Lab4

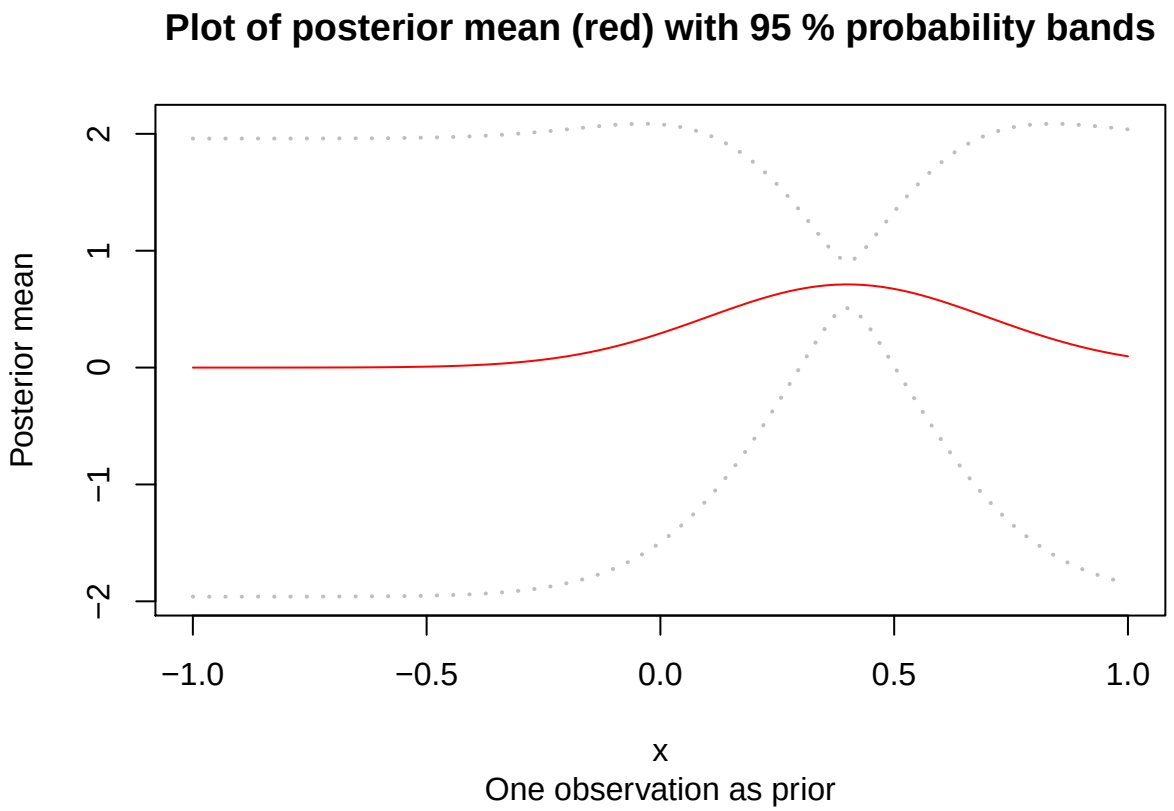
Christian von Koch

2020-10-16

Assignment 1

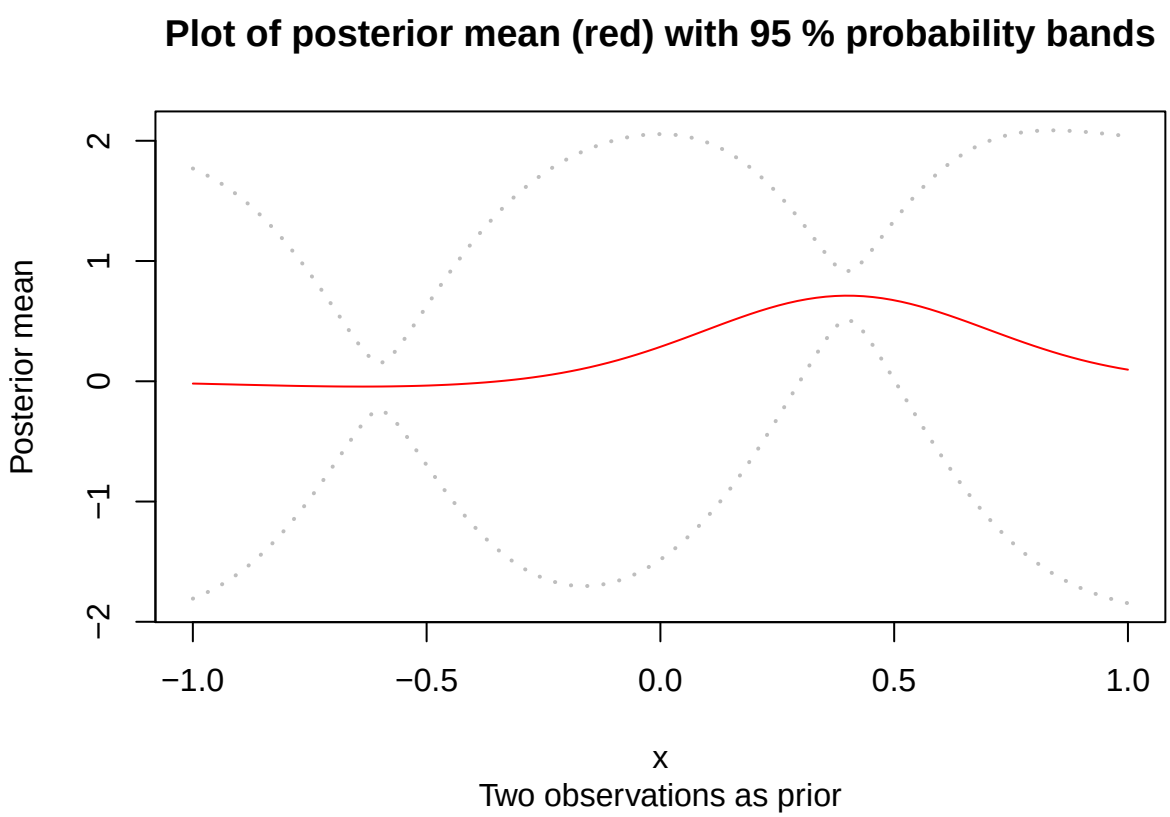
```
  library(mvtnorm)

# 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)
}

posteriorGP = function(X, y, XStar, sigmaNoise, k, ...) {
  n=length(X)
  K=k(X, X, ...)
  kStar=k(X,XStar, ...)
  L = t(chol(K + sigmaNoise^2*diag(n)))
  alpha=solve(t(L),solve(L, y))
  predMean=t(kStar)%*%alpha
  v=solve(L, kStar)
  predVar=k(XStar, XStar, ...)-t(v)%*%v
  return(list(mean=predMean, var=predVar))
}

# 2. Plot one draw from posterior

xTest = seq(-1, 1, length=100)

# Plotting one draw
obs=data.frame(x=0.4, y=0.719)
sigmaF <- 1
sigmaN=0.1
l <- 0.3
posteriorSim=posteriorGP(obs$x, obs$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim$mean, type="l",
     ylim=c(min(posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var))), max(posteriorSim$mean + 1.96*sq
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub="One observation as prior")
```

```
lines(xTest, posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim$mean + 1.96*sqrt(diag(posteriorSim$var)), col = "gray", lwd = 2, lty=21)
```



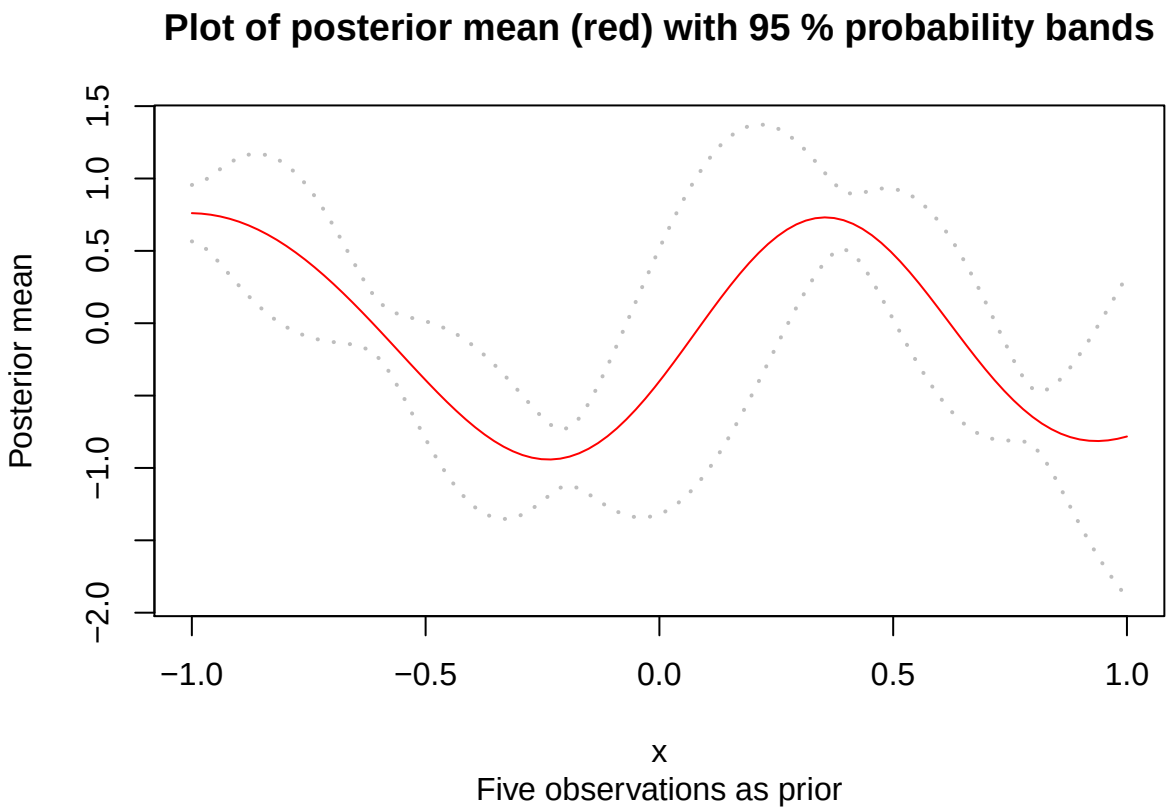
It is evident from the plot that the distribution obtained has tighter probability bands arount the point which was used as prior for the distribution.

```
x=c(0.4, -0.6)
y=c(0.719, -0.044)
obs2=data.frame(x=x, y=y)
posteriorSim2=posteriorGP(obs2$x, obs2$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim2$mean, type="l",
     ylim=c(min(posteriorSim2$mean-1.96*sqrt(diag(posteriorSim2$var))), max(posteriorSim2$mean+1.96*sqr
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub="Two observations as prior")
lines(xTest, posteriorSim2$mean - 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim2$mean + 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
```



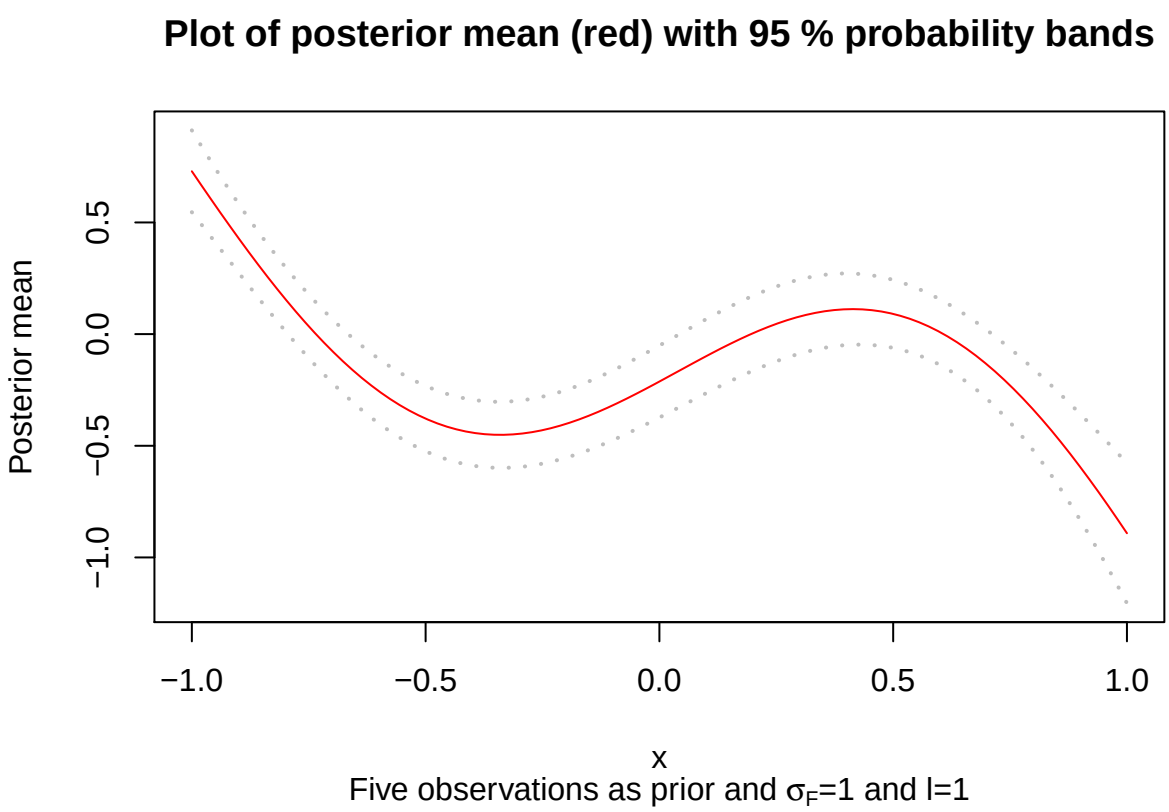
This gets even more notable when we have two points as prior as is the case in the plot above.

```
x=c(-1, -0.6, -0.2, 0.4, 0.8)
y=c(0.768, -0.044, -0.940, 0.719, -0.664)
obs2=data.frame(x=x, y=y)
posteriorSim2=posteriorGP(obs2$x, obs2$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim2$mean, type="l",
     ylim=c(min(posteriorSim2$mean-1.96*sqrt(diag(posteriorSim2$var))), max(posteriorSim2$mean+1.96*sqr
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub="Five observations as prior")
lines(xTest, posteriorSim2$mean - 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim2$mean + 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
```



The same pattern appears in this plot where we can note that the probable bands moves like a rollercoaster - getting tighter when closing in on an observation and getting wider whilst moving far away from observations.

```
sigmaF=1
l=1
posteriorSim2=posteriorGP(obs2$x, obs2$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim2$mean, type="l",
     ylim=c(min(posteriorSim2$mean-1.96*sqrt(diag(posteriorSim2$var))), max(posteriorSim2$mean+1.96*sqr
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub=expression(paste("Five observations as prior and ", sigma[F], "=1 and l=1")))
lines(xTest, posteriorSim2$mean - 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim2$mean + 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
```



In the plot above the smoothing parameter has been changed to 1. This means that more weight is given to data points further away from the data points in the prior. The result of this is expressed well in the plot above. The probability bands are more smoothly distributed along the posterior distribution which is due to that with the chosen value of smoothing parameter, we have enough observations to cover the investigated grid, i.e. data points in the whole grid will be weighted enough for the confidence to remain high.

Assignment 2

```
## GP regression with kernlab

# Fetch data from source

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")

# Restructuring data

data$date = as.Date(data$date, "%d/%m/%y")
time=seq(1:2190)
day=time %% 365
day[which(day == 0)] = 365
id = seq(from=1, to=2186, 5)
time=time[id]
day=day[id]
```

```r
## 1) Familirarization of the kernlab library. Calculating covariance matrix from two input vectors

# This is just to test how one evaluates a kernel function
# and how one computes the covariance matrix from a kernel function.
library(kernlab)
X <- as.vector(c(1,3,4)) # Simulating some data.
Xstar <- as.vector(c(2,3,4))
ell <- 1

# 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)
}

NestedSquaredExpKernel <- function(sigmaF=1,l=3){
  EvaluExpKernel = function(x, xStar) {
    n1 <- length(x)
    n2 <- length(xStar)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x-xStar[i])/l)^2 )
    }
    return(K)
  }
  class(EvaluExpKernel)='kernel'
  return(EvaluExpKernel)
}

SEkernel <- NestedSquaredExpKernel() # Note how I reparametrize the rbfdot (which is the SE kernel) in
print(paste("The kernel evaluated at (1,2) yields in the value", round(as.numeric(SEkernel(1,2)), 3)))
```

```
## [1] "The kernel evaluated at (1,2) yields in the value 0.946"
```

```r
# Computing the whole covariance matrix K from the kernel. Just a test.
print("... and the following is the covariate matrix:") # So this is K(X,Xstar)
```

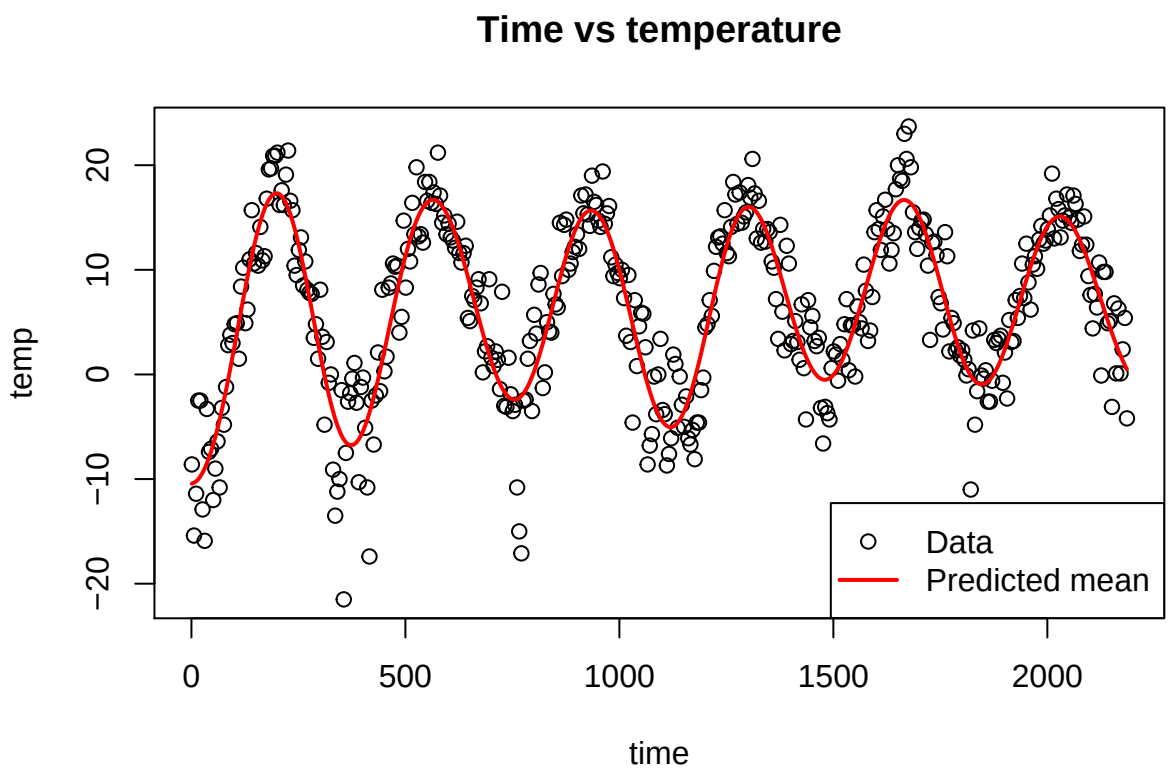
```
## [1] "... and the following is the covariate matrix:"
```

```r
kernelMatrix(kernel = SEkernel, x = X, y = Xstar)
```

```
## 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
```

```r
temp=data$temp[id]
fit = lm(temp ~ time+time^2)
sigmaN=sd(fit$residuals)
```

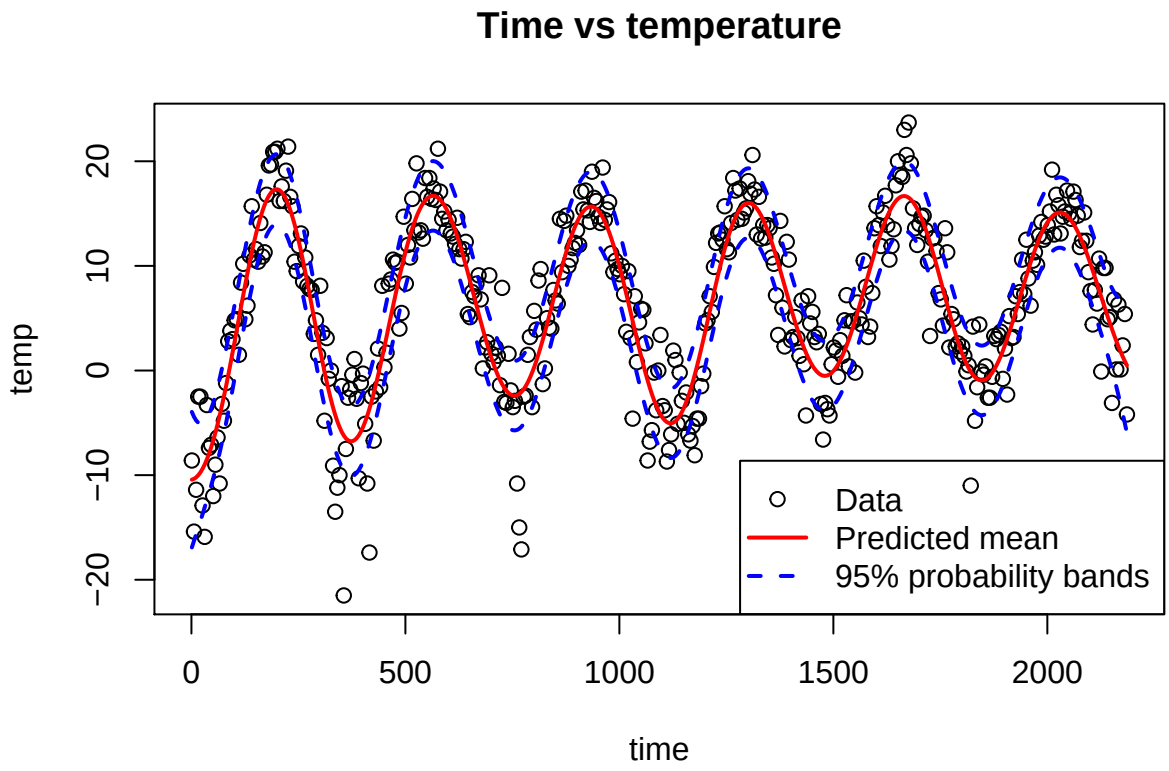
```
sigmaF=20
l=0.2
SEkernel = NestedSquaredExpKernel(sigmaF, l)
model = gausspr(time, temp, kernel=SEkernel, var=sigmaN^2)
predictedMean=predict(model, newdata=time)
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
legend("bottomright", legend=c("Data", "Predicted mean"), pch=c(1, NA), lty=c(NA, 1), lwd=c(NA, 2), col=
```



As seen in the plot above the posterior mean follows the data quite well. It does not however capture the outlier points at for example around time point 400.

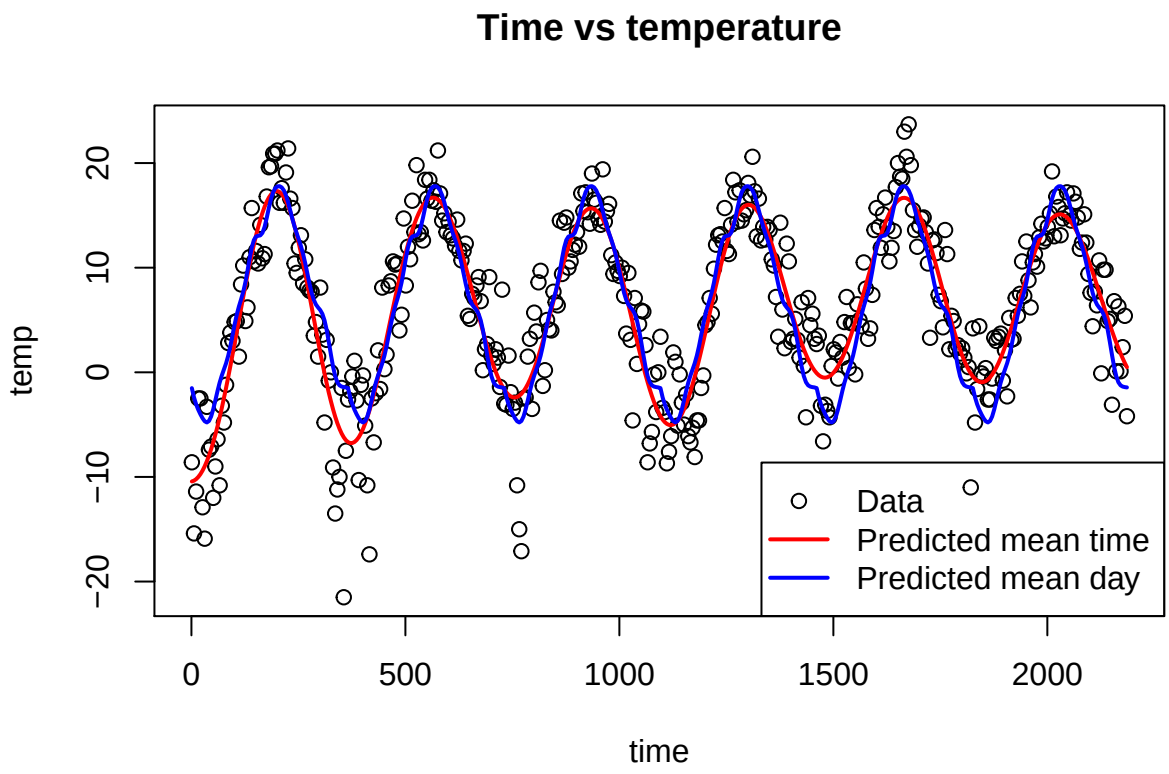
```
posteriorGP = function(X, y, XStar, sigmaNoise, k, ...) {
  n = length(X)
  K=k(X, X, ...)
  kStar=k(X,XStar, ...)
  L = t(chol(K + sigmaNoise^2*diag(n)))
  alpha=solve(t(L),solve(L, y))
  predMean=t(kStar)%*%alpha
  v=solve(L, kStar)
  predVar=k(XStar, XStar, ...)-t(v)%*%v
  return(list(mean=predMean, var=predVar))
}

scale_mean=mean(temp)
scale_var=sqrt(var(temp))
posterior = posteriorGP(scale(time), scale(temp), scale(time), sigmaN, SquaredExpKernel, sigmaF, l)
```

```
postVar = posterior$var
postMean = posterior$mean
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
lines(time, postMean*scale_var+scale_mean+1.96*sqrt(diag(postVar)), lwd=2, lty=2, col="blue")
lines(time, postMean*scale_var+scale_mean-1.96*sqrt(diag(postVar)), lwd=2, lty=2, col="blue")
legend("bottomright", legend=c("Data", "Predicted mean", "95% probability bands"), pch=c(1, NA, NA), lty
       lwd=c(NA, 2, 2), col=c("black", "red", "blue"))
```



The probable bands capture some more of the data but still do not handle the extreme outlier points.

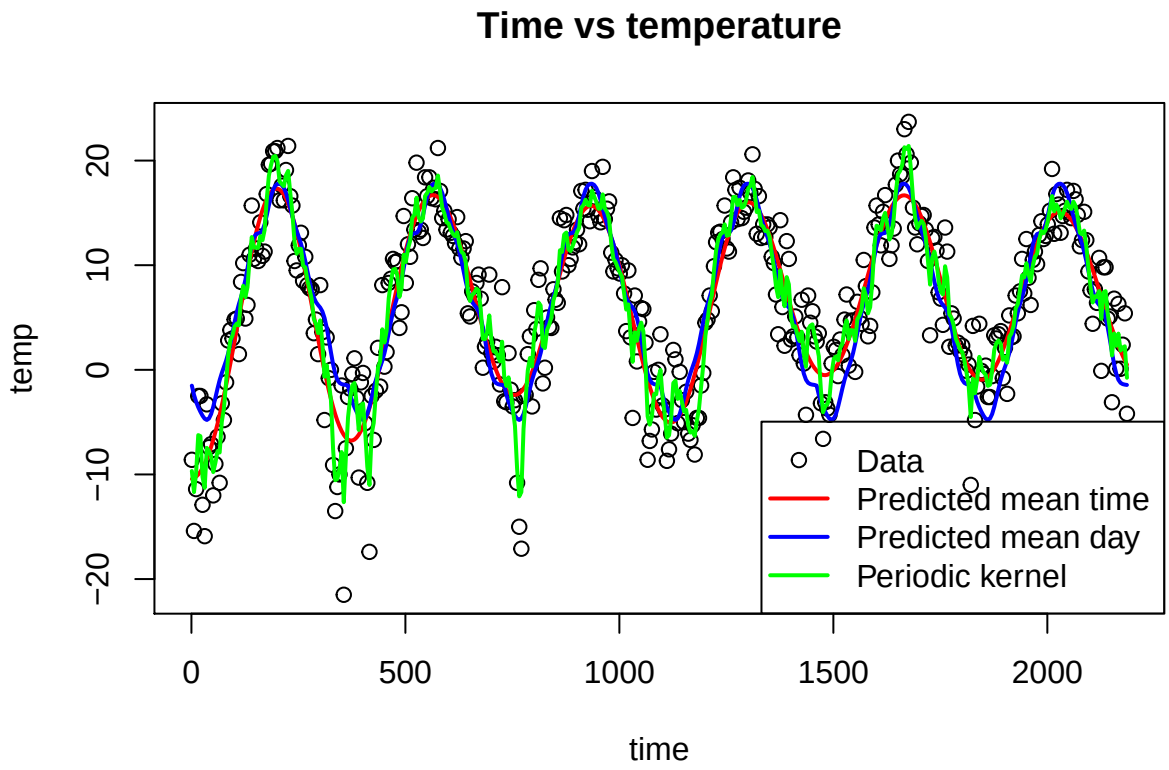
```
sigmaF=20
l=0.2
SEkernel = NestedSquaredExpKernel(sigmaF, l)
model2 = gausspr(day, temp, kernel=SEkernel, var=sigmaN^2)
predictedMean2=predict(model2, newdata=day)
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
lines(time, predictedMean2, type="l", lwd=2, col="blue")
legend("bottomright", legend=c("Data", "Predicted mean time", "Predicted mean day"), pch=c(1, NA, NA), l
       lwd=c(NA, 2, 2), col=c("black", "red", "blue"))
```



As evident from the plot, when using the day set no difference is being made from year to year. This seems to have worsen the posterior in some areas (for example during the early time points) but seems to have improved the posterior in other areas (for example around the time point of 1500).

```
PeriodicKernel <- function(sigmaF=1,l1=1, l2=10, d){
  val = function(x, xStar) {
    diff = abs(x-xStar)
    return(sigmaF^2*exp(-((2*sin(pi*diff))/d)/l1^2)*exp(-0.5*diff^2/l2^2))
  }
  class(val)='kernel'
  return(val)
}

sigmaF=20
l1=1
l2=10
d=365/sd(time)

PerKernel = PeriodicKernel(sigmaF, l1, l2, d)
model3 = gausspr(time, temp, kernel=PerKernel, var=sigmaN^2)
predictedMean3=predict(model3, newdata=time)
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
lines(time, predictedMean2, type="l", lwd=2, col="blue")
lines(time, predictedMean3, type="l", lwd=2, col="green")
legend("bottomright", legend=c("Data", "Predicted mean time", "Predicted mean day", "Periodic kernel"),
       pch=c(1, NA, NA, NA), lty=c(NA, 1, 1, 1),
```

```
       lwd=c(NA, 2, 2, 2), col=c("black", "red", "blue", "green"))
```



The plot above shows the periodic kernel's ability to fit the data more precisely than the other kernels. It is less smooth which makes it capture more data points which are located further away from the posterior mean.

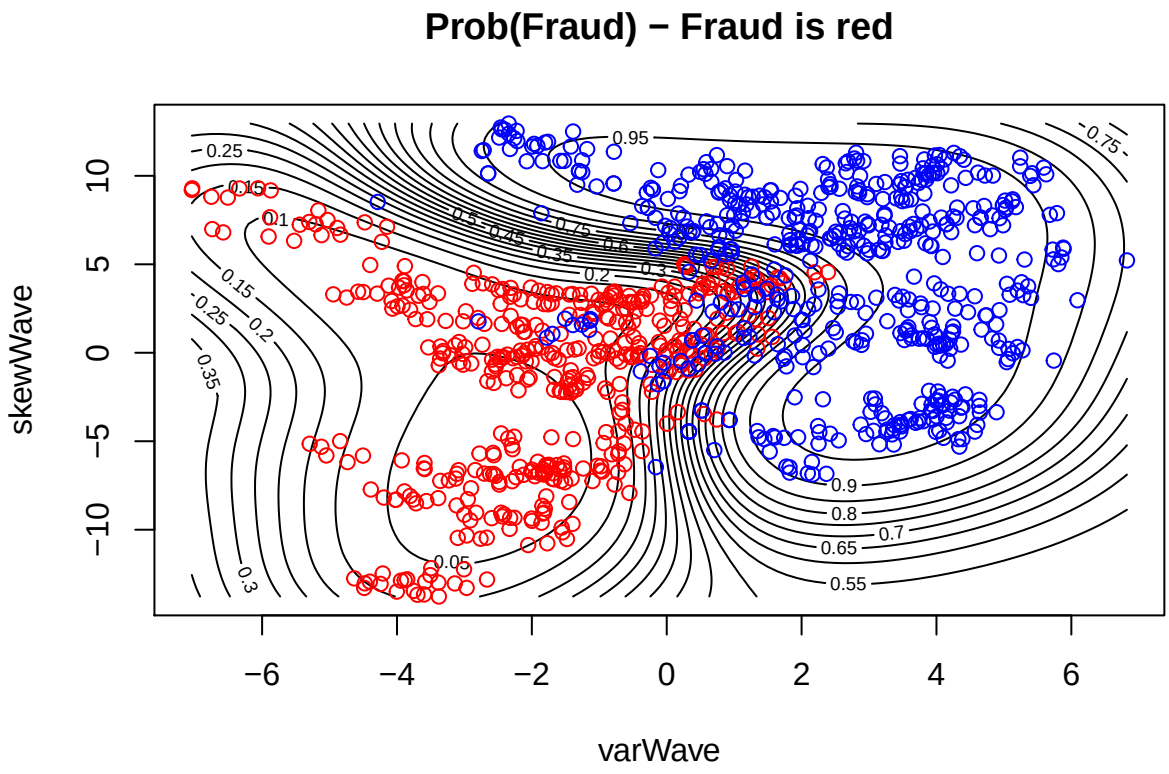
Assignment 3

```
# Fetching data
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])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train=data[SelectTraining,]
test=data[-SelectTraining,]

## 1) Use the R package kernlab to fit a Gaussian process classification model for fraud on the trainin
## and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contour
## over a suitable grid of values of varWave and skewWave. Overlay the training data for fraud=1 (as bl
## (as red points). You can reuse code from the file KernLabDemo.R available on course website. Compute
## for the classifier and its accuracy.
```

```
model = gausspr(fraud ~ varWave+skewWave, data=train)
```

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

```
predictedTrain = predict(model, newdata=train)
confusionMatrix = table(predictedTrain, train[,5]) # confusion matrix

# class probabilities
probPreds <- predict(model, train, type="probabilities")
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(subset(train, select=c("varWave", "skewWave")))
probPreds <- predict(model, gridPoints, type="probabilities")

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




```
# Confusion matrix and accuracy
print("The confusion matrix is as follow:")
```

```
## [1] "The confusion matrix is as follow:"
```

```
confusionMatrix
```

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

```
print("... and the accuracy is the following:")
```

```
## [1] "... and the accuracy is the following:"
```

```
sum(diag(confusionMatrix))/sum(confusionMatrix)
```

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

As seen from the outputs above the confusion matrix, accuracy as well as a plot over the contours of the prediction probabilities for fraud where $fraud = 1$ corresponds to the *red* points.

```
## 2) Using the estimated model from 1), make predictions on the test set.

predictedTest = predict(model, newdata=test)
confusionMatrix_test = table(predictedTest, test[,5])
print("The accuracy for the test set is as follows:")
```

```
## [1] "The accuracy for the test set is as follows:"
```

```
sum(diag(confusionMatrix_test))/sum(confusionMatrix_test)
```

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

The output above shows the calculated accuracy for the test set. It is a bit lower than for the train set which is natural but is still considered a high accuracy.

```
## 3) Train a model using all four covariates. Make predictions on the test set and compare the accurac
## covariates

model2 = gausspr(fraud ~., data=train)
```

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

```
predictedTest2 = predict(model2, newdata=test)
confusionMatrix_test2 = table(predictedTest2, test[,5])
print("The confusion matrix is as follow:")
```

```
## [1] "The confusion matrix is as follow:"
```

```
confusionMatrix_test2
```

```
##
## predictedTest2   0   1
##              0 216   0
##              1   2 154
```

```
print("... and the accuracy is the following:")
```

```
## [1] "... and the accuracy is the following:"
```

```
sum(diag(confusionMatrix_test2))/sum(confusionMatrix_test2)
```

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

The confusion matrix as well as the accuracy for the model using all covariates in training is shown above. As seen above the accuracy is even higher in comparison to the model which only used two covariates. This is normal behaviour in linear regression models and models with more explanatory variables are always more accurate than models with less covariates since the new covariates can only add new information and never remove vital information.

Appendix

```r
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)
}

posteriorGP = function(X, y, XStar, sigmaNoise, k, ...) {
  n=length(X)
  K=k(X, X, ...)
  kStar=k(X,XStar, ...)
  L = t(chol(K + sigmaNoise^2*diag(n)))
  alpha=solve(t(L),solve(L, y))
  predMean=t(kStar)%*%alpha
  v=solve(L, kStar)
  predVar=k(XStar, XStar, ...)-t(v)%*%v
  return(list(mean=predMean, var=predVar))
}

# 2. Plot one draw from posterior

xTest = seq(-1, 1, length=100)

# Plotting one draw
obs=data.frame(x=0.4, y=0.719)
sigmaF <- 1
sigmaN=0.1
l <- 0.3
posteriorSim=posteriorGP(obs$x, obs$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim$mean, type="l",
     ylim=c(min(posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var))), max(posteriorSim$mean + 1.96*sq
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub="One observation as prior")
lines(xTest, posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim$mean + 1.96*sqrt(diag(posteriorSim$var)), col = "gray", lwd = 2, lty=21)

## 3. Now update posterior with two observations

x=c(0.4, -0.6)
y=c(0.719, -0.044)
obs2=data.frame(x=x, y=y)
posteriorSim2=posteriorGP(obs2$x, obs2$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim2$mean, type="l",
     ylim=c(min(posteriorSim2$mean-1.96*sqrt(diag(posteriorSim2$var))), max(posteriorSim2$mean+1.96*sqr
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
```

```
     sub="Two observations as prior")
lines(xTest, posteriorSim2$mean - 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim2$mean + 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)

## 4. Now use 5 observations and plot the posterior

x=c(-1, -0.6, -0.2, 0.4, 0.8)
y=c(0.768, -0.044, -0.940, 0.719, -0.664)
obs2=data.frame(x=x, y=y)
posteriorSim2=posteriorGP(obs2$x, obs2$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim2$mean, type="l",
     ylim=c(min(posteriorSim2$mean-1.96*sqrt(diag(posteriorSim2$var))), max(posteriorSim2$mean+1.96*sqr
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub="Five observations as prior")
lines(xTest, posteriorSim2$mean - 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim2$mean + 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)

## 5. Repeat 4 with hyperparam sigmaF=1 and l=1

sigmaF=1
l=1
posteriorSim2=posteriorGP(obs2$x, obs2$y, xTest, sigmaN, SquaredExpKernel, sigmaF, l)
plot(xTest, posteriorSim2$mean, type="l",
     ylim=c(min(posteriorSim2$mean-1.96*sqrt(diag(posteriorSim2$var))), max(posteriorSim2$mean+1.96*sqr
     col="red", main="Plot of posterior mean (red) with 95 % probability bands", xlab="x", ylab="Poster
     sub=expression(paste("Five observations as prior and ", sigma[F], "=1 and l=1")))
lines(xTest, posteriorSim2$mean - 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)
lines(xTest, posteriorSim2$mean + 1.96*sqrt(diag(posteriorSim2$var)), col = "gray", lwd = 2, lty=21)

## GP regression with kernlab

# Fetch data from source

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")

# Restructuring data

data$date = as.Date(data$date, "%d/%m/%y")
time=seq(1:2190)
day=time %% 365
day[which(day == 0)] = 365
id = seq(from=1, to=2186, 5)
time=time[id]
day=day[id]

## 1) Familirarization of the kernlab library. Calculating covariance matrix from two input vectors

# This is just to test how one evaluates a kernel function
# and how one computes the covariance matrix from a kernel function.
library(kernlab)
X <- as.vector(c(1,3,4)) # Simulating some data.
Xstar <- as.vector(c(2,3,4))
```

```r
ell <- 1

# 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)
}

NestedSquaredExpKernel <- function(sigmaF=1,l=3){
  EvaluExpKernel = function(x, xStar) {
    n1 <- length(x)
    n2 <- length(xStar)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x-xStar[i])/l)^2 )
    }
    return(K)
  }
  class(EvaluExpKernel)='kernel'
  return(EvaluExpKernel)
}

SEkernel <- NestedSquaredExpKernel() # Note how I reparametrize the rbfdot (which is the SE kernel) in
SEkernel(1,2) # Just a test - evaluating the kernel in the points x=1 and x'=2.
# Computing the whole covariance matrix K from the kernel. Just a test.
kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(X,Xstar).

## 2) Consider model: temp=f(time)+epsilon with epsilon~N(0, sigmaN^2). f~GP(0, k(time, time'))
## Let sigmaN be the residual variance from a simple quadratic regression fit. Estimate the above gauss
## model using the squared exponentional function from 1) with sigmaF=20 and l=0.2. Use the predict fun
## posterior mean at every data point in the training dataset. Make a scatterplot of the data and super
## of f as a curve (use type="l" in the plot function). Play around with different values on sigmaF and

temp=data$temp[id]
fit = lm(temp ~ time+time^2)
sigmaN=sd(fit$residuals)

sigmaF=20
l=0.2
SEkernel = NestedSquaredExpKernel(sigmaF, l)
model = gausspr(time, temp, kernel=SEkernel, var=sigmaN^2)
predictedMean=predict(model, newdata=time)
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
legend("bottomright", legend=c("Data", "Predicted mean"), pch=c(1, NA), lty=c(NA, 1), lwd=c(NA, 2), col=

## 3) Kernlab can compute posterior variance of f, but it seems to be a bug in the code. So, do your ow
## posterior variance of f and plot the 95 % probability (pointwise) bands for f. Superimpose these ban
```

```r
## posterior mean that you obtained in (2)

posteriorGP = function(X, y, XStar, sigmaNoise, k, ...) {
  n = length(X)
  K=k(X, X, ...)
  kStar=k(X,XStar, ...)
  L = t(chol(K + sigmaNoise^2*diag(n)))
  alpha=solve(t(L),solve(L, y))
  predMean=t(kStar)%*%alpha
  v=solve(L, kStar)
  predVar=k(XStar, XStar, ...)-t(v)%*%v
  return(list(mean=predMean, var=predVar))
}

posterior = posteriorGP(scale(time), scale(temp), scale(time), sigmaN, SquaredExpKernel, sigmaF, l)
postVar = posterior$var
postMean = posterior$mean
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
lines(time, postMean*sqrt(diag(postVar))+predictedMean + 1.96*sqrt(diag(postVar)), lwd=2, lty=21, col="
lines(time, postMean*sqrt(diag(postVar))+predictedMean -1.96*sqrt(diag(postVar)), lwd=2, lty=21, col="gr
legend("bottomright", legend=c("Data", "Predicted mean", "95% probability bands"), pch=c(1, NA, NA), lty
       lwd=c(NA, 2, 2), col=c("black", "red", "grey"))

## 4) Consider now the following model: temp = f(day) + epsilon with epsilon ~ N(0, sigmaN^2) and f~GP(
## Estimate the model using the squared exponential function with sigmaF=20 and l=0.2. Superimpose the
## from this model on the posterior mean from the model in (2). Note that this plot should also have ti
## horizontal axis. Compare the results of both models. What are the pros and cons of each model?

sigmaF=20
l=0.2
SEkernel = NestedSquaredExpKernel(sigmaF, l)
model2 = gausspr(day, temp, kernel=SEkernel, var=sigmaN^2)
predictedMean2=predict(model2, newdata=day)
plot(time, temp, type="p", main="Time vs temperature")
lines(time, predictedMean, type="l", lwd=2, xlab="Time", ylab="Temp", col="red")
lines(time, predictedMean2, type="l", lwd=2, col="blue")
legend("bottomright", legend=c("Data", "Predicted mean time", "Predicted mean day"), pch=c(1, NA, NA),
       lwd=c(NA, 2, 2), col=c("black", "red", "blue"))

## Finally, implement a generalization of the periodic kernel given in the lectures. Note that Note tha
## length scales here, and `2 controls the correlation between the same day in different years. Estimat
## 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 inputs to have standa
## Compare the fit to the previous two models (with sigmaF = 20 and l = 0.2). Discuss the results.

PeriodicKernel <- function(sigmaF=1,l1=1, l2=10, d){
  val = function(x, xStar) {
    diff = abs(x-xStar)
    return(sigmaF^2*exp(-((2*sin(pi*diff))/d)/l1^2)*exp(-0.5*diff^2/l2^2))
  }
  class(val)='kernel'
  return(val)
```

```
}

sigmaF=20
l1=1
l2=10
d=365/sd(time)

PerKernel = PeriodicKernel(sigmaF, l1, l2, d)
model3 = gausspr(time, temp, kernel=PerKernel, var=sigmaN^2)
predictedMean3=predict(model3, newdata=time)
lines(time, predictedMean3, type="l", lwd=2, col="green")
legend("bottomright", legend=c("Data", "Predicted mean time", "Predicted mean day", "Periodic kernel"),
       pch=c(1, NA, NA, NA), lty=c(NA, 1, 1, 1),
       lwd=c(NA, 2, 2, 2), col=c("black", "red", "blue", "green"))

# Fetching data
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])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train=data[SelectTraining,]
test=data[-SelectTraining,]

## 1) Use the R package kernlab to fit a Gaussian process classification model for fraud on the trainin
## and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contour
## over a suitable grid of values of varWave and skewWave. Overlay the training data for fraud=1 (as bl
## (as red points). You can reuse code from the file KernLabDemo.R available on course website. Compute
## for the classifier and its accuracy.

model = gausspr(fraud ~ varWave+skewWave, data=train)
predictedTrain = predict(model, newdata=train)
predictedTrain
confusionMatrix = table(predictedTrain, train[,5]) # confusion matrix

# class probabilities
probPreds <- predict(model, train, type="probabilities")
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(subset(train, select=c("varWave", "skewWave")))
probPreds <- predict(model, gridPoints, type="probabilities")

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

```r
# Confusion matrix and accuracy
confusionMatrix
sum(diag(confusionMatrix))/sum(confusionMatrix)

## 2) Using the estimated model from 1), make predictions on the test set.

predictedTest = predict(model, newdata=test)
confusionMatrix_test = table(predictedTest, test[,5])
sum(diag(confusionMatrix_test))/sum(confusionMatrix_test)

## 3) Train a model using all four covariates. Make predictions on the test set and compare the accuracy
## covariates

model2 = gausspr(fraud ~., data=train)
predictedTest2 = predict(model2, newdata=test)
confusionMatrix_test2 = table(predictedTest2, test[,5])
sum(diag(confusionMatrix_test2))/sum(confusionMatrix_test2)
```