, the dashed line assumes a fair coin, and the dotted line is the Jeffreys prior. Figure from [4].

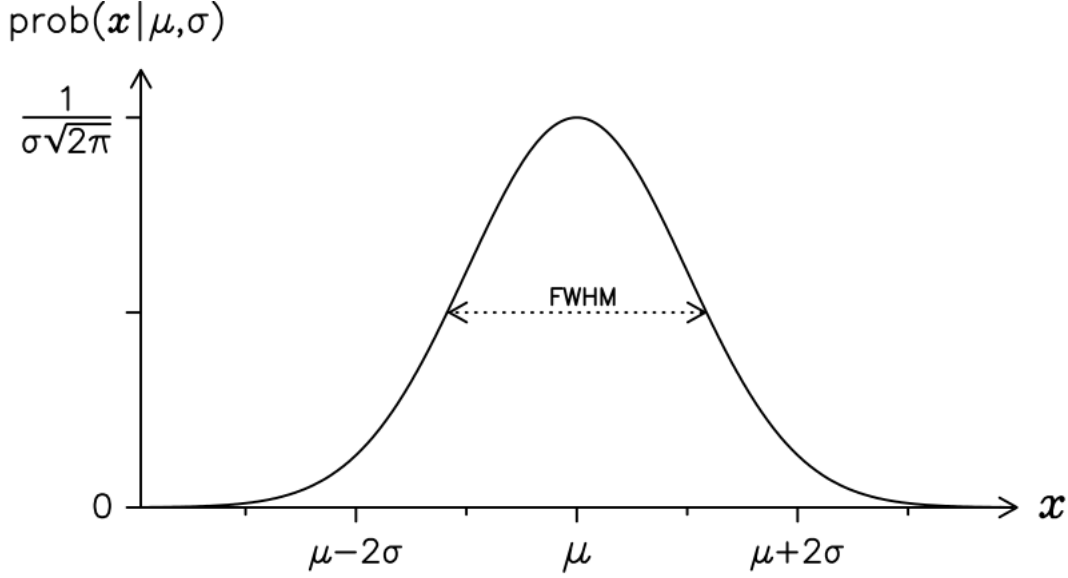


Figure 1.2: A Gaussian probability distribution. The maximum is at the mean value μ , with a full width at half maximum (FWHM) at around 2.35σ . Figure from [4].

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. (1.2).

1.1.4 Covariance

The final term needed before embarking on Gaussian processes is *covariance*. Covariance is discussed in more detail in [4], a brief summary is given here. In most cases the expressions one has to solve are not as simple as Eq. (1.6), as the probability distribution will normally have several quantities of interest $\{X_i\}$. In that case, one has to solve a set of *simultaneous equations* in order to get the best estimate

$$\left. \frac{dP}{dX_i} \right|_{X_{0j}} = 0. \quad (1.8)$$

As a simple example consider the problem in two dimensions, so that $\{X_i\} = (X, Y)$. The Taylor expansion of L is then

$$\begin{aligned} L = & L(X_0, Y_0) + \frac{1}{2} \left[\left. \frac{d^2 L}{dX^2} \right|_{X_0, Y_0} (X - X_0)^2 \right. \\ & \left. + \left. \frac{d^2 L}{dY^2} \right|_{X_0, Y_0} (Y - Y_0)^2 + 2 \left. \frac{d^2 L}{dX dY} \right|_{X_0, Y_0} (X - X_0)(Y - Y_0) \right] + \dots \end{aligned} \quad (1.9)$$

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 part of 1.9 in matrix form gives

$$Q = \begin{pmatrix} X - X_0 & Y - Y_0 \end{pmatrix} \begin{pmatrix} A & C \\ C & B \end{pmatrix} \begin{pmatrix} X - X_0 \\ Y - Y_0 \end{pmatrix}, \quad (1.10)$$

where the matrix elements are

$$A = \frac{\partial^2 L}{\partial X^2} \Big|_{X_0, Y_0}, \quad B = \frac{\partial^2 L}{\partial Y^2} \Big|_{X_0, Y_0}, \quad C = \frac{\partial^2 L}{\partial X \partial Y} \Big|_{X_0, Y_0}. \quad (1.11)$$

The *variance* is defined as the expectation value of the square of deviations from the mean. In the two-dimensional case this becomes [4]

$$\text{Var}(X) = \sigma_x^2 = \langle (X - X_0)^2 \rangle = \int \int (X - X_0)^2 P(X, Y | \{\text{data}\}, I) dX dY. \quad (1.12)$$

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 the two-dimensional case

$$\sigma_{XY}^2 = \langle (X - X_0)(Y - Y_0) \rangle = \int \int (X - X_0)(Y - Y_0) P(X, Y | \{\text{data}\}, I) dX dY. \quad (1.13)$$

The covariance indicates how an over- or underestimation of one parameter affects the other 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. (1.3). It can be shown that [4]

$$\text{cov} = \begin{pmatrix} \sigma_X^2 & \sigma_{XY}^2 \\ \sigma_{XY}^2 & \sigma_Y^2 \end{pmatrix} = - \begin{pmatrix} A & C \\ C & B \end{pmatrix}^{-1}. \quad (1.14)$$

This is called the *covariance matrix*.

1.2 Gaussian Process Regression

Now that some basics of Bayesian statistics have been covered it is time to delve into the subject of Gaussian processes. Gaussian processes (GP) is a supervised

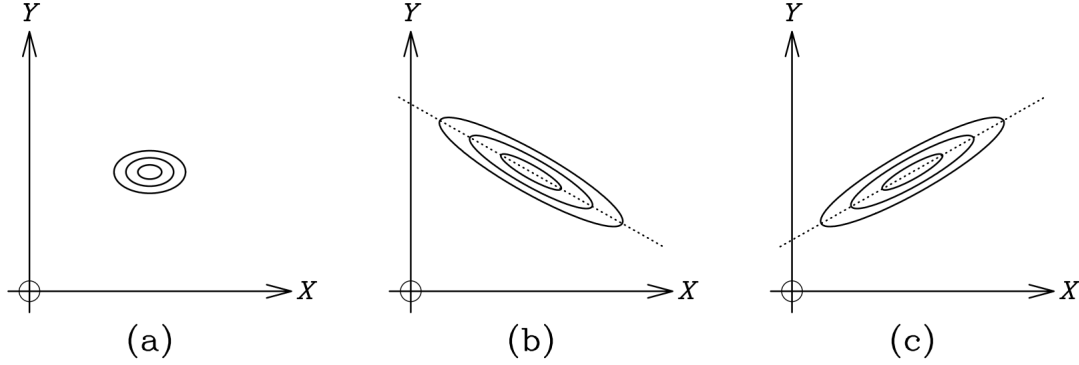


Figure 1.3: A schematic illustration of covariance and correlation. (a) The contours of a posterior pdf with zero covariance, where the inferred values of X and Y are uncorrelated. (b) The corresponding plot when the covariance is large and negative; $Y + mX = \text{constant}$ along the dotted line (where $m > 0$), 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 $Y - mX$; this is constant along the dotted line. Figure from [4]

machine learning method, designed to solve regression and probabilistic classification problems, although this thesis will only use regression. As experience dictates that Gaussian processes can be hard to grasp, it will serve both the reader and the author well to begin on a conceptual level: *what do Gaussian processes do?*

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 Fig. (1.4). 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 . One can view the predicted target y^* as a linear combination of the known targets y_i , where the weights are the covariances between \mathbf{x}_i and the test point \mathbf{x}^* .

For a real process $f(\mathbf{x})$ the mean $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ are defined as

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \quad (1.15)$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \quad (1.16)$$

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

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \quad (1.17)$$

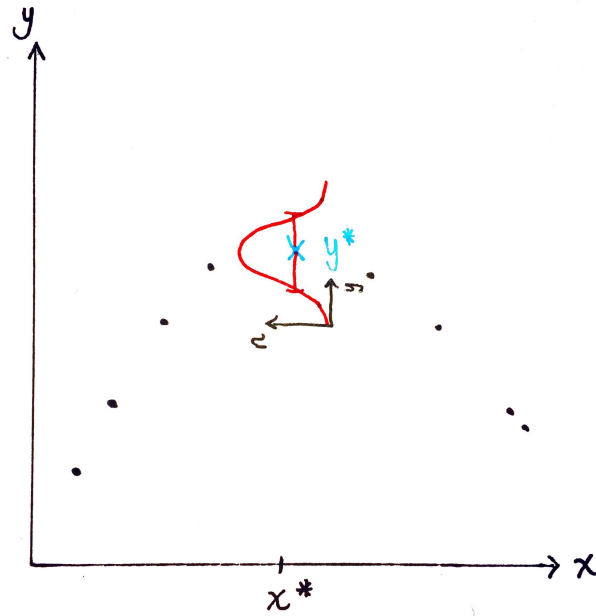


Figure 1.4: An illustration of a GP prediction of the target value y^* (blue cross), given the known set of points $\{x_i, y_i\}$ (black dots). The prediction is a Gaussian distribution in y with mean y^* and variance σ^2 . The Gaussian distribution is drawn in red with y on the vertical axis and the number of events n on the horizontal axis, with uncertainty in the y -direction.

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

$$k(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}|\mathbf{x}_i - \mathbf{x}_j|^2\right). \quad (1.18)$$

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 [3]. 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

$$f(x) \sim \mathcal{N}(0, K(X^*, X^*)). \quad (1.19)$$

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. (1.5) samples are drawn from both the prior and posterior distribution. Samples are here 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 their mean value 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.

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

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) & K(X, X^*) \\ K(X, X^*) & K(X^*, X^*) \end{bmatrix}\right) \quad (1.20)$$

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 obtained⁴ [3]

$$\mathbf{f}_* | X_*, X, \mathbf{f} \sim \mathcal{N}(K(X_*, X)K(X, X)^{-1}\mathbf{f}, \quad (1.21)$$

$$K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)). \quad (1.22)$$

²The mean does not have to be zero, it could for example be the mean of the training data.

³Assuming there is no noise in the data.

⁴For more details, see Appendix A.2 in [3].

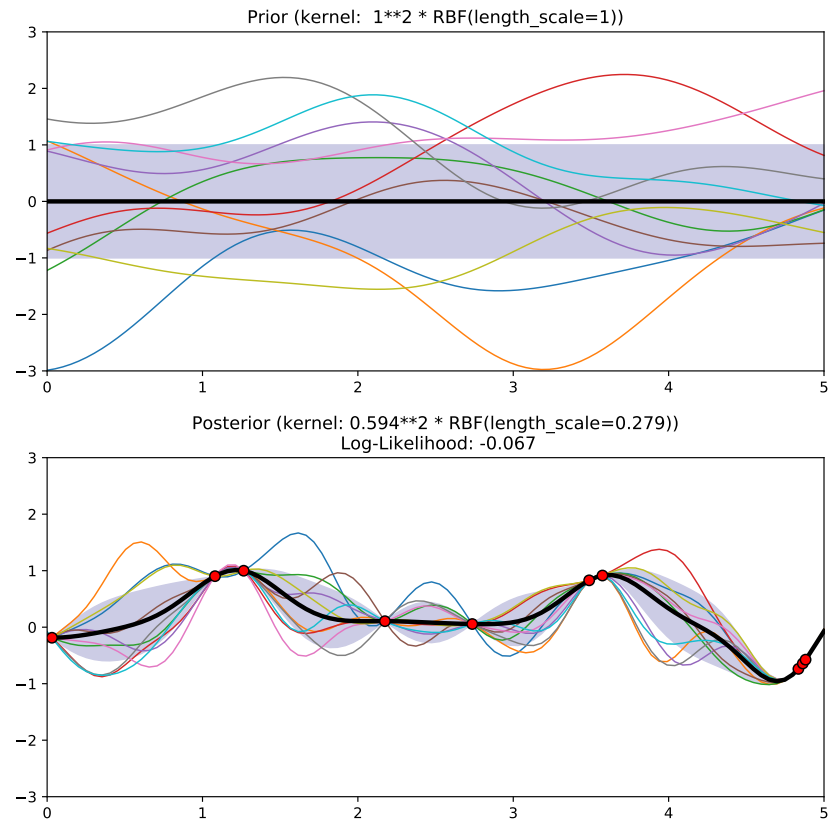


Figure 1.5: Drawing functions from the prior (top) and posterior (bottom) distributions. The thick, black line represents the mean of the distribution, while the shaded, blue area is the uncertainty. The multiple colored lines are functions drawn randomly from the prior and posterior distributions, whose 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.

1.2.1 Gaussian Noise Model

In the real world, noise-free observations are rare. In many cases, targets will be on the form $y = f(\mathbf{x}) + \varepsilon$, where the noise ε is assumed to follow a Gaussian distribution $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$. This is the *Gaussian noise model*. The covariance can then be expressed as

$$\text{cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij} \quad (1.23)$$

which gives for the prior distribution

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) + \sigma_n^2 \mathbb{I} & K(X, X^*) \\ K(X, X^*) & K(X^*, X^*) \end{bmatrix}\right). \quad (1.24)$$

The conditioned distribution is then

$$\mathbf{f}^* | X^*, X, \mathbf{f} \sim \mathcal{N}(\bar{\mathbf{f}}^*, \text{cov}(\mathbf{f}^*)), \text{ where} \quad (1.25)$$

$$\bar{\mathbf{f}}^* = K(X^*, X)[K(X, X) + \sigma_n^2 \mathbb{I}]^{-1} \mathbf{y}, \quad (1.26)$$

$$\text{cov}(\mathbf{f}^*) = K(X^*, X^*) - K(X^*, X)[K(X, X) + \sigma_n^2 \mathbb{I}]^{-1} K(X, X^*) \quad (1.27)$$

In order to tidy up the expression somewhat, 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

$$\bar{f}^* = \mathbf{k}^{*T} (K + \sigma_n^2 \mathbb{I})^{-1} \mathbf{y}, \quad (1.28)$$

$$\mathbb{V}[f^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^{*T} (K + \sigma_n^2 \mathbb{I})^{-1} \mathbf{k}^*. \quad (1.29)$$

Note that, as previously mentioned, the prediction of the 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. (1.28)-(1.29) form the basis for GP prediction in `scikit-learn` [2]. The algorithm is shown in Algorithm (1), and uses the Cholesky decomposition of the covariance matrix.

1.3 Covariance Functions

Covariance functions and kernels have been mentioned a few times already, and it is time to further investigate them. 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

⁵Invariant to rigid rotations in input space

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 [3].

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

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \quad (1.30)$$

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

$$k(\mathbf{x}_i, \mathbf{x}_j)_{noise} = C\delta_{ij}, \quad (1.31)$$

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.

1.3.1 The Squared Exponential Covariance Function

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

$$k_{SE}(r) = \exp\left(-\frac{r^2}{2\ell^2}\right), \quad (1.32)$$

where ℓ is the *characteristic length scale*. The length scale determines how smooth the function is, for example, for a large length scale one should expect a very slowly varying function, while a shorter length scale means a more rapidly varying function, see Fig. (1.6). The SE is infinitely differentiable, and so is very smooth.

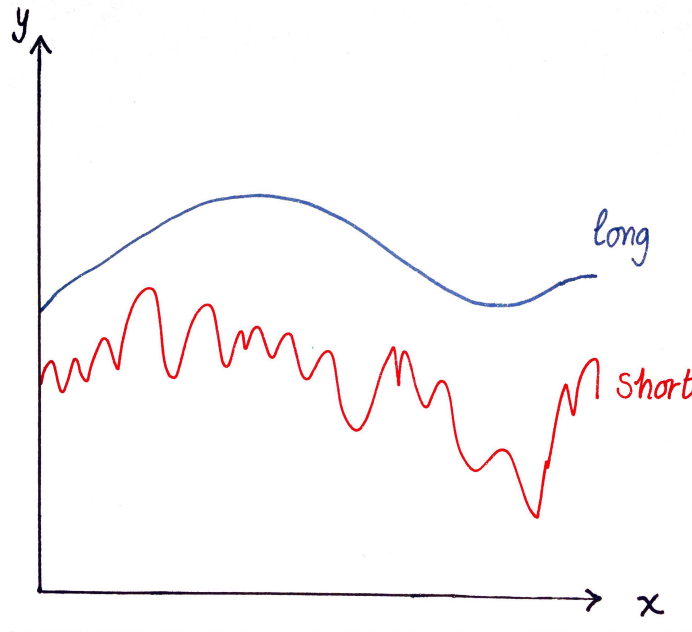


Figure 1.6: The effect of varying the length scale ℓ . A large length scale (blue) gives a smooth, slowly varying function, while a small length scale (red) gives a more staccato, quickly varying function.

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))
```

1.3.2 The Matérn Class

The *Matérn class of covariance functions* is given by

$$k_{\text{Matérn}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}r}{\ell} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}r}{\ell} \right), \quad (1.33)$$

where $\nu, \ell > 0$, and K_ν is a modified Bessel function. For $\nu \rightarrow \infty$ this becomes the SE. In the case of ν being half integer, $\nu = p + \frac{1}{2}$, the covariance function is simply the product of an exponential and a polynomial

$$k_{\nu=p+\frac{1}{2}} = \exp \left(-\frac{\sqrt{2\nu}r}{\ell} \right) \frac{\Gamma(p+1)}{\Gamma(2p+1)} \sum_{i=0}^p \frac{(p+i)!}{i!(p-i)!} \left(\frac{\sqrt{8\nu}r}{\ell} \right)^{p-i}. \quad (1.34)$$

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

$$k_{\nu=3/2}(r) = \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right), \quad (1.35)$$

$$k_{\nu=5/2}(r) = \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right). \quad (1.36)$$

The Matérn kernel is considered more appropriate for physical processes [3], 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)
```

Other Kernels

There are other 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 [3].

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)

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T M (\mathbf{x}_i - \mathbf{x}_j) + \sigma_n^2 \delta_{ij}\right). \quad (1.37)$$

The matrix M can have several forms, for example

$$M_1 = \ell^{-2} \mathbb{I}, \quad M_2 = \text{diag}(\boldsymbol{\ell})^{-2}. \quad (1.38)$$

Choosing $\boldsymbol{\ell}$ to be a vector in stead of a scalar can in many cases be 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} .

1.4 Model Selection

It becomes apparent that the choice of kernel and hyperparameters is important for the quality of the GP prediction. Model selection means finding the hyperparameters and kernel that best fit the data. In this section Bayesian model selection is quickly overviewed, and the log marginal likelihood and cross validation are considered for their ability to optimize the GP.

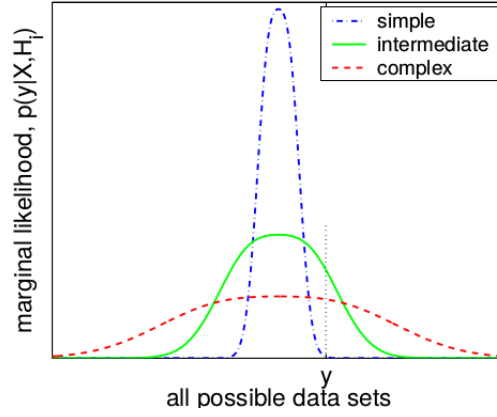


Figure 1.7: The marginal likelihood is the probability of the data, given the model. The number of data points n and inputs X are fixed. The horizontal axis represents an idealized set of all possible vectors of targets \mathbf{y} . Since the marginal likelihood is a probability distribution it must normalize to unity. For a particular set \mathbf{y} , indicated by the dotted gray line, the intermediate model is preferred to the simple and complex ones. Figure from [3].

1.4.1 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 [3],

$$p(\mathbf{w}|\mathbf{y}, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)}{p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)}, \quad (1.39)$$

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

$$p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)d\mathbf{w} \quad (\text{marginal likelihood}). \quad (1.40)$$

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. (1.7).

1.4.2 Log Marginal Likelihood

Gaussian process regression models with Gaussian noise have the wonderful trait of providing analytically tractable integrals for the marginal likelihood. The exact expression for the log marginal likelihood ⁶ can be shown to be [3]

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T K_y^{-1} \mathbf{y} - \frac{1}{2} \log |K_y| - \frac{n}{2} \log 2\pi. \quad (1.41)$$

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 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)

$$\frac{\partial}{\partial \theta_j} \log p(\mathbf{y}|X, \boldsymbol{\theta}) = \frac{1}{2} \mathbf{y}^T K^{-1} \frac{\partial K}{\partial \theta_j} K^{-1} \mathbf{y} - \frac{1}{2} \text{tr}(K^{-1} \frac{\partial K}{\partial \theta_j}). \quad (1.42)$$

Computing the inverse of a matrix required here, 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. (1.8). These correspond to different interpretations of the data. The rightmost optima in Fig. (1.8) 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 optima 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 high 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.

1.4.3 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 $k-1$ to train the model, and a single set to validate it. This is repeated k times. Cross-validation requires a loss function, such as the mean relative deviance.

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 cross validation score uses the R^2 value, given by

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1} (y_i - \bar{y})^2}, \quad (1.43)$$

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

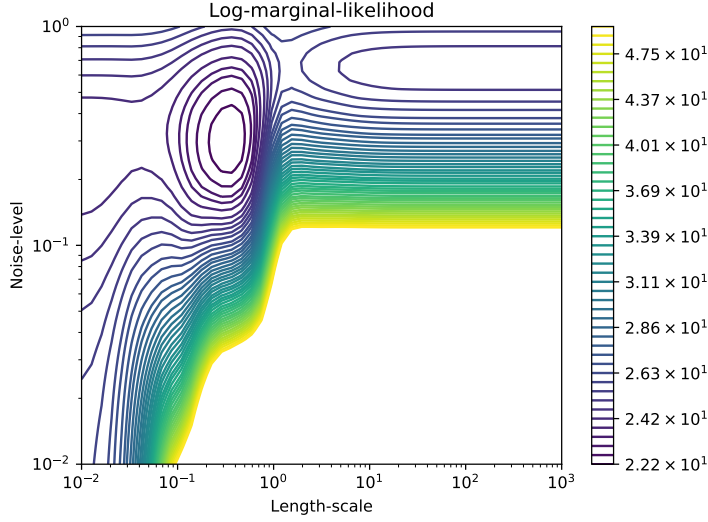


Figure 1.8: A contour plot of the log marginal likelihood with two local optima. The rightmost optima favours a short length scale and low noise, while the leftmost favors a high noise level and therefore several large length scales. Plot generated using scikit-learn.

where \hat{y}_i is the predicted value of the i th sample, y_i is the true value and $\bar{y} = \frac{1}{N} \sum_{i=0}^{N-1} y_i$ for N samples. The learning curve plots the training score and validation score, which are 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. *Underfitting* occurs when the model is not able to capture the underlying structure of the data.

Examples of learning curves are shown in Fig. (1.9) for Naive Bayes and SVM estimators. 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.

1.4.4 Loss Functions

In this project the main loss function used is the relative deviance. For true values f_{true} and the GP prediction f_{GP} this is given by

$$\varepsilon = \frac{f_{true} - f_{GP}}{f_{true}}. \quad (1.44)$$

The relative deviance is used because of the large span of the target values, ranging from about 10^{-30} to 10^9 .

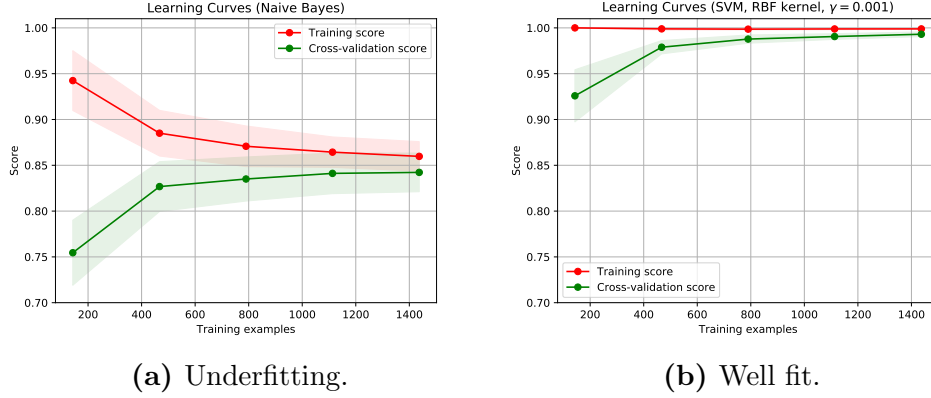


Figure 1.9: Learning curves for two different estimators.

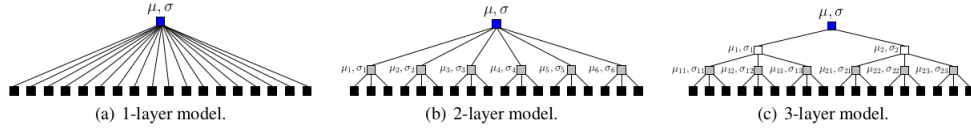


Figure 1.10: From [1].

R-Factor

1.5 Distributed Gaussian Processes

1.5.1 Limitations of Gaussian Processes

Problem because of $(K + \sigma_n^2 \mathbb{I})^{-1}$, means inverting an $n \times n$ -matrix. Training and predicting limits of $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$. Limit = $\mathcal{O}(10^4)$. Some solutions exist, but **no prediction of variance is given with p-o-e**.

1.5.2 Product-of-Experts

Divide data between experts. "The assumption of independent GP experts leads to a block-diagonal approximation of the kernel matrix, which (i) allows for efficient training and predicting (ii) can be computed efficiently (time and memory) by parallelisation" [1].

Independence assumption

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \approx \prod_{k=1}^M p_k(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) \quad (1.45)$$

$$\log p(\mathbf{y}^{(k)}|\mathbf{X}^{(k)}, \boldsymbol{\theta}) = -\frac{1}{2} \mathbf{y}^{(k)} (\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2 \mathbf{I})^{-1} \mathbf{y}^{(k)} - \frac{1}{2} \log |\mathbf{K}_{\psi}^{(k)} + \sigma_{\varepsilon}^2 \mathbf{I}| \quad (1.46)$$

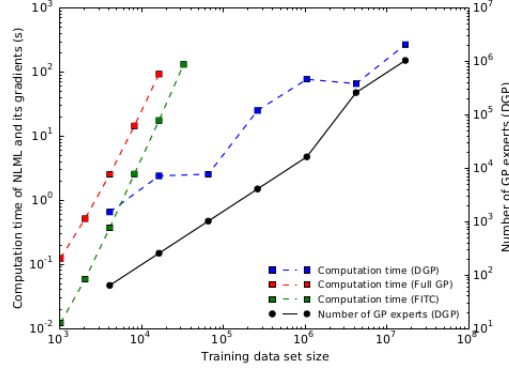


Figure 1.11: From [1].

1.5.3 Algorithm

$$\mu_*^{rbcm} = (\sigma_*^{rbcm})^2 \sum_k \beta_k \sigma_k^{-2}(\mathbf{x}_*) \mu_k(\mathbf{x}_*), \quad (1.47)$$

$$(\sigma_*^{rbcm})^{-2} = \sum_{k=1}^M \beta_k \sigma_k^{-2}(\mathbf{x}_*) + \left(1 - \sum_{k=1}^M \beta_k\right) \sigma_{**}^{-2}. \quad (1.48)$$

The posterior distribution for the test point \mathbf{x}_* is given by a Gaussian with mean and variance

$$\mu(\mathbf{x}_*) = \mathbf{k}_*^T (\mathbf{K} + \sigma_\epsilon^2 \mathbb{I})^{-1} \mathbf{y}, \quad (1.49)$$

$$\sigma^2(\mathbf{x}_*) = k_{**} - \mathbf{k}_*^T (\mathbf{K} + \sigma_\epsilon^2 \mathbb{I})^{-1} \mathbf{k}_*. \quad (1.50)$$

Plot of time from the article

1.5.4 Implementing the Algorithm

Parallelizing

1.5.5 Benchmark

```

Data:  $N_{experts}$  (number of experts),  $X$  (inputs),  $\mathbf{y}$  (targets),  $k$  (covariance
        function/kernel),  $\sigma_n^2$  (noise level),  $\mathbf{x}^*$  (test input),  $\mathbf{y}^*$  (test target)
 $X_{train}, X_{test}, y_{train}, y_{test} = \text{train-test-split}(X, y)$  (scikit-learn) ;
 $y = \log_{10}(y)$  ;
 $n = \frac{\text{Number of data points}}{N_{experts}}$  ;
 $subsets = \text{array\_split}(X_{train}, n)$  ;
 $\mu_{rbcm} = []$ ,  $\sigma_{rbcm} = []$  (empty lists to be filled later);
for each expert do
     $gp_{temporary} = \text{GaussianProcessRegressor.fit}(X_{expert}, y_{expert})$  ;
    for each  $y^*$  in  $\mathbf{y}^*$  do
         $\mu_*, \sigma_*^2 = gp_{temporary}.predict(x^*)$  ;
         $\sigma_{**}^2 = k(x^*, x^*)$  ;
        (fill in the values) ;
         $\mu[\text{expert}][x^*] = \mu_*^2$  (mean value from this expert);
         $\sigma^2[\text{expert}][x^*] = \sigma_*^2$  (variance from this expert);
         $\sigma_{**}^2[\text{expert}][x^*] = \sigma_{**}^2$  (variance from initial kernel)
    end
end
for each expert do
    for each  $y_*$  in  $\mathbf{y}_*$  do
         $\mu_* = \mu[\text{expert}][x_*]$  (retrieve relevant values);
         $\sigma_*^2 = \sigma^2[\text{expert}][x^*]$  ;
         $\sigma_{**}^2 = \sigma_{**}^2[\text{expert}][x^*]$  ;
         $\beta = \frac{1}{2}(\log(\sigma_{**}^2) - \log(\sigma_*^2))$  ;
         $(\sigma_*^{rbcm})^{-2}[y_*] + = \beta \sigma_*^{-2} + (\frac{1}{n_{experts}} - \beta) \sigma_{**}^{-2}$ 
    end
end
for each expert do
    for each  $y_*$  in  $\mathbf{y}_*$  do
         $\mu_* = \mu[\text{expert}][x_*]$  (retrieve relevant values);
         $\sigma_*^2 = \sigma^2[\text{expert}][x^*]$  ;
         $\sigma_{**}^2 = \sigma_{**}^2[\text{expert}][x^*]$  ;
         $\beta = \frac{1}{2}(\log(\sigma_{**}^2) - \log(\sigma_*^2))$  ;
         $\mu_*^{rbcm}[y_*] + = (\sigma_*^{rbcm})^2 \beta \sigma_*^{-2} \mu_*$ 
    end
end
 $\epsilon = \frac{10^{\mu_{rbcm}} - 10^{y_{test}}}{10^{y_{test}}}$  (relative error);
Result: Approximative distribution of  $f_* = f(\mathbf{x}_*)$  with mean  $\mu_*^{rbcm}$  and
        variance  $(\sigma_*^{rbcm})^2$ .

```

Algorithm 2: Algorithm for using rBCM on a single test point \mathbf{x}_* . The *GaussianProcessRegressor.fit()*-function is a function in scikit-learn, that uses Algorithm (??).

Bibliography

- [1] Marc Peter Deisenroth and Jun Wei Ng. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843*, 2015.
- [2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [3] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.
- [4] Devinderjit Sivia and John Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.