Advanced Machine Learning
LAB 4: GAUSSIAN PROCESSES

Mohammed Bakheet (mohba508)

15/10/2020

Contents

Question 1: GP Regression

Implementing GP Regression. This first exercise will have you writing your own code for the Gaussian process regression model:

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

You must implement Algorithm 2.1 on page 19 of Rasmussen and Willams' book. The algorithm uses the Cholesky decomposition (chol in R) to attain numerical stability. Note that L in the algorithm is a lower triangular matrix, whereas the R function returns an upper triangular matrix. So, you need to transpose the output of the R function. In the algorithm, the notation A /bb means the vector x that solves the equation A x = b (see p. xvii in the book). This is implemented in R with the help of the function solve.

Question1-1:

Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it posteriorGP) should return a vector with the posterior mean and variance of f , both evaluated at a set of x-values X_*. You can assume that the prior mean of f is zero for all x. The function should have the following inputs:

- X: Vector of training inputs.
- y: Vector of training targets/outputs.
- *XStar*: Vector of inputs where the posterior distribution is evaluated, i.e. X_* .
- *sigmaNoise*: Noise standard deviation σ_n .
- k: Covariance function or kernel. That is, the kernel should be a separate function

The exponential kernel method:

```
#The squared exponential function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=0.3){
  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)
}
```

Now we can set the parameters:

```
sigmaF = 1
l = 0.3
sigmaNoise = 0.1
xGrid <- seq(-1,1,length=20)
nSim = 1
```

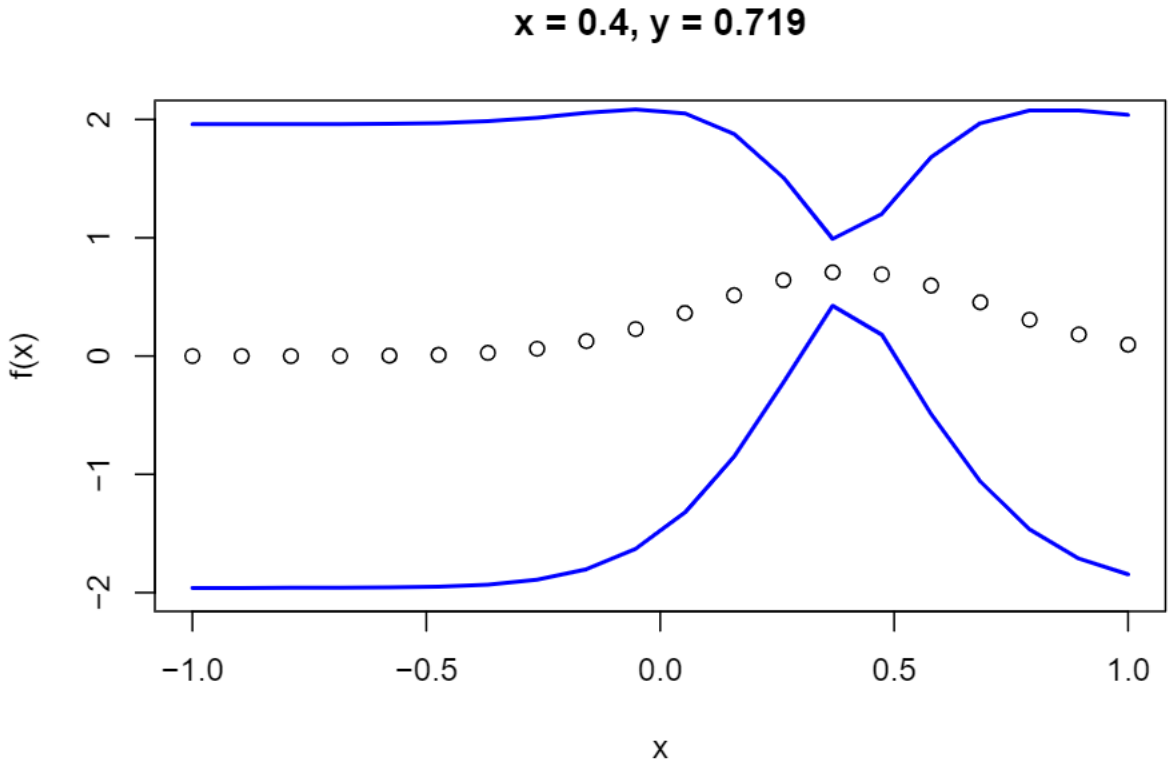
Posterior Gaussian Process Function

```
#Question1:  The posterior function
posteriorGP = function(X, y, XStar, sigmaNoise, k){
  #The algorithm
  K = k(X, X, sigmaF = sigmaF, l = l)
  n = length(X)

  L = t(chol(K + sigmaNoise^2*diag(n)))
  alpha = solve(t(L),solve(L,y))
  KStar = k(X,XStar, sigmaF = sigmaF, l = l)
  FStar = t(KStar)%*%alpha
  v = solve(L,KStar)
  VFStar = k(XStar, XStar, sigmaF = sigmaF, l = l)-t(v)%*%v

  logP = -0.5*t(y)%*%alpha - sum(log(diag(L))) - (n/2)*log(2*pi)

  return(list('mean'=FStar, 'variance'= VFStar, 'logMargLik' = logP))
}
```

Question1-2:

Evaluating the function on $(x,y) = (0.4,0.719)$

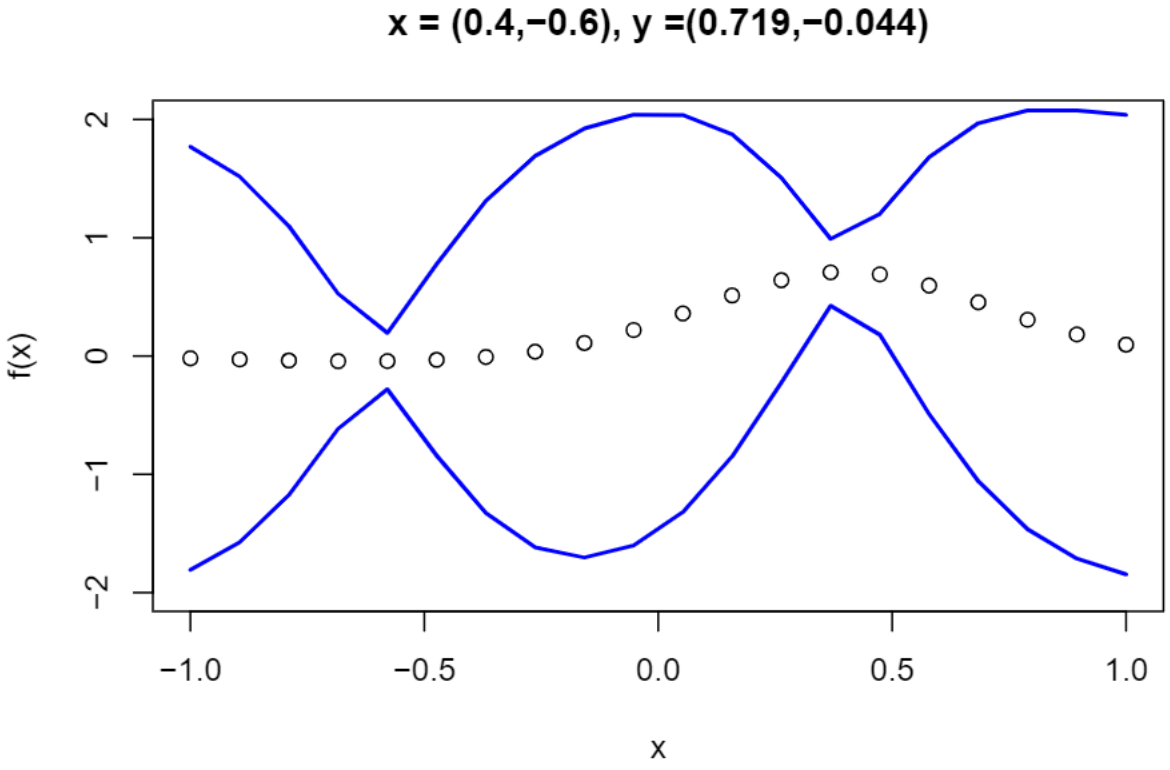
```
#Question2: evaluating on (0.4,0.719)
x = 0.4
y = 0.719
posteriorValue = posteriorGP(X = x,
                             y = y,
                             XStar = xGrid,
                             sigmaNoise = sigmaNoise,
                             k = SquaredExpKernel)
postMean = posteriorValue$mean
postVar = posteriorValue$variance

#Function for plotting the posterior mean
plotMean = function(mean,var, title){
  plot(xGrid, mean, type="p", ylab="f(x)", xlab="x", ylim = c(-2,2), main = title)

  #Plotting confidence band
  lines(xGrid, mean - 1.96*sqrt(diag(var)), col = "blue", lwd = 2)
  lines(xGrid, mean + 1.96*sqrt(diag(var)), col = "blue", lwd = 2)
}
plotMean(postMean,postVar, title = 'x = 0.4, y = 0.719')
```



Question1-3:

Updating the posterior with observation (-0.6, -0.044)

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

posteriorValue2 = posteriorGP(X = x,
                              y = y,
                              XStar = xGrid,
                              sigmaNoise = sigmaNoise,
                              k = SquaredExpKernel)
postMean2 = posteriorValue2$mean
postVar2 = posteriorValue2$variance

plotMean(postMean2, postVar2, title = 'x = (0.4,-0.6), y =(0.719,-0.044)')
```

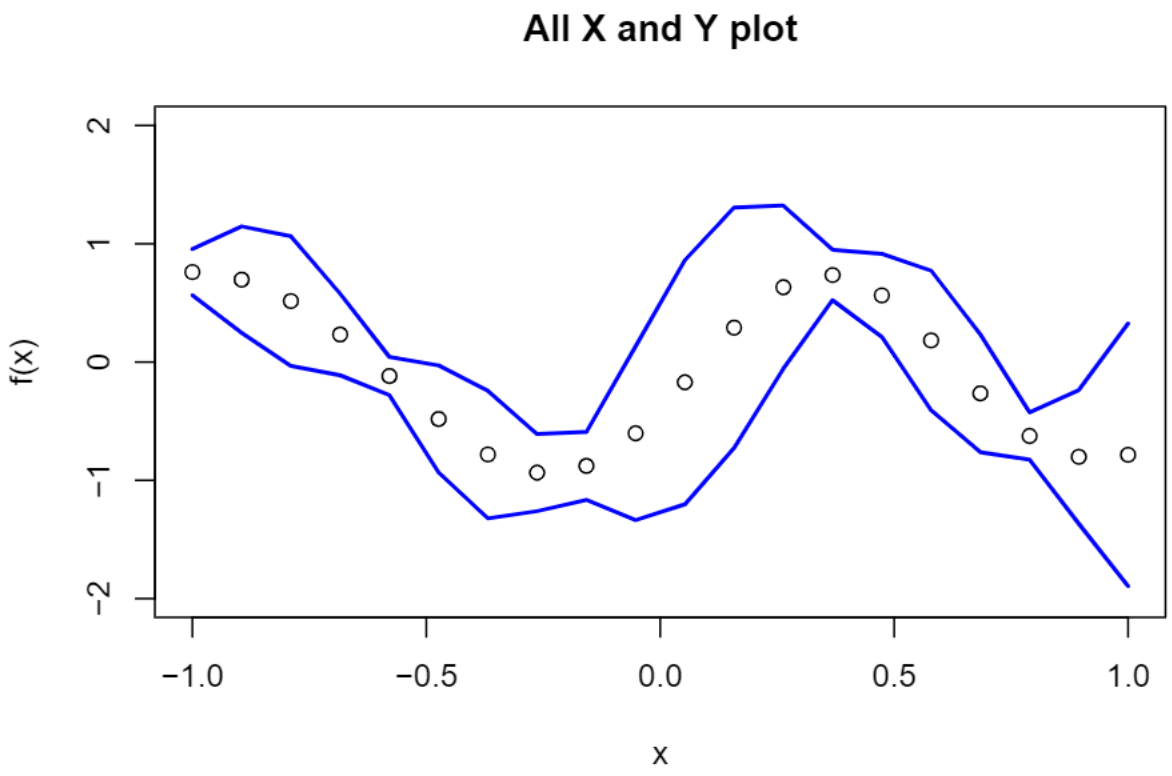


Question1-4:

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*.

The posterior with all five observations plus the previous two observations:

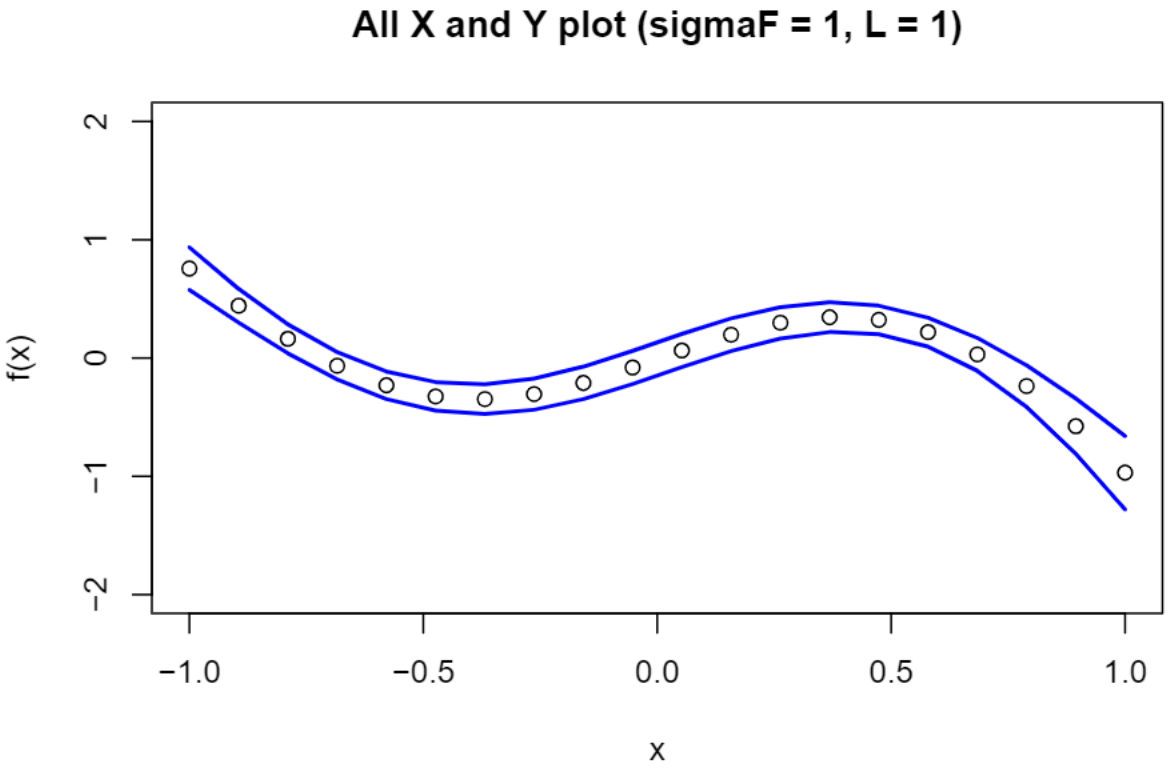
```
#Question4: The posterior with five observations
x = c(0.4, -0.6, -1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.719, -0.044, 0.768, -0.044, -0.940, 0.719, -0.664)

posteriorValue3 = posteriorGP(X = x,
                              y = y,
                              XStar = xGrid,
                              sigmaNoise = sigmaNoise,
                              k = SquaredExpKernel)
postMean3 = posteriorValue3$mean
postVar3 = posteriorValue3$variance


plotMean(postMean3,postVar3, title = "All X and Y plot")
```



Question1-5:

Repeating with different hyperparameters:

```
#Question5: Repeating with different hyperparameters.

sigmaF = 1
l = 1
posteriorValue4 = posteriorGP(X = x,
                              y = y,
                              XStar = xGrid,
                              sigmaNoise = sigmaNoise,
                              k = SquaredExpKernel)
postMean4 = posteriorValue4$mean
postVar4 = posteriorValue4$variance

plotMean(postMean4,postVar4, title = "All X and Y plot (sigmaF = 1, L = 1)")
```



When updating the hyperparameters with $\sigma_f = 1$ and $l = 1$ and calculating the posterior. It can be seen from the plot above that the posterior function is smoother, so is the confidence interval. When we increase the value of l, this will decrease the value of the exponential due to its division by l and taking the the negative value of that. Hence, the value of our function will increase, and this will give us a smoother function. On the contrary, when we use a smaller value of l, this will result in a noisy function, as we can see in the plot when we used a value of 0.3 for l. A higher value of l also indicates a more accurate confidence band at the evaluated points, whereas when using a smaller value of l the confidence band is higher around all values other than the ones our function was evaluated on.

When we used a value of 0.3 for l and the evaluation was done on one and two points respectively, the confidence band was narrower around the point/s of evaluation, and it was high for the rest of our function values.

Question 2:

GP Regression with kernlab:

In this exercise, you will work with the daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. We have removed the leap year day February 29, 2012 to make things simpler.
Create the variable time which records the day number since the start of the dataset (i.e., time= 1, 2, . . ., $365 \times 6 = 2190$). Also, create the variable day that records the day number since the start of each year (i.e., day= 1, 2, . . ., 365, 1, 2, . . ., 365). Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the data and use only every fifth observation. This means that your time and day variables are now time= 1, 6, 11, . . ., 2186 and day= 1, 6, 11, . . ., 361, 1, 6, 11, . . ., 361.

```
#Takinng every fifth observation and scaling time

time = seq(from = 1, to = length(time), by = 5)
day = day[time]
```

```
scaledTime = (time-mean(time))/sd(time)
```

Question2-1:

Defining and evaluating square exponential kernel function

```
Matern32 <- function(sigmaf = 1, ell = 1)
{
  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)
}

#Evaluating the function
MaternFunc = Matern32(sigmaf = 20, ell = 0.2) # MaternFunc is a kernel FUNCTION
cat('The function evaluation at the point (1,2) is: ',MaternFunc(1,2))
```

```
## The function evaluation at the point (1,2) is:  0.6698044
```

```
#Computing the covarance matrix K
X = c(1,3,4)
Xstar = c(2,3,4)

K <- kernelMatrix(kernel = MaternFunc, x = X, y = Xstar)
```

The covariance matrix for 1, 3, 4 and 2, 3, 4 is:

```
## An object of class "kernelMatrix"
##              [,1]         [,2]         [,3]
## [1,] 0.6698044031 2.201894e-04 5.620987e-08
## [2,] 0.6698044031 4.000000e+02 6.698044e-01
## [3,] 0.0002201894 6.698044e-01 4.000000e+02
```

Question2-2:

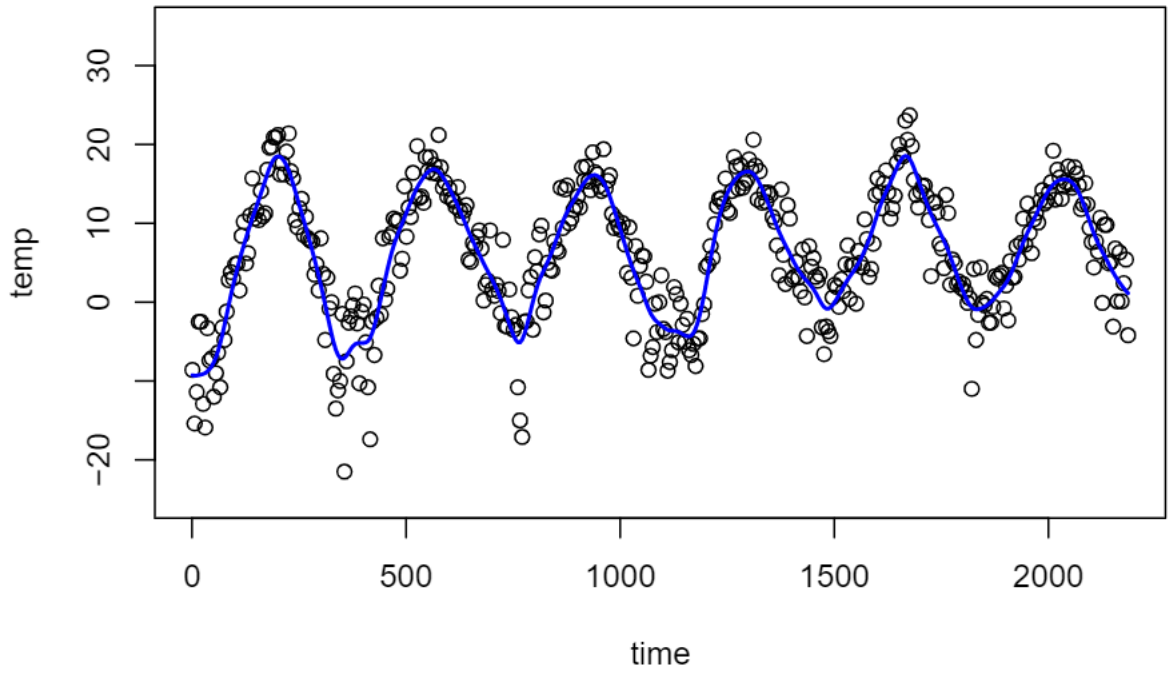
The model is:
$\sqcup emp = f(time) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ and $f \sim \mathcal{GP}(0, k(time, time'))$

```
#Letting sigma noise equal to residual variance of a quadratic regression fit using lm
modelData = data.frame(data$temp[time], scaledTime)
colnames(modelData) = c('temp', 'time')
polyFit <- lm(temp ~ time + I(time^2) + I(time^3), data = modelData)
sigmaNoise = sd(polyFit$residuals)
cat('sigmaNoise is: ',sigmaNoise)
```

```
## sigmaNoise is:  8.142339
```

```
plot(time,data$temp[time], ylim = c(-25,35), xlab = 'time', ylab = 'temp')

# Fit the GP with Square expontial kernel
MaternFunc = Matern32(sigmaf = 20, ell = 0.2)
GPfit <- gausspr(scaledTime, data$temp[time],
```

```
                kernel = MaternFunc,
                kpar = list(sigmaf = 20, ell = 0.2),
                var = sigmaNoise^2)
meanPred <- predict(GPfit, scaledTime)
lines(time, meanPred, col="blue", lwd = 2)
```

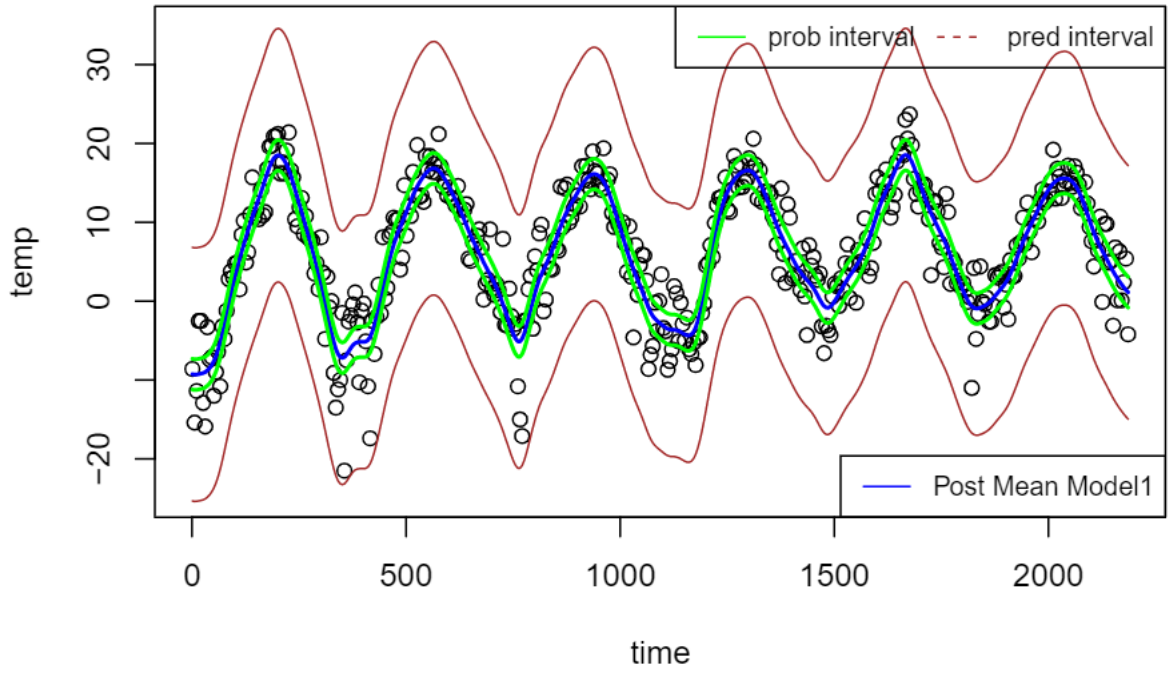


Question2-3:

kernlab can compute the posterior variance of f , but it seems to be a bug in the code. So, do your own computations for the posterior variance of f and plot the 95 % probability (pointwise) bands for f . Superimpose these bands on the figure with the posterior mean that you obtained in (2)

```
ell <- 0.3
SEkernel <- rbfdot(sigma = 1/(2*ell^2))

x<-time
xs<-time
n <- length(x)
Kss <- kernelMatrix(kernel = SEkernel, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = SEkernel, x = x, y = x)
Kxs <- kernelMatrix(kernel = SEkernel, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for fStar
plot(time,data$temp[time], ylim = c(-25,35), xlab = 'time', ylab = 'temp')
lines(time, meanPred, col="blue", lwd = 2)


lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)

# Prediction intervals for yStar
```

```
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "brown")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "brown")

legend("bottomright",
       legend = c('Post Mean Model1'),
       col = c( 'blue'),
       lty=1:2,
       cex=0.8)
legend("topright",
       legend = c('prob interval',
                  'pred interval'),
       col = c('green', 'brown'),
       lty=1:2,
       cex=0.8,
       horiz = TRUE)
```



Question2-4:

The model is:

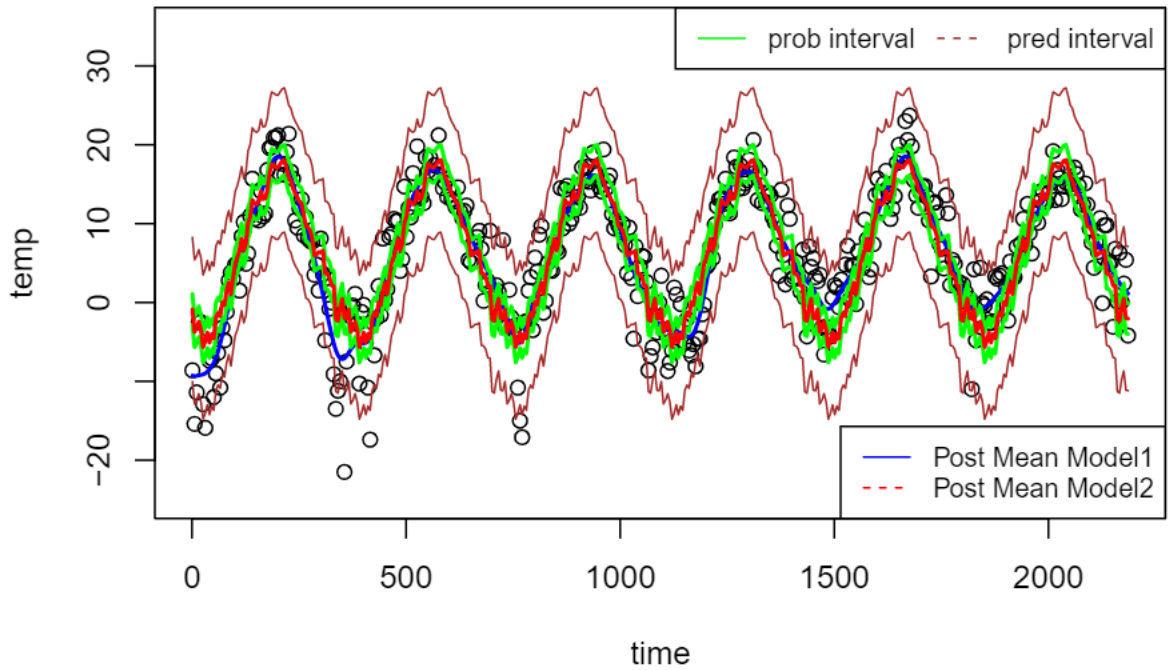
$temp = f(day) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ and $f \sim \mathcal{G}P(0, k(day, day'))$

```
#Letting sigma noise equal to residual variance of a quadratic regression fit using lm
modelData = data.frame(data$temp[time], day)
colnames(modelData) = c('temp', 'day')
polyFit <- lm(temp ~ day + I(day^2) + I(day^3), data = modelData)
sigmaNoise = sd(polyFit$residuals)
cat('sigmaNoise is: ',sigmaNoise)
```

```
## sigmaNoise is:  4.558032
```

```
# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)
```

```
MaternFunc = Matern32(sigmaf = 20, ell = 0.2)
GPfit <- gausspr(day,
                 data$temp[time],
                 kernel = MaternFunc,
                 kpar = list(sigmaf = 20, ell = 0.2),
                 var = sigmaNoise^2)
meanPred2 <- predict(GPfit, day)

plot(time,data$temp[time], ylim = c(-25,35), xlab = 'time', ylab = 'temp')
lines(time, meanPred, col="blue", lwd = 2)
# Probability intervals for fStar
lines(xs, meanPred2 - 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)
lines(xs, meanPred2 + 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)

# Prediction intervals for yStar
lines(xs, meanPred2 - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "brown")
lines(xs, meanPred2 + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "brown")
lines(time, meanPred2, col="red", lwd = 2)
legend("bottomright",
       legend = c('Post Mean Model1',
                  'Post Mean Model2'),
       col = c( 'blue', 'red'),
       lty=1:2,
       cex=0.8)
legend("topright",
       legend = c('prob interval',
                  'pred interval'),
       col = c('green', 'brown'),
       lty=1:2,
       cex=0.8,
       horiz = TRUE)
```



The prediction band for this model (using the day variable) is narrower than when using the time variable. And

that's because when the time variable is used the model is using an increasing value of time with an interval of five. Whereas, when the day is used, the model here has an idea of which day it should predict, due to the repetition of one day in many years, so, the day variable gives the model a better idea of what to predict given a specific day. And that's why the prediction band of the second model is more certain than the first model.

Nonetheless, using the day variables results in a noisy fit to the data with the same noise frequency in all years, and that's because the day variable considers the seasonality of temperature ruing the entire period. Even though, our l value is small, which should assure a smoother fit, but since the day variable doesn't consider the trend and only seasonality (maybe the trend during a single year to be more accurate). The day model is smooth between all years, but noiscy within each year.

Question2-5:

Implement a generalization of the periodic kernel

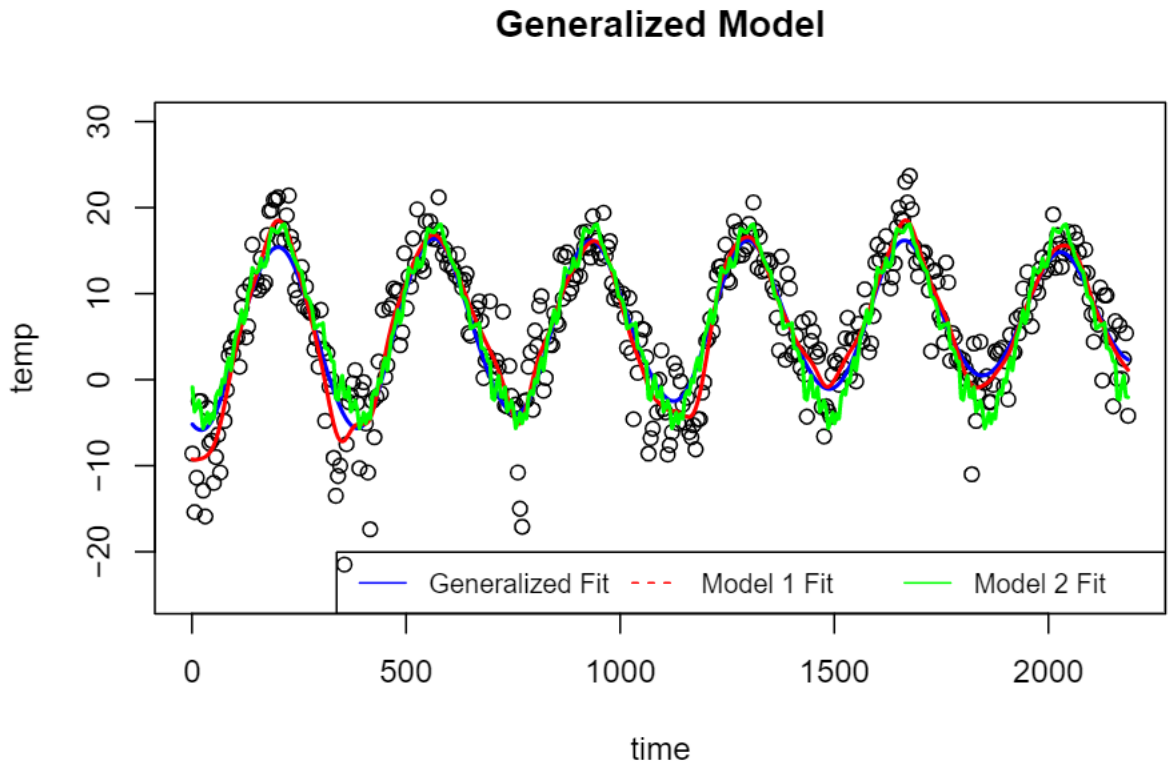
$$k(x,x') = \sigma_f^2]xp\frac{-2\int in^2(\pi|x-x'|/d)}{l_1^2} * exp-\frac{1}{2}\frac{|x-x'|^2}{l_2^2}$$

```
GPKernel = function(l1,l2, sigmaF, d){

  rval <- function(x1, x2 = NULL) {
    return(sigmaF^2 * exp(-(2*(sin(pi*abs(x1-x2)/d))^2)/l1^2)*(exp(-0.5*abs(x1-x2)^2)/l2^2))
  }
  class(rval) <- "kernel"
  return(rval)

}

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

GPKernelFun = GPKernel(l1,l2,sigmaF, d)
cat('By evaluating the kernel method on the point (1,2) we get',GPKernelFun(1,2))
```

```
## By evaluating the kernel method on the point (1,2) we get 0.8074298
```

```
#Fitting the generlaized model
GPfit <- gausspr(scaledTime, data$temp[time],
                 kernel = GPKernelFun,
                 kpar = list(sigmaF = sigmaF, l1 = l1, l2 = l2, d=d),
                 var = sigmaNoise^2)
meanPred3 <- predict(GPfit, scaledTime)
plot(time,data$temp[time], ylim = c(-25,30), main = 'Generalized Model', xlab = 'time', ylab = 'temp')
lines(time, meanPred3, col="blue", lwd = 2)
lines(time, meanPred, col="red", lwd = 2)
lines(time, meanPred2, col="green", lwd = 2)
legend("bottomright",
       legend = c('Generalized Fit',
                  'Model 1 Fit',
                  'Model 2 Fit'),
       col = c('blue', 'red', 'green'),
```

```
       lty=1:2,
       cex=0.8,
       horiz = TRUE)
```



From the plot, it can be seen that model 1 (time variable) fits the data more accurately than model 2 (day variable).The generalized model gives a smoother fit to our model than that of both first and second models, and that's because we added a second length scale *l2* that controls the correlation between the same day in different years. As it can be seen from the graph, the day model (green line) fits the data quite well, but it's not smooth, and the time model (red line) is smoother than the day model (green line) and less smoother than the generalized model (blue line).

Question 3:

GP Classification with kernlab

Question3-1:

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.

```
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])
```

```r
#Dividing the data into training and testing
set.seed(111)
SelectTraining = sample(1:dim(data)[1],
                        size = 1000,
                        replace = FALSE)
trainingData = data[SelectTraining,]
testingData = data[-SelectTraining,]

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

```
## 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 training set
trainPredict = predict(GPfitFraud,trainingData[,1:2])
confusionMatrix = table(trainPredict, trainingData[,5])
trainAccuracy = sum(diag(confusionMatrix))/sum(confusionMatrix)
cat('The accuracy of prediction on the training dataset is: ',trainAccuracy, '\n\n')
```

```
## The accuracy of prediction on the training dataset is:  0.932
```

```r
cat('The confusion matrix for the training dataset is: \n')
```

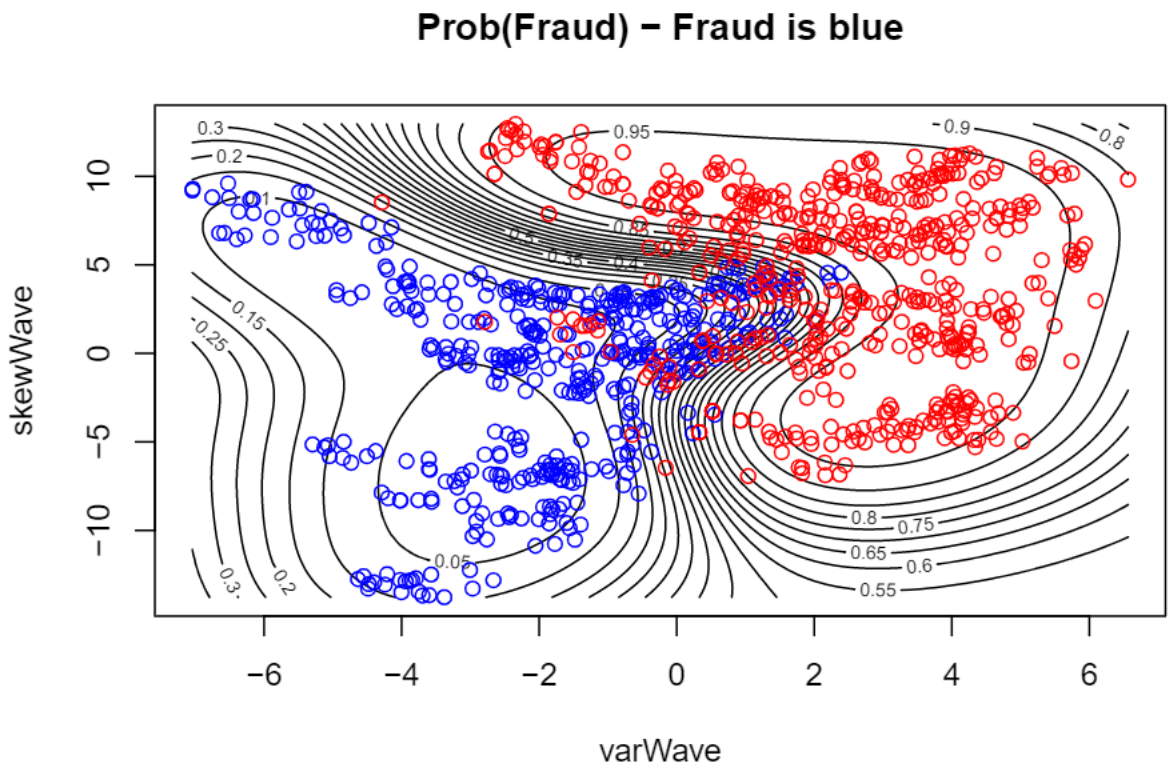
```
## The confusion matrix for the training dataset is:
```

```r
print(confusionMatrix)
```

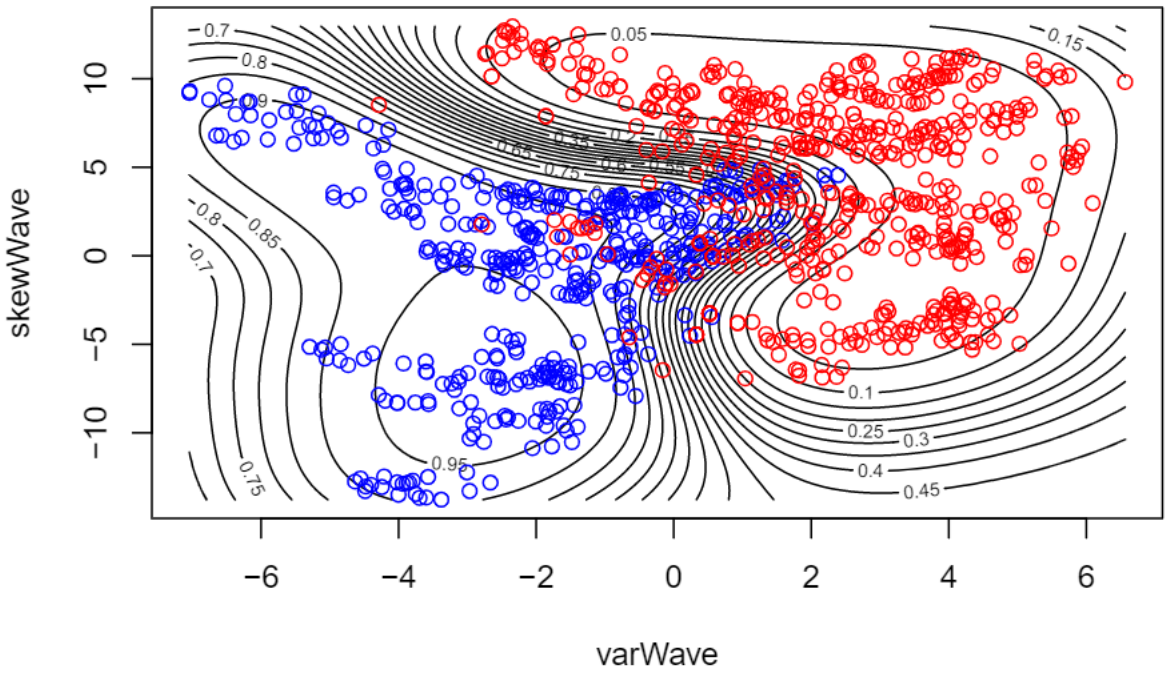
```
##
## trainPredict   0   1
##            0 512  24
##            1  44 420
```

```r
#Plotting class probabilities
# class probabilities
probPreds <- predict(GPfitFraud, trainingData[,1:2], type="probabilities")
x1 <- seq(min(trainingData[,1]),max(trainingData[,1]),length=100)
x2 <- seq(min(trainingData[,2]),max(trainingData[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

# Plotting for Prob(Fraud)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE),
        20,
```

```
        xlab = "varWave",
        ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is blue')
points(trainingData[trainingData[,5]==1,1],trainingData[trainingData[,5]==1,2],col="blue")
points(trainingData[trainingData[,5]==0,1],trainingData[trainingData[,5]==0,2],col="red")
```




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



Question3-2:

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

```
testPredict = predict(GPfitFraud,testingData[,1:2])
confusionMatrix = table(testPredict, testingData[,5])
testAccuracy = sum(diag(confusionMatrix))/sum(confusionMatrix)
cat('The accuracy for the testing dataset is: ', testAccuracy)
```

```
## The accuracy for the testing dataset is:  0.9354839
```

Question3-3:

Train a model using all four covariates. Make predictions on the test set and compare the accuracy to the model with only two covariates.

```
GPfitFraudAll <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=trainingData)
```

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

```
GPfitFraudAll
```

```
## 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
```

```r
testAllPredict = predict(GPfitFraudAll,testingData[,1:4])
confusionMatrixAll = table(testAllPredict, testingData[,5])
testAllAccuracy = sum(diag(confusionMatrixAll))/sum(confusionMatrixAll)
cat('The test data accuracy when fitting the model with all the data is: ',testAllAccuracy)
```

```
## The test data accuracy when fitting the model with all the data is:  0.9973118
```

When fitting the model with all the data, the accuracy of prediction for the testing dataset increases up to 99 percent, and that's because the model uses all the variable effecting the decision of whether a banknote is fraud or not, whereas in the first model the model used only two variables out of four.

References

Course Documents
https://stackoverflow.com/
https://rpruim.github.io/s341/S19/from-class/MathinRmd.html http://archive.ics.uci.edu/ml/datasets/banknote+authentication

Appendix

```r
RNGversion('3.5.1')
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
library(AtmRay)
library(mvtnorm)
set.seed(111)
#The squared exponential function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=0.3){
  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)
}
sigmaF = 1
l = 0.3
sigmaNoise = 0.1
xGrid <- seq(-1,1,length=20)
nSim = 1


#Question1:  The posterior function
posteriorGP = function(X, y, XStar, sigmaNoise, k){
  #The algorithm
  K = k(X, X, sigmaF = sigmaF, l = l)
  n = length(X)

  L = t(chol(K + sigmaNoise^2*diag(n)))
```

```
  alpha = solve(t(L),solve(L,y))
  KStar = k(X,XStar, sigmaF = sigmaF, l = l)
  FStar = t(KStar)%*%alpha
  v = solve(L,KStar)
  VFStar = k(XStar, XStar, sigmaF = sigmaF, l = l)-t(v)%*%v

  logP = -0.5*t(y)%*%alpha - sum(log(diag(L))) - (n/2)*log(2*pi)

  return(list('mean'=FStar, 'variance'= VFStar, 'logMargLik' = logP))
}

#Question2: evaluating on (0.4,0.719)
x = 0.4
y = 0.719
posteriorValue = posteriorGP(X = x,
                             y = y,
                             XStar = xGrid,
                             sigmaNoise = sigmaNoise,
                             k = SquaredExpKernel)
postMean = posteriorValue$mean
postVar = posteriorValue$variance

#Function for plotting the posterior mean
plotMean = function(mean,var, title){
  plot(xGrid, mean, type="p", ylab="f(x)", xlab="x", ylim = c(-2,2), main = title)

  #Plotting confidence band
  lines(xGrid, mean - 1.96*sqrt(diag(var)), col = "blue", lwd = 2)
  lines(xGrid, mean + 1.96*sqrt(diag(var)), col = "blue", lwd = 2)
}
plotMean(postMean,postVar, title = 'x = 0.4, y = 0.719')
x = c(0.4,-0.6)
y = c(0.719,-0.044)

posteriorValue2 = posteriorGP(X = x,
                              y = y,
                              XStar = xGrid,
                              sigmaNoise = sigmaNoise,
                              k = SquaredExpKernel)
postMean2 = posteriorValue2$mean
postVar2 = posteriorValue2$variance

plotMean(postMean2, postVar2, title = 'x = (0.4,-0.6), y =(0.719,-0.044)')



#Question4: The posterior with five observations
x = c(0.4, -0.6, -1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.719, -0.044, 0.768, -0.044, -0.940, 0.719, -0.664)

posteriorValue3 = posteriorGP(X = x,
                              y = y,
                              XStar = xGrid,
```

```
                                sigmaNoise = sigmaNoise,
                                k = SquaredExpKernel)
postMean3 = posteriorValue3$mean
postVar3 = posteriorValue3$variance


plotMean(postMean3,postVar3, title = "All X and Y plot")


#Question5: Repeating with different hyperparameters.

sigmaF = 1
l = 1
posteriorValue4 = posteriorGP(X = x,
                                y = y,
                                XStar = xGrid,
                                sigmaNoise = sigmaNoise,
                                k = SquaredExpKernel)
postMean4 = posteriorValue4$mean
postVar4 = posteriorValue4$variance

plotMean(postMean4,postVar4, title = "All X and Y plot (sigmaF = 1, L = 1)")

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c

date = data$date
time = c(1:dim(data)[1])
temp = data$temp
day = rep(c(1:365), length(time)/365)

#Takinng every fifth observation and scaling time

time = seq(from = 1, to = length(time), by = 5)
day = day[time]
scaledTime = (time-mean(time))/sd(time)
Matern32 <- function(sigmaf = 1, ell = 1)
{
  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)
}

#Evaluating the function
MaternFunc = Matern32(sigmaf = 20, ell = 0.2) # MaternFunc is a kernel FUNCTION
cat('The function evaluation at the point (1,2) is: ',MaternFunc(1,2))

#Computing the covarance matrix K
X = c(1,3,4)
Xstar = c(2,3,4)
```

```
K <- kernelMatrix(kernel = MaternFunc, x = X, y = Xstar)
K
#Letting sigma noise equal to residual variance of a quadratic regression fit using lm
modelData = data.frame(data$temp[time], scaledTime)
colnames(modelData) = c('temp', 'time')
polyFit <- lm(temp ~ time + I(time^2) + I(time^3), data = modelData)
sigmaNoise = sd(polyFit$residuals)
cat('sigmaNoise is: ',sigmaNoise)

plot(time,data$temp[time], ylim = c(-25,35), xlab = 'time', ylab = 'temp')

# Fit the GP with Square expontial kernel
MaternFunc = Matern32(sigmaf = 20, ell = 0.2)
GPfit <- gausspr(scaledTime, data$temp[time],
                 kernel = MaternFunc,
                 kpar = list(sigmaf = 20, ell = 0.2),
                 var = sigmaNoise^2)
meanPred <- predict(GPfit, scaledTime)
lines(time, meanPred, col="blue", lwd = 2)

ell <- 0.3
SEkernel <- rbfdot(sigma = 1/(2*ell^2))

x<-time
xs<-time
n <- length(x)
Kss <- kernelMatrix(kernel = SEkernel, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = SEkernel, x = x, y = x)
Kxs <- kernelMatrix(kernel = SEkernel, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for fStar
plot(time,data$temp[time], ylim = c(-25,35), xlab = 'time', ylab = 'temp')
lines(time, meanPred, col="blue", lwd = 2)


lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)

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

legend("bottomright",
       legend = c('Post Mean Model1'),
       col = c( 'blue'),
       lty=1:2,
       cex=0.8)
legend("topright",
       legend = c('prob interval',
                  'pred interval'),
       col = c('green', 'brown'),
       lty=1:2,
```

```
       cex=0.8,
       horiz = TRUE)

#Letting sigma noise equal to residual variance of a quadratic regression fit using lm
modelData = data.frame(data$temp[time], day)
colnames(modelData) = c('temp', 'day')
polyFit <- lm(temp ~ day + I(day^2) + I(day^3), data = modelData)
sigmaNoise = sd(polyFit$residuals)
cat('sigmaNoise is: ',sigmaNoise)

# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)

MaternFunc = Matern32(sigmaf = 20, ell = 0.2)
GPfit <- gausspr(day,
                 data$temp[time],
                 kernel = MaternFunc,
                 kpar = list(sigmaf = 20, ell = 0.2),
                 var = sigmaNoise^2)
meanPred2 <- predict(GPfit, day)

plot(time,data$temp[time], ylim = c(-25,35), xlab = 'time', ylab = 'temp')
lines(time, meanPred, col="blue", lwd = 2)
# Probability intervals for fStar
lines(xs, meanPred2 - 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)
lines(xs, meanPred2 + 1.96*sqrt(diag(Covf)), col = "green", lwd = 2)

# Prediction intervals for yStar
lines(xs, meanPred2 - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "brown")
lines(xs, meanPred2 + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "brown")
lines(time, meanPred2, col="red", lwd = 2)
legend("bottomright",
       legend = c('Post Mean Model1',
                  'Post Mean Model2'),
       col = c( 'blue', 'red'),
       lty=1:2,
       cex=0.8)
legend("topright",
       legend = c('prob interval',
                  'pred interval'),
       col = c('green', 'brown'),
       lty=1:2,
       cex=0.8,
       horiz = TRUE)



GPKernel = function(l1,l2, sigmaF, d){

  rval <- function(x1, x2 = NULL) {
    return(sigmaF^2 * exp(-(2*(sin(pi*abs(x1-x2)/d))^2)/l1^2)*(exp(-0.5*abs(x1-x2)^2)/l2^2))
  }
  class(rval) <- "kernel"
  return(rval)
```

```
}

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

GPKernelFun = GPKernel(l1,l2,sigmaF, d)
cat('By evaluating the kernel method on the point (1,2) we get',GPKernelFun(1,2))

#Fitting the generlaized model
GPfit <- gausspr(scaledTime, data$temp[time],
                 kernel = GPKernelFun,
                 kpar = list(sigmaF = sigmaF, l1 = l1, l2 = l2, d=d),
                 var = sigmaNoise^2)
meanPred3 <- predict(GPfit, scaledTime)
plot(time,data$temp[time], ylim = c(-25,30), main = 'Generalized Model', xlab = 'time', ylab = 'temp')
lines(time, meanPred3, col="blue", lwd = 2)
lines(time, meanPred, col="red", lwd = 2)
lines(time, meanPred2, col="green", lwd = 2)
legend("bottomright",
       legend = c('Generalized Fit',
                  'Model 1 Fit',
                  'Model 2 Fit'),
       col = c('blue', 'red', 'green'),
       lty=1:2,
       cex=0.8,
       horiz = TRUE)


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])

#Dividing the data into training and testing
set.seed(111)
SelectTraining = sample(1:dim(data)[1],
                        size = 1000,
                        replace = FALSE)
trainingData = data[SelectTraining,]
testingData = data[-SelectTraining,]

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

# predict on the training set
trainPredict = predict(GPfitFraud,trainingData[,1:2])
confusionMatrix = table(trainPredict, trainingData[,5])
trainAccuracy = sum(diag(confusionMatrix))/sum(confusionMatrix)
cat('The accuracy of prediction on the training dataset is: ',trainAccuracy, '\n\n')

cat('The confusion matrix for the training dataset is: \n')
print(confusionMatrix)
```

```r
#Plotting class probabilities
# class probabilities
probPreds <- predict(GPfitFraud, trainingData[,1:2], type="probabilities")
x1 <- seq(min(trainingData[,1]),max(trainingData[,1]),length=100)
x2 <- seq(min(trainingData[,2]),max(trainingData[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

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

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

testPredict = predict(GPfitFraud,testingData[,1:2])
confusionMatrix = table(testPredict, testingData[,5])
testAccuracy = sum(diag(confusionMatrix))/sum(confusionMatrix)
cat('The accuracy for the testing dataset is: ', testAccuracy)


GPfitFraudAll <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=trainingData)
GPfitFraudAll

testAllPredict = predict(GPfitFraudAll,testingData[,1:4])
confusionMatrixAll = table(testAllPredict, testingData[,5])
testAllAccuracy = sum(diag(confusionMatrixAll))/sum(confusionMatrixAll)
cat('The test data accuracy when fitting the model with all the data is: ',testAllAccuracy)
```