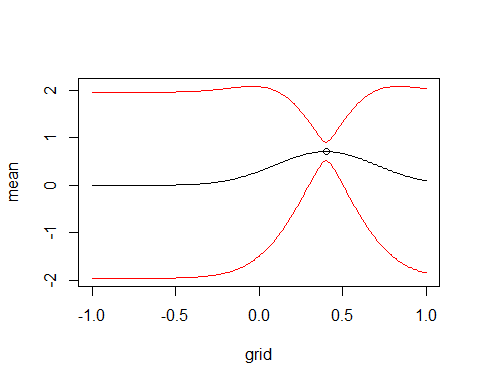
Report.R

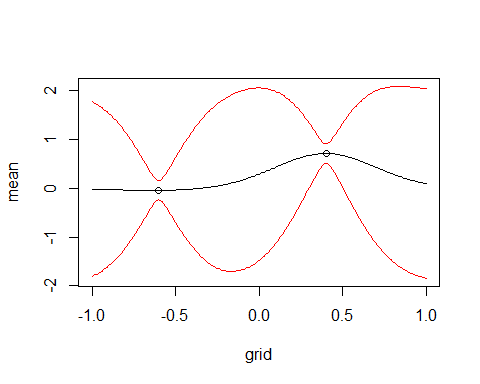
Arun

2020-10-17

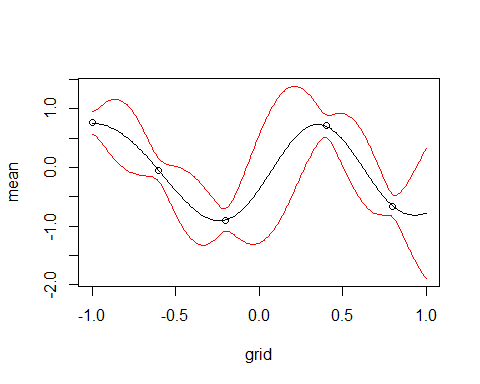
library(kernlab)  
  
  
  
### Assignment 1###   
  
# Covariance function  
# The function takes in input values x1, x2 and computes the exponential kernel which gives  
# the resulting covariance between the two input values  
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){  
 n1 <- length(x1)  
 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, hyperParameter){  
 #n is how many functions we will need, as many as the nr of inputs  
 n = length(X)  
   
 #Compute the covarance matrix [K = K(X,X)]  
 K = SquaredExpKernel(X,X, hyperParameter[1], hyperParameter[2])  
 sigmaNoise = sigmaNoise^2  
   
 #Compute L by using the cholesky decomposition  
 L\_trans = chol(K + sigmaNoise\*diag(n))  
 #Need to take the transpose since L in the algorithm is a lower triangular matrix where as  
 # the R function returns an uper triangular matrix  
 L = t(L\_trans)  
   
 ### Predictive mean f\_bar\*  
 ##Comute the predictive mean by solving the equations  
 # L\y means the vector x that solves the equation Lx = y. On paper it can be solved by  
 # multiplying by the inverse but is better solved by using the function solve  
 # [alpha = t(L)\(L\y)]  
 alpha = solve(L\_trans, solve(L,y))  
   
 ##Compute f\_bar\*  
 #f\_bar\* = alpha \* t(K\*)  
 # [K\* = K(X, X\*) => t(K\*) = K(X\*, X)]  
 K\_X\_Xstar = SquaredExpKernel(X, XStar, hyperParameter[1], hyperParameter[2])  
 f\_bar\_star = t(K\_X\_Xstar) %\*% alpha  
   
 ### Predictive variance f\_star  
 #Compute v, [v = L\K\*]  
 v = solve(L, K\_X\_Xstar)  
   
 ## Compute the variance of f\*  
 #V[f\_star] = K(X\*, X\*) - t(v)\*v  
 # First need to compute K[X\*, X\*]  
 K\_XStar\_XStar = SquaredExpKernel(XStar, XStar, hyperParameter[1], hyperParameter[2])  
 #Compute V\_f\*  
 v\_f\_star = K\_XStar\_XStar - t(v) %\*% v  
 #To draw from the posterior we only need the variance of f  
 v\_f\_star = diag(v\_f\_star)  
   
 result = list("Predictive mean" = f\_bar\_star,  
 "Predicitive variance" = v\_f\_star)  
}  
  
  
### Part 1.2  
# Let the hyperparameters be the following: sigmaf = 1, el = 0,3, using a singel observation (x,y) = (0.4, 0.719), sigmanoise = 0.1  
# Plot the posterior  
  
  
#Function for plotting the result  
plotGP = function(mean,variance,grid,x,y){  
 plot(grid,mean,ylim = c(min(mean-1.96\*sqrt(variance))  
 ,max(mean+1.96\*sqrt(variance))),  
 type = "l")  
 lines(grid,  
 mean+1.96\*sqrt(variance),   
 col = "red")  
 lines(grid,  
 mean-1.96\*sqrt(variance),   
 col = "red")  
 points(x,y)  
}  
  
sigmaF = 1  
ell = 0.3  
obs = data.frame(0.4, 0.719)  
sigmanoise = 0.1  
xGrid = seq(-1,1,0.01)  
  
GP = PosteriorGP(obs[,1], obs[,2], xGrid, sigmanoise,c(sigmaF, ell))  
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs[1,1], obs[1,2])



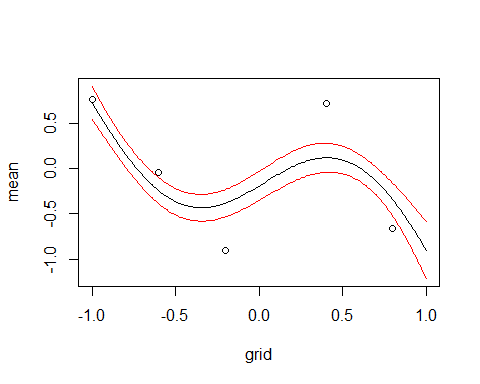
### Part 1.3  
#Update the posterior with another observation (x,y) = (-0.6, -0.044) and plot the posterior mean of f and the probability bands  
  
  
  
newobs = c(-0.6, -0.044)  
obs\_1.3 = rbind(obs, newobs)  
GP = PosteriorGP(obs\_1.3[,1], obs\_1.3[,2], xGrid, sigmanoise,c(sigmaF, ell))  
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs\_1.3[,1], obs\_1.3[,2])



### Part 1.4  
#Compute now the posterior distirbution of f using all available data points  
  
obs\_1.4 = data.frame(x=c(-1,-0.6, -0.2, 0.4, 0.8), y=c(0.768, -0.044, -0.904, 0.719,-0.664))  
GP = PosteriorGP(obs\_1.4[,1], obs\_1.4[,2], xGrid, sigmanoise,c(sigmaF, ell))  
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs\_1.4[,1], obs\_1.4[,2])



#Part 1.5  
#Repeat the exercise, this time wiht hyperparameters sigmaf = 1 and ell = 1  
# Compare the results  
sigmaF = 1  
ell = 1  
GP = PosteriorGP(obs\_1.4[,1], obs\_1.4[,2], xGrid, sigmanoise,c(sigmaF, ell))  
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs\_1.4[,1], obs\_1.4[,2])



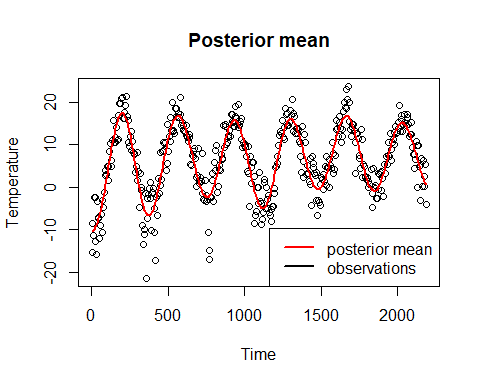
################################## Assignment 2##########################################  
################################## GP Regresssion with Kernlab###########################  
  
tempData = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/  
Code/TempTullinge.csv", header=TRUE, sep=";")  
  
library(kernlab)  
  
time = seq(1, nrow(tempData))  
  
day = c()  
counter = 1  
for (i in 1:nrow(tempData)){  
   
 if(counter > 365){  
 counter = 1  
 }  
   
 day = append(day, counter)  
 counter = counter +1  
   
}  
  
  
five\_sequence = seq(1, nrow(tempData), 5)  
time\_selection = time[five\_sequence]  
day\_selection = day[five\_sequence]  
temperature = tempData$temp  
temperature\_selection = temperature[five\_sequence]  
  
  
### 2.1  
# Define your own square exponential kernel function (with parameters ` (ell) and   
# σf (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  
  
x = 1  
x\_prime = 2  
X = c(1,3,4)  
X\_prime = c(2,3,4)  
  
SE\_Kernel <- function(sigmaF=1,l=1){  
 rval = function(x, y = NULL){  
 n1 <- length(x)  
 n2 <- length(y)  
 res = sigmaF^2\*exp(-0.5\*( (x-y)/l)^2 )  
 return(res)  
   
 }  
 class(rval) = "kernel"  
 return(rval)  
   
}  
  
## Compute the covariance for (x, x')  
# Initialize the kernel, values ell and sigmaF = 1  
kernel = SE\_Kernel()  
  
# Evalute the kernel in x = 1 and x' = 2  
covariance = kernel(x = 1, y = 2)  
covariance

## [1] 0.6065307

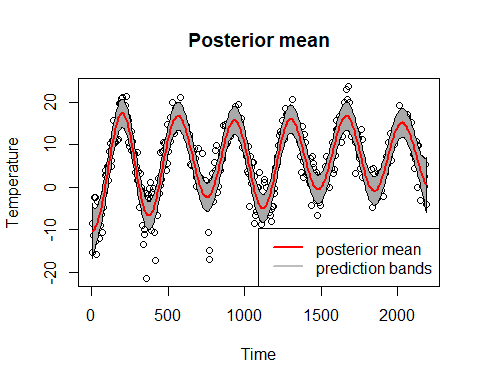
# Compute the covariance matrix for the input vectors X, X\_prime K[X, X\_prime]  
# X = c(1,3,4)  
# X\_prime = c(2,3,4)  
  
cov\_matrix = kernelMatrix(kernel = SE\_Kernel(), x = X, y = X\_prime)  
cov\_matrix

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

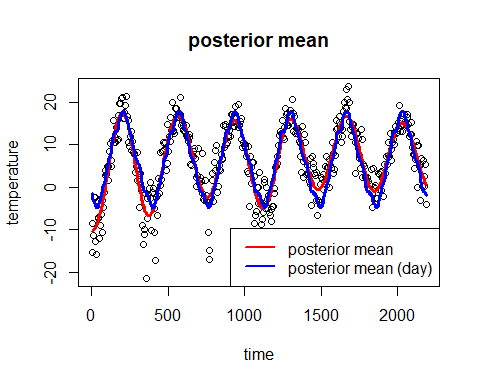
### 2.2  
#Consider the following model:  
#temp = f(time) + epsilon with epsilon ∼ N (0, σ2n) and f ∼ GP(0, k(time, time′))  
  
#Let σ2n be the residual variance from a simple quadratic regression fit (using the lm function in R).   
#Estimate the above Gaussian process regression model using the squaredexponential function from (1) with σf = 20 and ` = 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 σf and ` (no needto write this in the report though).  
  
sigmaf = 20  
ell = 0.2  
  
# Fit quadratic regression with the scaled data  
regression\_fit = lm(temperature\_selection ~ time\_selection + I(time\_selection)^2)  
  
#Compute the residual variance  
sigmaNoise = sd(regression\_fit$residuals)  
  
hyperparam = c(sigmaf, ell)  
  
#Compute GP regression  
  
GP\_fit = gausspr(x = time\_selection,   
 y = temperature\_selection,  
 kernel = SE\_Kernel(sigmaF = sigmaf, l = ell),  
 var = sigmaNoise^2)  
  
#Compute the posterior mean at every data point in the training  
# dataset  
meanPred = predict(GP\_fit, time\_selection)  
  
plot(time\_selection, temperature\_selection, main = "Posterior mean",  
 ylab = "Temperature", xlab = "Time")  
lines(time\_selection, meanPred, col = "red", lwd = 2)  
legend("bottomright", legend = c("posterior mean", "observations"),col=c("red", "black"),   
 lty = c(1, 1), lwd = c(2,2))



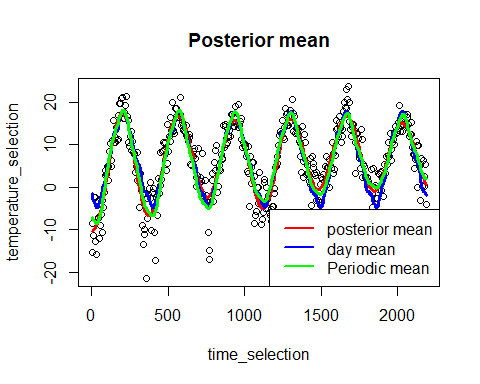
### 2.3  
# Make own computations to obtain the posterior variance of f and plot the  
# 95 % probability bands for f. To do this we can use the prviously computed  
# function PosteriorGP  
  
sigmaf = 20  
ell = 0.2  
hyperparam = c(sigmaf, ell)  
  
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){  
 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, hyperParameter){  
   
 n = length(X)  
 #Compute the covarance matrix [K = K(X,X)]  
 K = SquaredExpKernel(X,X, hyperParameter[1], hyperParameter[2])  
   
 L\_trans = chol(K + sigmaNoise^2\*diag(n))  
 #Need to take the transpose since L in the algorithm is a lower triangular matrix where as  
 # the R function returns an uper triangular matrix  
 L = t(L\_trans)  
 alpha = solve(L\_trans, solve(L,y))  
   
 ##Compute f\_bar\*  
 K\_X\_Xstar = SquaredExpKernel(X, XStar, hyperParameter[1], hyperParameter[2])  
   
 f\_bar\_star = t(K\_X\_Xstar) %\*% alpha  
   
 v = solve(L, K\_X\_Xstar)  
   
 K\_XStar\_XStar = SquaredExpKernel(XStar, XStar, hyperParameter[1], hyperParameter[2])  
 #Compute V\_f\*  
 v\_f\_star = K\_XStar\_XStar - t(v) %\*% v  
 #To draw from the posterior we only need the variance of f  
 v\_f\_star = diag(v\_f\_star)  
   
 result = list("Predictive mean" = f\_bar\_star,  
 "Predicitive variance" = v\_f\_star)  
}  
  
  
posterior = PosteriorGP(X = scale(time\_selection),  
 y = scale(temperature\_selection),  
 XStar = scale(time\_selection),  
 sigmaNoise = sigmaNoise,  
 hyperParameter = hyperparam)  
  
  
## Compute the variance for f  
posterior\_variance = posterior$`Predicitive variance`  
posterior\_mean = posterior$`Predictive mean`  
posterior\_mean = posterior\_mean\*sd(temperature\_selection) + mean(temperature\_selection)  
  
  
  
L = posterior\_mean - 1.96\*sqrt(posterior\_variance)  
U = posterior\_mean + 1.96\*sqrt(posterior\_variance)  
  
# Plot the meanPred, and the prediction bands for the posterior variance  
plot(time\_selection, temperature\_selection, main = "Posterior mean",  
 ylab = "Temperature", xlab = "Time")  
polygon(c(time\_selection, rev(time\_selection)),  
 c(L, rev(U)), col = "darkgray")  
lines(time\_selection, posterior\_mean, col = "red", lwd = 2)  
legend("bottomright", legend = c("posterior mean", "prediction bands"),col=c("red", "gray"),   
 lty = c(1, 1), lwd = c(2,2))



### Part 2.4 ###  
regression\_fit\_4 = lm(temperature\_selection ~ day\_selection + I(day\_selection)^2)  
sigmaNoise\_day = sd(regression\_fit\_4$residuals)  
  
ell = 0.2  
sigmaf = 20  
  
GP\_fit\_day = gausspr(x = day\_selection,  
 y = temperature\_selection,  
 kernel = SE\_Kernel(sigmaF = sigmaf, l = ell),  
 var = sigmaNoise\_day)  
  
meanPred\_day = predict(GP\_fit\_day, day\_selection)  
  
plot(time\_selection, temperature\_selection, main = "posterior mean",  
 ylab = "temperature", xlab = "time")  
points(time\_selection, temperature\_selection)  
lines(time\_selection, meanPred, col = "red", lwd = 3)  
lines(time\_selection, meanPred\_day, col = "blue", lwd = 3)  
legend("bottomright", legend = c("posterior mean", "posterior mean (day)"),col=c("red", "blue"),   
 lty = c(1, 1), lwd = c(2,2))



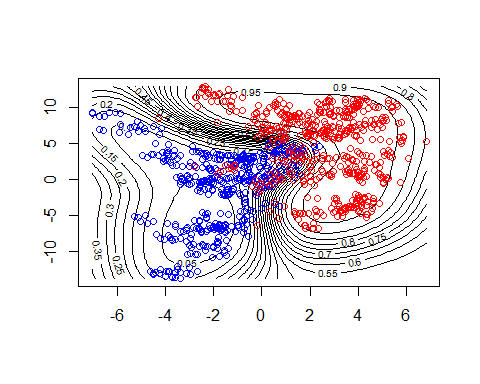
### Part 2.5 ###   
#Implement a generalization of the periodic kernel given in the lectures  
# Note that we have two different l which controls the correlation between  
# the same day in different years. Estimate the GP mmodel using the time variable  
# with this kernel and the hyperparamerers:  
  
sigmaf = 20   
ell1 = 1  
ell2 = 10  
d = 365/sd(time\_selection)  
  
#Create the periodic kernel  
periodic\_kernel = function(sigmaf, l1, l2, d){  
 Periodic\_K = function(x,y){  
 result = (sigmaf^2)\*exp(-2\*((sin(pi\*abs(x-y)/d)^2)/(l1^2)))\*  
 exp(-0.5\*((x-y)^2)/(l2^2))  
 return(result)  
 }  
 class(Periodic\_K) = "kernel"  
 return(Periodic\_K)  
}  
  
GP\_periodic = gausspr(x = time\_selection,  
 y = temperature\_selection,  
 kernel = periodic\_kernel(sigmaf, ell1, ell2, d),  
 var = sigmaNoise)  
  
meanPred\_periodic = predict(GP\_periodic, time\_selection)  
  
plot(time\_selection, temperature\_selection, main = "Posterior mean")  
lines(time\_selection, meanPred, col = "red", lwd = 3)  
lines(time\_selection, meanPred\_day, col = "blue", lwd = 3)  
lines(time\_selection, meanPred\_periodic, col = "green", lwd = 3)  
legend("bottomright", legend = c("posterior mean", "day mean", "Periodic mean"),col=c("red", "blue", "green"), lwd = c(2,2))



######################## Assignment 3 ############################################  
####################### GP Classicatio############################################  
#Assignment 3 - GP Classification with kernlab  
  
library(kernlab)  
library(AtmRay)  
  
data <-  
 read.csv(  
 "https://github.com/STIMALiU/AdvMLCourse/raw/master/  
GaussianProcess/Code/banknoteFraud.csv",  
 header = FALSE,  
 sep = ","  
 )  
names(data) <-  
 c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")  
  
data[, 5] <- as.factor(data[, 5])  
  
#Select 1000 datapoints  
set.seed(111)  
SelectTraining <-  
 sample(1:dim(data)[1], size = 1000, replace = FALSE)  
train = data[SelectTraining, ]  
test = data[-SelectTraining, ]  
  
# Part 3.1  
GP.fit = gausspr(fraud ~ varWave + skewWave, data = train)

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

x1 <- seq(min(train$varWave), max(train$varWave), length = 100)  
x2 <- seq(min(train$skewWave), max(train$skewWave), length = 100)  
gridPoints <- meshgrid(x1, x2)  
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))  
gridPoints <- data.frame(gridPoints)  
names(gridPoints) <- names(data)[1:2]  
probPreds <- predict(GP.fit, gridPoints, type = "probabilities")  
  
contour(x1, x2, matrix(probPreds[, 1], 100, byrow = TRUE), 20)  
points(train[train$fraud == 1, "varWave"], train[train$fraud == 1, "skewWave"], col = "blue")  
points(train[train$fraud == 0, "varWave"], train[train$fraud == 0, "skewWave"], col = "red")



### Part 3.2  
# Compute the accuracy on predicting the test data  
test\_prediction = predict(GP.fit, test, type = "response")  
accuracy = mean(test\_prediction == test$fraud)  
accuracy

## [1] 0.9247312

### Part 3.3  
GP.fit\_all = gausspr(fraud ~ ., data = train)

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

test\_prediction\_all = predict(GP.fit\_all, test, type = "response")  
accuracy\_all = mean(test\_prediction\_all == test$fraud)  
accuracy\_all

## [1] 0.9946237