Technical University of Denmark

Bayesian machine learning

Project Bayesian Machine Learning

Students :
Emmanuel Minois-Genin s233036
Hongjin Chen s232289
Jialu Chen Christiansen s194175
Chuang Sun Hembo s233427

March 31, 2024

Contents

1 Part 1: Objective functions for regression modelling

Task 1.1 The maximum likelihood estimator (MLE) is given by $\text{argmax}_w\, p(\boldsymbol{y}|\boldsymbol{w})$, since we model our likelihood as a Gaussian distribution (where each training sample is iid) we have:

$$p(\boldsymbol{y}|\boldsymbol{w}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{w}) \tag{1}$$

$$= \prod_{n=1}^{N} \mathcal{N}(y_n|f(\boldsymbol{x}_n), \beta^{-1}) \tag{2}$$

We are then going to apply the logarithm to $p(\boldsymbol{y}|\boldsymbol{w})$ to transform the product into the sum (since the logarithm is a monotonic function, this will not change the maximum), we will also use the density formula for the Gaussian distribution:

$$\ln p(\boldsymbol{y}|\boldsymbol{w}) = \sum_{n=1}^{N} \ln \mathcal{N}(y_n|f(\boldsymbol{x}_n), \beta^{-1}) \tag{3}$$

$$= \sum_{n=1}^{N} \ln \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp^{-\frac{1}{2}\frac{(y_n - f(\boldsymbol{x}_n))^2}{\beta^{-1}}} \tag{4}$$

$$= \sum_{n=1}^{N} \ln \frac{\sqrt{\beta}}{\sqrt{2\pi}} - \sum_{n=1}^{N} \frac{\beta}{2}(y_n - f(\boldsymbol{x}_n))^2 \tag{5}$$

$$= -\frac{\beta}{2} \sum_{n=1}^{N} (y_n - f(\boldsymbol{x}_n))^2 + K \tag{6}$$

Where $K = \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi$ is a constant with respect to $\boldsymbol{w}$ and therefore does not modify the maximum likelihood solution $\hat{\boldsymbol{w}}$. We also recognize the sum of squares $J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - f(\boldsymbol{x}_n))^2$, we can see that likelihood is negatively related to the sum of squares, so maximizing likelihood is equivalent to minimizing the sum of squares. The solution is the same, since the multiplicative constant β doesn't change the solution (the derivative is zero at the same point).

Task 1.2 The maximum a posteriori (MAP) estimator is given by $\text{argmax}_w\, p(\boldsymbol{w}|\boldsymbol{y})$, we can begin by using Bayes theorem $p(\boldsymbol{w}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y})}$ we see the prior $p(\boldsymbol{w})$, the evidence $p(\boldsymbol{y})$ and the likelihood $p(\boldsymbol{y}|\boldsymbol{w})$. Since the evidence $p(\boldsymbol{y})$ is a constant with respect to $\boldsymbol{w}$, we don't need to consider it, as it won't modify the solution $p(\boldsymbol{w}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w})$. Using the Gaussian likelihood for the data and a Gaussian prior we get:

$$p(\boldsymbol{w}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}) \tag{7}$$

$$\propto \prod_{n=1}^{N} \mathcal{N}(y_n|f(\boldsymbol{x}_n), \beta^{-1}) \prod_{i=1}^{M} \mathcal{N}(w_i|0, \alpha^{-1}) \tag{8}$$

$$\tag{9}$$

We will use the logarithm once again to transform the products into sums and we will use the densities of the Normal distribution.

$$\ln p(\boldsymbol{w}|\boldsymbol{y}) \propto \sum_{n=1}^{N} \ln \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp^{-\frac{1}{2}\frac{(y_n - f(\boldsymbol{x}_n))^2}{\beta^{-1}}} + \sum_{i=1}^{M} \ln \frac{1}{\sqrt{2\pi\alpha^{-1}}} \exp^{-\frac{1}{2}\frac{w_i^2}{\alpha^{-1}}} \tag{10}$$

$$\propto \sum_{n=1}^{N} \ln \frac{\sqrt{\beta}}{\sqrt{2\pi}} - \sum_{n=1}^{N} \frac{\beta}{2}(y_n - f(\boldsymbol{x}_n))^2 + \sum_{i=1}^{M} \ln \frac{\sqrt{\alpha}}{\sqrt{2\pi}} - \sum_{i=1}^{M} \frac{\alpha}{2} w_i^2 \tag{11}$$

$$\propto -\frac{1}{2}\sum_{n=1}^{N}(y_n - f(\boldsymbol{x}_n))^2 - \frac{\alpha}{\beta}(\frac{1}{2}\sum_{i=1}^{M} w_i^2) + K \tag{12}$$

Where $K = \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi + \frac{M}{2}\ln\alpha - \frac{M}{2}\ln 2\pi$ is a constant with respect to w. We can see that the maximum a posteriori estimator is indeed equivalent to the solution for ridge regression with $\lambda = \frac{\alpha}{\beta}$.

Task 1.3 We will proceed in the same way as in **Task 1.2**, but using a Laplace prior this time:

$$p(\boldsymbol{w}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}) \tag{13}$$

$$\propto \prod_{n=1}^{N} \mathcal{N}(y_n|f(\boldsymbol{x}_n), \beta^{-1}) \prod_{i=1}^{M} \text{Laplace}(w_i|0, b) \tag{14}$$

$$\tag{15}$$

Once again we apply the logarithm and replace with the density functions:

$$\ln p(\boldsymbol{w}|\boldsymbol{y}) \propto \sum_{n=1}^{N} \ln \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp^{-\frac{1}{2}\frac{(y_n - f(\boldsymbol{x}_n))^2}{\beta^{-1}}} + \sum_{i=1}^{M} \ln \frac{1}{2b} \exp^{-\frac{|w_i|}{b}} \tag{16}$$

$$\propto \sum_{n=1}^{N} \ln \frac{\sqrt{\beta}}{\sqrt{2\pi}} - \sum_{n=1}^{N} \frac{\beta}{2}(y_n - f(\boldsymbol{x}_n))^2 - \sum_{i=1}^{M} \ln 2b - \sum_{i=1}^{M} \frac{|w_i|}{b} \tag{17}$$

$$\propto -\frac{1}{2}\sum_{n=1}^{N}(y_n - f(\boldsymbol{x}_n))^2 - \frac{1}{b\beta}(\sum_{i=1}^{M}|w_i|) + K \tag{18}$$

Where $K = \frac{N}{2}\ln\beta - \frac{N}{2}\ln 2\pi - M\ln 2b$ is a constant with respect to w. We can see that the maximum a posteriori estimator with a Laplace prior is indeed equivalent to the solution for LASSO regression with $\lambda = \frac{1}{b\beta}$.

2 Part 2: Objective functions for classification

Task 2.1 We are going to follow the same steps as in **Task 1.1**, we have:

$$p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{n=1}^{N} \text{Ber}(y_n|\pi(\boldsymbol{x}_n)) \tag{19}$$

$$= \prod_{n=1}^{N} \pi(\boldsymbol{x}_n)^{y_n}(1 - \pi(\boldsymbol{x}_n))^{1-y_n} \tag{20}$$

We apply the logarithm to the likelihood:

$$\ln p(\boldsymbol{y}|\boldsymbol{f}) = \sum_{n=1}^{N} \ln \pi(\boldsymbol{x}_n)^{y_n} + \sum_{n=1}^{N} \ln(1 - \pi(\boldsymbol{x}_n))^{1-y_n} \tag{21}$$

$$= \sum_{n=1}^{N} y_n \ln \pi(\boldsymbol{x}_n) + (1 - y_n)\ln \pi(1 - \boldsymbol{x}_n) \tag{22}$$

We directly recognize the opposite of the binary cross-entropy, therefore maximizing the likelihood is equivalent to minimizing the binary cross-entropy.

Task 2.2 Now using the Categorical distribution for the likelihood of the data:

$$p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{n=1}^{N} \mathrm{Cat}(y_n|\boldsymbol{\pi}(\boldsymbol{x}_n)) \tag{23}$$

$$= \prod_{n=1}^{N}\prod_{i=1}^{K} \pi_i(\boldsymbol{x}_n)^{y_{n,i}} \tag{24}$$

Let's apply the logarithm to the likelihood of the data:

$$\ln p(\boldsymbol{y}|\boldsymbol{f}) = \sum_{n=1}^{N}\sum_{i=1}^{K} \ln \pi_i(\boldsymbol{x}_n)^{y_{n,i}} \tag{25}$$

$$= \sum_{n=1}^{N}\sum_{i=1}^{K} y_{n,i} \ln \pi_i(\boldsymbol{x}_n) \tag{26}$$

We recognize the opposite of the general cross-entropy which means that maximizing the likelihood of the data is equivalent to minimizing the general cross-entropy.

3 Part 3: Gaussian processes and covariance functions

Task 3.1 Given a Gaussian process (GP), $f_i(x) \sim \mathcal{GP}\left(0,\, k_i\left(x, x'\right)\right)$, the variance $V[f_i(x)]$ at a point x is directly given by the covariance function evaluated at this point, indeed in a Gaussian Process we have $k_i(x, x') = \mathrm{Cov}(f_i(x), f_i(x'))$ and since $\mathbb{V}[X] = \mathrm{Cov}(X, X)$ the variance is simply $\mathrm{Cov}(f_i(x), f_i(x)) = k_i(x, x)$.

For each of the covariance function, we evaluated the covariance at $x = x'$, then we could get:

$$\mathbb{V}[f_1(x)] = 2 \tag{27}$$

$$\mathbb{V}[f_2(x)] = 1 \tag{28}$$

$$\mathbb{V}[f_3(x)] = 4 + x^2 \tag{29}$$

$$\mathbb{V}[f_4(x)] = 1 \tag{30}$$

$$\mathbb{V}[f_5(x)] = 1 + 4x^2 \tag{31}$$

$$\mathbb{V}[f_6(x)] = \frac{1}{5} + x \tag{32}$$

Task 3.2 Covariance functions are said to be stationary if they only depend on the distance between x and x'. In other words, given $d = x - x'$ a stationary kernel satisfies $k(x, x') = k(d)$. Hence, we could recognize only k_1,k_2 and k_4 are stationary functions while the rest are non-stationary (because other covariance functions involve products between variables or taking the minimum between the two values).

Task 3.3 We generated the samples from the corresponding Gaussian process prior for each of six covariance functions (the code is given in the Appendix 4.1).

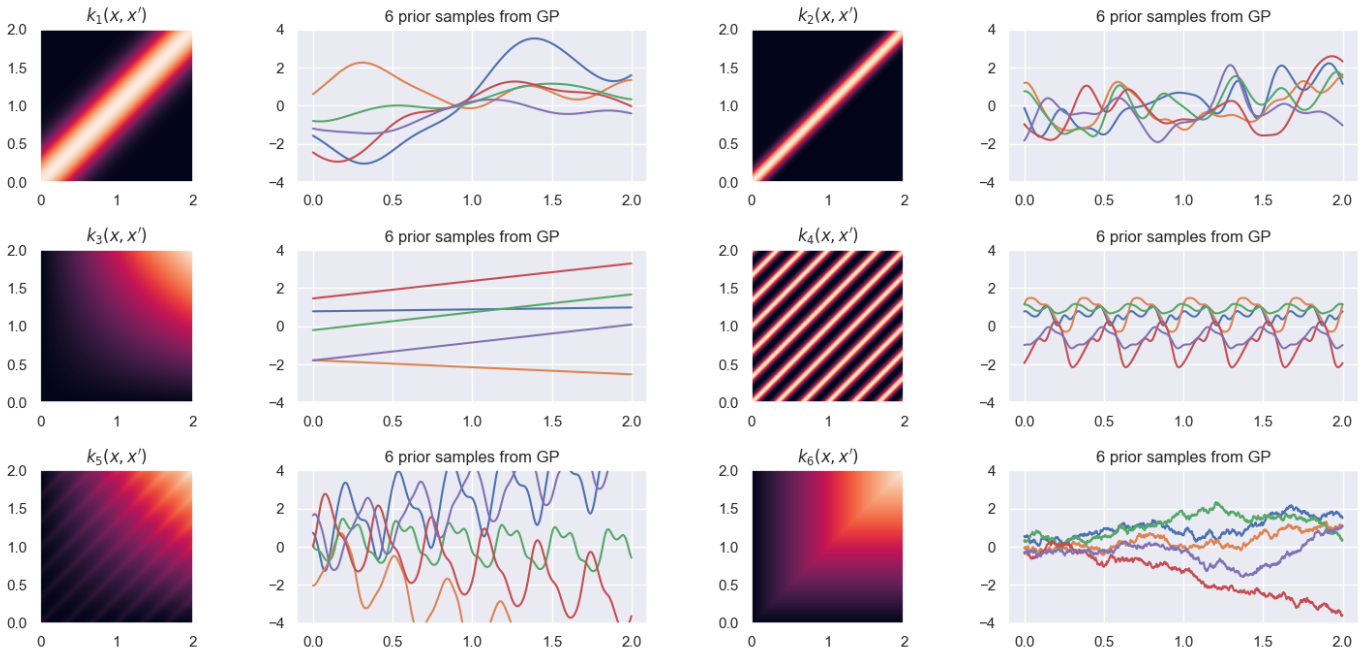

Figure 1: Covariance matrices and samples from the six different covariance functions

Based on 3 (and comparing with the plots in the assignment), we concluded that the k1 matches figure (d), k2 matches figure (a), k3 matches figure (f), k4 matches figure (e), k5 matches figure (c) and k6 matches figure (b) respectively. We can see that stationary covariance functions show no long-term trend, while non-stationary functions do.

Task 3.4 The covariance function given is a kernel in two dimensions $k(\mathbf{x}, \mathbf{x}') = (\kappa^2 + \lambda^2 \mathbf{x}^T \mathbf{x}')^2$,where $\mathbf{x} \in \mathbb{R}^2$. We could expand this expression to determine the equivalent feature expansion $\phi(x)$: $(\kappa^2 + \lambda^2 \mathbf{x}^T \mathbf{x}')^2 = \kappa^4 + 2\kappa^2 \lambda^2 \mathbf{x}^T \mathbf{x}' + \lambda^4 (\mathbf{x}^T \mathbf{x}')^2$. Assuming $x = [x_1, x_2]^T$ and $x' = [x_1', x_2']^T$, then we got $\mathbf{x}^T \mathbf{x}' = x_1 x_1' + x_2 x_2'$. We can further expand this $(\mathbf{x}^T \mathbf{x}')^2$ expression: $(\mathbf{x}^T \mathbf{x}')^2 = (x_1 x_1' + x_2 x_2')^2 = x_1^2 x_1'^2 + 2 x_1 x_1' x_2 x_2' + x_2^2 x_2'^2$. Then substitute this into the kernel function $k(x, x')$, we can get:

$$k(\mathbf{x}, \mathbf{x}') = \kappa^4 + 2\kappa^2 \lambda^2 \mathbf{x}^T \mathbf{x}' + \lambda^4 (\mathbf{x}^T \mathbf{x}')^2 \tag{33}$$

$$= \kappa^4 + 2\kappa^2 \lambda^2 (x_1 x_1' + x_2 x_2') + \lambda^4 (x_1^2 x_1'^2 + 2 x_1 x_1' x_2 x_2' + x_2^2 x_2'^2) \tag{34}$$

Now we want to express the function $\phi(\mathbf{x})$ such that $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$. So we need to define $\phi(\mathbf{x})$ such that each term in the above expansion is a product term from $\phi(\mathbf{x})^T \phi(\mathbf{x}')$. Thus, we could find the possible $\phi(\mathbf{x}) = [\kappa^2, \sqrt{2}\kappa\lambda x_1, \sqrt{2}\kappa\lambda x_2, \lambda^2 x_1^2, \sqrt{2}\lambda^2 x_1 x_2, \lambda^2 x_2^2]$ and $\phi(\mathbf{x}') = [\kappa^2, \sqrt{2}\kappa\lambda x_1', \sqrt{2}\kappa\lambda x_2', \lambda^2 x_1'^2, \sqrt{2}\lambda^2 x_1' x_2', \lambda^2 x_2'^2]$.

Task 3.5 We are working on a regression problem where we want to model the mean of the data, ie. $y_n = f(x_n) + b + \epsilon_n$, our assumptions about the priors give:

$$p(\mathbf{y}, \mathbf{f}, b) = p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|b)p(b) \tag{35}$$
$$= \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2)\mathcal{N}(\mathbf{f}|b \cdot \mathbf{1}, \mathbf{K})\mathcal{N}(b|0, \tau^2) \tag{36}$$

We want to compute the marginal prior distribution of $\mathbf{f}$, therefore we have to marginalize over b (not over $\mathbf{y}$ because $\mathbf{f}$ does not depend on it). We have:

$$p(\mathbf{f}) = \int p(\mathbf{f}|b)p(b)db \tag{37}$$
$$= \int \mathcal{N}(\mathbf{f}|b \cdot \mathbf{1}, \mathbf{K})\mathcal{N}(b|0, \tau^2)db \tag{38}$$

Using 3.38 from Murphy1 we get:

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K} + \tau^2\mathbf{J}_N) \tag{39}$$

Where $\mathbf{J}_N \in \mathbb{R}^{N \times N}$ is the matrix of ones. We can derive $p(\mathbf{y})$ in the same fashion, by marginalizing over $\mathbf{f}$:

$$p(\mathbf{y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})d\mathbf{f} \tag{40}$$
$$= \int \mathcal{N}(\mathbf{y}|\boldsymbol{I} \cdot \mathbf{f}, \sigma^2\boldsymbol{I})\mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K} + \tau^2\mathbf{J}_N)d\mathbf{f} \tag{41}$$

Using 3.38 in Murphy1 once again we get:

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \sigma^2\boldsymbol{I} + \mathbf{K} + \tau^2\mathbf{J}_N) \tag{42}$$

If we want to find the equivalent kernel $k_{equivalent}$, we need to add some terms to represent $\sigma^2\boldsymbol{I}$ (diagonal terms) and $\tau^2\mathbf{J}_N$ (constant). We get:

$$k_{equivalent}(x, x') = k(x, x') + \mathbb{1}_{x=x'}\sigma^2 + \tau^2 \tag{43}$$

Where $\mathbb{1}_{x=x'}$ is the indicator function for $d = x - x' = 0$. We can see the respective roles of σ^2 and τ^2. The former one is the variance inherent to the data (noise) which acts as the limit of a squared exponential kernel when the temperature goes to 0. The latter one is related to the intercept b. Since we use the same intercept for all the observations, this noise is present for any observation wether it's close to the data or not.

Task 3.6 We want to plOt the prior predictive distribution of our model, we use the marginal distribution $p(\mathbf{f})$ that we found before:

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K} + \tau^2 \mathbf{J}_N) \tag{44}$$

Where $\mathbf{K}$ is the kernel for the squared exponential covariance function. We therefore implement the new covariance function as follow:

```
def kernel_f(dist, kappa, lengthscale, tau):
    return kappa ** 2 * np.exp(-0.5 * dist ** 2 / lengthscale ** 2) +
   tau ** 2
```

Using this covariance function, we get the following prior predictive distribution for the bike dataset (on $[0, 370]$ as in the exercice 5) with 30 prior samples (the detailed code is given in 4.2).

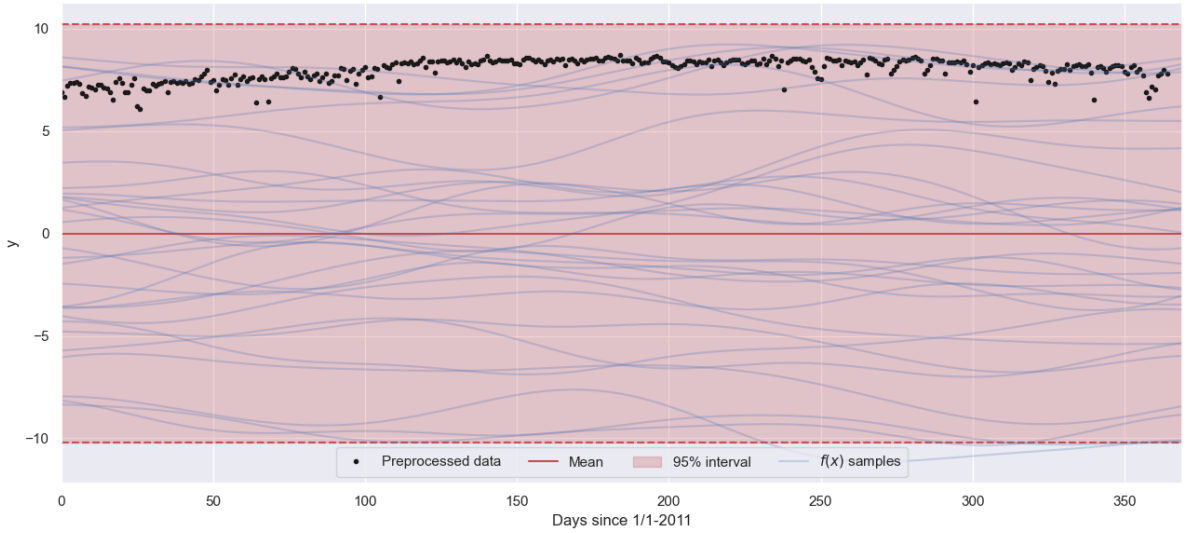

Figure 2: Prior predictive distribution for the bike dataset when we explicitly model the intercept (and 30 prior samples in blue), we used $\tau = 5$, $\ell = 50$, $\sigma = 0.1$ and $\kappa = 1$

Task 3.7 Using the marginal likelihood $p(\mathbf{y})$ and the code from exercise 5, we can use the data to find the best set of hyperparameters (τ, ℓ, σ and κ) that best explain the observed data. That means we minimize the negative marginal likelihood of the data using autograd. We get the following set of hyperparameters.

Hyperparameter	τ	ℓ	σ	κ
Estimated value	7.659	82.671	0.299	0.632

Table 1: Resulting hyperparameters when optimizing the marginal likelihood $p(\mathbf{y})$

The detailed code for this task and the following one is given in 4.3.

Task 3.8 Using the set of hyperparameters we found above, we can use the bike dataset to plot the posterior predictive distribution of our Gaussian Process:

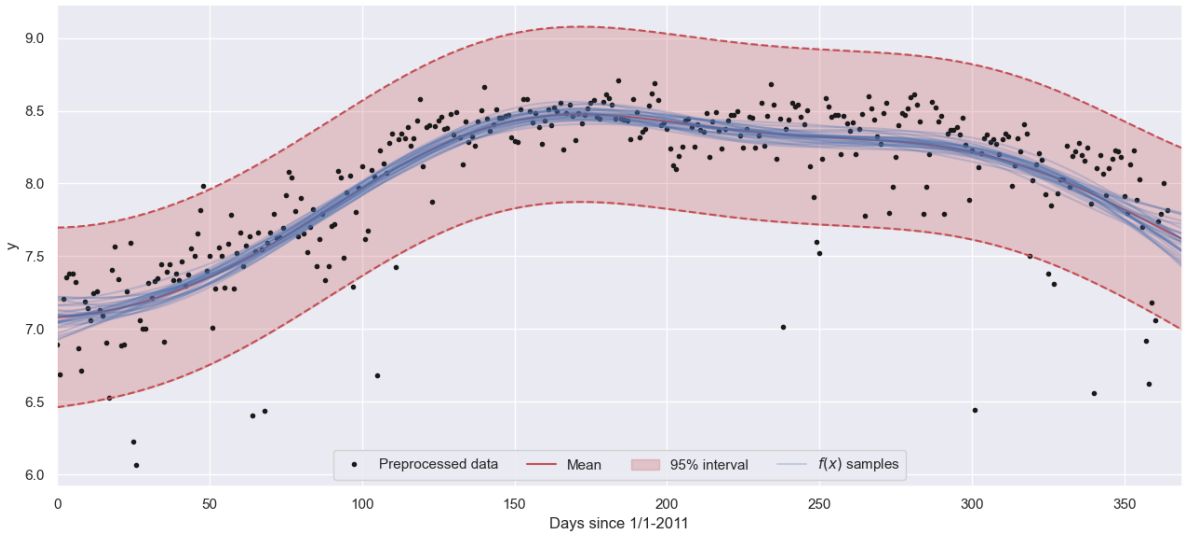

Figure 3: Posterior predictive distribution for the bike dataset when we explicitly model the intercept (and 30 posterior samples in blue) using the set of hyperparameters that maximizes the marginal log likelihood of the data (**Task 3.7**)

We can see that modelling the intercept b explicitly yields similar results that the ones we got in exercise 5, but this time we didn't have to standardize the data, therefore we can directly read the y value on the axis.

Task 3.9 We suppose we have a multi-class classification problem with 4 classes, with a dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$. In this case we have discrete and unordered targets, the usual distribution adopted is therefore the categorical distribution:

$$y_n|\mathbf{f}_{\cdot,n} \sim \text{Categorical}[\text{softmax}(\mathbf{f}_{\cdot,n})] \tag{45}$$

Where $\mathbf{f}_{\cdot,n} \in \mathbb{R}^4$ is the vector of logits (we use softmax to get the probabilities of each class) for the 4 different classes for observation n. Now we don't want to model the latent functions as linear models but rather as Gaussian processes, ie $\mathbf{f}_i \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_i)$ where $\mathbf{K}_i \in \mathbb{R}^{N\times N}$ and $\mathbf{f}_i$ is the latent function for class i (thus $\mathbf{f}_i \in \mathbb{R}^N$). We have one Gaussian Process for each latent function, therefore we can right the joint distribution as follow (as we assume each latent function and each observation to be independent):

$$p(\mathbf{y}, \mathbf{f}) = \prod_{n=1}^{N} p(y_n|\mathbf{f}) \prod_{i=1}^{4} p(\mathbf{f}_i) \tag{46}$$

$$= \prod_{n=1}^{N} \text{Categorical}[\text{softmax}(y_n|\mathbf{f}_{\cdot,n})] \prod_{i=1}^{4} \mathcal{N}(\mathbf{0}, \mathbf{K}_i) \tag{47}$$

Note that in this case $\mathbf{f} \in \mathbb{R}^{4\times N}$

4 Appendix

4.1 Task 3.3

```python
import numpy as np
import seaborn as snb
import matplotlib.pyplot as plt
snb.set_style('darkgrid')
snb.set_theme(font_scale=1)

# Defining the different covariance functions
kernel_1 = lambda x,y: 2 * np.exp(- (x - y) ** 2 / (2 * 0.3 ** 2))
kernel_2 = lambda x,y: np.exp(- (x - y) ** 2 / (2 * 0.1 ** 2))
kernel_3 = lambda x,y: 4 + 2 * x * y
kernel_4 = lambda x,y: np.exp(-2 * np.sin(3 * np.pi * np.abs(x - y)) ** 2)
kernel_5 = lambda x,y: np.exp(-2 * np.sin(3 * np.pi * np.abs(x - y)) ** 2) + 4 * x * y
kernel_6 = lambda x,y: 0.2 + np.minimum(x,y)

# Generate samples from kernel
def generate_samples(K, jitter = 1e-8, num_samples = 20):
    L = np.linalg.cholesky(K + jitter * np.identity(K.shape[0]))
    f_samples_or = np.random.multivariate_normal(np.zeros(K.shape[0]), np.identity(K.shape[0]), num_samples)
    f_samples = L @ f_samples_or.T
    return f_samples

# Plot of the different kernels
X = np.linspace(0, 2, 1000)
xs, ys = np.meshgrid(X,X)
fig, ax = plt.subplots(3,4)
fig.set_figwidth(15)
fig.set_figheight(7)
kernels = [kernel_1, kernel_2, kernel_3, kernel_4, kernel_5, kernel_6]
for idx, ker in enumerate(kernels):
    row, col = idx // 2,    2 * (idx % 2)
    ax[row,col].grid(False)
    ax[row,col].pcolormesh(xs, ys, ker(xs, ys))
    ax[row,col].set_aspect('equal')
    ax[row,col].set_title(fr"$k_{idx+1}(x,x')$")
    ax[row,col+1].plot(X, generate_samples(ker(xs, ys), num_samples=5))
    ax[row,col+1].set_ylim(-4,4)
    ax[row,col+1].set_title("6 prior samples from GP")
plt.tight_layout()
plt.show()
```

Listing 1: Plotting the different covariance functions and some samples

4.2 Task 3.6

```python
# Loading the bike dataset withtout standardizing
np.load('data_exercise5b.npz')
day = data['day']
bike_count = np.log(data['bike_count'])
```

```
5
6  # Defining the covariance function with tau
7  def kernel_f(dist, kappa, lengthscale, tau):
8      return kappa**2*np.exp(-0.5*dist**2/lengthscale**2) + tau ** 2
9
10 # Generating the samples with K and m
11 def generate_samples(m, K, num_samples, jitter=0):
12     N = len(K)
13     L = np.linalg.cholesky(K + jitter*np.identity(N))
14     zs = np.random.normal(0, 1, size=(len(K), num_samples))
15     f_samples = m[:, None] + np.dot(L, zs)
16     return f_samples
17
18 # How to construct kernel with tau and use it in GPs
19 class StationaryIsotropicKernel(object):
20     def __init__(self, kernel_fun, kappa=1., lengthscale=1., tau=1.):
21         self.kernel_fun = kernel_fun
22         self.kappa = kappa
23         self.lengthscale = lengthscale
24         self.tau = tau
25
26     def contruct_kernel(self, X1, X2, kappa=None, lengthscale=None, tau=
    None, jitter=1e-8):
27         # extract dimensions
28         N, M = X1.shape[0], X2.shape[0]
29         # prep hyperparameters
30         kappa = self.kappa if kappa is None else kappa
31         lengthscale = self.lengthscale if lengthscale is None else
    lengthscale
32         tau = self.tau if tau is None else tau
33         # compute all the pairwise distances efficiently
34         dists = np.sqrt(np.sum((np.expand_dims(X1, 1) - np.expand_dims(
    X2, 0))**2, axis=-1))
35         # squared exponential covariance function
36         K = self.kernel_fun(dists, kappa, lengthscale, tau)
37         # add jitter to diagonal for numerical stability
38         if len(X1) == len(X2) and np.allclose(X1, X2):
39             K = K + jitter*np.identity(len(X1))
40         return K
41
42 # Creating our kernel
43 kernel = StationaryIsotropicKernel(kernel_f)
44
45 # GPs implementation for bike dataset using tau as a hyperparameter
46 class GaussianProcessRegression(object):
47     def __init__(self, X, y, kernel, kappa=1., lengthscale=50., tau=5,
    sigma=.1, jitter=1e-8):
48         self.X = X
49         self.y = y
50         self.N = len(X)
51         self.kernel = kernel
52         self.jitter = jitter
53         self.set_hyperparameters(kappa, lengthscale, sigma, tau)
54
55     def set_hyperparameters(self, kappa, lengthscale, sigma, tau):
56         self.kappa = kappa
57         self.lengthscale = lengthscale
```

```
58          self.sigma = sigma
59          self.tau = tau
60
61      def posterior_samples(self, Xstar, num_samples):
62          mu, Sigma = self.predict_f(Xstar)
63          f_samples = generate_samples(mu.ravel(), Sigma, num_samples)
64          return f_samples
65
66      def predict_y(self, Xstar):
67          # prepare relevant matrices
68          mu, Sigma = self.predict_f(Xstar)
69          Sigma = Sigma + self.sigma**2 * np.identity(len(mu))
70          return mu, Sigma
71
72      def predict_f(self, Xstar):
73          k = self.kernel.contruct_kernel(Xstar, self.X, self.kappa, self.
    lengthscale, self.tau, jitter=self.jitter)
74          K = self.kernel.contruct_kernel(self.X, self.X, self.kappa, self
    .lengthscale, self.tau, jitter=self.jitter)
75          Kstar = self.kernel.contruct_kernel(Xstar, Xstar, self.kappa,
    self.lengthscale, self.tau, jitter=self.jitter)
76          # Compute C matrix
77          C = K + self.sigma**2*np.identity(len(self.X))
78          # computer mean and Sigma
79          mu = np.dot(k, np.linalg.solve(C, self.y))
80          Sigma = Kstar - np.dot(k, np.linalg.solve(C, k.T))
81          return mu, Sigma
82
83      def log_marginal_likelihood(self, kappa, lengthscale, sigma, tau):
84          # prepare kernels
85          K = self.kernel.contruct_kernel(self.X, self.X, kappa,
    lengthscale, tau)
86          C = K + sigma**2*np.identity(self.N)
87          # compute Cholesky decomposition
88          L = np.linalg.cholesky(C)
89          v = np.linalg.solve(L, self.y)
90          # compute log marginal likelihood
91          logdet_term = np.sum(np.log(np.diag(L)))
92          quad_term =  0.5*np.sum(v**2)
93          const_term = -0.5*self.N*np.log(2*np.pi)
94          return const_term - logdet_term - quad_term
95
96  # Plotting the distribution with uncertainty
97  def plot_with_uncertainty(ax, Xp, gp, color='r', color_samples='b',
    num_samples=0):
98      mu, Sigma = gp.predict_y(Xp)
99      mean = mu.ravel()
100     std = np.sqrt(np.diag(Sigma))
101     # plot distribution
102     ax.plot(Xp, mean, color=color, label='Mean')
103     ax.plot(Xp, mean + 2*std, color=color, linestyle='--')
104     ax.plot(Xp, mean - 2*std, color=color, linestyle='--')
105     ax.fill_between(Xp.ravel(), mean - 2*std, mean + 2*std, color=color,
        alpha=0.25, label='95% interval')
106     # generate samples
107     if num_samples > 0:
108         fs = gp.posterior_samples(Xp, num_samples)
```

```
109         ax.plot(Xp, fs[:,0], color=color_samples, alpha=.25, label="$f(x)$ samples")
110         ax.plot(Xp, fs[:, 1:], color=color_samples, alpha=.25)
111
112 # Creating our Gaussian Process without any data (prior)
113 gp_prior = GaussianProcessRegression(np.zeros((0, 1)), np.zeros((0, 1)), kernel)
114
115 # Generating the plot on [0,370]
116 Xstar = np.arange(0, 370)[:, None]
117 fig, ax = plt.subplots(1, 1, figsize=(13, 6))
118 ax.plot(day, bike_count, 'k.', label='Preprocessed data')
119 plot_with_uncertainty(ax, Xstar, gp_prior, num_samples=30)
120 ax.legend(loc='lower center', ncol=4)
121 ax.set(xlabel='Days since 1/1-2011', ylabel='y')
122 ax.set_xlim(Xstar.min(), Xstar.max())
123 plt.tight_layout()
124 plt.show()
```

Listing 2: Generating the prior predictive distribution plot for the bike dataset

4.3 Task 3.7 and Task 3.8

```
1  import autograd.numpy as np
2  from scipy.optimize import minimize
3  from autograd import value_and_grad
4  # How to optimize with an additional hyperparameter
5  def optimize_hyperparameters(gp, theta_init):
6      # define optimization objective as the negative log marginal likelihood
7      objective = lambda params: -gp.log_marginal_likelihood(np.exp(params[0]), np.exp(params[1]), np.exp(params[2]), np.exp(params[3]))
8      # optimize using gradients
9      res = minimize(value_and_grad(objective), np.log(theta_init), jac=True)
10     # check for success
11     if not res.success:
12         print('Warning: optimization failed!')
13     # return resultss
14     theta = np.exp(res.x)
15     return theta
16
17
18 # Creating our Gaussian Process ont the bike dataset
19 gp = GaussianProcessRegression(day, bike_count, kernel)
20
21 # Finding the optimized set of hyperparameters
22 kappa_hat, scale_hat, sigma_hat, tau_hat = optimize_hyperparameters(gp, theta_init=np.array([1,1,1,1]))
23 gp.set_hyperparameters(kappa_hat, scale_hat, sigma_hat, tau_hat)
24
25 # Generating the posterior predictive plot on [0,370]
26 Xstar = np.arange(0, 370)[:, None]
27 fig, ax = plt.subplots(1, 1, figsize=(13, 6))
28 ax.plot(day, bike_count, 'k.', label='Preprocessed data')
29 plot_with_uncertainty(ax, Xstar, gp, num_samples=30)
```

```
30 ax.legend(loc='lower center', ncol=4)
31 ax.set(xlabel='Days since 1/1-2011', ylabel='y')
32 ax.set_xlim(Xstar.min(), Xstar.max())
33 plt.tight_layout()
34 plt.show()
```

Listing 3: Optimizing the hyperparameters and generating the posterior predictive distribution plot for the bike dataset