Advanced Machine Learning - Lab 04

G. Hari Prasath (hargo729), Julius Kittler (julki029), Hector Plata (hecpl268) and Maximilian Pfundstein (maxpf364)

2019-10-08

Contents

1 Implementing GP Regression

This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \epsilon \text{ with } \epsilon \sim \mathcal{N}(0, \sigma_n^2) \text{ and } f \sim \mathcal{GP}(0, k(x, x'))$$

You must implement Algorithm 2.1 on page 19 of Rasmussen and Willams' book. The algorithm uses the Cholesky decomposition (`chol` in R) to attain numerical stability. Note that L in the algorithm is a lower triangular matrix, whereas the R function returns an upper triangular matrix. So, you need to transpose the output of the R function. In the algorithm, the notation $\mathbf{A}/\mathbf{b}$ means the vector $\mathbf{x}$ that solves the equation $\mathbf{Ax} = \mathbf{b}$ (see p. xvii in the book). This is implemented in R with the help of the function `solve`.

1.1 Simulating from the Posterior

Task: Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it `posteriorGP`) should return a vector with the posterior mean and variance of f, both evaluated at a set of x-values (X_*). You can assume that the prior mean of f is zero for all x. The function should have the following inputs:

- **X:** Vector of training inputs.
- **Y:** Vector of training targets/outputs.
- **XStar:** Vector of inputs where the posterior distribution is evaluated, i.e. X_*.

- **hyperParam:** Vector with two elements σ_f and ℓ.
- **sigmaNoise:** Noise standard deviation σ_n.

Hint: Write a separate function for the kernel (see the file `GaussianProcess.R` on the course web page).

```
#' se_kernel
#'
#' @param x Point X.
#' @param y Point Y.
#' @param d Parameter sigma_f.
#' @param ell Parameter ell.
#'
#' @return Returns the squared exponential kernel function applied to X and Y.
se_kernel = function(x, y, d=1, ell=0.3) {
  return(d^2 * exp(-0.5 * (((x-y)/ell)^2)))
}

#' cov_kernel
#'
#' @param X
#' @param kernel A kernel functions.
#' @param ... Parameters for the kernel function.
#'
#' @return Returns the covariance matrix where the kernel is being applied.
cov_kernel = function(X, Y, kernel, ...) {

  if (!is.vector(X) || !is.vector(Y))
    stop("X and Y must be vectors.")

  K = matrix(NA, nrow = length(X), ncol = length(Y))

  for (i in 1:length(X)) {
    for (j in 1:length(Y)) {
      K[i, j] = kernel(X[i], Y[j], ...)
    }
  }

  return(K)
}

#' posteriorGP
#'
#' @param X Vector of training inputs.
#' @param Y Vector of training targets/outputs.
#' @param XStar Vector of inputs where the posterior distribution is evaluated,
#'    i.e XStar.
#' @param sigmaNoise Noise standard deviation sigma_n.
#' @param kernel A kernel function.
#' @param ... Parameters for the kernel function.
#'
#' @return Vector with the posterior mean and variance of f evaluated on set XStar
posteriorGP = function(X, Y, XStar, sigmaNoise, kernel, ...) {

  K = t(cov_kernel(X, X, kernel, ...))
  L = t(chol(K + diag(sigmaNoise^2, length(X))))
```

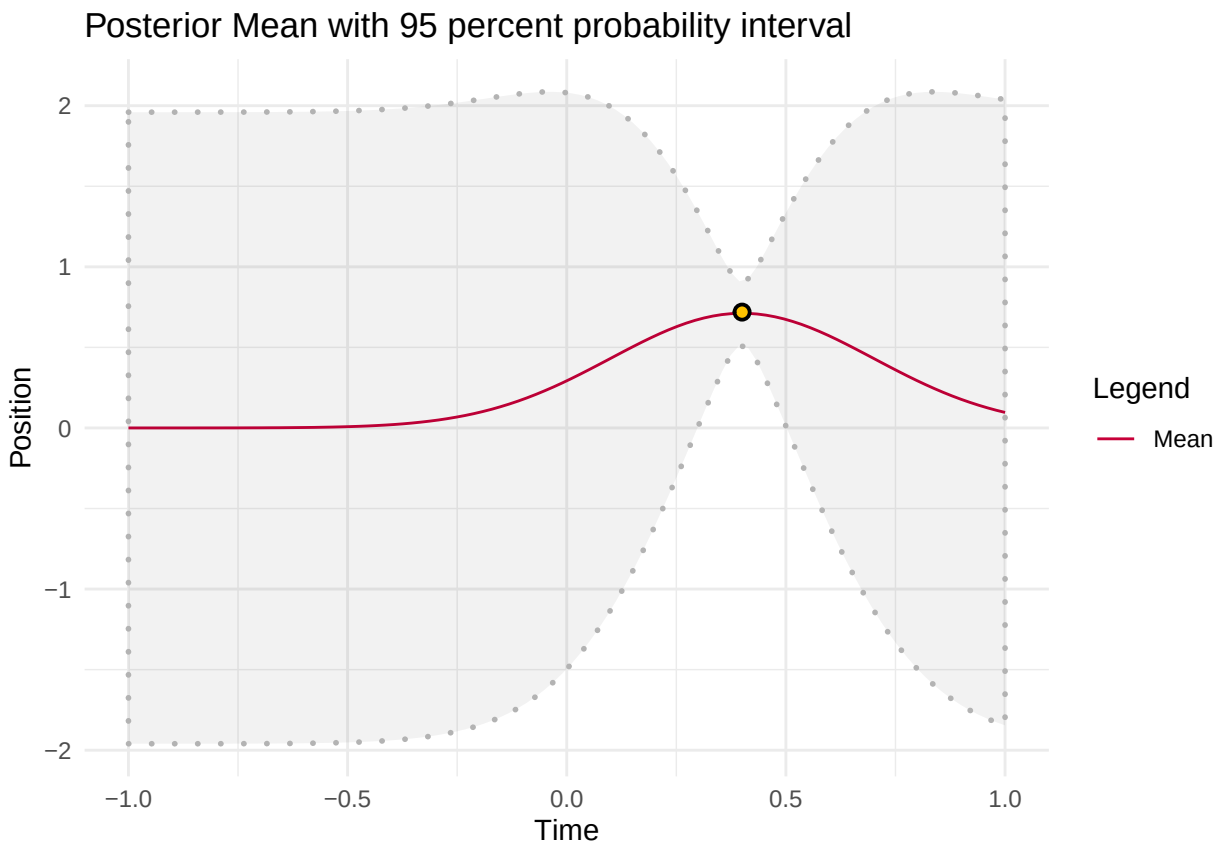
```
  # Predictive Mean
  alpha = solve(t(L), solve(L, Y))
  K_star = cov_kernel(X, XStar, kernel, ...)
  f_star = t(K_star) %*% alpha

  # Predictive Variance
  V = solve(L, K_star)
  V_f_star = cov_kernel(XStar, XStar, kernel, ...) - t(V) %*% V

  # Log Marginal Likelihood
  n = length(X)

  lml = -0.5 %*% t(Y) %*% alpha - sum(log(diag(L))) - (n/2) * log(2*pi)

  gp_object = list(mean=f_star,
                   variance=V_f_star,
                   lml = lml,
                   X=X, Y=Y,
                   XStar=XStar)

  class(gp_object) = "gp"

  return(gp_object)
}

#' plot.gp
#'
#' @param gp An Gaussian Process ("gp") object.
#' @param direct_plot TRUE by default. Determines if to plot directly or to
#'    return the plot.
#'
#' @return If direct_plot is set to TRUE, the plot object.
plot.gp = function(gp, direct_plot=TRUE) {
  df = data.frame(x = grid,
                  y = gp$mean,
                  y_upper = gp$mean + 1.96*sqrt(diag(gp$variance)),
                  y_lower = gp$mean - 1.96*sqrt(diag(gp$variance)))

  points = data.frame(x = gp$X, y = gp$Y)

  p = ggplot() +
  geom_line(aes(x = df$x, y = df$y, colour = "Mean")) +
  geom_ribbon(aes(x = df$x, ymin = df$y_lower, ymax = df$y_upper),
                  alpha=0.05,
                  linetype=3,
                  colour="grey70",
                  size=1,
                  fill="black") +
  geom_point(aes(x = points$x, y = points$y), color = "black",
          fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
  labs(title = "Posterior Mean with 95 percent probability interval",
       y = "Position", x = "Time", color = "Legend") +
  scale_color_manual(values = c("#C70039", "#FF5733", "#581845")) +
```

```
  theme_minimal()

  if (!direct_plot)
    return(p)

  print(p)
}
```

1.2 Prior Observation

Task: Now, let the prior hyperparameters be $\sigma_f = 1$ and $\ell = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that $\sigma_n = 0.1$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 percent probability (pointwise) bands for f.

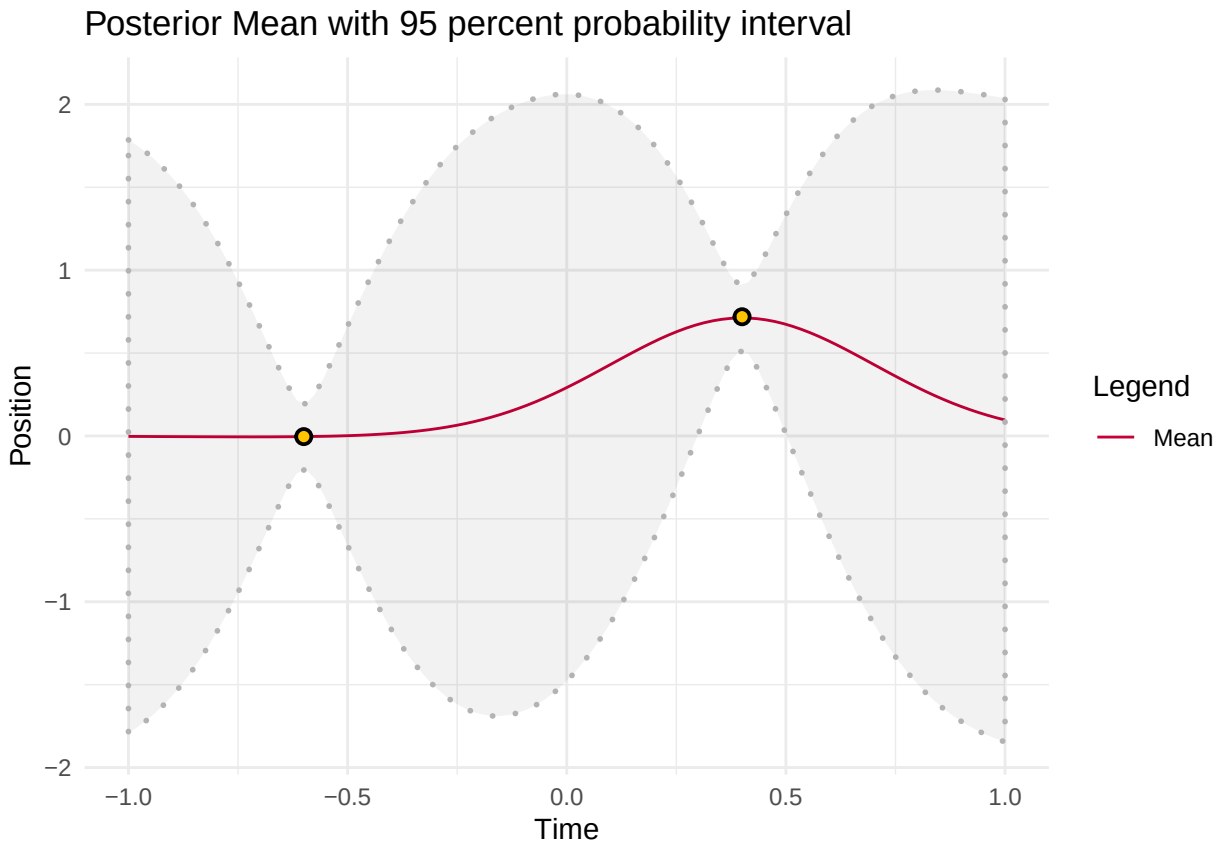
```
sigma_f = 1
ell = 0.3
sigma_n = 0.1

x = 0.4
y = 0.719

grid = seq(from=-1, to=1, length.out = 100)

res = posteriorGP(x, y, grid, sigma_n, se_kernel, d=sigma_f, ell=ell)
```



1.3 Another Observation

Task: Update your posterior from (2) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 percent probability (pointwise) bands for f.

```
x = c(0.4, -0.6)
y = c(0.719, -0.004)
res = posteriorGP(x, y, grid, sigma_n, se_kernel, d=sigma_f, ell=ell)
```



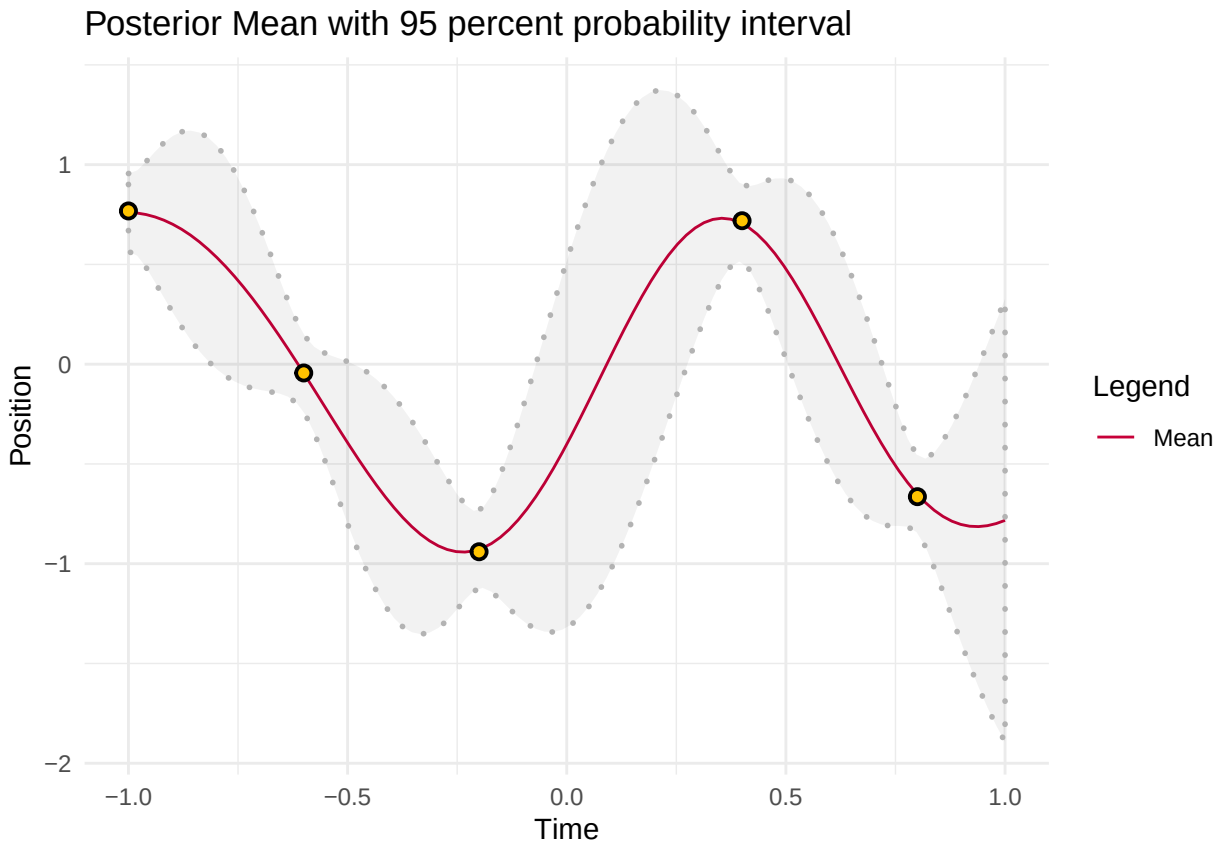
Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations.

1.4 Five Observations

Task: Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 percent probability (pointwise) bands for f.

```
##      [,1]   [,2]  [,3]  [,4]   [,5]
## x -1.000 -0.600 -0.20 0.400  0.800
## y  0.768 -0.044 -0.94 0.719 -0.664
```

```
res = posteriorGP(input[1,], input[2,], grid, sigma_n, se_kernel, d=sigma_f,
                  ell=ell)
```



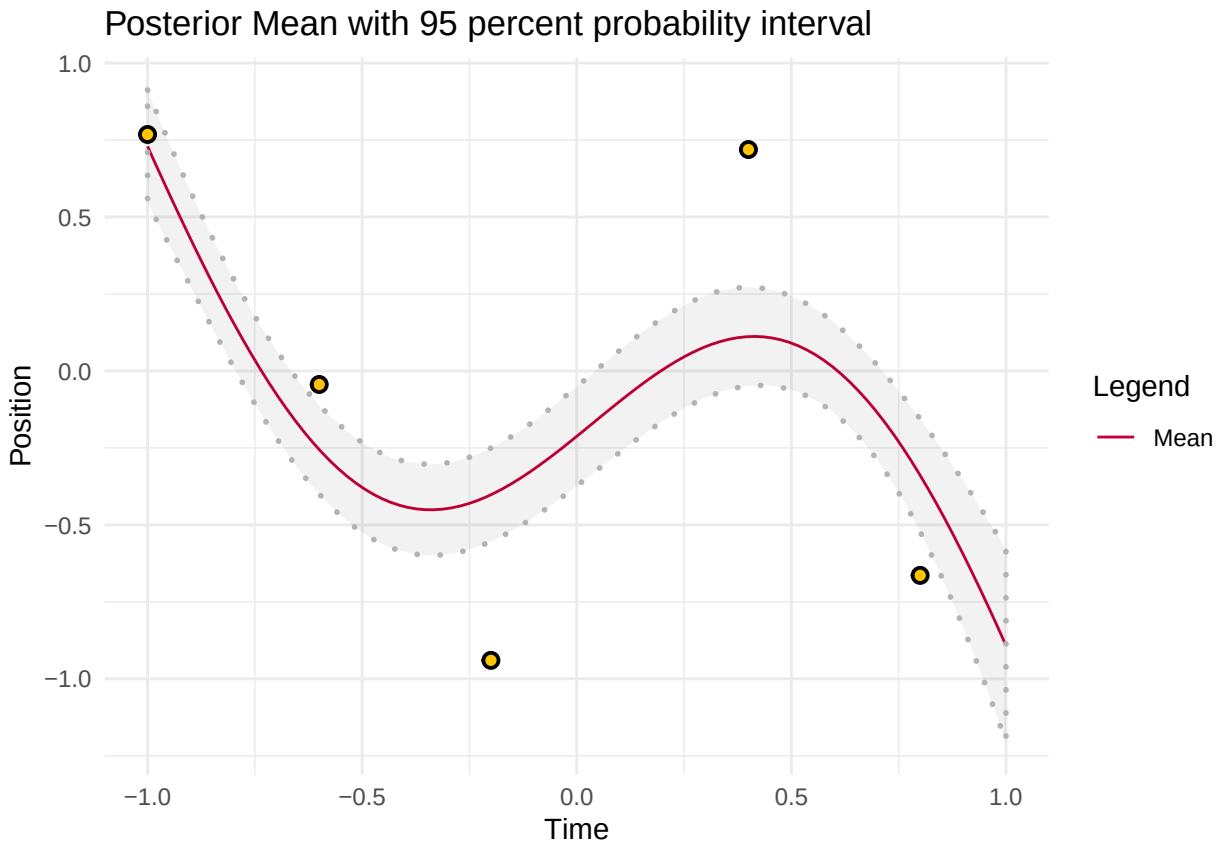
1.5 Different Hypterparameters

Task: Repeat (4), this time with hyperparameters $\sigma_f = 1$ and $\ell = 1$. Compare the results.

Answer: Increasing ℓ results is a smoother posterior which derives from the kernel function as ℓ states which points are considered close and which are not. This enables the function to pass the training points with a larger distance and thus become smoother. We basically want to tune ℓ to obtain a model that captures the trend of the data; we can utilise it to prevent possible overfitting. Thereof the parameter ℓ is a tradeoff between smoothness and overfit.

Also note that the variance and thus the probability interval for the posterior has been reduced, even thought the parameter σ_n has not been touched. This rises from the fact that a smoothed function cannot vary that muched compared to its bumpy counterpart.

```
sigma_f = 1
ell = 1

res = posteriorGP(x, y, grid, sigma_n, se_kernel, d=sigma_f, ell=ell)
```



2 $\mathcal{GP}$ Regression with `kernlab`

Task: In this exercise, you will work with the daily mean tem- perature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. We have removed the leap year day February 29, 2012 to make things simpler. You can read the dataset with the command:

```
read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/\
        Code/TempTullinge.csv", header=TRUE, sep=";")
```

Create the variable time which records the day number since the start of the dataset (i.e., `time` $= 1, 2, ..., 365 \times 6 = 2190$). Also, create the variable `day` that records the day number since the start of each year (i.e., `day` $= 1, 2, ..., 365, 1, 2, ..., 365$). Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the data and use only every fifth observation. This means that your time and day variables are now `time` $= 1, 6, 11, ..., 2186$ and `day` $= 1, 6, 11, ..., 361, 1, 6, 11, ..., 361$.

2.1 Computing the Covariance Matrix

Task: Familiarize yourself with the functions `gausspr` and `kernelMatrix` in `kernlab`. Do `?gausspr` and read the input arguments and the output. Also, go through the file `KernLabDemo.R` available on the course website. You will need to understand it. Now, define your own square exponential kernel function (with parameters ℓ (`ell`) and σ_f (`sigmaf`)), evaluate it in the point $x = 1$, $x' = 2$ and and use the `kernelMatrix` function to compute the covariance matrix $K(X, X_*)$ for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

2.2 Estimate the above $\mathcal{GP}$ regression model

Task: Consider first the following model:

$$\texttt{time} = f(\texttt{time}) + \epsilon \text{ with } \epsilon \sim \mathcal{N}(0, \sigma_n^2) \text{ and } f \sim \mathcal{GP}(0, k(\texttt{time}, \texttt{time}'))$$

Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the `lm` function in R). Estimate the above Gaussian process regression model using the squared exponential function from (1) with $\sigma_f = 20$ and $\ell = 0.2$. Use the predict function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use `type="l"` in the plot function). Play around with different values on σ_f and ℓ (no need to write this in the report though).

2.3 Posterior Variance

Task: `kernlab` can compute the posterior variance of f, but it seems to be a bug in the code. So, do your own computations for the posterior variance of f and plot the 95 percent probability (pointwise) bands for f. Superimpose these bands on the figure with the posterior mean that you obtained in (2).

Hint: Note that Algorithm 2.1 on page 19 of Rasmussen and Willams' book already does the calculations required. Note also that `kernlab` scales the data by default to have zero mean and standard deviation one. So, the output of your implementation of Algorithm 2.1 will not coincide with the output of `kernlab` unless you scale the data first. For this, you may want to use the R function scale.

2.4 Another Model

Task: Consider first the following model:

$$\texttt{time} = f(\texttt{temp}) + \epsilon \text{ with } \epsilon \sim \mathcal{N}(0, \sigma_n^2) \text{ and } f \sim \mathcal{GP}(0, k(\texttt{day}, \texttt{day}'))$$

Estimate the model using the squared exponential function with $\sigma_f = 20$ and $\ell = 0.2$. Superimpose the posterior mean from this model on the posterior mean from the model in (2). Note that this plot should also have the time variable on the horizontal axis. Compare the results of both models. What are the pros and cons of each model?

2.5 Periodic Kernel Implementation

Task: Finally, implement a generalization of the periodic kernel given in the lectures:

$$k(x, x') = \sigma_f^2 \exp\left\{ \frac{2\sin(\pi|x - x'|/d)}{\ell_1^2} \right\} \exp\left\{ -\frac{1}{2} \frac{|x - x'|^2}{\ell_2^2} \right\}$$

Note that we have two different length scales here, and l2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20$, $\ell_1 = 1$, $\ell_2 = 10$ and $d = 365/\texttt{sd}(\texttt{time})$. The reason for the rather strange period here is that `kernlab` standardizes the in- puts to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20$ and $\ell = 0.2$. Discuss the results.

3 $\mathcal{GP}$ Classification with `kernlab`

Download the banknote fraud data:

```
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/\
                GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")
names(data) = c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] = as.factor(data[,5])
```

You can read about this dataset here. Choose 1000 observations as training data using the following command (i.e., use the vector `SelectTraining` to subset the training observations):

```
set.seed(111);
SelectTraining = sample(1:dim(data)[1], size = 1000, replace = FALSE)
```

3.1 Using `kernlab` for Training a Model (Limited Covariates)

Task: Use the R package `kernlab` to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates `varWave` and `skewWave` in the model. Plot contours of the prediction probabilities over a suitable grid of values for `varWave` and `skewWave`. Overlay the training data for fraud = 1 (as blue points) and fraud = 0 (as red points). You can reuse code from the file `KernLabDemo.R` available on the course website. Compute the confusion matrix for the classifier and its accuracy.

3.2 Predicting on the Test Set

Task: Using the estimated model from (1), make predictions for the test set. Compute the accuracy.

3.3 Using all Covariates

Task: Train a model using all four covariates. Make predictions on the test set and compare the accuracy to the model with only two covariates.

```
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
set.seed(12345)

#' se_kernel
#'
#' @param x Point X.
#' @param y Point Y.
#' @param d Parameter sigma_f.
#' @param ell Parameter ell.
#'
#' @return Returns the squared exponential kernel function applied to X and Y.
se_kernel = function(x, y, d=1, ell=0.3) {
  return(d^2 * exp(-0.5 * (((x-y)/ell)^2)))
}

#' cov_kernel
#'
#' @param X
```

```r
#' @param kernel A kernel functions.
#' @param ... Parameters for the kernel function.
#'
#' @return Returns the covariance matrix where the kernel is being applied.
cov_kernel = function(X, Y, kernel, ...) {

  if (!is.vector(X) || !is.vector(Y))
    stop("X and Y must be vectors.")

  K = matrix(NA, nrow = length(X), ncol = length(Y))

  for (i in 1:length(X)) {
    for (j in 1:length(Y)) {
      K[i, j] = kernel(X[i], Y[j], ...)
    }
  }

  return(K)
}

#' posteriorGP
#'
#' @param X Vector of training inputs.
#' @param Y Vector of training targets/outputs.
#' @param XStar Vector of inputs where the posterior distribution is evaluated,
#'    i.e XStar.
#' @param sigmaNoise Noise standard deviation sigma_n.
#' @param kernel A kernel function.
#' @param ... Parameters for the kernel function.
#'
#' @return Vector with the posterior mean and variance of f evaluated on set XStar
posteriorGP = function(X, Y, XStar, sigmaNoise, kernel, ...) {

  K = t(cov_kernel(X, X, kernel, ...))
  L = t(chol(K + diag(sigmaNoise^2, length(X))))

  # Predictive Mean
  alpha = solve(t(L), solve(L, Y))
  K_star = cov_kernel(X, XStar, kernel, ...)
  f_star = t(K_star) %*% alpha

  # Predictive Variance
  V = solve(L, K_star)
  V_f_star = cov_kernel(XStar, XStar, kernel, ...) - t(V) %*% V

  # Log Marginal Likelihood
  n = length(X)

  lml = -0.5 %*% t(Y) %*% alpha - sum(log(diag(L))) - (n/2) * log(2*pi)

  gp_object = list(mean=f_star,
                   variance=V_f_star,
                   lml = lml,
```

```
                    X=X, Y=Y,
                    XStar=XStar)

  class(gp_object) = "gp"

  return(gp_object)
}

#' plot.gp
#'
#' @param gp An Gaussian Process ("gp") object.
#' @param direct_plot TRUE by default. Determines if to plot directly or to
#'    return the plot.
#'
#' @return If direct_plot is set to TRUE, the plot object.
plot.gp = function(gp, direct_plot=TRUE) {
  df = data.frame(x = grid,
                  y = gp$mean,
                  y_upper = gp$mean + 1.96*sqrt(diag(gp$variance)),
                  y_lower = gp$mean - 1.96*sqrt(diag(gp$variance)))

  points = data.frame(x = gp$X, y = gp$Y)

  p = ggplot() +
  geom_line(aes(x = df$x, y = df$y, colour = "Mean")) +
  geom_ribbon(aes(x = df$x, ymin = df$y_lower, ymax = df$y_upper),
                  alpha=0.05,
                  linetype=3,
                  colour="grey70",
                  size=1,
                  fill="black") +
  geom_point(aes(x = points$x, y = points$y), color = "black",
           fill = "#FFC300", shape = 21, size = 2, stroke = 1) +
  labs(title = "Posterior Mean with 95 percent probability interval",
       y = "Position", x = "Time", color = "Legend") +
  scale_color_manual(values = c("#C70039", "#FF5733", "#581845")) +
  theme_minimal()

  if (!direct_plot)
    return(p)

  print(p)
}


sigma_f = 1
ell = 0.3
sigma_n = 0.1

x = 0.4
y = 0.719

grid = seq(from=-1, to=1, length.out = 100)
```

```r
res = posteriorGP(x, y, grid, sigma_n, se_kernel, d=sigma_f, ell=ell)


plot(res)


x = c(0.4, -0.6)
y = c(0.719, -0.004)
res = posteriorGP(x, y, grid, sigma_n, se_kernel, d=sigma_f, ell=ell)



plot(res)


x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)

input = matrix(c(x,y), nrow=2, byrow=TRUE)
rownames(input) = c("x", "y")
input


res = posteriorGP(input[1,], input[2,], grid, sigma_n, se_kernel, d=sigma_f,
                  ell=ell)


plot(res)


sigma_f = 1
ell = 1

res = posteriorGP(x, y, grid, sigma_n, se_kernel, d=sigma_f, ell=ell)


plot(res)


read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/\
         Code/TempTullinge.csv", header=TRUE, sep=";")

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/\
                GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")
names(data) = c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] = as.factor(data[,5])

set.seed(111);
SelectTraining = sample(1:dim(data)[1], size = 1000, replace = FALSE)
```