Lab4

Andreas Hertin

2021-10-10

(1)

Implementing GP Regression

(1.1)

First task is implementing a function that derives the posterior distribution from given prior points and training data.

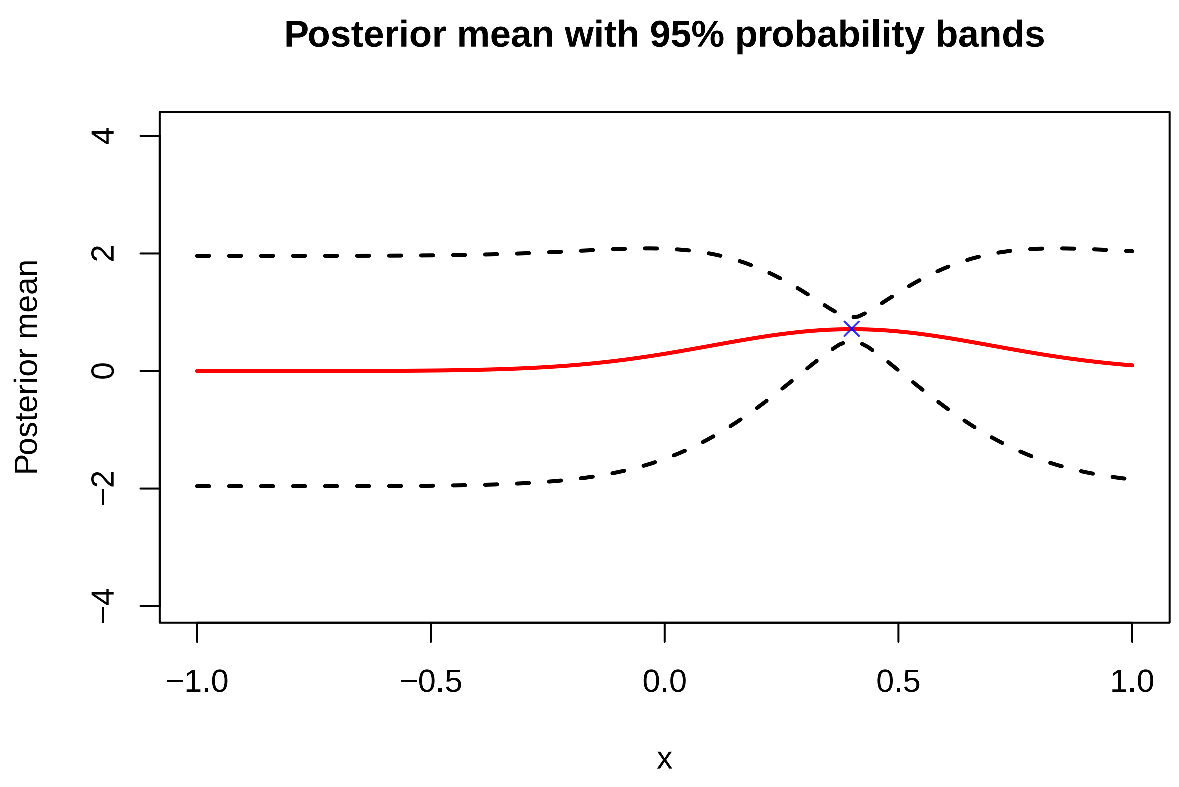
```r
# Inspired by given code from José
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}


posteriorGP <- function(x, y, XStar, sigmaNoise, k, sigmaF=1, l=3){
  n <- length(x)
  K <- k(x, x, sigmaF, l)
  L <- t(chol(K + sigmaNoise^2*diag(n)))
  L_t <- t(L)
  alpha <- solve(L_t, solve(L, y))
  K_star <- k(x, XStar, sigmaF, l)
  f_star <- t(K_star) %*% alpha

  v <- solve(L, K_star)
  K_star_star <- k(XStar, XStar, sigmaF, l)
  cov <- K_star_star - t(v) %*% v
  var <- diag(cov)

  return(list(mean = f_star, var = var))

}
```

(1.2)

In this section the function is run with $\sigma_f = 1$ and $l = 0.3$ and a single point.

```r
plotGP <- function(mean, var = NULL, interval, x, y, title){

  if(!is.null(var)){

    conf_int <- list(upper = mean + 1.96 * sqrt(var),
                     lower = mean - 1.96 * sqrt(var))

    ylim <- c(min(conf_int$lower) - 2,
              max(conf_int$upper) + 2)

    plot(interval,
         mean,
         type = "l",
         ylim = ylim,
         col = "red",
         main = title,
         xlab = "x",
         ylab= "Posterior mean",
         lwd = 2)

    lines(x = interval, y = conf_int$lower, col = "black", lwd = 2, lty = 2)
    lines(x = interval, y = conf_int$upper, col = "black", lwd = 2, lty = 2)

  } else {

    ylim <- c(min(y) - 1,
              max(y) + 1)

    plot(interval,
         mean,
         type = "l",
         main = title,
         ylim = ylim,
         col = "red",
         xlab = "time",
         ylab = "temp",
         lwd = 2)

  }

  points(x,
         y,
         col = alpha("blue", 0.8),
         pch = 4)
}

x <- 0.4
y <- 0.719
sigN <- 0.1
sigF <- 1
```

```r
l <- 0.3
x_star <- seq(-1, 1, length=100)

posterior <- posteriorGP(x, y, x_star, sigN, SquaredExpKernel, sigF, l)

plotGP(posterior$mean,
       posterior$var,
       x_star,
       x,
       y,
       title = "Posterior mean with 95% probability bands")
```

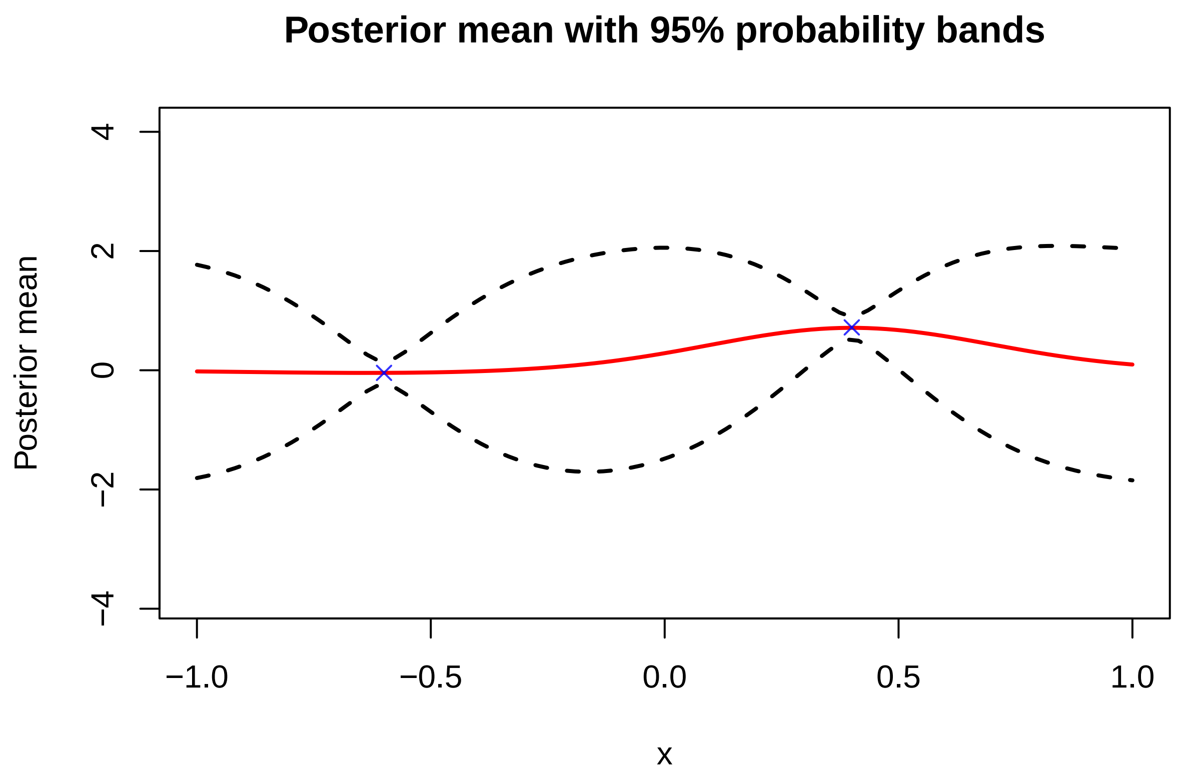


The probability bands is only close to the mean close to the given point. It is noit at the point because of assumed noise.

(1.3)

Another point is added to see the effect on mean and their probability bands.

```r
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)

posterior <- posteriorGP(x, y, x_star, sigN, SquaredExpKernel, sigF, l)

plotGP(posterior$mean,
       posterior$var,
       x_star,
       x,
       y,
       title = "Posterior mean with 95% probability bands")
```

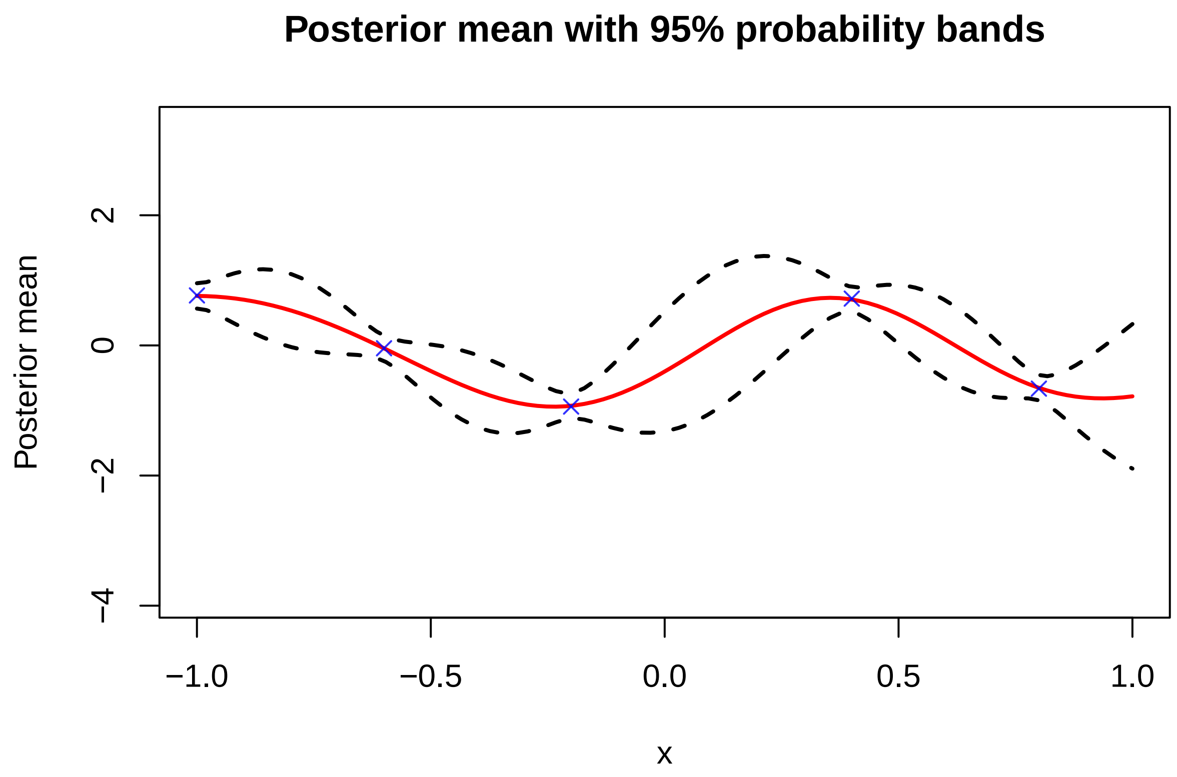


A similar response around the additional point as in **1.2**

(1.4)

Even more points are added.

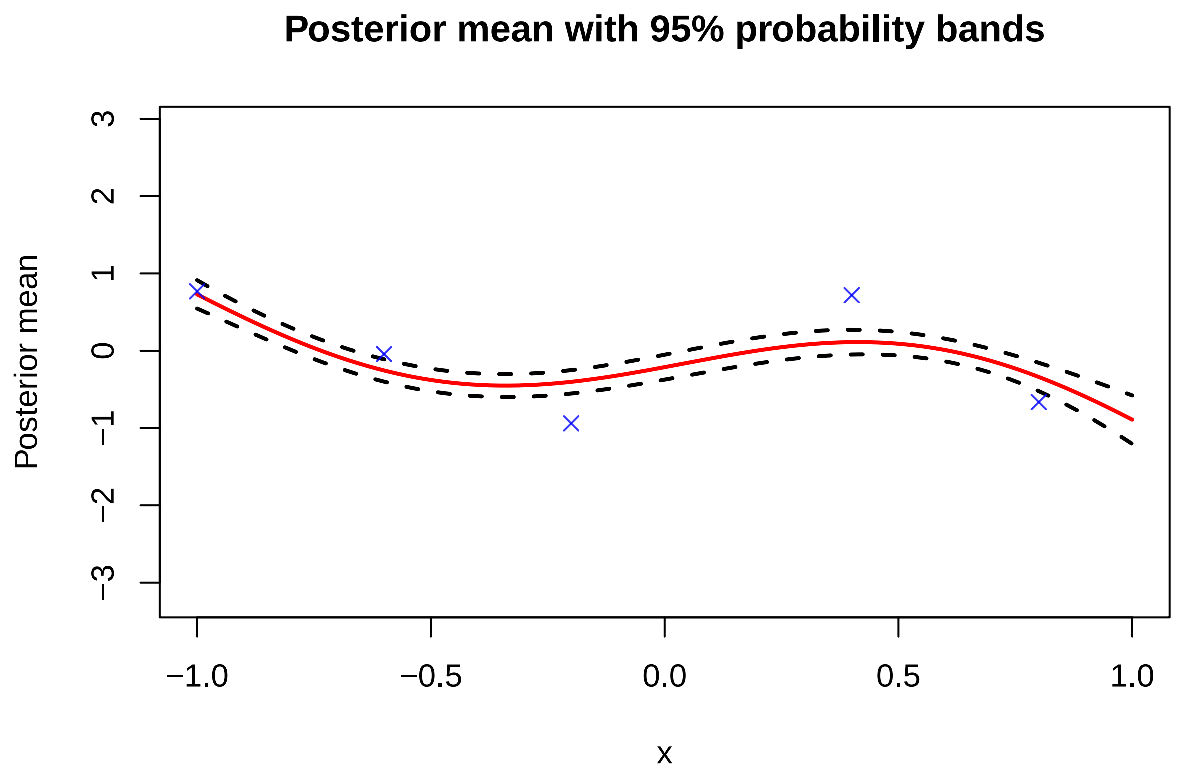
```
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)

posterior <- posteriorGP(x, y, x_star, sigN, SquaredExpKernel, sigF, l)

plotGP(posterior$mean,
       posterior$var,
       x_star,
       x,
       y,
       title = "Posterior mean with 95% probability bands")
```



Points are now so close that the probability between them never reaches previous levels as they influence enough between eachother.

(1.5)

```
sigF <- 1
l <- 1

posterior <- posteriorGP(x, y, x_star, sigN, SquaredExpKernel, sigF, l)

plotGP(posterior$mean,
       posterior$var,
       x_star,
       x,
       y,
       title = "Posterior mean with 95% probability bands")
```



A quite drastic change can be observed when having σ_f and l both equal 1 in the kernel function when comparing the graphs from **1.4** and **1.5**. Having a higher l means that correlation between two adjacent points and gives a smoother function. So graph in **1.5** is more smooth.

(2)

GP regression on temperature data using `kernlab`

(2.1)

A exponential kernel is created so that it can take in σ_f and l

```
KernelFunction <- function(sigmaF, l) {
  sqEx <- SquaredExpKernel <- function(x1, x2 = NULL){
    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)
  }
  class(sqEx) <- "kernel"
  return(sqEx)
}

x <- c(1, 2, 3)
x_star <- c(2, 3, 4)

kernel <- KernelFunction(sigmaF = 1, l = 0.3)

kernel_test <- kernel(1, 2)
kernel_test
```

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

```
cov_matrix <- kernelMatrix(x, x_star)
cov_matrix
```

```
## An object of class "kernelMatrix"
##              [,1]       [,2]         [,3]
## [1,] 1.000000e+00 0.00386592 2.233631e-10
## [2,] 3.865920e-03 1.00000000 3.865920e-03
## [3,] 2.233631e-10 0.00386592 1.000000e+00
```

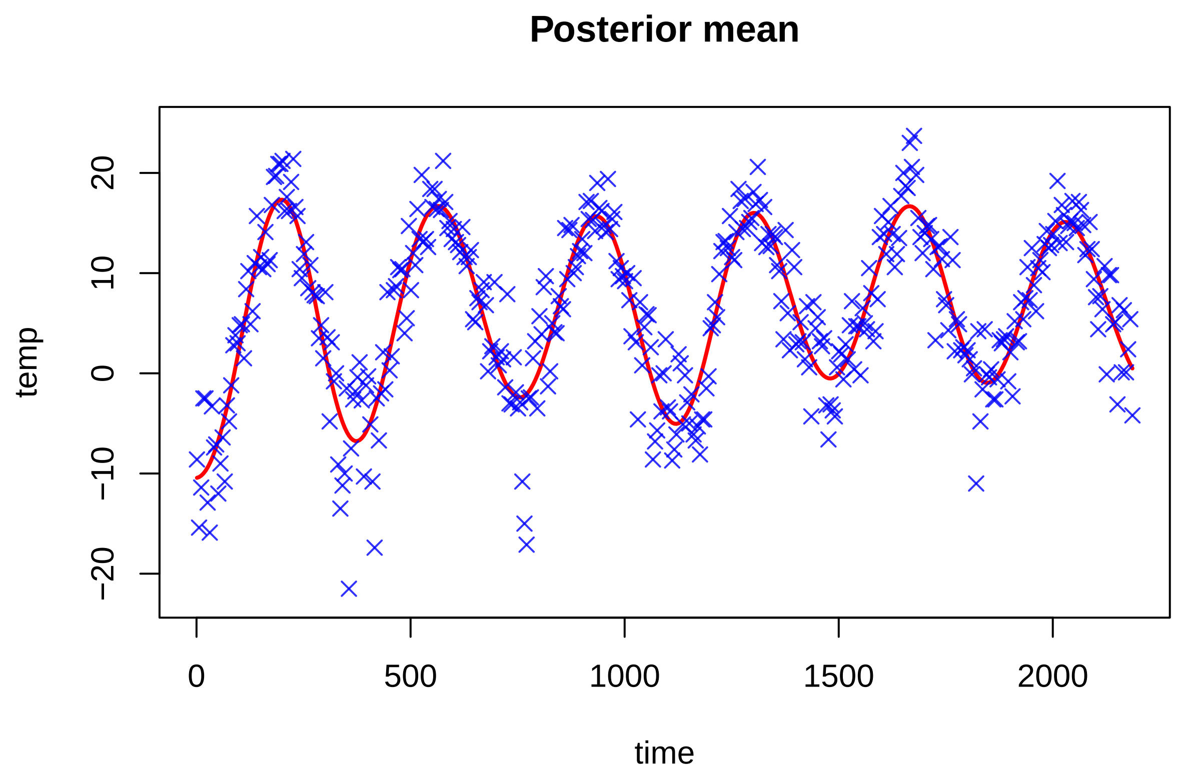
(2.2)

```
temps_csv <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulli

by_ <- 5

time <- seq(1, 2190, by = by_)

temps <- temps_csv$temp[time]

fit <- lm(temps ~ time + I(time^2))

sigN <- sd(fit$residuals)
sigN
```

```
## [1] 8.176288
```

The estimate of the measurement noise (σ_n) is obtained by getting the standard deviation of the residuals when fitting a simple quadratic model to the data.

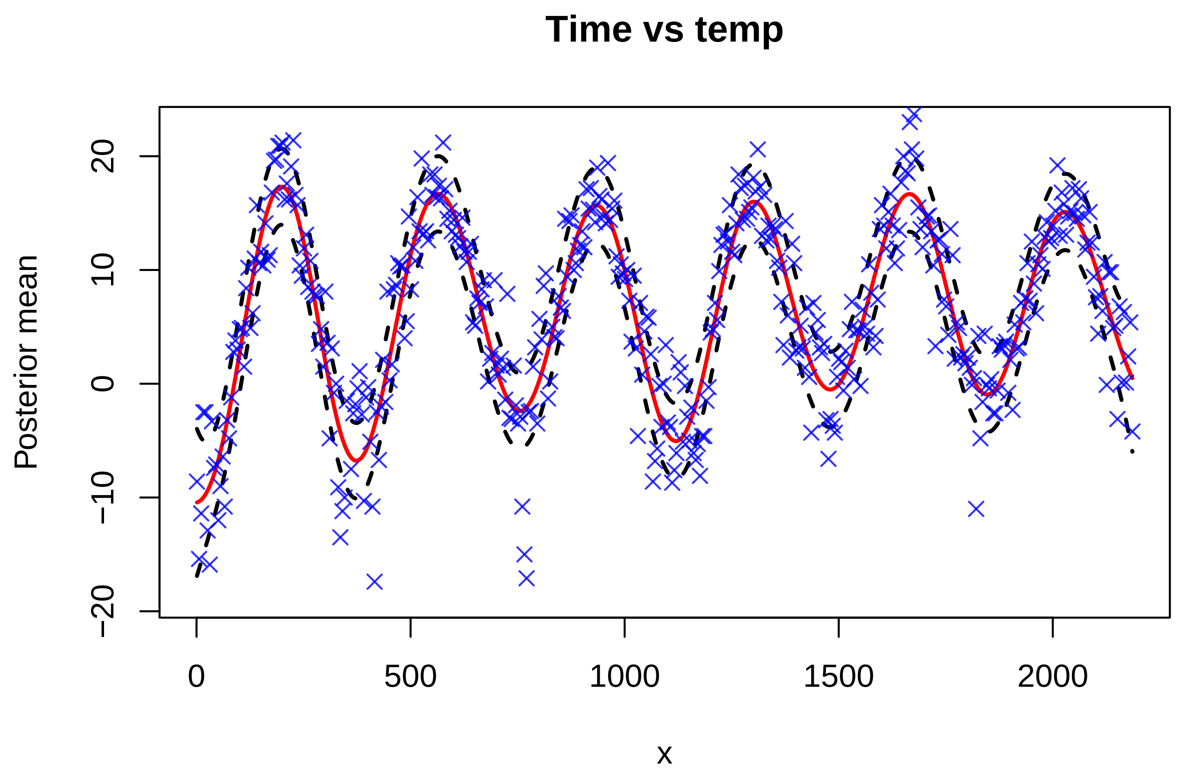
```
sigF <- 20
l <- 0.2

GPfit <- gausspr(x = time, y = temps, kernel = KernelFunction, kpar = list(sigmaF = sigF, l = l), var =

GPmean <- predict(GPfit, time)

plotGP(mean = GPmean, interval = time, x = time, y = temps, title = "Posterior mean")
```



The posterior follows the data well. It does has a hard time coming close to extremes however.

(2.3)

```
kernel <- KernelFunction(sigmaF = sigmaF, l = l)

GPpred <- posteriorGP(scale(time), scale(temps), scale(time), sigN, SquaredExpKernel, sigmaF = sigF, l =

sdTemps <- sd(temps)
mnTemps <- mean(temps)

meanPred <- GPpred$mean * sdTemps + mnTemps

plotGP(mean = meanPred,
       GPpred$var,
       interval = time,
       x = time,
       y = temps,
       title = "Time vs temp")
```

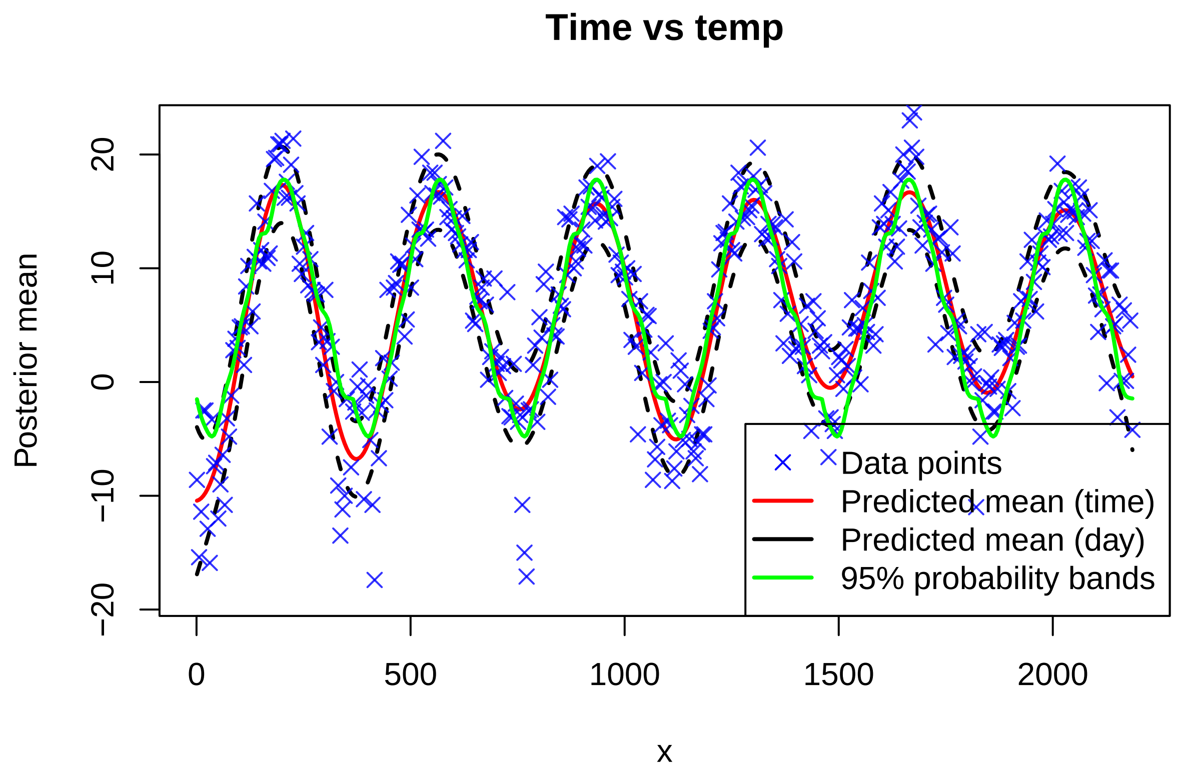


We can see that the bands incorporates more data points but fail to come close to the extremes (i.e at time ~750)

(2.4)

```
days <- rep(seq(1, 365, by = by_), times = 6)

GPdaily <- gausspr(days,
                   temps,
                   kernel = KernelFunction,
                   kpar = list(sigmaF = sigF, l = l),
                   var = sigN^2)

dailyMean <- predict(GPdaily, days)

plotGP(mean = GPmean,
       var = GPpred$var,
       interval = time,
       x = time,
       y = temps,
       title = "Time vs temp")

lines(x = time, y = dailyMean, col = "green", lwd = 2)
```

```r
legend("bottomright",
       legend = c("Data points",
                  "Predicted mean (time)",
                  "Predicted mean (day)",
                  "95% probability bands"),
       col = c("blue", "red", "black", "green"),
       pch = c(4, NA, NA, NA),
       lwd = c(NA, 2, 2, 2))
```

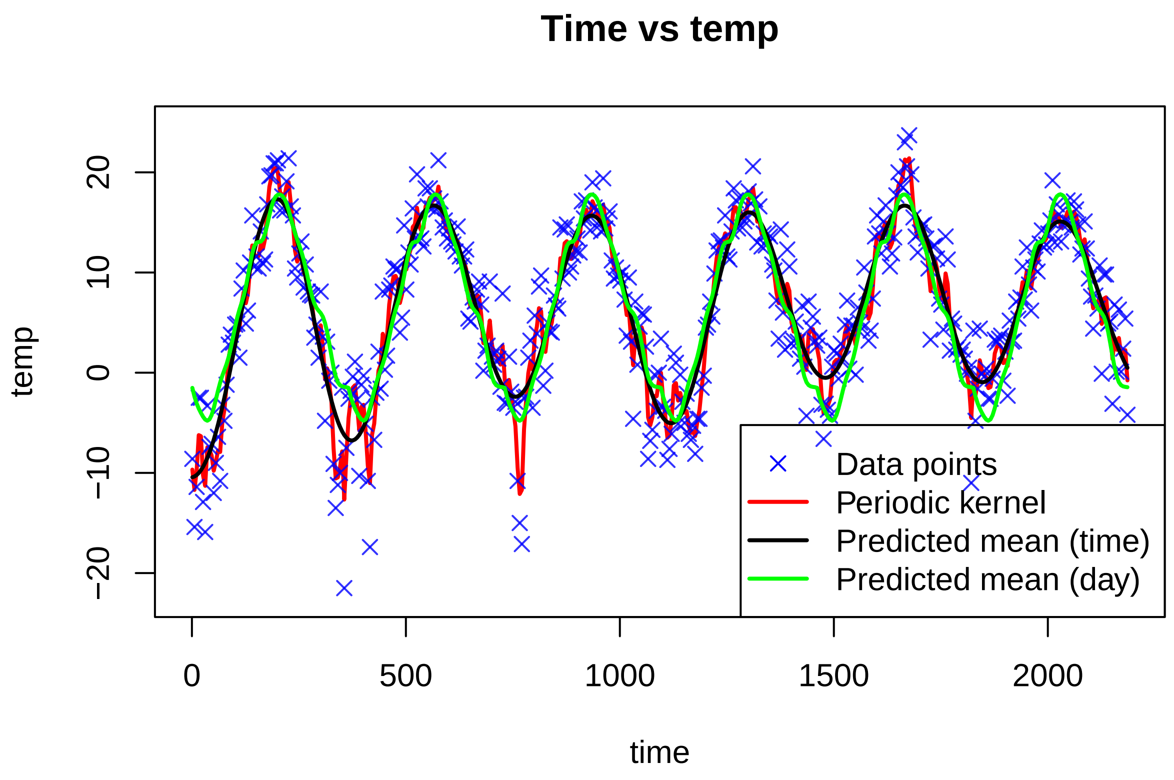


Some points have improved when using the day set compared to the all-times set but they are mostly similar.

(2.5)

```r
PeriodicKernel <- function(sigmaF, l1, l2, d){
  val <- function(x, x_star){
    dif <- abs(x - x_star)
    out <- sigmaF^2*exp(-((2*sin(pi*dif)) / d) / l1^2) * exp(-0.5 * dif^2 / l2^2)
    return(out)
  }
  class(val) <- "kernel"
  return(val)
}

l1 <- 1
```

```r
l2 <- 10
d <- 365 / sd(time)

periodicKernel <- PeriodicKernel(sigF, l1, l2, d)
GPper <- gausspr(time, temps, kernel = periodicKernel, var = sigN^2)
perMean <- predict(GPper, time)
plotGP(mean = perMean,
       interval = time,
       x = time,
       y = temps,
       title = "Time vs temp")

lines(x = time, y = GPmean, col = "black", lwd = 2)
lines(x = time, y = dailyMean, col = "green", lwd = 2)

legend("bottomright",
       legend = c("Data points",
                  "Periodic kernel",
                  "Predicted mean (time)",
                  "Predicted mean (day)"),
       col = c("blue", "red", "black", "green"),
       pch = c(4, NA, NA, NA),
       lwd = c(NA, 2, 2, 2))
```



The periodic kernel is not as smooth as the other kernels but it does fit the data better. It captures more data further from the posterior mean.

(3)

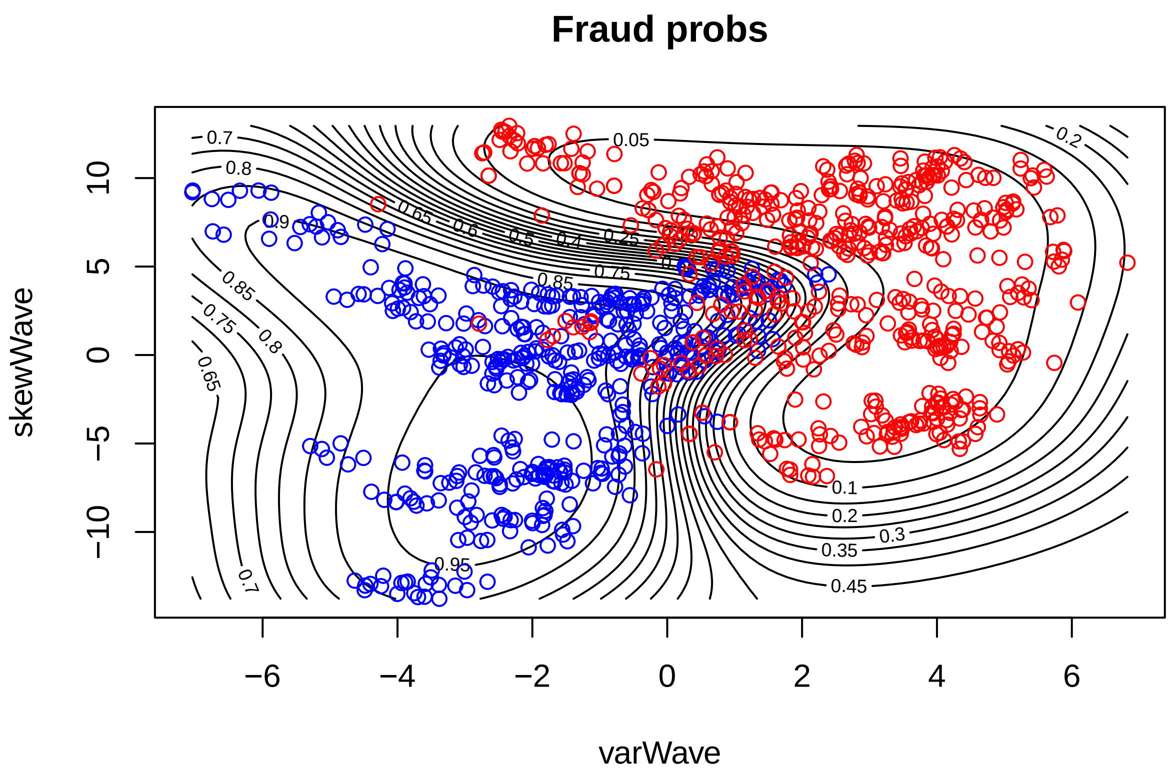
```r
library(AtmRay)

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, ]
```

(3.1)

```r
fraudModel <- gausspr(fraud ~ varWave + skewWave, data = trainData)
```

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

```r
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) <- c("varWave", "skewWave")

probPreds <- predict(fraudModel, gridPoints, type="probabilities")
trainPreds <- predict(fraudModel, trainData)

frauds <- which(trainData$fraud == 1)

contour(x1,
        x2,
        matrix(probPreds[, 2], 100, byrow = TRUE),
        20,
        xlab = "varWave",
        ylab = "skewWave",
        main = 'Fraud probs')

points(x = trainData$varWave[frauds],
       y = trainData$skewWave[frauds],
       col="blue")
points(x = trainData$varWave[-frauds],
       y = trainData$skewWave[-frauds],
       col="red")
```



```r
confMatrix = table(trainPreds, trainData$fraud)
confMatrix
```

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

```r
acc <- sum(diag(confMatrix)) / sum(confMatrix)
acc
```

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

The fraud points are red in the plot and the accuracy is 94.1%.

(3.2)

```r
testPreds <- predict(fraudModel, newdata = testData)

confMatrix = table(testPreds, testData$fraud)
confMatrix
```

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

```
acc <- sum(diag(confMatrix)) / sum(confMatrix)
acc
```

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

The accuracy of the test data is high, but it naturally isn't as good as that of the training data.

(3.3)

```
fraudModel_2 <- gausspr(fraud ~., data = trainData)
```

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

```
testPreds_2 <- predict(fraudModel_2, newdata = testData)

confMatrix = table(testPreds_2, testData$fraud)
confMatrix
```

```
##
## testPreds_2   0   1
##           0 216   0
##           1   2 154
```

```
acc <- sum(diag(confMatrix)) / sum(confMatrix)
acc
```

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

Having more information leads to better accuracy since the new covariates only add additional information and do not remove any.