Advanced Machine Learning

Hugo Morvan (hugmo418)

2024-10-11

Lab 4: Gaussian Processes

1. 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 $f \sim GP(0, k(x, x'))$

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 qmatrix. So, you need to transpose the output of the R function. In the algorithm, the notation A/b 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`.

Here is what you need to do:

1.1)

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).

```
library(kernlab)

posteriorGP <- function(X, y, Xstar, sigmaNoise, k){
  #Algorithm 2.1 on page 19 of Rasmussen and Willams' book.

  K = kernelMatrix(kernel = k, x = X, y = X) #Covariance K(X,X)
  K_star = kernelMatrix(kernel = k, x = X, y = Xstar) #Covariance K(X, X*)
  K_star_star = kernelMatrix(kernel = k, x = Xstar, y = Xstar) #Covariance K(X*, X*)
```

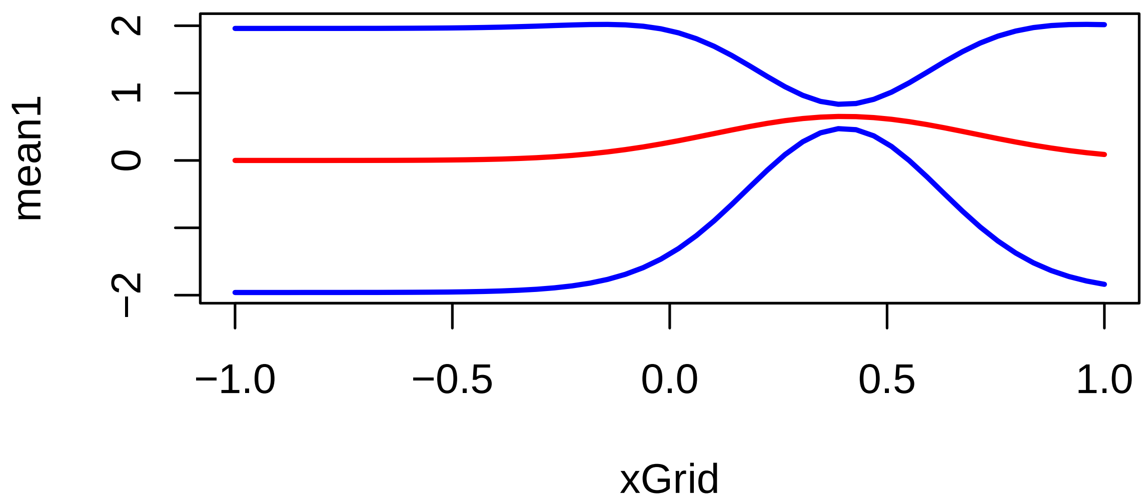
```r
  L = chol(K + sigmaNoise*diag(dim(K)[1])) #Cholesky

  #Note that L in the algorithm is a lower triangular matrix,
  #whereas the R function returns an upper triangular qmatrix
  L = t(L)

  #Predictive mean
  alpha = solve(t(L),solve(L,y))
  pred_mean = t(K_star) %*% alpha

  #Predictive variance
  v = solve(L, K_star)
  pred_var = K_star_star - t(v)%*%v

  return(list("mean" = pred_mean, "var" = pred_var))
}
```

1.2)

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 .

```r
#Squared exponential kernel:
SEkernel <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
      r = sqrt(crossprod(x-y));
      return(sigmaf**2 * exp(-(r**2)/(2*ell**2)))
    }
  class(rval) <- "kernel"
  return(rval)
}

sigmaf = 1
sigma_n = 0.1
ell = 0.3
xGrid <- matrix(seq(-1,1,length=50))
k1 = SEkernel(sigmaf = sigmaf, ell = ell)
#Updating prior with a single observation
updated_prior = posteriorGP(X = matrix(0.4), y = matrix(0.719), Xstar = xGrid, sigmaNoise = sigma_n, k

mean1 = updated_prior$mean
var1 = diag(updated_prior$var)
plot(xGrid, mean1, col="red", type='l', lwd = 2, ylim = range(mean1,mean1+1.96*var1, mean1-1.96*var1))
lines(xGrid, mean1+1.96*var1,col="blue", lw=2)
lines(xGrid, mean1-1.96*var1,col="blue", lw=2)
```

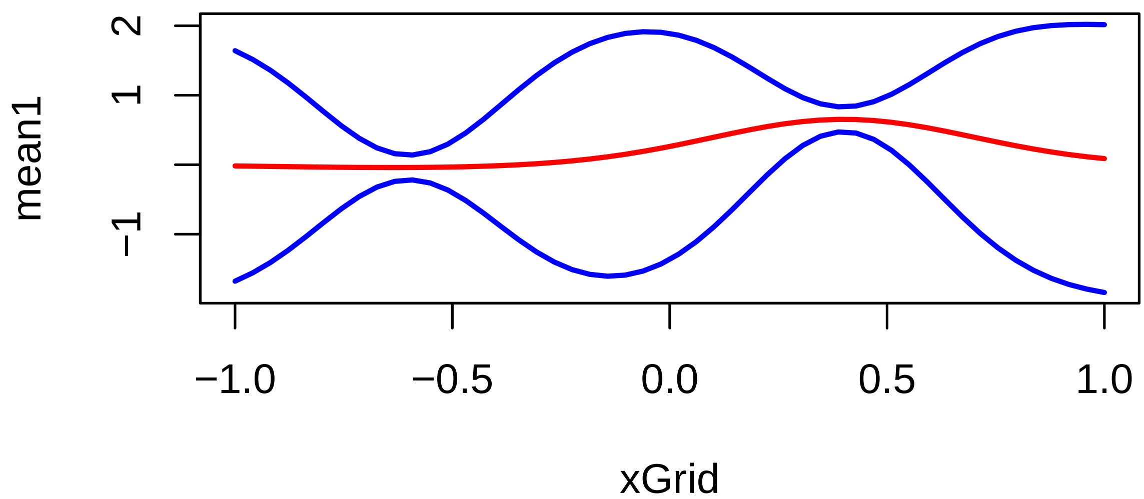


1.3)

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 (point-wise) 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.

```
updated_prior = posteriorGP(X = matrix(c(0.4, -0.6)), y = matrix(c(0.719, -0.044)), Xstar = xGrid, sigma

mean1 = updated_prior$mean
var1 = diag(updated_prior$var)
plot(xGrid, mean1, col="red", type='l', lwd = 2, ylim = range(mean1,mean1+1.96*var1, mean1-1.96*var1))
lines(xGrid, mean1+1.96*var1,col="blue", lw=2)
lines(xGrid, mean1-1.96*var1,col="blue", lw=2)
```



1.4)

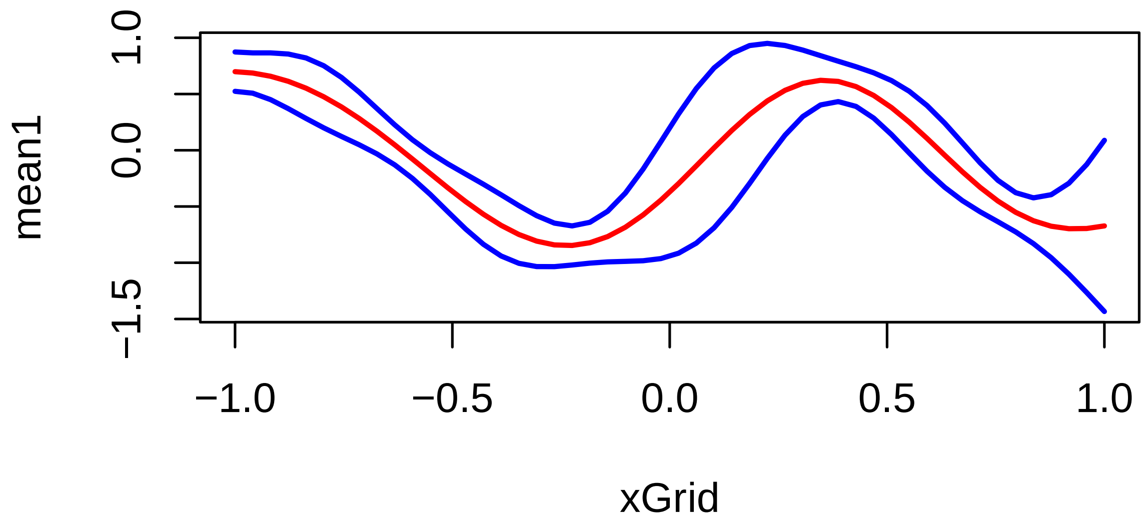
Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.

x	-1.0	-0.6	-0.2	0.4	0.8
y	0.768	-0.044	-0.940	0.719	-0.664

1.5)

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

```
updated_prior = posteriorGP(X = matrix(c(-1.0 , -0.6 , -0.2 , 0.4 , 0.8)),
                            y = matrix(c(0.768 , -0.044 , -0.940 , 0.719 , -0.664)),
                            Xstar = xGrid, sigmaNoise = sigma_n, k = k1 )

mean1 = updated_prior$mean
var1 = diag(updated_prior$var)
plot(xGrid, mean1, col="red", type='l', lwd = 2, ylim = range(mean1,mean1+1.96*var1, mean1-1.96*var1))
lines(xGrid, mean1+1.96*var1,col="blue", lw=2)
lines(xGrid, mean1-1.96*var1,col="blue", lw=2)
```



2. GP Regression with kernlab.

In this exercise, you will work with the daily mean temperature 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`$= 1, 2, \ldots, 365 * 6 = 2190$). Also, create the variable `day` that records the day number since the start of each year (i.e., `day`$= 1, 2, \ldots, 365, 1, 2, \ldots, 365$). 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`$= 1, 6, 11, \ldots, 2186$ and `day`$= 1, 6, 11, \ldots, 361, 1, 6, 11, \ldots, 361$.

```
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
dates = data$date
dates = as.Date(dates, "%d/%m/%y")
temps = as.numeric(data$temp)


t0 = dates[1]

days_to_t0 = function(day, t0){
  return(as.numeric(difftime(day, t0, units = "days"))+1)
}
time = sapply(dates, days_to_t0, t0 = t0)

days_to_newyear = function(day, t0){
```

```r
  return(as.numeric(difftime(day, t0, units = "days"))%%365+1)
}
day = sapply(dates, days_to_newyear, t0 = t0)

#Subsampling every 5 datapoints
subsample <- function(x,by){
  end = length(x)
  idx = seq.int(1, end, by = by)
  return(x[idx])
}
#test = seq.int(1,100)
#subsample(test, 5)
time = subsample(time, 5)

day = subsample(day, 5)

temp = subsample(temps, 5)
```

2.1)

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 $X* = (2, 3, 4)^T$.

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

k = SEkernel()
print(k(1,2))
```

```
##           [,1]
## [1,] 0.6065307
```

```r
X = matrix(t(c(1,3,4)))
Xstar = matrix(t(c(2,3,4)))
print(kernelMatrix(kernel = k, x = X, y = Xstar) )
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

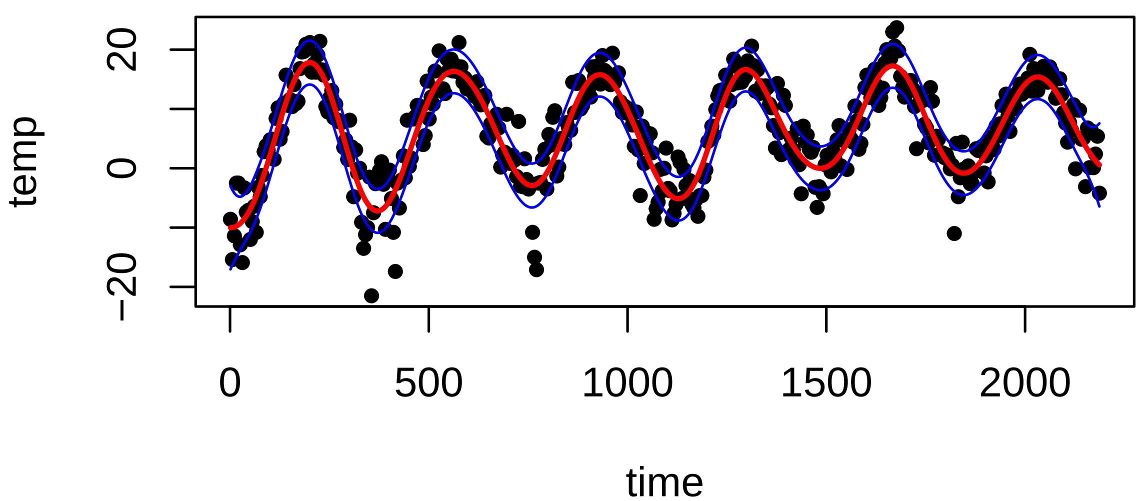
2.2)

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 `gausspr` function with the squared exponential function from (1) with $\sigma_f = 20$ and $l = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these σ_f and l values are not suitable). 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). Plot also the 95 % probability (pointwise) bands for f. Play around with different values on σ_f and l (no need to write this in the report though).

```
quadFit <- lm(temp ~  time + I(time^2))
sigmaNoise = sd(quadFit$residuals)
plot(time,temp, pch=20)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
k_20_100 <- SEkernel(sigmaf = 20, ell = 100)
GPfit <- gausspr(time, temp, kernel = k_20_100 , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE)
meanPred1 <- predict(GPfit, time)
lines(time, meanPred1, col="red", lwd = 2)
lines(time, meanPred1+1.96*predict(GPfit,time, type="sdeviation"),col="blue")
lines(time, meanPred1-1.96*predict(GPfit,time, type="sdeviation"),col="blue")
```



TO DO: play around with different values of sigmaf and l.

2.3)

Repeat the previous exercise, but now use Algorithm 2.1 on page 19 of Rasmussen and Willams' book to compute the posterior mean and variance of f .

```
quadFit <- lm(temp ~  time + I(time^2))
sigmaNoise = sd(quadFit$residuals)
plot(time,temp, pch=20)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
k_20_100 <- SEkernel(sigmaf = 20, ell = 100)
GPfit <- gausspr(time,temp, kernel = k_20_100 , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE)
res = posteriorGP(X = time, y = temp, Xstar, sigmaNoise = sigmaNoise, k = k_20_100)
meanPred = res[1]
varPred = res[2]
meanPred <- predict(GPfit, time)
lines(time, meanPred, col="red", lwd = 2)
lines(time, meanPred+1.96*varPred,col="blue")
lines(time, meanPred-1.96*varPred,col="blue")
```

2.4)

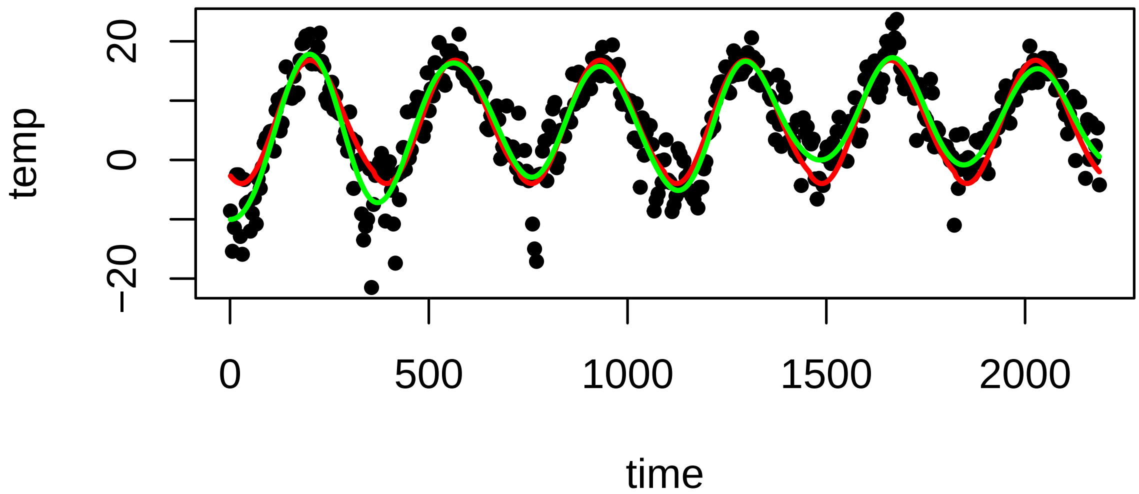
Consider now the following model:

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

Estimate the model using the `gausspr` function with the squared exponential function from (1) with $\sigma_f = 20$ and $l = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these σ_f and l values are not suitable). 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.

```
quadFit <- lm(temp ~  day + I(day^2))
sigmaNoise = sd(quadFit$residuals)
plot(time,temp, pch=20)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
k_20_100 <- SEkernel(sigmaf = 20, ell = 100)
GPfit <- gausspr(day, temp, kernel = k_20_100 , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE)
meanPred2 <- predict(GPfit, day)
lines(time, meanPred2, col="red", lwd = 2)
lines(time, meanPred1, col="green", lwd = 2)
```




```
#lines(time, meanPred2+1.96*predict(GPfit,day, type="sdeviation"),col="blue")
#lines(time, meanPred2-1.96*predict(GPfit,day, type="sdeviation"),col="blue")
```

Compare the results of both models. What are the pros and cons of each model?

2.5)

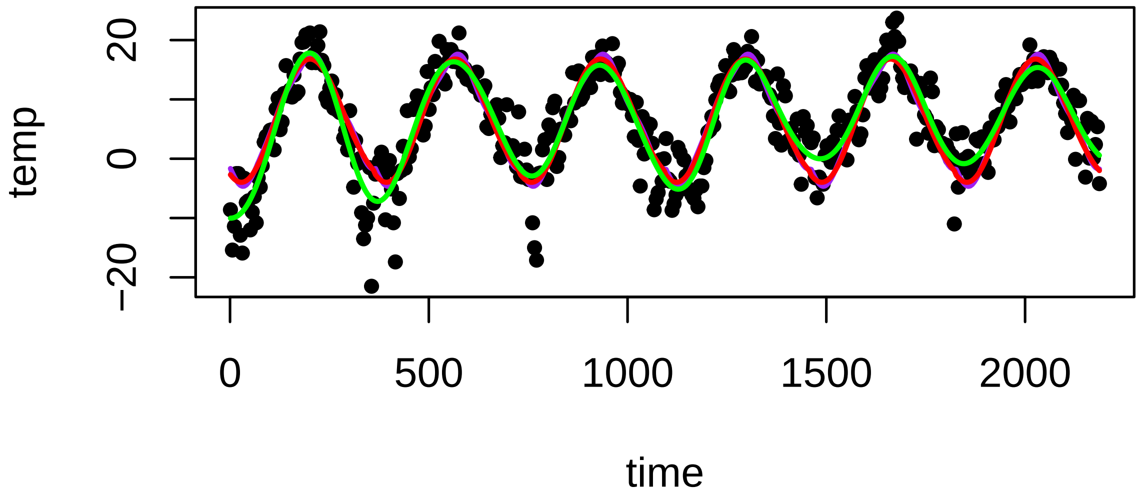
Finally, implement the following extension of the squared exponential kernel with a periodic kernel (a.k.a. locally periodic kernel):

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

Note that we have two different length scales in the kernel. Intuitively, l_1 controls the correlation between two days in the same year, 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 $\sigma_f = 20$, $l_1 = 1$, $l_2 = 100$ and $d = 365$. Use the `gausspr` function with the option `scaled=FALSE`, otherwise these σ_f , l_1 and l_2 values are not suitable. Compare the fit to the previous two models (with $\sigma_f = 20$ and $l = 100$). Discuss the results.

```
SEkernelPeriodic <- function(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365)
{
  rval <- function(x, y = NULL) {
      r = sqrt(crossprod(x-y));
      return(sigmaf**2 * exp(-(2*sin(pi*abs(x-y)/d)**2)/ell1**2) * exp(-0.5*(abs(x-y)**2)/ell2**2))
    }
  class(rval) <- "kernel"
  return(rval)
}
```

```r
plot(time,temp, pch=20)
k_periodic <- SEkernelPeriodic()
GPfit <- gausspr(day, temp, kernel = k_periodic , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE
meanPred3 <- predict(GPfit, day)
lines(time, meanPred3, col="purple", lwd = 2)
lines(time, meanPred2, col="red", lwd = 2)
lines(time, meanPred1, col="green", lwd = 2)
```



Discussion of the results:

3. GP Classification with kernlab.

Download the banknote fraud data:

```r
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])
```

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):

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

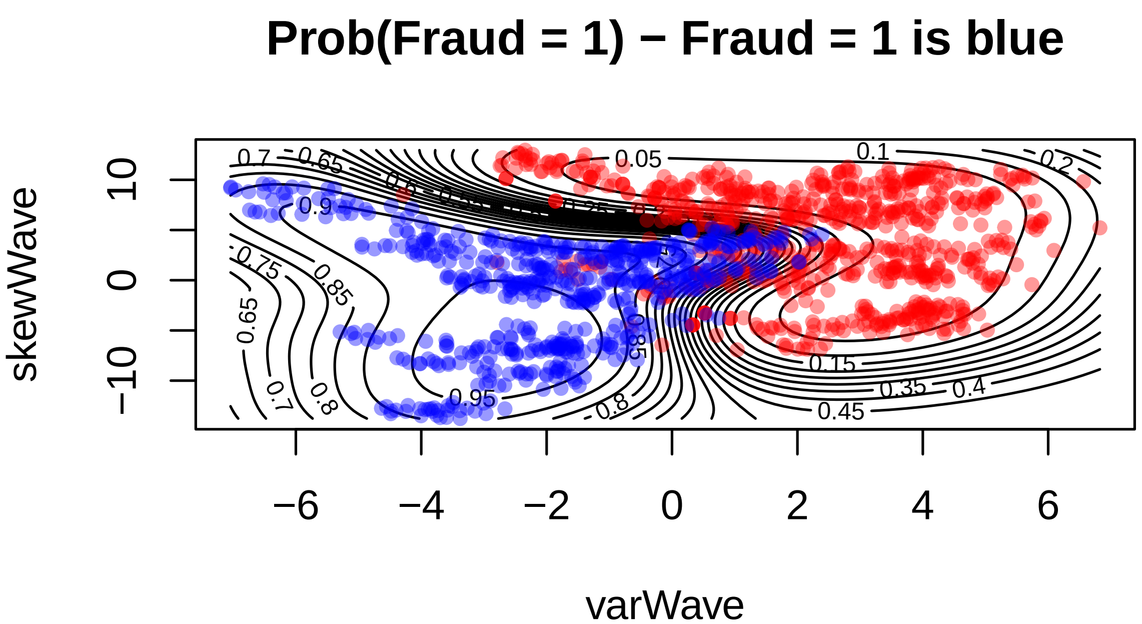
3.1)

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 `KernLabDemo.R` available on the course website. Compute the confusion matrix for the classifier and its accuracy.

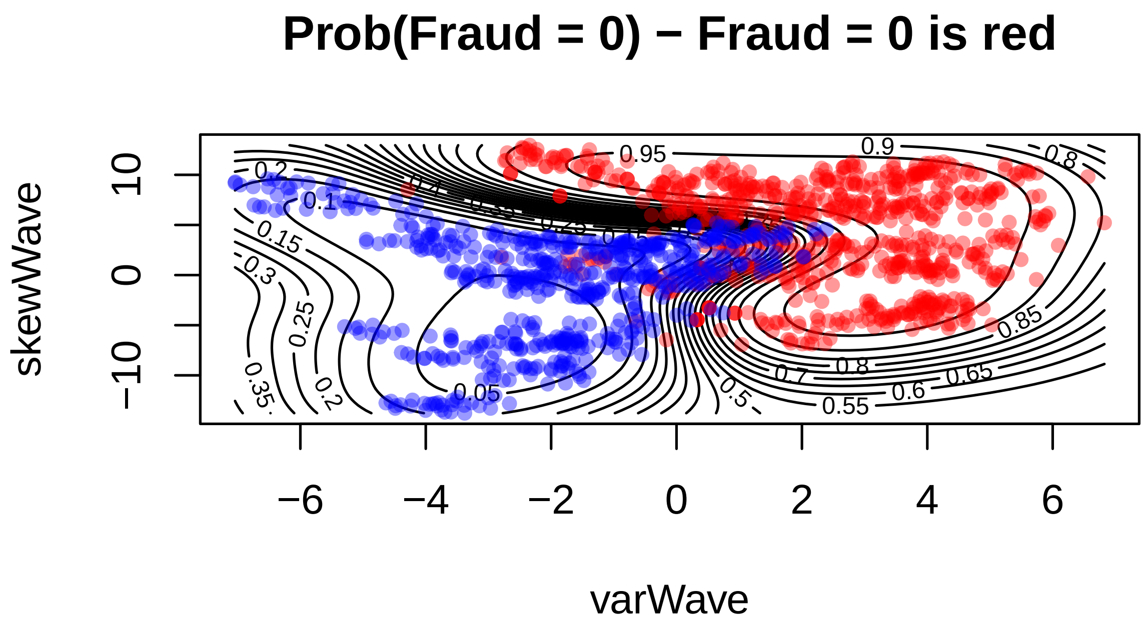
```
library(AtmRay)
library(scales)

GPfitfraud <- gausspr(fraud ~  varWave + skewWave , data=data[SelectTraining,])
```

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

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

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

# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud = 1) - Fraud = 1 is blue')
points(data[data[,5]==0,1],data[data[,5]=='0',2],col=alpha("red",0.4), pch=20) # Not fraud
points(data[data[,5]==1,1],data[data[,5]=='1',2],col=alpha("blue",0.4), pch=20) #Fraud
```



```r
# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud = 0) - Fraud = 0 is red')
points(data[data[,5]==0,1],data[data[,5]=='0',2],col=alpha("red",0.4), pch=20) # Not fraud
points(data[data[,5]==1,1],data[data[,5]=='1',2],col=alpha("blue",0.4), pch=20) #Fraud
```



3.2)

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

3.3)

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