Lab4 - Gaussian processes

Thijs Quast (thiqu264)

10/15/2019

Contents

Question 2.1 Implementing GP Regression.

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

$$y = f(x) + \epsilon, \epsilon \sim N(0, \sigma^2)$$

$$and$$

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

You must implement Algorithm 2.1 on page 19 of Rasmussen and Willams' book. The algo- rithm 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 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.

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.

Here is what you need to do:

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*. - hyperParam: Vector with two elements, sigmaf and l. - sigmaNoise: Noise standard deviation sigman.

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

```r
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=0.3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}
```

```r
# Own solution
posteriorGP <- function(X, y, XStar, hyperParam, sigmaNoise){
  K <- SquaredExpKernel(x1 = X, x2 = X, sigmaF = hyperParam[1], l = hyperParam[2])
  L <- t(chol(K + sigmaNoise^2*diag(length(X))))
  alpha <- solve(t(L), solve(L, y))

  kstar <- SquaredExpKernel(x1 = X, x2 = XStar, sigmaF = hyperParam[1], l = hyperParam[2])
  fstar <- t(kstar)%*%alpha

  v <- solve(L, kstar)
  Vfstar <- SquaredExpKernel(x1 = XStar, x2 = XStar, sigmaF = hyperParam[1], l = hyperParam[2]) - t(v)%

  #log_mar_ll <- -(0.5) %*% t(y) %*% alpha - sum(L) - (length(X)/2)*log(2*pi)

  outcome <- list("fstar" = fstar, "Vfstar" = Vfstar)

  return(outcome)
}
```

2

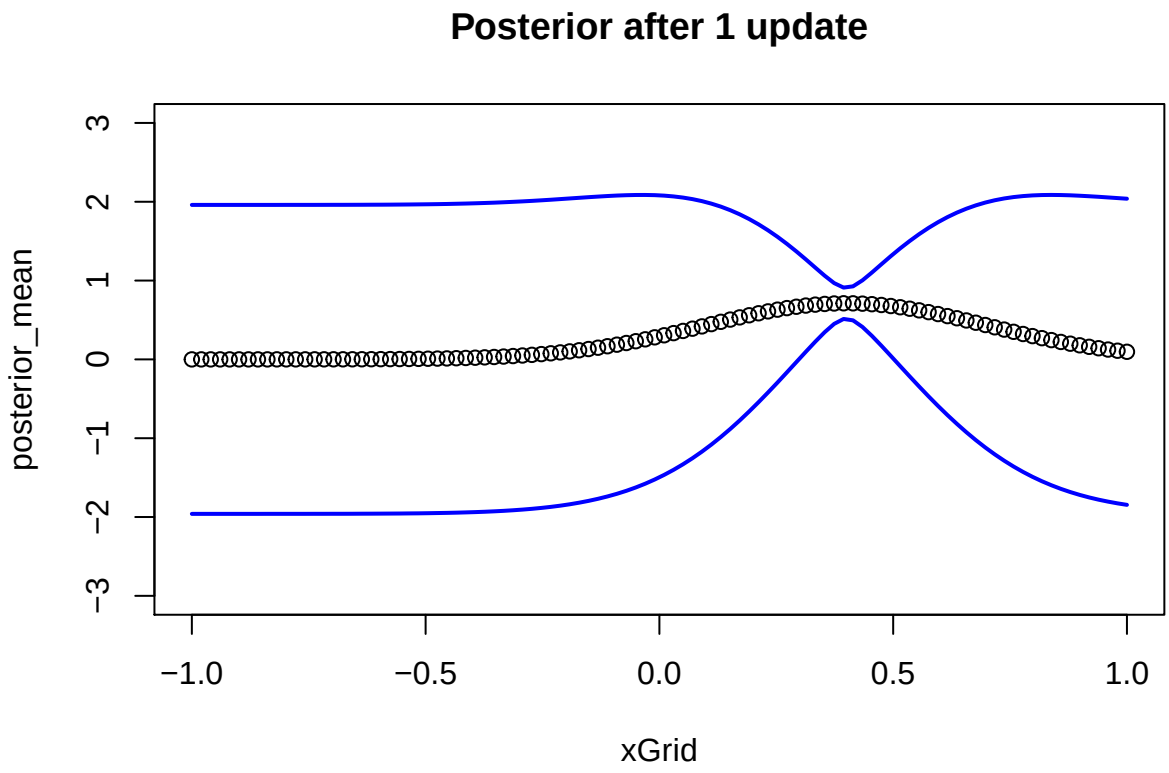
Now, let the prior hyperparameters be sigmaf = 1 and l = 0.3. Update this prior with a single observation: (x, y) = (0.4, 0.719). Assume that sigman = 0.1. Plot the posterior mean of f over the interval x : [-1, 1]. Plot also 95 % probability (pointwise) bands for f.

```r
hyperparameters <- c(1, 0.3)
x <- 0.4
y <- 0.719
sigmaN <- 0.1
xGrid <- seq(-1,1,length=100)
```

```r
update1 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigmaN)
```

```r
posterior_mean <- update1$fstar
posterior_variance <- diag(update1$Vfstar)

# Plot posterior mean
plot(xGrid, posterior_mean, type="p", ylim = c(-3,3))

# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
```

```
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
title("Posterior after 1 update")
```



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 : [-1, 1]. Plot also 95 % probability (point- wise) bands for f.

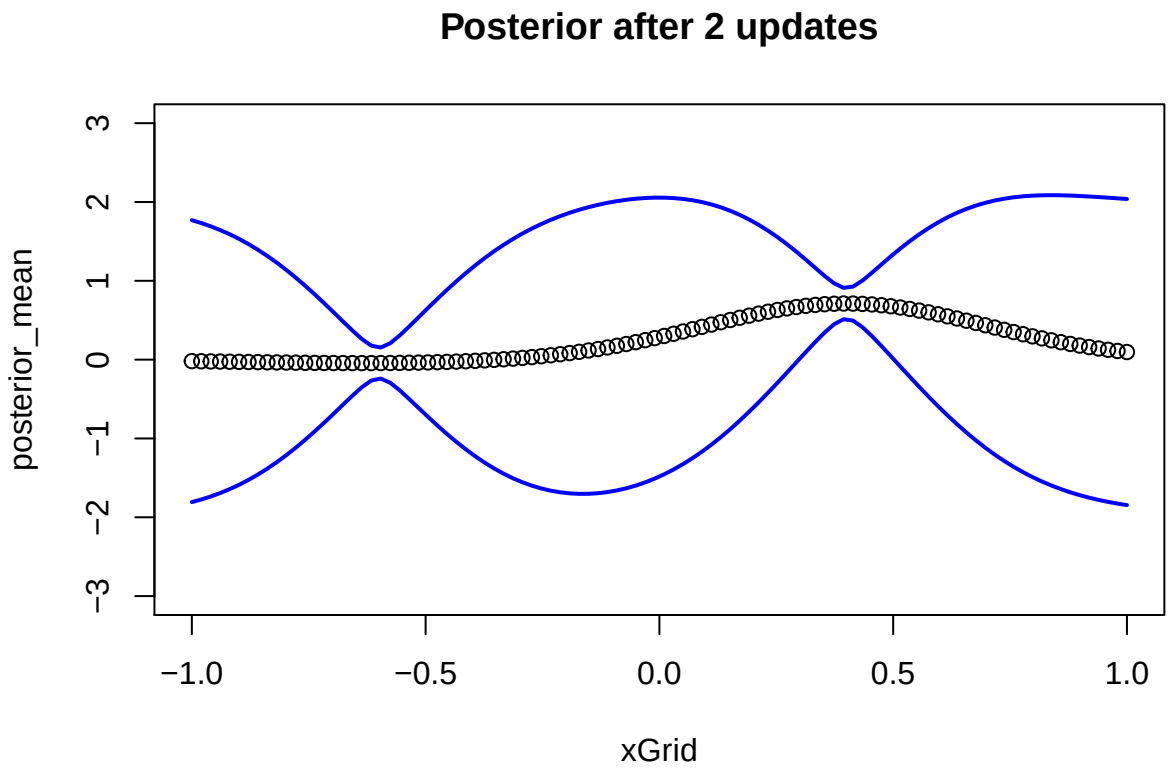
Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations.

```
hyperparameters <- c(1, 0.3)
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
sigmaN <- 0.1
xGrid <- seq(-1,1,length=100)
```

```
update2 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigmaN)
```

```
posterior_mean <- update2$fstar
posterior_variance <- diag(update2$Vfstar)

# Plot posterior mean
plot(xGrid, posterior_mean, type="p", ylim = c(-3,3))
```

```
# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
title("Posterior after 2 updates")
```



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 : [-1, 1]. Plot also 95 % probability (pointwise) bands for f.

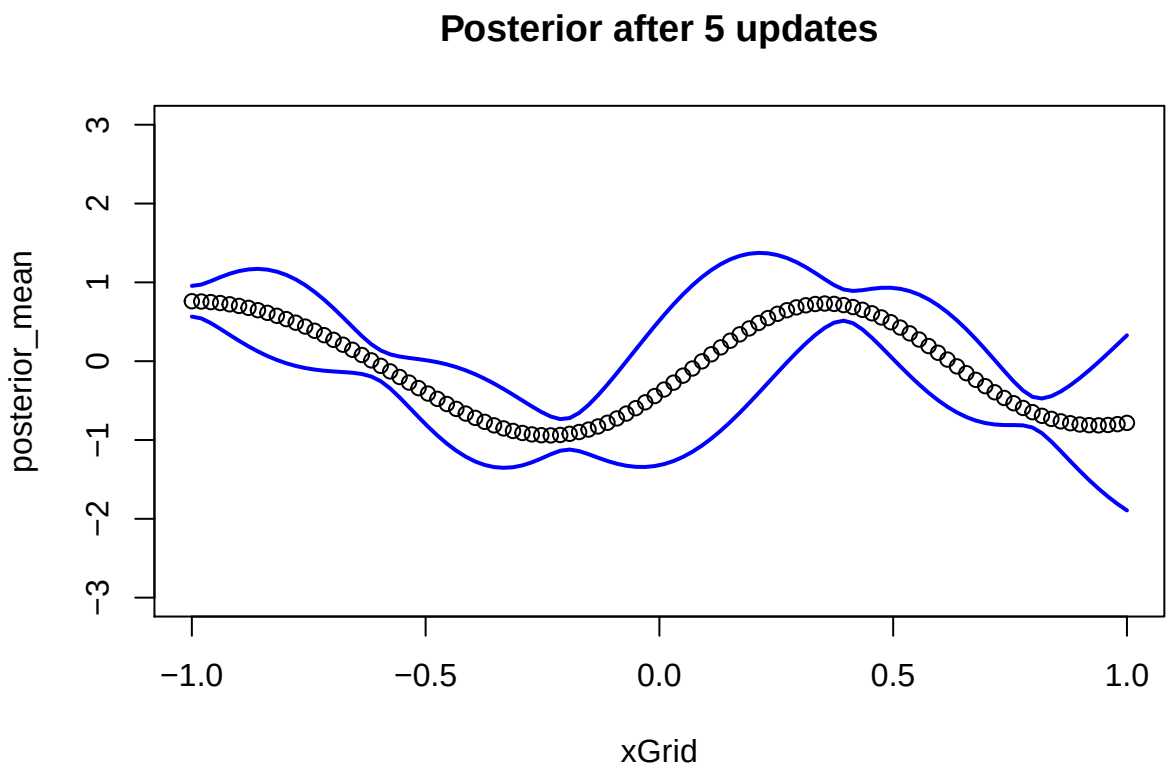
x [-1.0, -0.6, -0.2, 0.4 0.8] y [0.768, -0.044, -0.940, 0.719, -0.664]

```
hyperparameters <- c(1, 0.3)
x <- c(-1.0 ,-0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
sigmaN <- 0.1
xGrid <- seq(-1,1,length=100)
```

```
update5 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigmaN)
```

```
posterior_mean <- update5$fstar
posterior_variance <- diag(update5$Vfstar)
```

```r
# Plot posterior mean
plot(xGrid, posterior_mean, type="p", ylim = c(-3,3))

# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
title("Posterior after 5 updates")
```



5

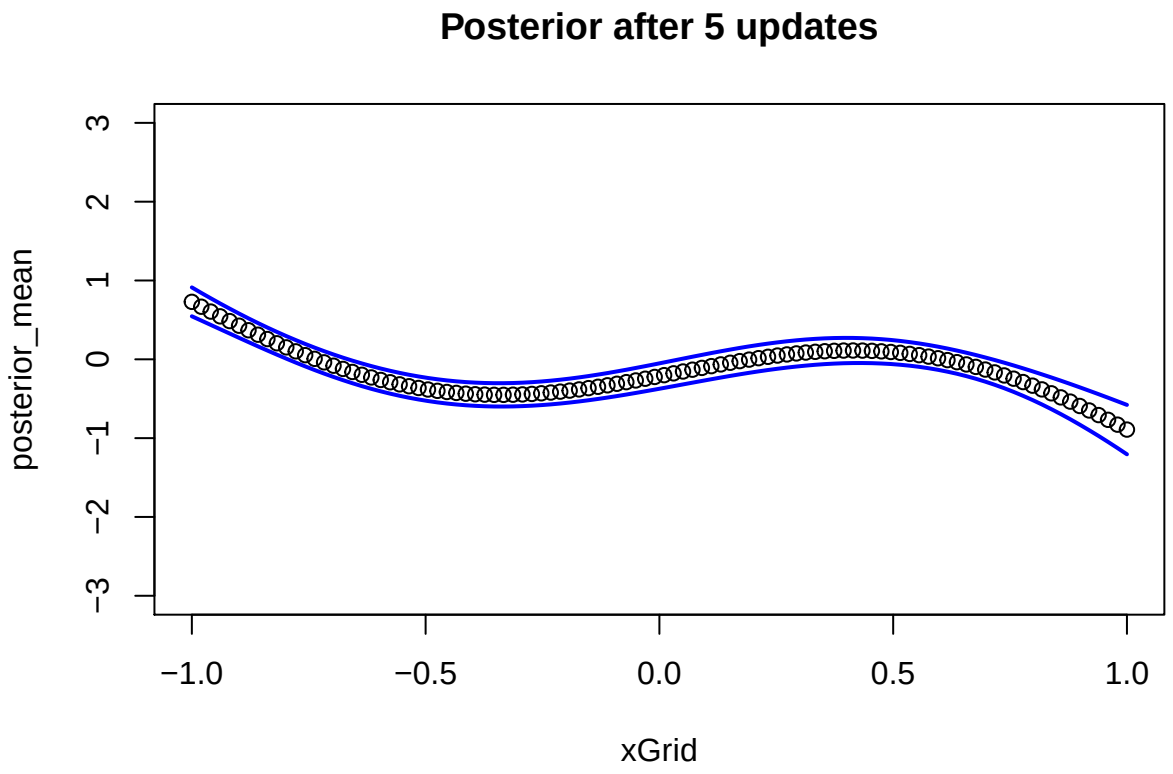
Repeat (4), this time with hyperparameters sigmaf = 1 and l = 1. Compare the results.

```r
hyperparameters <- c(1, 1)
x <- c(-1.0 ,-0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
sigmaN <- 0.1
xGrid <- seq(-1,1,length=100)
```

```r
update5 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigmaN)
```

```r
posterior_mean <- update5$fstar
posterior_variance <- diag(update5$Vfstar)
```

```r
# Plot posterior mean
plot(xGrid, posterior_mean, type="p", ylim = c(-3,3))

# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
title("Posterior after 5 updates")
```



By setting the increasing the l-parameter the posterior of f becomes more smoother.

Question 2.2 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.

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.

```r
# You can read the dataset with the command:
temperature <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTu
```

```
temperature$time <- c(1:nrow(temperature))
temperature$day <- rep(c(1:365))
```

```
index <- seq(from=1, to=nrow(temperature), by = 5)
temperature <- temperature[index,]
```

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 sigmaf (sigmaf)), evaluate it in the point x = 1, x' = 2, and use the kernelMatrix function to compute the covariance matrix K(X, Xstar) for the input vectors X = (1, 3, 4)T and Xstar = (2,3,4)T .

```
library(kernlab)
# SE kernel, so that it can go into kernelmatrix function
SEkernel <- function(sigmaf=1,ell=0.3)
  {
    rval <- function(x1, x2){
      n1 <- length(x1)
      n2 <- length(x2)
      K <- matrix(NA,n1,n2)
      for (i in 1:n2){
        K[,i] <- sigmaf^2*exp(-0.5*( (x1-x2[i])/ell)^2 )
      }
      return(K)
      }
    class(rval) <- "kernel"
    return(rval)
}
```

```
kernel_function <- SEkernel()
kernel_function(x1 = 1, x2 = 2)
```

```
##             [,1]
## [1,] 0.00386592
```

```
X <- c(1,3,4)
Xstar <- c(2,3,4)

# K is K(X, Xstar)
K <- kernelMatrix(kernel = kernel_function, x = X, y = Xstar)
```

2

Consider first the following model:

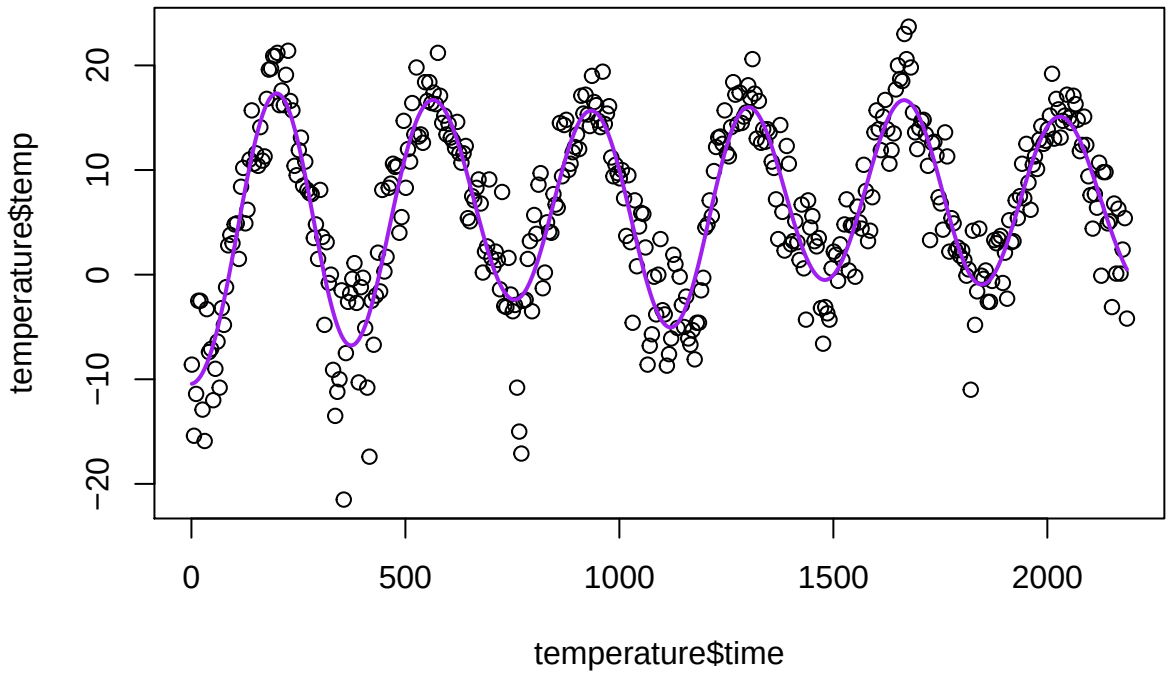
$$temp = f(time) + \epsilon$$
$$with \epsilon \sim N(0, \sigma n2)$$

$$andf \sim GP(0, k(time, time'))$$

Let sigmaN2 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 sigmaf = 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 sigmaf and l (no need to write this in the report though).

```
polyFit <- lm(temperature$temp ~ temperature$time + I(temperature$time^2))
sigmaNoise = sd(polyFit$residuals)
```

```
# Fit the GP with home made SE Kernel
sigmaf <- 20
ell <- 0.2
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1)) # NOTE: this also works and is the sam

GPfit <- gausspr(temperature$time, temperature$temp, kernel = SEkernel, kpar = list(sigmaf = sigmaf, el
meanPred <- predict(GPfit, temperature$time)
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="purple", lwd = 2)
```



3

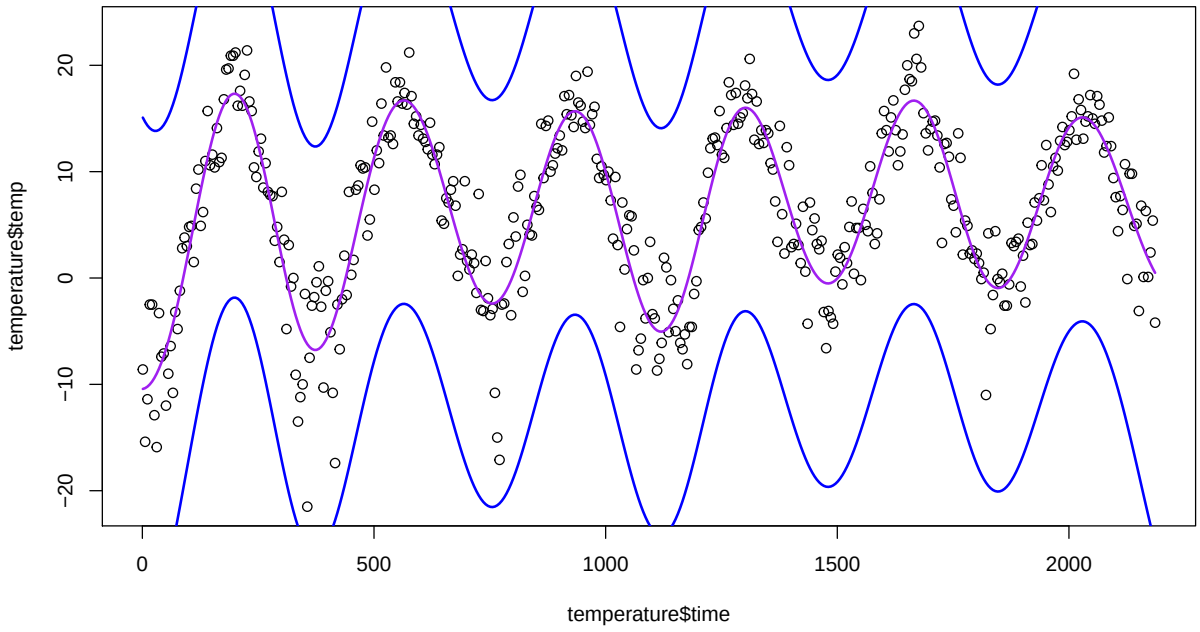
kernlab can compute the posterior variance of f, but it seems to be a bug in the code. So, do your own computations for the posterior variance of f and plot the 95 % probability (pointwise) bands for f. Superimpose

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.

```
polyFit <- lm(temperature$temp ~ scale(temperature$time) + scale(I(temperature$time^2)))
sigmaNoise = sd(polyFit$residuals)
```

```
XStar <- seq(1, length(temperature$time), by = 1)
hyperparameters <- c(20, 0.2)
posterior <- posteriorGP(X = scale(temperature$time), y = temperature$temp,
            XStar = scale(temperature$time), hyperParam = hyperparameters,
            sigmaNoise = sigmaNoise^2)

posterior_variance <- diag(posterior$Vfstar)
```

```
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="purple", lwd = 2)
lines(temperature$time, meanPred + 1.96*sqrt(posterior_variance), col="blue", lwd=2)
lines(temperature$time, meanPred - 1.96*sqrt(posterior_variance), col="blue", lwd=2)
```



4

Consider now the following model:

$$temp = f(day) + \epsilon$$

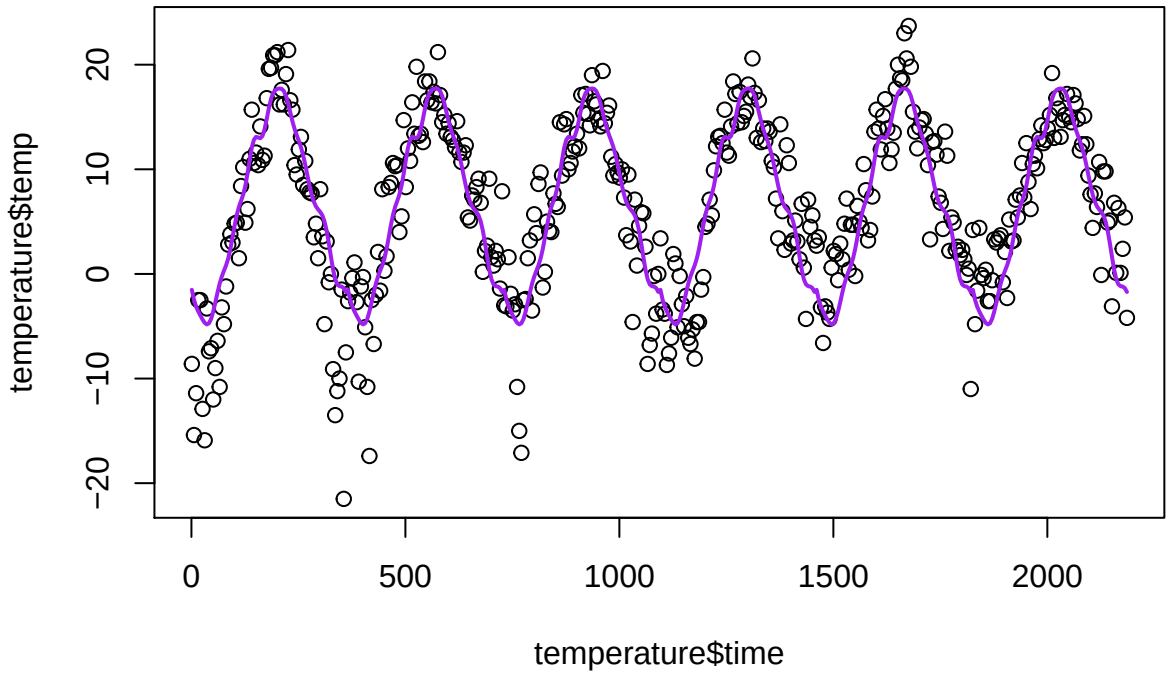
$$with\epsilon \sim N(0, \sigma n2)$$

$$andf \sim GP(0, k(day, daystar))$$

Estimate the model using the squared exponential function with sigmaf = 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?

```
polyFit <- lm(temperature$temp ~ temperature$day + I(temperature$day^2))
sigmaNoise = sd(polyFit$residuals)
```

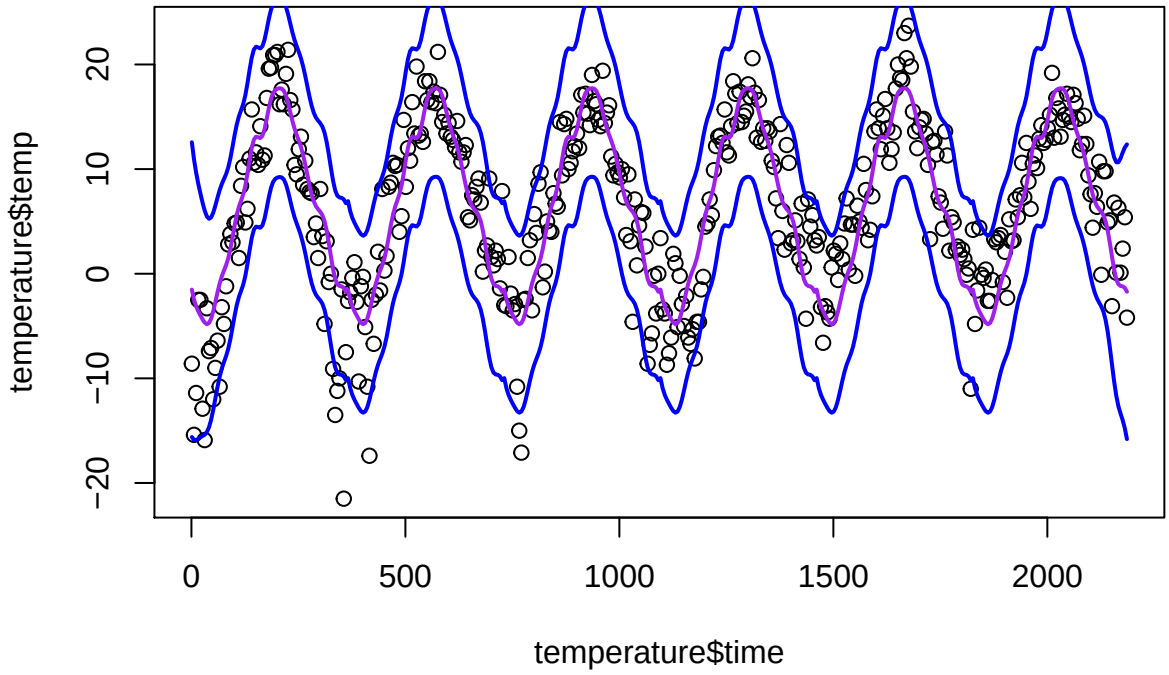
```
# Fit the GP with home made SE Kernel
sigmaf <- 20
ell <- 0.2
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1)) # NOTE: this also works and is the sam

GPfit <- gausspr(temperature$day, temperature$temp, kernel = SEkernel, kpar = list(sigmaf = sigmaf, ell=
meanPred <- predict(GPfit, temperature$day)
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="purple", lwd = 2)
```




```
XStar <- seq(1, length(temperature$time), by = 1)
hyperparameters <- c(20, 0.2)
```

```
posterior <- posteriorGP(X = scale(temperature$day), y = scale(temperature$temp),
            XStar = scale(XStar), hyperParam = hyperparameters,
            sigmaNoise = sigmaNoise^2)

posterior_variance <- diag(posterior$Vfstar)
```

```
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="purple", lwd = 2)
lines(temperature$time, meanPred + 1.96*sqrt(posterior_variance), col="blue", lwd=2)
lines(temperature$time, meanPred - 1.96*sqrt(posterior_variance), col="blue", lwd=2)
```

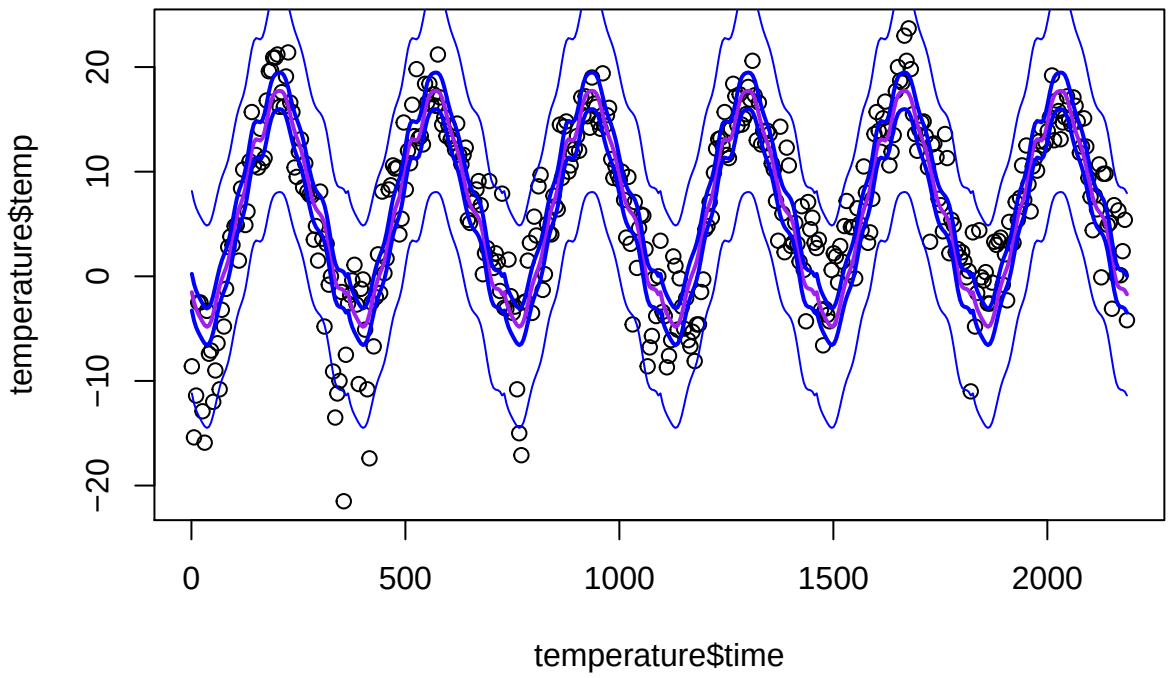



```
# Altrenatively,
polyFit <- lm(temperature$temp ~ temperature$day + I(temperature$day^2))
sigmaNoise = sd(polyFit$residuals)
```

```
# SE kernel, so that it can go into kernelmatrix function
SEkernel <- function(sigmaf=1,ell=0.3)
  {
    rval <- function(x1, x2){
      n1 <- length(x1)
      n2 <- length(x2)
      K <- matrix(NA,n1,n2)
      for (i in 1:n2){
        K[,i] <- sigmaf^2*exp(-0.5*( (x1-x2[i])/ell)^2 )
```

```
      }
      return(K)
      }
    class(rval) <- "kernel"
    return(rval)
}
```

```
# Fit the GP with home made SE Kernel
sigmaf <- 20
ell <- 0.2
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1)) # NOTE: this also works and is the sam

GPfit <- gausspr(temperature$day, temperature$temp, kernel = SEkernel, kpar = list(sigmaf = sigmaf, ell=
meanPred <- predict(GPfit, temperature$day)
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="purple", lwd = 2)

SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note the reparametrization

# Probability and prediction interval implementation.
x<-temperature$day
xs<-temperature$day # XStar
n <- length(x)
Kss <- kernelMatrix(kernel = SEkernel, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = SEkernel, x = x, y = x)
Kxs <- kernelMatrix(kernel = SEkernel, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs) # Covariance matrix of fStar

# Probability intervals for fStar
lines(temperature$time, meanPred - 1.96*sqrt(diag(Covf)), col = "blue", lwd = 2)
lines(temperature$time, meanPred + 1.96*sqrt(diag(Covf)), col = "blue", lwd = 2)

# Prediction intervals for yStar
lines(temperature$time, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "blue")
lines(temperature$time, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "blue")
```



Regression results on time seem to show better, more smooth results than on day.

5

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

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

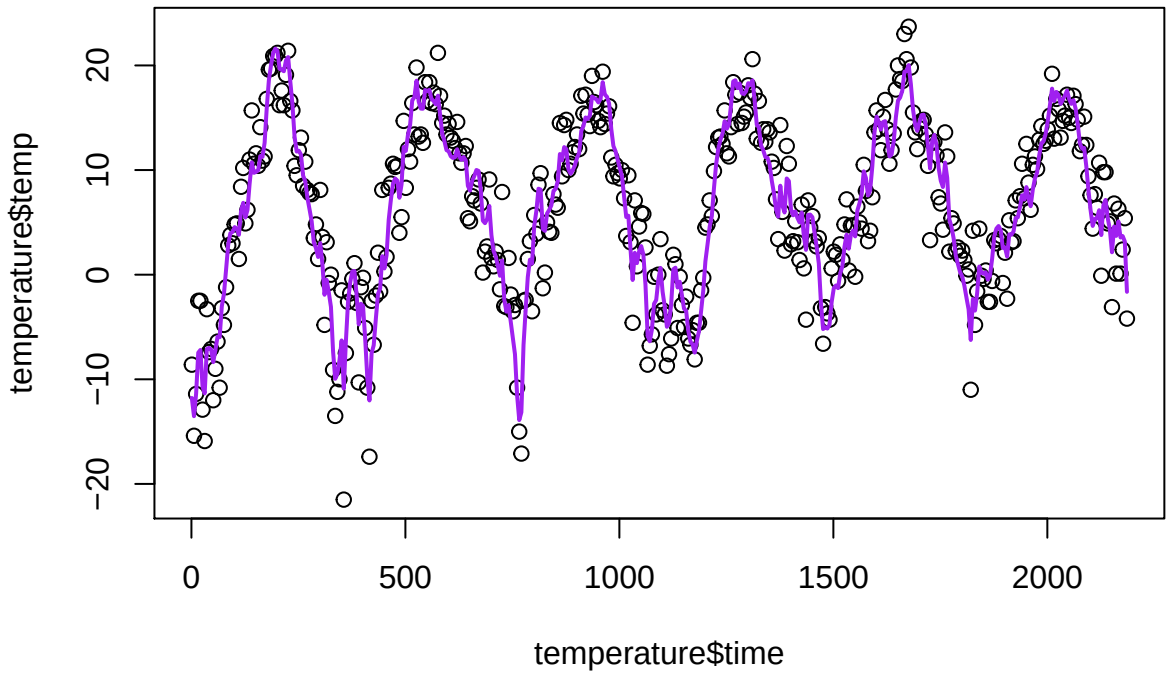
Note that we have two different length scales here, and l2 controls the correlation be- tween the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters sigmaf = 20, l1 = 1, l2 = 10 and d = 365/sd(time). The reason for the rather strange period here is that kernlab standardizes the in- puts to have standard deviation of 1. Compare the fit to the previous two models (with sigmaf = 20 and l = 0.2). Discuss the results.

```
# SE kernel, so that it can go into kernelmatrix function
Periodickernel <- function(sigmaf,ell1, ell2, d)
  {
    rval <- function(x1, x2){
      n1 <- length(x1)
      n2 <- length(x2)
      K <- matrix(NA,n1,n2)
      for (i in 1:n2){
        numerator1 <- 2*(sin(pi*abs(x1-x2[i])/d))
        numerator2 <- abs(x1-x2[i])^2
```

```
        K[,i] <- sigmaf^2*exp(-numerator1/(ell1^2))*exp(-0.5*(numerator2)/ell2^2)
      }
      return(K)
      }
    class(rval) <- "kernel"
    return(rval)
}
```

```
polyFit <- lm(temperature$temp ~ temperature$time + I(temperature$time^2))
sigmaNoise = sd(polyFit$residuals)
```

```
# Fit the GP with home made Periodic Kernel
sigmaf <- 20
ell1 <- 1
ell2 <- 10
d <- 365/sd(temperature$time)
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1)) # NOTE: this also works and is the sam

GPfit <- gausspr(temperature$time, temperature$temp, kernel = Periodickernel, kpar = list(sigmaf = sigma

meanPred <- predict(GPfit, temperature$time)
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="purple", lwd = 2)
```



This model seems to overfit. Probably due to the fact that we we have multiple parameters we can specify in

the periodic kernel. I'd say the Squared Exponential Kernel is more suitable as it provides a more smoother fit through the data.

Question 2.3 - GP Classification with kernlab.

Download the banknote fraud data:

```
library(AtmRay)
```

```
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])

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

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 observa- tions):

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.

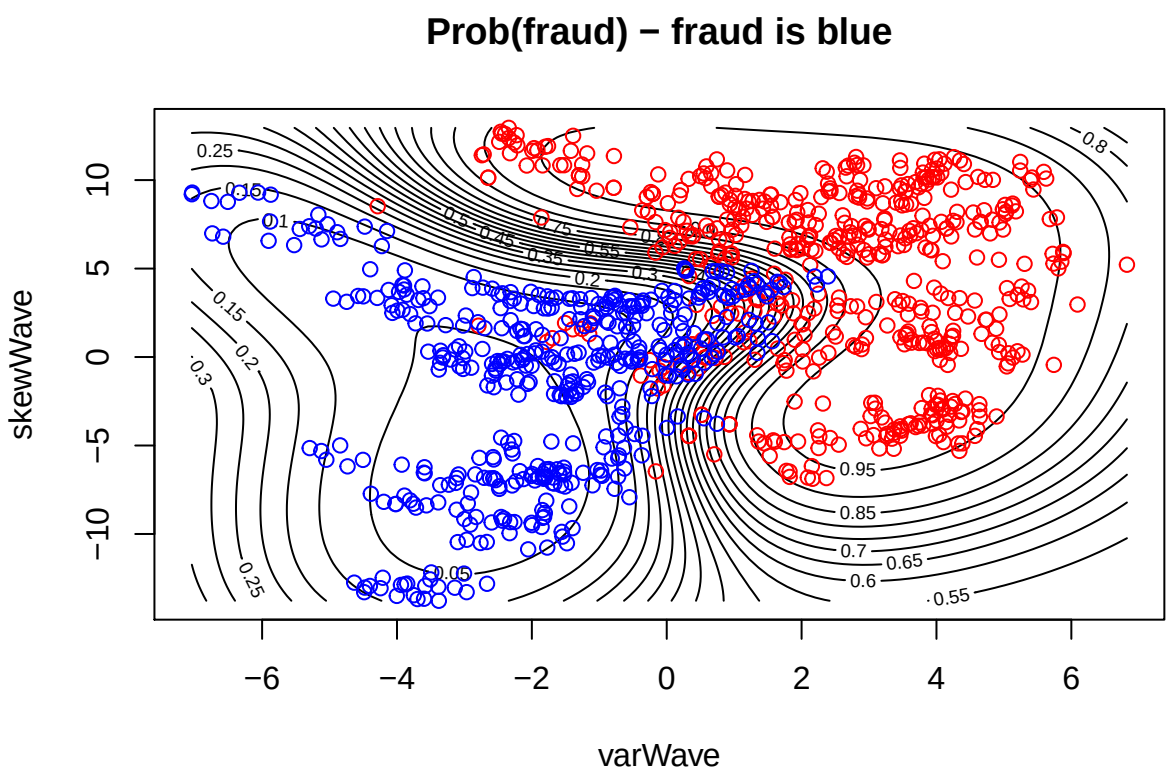
```
set.seed(111)
GPfit <- gausspr(fraud ~ varWave + skewWave, data=Training)
```

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

```
prediction <- predict(GPfit, Training[,1:2])
```

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

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

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




```
# predict on the training set
confusion <- table(prediction, Training[,5]) # confusion matrix
accuracy <- (confusion[1,1] + confusion[2,2])/sum(confusion)

confusion
```

```
##
## prediction   0   1
##          0 503  21
##          1  41 435
```

```
accuracy
```

```
## [1] 0.938
```

2

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

```r
# predict on the testset
prediction <- predict(GPfit,Test[,1:2])
confusion <- table(predict(GPfit, Test[,1:2]), Test[,5]) # confusion matrix

accuracy <- (confusion[1,1] + confusion[2,2])/sum(confusion)

confusion
```

```
##
##       0   1
##   0 199   9
##   1  19 145
```

```r
accuracy
```

```
## [1] 0.9247312
```

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.

```r
GPfit <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=Training)
```

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

```r
GPfit
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.449772997200818
##
## Number of training instances learned : 1000
## Train error : 0.003
```

```r
# predict on the testset
prediction <- predict(GPfit,Test[,1:4])
confusion <- table(predict(GPfit, Test[,1:4]), Test[,5]) # confusion matrix

accuracy <- (confusion[1,1] + confusion[2,2])/sum(confusion)

confusion
```

```
##
##       0   1
##   0 216   0
##   1   2 154
```

```
accuracy
```

```
## [1] 0.9946237
```

By running the model and predicting it using 4 covariates, the model improves to an accuracy of above 99%.