Report

Aman Kumar Nayak

10/13/2020

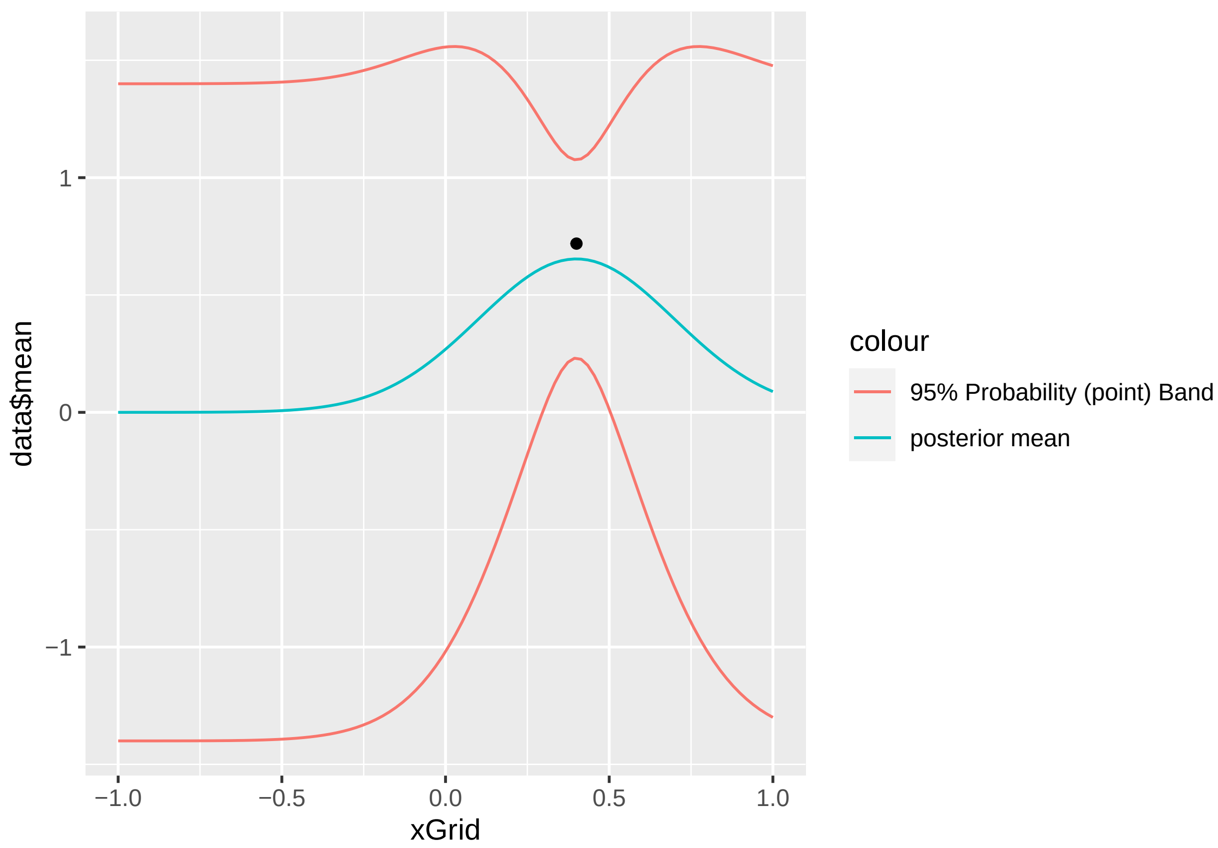
#1. Implementing GP Regression

1.1

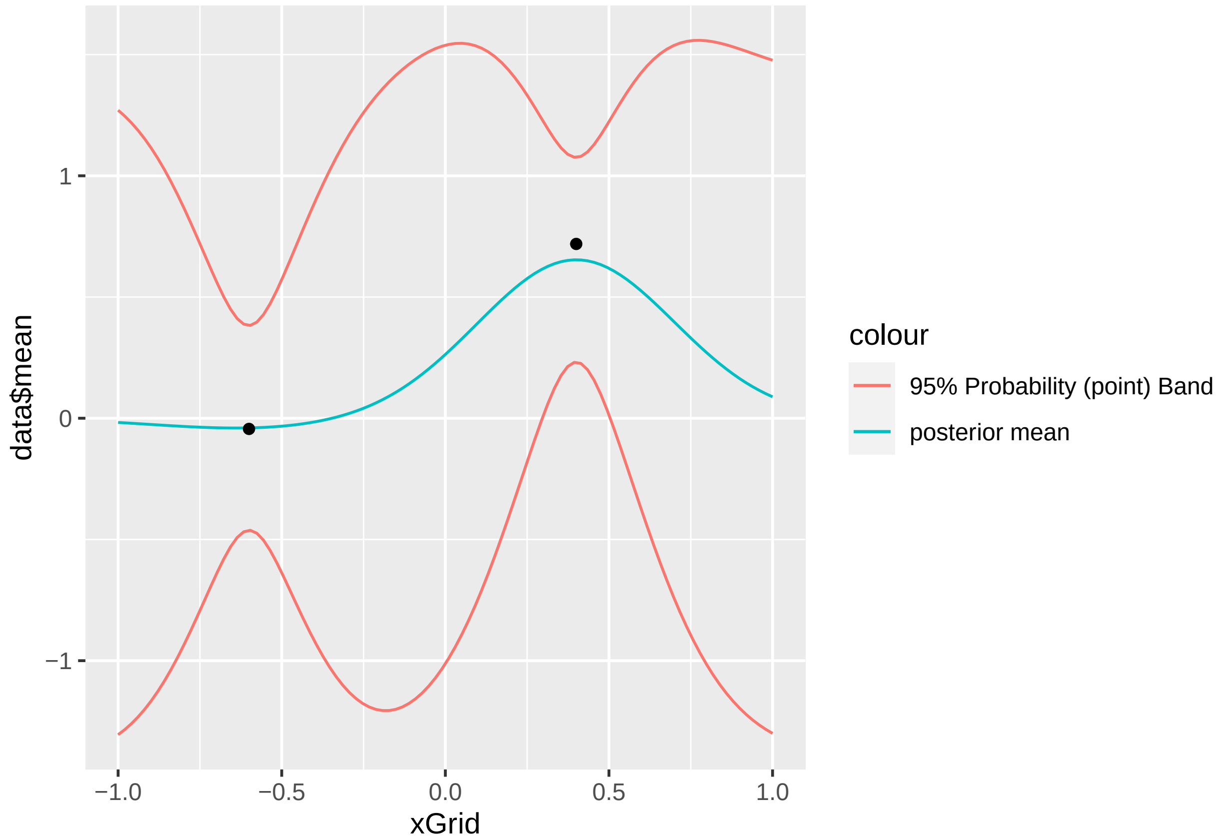
```
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}


posteriorGP = function(X , y , XStar, sigmaNoise , sigmaF , l){
  #Inputs
  #X: Vector of training inputs.
  #y: Vector of training targets/outputs.
  #XStar: Vector of inputs where the posterior distribution is evaluated
  #sigmaNoise: Noise standard deviation

  #k: Covariance function or kernel. #SquaredExpKernel
   k = SquaredExpKernel(x1 = X, x2 = X , sigmaF = sigmaF , l = l)

   L_upper = chol(k + (sigmaNoise * diag(length(diag(k)))))

   #since as per documentation of chol function, it returns upper triangle,
   #we have to take transpose of it

   L = t(L_upper)

   #now in order to calculate alpha we have alpha = L.Transpose / (L/y)
   #now using Ax = b => x = b/A so to find L/y solution we can use solve function

   L_by_y = solve(L , y)

   #now trans(x.trans) = x so we can use L_Upper directly
   alpha = solve(L_upper , L_by_y)
```

```r
  K_Star = SquaredExpKernel(x1 = X, x2 = XStar , sigmaF = sigmaF , l = l)

  #predicted mean
  f.Star = t(K_Star) %*% alpha

  v = solve(L , K_Star)
  #predicted Variance
  V_f.Star = SquaredExpKernel(x1 = XStar, x2 = XStar ,
                              sigmaF = sigmaF , l = l) - (t(v) %*% v)

  #taking diagnol elements of covariance matrix for variance in ii
  V_f.Star = diag(V_f.Star)
  #log marginal likelihood
  #logMargLikeli = -(0.5 * t(y) %*% alpha) -

  return(list("mean" = f.Star , "Variance" = V_f.Star))
}#posteriorDist
```

1.2

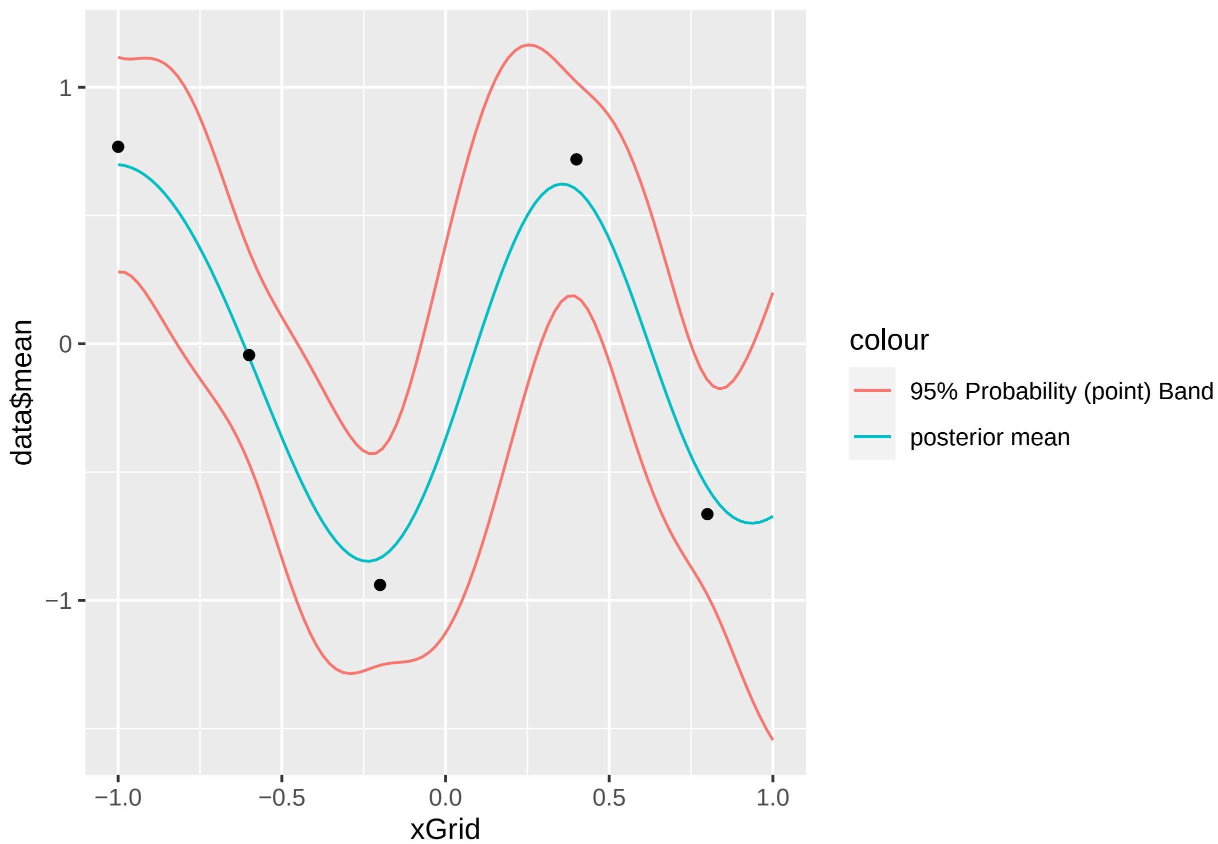


1.3

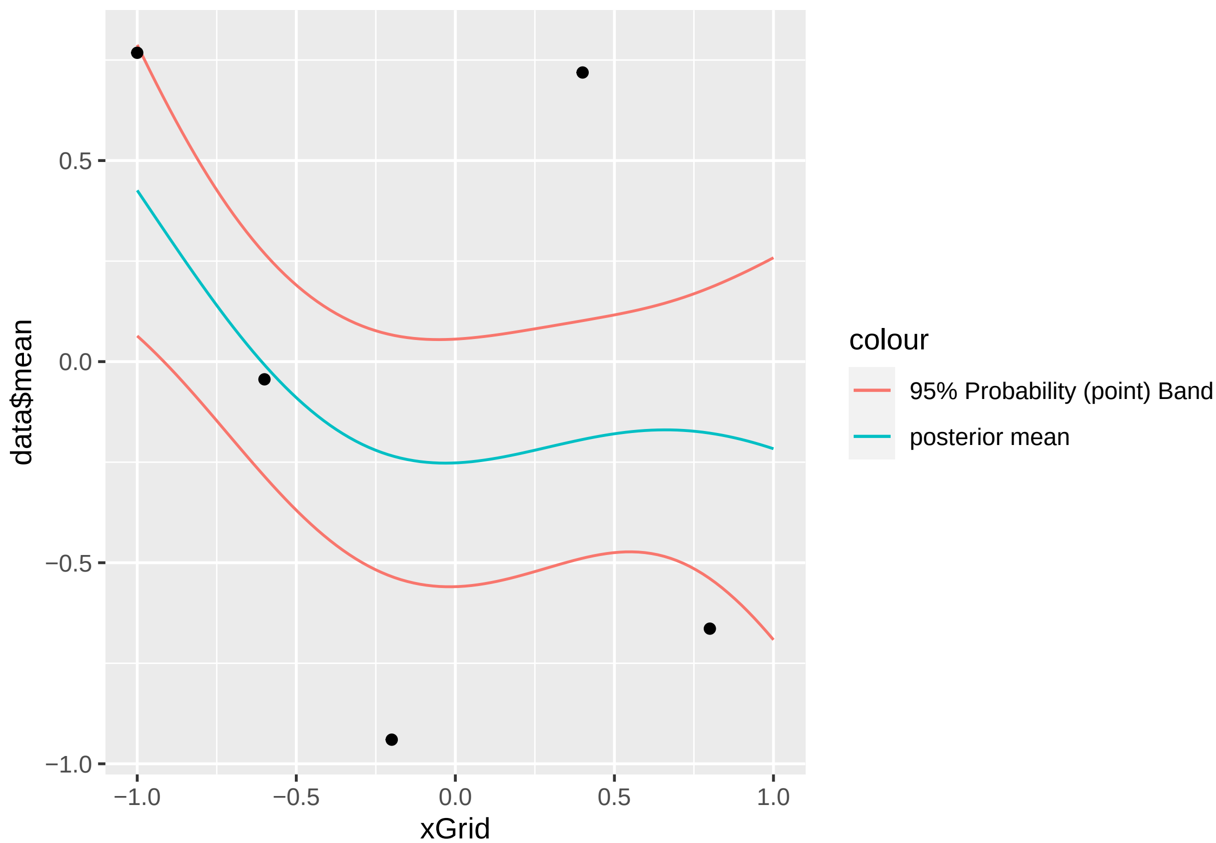


1.4

```
##          Table of data points

##      [,1]   [,2]  [,3]  [,4]   [,5]
## x -1.000 -0.600 -0.20 0.400  0.800
## y  0.768 -0.044 -0.94 0.719 -0.664
```


1
0
−1
data$mean
−1.0
−0.5
0.0
0.5
1.0
xGrid
colour
95% Probability (point) Band
posterior mean

1.5

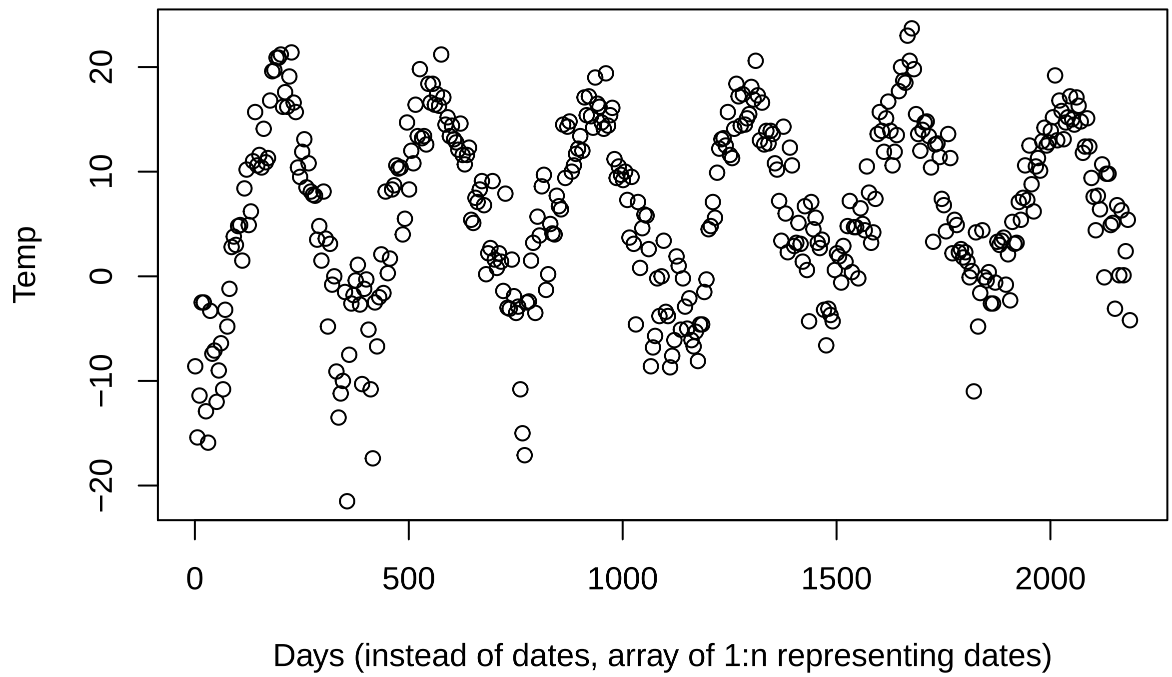


#2. GP Regression with kernlab

2.1

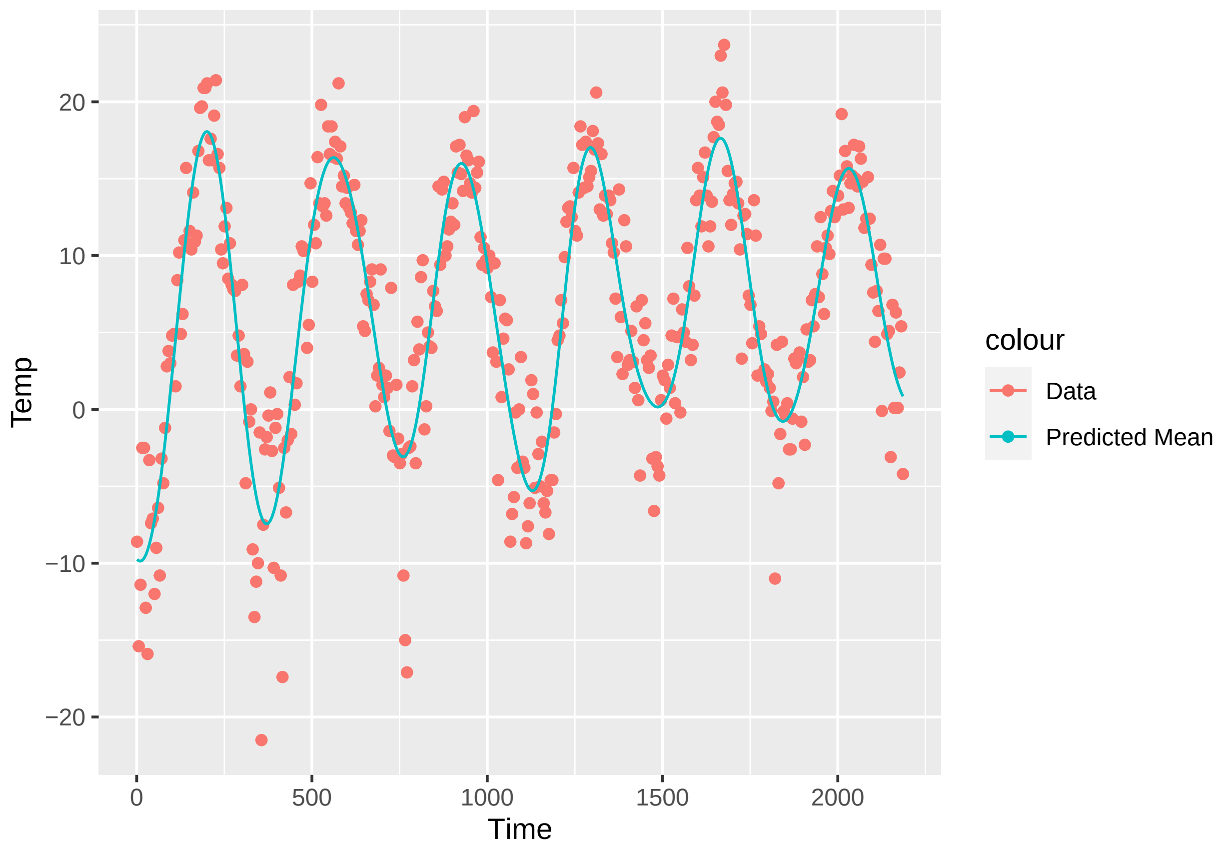
```
library(kernlab)
library(AtmRay)
#import data

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

#create vector for time and date
#as per task comments
# Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the
# only every fifth observation. This means that your time and day variables are now time= 1, 6, 11, . .
# day = 1, 6, 11, . . ., 361, 1, 6, 11, . . ., 361.
n = length(data$date)
time = seq(1 , n , by = 5)
#leap year consideration is not catered for simplicity reasons
data.Day = rep(seq(1 , 365 , by = 5) , times =  (n/365))
data.Temp = data$temp[time]



plot(time , data.Temp , xlab = "Days (instead of dates, array of 1:n representing dates)" ,
     ylab = "Temp" , main = "Selected Data"
     )
```

Selected Data



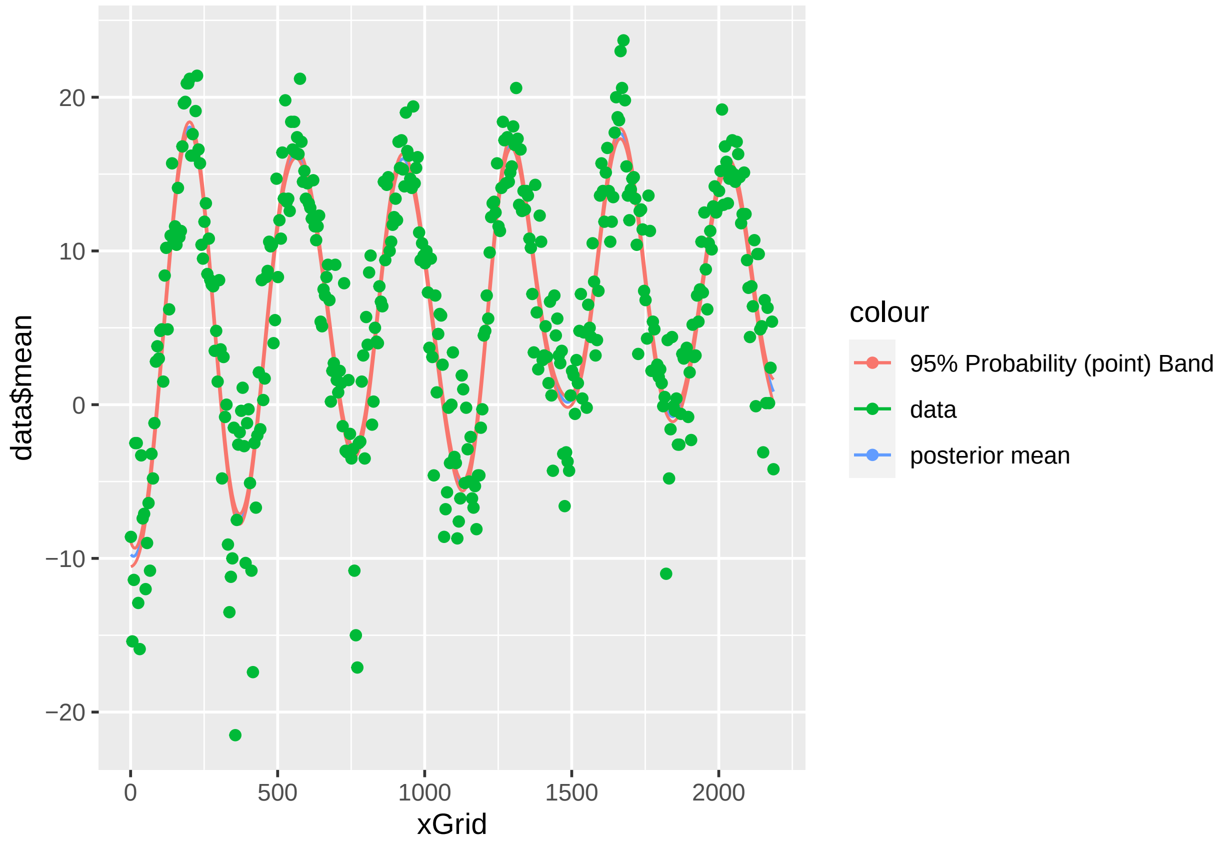

```
## Computed CoVariance Matrix is :

## 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



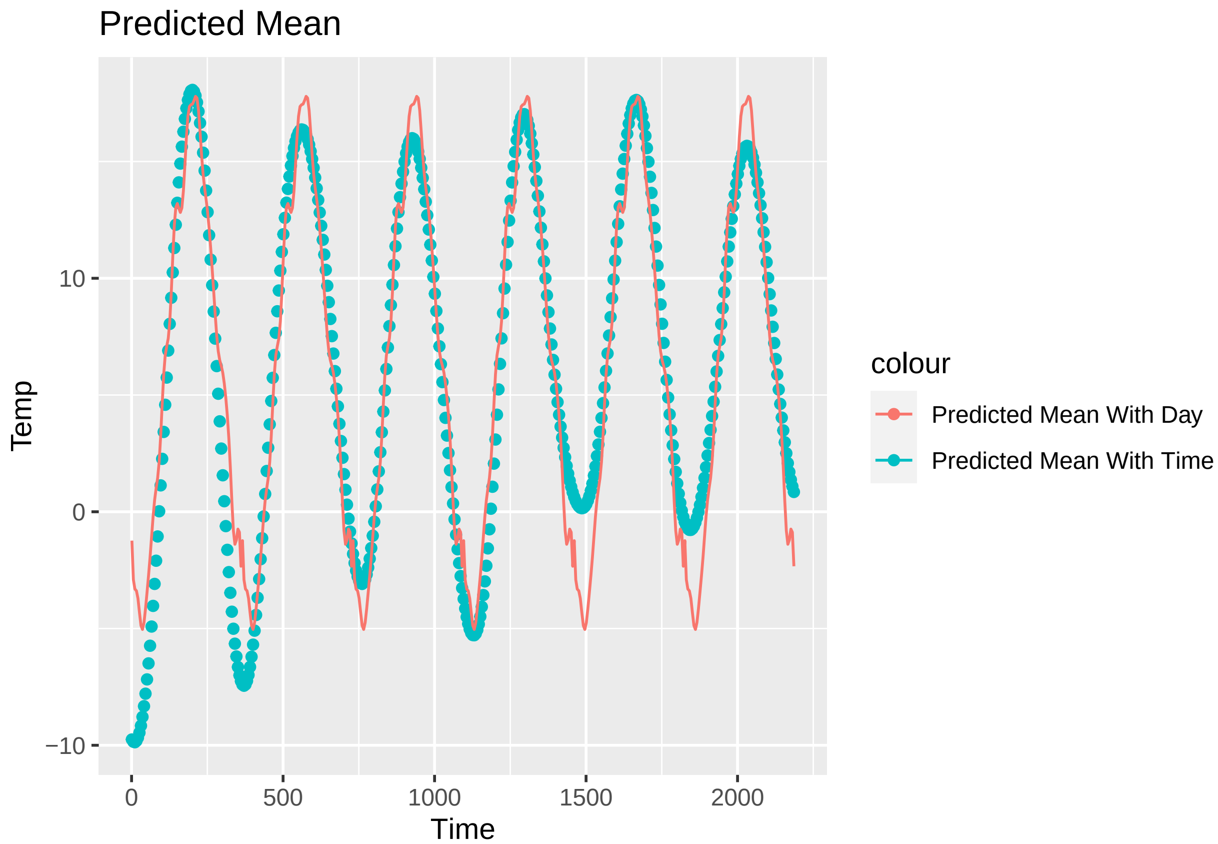
Temp
20
10
0
−10
−20
0
500
1000
1500
2000
Time
colour
Data
Predicted Mean

2.3

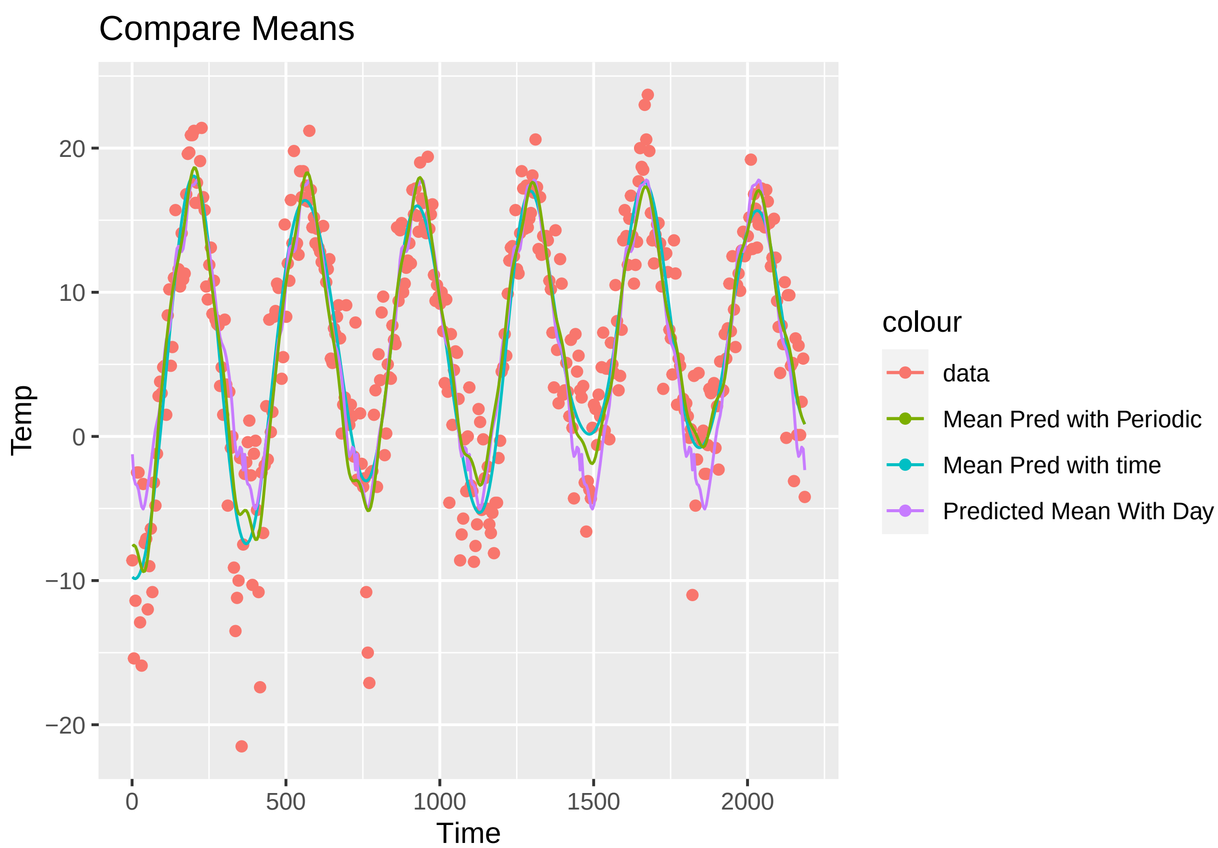


data$mean
20
10
0
−10
−20
0
500
1000
1500
2000
xGrid
colour
95% Probability (point) Band
data
posterior mean

1.4


Predicted Mean
Temp
Time
colour
Predicted Mean With Day
Predicted Mean With Time
10
0
−10
0
500
1000
1500
2000

1.5



With mean value predicted with day values we can that it is repeating in cycle which could be understood in terms of repeating value of days in band of 1 to 365 while mean predicted by Time is very smooth but it does not follow shift in data mass which is accurately captured in mean calculated by periodic kernel.

GP Classification with kernlab

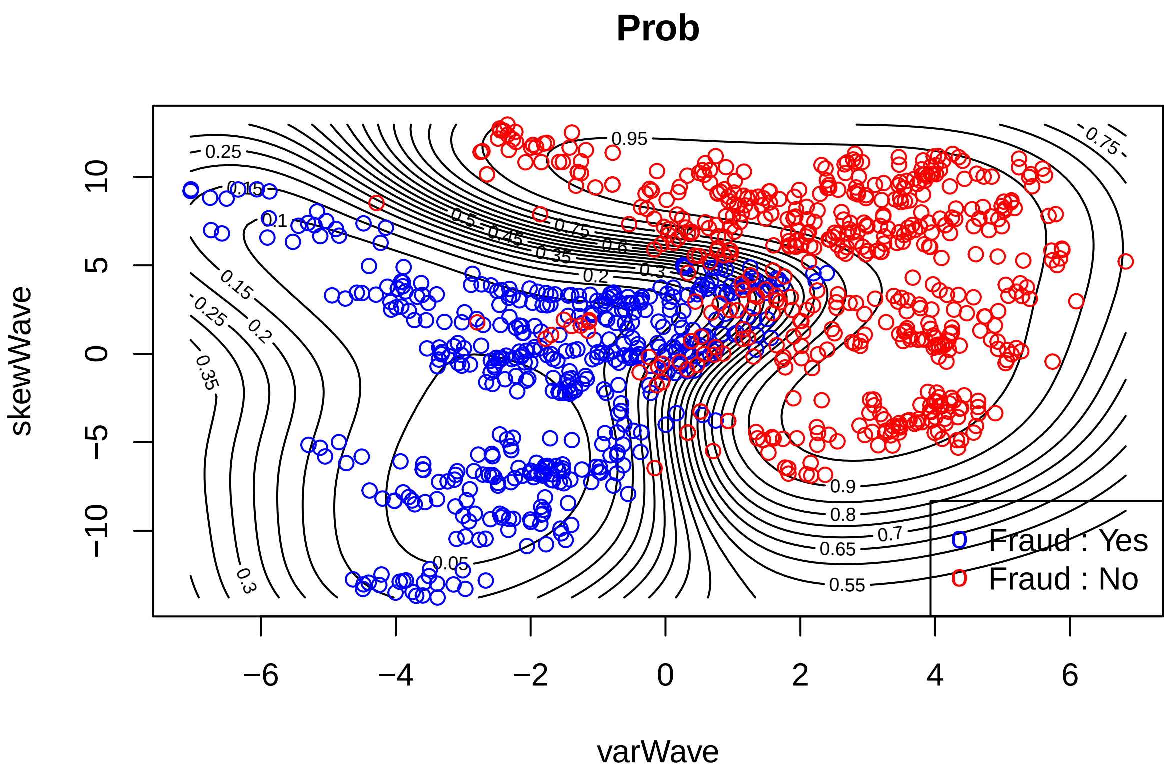
3.1

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

```
## Confusion Matrix for Train Data is :
```

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

```
## Accuracy with Train data is :  0.941
```



3.2

```
## Confusion Matrix for Test Data is :

##
## fraudPred.Test   0   1
##              0 199   9
##              1  19 145

## Accuracy with Test data is :  0.9247312
```

3.3

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

## Confusion Matrix for Test Data using all parameter is :

##
## fraudPred.All   0   1
##             0 216   0
##             1   2 154

## Accuracy for Test data  with all parameters is :  0.9946237
```

We can see test accuracy with test data when only two features were used is 92.42% while when all feature are used it is around 99%. It can be seen that only two features namely varWave and skewWave are important.

```
## rpart variable importance
##
##             Overall
## varWave      100.00
## skewWave      72.53
## kurtWave      11.45
## entropyWave    0.00
```

From above importance values, we can reconfirm our belief that only 2 parameters are majorly important and last one is not at all useful.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
# 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 , sigmaF , l){
  #Inputs
  #X: Vector of training inputs.
  #y: Vector of training targets/outputs.
  #XStar: Vector of inputs where the posterior distribution is evaluated
  #sigmaNoise: Noise standard deviation

  #k: Covariance function or kernel. #SquaredExpKernel
   k = SquaredExpKernel(x1 = X, x2 = X , sigmaF = sigmaF , l = l)

   L_upper = chol(k + (sigmaNoise * diag(length(diag(k)))))

   #since as per documentation of chol function, it returns upper triangle,
   #we have to take transpose of it

   L = t(L_upper)

   #now in order to calculate alpha we have alpha = L.Transpose / (L/y)
   #now using Ax = b => x = b/A so to find L/y solution we can use solve function

   L_by_y = solve(L , y)

   #now trans(x.trans) = x so we can use L_Upper directly
   alpha = solve(L_upper , L_by_y)
```

```
   K_Star = SquaredExpKernel(x1 = X, x2 = XStar , sigmaF = sigmaF , l = l)

   #predicted mean
   f.Star = t(K_Star) %*% alpha

   v = solve(L , K_Star)
   #predicted Variance
   V_f.Star = SquaredExpKernel(x1 = XStar, x2 = XStar ,
                               sigmaF = sigmaF , l = l) - (t(v) %*% v)

   #taking diagnol elements of covariance matrix for variance in ii
   V_f.Star = diag(V_f.Star)
   #log marginal likelihood
   #logMargLikeli = -(0.5 * t(y) %*% alpha) -

   return(list("mean" = f.Star , "Variance" = V_f.Star))
}#posteriorDist



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

#prior
xPrior = 0.4
yPrior = 0.719

#hyperParameter
sigmaF = 1
l = 0.3

SigmaN = 0.1

posteriorF = posteriorGP(X = xPrior , y = yPrior , XStar = xGrid ,
                          sigmaNoise = SigmaN ,
                          sigmaF = sigmaF , l = l
                         )


#95% Z = 1.96
library(ggplot2)

fnPlot = function(data, xGrid , main.T , xhigh , yhigh , high = TRUE,
                  XStarPlot = FALSE, OrigData = NA ){
   l.band = data$mean - sqrt(1.96 * data$Variance)
   u.band = data$mean + sqrt(1.96 * data$Variance)

   p = ggplot()+
      geom_line(aes(x = xGrid , y = data$mean , color = "posterior mean"))+
      geom_line(aes(x = xGrid , y = l.band , color = "95% Probability (point) Band"))+
      geom_line(aes(x = xGrid , y = u.band , color = "95% Probability (point) Band"))

   if(high == TRUE)
      p = p + geom_point(aes(x = xhigh , y = yhigh ))
```

```r
  if(XStarPlot == TRUE)
    p = p + geom_point(aes(x = xGrid , y = OrigData , color = "data"))

  p + xlab("X Grid") + ylab("Posterior Mean") +
    ggtitle(main.T)

  return(p)

}#fnPlot


fnPlot(data = posteriorF , xGrid = xGrid , main.T = "Posterior Mean" ,
       xhigh = 0.4 , yhigh = 0.719 )



xNew = c(0.4 , -0.6 )
yNew = c(0.719 , -0.044)

posteriorF2 = posteriorGP(X = xNew , y = yNew , XStar = xGrid ,
                          sigmaNoise = SigmaN ,
                          sigmaF = sigmaF , l = l
                         )

fnPlot(data = posteriorF2 , xGrid = xGrid , main.T = "Posterior Mean" ,
       xhigh = xNew  , yhigh = yNew )



data = c(-1,-0.6,-0.2,0.4,0.8,0.768,-0.044,-0.940,0.719,-0.664)

data = matrix(data,nrow =  2 , byrow = TRUE)
rownames(data) = c("x" , "y")

cat("\n")
cat("        Table of data points")
cat("\n")
cat("\n")
data
cat("\n")

posteriorF2 = posteriorGP(X = data[1,] , y = data[2,] , XStar = xGrid ,
                          sigmaNoise = SigmaN ,
                          sigmaF = sigmaF , l = l
                         )

fnPlot(data = posteriorF2 , xGrid = xGrid , main.T = "Posterior Mean" ,
       xhigh = data[1,]  , yhigh = data[2,] )


#updated l to 1 and sigmaF is already 1

posteriorF2 = posteriorGP(X = data[1,] , y = data[2,] , XStar = xGrid ,
```

```r
                             sigmaNoise = SigmaN ,
                             sigmaF = sigmaF , l = 1
                            )

fnPlot(data = posteriorF2 , xGrid = xGrid , main.T = "Posterior Mean" ,
       xhigh = data[1,]  , yhigh = data[2,] )


library(kernlab)
library(AtmRay)
#import data

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

#create vector for time and date
#as per task comments
# Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the
# only every fifth observation. This means that your time and day variables are now time= 1, 6, 11, . .
# day = 1, 6, 11, . . ., 361, 1, 6, 11, . . ., 361.
n = length(data$date)
time = seq(1 , n , by = 5)
#leap year consideration is not catered for simplicity reasons
data.Day = rep(seq(1 , 365 , by = 5) , times =  (n/365))
data.Temp = data$temp[time]



plot(time , data.Temp , xlab = "Days (instead of dates, array of 1:n representing dates)" ,
     ylab = "Temp" , main = "Selected Data"
     )
#given
x = 1
x2 = 2
X = c(1,3,4)
XStar = c(2,3,4)
library(kernlab)
#ell = l
SqrExpFn <- function(sigmaf = 1, l = 1)
{
   SquaredExpKernel2 <- function(x , y)
   {
     n1 <- length(x)
     n2 <- length(y)
     K <- matrix(NA,n1,n2)
     for (i in 1:n2)
       K[,i] <- sigmaf^2*exp(-0.5*( (x-y[i])/l)^2 )

     return(K)
   }#SquaredExpKernel2

   class(SquaredExpKernel2) <- "kernel"

   return(SquaredExpKernel2)
```

```r
}#SqrExpFn

#to compute covariance matrix
K = kernelMatrix(kernel = SqrExpFn() , x = X , y = XStar)

cat("Computed CoVariance Matrix is :")
cat("\n")
K
cat("\n")

#calculate sigmaN as residual variance from a simple quadratic regression fit
#since f is distributed with mean 0 , we have to scale our data

scaledData.Temp = scale(data.Temp)
scaleData.Time = scale(time)

quadReg = lm(scaledData.Temp ~ scaleData.Time + I(scaleData.Time)**2)
SigmaN = sd(quadReg$residuals)

#New
#sigmaF = 20
#l = 0.2

#gausspr is an implementation of Gaussian processes for classification and regression

GaussianModel = gausspr(x = time , y = data.Temp ,
                        kernel = SqrExpFn(sigmaf = 20 , l = 0.2) ,
                        var = SigmaN**2,
                        type = "regression"
                        )


meanPredicted = predict(GaussianModel , time)

ggplot()+
   geom_point(aes(x = time , y = data.Temp , color = "Data"))  +
   geom_line(aes(x = time , y = meanPredicted[,1] , color = "Predicted Mean"))+
   xlab("Time") + ylab("Temp")

#now posterior mean can be calculated using our function : posteriorGP
#as in this function XStar: Vector of inputs where the posterior distribution is evaluated, we can pass

#we have to use scaled data in order to match output of gausspr function
#scaling is already implemented for LinReg Part


#posteriorGP = function(X , y , XStar, sigmaNoise , sigmaF , l)

posterVariance = posteriorGP(X = scaleData.Time , y = scaledData.Temp ,
                             XStar = scaleData.Time , sigmaNoise = SigmaN,
                             sigmaF = 20 , l =  0.2
                             )$Variance
```

```
#unscaledPostVar = posterVariance * sd(posterVariance) + mean(posterVariance) + posterVariance

lPlot = list("mean" = meanPredicted , "Variance" = posterVariance)

fnPlot(data = lPlot , xGrid = time , main.T = "Posterior Mean and Variance" ,
       xhigh = 0   , yhigh =0, high = FALSE , XStarPlot = TRUE , OrigData = data.Temp )


scaleData.Day = scale(data.Day)

quadReg2 = lm(scaledData.Temp ~ scaleData.Day + I(scaleData.Day)**2)
SigmaN2 = sd(quadReg2$residuals)

#New
#sigmaF = 20
#l = 0.2

#gausspr is an implementation of Gaussian processes for classification and regression

GaussianModel2 = gausspr(x = data.Day , y = data.Temp ,
                        kernel = SqrExpFn(sigmaf = 20 , l = 0.2) ,
                        var = SigmaN**2,
                        type = "regression"
                        )


meanPredicted2 = predict(GaussianModel2 , data.Day)

ggplot()+
   geom_point(aes(x = time , y = meanPredicted[,1] , color = "Predicted Mean With Time"))  +
   geom_line(aes(x = time , y = meanPredicted2[,1] , color = "Predicted Mean With Day"))+
   xlab("Time") + ylab("Temp") + ggtitle("Predicted Mean")





SqrExpFn2 <- function(sigmaF , l1 , l2, d )
{
   SquaredExpKernel2 <- function(x1,x2)
   {
      e1 = exp(-(2*sin(pi * abs(x1 - x2) / d)**2) / l1**2)
      e2 = exp(-(0.5 * abs(x1 - x2)**2) / l2**2  )
      K = (sigmaF^2) * e1  * e2
      return(K)
   }

   class(SquaredExpKernel2) <- "kernel"

   return(SquaredExpKernel2)
}#SqrExpFn

d = 365 / sd(time)
```

```
GaussianModel2 = gausspr(x = time , y = data.Temp ,
                         kernel = SqrExpFn2(sigmaF = 20 , l1 = 1 , l2 = 10,
                                            d = d) ,
                         var = SigmaN**2,
                         type = "regression"
                         )


predMean3 = predict(GaussianModel2, time)

ggplot() +
   geom_point(aes(x = time , y = data.Temp , color = "data"))+
   geom_line(aes(x = time , y = meanPredicted[,1] , color = "Mean Pred with time" ))+
   geom_line(aes(x = time , y = meanPredicted2[,1] , color = "Predicted Mean With Day"))+
   geom_line(aes(x = time , y = predMean3[,1] , color = "Mean Pred with Periodic" ))+
   xlab("Time") +
   ylab("Temp")+
   ggtitle("Compare Means")





#import data and ranames columns

regData = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFra

names(regData) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")

regData[,5] <- as.factor(regData[,5])

#split in test and train
set.seed(111);
SelectTraining <- sample(1:dim(regData)[1], size = 1000, replace = FALSE)
train = regData[SelectTraining,]
test = regData[-SelectTraining,]


gpClassification = gausspr(fraud ~ varWave + skewWave , data = train,
                             type = "classification")

fraudPred = predict(gpClassification , train)

CM = table(fraudPred , train$fraud)

Accuracy = sum(diag(CM)) / sum(CM)

cat("\n")
cat("Confusion Matrix for Train Data is : ")
cat("\n")
CM
cat("\n")
```

```r
cat("\n")
cat("Accuracy with Train data is : " , Accuracy)

#inorder to select suitable values we can try with range of values
#between min and max range
#as otherwise it is getting hard to justify min and max values

grid.varWave = seq(min(train$varWave) , max(train$varWave) , length = 100)
grid.skewWave = seq(min(train$skewWave) , max(train$skewWave) , length = 100)

gridPoints <- meshgrid(grid.varWave, grid.skewWave)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]

probPreds <- predict(gpClassification, gridPoints, type="probabilities")


contour( grid.varWave, grid.skewWave , matrix(probPreds[,1],100,byrow = TRUE) ,
        nlevels = 20 ,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob'
        )
points(x = train$varWave[train$fraud == 1] , y = train$skewWave[train$fraud == 1] ,
       col = "blue" )
points(x = train$varWave[train$fraud == 0] , y = train$skewWave[train$fraud == 0] ,
       col = "red")
legend("bottomright", legend = c("Fraud : Yes" , "Fraud : No") ,
       pch=c("o" , "o"),
       col = c("blue" , "red"))


#Test Data

fraudPred.Test = predict(gpClassification , test)

CM.Test = table(fraudPred.Test , test$fraud)

Accuracy.Test = sum(diag(CM.Test)) / sum(CM.Test)

cat("\n")
cat("Confusion Matrix for Test Data is : ")
cat("\n")
CM.Test
cat("\n")
cat("\n")
cat("Accuracy with Test data is : " , Accuracy.Test)


gpClassification.All = gausspr(fraud ~ . , data = train,
                            type = "classification")

fraudPred.All = predict(gpClassification.All , test)
```

```r
CM.All = table(fraudPred.All , test$fraud)

Accuracy.All = sum(diag(CM.All)) / sum(CM.All)


cat("\n")
cat("Confusion Matrix for Test Data using all parameter is : ")
cat("\n")
CM.All
cat("\n")
cat("\n")
cat("Accuracy for Test data  with all parameters is : " , Accuracy.All)

library(caret)
set.seed(100)

Mod = train(fraud ~ . , data = train , method = "rpart")

impFeatures = varImp(Mod)

print(impFeatures)
```