AML_Lab4

CHAO FU

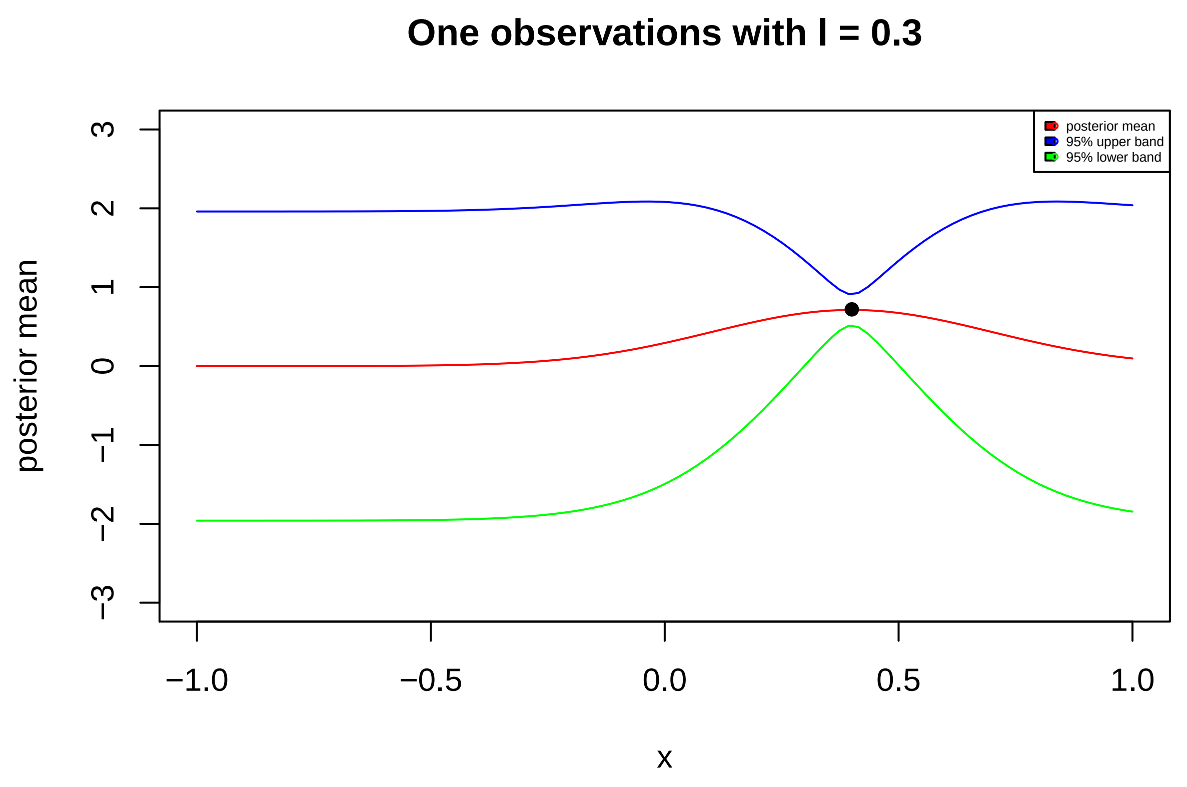
13 October, 2021

1. Implementing GP Regression

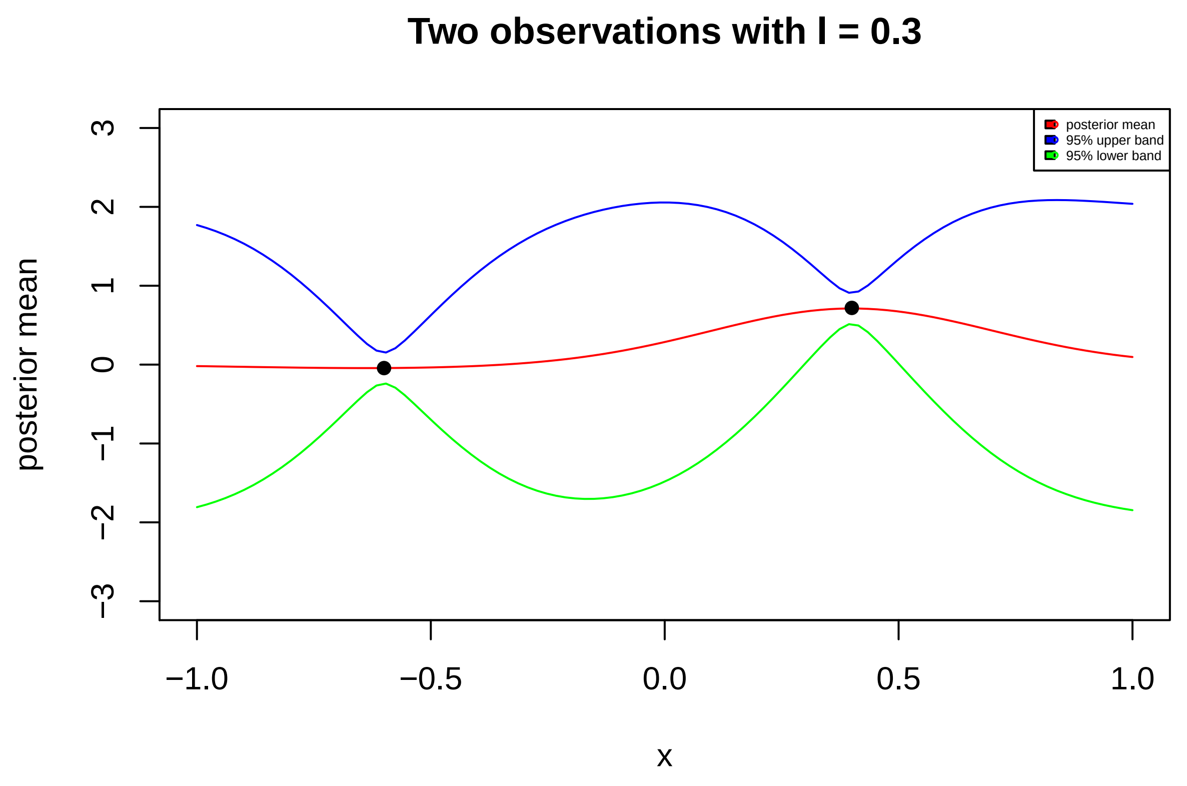
(1)

```r
# set squared exponential kernel function
squared_exp_Kernel <- function(x1, x2, sigmaF, l){
  # 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 )
  # }
  K <- outer(x1, x2, function(x, y) sigmaF ^ 2 * exp(-0.5 * ( (x - y) / l) ^ 2 ))
  return(K)
}

# set posterior Gaussian Process function
posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...){
  # number of training data
  n <- length(X)
  # Algorithm 2.1 in Rasmussen and Williams
  kstar <- k(X, XStar, ...)
  L <- (k(X, X, ...) + sigmaNoise ^ 2 * diag(n)) %>% chol() %>% t()
  alpha <- solve(t(L), solve(L, y))
  fstarmean <- t(kstar) %*% alpha
  v <- solve(L, kstar)
  fstarvar <- k(XStar, XStar, ...) - t(v) %*% v
  result <- list("fmean" = fstarmean, "fvar" = fstarvar)
  return(result)
}
```

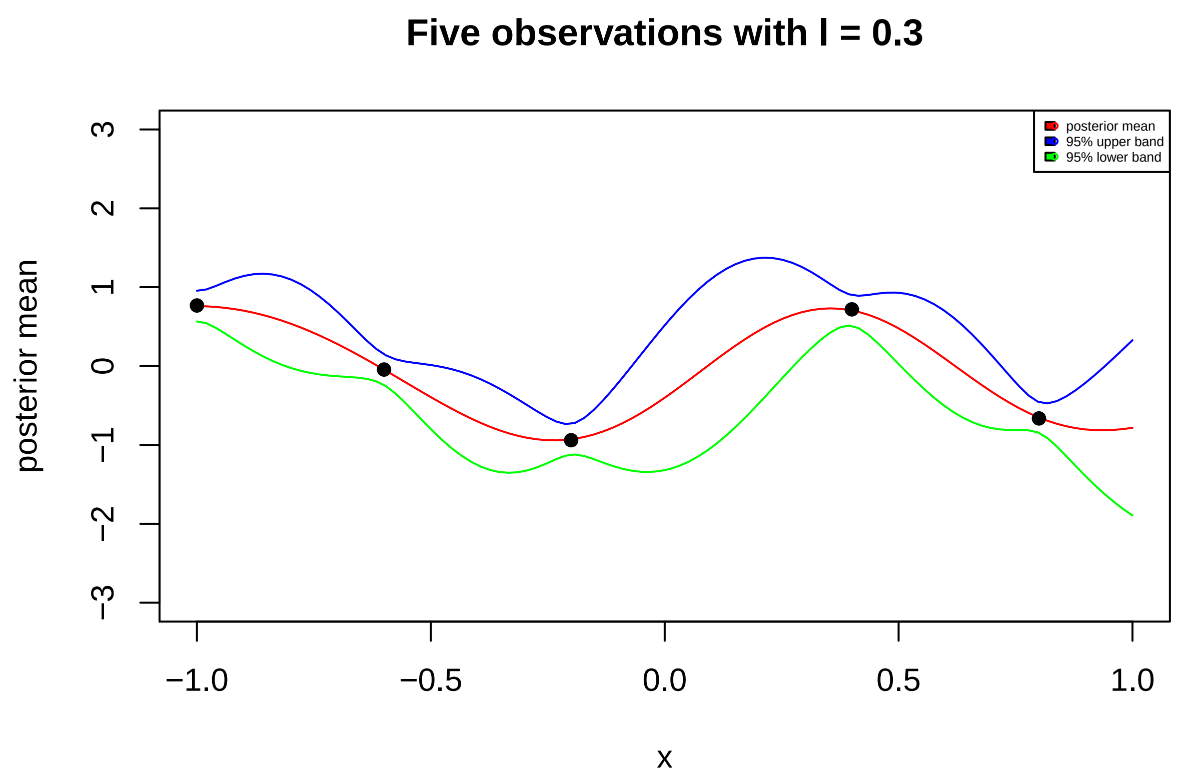
(2)


One observations with l = 0.3
posterior mean
95% upper band
95% lower band
posterior mean
x
−1.0
−0.5
0.0
0.5
1.0
−3
−2
−1
0
1
2
3

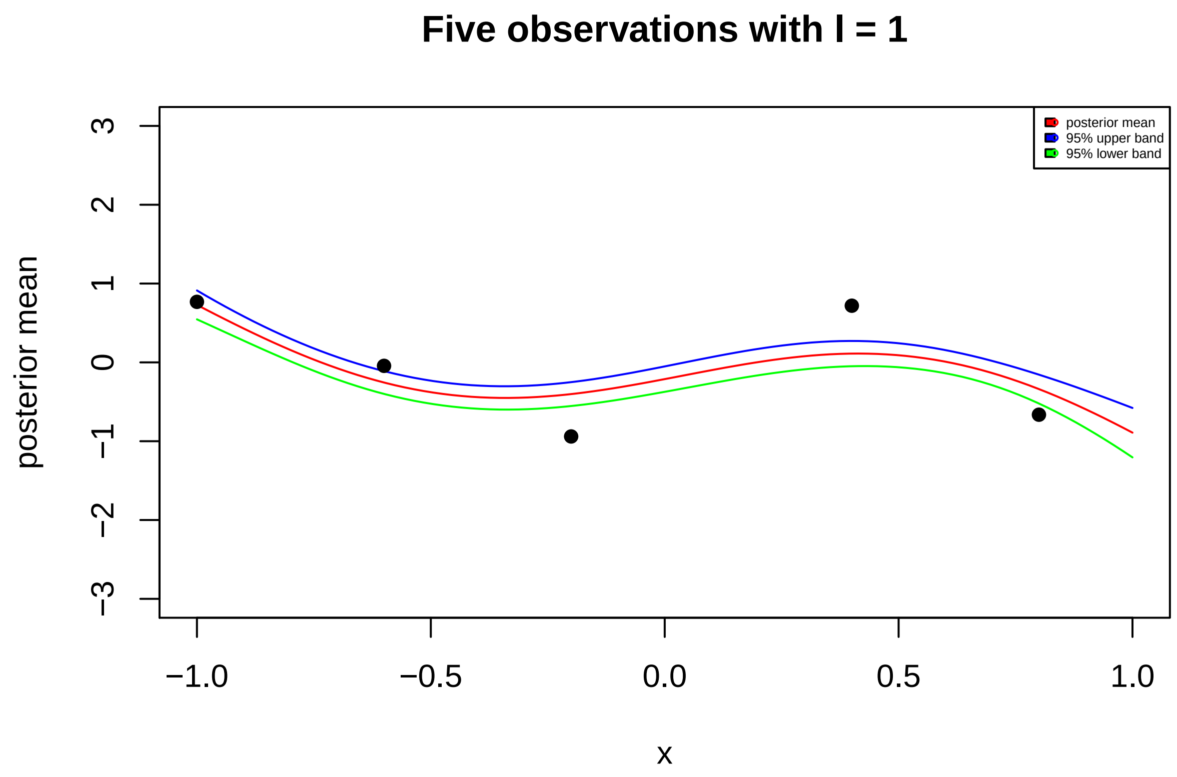
(3)


Two observations with l = 0.3
posterior mean
95% upper band
95% lower band
posterior mean
x
−1.0
−0.5
0.0
0.5
1.0
−3
−2
−1
0
1
2
3

(4)


Five observations with l = 0.3
posterior mean
95% upper band
95% lower band
posterior mean
x
−3
−2
−1
0
1
2
3
−1.0
−0.5
0.0
0.5
1.0

(5)


Five observations with l = 1
posterior mean
95% upper band
95% lower band
posterior mean
x
−3
−2
−1
0
1
2
3
−1.0
−0.5
0.0
0.5
1.0

Comment

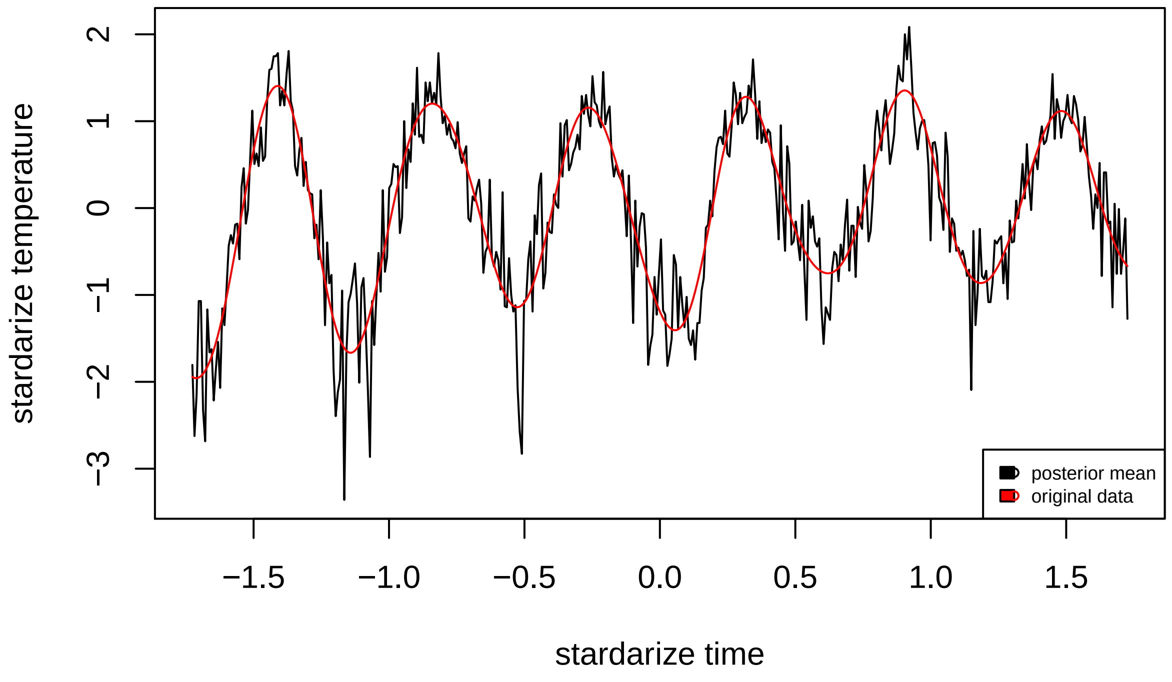
The l is a smoothing factor. The larger l value can make the lines smoother. Hence, the lines in $l = 1$ graph is smoother than $l = 0.3$.

2. GP Regression with kernlab

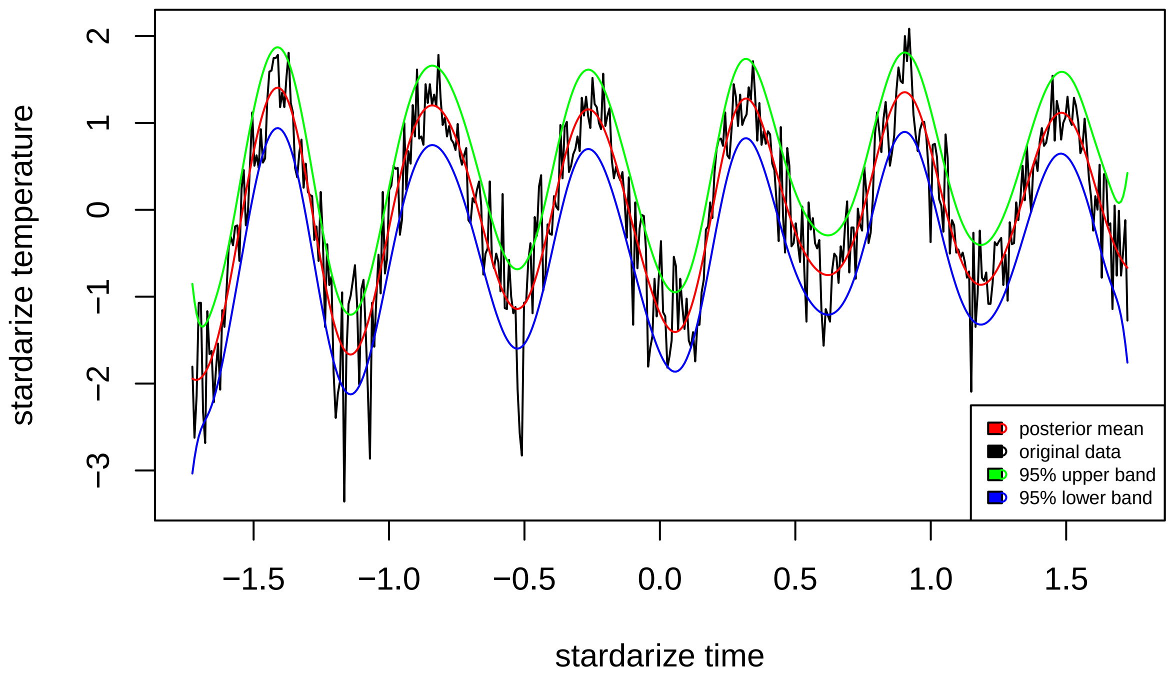
(1)

```
## [1] "The evaluation of the point 1 and 2 is :"

##           [,1]
## [1,] 0.6065307

## [1] "The covariance matrix K(X,XStar) 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
```

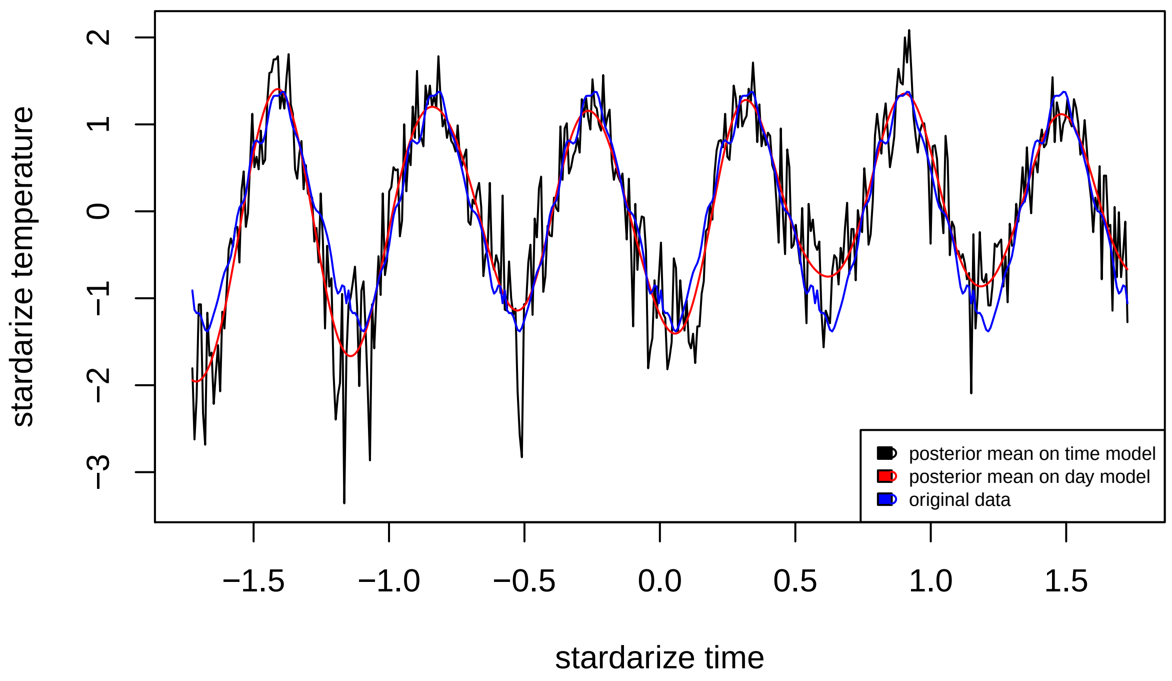
(2)



(3)



(4)

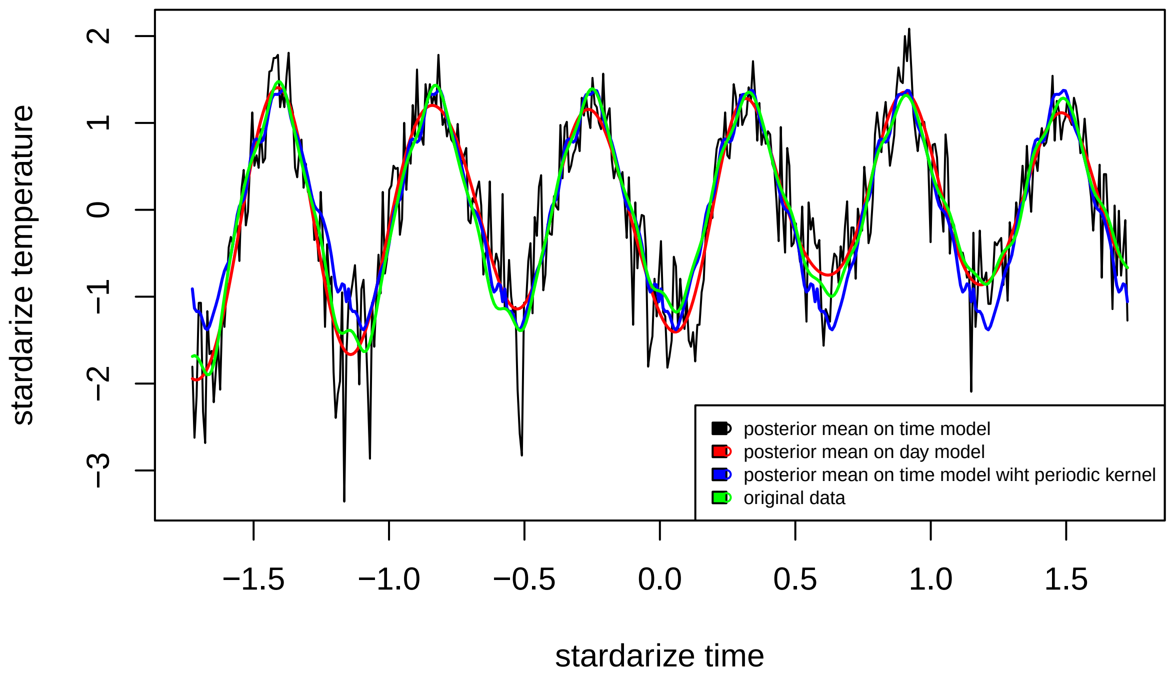


Comment

The "time" model is smoother than "day" model. But it can not fit the peaks and trough better than "day" model.

The "day" model can fit the peaks and trough better. However, it is overfitting on the training data.

(5)



Comment

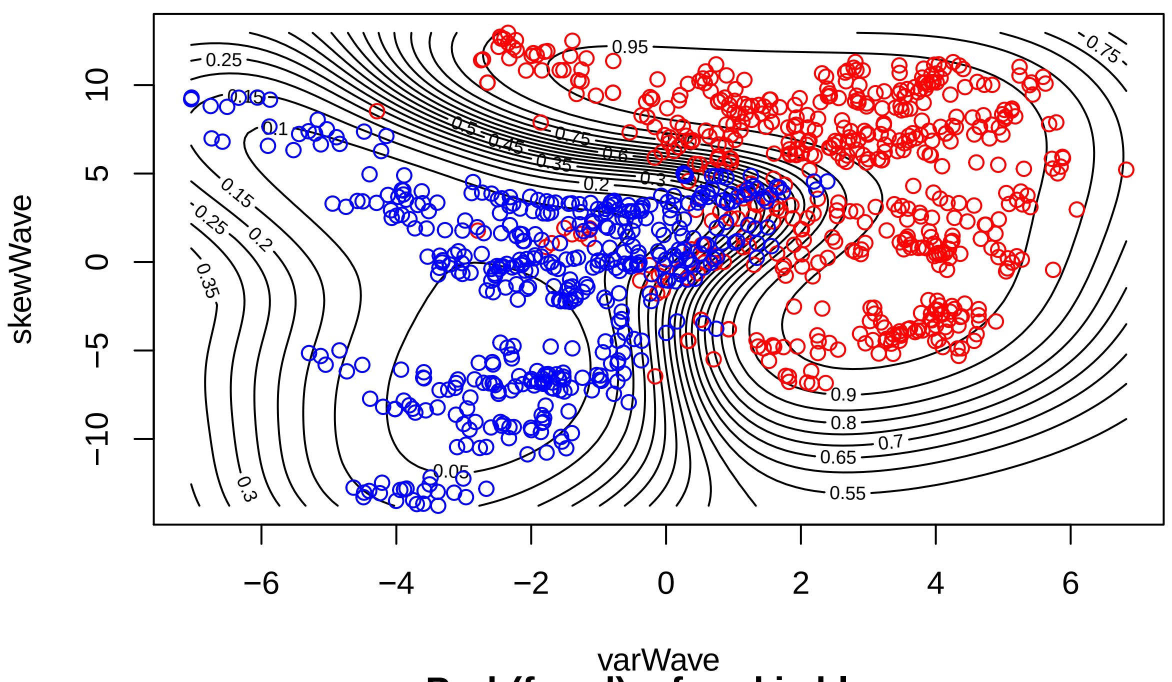
The "time" model with periodic kernel is better than previous two models. It is smooth and fit the training data well.

3. GP Classification with kernlab

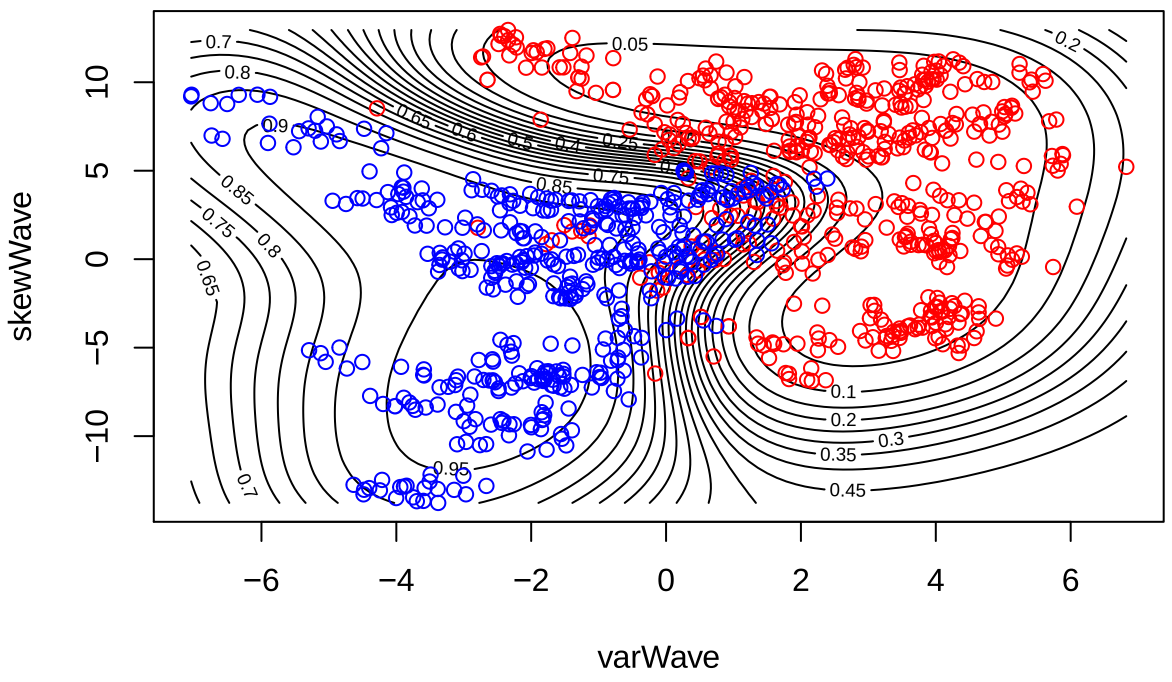
(1)

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

## [1] "The confusion matrix is :"

##
## my_pred   0   1
##       0 503  18
##       1  41 438

## [1] "The accuracy is :"

## [1] 0.941
```

Prob(not fraud) − not fraud is red



Prob(fraud) − fraud is blue



(2)

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

##
## my_pred1   0   1
##        0 199   9
##        1  19 145
```

```
## [1] "The accuracy is :"
```

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

(3)

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

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

```
##
## my_pred2   0   1
##        0 216   0
##        1   2 154
```

```
## [1] "The accuracy is :"
```

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

Comment

The model with all four covariates has higher accuracy than the model with only two covariates.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(mvtnorm)
library(kernlab)
library(AtmRay)
library(magrittr)
# set squared exponential kernel function
squared_exp_Kernel <- function(x1, x2, sigmaF, l){
  # 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 )
  # }
  K <- outer(x1, x2, function(x, y) sigmaF ^ 2 * exp(-0.5 * ( (x - y) / l) ^ 2 ))
  return(K)
}

# set posterior Gaussian Process function
posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...){
  # number of training data
  n <- length(X)
  # Algorithm 2.1 in Rasmussen and Williams
  kstar <- k(X, XStar, ...)
  L <- (k(X, X, ...) + sigmaNoise ^ 2 * diag(n)) %>% chol() %>% t()
  alpha <- solve(t(L), solve(L, y))
  fstarmean <- t(kstar) %*% alpha
  v <- solve(L, kstar)
  fstarvar <- k(XStar, XStar, ...) - t(v) %*% v
  result <- list("fmean" = fstarmean, "fvar" = fstarvar)
  return(result)
```

```r
}
# set parameters
varF <- 1
ll <- 0.3
X1 <- c(0.4)
y1 <- c(0.719)
sigmaNoise <- 0.1
XStar <- seq(-1, 1, length = 100)
k = squared_exp_Kernel
# run Gaussian Process function
my_GP_1 <- posteriorGP(X1, y1, XStar, sigmaNoise, k, varF, ll)
# set plot function
my_plot <- function(x, y, z, z1, ll){
  number <- c("One", "Two", "Three", "Four", "Five")
  index <- length(z)
  plot(x, y$fmean, col = "red", type = "l",
     ylab = "posterior mean", main = paste(number[index], "observations with l =", ll), ylim = c(-3, 3)
  lines(x, y$fmean + 1.96 * sqrt(diag(y$fvar)), col = "blue")
  lines(x, y$fmean - 1.96 * sqrt(diag(y$fvar)), col = "green")
  legend("topright", pch = 1, cex = 0.4,
         legend = c("posterior mean", "95% upper band", "95% lower band"),
         col = c("red", "blue", "green"), fill = c("red", "blue", "green"))
  points(z, z1, pch = 16, col = "black", lwd = 1)
}
my_plot(XStar, my_GP_1, X1, y1,ll)
X2 <- c(0.4, -0.6)
y2 <- c(0.719, -0.044)
my_GP_2 <- posteriorGP(X2, y2, XStar, sigmaNoise, k, varF, ll)
my_plot(XStar, my_GP_2, X2, y2, ll)
X3 <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y3 <- c(0.768, -0.044, -0.940, 0.719, -0.664)
my_GP_3 <- posteriorGP(X3, y3, XStar, sigmaNoise, k, varF, ll)
my_plot(XStar, my_GP_3, X3, y3, ll)
varF <- 1
ll <- 1
my_GP_4 <- posteriorGP(X3, y3, XStar, sigmaNoise, k, varF, ll)
my_plot(XStar, my_GP_4, X3, y3, ll)
# read data
my_data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTulling
# set data
n <- dim(my_data)[1]
index <- seq(1, n, 5)
time <- seq_len(n)
day <- rep(seq_len(365), 6)
new_temp <- my_data$temp[index]
new_time <- time[index]
new_day <- day[index]
# set my squared exponential kernel function
SEkernel <- function(sigmaF = 1, ell = 1){
  rval <- function(x1, x2) {
    K <- outer(x1, x2, function(x, y) sigmaF ^ 2 * exp(-0.5 * ( (x - y) / ell) ^ 2 ))
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}
# define squared exponential kernel function
sigmaF <- 1
ell <- 1
my_SEkernel <- SEkernel(sigmaF, ell)
# evaluate the point x =1, x` = 2
print("The evaluation of the point 1 and 2 is :")
my_SEkernel(1, 2)
cat("\n")
print("The covariance matrix K(X,XStar) is:")
kernelMatrix(my_SEkernel, c(1, 3, 4), c(2, 3, 4))
sigmaF <- 20
ell <- 0.2
my_SEkernel_1 <- SEkernel(sigmaF, ell)
c_temp <- (new_temp - mean(new_temp)) / sd(new_temp) # Standarize the temp.
c_time<-(new_time - mean(new_time)) / sd(new_time) # Standarize the time.
# Estimating the noise variance from a third degree polynomial fit. I() is needed because, otherwise
# time^2 reduces to age in the formula, i.e. time^2 means adding the main effect and the second order
# interaction, which in this case do not exist.
polyFit <- lm(c_temp ~  c_time + I(c_time ^ 2) + I(c_time ^ 3))
sigmaNoise1 = sd(polyFit$residuals)

# training the model and make prediction on the training data
my_GP1 <- gausspr(x = c_time, y = c_temp, type = "regression",
                     kernel = my_SEkernel_1, var = sigmaNoise1) %>% predict(c_time)
# plot the graph
plot(x = c_time, y = c_temp, col = "black", type = "l",
     xlab = "stardarize time", ylab = "stardarize temperature")
lines(x = c_time, my_GP1, col = "red")
legend("bottomright", pch = 1, cex = 0.6,
         legend = c("posterior mean", "original data"),
         col = c("black", "red"), fill = c("black", "red"))
# calculate the variance of the posterior
my_GP2 <- posteriorGP(X = c_time, y = c_temp, XStar = c_time, sigmaNoise = sigmaNoise1, k = my_SEkernel
# plot the graph
plot(x = c_time, y = c_temp, col = "black", type = "l",
     xlab = "stardarize time", ylab = "stardarize temperature")
lines(x = c_time, y = my_GP1, col = "red")
lines(x = c_time, y = my_GP1 + 1.96 * sqrt(diag(my_GP2$fvar)), col = "green")
lines(x = c_time, y = my_GP1 - 1.96 * sqrt(diag(my_GP2$fvar)), col = "blue")
legend("bottomright", pch = 1, cex = 0.6,
         legend = c("posterior mean", "original data", "95% upper band", "95% lower band"),
       col = c("red", "black", "green", "blue"),
       fill = c("red", "black", "green", "blue"))
# Standarize the day
c_day <- (new_day - mean(new_day)) / sd(new_day)
# estimate the noise variance
polyFit2 <- lm(c_temp ~  c_day + I(c_day ^ 2) + I(c_day ^ 3))
sigmaNoise2 = sd(polyFit2$residuals)
# training on the new model
my_GP3 <- gausspr(x = c_day, y = c_temp, type = "regression",
                     kernel = my_SEkernel_1, var = sigmaNoise2) %>% predict(c_day)
# plot the graph
plot(x = c_time, y = c_temp, col = "black", type = "l",
     xlab = "stardarize time", ylab = "stardarize temperature")
lines(x = c_time, my_GP1, col = "red")
lines(x = c_time, my_GP3, col = "blue")
legend("bottomright", pch = 1, cex = 0.6,
         legend = c("posterior mean on time model", "posterior mean on day model", "original data"),
         col = c("black", "red", "blue"), fill = c("black", "red", "blue"))
# set the locally periodic kernel function
SEkernel1 <- function(sigmaF = 1, el1 = 1, el2 = 2, d = 365){
  rval1 <- function(x1, x2) {
    K <- outer(x1, x2, function(x, y) sigmaF ^ 2 * exp(-2 * sin(pi * abs(x - y) / d) ^ 2 / el1 ^ 2) * e
    return(K)
  }
  class(rval1) <- "kernel"
  return(rval1)
}
# set initial parameters
sigmaF <- 20
el1 <- 1
el2 <- 10
d <- 365 / sd(new_time)
my_SEkernel_2 <- SEkernel1(sigmaF, el1, el2, d)
# training the time model on new kernel function
my_GP4 <- gausspr(x = c_time, y = c_temp, type = "regression",
                     kernel = my_SEkernel_2, var = sigmaNoise1) %>% predict(c_time)
# plot the graph
plot(x = c_time, y = c_temp, col = "black", type = "l",
     xlab = "stardarize time", ylab = "stardarize temperature")
lines(x = c_time, my_GP1, col = "red", lwd = 1.5)
lines(x = c_time, my_GP3, col = "blue", lwd = 1.5)
lines(x = c_time, my_GP4, col = "green", lwd = 1.5)
legend("bottomright", pch = 1, cex = 0.6,
         legend = c("posterior mean on time model", "posterior mean on day model", "posterior mean on t
         col = c("black", "red", "blue", "green"), fill = c("black", "red", "blue", "green"))
# set the dataset
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train_data <- data[SelectTraining, ]
test_data <- data[-SelectTraining, ]
#training on the model and take prediction
my_GP_fraud <- gausspr(fraud ~  varWave + skewWave, data = train_data)
my_pred <- predict(my_GP_fraud, train_data[, 1:2])
# confusion matrix
my_table <- table(my_pred, train_data[, 5])
print("The confusion matrix is :")
my_table
cat("\n")
# accuracy
```

```r
print("The accuracy is :")
sum(diag(my_table)) / sum(my_table)

#plot contours of the prediction probabilities
# class probabilities
x1 <- seq(min(train_data[, 1]), max(train_data[, 1]), length = 100)
x2 <- seq(min(train_data[, 2]), max(train_data[, 2]), 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(my_GP_fraud, gridPoints, type = "probabilities")

# Plotting for Prob(not fraud)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main = 'I
points(train_data[train_data[,5]=='0',1],train_data[train_data[,5]=='0',2],col="red")
points(train_data[train_data[,5]=='1',1],train_data[train_data[,5]=='1',2],col="blue")


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

my_pred1 <- predict(my_GP_fraud, test_data[, 1:2])
# confusion matrix
my_table1 <- table(my_pred1, test_data[, 5])
print("The confusion matrix is :")
my_table1
cat("\n")
# accuracy
print("The accuracy is :")
sum(diag(my_table1)) / sum(my_table1)
my_pred2 <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data = train_data) %>% predic
# confusion matrix
my_table2 <- table(my_pred2, test_data[, 5])
print("The confusion matrix is :")
my_table2
cat("\n")
# accuracy
print("The accuracy is :")
sum(diag(my_table2)) / sum(my_table2)
```