AML Lab 4 (GAUSSIAN PROCESSES)

Roshni Sundaramurthy

10/10/2019

Contents

The purpose of the lab is to put in practice some of the concepts covered in the lectures.

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), $\mathbf{x}_*$ (test input)

2: $L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$ } predictive mean eq. (2.25)

4: $\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$

$\mathbf{v} := L \backslash \mathbf{k}_*$ } predictive variance eq. (2.26)

6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$

$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$ eq. (2.30)

8: **return**: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

Algorithm 2.1: Predictions and log marginal likelihood for Gaussian process regression. The implementation addresses the matrix inversion required by eq. (2.25) and (2.26) using Cholesky factorization, see section A.4. For multiple test cases lines 4-6 are repeated. The log determinant required in eq. (2.30) is computed from the Cholesky factor (for large n it may not be possible to represent the determinant itself). The computational complexity is $n^3/6$ for the Cholesky decomposition in line 2, and $n^2/2$ for solving triangular systems in line 3 and (for each test case) in line 5.

Figure 1: Algorithm (posterior GP) 2.1 on page 19 of Rasmussen and Willams book

Implementing GP Regression

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

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

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 **$\underline{A}$** means the vector x that solves the equation** Ax = b** (see p. xvii in the book). This is implemented in R with the help of the function *solve*.

(1) 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_*.
- *sigmaNoise*: Noise standard deviation σ_n.
- k: Covariance function or kernel. That is, the kernel should be a separate function (see the file *GaussianProcesses.R* on the course web page).

Squared exponential kernel

$$k(x, x') = cov(f(x), f(x')) = \sigma_f^2 exp \frac{-|x - x'|^2}{2l^2}$$

(Ref GP_L1, slide:11)

Functions:

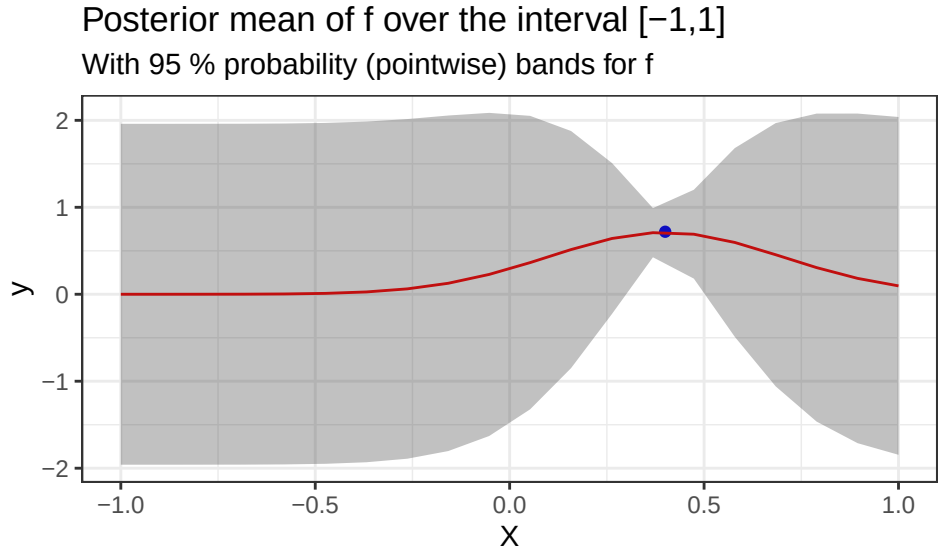
```r
# Parameters:
#   X1, X2 = vectors
#   l = the scale length parameter
# Returns:
#   a covariance matrix

# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF,l){
  K <- matrix(NA,length(x1),length(x2))
  for (i in 1:length(x2)){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

# Parameters:
#    X: Vector of training inputs
#  y: Vector of training targets/outputs
#  XStar: Vector of inputs where the posterior distribution is evaluated, i.e. X*
#  sigmaNoise: Noise standard deviation $\sigma_n$
#  k: Covariance function or kernel. That is, the kernel should be a separate function

# Returns:
#   Posterior mean and variance

# posteriorGP function
posteriorGP <- function(X,y,XStar,sigmaNoise,k,sigma_l){
  sigmaF <- sigma_l[1]
  l <- sigma_l[2]
  I <- diag(1, nrow(k),ncol(k))
  L <- t(chol(k+(sigmaNoise^2*I)))
  alpha <- solve(t(L),solve(L,y))
  Mean <- t(SquaredExpKernel(X,XStar,sigmaF,l)) %*% alpha
  v <- solve(L,SquaredExpKernel(X,XStar,sigmaF,l))
  Variance <- SquaredExpKernel(XStar,XStar,sigmaF,l) - (t(v) %*% v)
  return(list(Mean,Variance))

}

# plot function

plotGP <- function(X,y,postGP){
  xGrid = seq(-1,1,length=20)
  mean = postGP[[1]]
  varianc = postGP[[2]]
  library(ggplot2)
  ggplot() +
    geom_point(aes(X, y = y),col = "blue", alpha = 1)+
    labs(title="Posterior mean of f over the interval [-1,1]",
         subtitle = "With 95 % probability (pointwise) bands for f")+
    geom_line(aes(xGrid,mean),col="red", alpha=1)+
    geom_ribbon(aes(ymin = mean - 1.96*sqrt(diag(varianc)),
                    ymax = mean + 1.96*sqrt(diag(varianc)),
```

```
                    xGrid), alpha = 0.3)+
    theme_bw()
}
```

(2) Now, let the prior hyperparameters be $\sigma_f = 1$ and $l = 0.3$. 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% probability (pointwise) bands for f.

```
xGrid = seq(-1,1,length=20)
sigmaF<-1;l=0.3;sigmaNoise=0.1 # for exercise 2,3,4

postGP1 <- posteriorGP(X=0.4,y=0.719,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(0.4,0.4,sigmaF=1,l=0.3),
                       sigma_l = c(1,0.3))
plotGP(.4,.719,postGP1)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

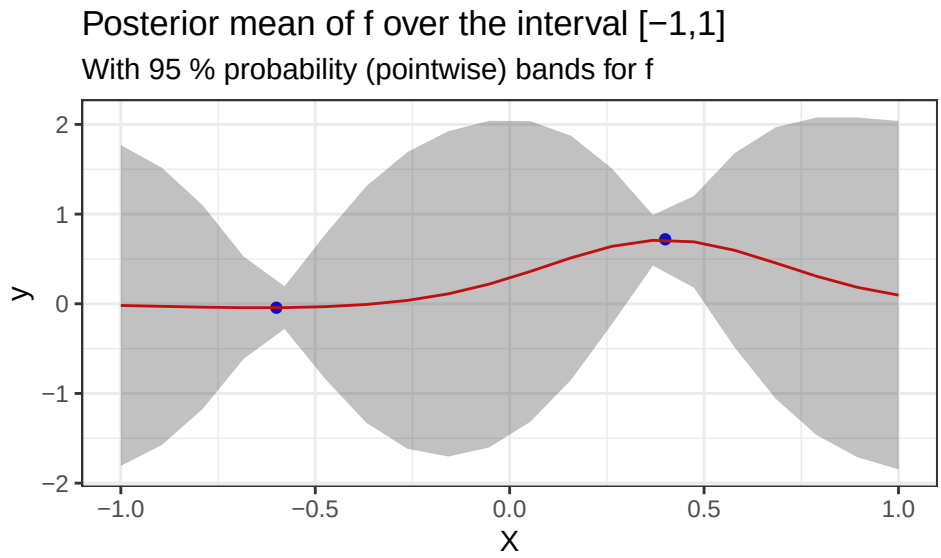


(3) Update your posterior from (2) with another observation: (x,y) = (-0.6, -0.044). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for f. Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations.

```
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
data.frame(x,y)
```

```
##      x      y
## 1  0.4  0.719
## 2 -0.6 -0.044
```

```
postGP2 <- posteriorGP(X=x,y=y,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(x,x,sigmaF=1,l=0.3),
                       sigma_l = c(1,0.3))
plotGP(x,y,postGP2)
```

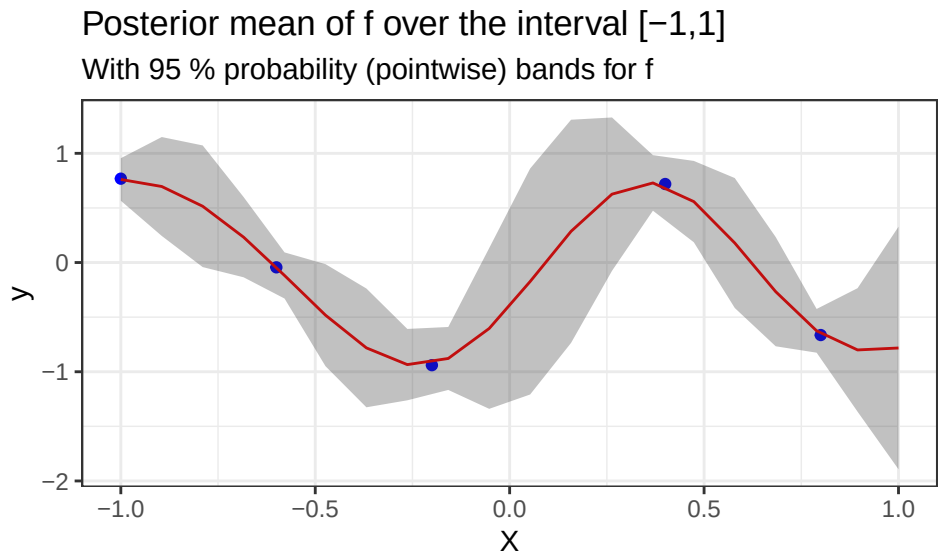


(4) Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95% probability (pointwise) bands for f.

```
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
data.frame(x,y)
```

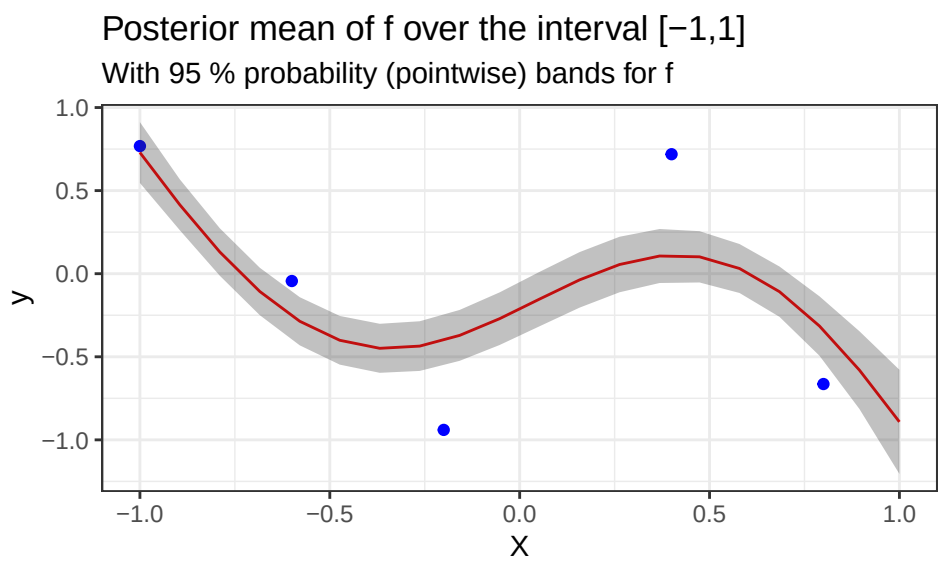
```
##      x      y
## 1 -1.0  0.768
## 2 -0.6 -0.044
## 3 -0.2 -0.940
## 4  0.4  0.719
## 5  0.8 -0.664
```

```
postGP3 <- posteriorGP(X=x,y=y,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(x,x,sigmaF=1,l=0.3),
                       sigma_l = c(1,0.3))
plotGP(x,y,postGP3)
```



(5) Repeat (4), this time with hyperparameters $\sigma_f = 1$ and $l = 1$. Compare the results.

```
sigmaF<-1;l=1;sigmaNoise=0.1 # for exercise 5
postGP4 <- posteriorGP(X=x,y=y,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(x,x,sigmaF=1,l=1),
                       sigma_l = c(1,1))
plotGP(x,y,postGP4)
```



For hyperparameters $\sigma_f = 1$ and $l = 0.3$: When we had one observation i.e, x=0.4, y=0.719, the posterior mean function exactly passes through the observation. This happend same for two observations. But when we had five observations, the mean function fails to pass through few observations.

For hyperparameters $\sigma_f = 1$ and $l = 1$: The mean function does not pass through the observations (five points). The function is so smooth here.

GP Regression with kernlab

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 * 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.

```
# data

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.cs
data$time <- c(1:2190)
data$day <- rep(1:365,6)
# data subsampling
# newdf <- df[c(rep(FALSE,4),TRUE), ] # to extract 5,10
df <- data[c(TRUE,rep(FALSE,4)), ] # extract 1,6,11
```

(1) 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 use the kernelMatrix function to compute the covariance matrix $K(X, X_*)$ for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

```
#Matern32 <- function(sigmaf = 1, ell = 1)
#{
#  rval <- function(x, y = NULL) {
#      r = sqrt(crossprod(x-y));
#      return(sigmaf^2*(1+sqrt(3)*r/ell)*exp(-sqrt(3)*r/ell))
#    }
#  class(rval) <- "kernel"
#  return(rval)
#}

# Squared exponential, k
k <- function(sigmaf, ell)
{
  rval <- function(x, y = NULL)
  {
    r = sqrt(crossprod(x-y))
    return(sigmaf^2*exp(-r^2/(2*ell^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}

library(kernlab)
```

```
##
```

```
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
# Simulating from the prior for ell = 2
kernel02 <- k(sigmaf = 1, ell = 2) # This constructs the covariance function (kernel)
X <- matrix(c(1,3,4), 3, 1) # Simulating some data
Xstar <- matrix(c(2,3,4), 3, 1)
# Computing the whole covariance matrix K from the kernel
K = kernelMatrix(kernel = kernel02, x=X, y=Xstar) # K(X,Xstar)
paste("Covariance matrix K(X,X*):")
```

```
## [1] "Covariance matrix K(X,X*):"
```

```
K
```

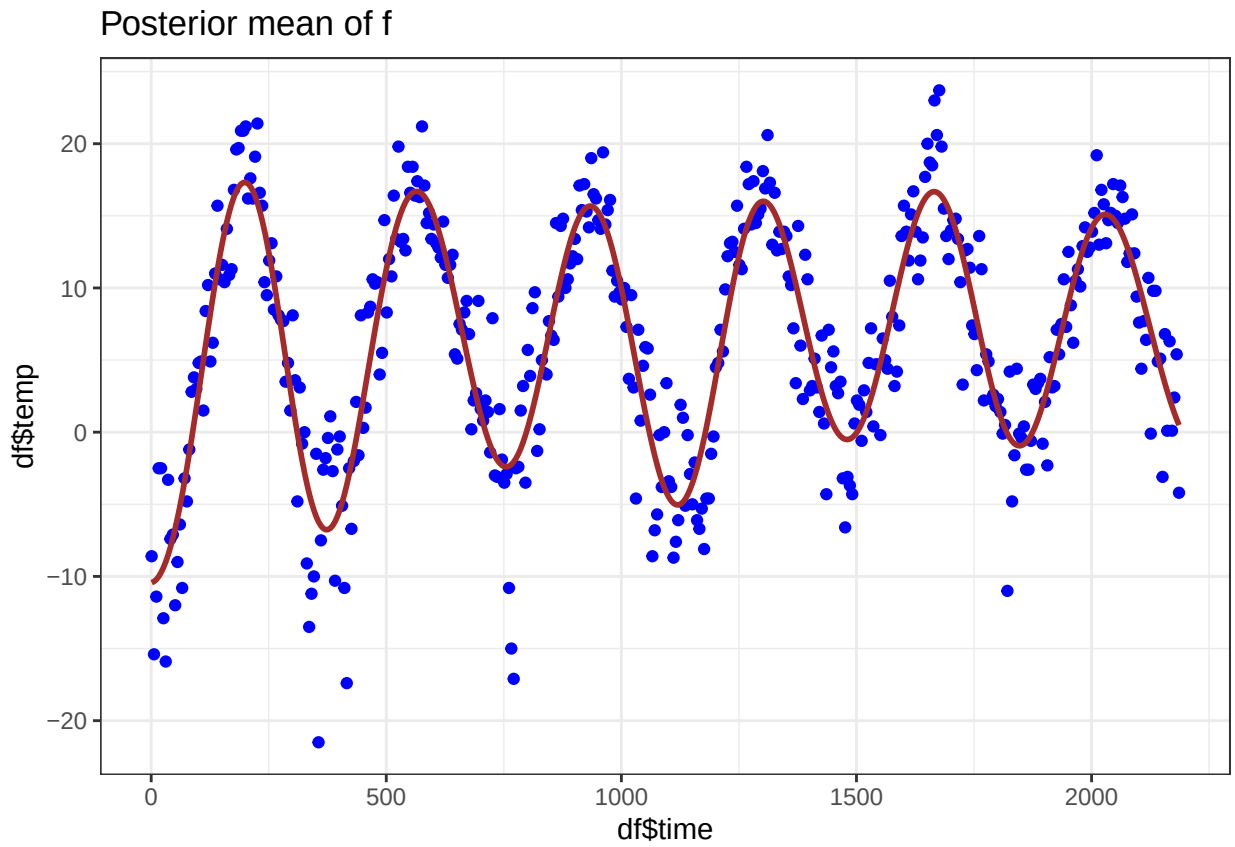
```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.8824969 0.6065307 0.3246525
## [2,] 0.8824969 1.0000000 0.8824969
## [3,] 0.6065307 0.8824969 1.0000000
```

(2) Consider first the following model:

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

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 $l = 0.2$. 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.

```
fit <- lm(temp ~ time + time^2, data = df)
#plot(fit)

sigmaNoise <- var(fit$residuals)
kern_func <- k(sigmaf = 20, ell = 0.2)
estimateGP <- gausspr(df$time,df$temp,kernel = kern_func,var = sigmaNoise)

# computing posterior mean
post_mean_time <- predict(estimateGP, df$time)

ggplot()+
geom_point(aes(df$time, df$temp), col="blue")+
  geom_line(aes(df$time, post_mean_time), col="brown",lwd=1)+
  ggtitle("Posterior mean of f")+theme_bw()
```

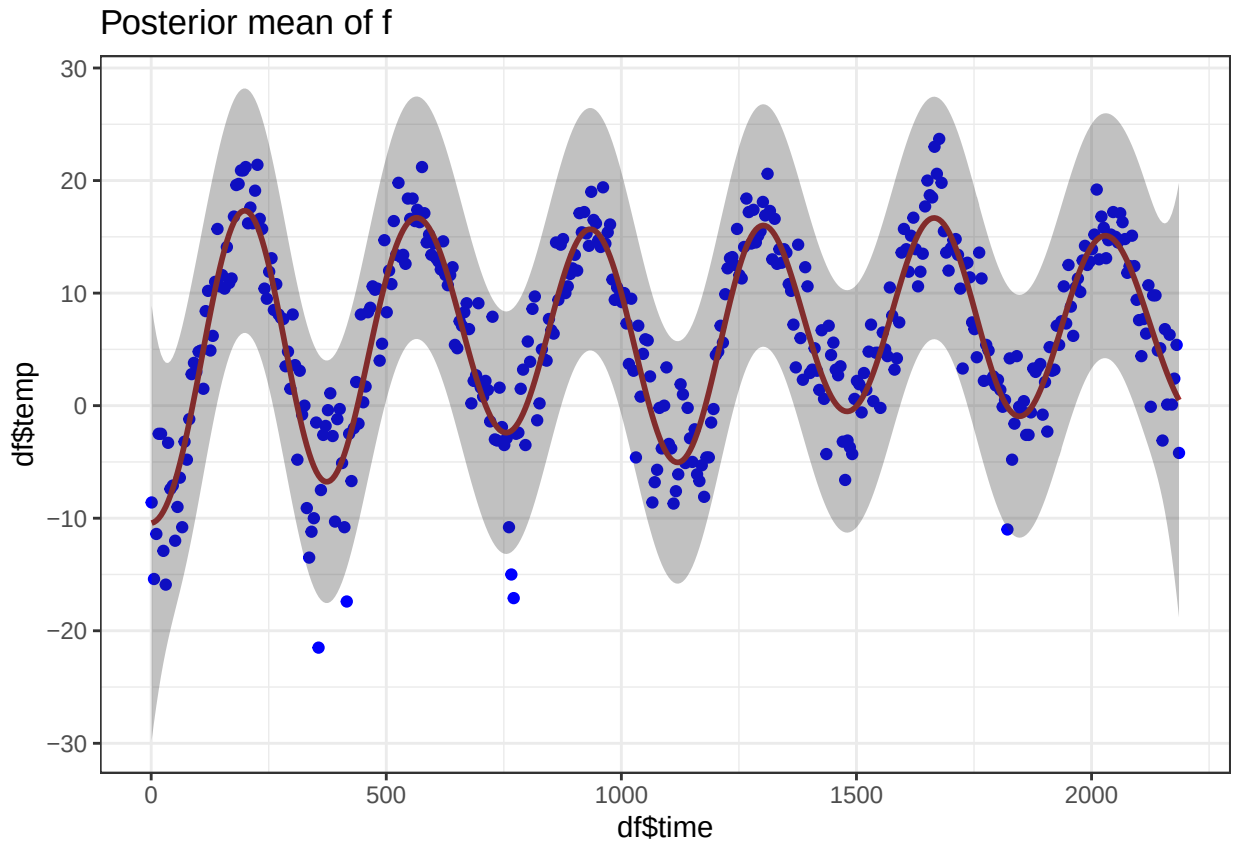


The posterior mean at every data point in the training dataset is computed using predict() function. With $\sigma_f = 20$ and $l = 0.2$, the posterior mean function passes well through the observations. But still it misses few data points to cover. It appears as jagged function. When l has been increased, the function becomes so smooth and fails to cover the data points.

(3) *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 % 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. When plotting the results, do not forget to unscale by multiplying with the standard deviation and adding the mean.

```
s_time = scale(df$time)
s_temp = scale(df$temp)
post <- posteriorGP(s_time, s_temp , s_time, sigmaNoise = sqrt(sigmaNoise),
                    k=SquaredExpKernel(s_time,s_time,sigmaF=20,l=0.2), c(20, 0.2))
# unscaling for plots
mean <- post[[1]]*attr(s_temp, 'scaled:scale')+ attr(s_temp, 'scaled:center')
var <- post[[2]]*attr(s_temp, 'scaled:scale')+ attr(s_temp, 'scaled:center')


ggplot()+
geom_point(aes(df$time, df$temp), col="blue")+
  geom_line(aes(df$time, post_mean_time), col="brown",lwd=1)+
```

```
  ggtitle("Posterior mean of f")+theme_bw()+
  geom_ribbon(aes(ymin = mean - 1.96*sqrt(diag(var)),
                  ymax = mean + 1.96*sqrt(diag(var)), df$time),
              alpha = 0.3)
```



The posterior mean and variance at every data point in the training dataset is computed using posteriorGP() function and 95 % probability bands for f is superimposed on the above plot. These bands cover almost all data points.

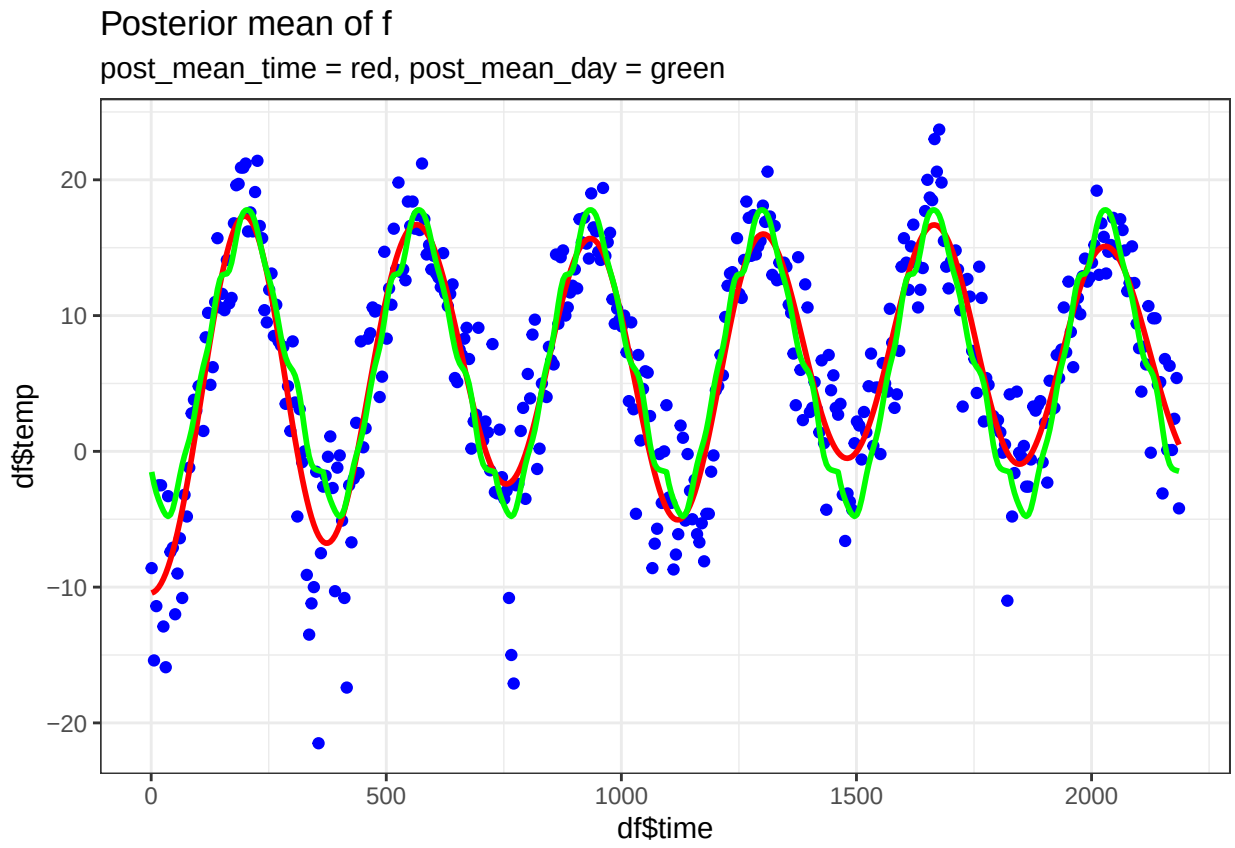
(4) Consider now the following model:

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

Estimate the model using the squared exponential function with $\sigma_f = 20$ and $l = 0.2$. 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?

```
fit <- lm(temp ~ day + day^2, data = df)
#plot(fit)

sigmaNoise <- var(fit$residuals)
kern_func <- k(sigmaf = 20, ell = 0.2)
estimateGP <- gausspr(df$day,df$temp,kernel = kern_func,var = sigmaNoise)


post_mean_day <- predict(estimateGP, df$day)
```

```
ggplot()+
geom_point(aes(df$time, df$temp), colour="blue")+
  geom_line(aes(df$time, post_mean_time), colour="red",lwd=1, show.legend = T)+
  geom_line(aes(df$time, post_mean_day), colour="green",lwd=1,show.legend = T)+
  ggtitle("Posterior mean of f", subtitle = "post_mean_time = red, post_mean_day = green")+theme_bw()
```




```
#  scale_colour_manual("",values = c("post_mean_time"="brown"
# , "post_mean_day"="black"),labels = c("a", "b"))

#scale_color_manual(values = c("blue", "brown","black"))
#name = "Scores", labels = c("post_mean_time", "post_mean_day"),
```

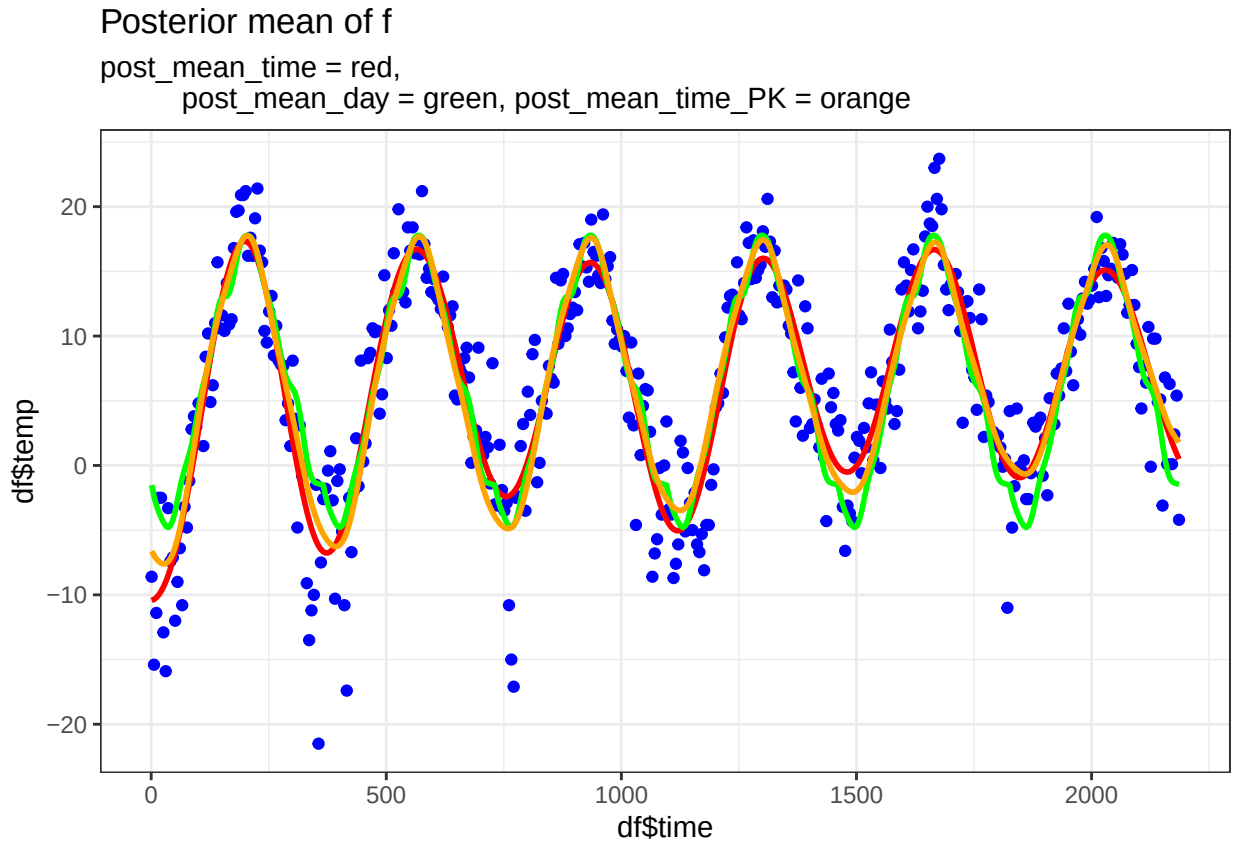
The posterior mean for both time and day has been plotted. From the above results, we can say that the green curve (day) seems to pass through more data points than the red curve (time). But, it may lead to overfitting.

(5) Finally, implement a generalization of the periodic kernel given in the lectures:

$$k(x,x^{'}) = \sigma_f^2 exp\left(-\frac{2sin^2(\pi|x-x^{'}|/d)}{l_1^2}\right) exp\left(-\frac{1}{2}\frac{|x-x^{'}|^2}{l_2^2}\right)$$

Note that we have two different length scales here, and l_2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20$, $l_1 = 1$, $l_2 = 10$ and $d = 365/sd(time)$. The reason for the rather strange period here is that kernlab standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20$ and $l = 0.2$). Discuss the results.

```r
# implementing a generalization of the periodic kernel
periodic_kernel <- function(sigmaf, ell1, ell2, d){
  rval <- function(x, y = NULL)
  {
    r = abs(x - y)
    return(sigmaf^2 * exp((-2 * sin(pi * r/d)^2)/ell1^2) * exp((-r^2/(2*ell2^2))) )
  }
  class(rval) <- "kernel"
  return(rval)
}



fit <- lm(temp ~ time + time^2, data = df)
#plot(fit)

sigmaNoise <- var(fit$residuals)
kern_func <- periodic_kernel(sigmaf = 20, ell1 = 1, ell2 = 10, d = 365/sd(df$time))
estimateGP <- gausspr(df$time,df$temp,kernel = kern_func,var = sigmaNoise)


post_mean_time_PK <- predict(estimateGP, df$time)

ggplot()+
geom_point(aes(df$time, df$temp), colour="blue")+
  geom_line(aes(df$time, post_mean_time), colour="red",lwd=1)+
  geom_line(aes(df$time, post_mean_day), colour="green",lwd=1)+
  geom_line(aes(df$time, post_mean_time_PK), colour="orange",lwd=1)+
  ggtitle("Posterior mean of f",
          subtitle = "post_mean_time = red,
          post_mean_day = green, post_mean_time_PK = orange")+
  theme_bw()
```



The GP model using the time variable is computed with the periodic kernel with hyperparameters $\sigma_f = 20$, $l_1 = 1$, $l_2 = 10$ and $d = 365/sd(time)$. But here two lengths are used where l2 is used to control correlation between the same day in different years. This periodic kernel's output seems to be more similar to the output obtained by previous squared exponential kernel.

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)

(1) 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 KernLab-Demo.R available on the course website. Compute the confusion matrix for the classifier and its accuracy.

First we splitted the data as train and test. Then fitted a Gaussian process classification model for fraud on the training data using only varWave and skewWave variables. The contour plots for fraud = 0 and 1 are displayed below.

```r
# load data
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud

names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5]) # fraud variable

# splitting data
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining, ] # train data (1000)
test <- data[-SelectTraining, ] # test data (372)

# fitting a Gaussian process classification model for fraud on the training data

# Using only varWave and skewWave to classify
GPfitbank <- gausspr(fraud ~  varWave + skewWave, data=train)
```

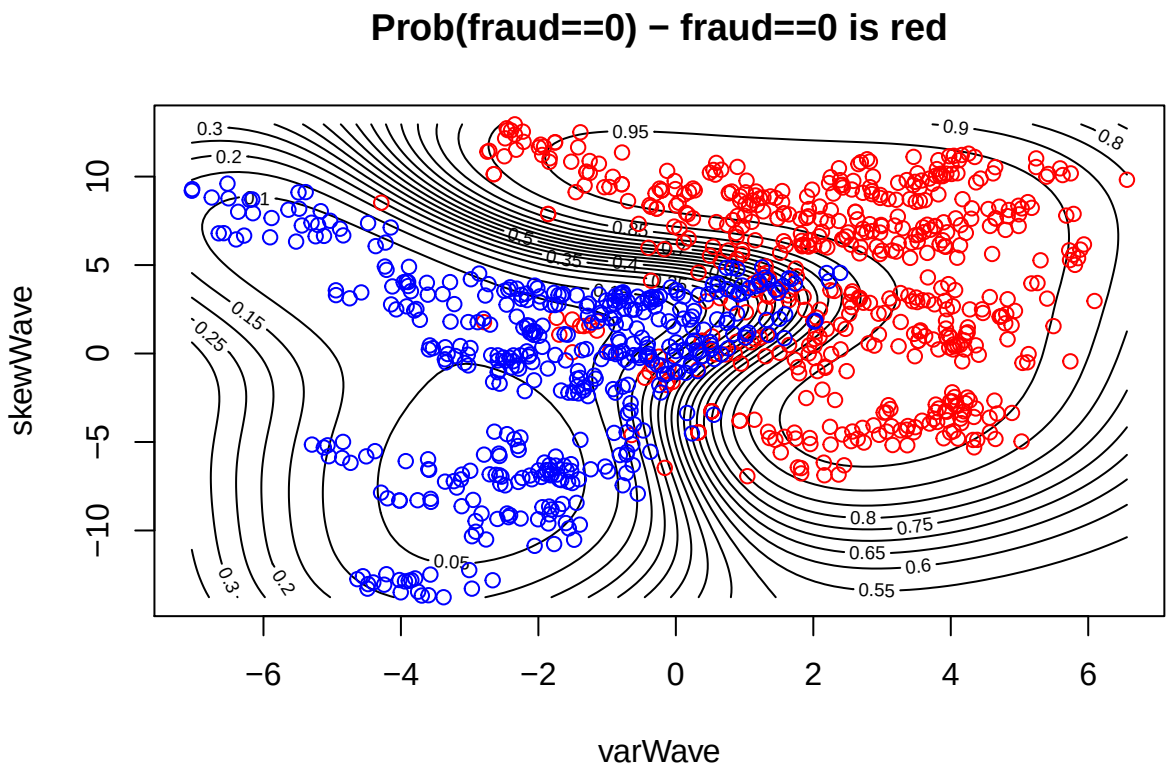
```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```r
# predict on the training set
prediction <- predict(GPfitbank,train[,1:2])
conf_mat_tr <- table(prediction, train[,5]) # confusion matrix
```

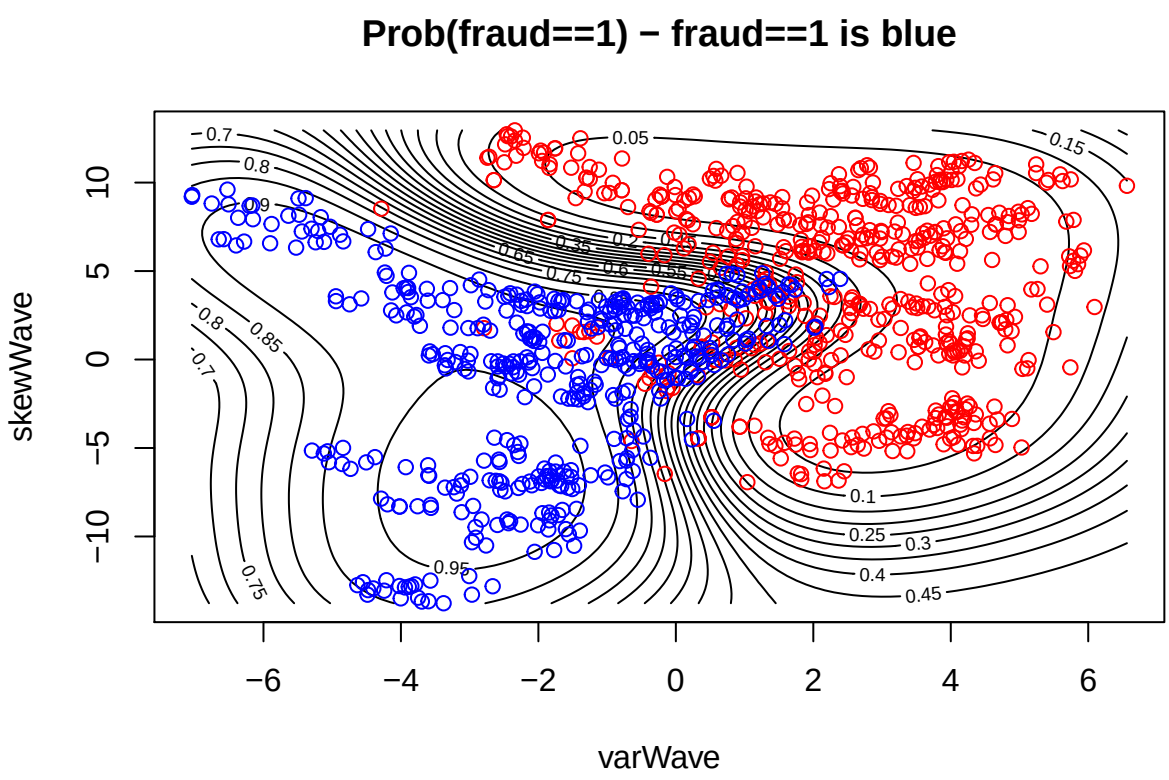
```
## Confusion Matrix for test data :
```

```
##
## prediction   0   1
##          0 512  24
##          1  44 420
```

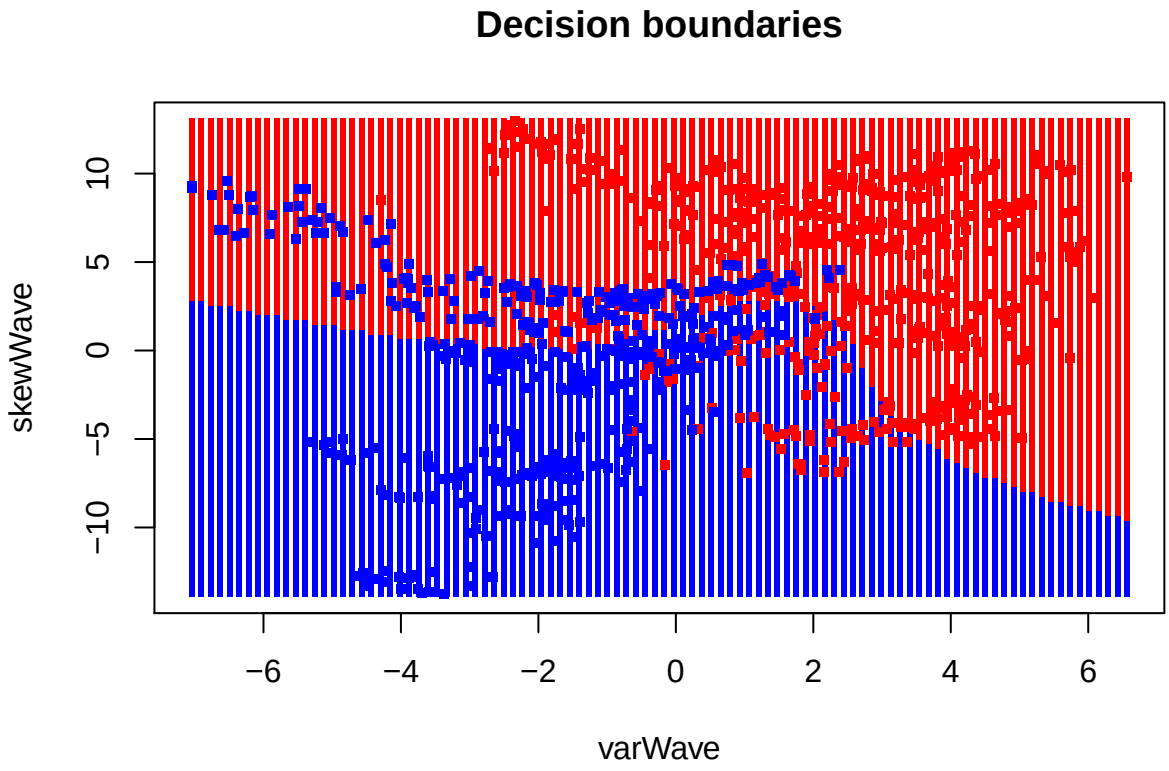
```r
library("AtmRay") # To make 2D grid

# class probabilities
probPreds <- predict(GPfitbank, train[,1:2], type="probabilities")
x1 <- seq(min(train[,1]),max(train[,1]),length=100)
x2 <- seq(min(train[,2]),max(train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]
probPreds <- predict(GPfitbank, gridPoints, type="probabilities")

# Plotting for Prob(fraud==0) RED
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud==0) - fraud==0 is red')
points(train[train[,5]==0,1],train[train[,5]==0,2],col="red")
points(train[train[,5]==1,1],train[train[,5]==1,2],col="blue")
```




```
# Plotting for Prob(fraud==1) BLUE
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud==1) - fraud==1 is blue')
points(train[train[,5]==0,1],train[train[,5]==0,2],col="red")
points(train[train[,5]==1,1],train[train[,5]==1,2],col="blue")
```




```
# Plotting the decision boundaries
meanPred <- matrix(max.col(probPreds),100,byrow = TRUE)
plot(gridPoints,  pch=".", cex=3, col=ifelse(meanPred==1, "red", "blue"),
     main = 'Decision boundaries')
points(train[train[,5]==0,1],train[train[,5]==0,2],col="red", cex=5, pch=".")
points(train[train[,5]==1,1],train[train[,5]==1,2],col="blue",  cex=5, pch=".")
```



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

```
# predict on the test set
prediction <- predict(GPfitbank,test[,1:2])
conf_mat_test <- table(prediction, test[,5]) # confusion matrix
accuracy <- (sum(diag(conf_mat_test))/sum(conf_mat_test))*100
```

```
## Confusion Matrix for test data :
```

```
##
## prediction   0   1
##          0 191   9
##          1  15 157
```

```
## Accuracy for test data :  93.5483870967742
```

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

```
# Using varWave, skewWave, kurtWave and entropyWave to classify
GPfitbank <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=train)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
# predict on the test set
prediction <- predict(GPfitbank,test[,1:4])
conf_mat_test <- table(prediction, test[,5]) # confusion matrix
accuracy <- (sum(diag(conf_mat_test))/sum(conf_mat_test))*100
```

```
## Confusion Matrix for test data :
```

```
##
## prediction   0   1
##          0 205   0
##          1   1 166
```

```
## Accuracy for test data :  99.7311827956989
```

The accuracy for the model with all four covariates is observed to be higher than the model with only two covariates. Hence, we can conclude that all the for covariates ($p(fraud/varWave,skewWave,kurtWave,entropyWave)$) are equally contributing for prediction giving the better accuracy than the ($p(fraud/varWave,skewWave)$).

Reference

1. https://stackoverflow.com/questions/10287545/backtransform-scale-for-plotting

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
knitr::include_graphics("post_GP.png")
# Parameters:
#   X1, X2 = vectors
#   l = the scale length parameter
# Returns:
#   a covariance matrix

# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF,l){
  K <- matrix(NA,length(x1),length(x2))
  for (i in 1:length(x2)){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

# Parameters:
#    X: Vector of training inputs
#  y: Vector of training targets/outputs
#  XStar: Vector of inputs where the posterior distribution is evaluated, i.e. X*
#  sigmaNoise: Noise standard deviation $\sigma_n$
#  k: Covariance function or kernel. That is, the kernel should be a separate function

# Returns:
#   Posterior mean and variance

# posteriorGP function
posteriorGP <- function(X,y,XStar,sigmaNoise,k,sigma_l){
  sigmaF <- sigma_l[1]
  l <- sigma_l[2]
  I <- diag(1, nrow(k),ncol(k))
  L <- t(chol(k+(sigmaNoise^2*I)))
```

```
  alpha <- solve(t(L),solve(L,y))
  Mean <- t(SquaredExpKernel(X,XStar,sigmaF,l)) %*% alpha
  v <- solve(L,SquaredExpKernel(X,XStar,sigmaF,l))
  Variance <- SquaredExpKernel(XStar,XStar,sigmaF,l) - (t(v) %*% v)
  return(list(Mean,Variance))

}

# plot function

plotGP <- function(X,y,postGP){
  xGrid = seq(-1,1,length=20)
  mean = postGP[[1]]
  varianc = postGP[[2]]
  library(ggplot2)
  ggplot() +
    geom_point(aes(X, y = y),col = "blue", alpha = 1)+
    labs(title="Posterior mean of f over the interval [-1,1]",
         subtitle = "With 95 % probability (pointwise) bands for f")+
    geom_line(aes(xGrid,mean),col="red", alpha=1)+
    geom_ribbon(aes(ymin = mean - 1.96*sqrt(diag(varianc)),
                    ymax = mean + 1.96*sqrt(diag(varianc)),
                    xGrid), alpha = 0.3)+
    theme_bw()
}

xGrid = seq(-1,1,length=20)
sigmaF<-1;l=0.3;sigmaNoise=0.1 # for exercise 2,3,4

postGP1 <- posteriorGP(X=0.4,y=0.719,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(0.4,0.4,sigmaF=1,l=0.3),
                       sigma_l = c(1,0.3))
plotGP(.4,.719,postGP1)
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
data.frame(x,y)
postGP2 <- posteriorGP(X=x,y=y,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(x,x,sigmaF=1,l=0.3),
                       sigma_l = c(1,0.3))
plotGP(x,y,postGP2)
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
data.frame(x,y)
postGP3 <- posteriorGP(X=x,y=y,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(x,x,sigmaF=1,l=0.3),
                       sigma_l = c(1,0.3))
plotGP(x,y,postGP3)
sigmaF<-1;l=1;sigmaNoise=0.1 # for exercise 5
postGP4 <- posteriorGP(X=x,y=y,XStar=xGrid,sigmaNoise=0.1,
                       k=SquaredExpKernel(x,x,sigmaF=1,l=1),
                       sigma_l = c(1,1))
plotGP(x,y,postGP4)
# data
```

```r
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.cs
data$time <- c(1:2190)
data$day <- rep(1:365,6)
# data subsampling
# newdf <- df[c(rep(FALSE,4),TRUE), ] # to extract 5,10
df <- data[c(TRUE,rep(FALSE,4)), ] # extract 1,6,11

#Matern32 <- function(sigmaf = 1, ell = 1)
#{
#  rval <- function(x, y = NULL) {
#      r = sqrt(crossprod(x-y));
#      return(sigmaf^2*(1+sqrt(3)*r/ell)*exp(-sqrt(3)*r/ell))
#    }
#  class(rval) <- "kernel"
#  return(rval)
#}

# Squared exponential, k
k <- function(sigmaf, ell)
{
  rval <- function(x, y = NULL)
  {
    r = sqrt(crossprod(x-y))
    return(sigmaf^2*exp(-r^2/(2*ell^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}

library(kernlab)
# Simulating from the prior for ell = 2
kernel02 <- k(sigmaf = 1, ell = 2) # This constructs the covariance function (kernel)
X <- matrix(c(1,3,4), 3, 1) # Simulating some data
Xstar <- matrix(c(2,3,4), 3, 1)
# Computing the whole covariance matrix K from the kernel
K = kernelMatrix(kernel = kernel02, x=X, y=Xstar) # K(X,Xstar)
paste("Covariance matrix K(X,X*):")
K

fit <- lm(temp ~ time + time^2, data = df)
#plot(fit)

sigmaNoise <- var(fit$residuals)
kern_func <- k(sigmaf = 20, ell = 0.2)
estimateGP <- gausspr(df$time,df$temp,kernel = kern_func,var = sigmaNoise)

# computing posterior mean
post_mean_time <- predict(estimateGP, df$time)

ggplot()+
geom_point(aes(df$time, df$temp), col="blue")+
  geom_line(aes(df$time, post_mean_time), col="brown",lwd=1)+
  ggtitle("Posterior mean of f")+theme_bw()
```

```r
s_time = scale(df$time)
s_temp = scale(df$temp)
post <- posteriorGP(s_time, s_temp , s_time, sigmaNoise = sqrt(sigmaNoise),
                    k=SquaredExpKernel(s_time,s_time,sigmaF=20,l=0.2), c(20, 0.2))
# unscaling for plots
mean <- post[[1]]*attr(s_temp, 'scaled:scale')+ attr(s_temp, 'scaled:center')
var <- post[[2]]*attr(s_temp, 'scaled:scale')+ attr(s_temp, 'scaled:center')


ggplot()+
geom_point(aes(df$time, df$temp), col="blue")+
  geom_line(aes(df$time, post_mean_time), col="brown",lwd=1)+
  ggtitle("Posterior mean of f")+theme_bw()+
  geom_ribbon(aes(ymin = mean - 1.96*sqrt(diag(var)),
                  ymax = mean + 1.96*sqrt(diag(var)), df$time),
              alpha = 0.3)


fit <- lm(temp ~ day + day^2, data = df)
#plot(fit)

sigmaNoise <- var(fit$residuals)
kern_func <- k(sigmaf = 20, ell = 0.2)
estimateGP <- gausspr(df$day,df$temp,kernel = kern_func,var = sigmaNoise)


post_mean_day <- predict(estimateGP, df$day)

ggplot()+
geom_point(aes(df$time, df$temp), colour="blue")+
  geom_line(aes(df$time, post_mean_time), colour="red",lwd=1, show.legend = T)+
  geom_line(aes(df$time, post_mean_day), colour="green",lwd=1,show.legend = T)+
  ggtitle("Posterior mean of f", subtitle = "post_mean_time = red, post_mean_day = green")+theme_bw()


#  scale_colour_manual("",values = c("post_mean_time"="brown"
# , "post_mean_day"="black"),labels = c("a", "b"))

#scale_color_manual(values = c("blue", "brown","black"))
#name = "Scores", labels = c("post_mean_time", "post_mean_day"),
# implementing a generalization of the periodic kernel
periodic_kernel <- function(sigmaf, ell1, ell2, d){
  rval <- function(x, y = NULL)
  {
    r = abs(x - y)
    return(sigmaf^2 * exp((-2 * sin(pi * r/d)^2)/ell1^2) * exp((-r^2/(2*ell2^2))) )
  }
  class(rval) <- "kernel"
  return(rval)
}
```

```r
fit <- lm(temp ~ time + time^2, data = df)
#plot(fit)

sigmaNoise <- var(fit$residuals)
kern_func <- periodic_kernel(sigmaf = 20, ell1 = 1, ell2 = 10, d = 365/sd(df$time))
estimateGP <- gausspr(df$time,df$temp,kernel = kern_func,var = sigmaNoise)


post_mean_time_PK <- predict(estimateGP, df$time)

ggplot()+
geom_point(aes(df$time, df$temp), colour="blue")+
  geom_line(aes(df$time, post_mean_time), colour="red",lwd=1)+
  geom_line(aes(df$time, post_mean_day), colour="green",lwd=1)+
  geom_line(aes(df$time, post_mean_time_PK), colour="orange",lwd=1)+
  ggtitle("Posterior mean of f",
          subtitle = "post_mean_time = red,
          post_mean_day = green, post_mean_time_PK = orange")+
  theme_bw()


# load data
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud

names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5]) # fraud variable

# splitting data
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train <- data[SelectTraining, ] # train data (1000)
test <- data[-SelectTraining, ] # test data (372)

# fitting a Gaussian process classification model for fraud on the training data

# Using only varWave and skewWave to classify
GPfitbank <- gausspr(fraud ~  varWave + skewWave, data=train)
# predict on the training set
prediction <- predict(GPfitbank,train[,1:2])
conf_mat_tr <- table(prediction, train[,5]) # confusion matrix

cat(paste("Confusion Matrix for test data : "))
conf_mat_tr
library("AtmRay") # To make 2D grid

# class probabilities
probPreds <- predict(GPfitbank, train[,1:2], type="probabilities")
x1 <- seq(min(train[,1]),max(train[,1]),length=100)
x2 <- seq(min(train[,2]),max(train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
```

```
names(gridPoints) <- names(train)[1:2]
probPreds <- predict(GPfitbank, gridPoints, type="probabilities")

# Plotting for Prob(fraud==0) RED
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud==0) - fraud==0 is red')
points(train[train[,5]==0,1],train[train[,5]==0,2],col="red")
points(train[train[,5]==1,1],train[train[,5]==1,2],col="blue")

# Plotting for Prob(fraud==1) BLUE
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud==1) - fraud==1 is blue')
points(train[train[,5]==0,1],train[train[,5]==0,2],col="red")
points(train[train[,5]==1,1],train[train[,5]==1,2],col="blue")

# Plotting the decision boundaries
meanPred <- matrix(max.col(probPreds),100,byrow = TRUE)
plot(gridPoints,  pch=".", cex=3, col=ifelse(meanPred==1, "red", "blue"),
     main = 'Decision boundaries')
points(train[train[,5]==0,1],train[train[,5]==0,2],col="red", cex=5, pch=".")
points(train[train[,5]==1,1],train[train[,5]==1,2],col="blue",  cex=5, pch=".")
# predict on the test set
prediction <- predict(GPfitbank,test[,1:2])
conf_mat_test <- table(prediction, test[,5]) # confusion matrix
accuracy <- (sum(diag(conf_mat_test))/sum(conf_mat_test))*100
cat(paste("Confusion Matrix for test data : "))
conf_mat_test
cat(paste("Accuracy for test data : ", accuracy))
# Using varWave, skewWave, kurtWave and entropyWave to classify
GPfitbank <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=train)
# predict on the test set
prediction <- predict(GPfitbank,test[,1:4])
conf_mat_test <- table(prediction, test[,5]) # confusion matrix
accuracy <- (sum(diag(conf_mat_test))/sum(conf_mat_test))*100
cat(paste("Confusion Matrix for test data : "))
conf_mat_test
cat(paste("Accuracy for test data : ", accuracy))
```