

AutoML: Gaussian Processes

Gaussian Process Training

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Training of a Gaussian Process

- To make predictions for a regression task by a Gaussian process, one simply needs to perform matrix computations.
- But for this to work out, we assume that the covariance functions is fully given, including all of its hyperparameters.
- A very nice property of GPs is that we can learn the numerical hyperparameters of a selected covariance function directly during GP training.

Training a GP via the Maximum Likelihood I

- Let us assume $y = f(\mathbf{x}) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, where $f(\mathbf{x}) \sim \mathcal{G}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}))$.
- Noticing that $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma^2 \mathbf{I})$, we can find the marginal log-likelihood (or evidence):

$$\begin{aligned} \log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) &= \log \left[(2\pi)^{-n/2} |\mathbf{K}_y|^{-1/2} \exp \left(-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} \right) \right] \\ &= -\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi. \end{aligned}$$

with $\mathbf{K}_y := \mathbf{K} + \sigma^2 \mathbf{I}$ and $\boldsymbol{\theta}$ denoting the parameters of the covariance function (i.e., the hyperparameters).

Training a GP via the Maximum Likelihood II

Recalling that the increase of the length-scale reduces the model flexibility, the three terms of the marginal likelihood can be interpreted as follows.

- The data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$. The data fit tends to decrease by increasing the length-scale.
- The complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$, which depends on the covariance function. This term decreases with the increase of the length-scale (the model gets less complex as the length-scale grows).
- The normalization constant $-\frac{n}{2} \log 2\pi$.

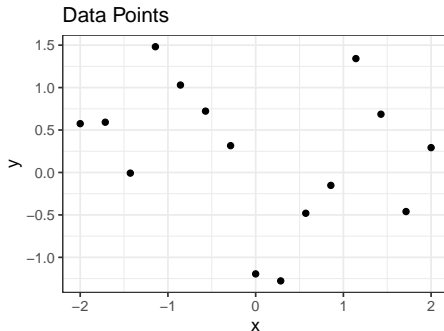
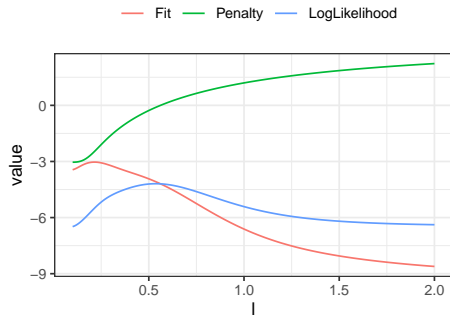
Training a GP: Example I

To visualize this, let us consider a zero-mean GP with a squared exponential kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp \left(-\frac{1}{2\ell^2} \|\mathbf{x} - \mathbf{x}'\|^2 \right).$$

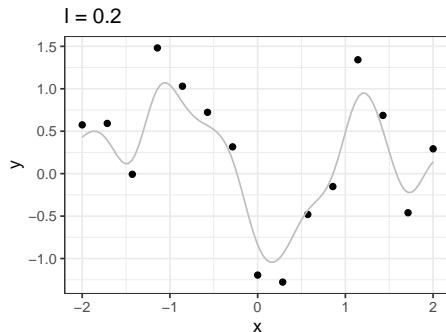
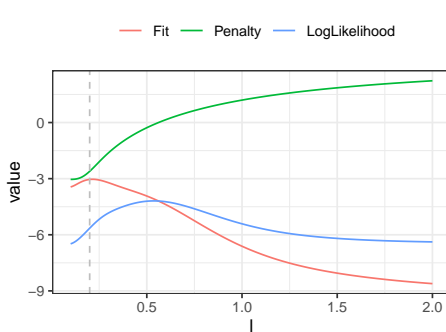
- Recall that the model becomes smoother and less complex as the length-scale ℓ increases.
- We will show how each of the following terms behaves if the value of ℓ increases:
 - ▶ the data fit $-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y}$,
 - ▶ the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$,
 - ▶ the overall value of the marginal likelihood $\log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta})$.

Training a GP: Example II



i The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.

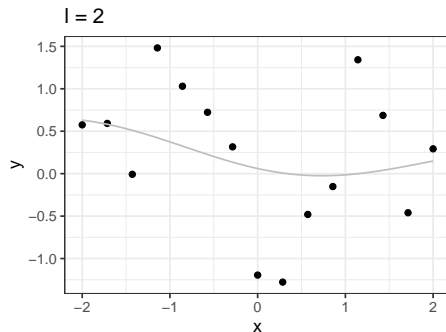
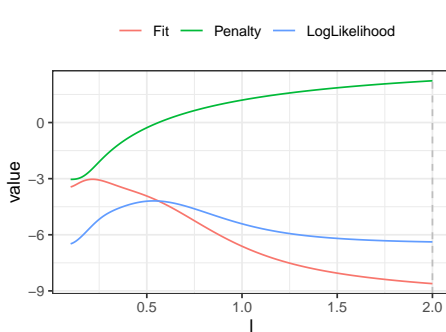
Training a GP: Example III



i The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.

💡 A small ℓ leads to a good fit, but, to a high complexity penalty.

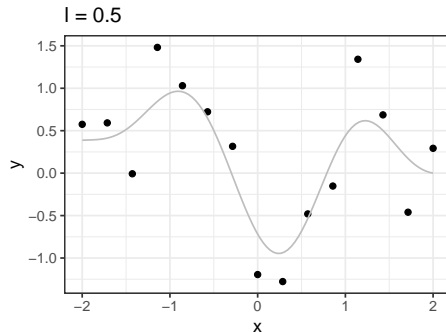
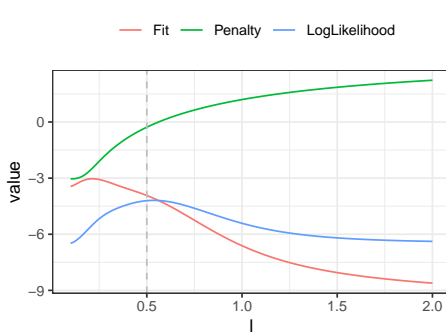
Training a GP: Example IV



i The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.

💡 A large ℓ results in a poor fit.

Training a GP: Example V



i The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.

💡 The maximizer of the log-likelihood ($\ell = 0.5$) balances the complexity and data the fit.

Training a GP via the Maximum Likelihood I

To choose the hyperparameters by maximizing the marginal likelihood, we need to find the partial derivatives of the likelihood w.r.t. the hyperparameters:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \left(-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi \right) \\ &= \frac{1}{2} \mathbf{y}^\top \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1} \mathbf{y} - \frac{1}{2} \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \right) \\ &= \frac{1}{2} \text{tr} \left((\mathbf{K}^{-1} \mathbf{y} \mathbf{y}^\top \mathbf{K}^{-1} - \mathbf{K}^{-1}) \frac{\partial \mathbf{K}}{\partial \theta_j} \right)\end{aligned}$$

💡 Above, we used the following identities:

$$\frac{\partial}{\partial \theta_j} \mathbf{K}^{-1} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1} \text{ and } \frac{\partial}{\partial \boldsymbol{\theta}} \log |\mathbf{K}| = \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}} \right)$$

Training a GP via the Maximum Likelihood II

- The complexity and the runtime of training a Gaussian process is dominated by the computational task of inverting \mathbf{K} - or let's rather say for decomposing it.
- Standard methods require $\mathcal{O}(n^3)$ time (!) for this.
- Once \mathbf{K}^{-1} - or rather the decomposition - is known, the computation of the partial derivatives requires only $\mathcal{O}(n^2)$ time per hyperparameter.
- 💡 Thus, the computational overhead of computing derivatives is small, and using a gradient based optimizer is advantageous.

Training a GP via the Maximum Likelihood III

Workarounds to make GP estimation feasible for big data include:

- Using kernels that yield sparse \mathbf{K} : cheaper to invert.
- Subsampling the data to estimate θ ; $\mathcal{O}(m^3)$ for subset of size m .
- Combining estimates on different subsets of size m : **Bayesian committee**; $\mathcal{O}(nm^2)$.
- Exploiting low-rank approximations of \mathbf{K} by using only a representative subset (inducing points) of m training data \mathbf{X}_m : **Nyström approximation** $\mathbf{K} \approx \mathbf{K}_{nm} \mathbf{K}_{mm}^{-1} \mathbf{K}_{mn}$, with $\mathcal{O}(nmk + m^3)$ for a rank-k-approximate inverse of \mathbf{K}_{mm} .
- Utilizing structure in \mathbf{K} induced by the kernel: exact solutions but complicated maths, not applicable for all kernels.

... this is still an active area of research.