TDDE15-Lab 4

Fredrik Ramberg

Part 2.1

Imports

```r
library("mvtnorm")
```

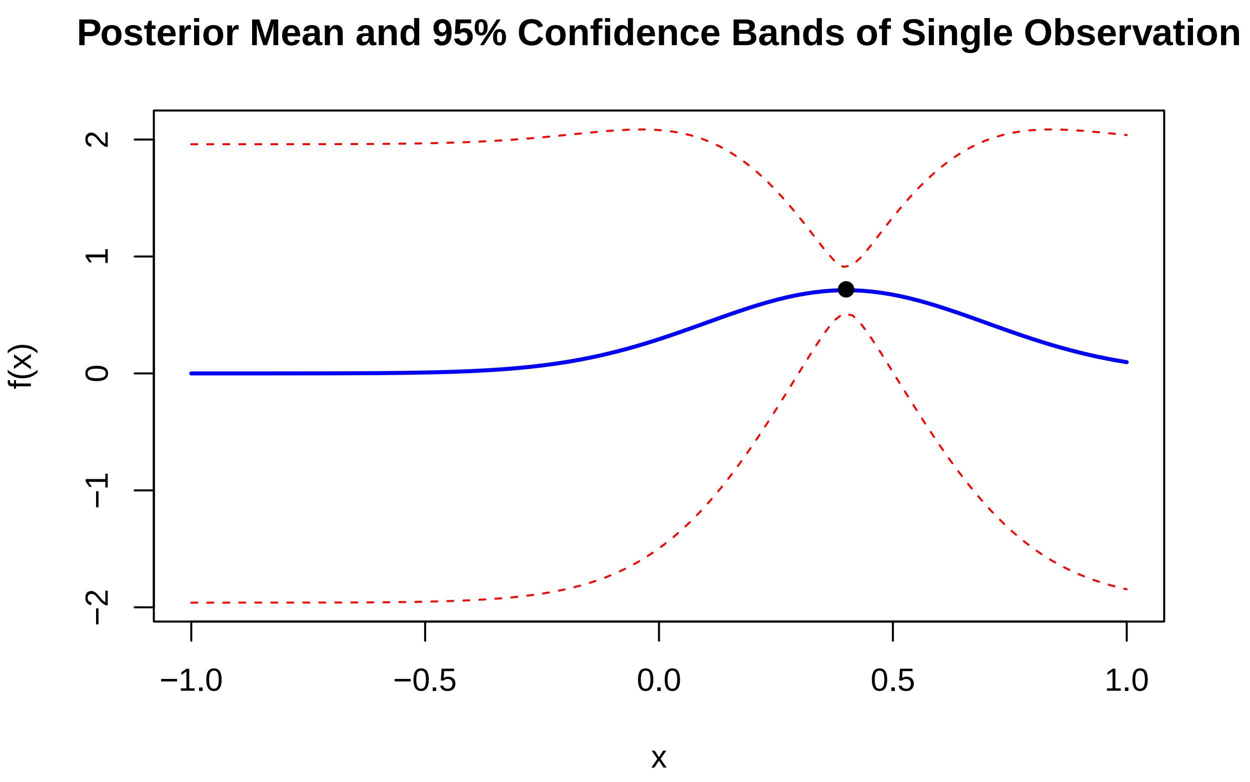
2.1. Implementing GP Regression.

This first exercise will have you writing your own code for the Gaussian process regression model: y = f (x) + *Epsilon* with *Epsilon* ~ N (0, sigmaˆ2 n) and f ~ GP(0, k(x, x'))

```
   input: $X$ (inputs), $\mathbf{y}$ (targets), $k$ (covariance function), $\sigma_n^2$ (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)
```

```r
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=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)
}

posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...){
  n <- length(X)
  K <- k(X, X, ...)
  kStar <- k(X,XStar, ...)

  #Cholesky
  L <- t(chol(K + sigmaNoise^2*diag(n)))

  #alpha
  alpha <- solve(t(L),solve(L,y))
  #Posterior mean = fStar
  kStar <- k(X, XStar, ...)
  fStar <- t(kStar)%*%alpha

  #Posterior variance
  v <- solve(L,kStar)
  variance <- k(XStar, XStar, ...) - t(v)%*%v

  return(list(mean =fStar, variance =variance))
}
```

```r
#Hyperparameters
sigmaF <- 1
l <- 0.3
sigmaNoise <- 0.1


#observation
obs <- data.frame(X = 0.4, Y = 0.719)

#test points
XStar <- seq(-1,1,length=100)

posterior <- posteriorGP(obs$X, obs$Y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)

# mean and posterior
posterior_mean <- posterior$mean
posterior_variance <- diag(posterior$variance)

# 95% confidence intervals
upper_bound <- posterior_mean + 1.96 * sqrt(posterior_variance)
lower_bound <- posterior_mean - 1.96 * sqrt(posterior_variance)

# Plot posterior mean and 95% confidence intervals
plot(XStar, posterior_mean, type = "l", col = "blue", lwd = 2,
     ylim = c(min(lower_bound), max(upper_bound)),
     xlab = "x", ylab = "f(x)", main = "Posterior Mean and 95% Confidence Bands of Single Observation")
lines(XStar, upper_bound, col = "red", lty = 2)
lines(XStar, lower_bound, col = "red", lty = 2)
points(obs$X, obs$Y, col = "black", pch = 19)  # Plot the training point
```

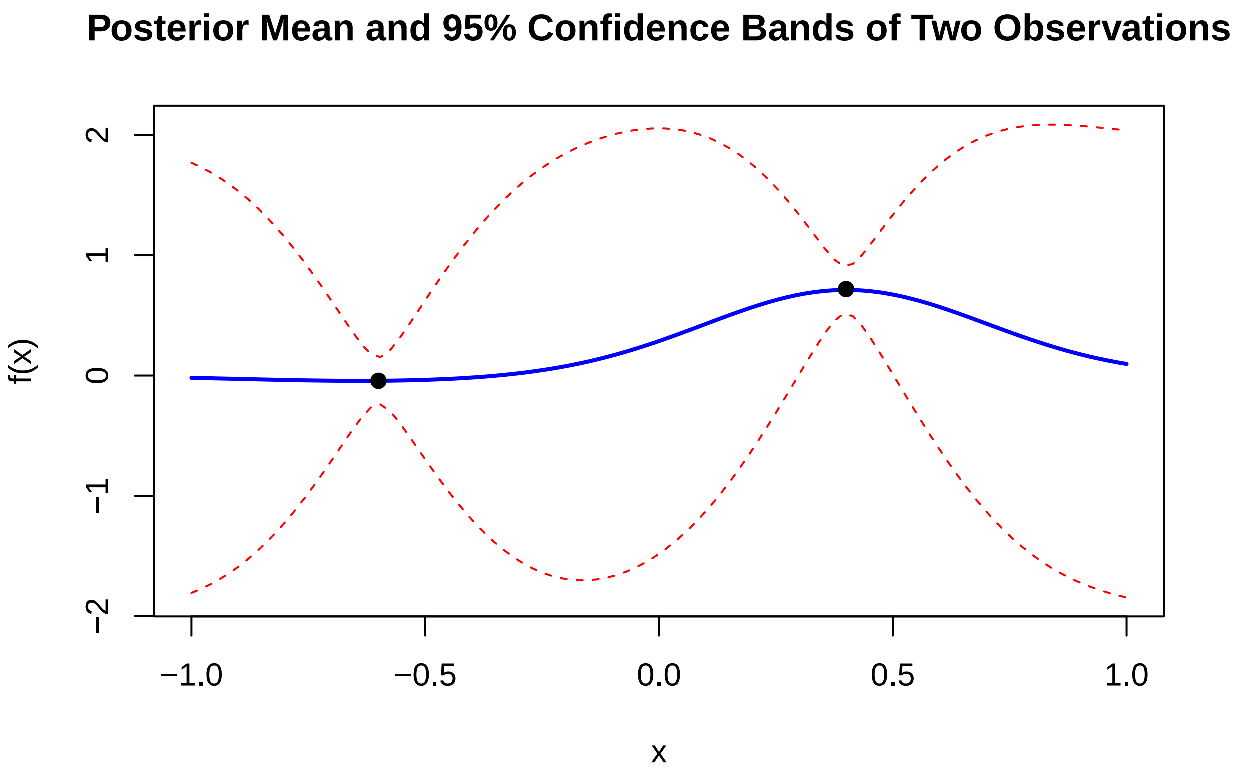


Answer: The confidence interval is the narrowest nearby the point which is expected with a distance decaying covariance.

(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 E [-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.

```r
#Calculating the posterior for two points
obs2 <- rbind(obs, data.frame(X= -0.6, Y = -0.044))
posterior <- posteriorGP(obs2$X, obs2$Y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)

# mean and posterior
posterior_mean <- posterior$mean
posterior_variance <- diag(posterior$variance)

# 95% confidence intervals
upper_bound <- posterior_mean + 1.96 * sqrt(posterior_variance)
lower_bound <- posterior_mean - 1.96 * sqrt(posterior_variance)

# Plot posterior mean and 95% confidence intervals
plot(XStar, posterior_mean, type = "l", col = "blue", lwd = 2,
     ylim = c(min(lower_bound), max(upper_bound)),
     xlab = "x", ylab = "f(x)", main = "Posterior Mean and 95% Confidence Bands of Two Observations")
lines(XStar, upper_bound, col = "red", lty = 2)
lines(XStar, lower_bound, col = "red", lty = 2)
points(obs2$X, obs2$Y, col = "black", pch = 19)  # Plot the training point
```

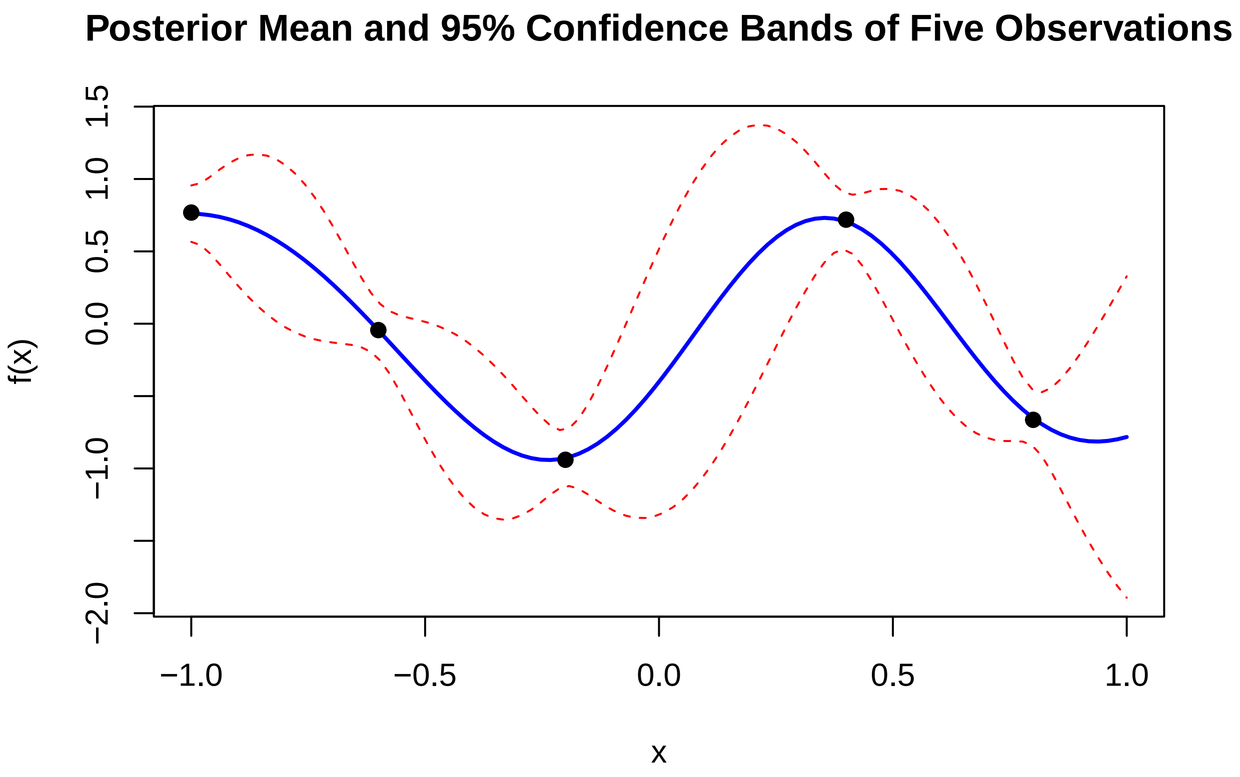


Now the the confidence intervals narrows at both of the points.

(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

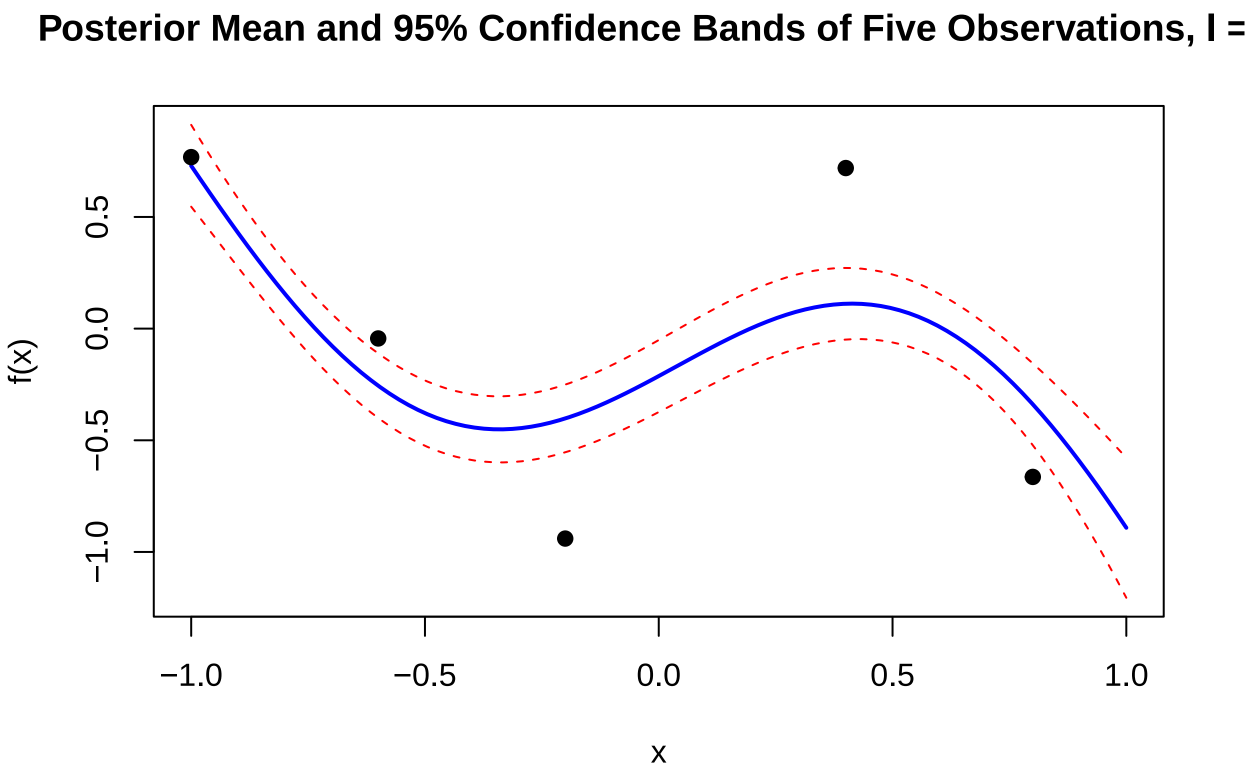
```r
#Calculating the posterior for all five points
obs5 <- data.frame(X = c(-1,-0.6,-0.2,0.4,0.8),
                   Y = c(0.768,-0.044,-0.940,0.719,-0.664))
posterior <- posteriorGP(obs5$X, obs5$Y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)

# mean and posterior
posterior_mean <- posterior$mean
posterior_variance <- diag(posterior$variance)

# 95% confidence intervals
upper_bound <- posterior_mean + 1.96 * sqrt(posterior_variance)
lower_bound <- posterior_mean - 1.96 * sqrt(posterior_variance)

# Plot posterior mean and 95% confidence intervals
plot(XStar, posterior_mean, type = "l", col = "blue", lwd = 2,
     ylim = c(min(lower_bound), max(upper_bound)),
     xlab = "x", ylab = "f(x)", main = "Posterior Mean and 95% Confidence Bands of Five Observations")
lines(XStar, upper_bound, col = "red", lty = 2)
lines(XStar, lower_bound, col = "red", lty = 2)
points(obs5$X, obs5$Y, col = "black", pch = 19)  # Plot the training point
```



Now the the confidence interval is smaller overall due to several points different from eachother effecting the results. We can also see a wave pattern emerging.

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

```r
#changing the hyperparameters, the rest is the same .....
posterior <- posteriorGP(obs5$X, obs5$Y, XStar, sigmaNoise, SquaredExpKernel, sigmaF = 1, l = 1)

# mean and posterior
posterior_mean <- posterior$mean
posterior_variance <- diag(posterior$variance)

# 95% confidence intervals
upper_bound <- posterior_mean + 1.96 * sqrt(posterior_variance)
lower_bound <- posterior_mean - 1.96 * sqrt(posterior_variance)

# Plot posterior mean and 95% confidence intervals
plot(XStar, posterior_mean, type = "l", col = "blue", lwd = 2,
     ylim = c(min(lower_bound), max(upper_bound)),
     xlab = "x", ylab = "f(x)", main = "Posterior Mean and 95% Confidence Bands of Five Observations, l
lines(XStar, upper_bound, col = "red", lty = 2)
lines(XStar, lower_bound, col = "red", lty = 2)
points(obs5$X, obs5$Y, col = "black", pch = 19)  # Plot the training point
```



A higher l will smooth out the function which is visible in the plot. This is due to minimizing dividing the distance with a lower number, which will lead values far from each other being "assigned" higher covariance. Here l clearly is to high resulting in underfitting the data.

Part 2.2

Imports and data structure

```
library(kernlab)

tempData <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulli
tempData <- cbind(tempData, time = 1:nrow(tempData))
tempData <- cbind(tempData, day = ((tempData$time-1)%%365)+1)

trainData <- subset(tempData, (time - 1)%%5 == 0)
```

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

```
#nested Square Exponetial Kernel
nestedSEK <- function(sigmaF=1,l=3) {
  fixedSEK <- function(x1,x2){
    n1 <- length(x1)
    n2 <- length(x2)
```

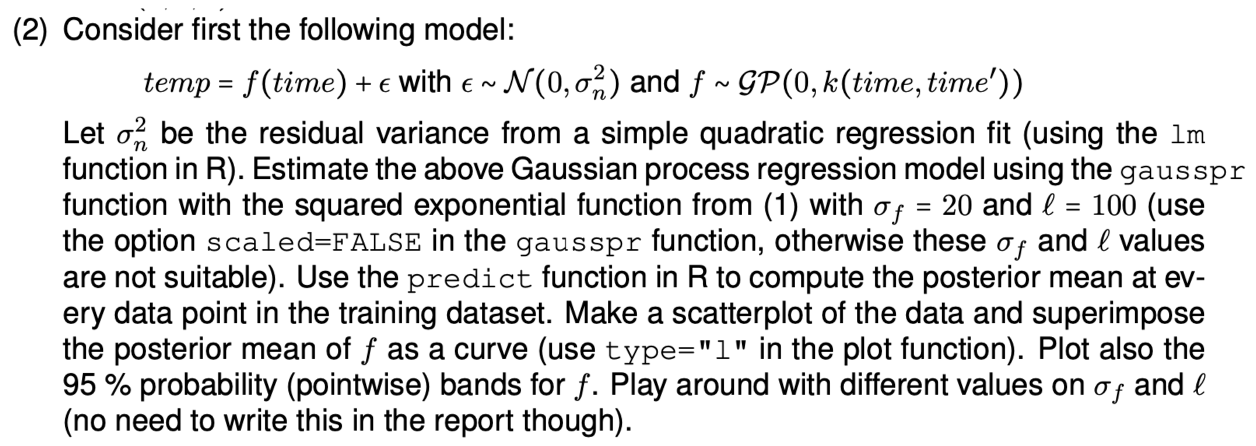
```r
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
    }
    return(K)
  }
  class(fixedSEK) <- 'kernel'
  return(fixedSEK)
}

SEK <- nestedSEK()

#testing kernal function for x=1, xstar=2
SEK(1,2)
```

```
##           [,1]
## [1,] 0.9459595
```

```r
# kernel matrix where x = X, y = Xstar
kernelMatrix(kernel = SEK, x = c(1,3,4), y =c(2,3,4))
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.9459595 0.8007374 0.6065307
## [2,] 0.9459595 1.0000000 0.9459595
## [3,] 0.8007374 0.9459595 1.0000000
```

Answer: We can see that SEK(1,2) is the same as positions (1,1) and (2,1) which is where the function is applied on 1 & 2.

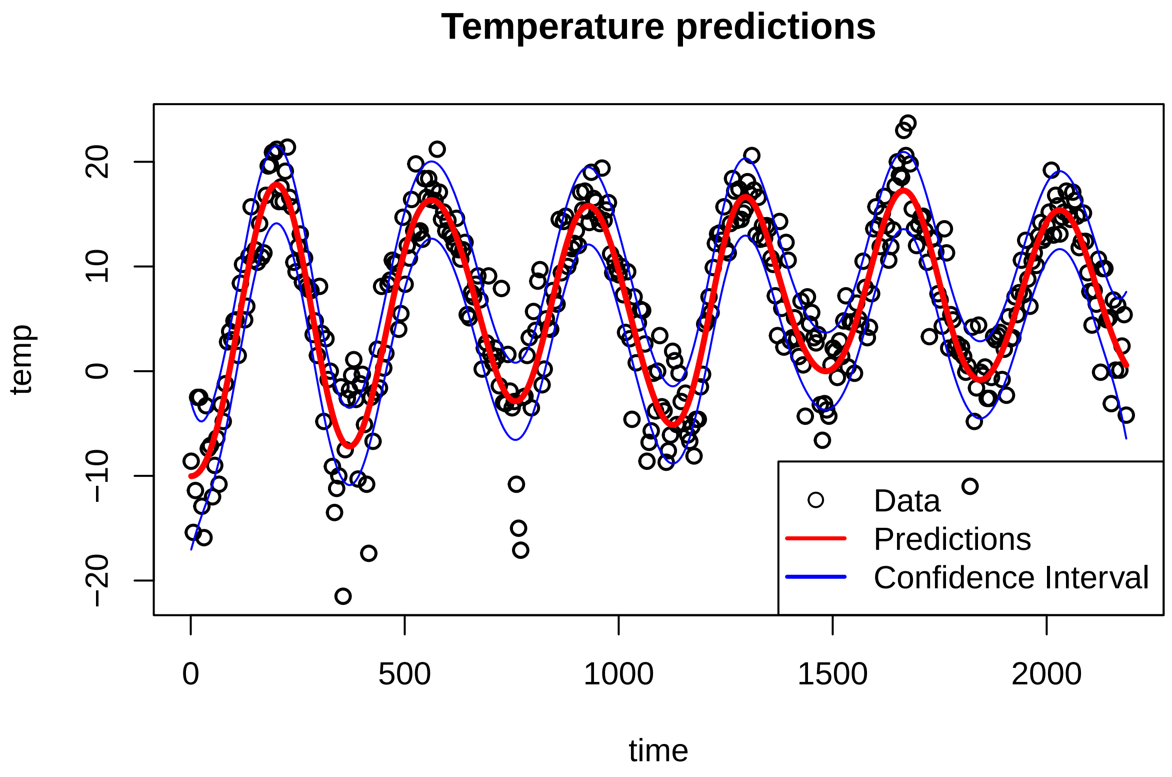

(2) Consider first the following model:

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

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 $\ell = 100$ (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).

```r
#Estimating sigmaNoise from fitting a two degree polynomial to data
polyFit <- lm(trainData$temp ~  trainData$time + I(trainData$time^2))
sigmaNoise <- sd(polyFit$residuals)

#setting hyperparameters in kernel function
SEK <- nestedSEK(sigmaF = 20, l = 100)

modelGP <- gausspr(trainData$time, trainData$temp, scaled = FALSE, kernel = SEK, var = sigmaNoise^2, va


time <- trainData$time
```

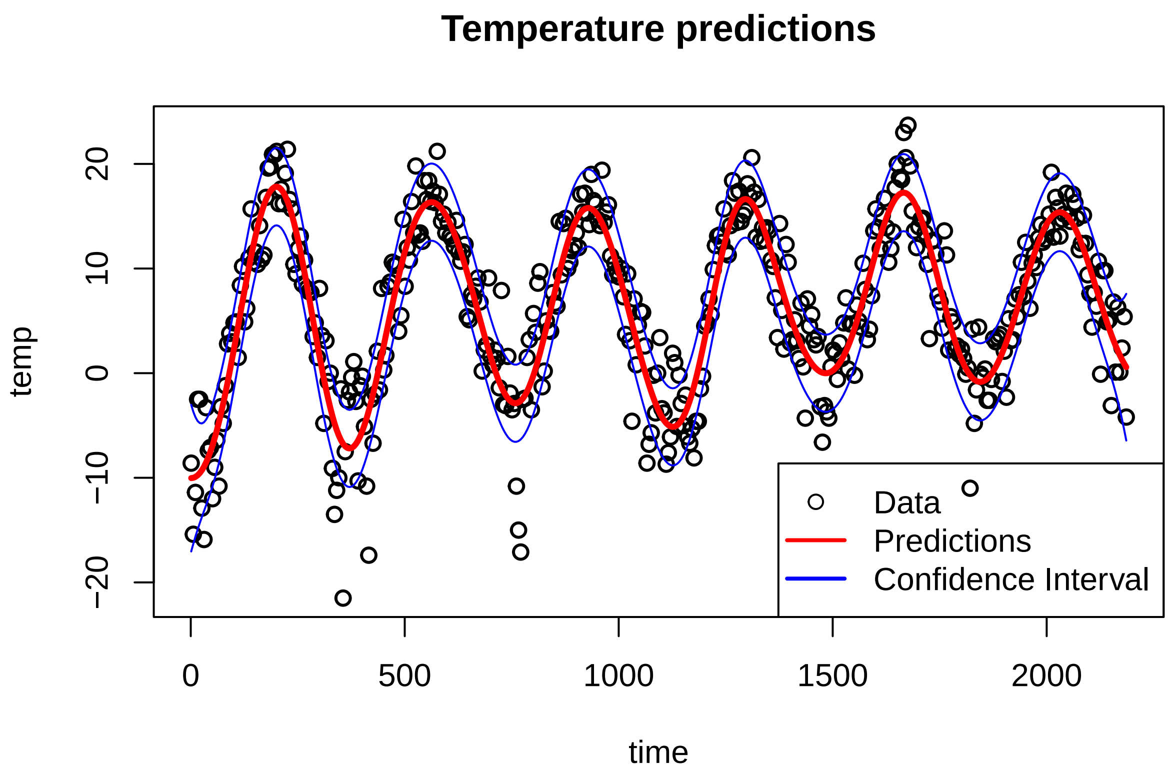
```r
posteriorMeanTime <- predict(modelGP, time)
sdMeanTime <- predict(modelGP, time, type="sdeviation")

upper <- posteriorMeanTime + 1.96 * sdMeanTime
lower <- posteriorMeanTime - 1.96 * sdMeanTime

plot(x= trainData$time, y = trainData$temp,
     xlab = "time", ylab = "temp", main = "Temperature predictions", lwd = 1.5)
lines(x=trainData$time, y = posteriorMeanTime, col = "red", lwd = 3)
lines(trainData$time, upper, col = "blue", lwd = 1)
lines(trainData$time, lower, col = "blue", lwd = 1)
legend("bottomright", legend=c("Data", "Predictions", "Confidence Interval"), pch=c(1, NA, NA), lty=c(N
```



(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.

```r
#setting hyperparameters in kernel function
SEK <- nestedSEK(sigmaF = 20, l = 100)

posterior <- posteriorGP(trainData$time, trainData$temp, trainData$time, sigmaNoise, SEK)

posteriorMean <- posterior$mean
posteriorVariance <- diag(posterior$variance)

# 95% confidence intervals
upper <- posteriorMean + 1.96 * sqrt(posteriorVariance)
lower <- posteriorMean - 1.96 * sqrt(posteriorVariance)
```

```r
plot(x= trainData$time, y = trainData$temp,
     xlab = "time", ylab = "temp", main = "Temperature predictions", lwd = 1.5)
lines(x=trainData$time, y = posteriorMean, col = "red", lwd = 3)
lines(trainData$time, upper, col = "blue", lwd = 1)
lines(trainData$time, lower, col = "blue", lwd = 1)
legend("bottomright", legend=c("Data", "Predictions", "Confidence Interval"), pch=c(1, NA, NA), lty=c(N
```



This gives the same posterior, this is expected due to the functions using the same kernels.

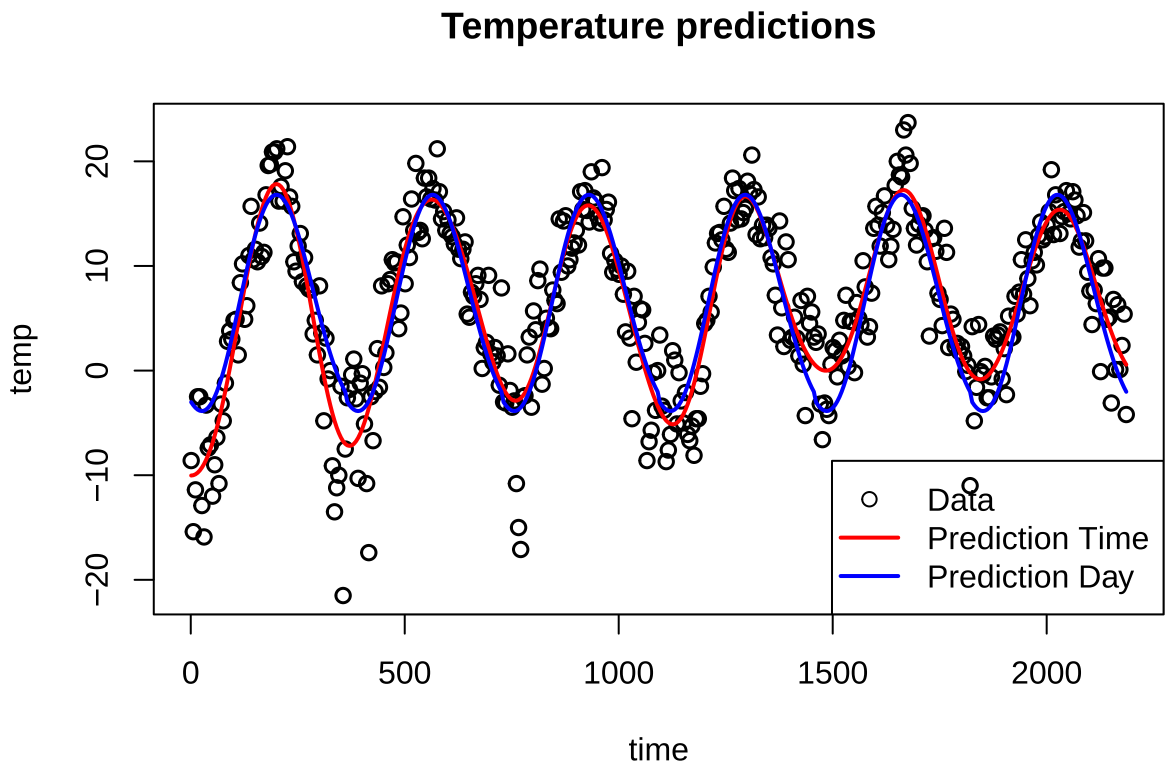
(4) Consider now the following model:

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

Estimate the model using the `gausspr` function with the squared exponential functior from (1) with $\sigma_f = 20$ and $\ell = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these σ_f and ℓ values are not suitable). Superimpose the posterioı mean from this model on the posterior mean from the model in (2). Note that this plo 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?

```r
SEK <- nestedSEK(sigmaF = 20, l = 100)

modelGP <- gausspr(trainData$day, trainData$temp, scaled = FALSE, kernel = SEK, var = sigmaNoise^2)

posteriorMeanDay <- predict(modelGP)


plot(x= trainData$time, y = trainData$temp,
     xlab = "time", ylab = "temp", main = "Temperature predictions", lwd = 1.5)
```

```r
lines(x=trainData$time, y = posteriorMeanTime, col = "red", lwd = 2)
lines(x=trainData$time, y = posteriorMeanDay, col = "blue", lwd = 2)
legend("bottomright", legend=c("Data", "Prediction Time", "Prediction Day"), pch=c(1, NA, NA), lty=c(NA
```



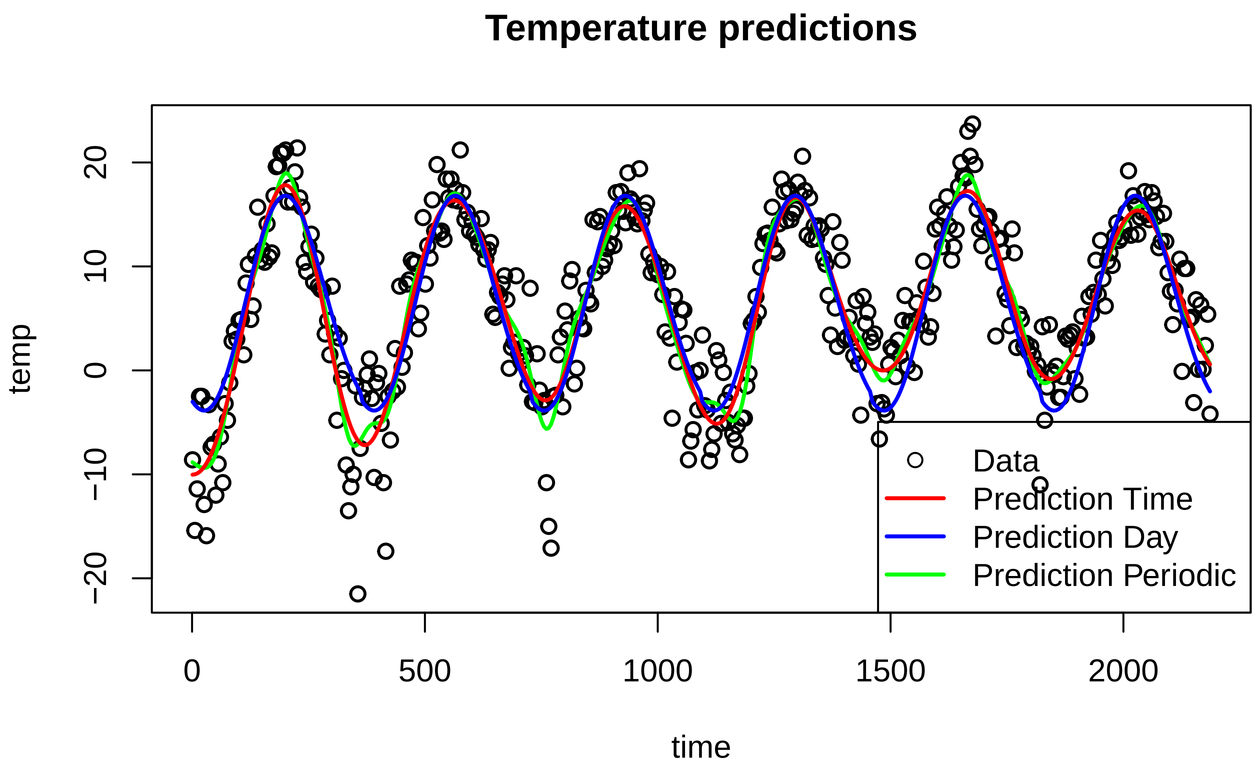
The models trained on Days vs Time gives different posteriors. The day posterior is similar every year whereas the time model differs. For instance the time model seems to show a slight increase in temperature over the years.

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

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

Note that we have two different length scales 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 $\sigma_f = 20$, $\ell_1 = 1$, $\ell_2 = 100$ 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 $\ell = 100$). Discuss the results.

```r
periodicKernel <- function(x, xstar, sigmaF = 20,l1 =1, l2 = 100, d=365){
  n1 <- length(x)
  n2 <- length(xstar)
  K <- matrix(NA,n1,n2)

  for (i in 1:n2){
    absDiff <- absDiff <- abs(x-xstar[i])
    K[,i] <- sigmaF^2*exp(-2*sin(pi*absDiff/d)^2/l1^2)*exp(-0.5*absDiff^2/l2^2)
  }
  return(K)
}
class(periodicKernel) <- 'kernel'


modelGP <- gausspr(trainData$time, trainData$temp, scaled = FALSE, kernel = periodicKernel, var = sigmal


posteriorMeanPeriodic <- predict(modelGP)


plot(x= trainData$time, y = trainData$temp,
     xlab = "time", ylab = "temp", main = "Temperature predictions", lwd = 1.5)
lines(x=trainData$time, y = posteriorMeanPeriodic, col = "green", lwd = 2)
lines(x=trainData$time, y = posteriorMeanTime, col = "red", lwd = 2)
lines(x=trainData$time, y = posteriorMeanDay, col = "blue", lwd = 2)
legend("bottomright", legend=c("Data", "Prediction Time", "Prediction Day", "Prediction Periodic"), pch=
```



The new periodic model fits the spikes of the data more closely than the previous models. This is due to the period of 365 which essentially adds the cyclical nature of years to the model.

Part 2.3

Import & Data

```r
#install.packages("https://cran.r-project.org/src/contrib/Archive/AtmRay/AtmRay_1.31.tar.gz", repos = N
library(kernlab)
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)
trainData <- data[SelectTraining,]
testData <- data[-SelectTraining,]

#accuracy function
accuracy <- function(true, pred){
  return(mean(true == pred))
}
```

(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.

```r
# Model and prediction
GPfitFraud <- gausspr(fraud ~ varWave + skewWave, data=trainData)
```

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

```r
predictionTrain <- predict(GPfitFraud)

#Confusionmatrix and accuracy
table(trainData$fraud, predictionTrain)
```

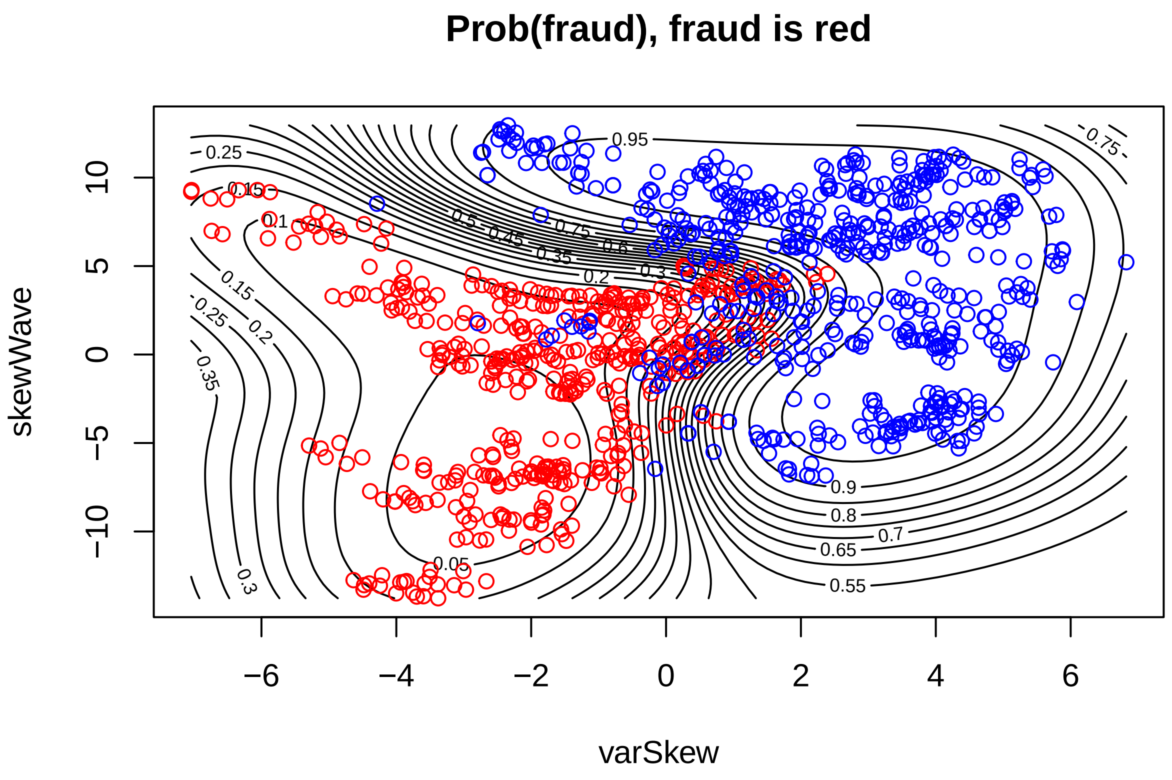
```
##    predictionTrain
##       0   1
##   0 503  41
##   1  18 438
```

```r
accuracy(trainData$fraud, predictionTrain)
```

```
## [1] 0.941
```

```r
# class probabilities
probPreds <- predict(GPfitFraud, type="probabilities")
x1 <- seq(min(trainData$varWave),max(trainData$varWave),length=100)
x2 <- seq(min(trainData$skewWave),max(trainData$skewWave),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

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



The contours of the probabilities follow the data closely.

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

```r
predictionTest <- predict(GPfitFraud, testData[,c(1,2)])

accuracy(testData$fraud, predictionTest)
```

```
## [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
GPfitFraud4 <- gausspr(fraud ~ ., data=trainData)
```

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

```r
prediction4 <- predict(GPfitFraud4, testData[,-5])
accuracy(testData$fraud, prediction4)
```

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

The accuracy of the model using all covariates has a significantly better accuracy.