# part4

*Omkar Bhutra (omkbh878)*

*9 January 2020*

LAB 4 stuff

## install.packages("mvtnorm")

library("mvtnorm")

## 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) }

## Mean function

MeanFunc <- function(x){ m <- sin(x) return(m) }

## Simulates nSim realizations (function) from a GP with mean m(x) and covariance K(x,x')

## over a grid of inputs (x)

SimGP <- function(m = 0,K,x,nSim,...){ n <- length(x) if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(x) covMat <- K(x,x,...) f <- rmvnorm(nSim, mean = meanVector, sigma = covMat) return(f) }

xGrid <- seq(-5,5,length=20)

## Plotting one draw

sigmaF <- 1 l <- 1 nSim <- 1 fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l) plot(xGrid, fSim[1,], type="p", ylim = c(-3,3)) if(nSim>1){ for (i in 2:nSim) { lines(xGrid, fSim[i,], type="p") } } lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3) lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2) lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2)

## Plotting using manipulate package

library(manipulate)

plotGPPrior <- function(sigmaF, l, nSim){ fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l) plot(xGrid, fSim[1,], type="l", ylim = c(-3,3), ylab="f(x)", xlab="x") if(nSim>1){ for (i in 2:nSim) { lines(xGrid, fSim[i,], type="l") } } lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3) lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2) lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))), col = "blue", lwd = 2) title(paste('length scale =',l,', sigmaf =',sigmaF)) }

manipulate( plotGPPrior(sigmaF, l, nSim = 10), sigmaF = slider(0, 2, step=0.1, initial = 1, label = "SigmaF"), l = slider(0, 2, step=0.1, initial = 1, label = "Length scale, l") )

#### Lab 4 setup

## Assignment 1

## a)

```
tullinge  <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTull
data <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
                 header=FALSE, sep=',')
# The kernel function
exp_kern <- function(x,xi,l, sigmaf ){

  return((sigmaf^2)*exp(-0.5*( (x - xi) / l )^2))
}
# The implementation, can take a custom kernel of any class.
linear_gp <- function(x,y,xStar,hyperParam,sigmaNoise,kernel){
  n <- length(x)
  kernel_f <- kernel
  # K = Covariance matrix calculation
  K <- function(X, XI,...){

    kov <- matrix(0,nrow = length(X), ncol = length (XI))

    for(i in 1:length(XI)){

      kov[,i]<- kernel_f(X,XI[i],...)

    }
    return(kov)
  }
  l <-hyperParam[1]
  sigmaf <- hyperParam[2]
  #K(X,X)
  K_xx   <- K(x,x, l = l, sigmaf = sigmaf) #, kernel = exp_kern
  #K(X*,X*)
  K_xsxs <- K(xStar,xStar, l = l, sigmaf = sigmaf) # kernel = exp_kern,
  #K(X,X*)
  K_xxs  <- K(x,xStar, l = l, sigmaf = sigmaf)  #kernel = exp_kern,
```

```
  # Algorithm in page 19 of the Rasmus/Williams book
  sI <- sigmaNoise^2 * diag(dim(as.matrix(K_xx))[1])
  # L is transposed according to a definition in the R & W book
  L_transposed <- chol(K_xx + sI)
  L <- t(L_transposed)

  alpha <- solve(t(L), solve(L,y))
  f_bar_star <- t(K_xxs) %*% alpha
  v <- solve(L,K_xxs)
  V_fs <- K_xsxs - t(v) %*% v
  log_mlike <- -0.5 %*% t(y) %*% alpha - sum( diag(L) - n/2 * log(2*pi) )
  return(list(fbar = f_bar_star, vf = V_fs, log_post= log_mlike))
}
```

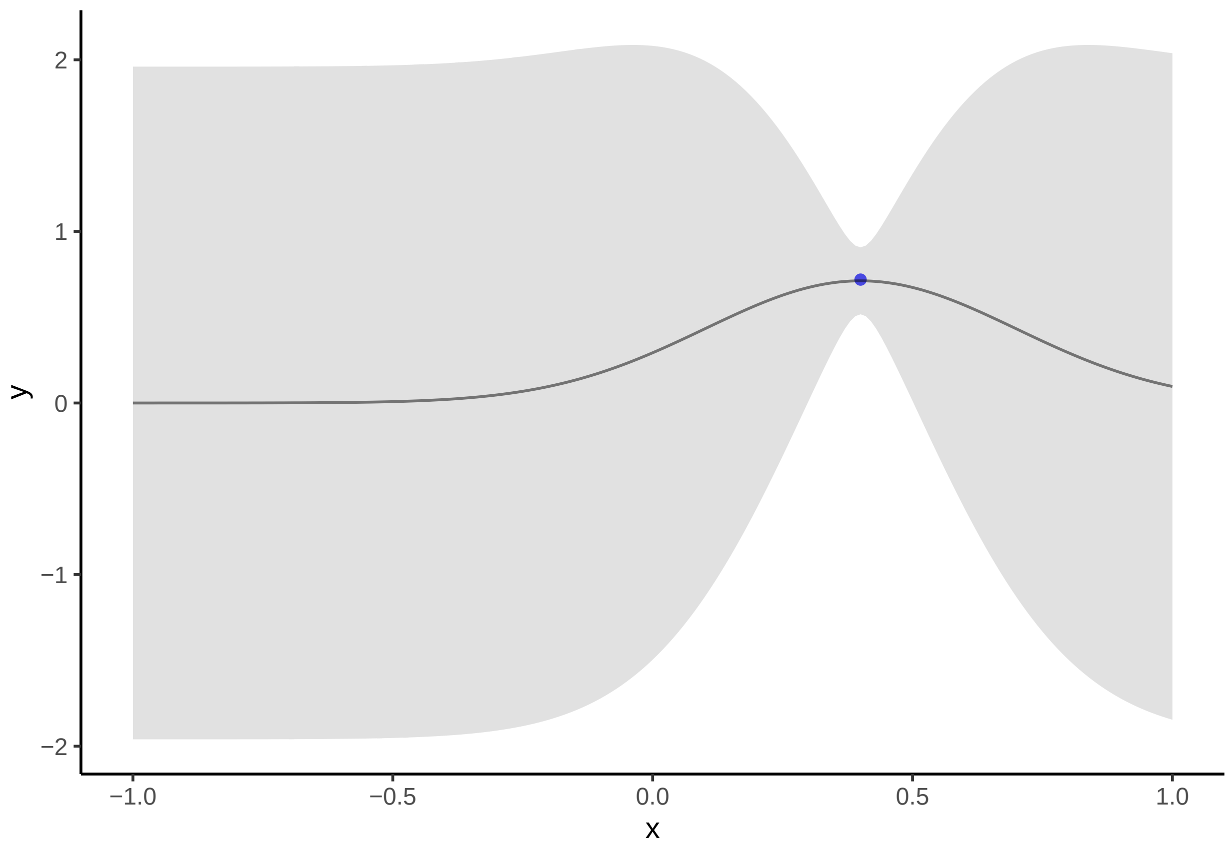
```
# Utility function for the tasks
plot_gp<- function(plot_it,band_it){
  ggplot() +

    geom_point(
      aes(x = x, y = y),
      data = plot_it,
      col = "blue",
      alpha = 0.7) +

    geom_line(
      aes(x = xs, y = fbar),
      data = band_it,
      alpha = 0.50) +

    geom_ribbon(
      aes(ymin = low, ymax = upp, xs),
      data = band_it,
      alpha = 0.15) +

    theme_classic()

}
```

```
# The data given
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
# The noise
sn <- 0.1
# The training grid
xs <- seq(-1,1,0.01)
# Hyperparameters l an sigma
hyperParam <- c(0.3, 1)
# Another utility function
repeter <- function(x,y,xs,sn,hyperParam,kernel){
  res <- linear_gp(x,y,xs,hyperParam,sn,kernel)
  # If you want the prediction band just add the noise variance (ie the sigma_n)
  upp <- res$fbar + 1.96*sqrt(diag(res$vf))
  low <- res$fbar - 1.96*sqrt(diag(res$vf))
```

```
  plot_it <- data.frame(x = x, y = y)
  band_it <- data.frame(xGrid = xs, fbar = res$fbar,  upp = upp, low = low)
  plot_gp(plot_it,band_it)
}
```

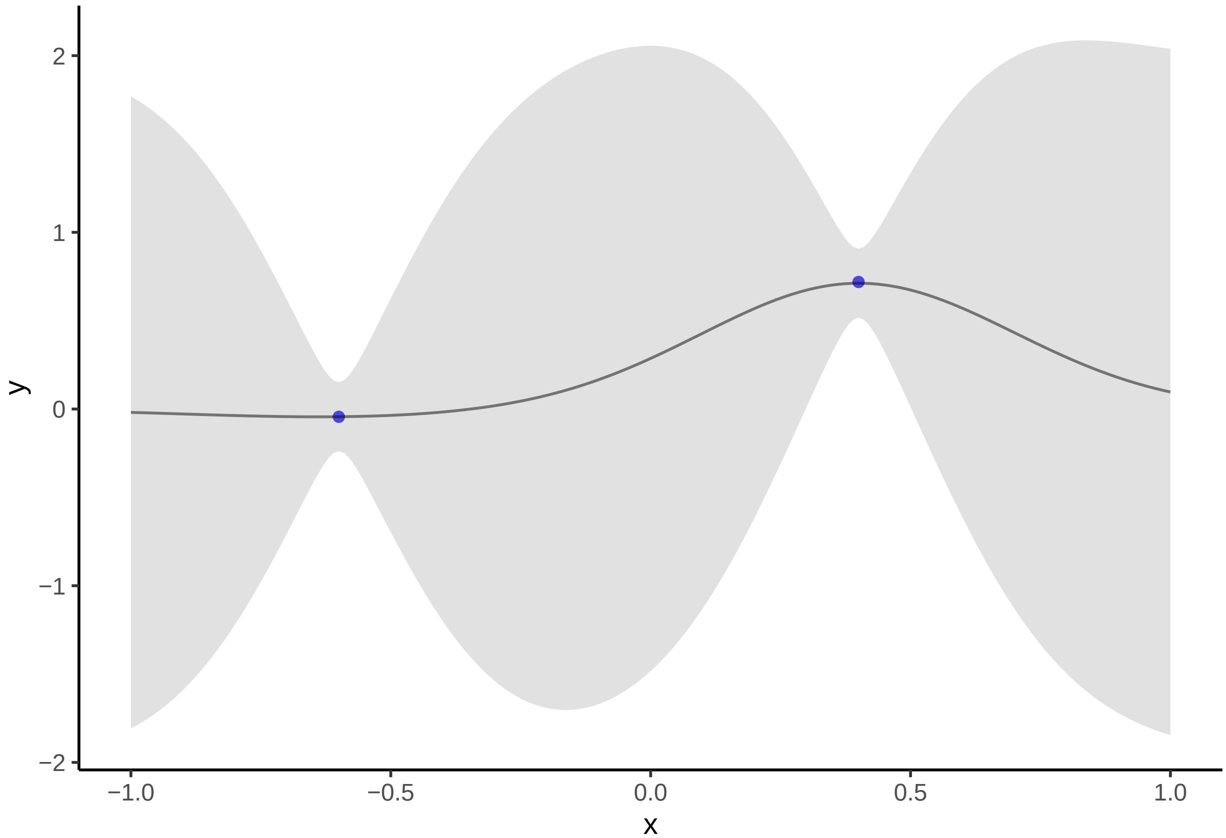
**b)**

```
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
```



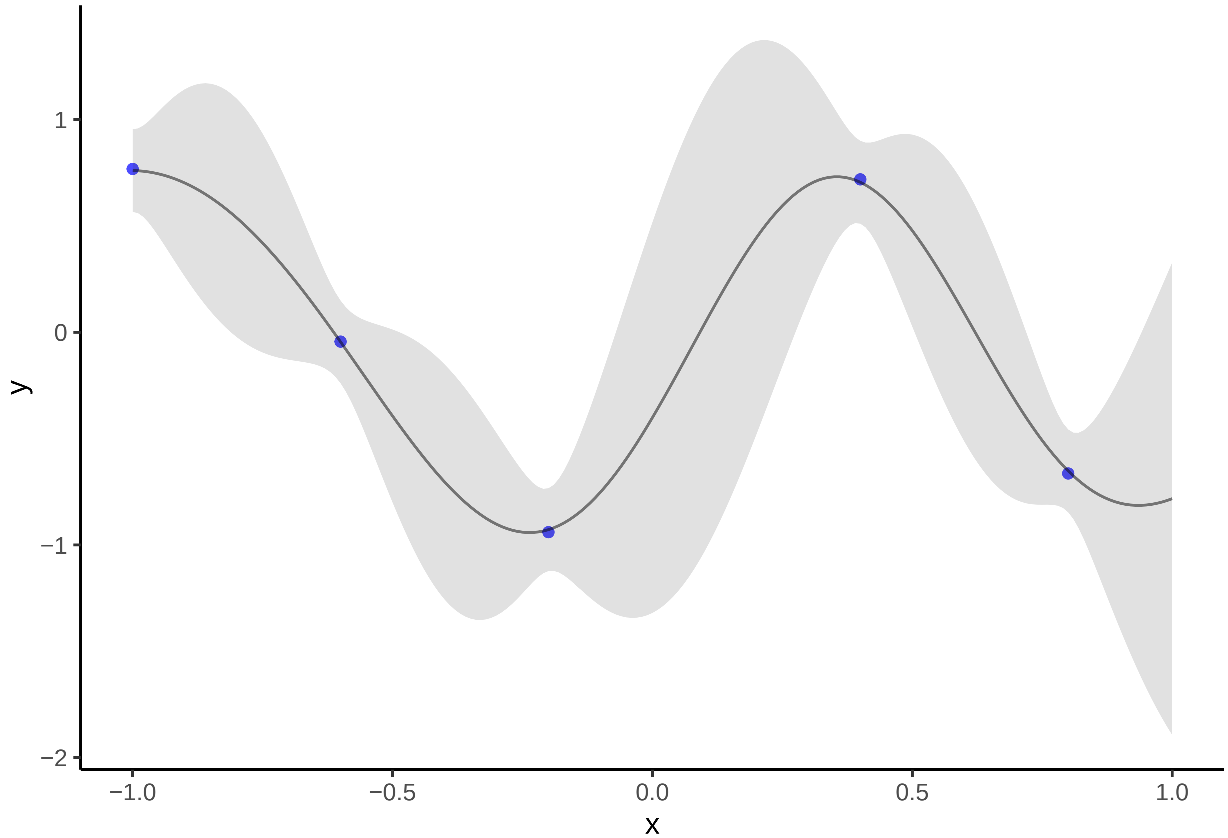

**c)**

```
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
```



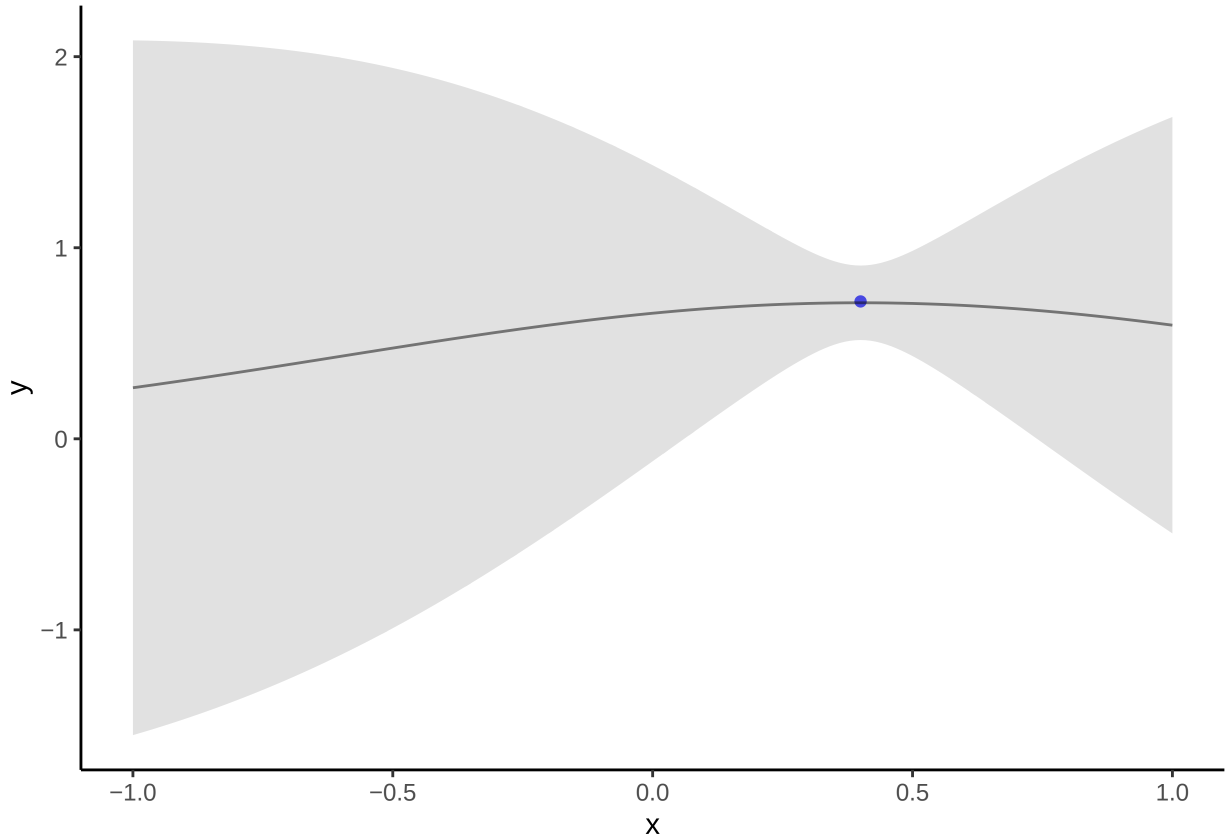

**d)**

```r
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
```

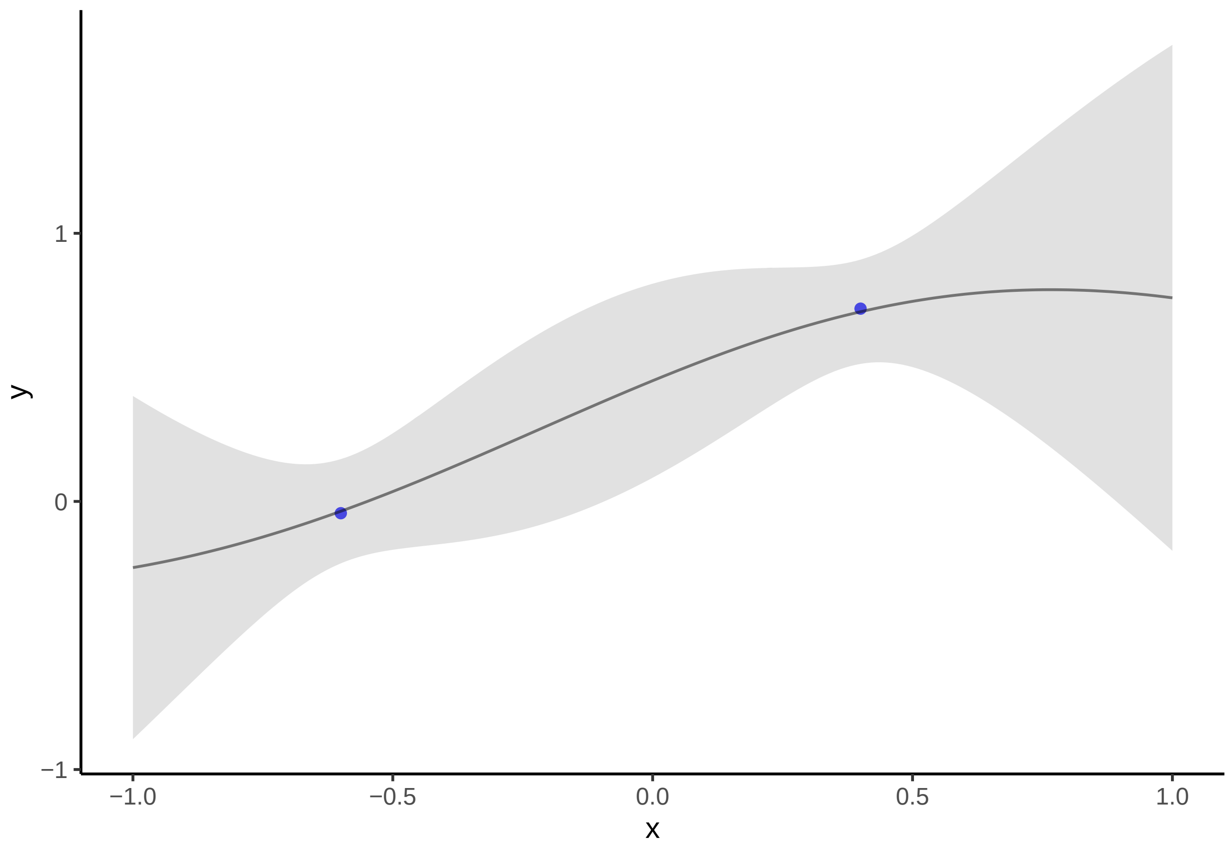



**e)**

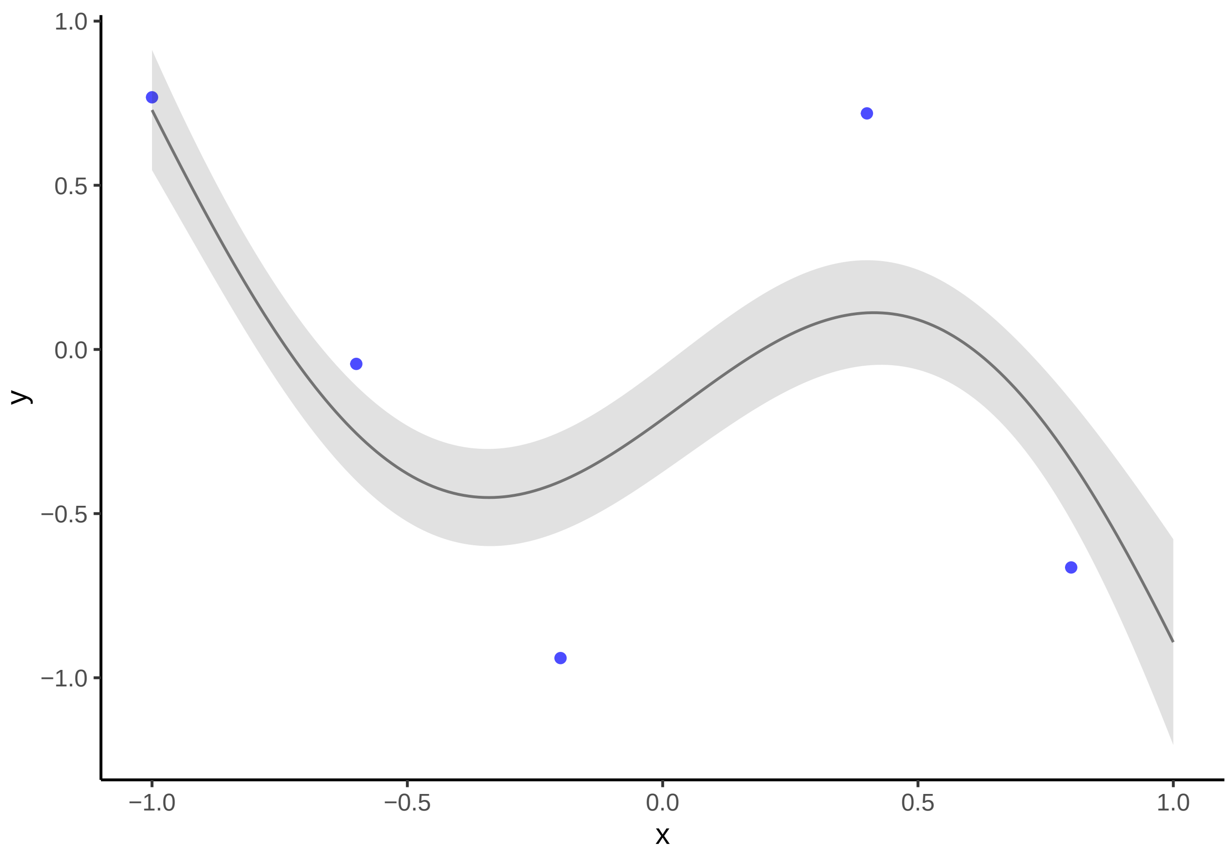
```
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
sn <- 0.1
xs <- seq(-1,1,0.01)
hyperParam <- c(1, 1)
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
```




```r
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
```




```
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
```




## Assignment 2

### Data preparations

```
tullinge$time <- 1:nrow(tullinge)
tullinge$day <- rep(1:365,6)
time_sub <- tullinge$time %in% seq(1,2190,5)
tullinge <- tullinge[time_sub,]
```

### a)

```
kern_maker <- function(l,sigmaf){

  exp_k <- function(x,y = NULL){

    return((sigmaf^2)*exp(-0.5*( (x - y) / l )^2))
  }

  class(exp_k) <- "kernel"
  return(exp_k)
}
```

```r
# gausspr()
# kernelMatrix()
ell <- 1
# SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdo (which is the SE kernel)
# SEkernel(1,2)
my_exp <- kern_maker(l = 10, sigmaf =20)
x <- c(1,3,4)
x_star <- c(2,3,4)
#my_exp(x,x_star)
kernelMatrix(my_exp,x,x_star)
```

```
## An object of class "kernelMatrix"
##          [,1]     [,2]    [,3]
## [1,] 398.0050 392.0795 382.399
## [2,] 398.0050 400.0000 398.005
## [3,] 392.0795 398.0050 400.000
```

### b)

```r
lm_tull <- lm(temp ~ time + I(time^2), data = tullinge)
sigma_2n <- var(resid(lm_tull))
a2b_kern <- kern_maker(l = 0.2, sigmaf = 20 )
gp_tullinge <- gausspr(x = tullinge$time,
                       y = tullinge$temp,
                       kernel = a2b_kern,
                       var = sigma_2n)
```

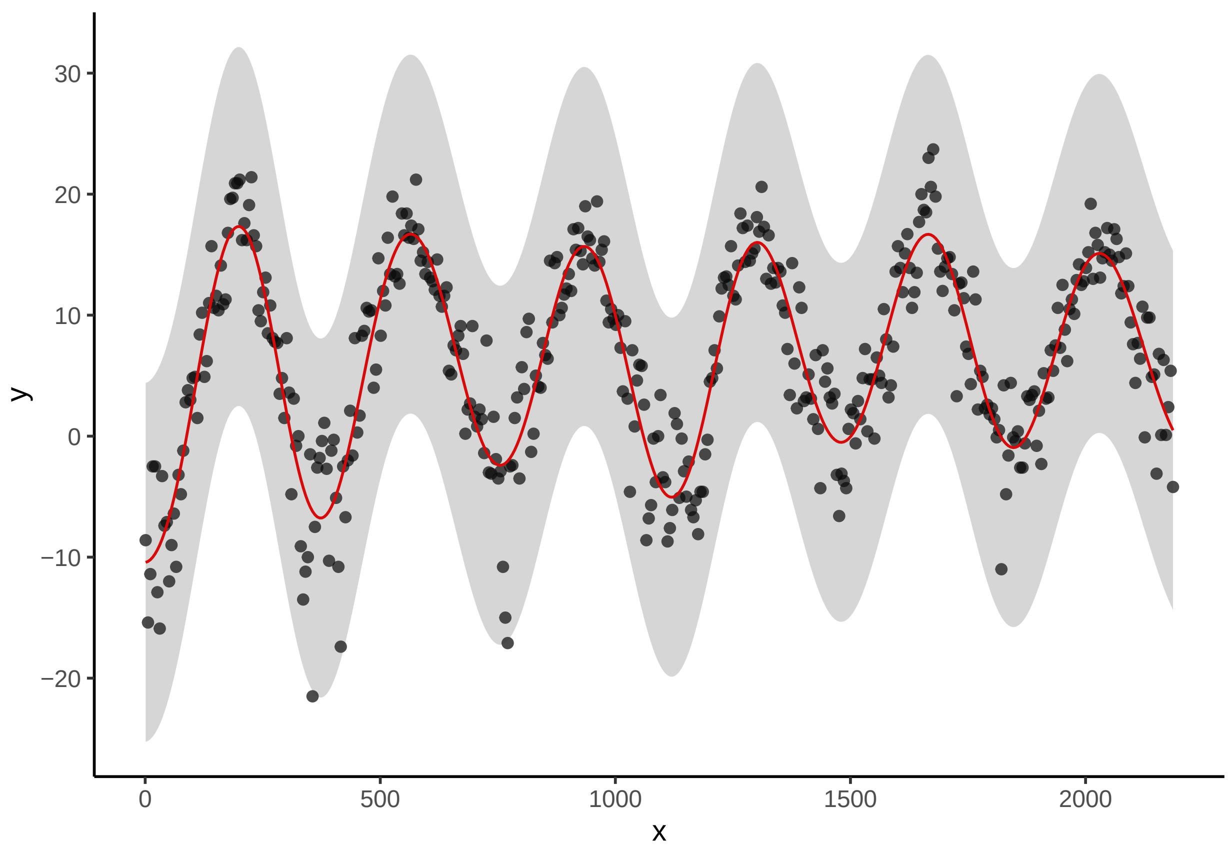
See task c) for the plot.

### c)

```r
sn_2c <- sqrt(sigma_2n)
xs_2c <- tullinge$time
hyperParam_2c <- c(0.2, 20)
res_2c<- linear_gp(x = tullinge$time,
                   y = tullinge$temp,
                   xStar = xs_2c,
                   sigmaNoise  = sn_2c,
                   hyperParam = hyperParam_2c,
                   kernel = exp_kern)
upp2c <- predict(gp_tullinge) + 1.96*sqrt(diag(res_2c$vf))
low2c <- predict(gp_tullinge) - 1.96*sqrt(diag(res_2c$vf))
plot_it1 <- data.frame(x = tullinge$time, y = tullinge$temp)
band_it1 <- data.frame(xGrid = xs_2c, fbar = res_2c$fbar,  upp = upp2c, low = low2c)
```

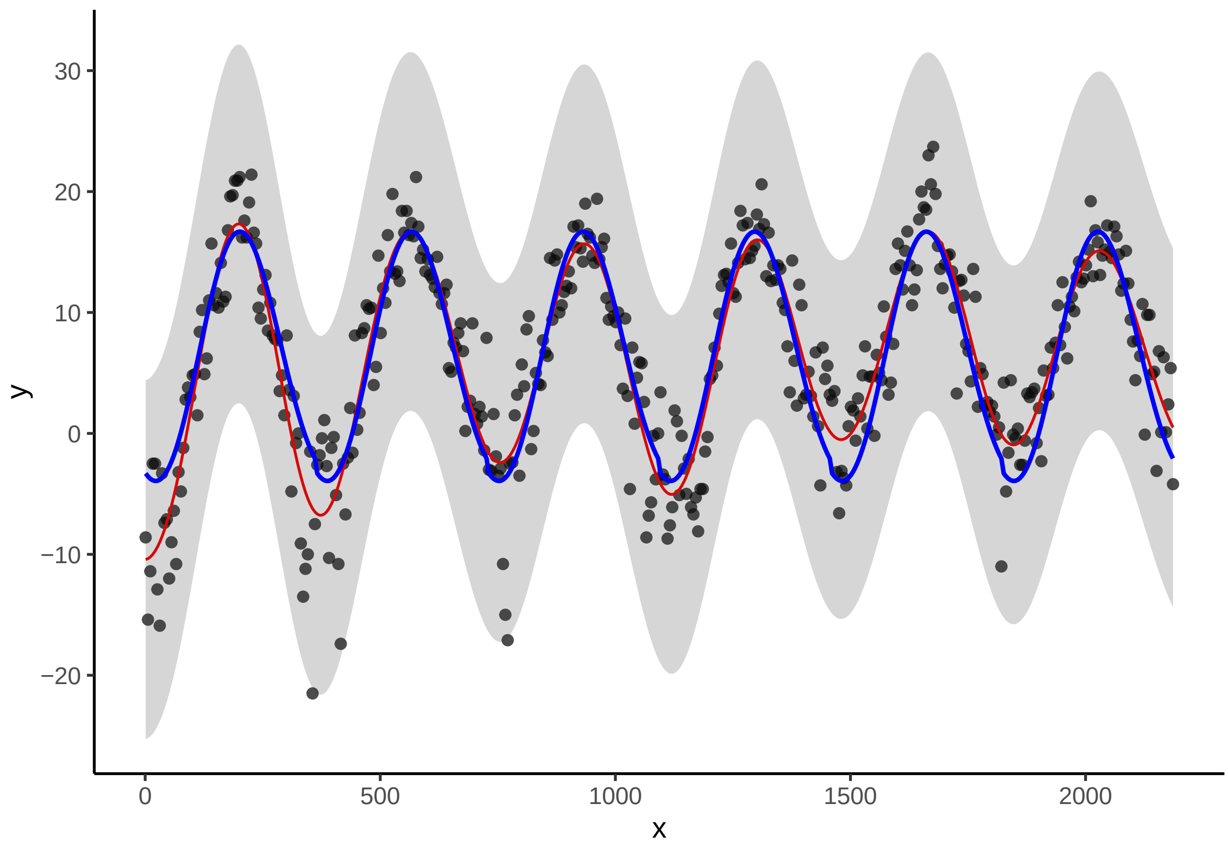
```r
C2 <- ggplot() +

  geom_point(
```

```
    aes(x = x, y = y),
    data = plot_it1,
    col = "black",
    alpha = 0.7) +

  geom_line(
    aes(x = xGrid, y = predict(gp_tullinge)),
    data = band_it1,
    alpha = 1,
    col = "red") +

  geom_ribbon(
    aes(ymin = low2c, ymax = upp2c, x = xGrid),
    data = band_it1,
    alpha = 0.2) +

  theme_classic()
#plot(x=band_it1$xGrid, y=band_it1$fbar, type = "l")
C2
```




**d)**

```
a2d_kern <- kern_maker(l = 1.2, sigmaf = 20 )
gp_tullinge_d <- gausspr(x = tullinge$day,
                         y = tullinge$temp,
                         kernel = a2d_kern,
                         var = sigma_2n)
C23 <- C2 + geom_line(aes(x= tullinge$time,y = predict(gp_tullinge_d)), col = "blue", size = 0.8)
#geom_point(data = tullinge, aes(x= time, y = temp)) +


C23
```




```
# plot(y = tullinge$temp, x = tullinge$day)
# #lines(x = tullinge$time, y = fitted(lm_tull), col = "red")
# lines(x = tullinge$day, y = predict(gp_tullinge_d), col = "red" , lwd = 1)
```

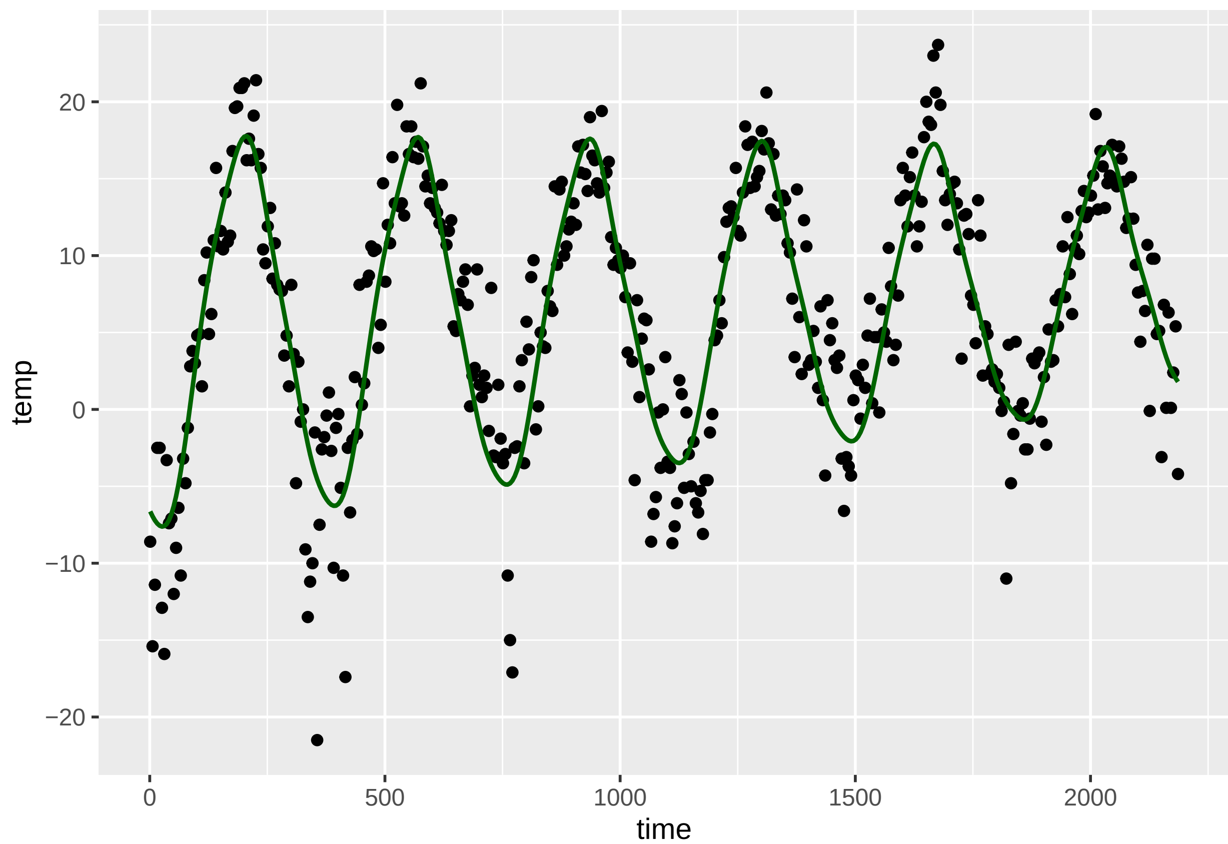
The process model after time has an advantage in that sense that you can capture a trend isolated to a specific time since your modeling using the closest observations in time rather than the day model that assumes that the clostes related temperature point is the one on the same day previous years.

## e)

```r
# periodic_kernel <- function(x,xi,sigmaf,d, l_1, l_2){
#
#   part1 <- exp(2 * sin(pi * abs(x - xi) / d)^2 / l_1^2 )
#   part2 <- exp(-0.5 * abs(x - xi)^2 / l_2)
#
#   sigmaf^2 * part1 * part2
#
# }
kern_maker2 <- function(sigmaf,d, l_1, l_2){

  periodic_kernel <- function(x,y = NULL){

    part1 <- exp(-2 * sin(pi * abs(x - y) / d)^2 / l_1^2 )
    part2 <- exp(-0.5 * abs(x - y)^2 / l_2^2)

    sigmaf^2 * part1 * part2

  }

  class(periodic_kernel) <- "kernel"
  return(periodic_kernel)
}
```

```r
sigmaff <- 20
l1 <- 1
l2 <- 10
d_est <- 365 / sd(tullinge$time)

periodic_kernel <- kern_maker2(sigmaf = sigmaff,
                               d = d_est ,
                               l_1 = l1,
                               l_2 = l2)
gp_tullinge_et <- gausspr(x = tullinge$time,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
gp_tullinge_ed <- gausspr(x = tullinge$day,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
```

```r
ggplot(data = tullinge, aes(x= time, y = temp)) +
  geom_point() +
  geom_line(aes(y = predict(gp_tullinge_et)), col = "darkgreen", size = 0.8)
```




This kernel seems to catch both variation in day and over time.

## Assignment 3

```
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)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
```

### a)

```
colnames(data)
```

```
## [1] "varWave"     "skewWave"    "kurtWave"    "entropyWave" "fraud"
```

```
GPfitFraud <- gausspr(fraud ~  varWave + skewWave, data = train)
```

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

```r
GPfitFraud
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.2043047635594
##
## Number of training instances learned : 1000
## Train error : 0.068
```

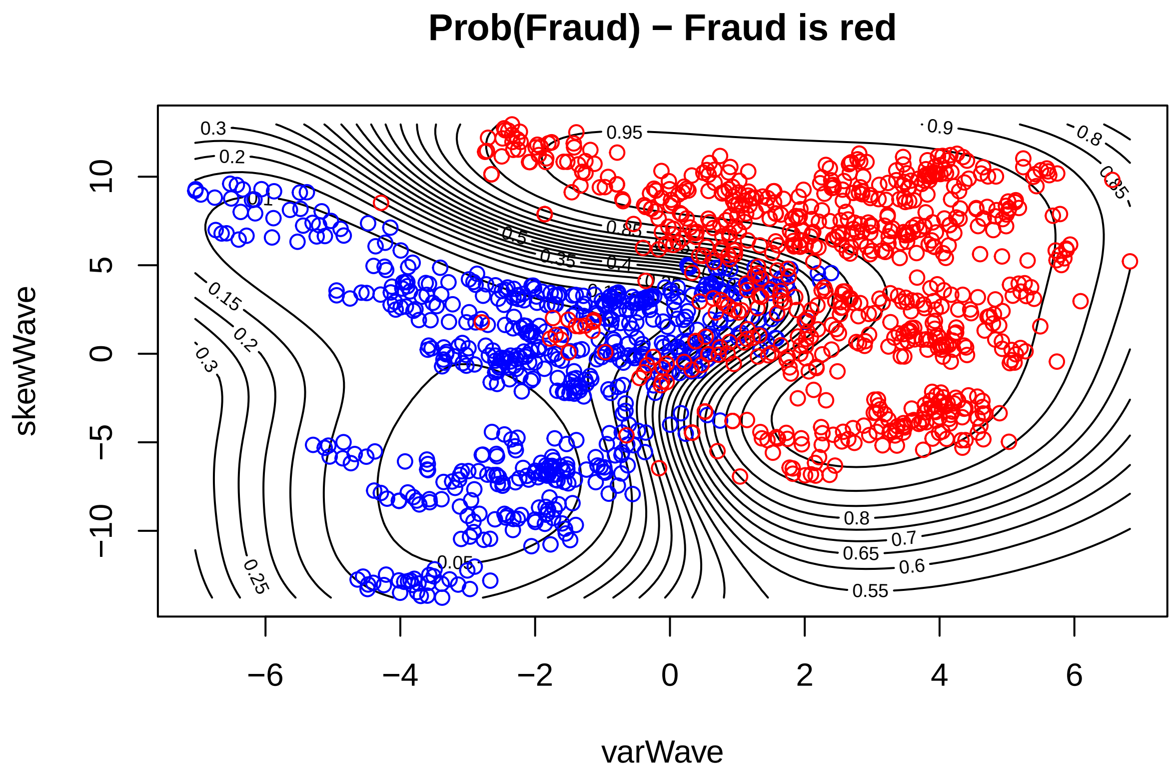
```r
# predict on the test set
fit_train<- predict(GPfitFraud,train[,c("varWave","skewWave")])
table(fit_train, train$fraud) # confusion matrix
```

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

```r
mean(fit_train == train$fraud)
```

```
## [1] 0.932
```

```r
# probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length=100)
x2 <- seq(min(data[,"skewWave"]),max(data[,"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(GPfitFraud, gridPoints, type="probabilities")
contour(x1,x2,t(matrix(probPreds[,1],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is red')
points(data[data[,5]== 1,"varWave"],data[data[,5]== 1,"skewWave"],col="blue")
points(data[data[,5]== 0,"varWave"],data[data[,5]== 0,"skewWave"],col="red")
```




**b)**

```r
# predict on the test set
fit_test<- predict(GPfitFraud,test[,c("varWave","skewWave")])
table(fit_test, test$fraud) # confusion matrix
```

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

```r
mean(fit_test == test$fraud)
```

```
## [1] 0.9354839
```

**c)**

```r
GPfitFraudFull <- gausspr(fraud ~ ., data = train)
```

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

```
GPfitFraudFull
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.399933221120042
##
## Number of training instances learned : 1000
## Train error : 0.004
```

```
# predict on the test set
fit_Full<- predict(GPfitFraudFull,test[,-ncol(test)])
table(fit_Full, test$fraud) # confusion matrix
```

```
##
## fit_Full   0   1
##        0 205   0
##        1   1 166
```

```
mean(fit_Full == test$fraud)
```

```
## [1] 0.9973118
```

```
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
library(AtmRay)
library(ggplot2)
tullinge  <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulli
data <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
                 header=FALSE, sep=',')
# The kernel function
exp_kern <- function(x,xi,l, sigmaf ){

  return((sigmaf^2)*exp(-0.5*( (x - xi) / l )^2))
}
# The implementation, can take a custom kernel of any class.
linear_gp <- function(x,y,xStar,hyperParam,sigmaNoise,kernel){
  n <- length(x)
  kernel_f <- kernel
  # K = Covariance matrix calculation
  K <- function(X, XI,...){

    kov <- matrix(0,nrow = length(X), ncol = length (XI))

    for(i in 1:length(XI)){

      kov[,i]<- kernel_f(X,XI[i],...)

    }
    return(kov)
```

```
  }
  l <-hyperParam[1]
  sigmaf <- hyperParam[2]
  #K(X,X)
  K_xx   <- K(x,x, l = l, sigmaf = sigmaf) #, kernel = exp_kern
  #K(X*,X*)
  K_xsxs <- K(xStar,xStar, l = l, sigmaf = sigmaf) # kernel = exp_kern,
  #K(X,X*)
  K_xxs  <- K(x,xStar, l = l, sigmaf = sigmaf)  #kernel = exp_kern,
  # Algorithm in page 19 of the Rasmus/Williams book
  sI <- sigmaNoise^2 * diag(dim(as.matrix(K_xx))[1])
  # L is transposed according to a definition in the R & W book
  L_transposed <- chol(K_xx + sI)
  L <- t(L_transposed)

  alpha <- solve(t(L), solve(L,y))
  f_bar_star <- t(K_xxs) %*% alpha
  v <- solve(L,K_xxs)
  V_fs <- K_xsxs - t(v) %*% v
  log_mlike <- -0.5 %*% t(y) %*% alpha - sum( diag(L) - n/2 * log(2*pi) )
  return(list(fbar = f_bar_star, vf = V_fs, log_post= log_mlike))
}
# Utility function for the tasks
plot_gp<- function(plot_it,band_it){
  ggplot() +

    geom_point(
      aes(x = x, y = y),
      data = plot_it,
      col = "blue",
      alpha = 0.7) +

    geom_line(
      aes(x = xs, y = fbar),
      data = band_it,
      alpha = 0.50) +

    geom_ribbon(
      aes(ymin = low, ymax = upp, xs),
      data = band_it,
      alpha = 0.15) +

    theme_classic()

}
# The data given
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
# The noise
sn <- 0.1
# The training grid
xs <- seq(-1,1,0.01)
# Hyperparameters l an sigma
```

```r
hyperParam <- c(0.3, 1)
# Another utility function
repeter <- function(x,y,xs,sn,hyperParam,kernel){
  res <- linear_gp(x,y,xs,hyperParam,sn,kernel)
  # If you want the prediction band just add the noise variance (ie the sigma_n)
  upp <- res$fbar + 1.96*sqrt(diag(res$vf))
  low <- res$fbar - 1.96*sqrt(diag(res$vf))
  plot_it <- data.frame(x = x, y = y)
  band_it <- data.frame(xGrid = xs, fbar = res$fbar,  upp = upp, low = low)
  plot_gp(plot_it,band_it)
}
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
sn <- 0.1
xs <- seq(-1,1,0.01)
hyperParam <- c(1, 1)
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
tullinge$time <- 1:nrow(tullinge)
tullinge$day <- rep(1:365,6)
time_sub <- tullinge$time %in% seq(1,2190,5)
tullinge <- tullinge[time_sub,]
kern_maker <- function(l,sigmaf){

  exp_k <- function(x,y = NULL){

    return((sigmaf^2)*exp(-0.5*( (x - y) / l )^2))
  }

  class(exp_k) <- "kernel"
  return(exp_k)
}
# gausspr()
# kernelMatrix()
ell <- 1
# SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdo (which is the SE kernel)
# SEkernel(1,2)
my_exp <- kern_maker(l = 10, sigmaf =20)
x <- c(1,3,4)
x_star <- c(2,3,4)
#my_exp(x,x_star)
kernelMatrix(my_exp,x,x_star)
lm_tull <- lm(temp ~ time + I(time^2), data = tullinge)
sigma_2n <- var(resid(lm_tull))
a2b_kern <- kern_maker(l = 0.2, sigmaf = 20 )
gp_tullinge <- gausspr(x = tullinge$time,
                       y = tullinge$temp,
                       kernel = a2b_kern,
                       var = sigma_2n)
```

```r
sn_2c <- sqrt(sigma_2n)
xs_2c <- tullinge$time
hyperParam_2c <- c(0.2, 20)
res_2c<- linear_gp(x = tullinge$time,
                   y = tullinge$temp,
                   xStar = xs_2c,
                   sigmaNoise  = sn_2c,
                   hyperParam = hyperParam_2c,
                   kernel = exp_kern)
upp2c <- predict(gp_tullinge) + 1.96*sqrt(diag(res_2c$vf))
low2c <- predict(gp_tullinge) - 1.96*sqrt(diag(res_2c$vf))
plot_it1 <- data.frame(x = tullinge$time, y = tullinge$temp)
band_it1 <- data.frame(xGrid = xs_2c, fbar = res_2c$fbar,  upp = upp2c, low = low2c)
C2 <- ggplot() +

  geom_point(
    aes(x = x, y = y),
    data = plot_it1,
    col = "black",
    alpha = 0.7) +

  geom_line(
    aes(x = xGrid, y = predict(gp_tullinge)),
    data = band_it1,
    alpha = 1,
    col = "red") +

  geom_ribbon(
    aes(ymin = low2c, ymax = upp2c, x = xGrid),
    data = band_it1,
    alpha = 0.2) +

  theme_classic()
#plot(x=band_it1$xGrid, y=band_it1$fbar, type = "l")
C2
a2d_kern <- kern_maker(l = 1.2, sigmaf = 20 )
gp_tullinge_d <- gausspr(x = tullinge$day,
                         y = tullinge$temp,
                         kernel = a2d_kern,
                         var = sigma_2n)
C23 <- C2 + geom_line(aes(x= tullinge$time,y = predict(gp_tullinge_d)), col = "blue", size = 0.8)
#geom_point(data = tullinge, aes(x= time, y = temp)) +


C23
# plot(y = tullinge$temp, x = tullinge$day)
# #lines(x = tullinge$time, y = fitted(lm_tull), col = "red")
# lines(x = tullinge$day, y = predict(gp_tullinge_d), col = "red" , lwd = 1)
# periodic_kernel <- function(x,xi,sigmaf,d, l_1, l_2){
#
#   part1 <- exp(2 * sin(pi * abs(x - xi) / d)^2 / l_1^2 )
#   part2 <- exp(-0.5 * abs(x - xi)^2 / l_2)
#
```

```r
#   sigmaf^2 * part1 * part2
#
# }
kern_maker2 <- function(sigmaf,d, l_1, l_2){

  periodic_kernel <- function(x,y = NULL){

    part1 <- exp(-2 * sin(pi * abs(x - y) / d)^2 / l_1^2 )
    part2 <- exp(-0.5 * abs(x - y)^2 / l_2^2)

    sigmaf^2 * part1 * part2

  }

  class(periodic_kernel) <- "kernel"
  return(periodic_kernel)
}
sigmaff <- 20
l1 <- 1
l2 <- 10
d_est <- 365 / sd(tullinge$time)

periodic_kernel <- kern_maker2(sigmaf = sigmaff,
                               d = d_est ,
                               l_1 = l1,
                               l_2 = l2)
gp_tullinge_et <- gausspr(x = tullinge$time,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
gp_tullinge_ed <- gausspr(x = tullinge$day,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
ggplot(data = tullinge, aes(x= time, y = temp)) +
  geom_point() +
  geom_line(aes(y = predict(gp_tullinge_et)), col = "darkgreen", size = 0.8)
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)
train <- data[SelectTraining,]
test <- data[-SelectTraining,]
colnames(data)
GPfitFraud <- gausspr(fraud ~  varWave + skewWave, data = train)
GPfitFraud
# predict on the test set
fit_train<- predict(GPfitFraud,train[,c("varWave","skewWave")])
table(fit_train, train$fraud) # confusion matrix
mean(fit_train == train$fraud)
# probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length=100)
x2 <- seq(min(data[,"skewWave"]),max(data[,"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(GPfitFraud, gridPoints, type="probabilities")
contour(x1,x2,t(matrix(probPreds[,1],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is red')
points(data[data[,5]== 1,"varWave"],data[data[,5]== 1,"skewWave"],col="blue")
points(data[data[,5]== 0,"varWave"],data[data[,5]== 0,"skewWave"],col="red")
# predict on the test set
fit_test<- predict(GPfitFraud,test[,c("varWave","skewWave")])
table(fit_test, test$fraud) # confusion matrix
mean(fit_test == test$fraud)
GPfitFraudFull <- gausspr(fraud ~ ., data = train)
GPfitFraudFull
# predict on the test set
fit_Full<- predict(GPfitFraudFull,test[,-ncol(test)])
table(fit_Full, test$fraud) # confusion matrix
mean(fit_Full == test$fraud)
# Question 4: SSMs

# Kalman filter implementation.

set.seed(12345)
start_time <- Sys.time()

T<-10000
mu_0<-50
Sigma_0<-10
R<-1
Q<-5

x<-vector(length=T)
z<-vector(length=T)
err<-vector(length=T)

for(t in 1:T){
  x[t]<-ifelse(t==1,rnorm(1,mu_0,Sigma_0),x[t-1]+1+rnorm(1,0,R))
  z[t]<-x[t]+rnorm(1,0,Q)
}

mu<-mu_0
Sigma<-Sigma_0*Sigma_0 # KF uses covariances
for(t in 2:T){
  pre_mu<-mu+1
  pre_Sigma<-Sigma+R*R # KF uses covariances
  K<-pre_Sigma/(pre_Sigma+Q*Q) # KF uses covariances
  mu<-pre_mu+K*(z[t]-pre_mu)
  Sigma<-(1-K)*pre_Sigma

  err[t]<-abs(x[t]-mu)
```

```
  cat("t: ",t,", x_t: ",x[t],", E[x_t]: ",mu," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time

# Repetition with the particle filter.

set.seed(12345)
start_time <- Sys.time()

T<-10000
n_par<-100
tra_sd<-1
emi_sd<-5
mu_0<-50
Sigma_0<-10

ini_dis<-function(n){
  return (rnorm(n,mu_0,Sigma_0))
}

tra_dis<-function(zt){
  return (rnorm(1,mean=zt+1,sd=tra_sd))
}

emi_dis<-function(zt){
  return (rnorm(1,mean=zt,sd=emi_sd))
}

den_emi_dis<-function(xt,zt){
  return (dnorm(xt,mean=zt,sd=emi_sd))
}

z<-vector(length=T)
x<-vector(length=T)

for(t in 1:T){
  z[t]<-ifelse(t==1,ini_dis(1),tra_dis(z[t-1]))
  x[t]<-emi_dis(z[t])
}

err<-vector(length=T)

bel<-ini_dis(n_par)
w<-rep(1/n_par,n_par)
for(t in 2:T){
  com<-sample(1:n_par,n_par,replace=TRUE,prob=w)
  bel<-sapply(bel[com],tra_dis)
```

```
  for(i in 1:n_par){
    w[i]<-den_emi_dis(x[t],bel[i])
  }
  w<-w/sum(w)

  Ezt<-sum(w * bel)
  err[t]<-abs(z[t]-Ezt)

  cat("t: ",t,", z_t: ",z[t],", E[z_t]: ",Ezt," , error: ",err[t],"\n")
  flush.console()
}

mean(err[2:T])
sd(err[2:T])

end_time <- Sys.time()
end_time - start_time

# KF works optimally (i.e. it computes the exact belief function in closed-form) since the SSM sampled
# linear-Gaussian. The particle filter on the other hand is approximate. The more particles the closer
# performance to the KF's but at the cost of increasing the running time.

# GPs
#source('KernelCode.R') # Reading the Matern32 kernel from file

Matern32 <- function(sigmaf = 0.5, ell= 0.5)
{
  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)
}


# Testing our own defined kernel function.
X <- matrix(rnorm(12), 4, 3) # Simulating some data
Xstar <- matrix(rnorm(15), 5, 3)
MaternFunc = Matern32(sigmaf = 1, ell = 2) # MaternFunc is a kernel FUNCTION
MaternFunc(c(1,1),c(2,2)) # Evaluating the kernel in x=c(1,1), x'=c(2,2)
# Computing the whole covariance matrix K from the kernel.
K <- kernlab::kernelMatrix(kernel = MaternFunc, x = X, y = Xstar) # So this is K(X,Xstar)

sigma2f = 1
ell = 0.5
zGrid <- seq(0.01, 1, by = 0.01)
count = 0
covs = rep(0,length(zGrid))
for (z in zGrid){
  count = count + 1
  covs[count] <- sigma2f*MaternFunc(0,z)
}
```

```r
plot(zGrid, covs, type = "l", xlab = "ell")

# The graph plots Cov(f(0),f(z)), the correlation between two FUNCTION VALUES, as a function of the dis
# two inputs (0 and z)
# As expected the correlation between two points on f decreases as the distance increases.
# The fact that points of f are dependent, as given by the covariance in the plot, makes the curves smo
# have nearby outputs when the correlation is large.

sigma2f = 0.5
ell = 0.5
zGrid <- seq(0.01, 1, by = 0.01)
count = 0
covs = rep(0,length(zGrid))
for (z in zGrid){
  count = count + 1
  covs[count] <- sigma2f*MaternFunc(0,z)
}
plot(zGrid, covs, type = "l", xlab = "ell")

# Changing sigma2f will have not effect on the relative covariance between points on the curve, i.e. wi
# smoothness. But lowering sigma2f makes the whole covariance curve lower. This means that the variance
# distribution over curves which is tighter (lower variance) around the mean function of the GP. This m
# from the GP will be less variable.

####################################
### GP inference with the ell = 1
####################################

library(kernlab)
#load("LidarData.txt")
# loading the data
LidarData <- read.delim2("C:/Users/Omkar/Downloads/Adv ML/LidarData.txt",header = TRUE, sep = "")
sigmaNoise = 0.05
x = LidarData$Distance
y = LidarData$LogRatio

# Set up the kernel function
kernelFunc <- Matern32(sigmaf = 1, ell = 1)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, xs) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernlab::kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernlab::kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernlab::kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)
```

```r
# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive in
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)

####################################
### GP inference with the ell = 5
####################################

load("lidar.RData") # loading the data
sigmaNoise = 0.05
x = distance
y = logratio

# Set up the kernel function
kernelFunc <- k(sigmaf = 1, ell = 5)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, xs) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive in
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)
```

```
# Discussion
# The larger length scale gives smoother fits. The smaller length scale seems to generate too jagged fi
# PF should outperform KF because the model is not liner-Gaussian, i.e. it is a mixture model.

# Question 4.1 (Hyperparameter selection via log marginal maximization)

tempData <- read.csv('https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulli
temp <- tempData$temp
time = 1:length(temp)

# Extract every 5:th observation
subset <- seq(1, length(temp), by = 5)
temp <- temp[subset]
time = time[subset]

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

SEKernel2 <- function(par=c(20,0.2),x1,x2){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- (par[1]^2)*exp(-0.5*( (x1-x2[i])/par[2])^2 )
  }
  return(K)
}

LM <- function(par=c(20,0.2),X,y,k,sigmaNoise){
  n <- length(y)
  L <- t(chol(k(par,X,X)+((sigmaNoise^2)*diag(n))))
  a <- solve(t(L),solve(L,y))
  logmar <- -0.5*(t(y)%*%a)-sum(diag(L))-(n/2)*log(2*pi)
  return(logmar)
}

bestLM<-LM(par=c(20,0.2),X=scale(time),y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit) # Grid searc
bestLM
besti<-20
bestj<-0.2
for(i in seq(1,50,1)){
  for(j in seq(0.1,10,0.1)){
    aux<-LM(par=c(i,j),X=scale(time),y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit)
    if(bestLM<aux){
      bestLM<-aux
      besti<-i
      bestj<-j
    }
  }
}
bestLM
besti
bestj
```

```r
foo<-optim(par = c(1,0.1), fn = LM, X=scale(time),y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit, mo
           lower = c(.Machine$double.eps, .Machine$double.eps),control=list(fnscale=-1)) # Alternatively

foo$value
foo$par[1]
foo$par[2]

# Question 4.2 (Hyperparameter selection via validation set)

library(kernlab)

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)
y <- data[,5]
X <- as.matrix(data[,1:4])
yTrain <- y[SelectTraining]
yTest <- y[-SelectTraining]
XTrain <- X[SelectTraining,]
XTest <- X[-SelectTraining,]
SelectVal <- sample(1:1000, size = 200, replace = FALSE) # 800 samples for training, 200 for validation
yVal <- yTrain[SelectVal]
XVal <- XTrain[SelectVal,]
yTrain <- yTrain[-SelectVal]
XTrain <- XTrain[-SelectVal,]

acVal <- function(par=c(0.1)){ # Accuracy on the validation set
  gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = list(sigma=par[1]))
  predVal <- predict(GPfitFraud,XVal[,selVars])
  table(predVal, yVal)
  accuracyVal <-sum(predVal==yVal)/length(yVal)
  return(accuracyVal)
}

selVars <- c(1,2,3,4)
GPfitFraud <- gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = 'automatic')
GPfitFraud
predVal <- predict(GPfitFraud,XVal[,selVars])
table(predVal, yVal)
accuracyVal <-sum(predVal==yVal)/length(yVal)
accuracyVal

bestVal<-accuracyVal # Grid search
for(j in seq(0.1,10,0.1)){
  aux <- acVal(j)
  if(bestVal<aux){
    bestVal<-aux
    bestj<-j
  }
}
bestVal
bestj
```

```r
gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = list(sigma=bestj)) # Accuracy on th
predTest <- predict(GPfitFraud,XTest[,selVars])
table(predTest, yTest)
sum(predTest==yTest)/length(yTest)

foo<-optim(par = c(0.1), fn = acVal, method="L-BFGS-B",
           lower = c(.Machine$double.eps),control=list(fnscale=-1)) # Alternatively, one can use optim

acVal(foo$par)
foo$par

gausspr(x = XTrain[,selVars], y = yTrain, kernel = "rbfdot", kpar = list(sigma=foo$par)) # Accuracy on
predTest <- predict(GPfitFraud,XTest[,selVars])
table(predTest, yTest)
sum(predTest==yTest)/length(yTest)
```