Advanced Machine Learning Lab 4 Individual Report

Akshath Srinivas(akssr921)

2023-10-06

Question 2.1

(1)

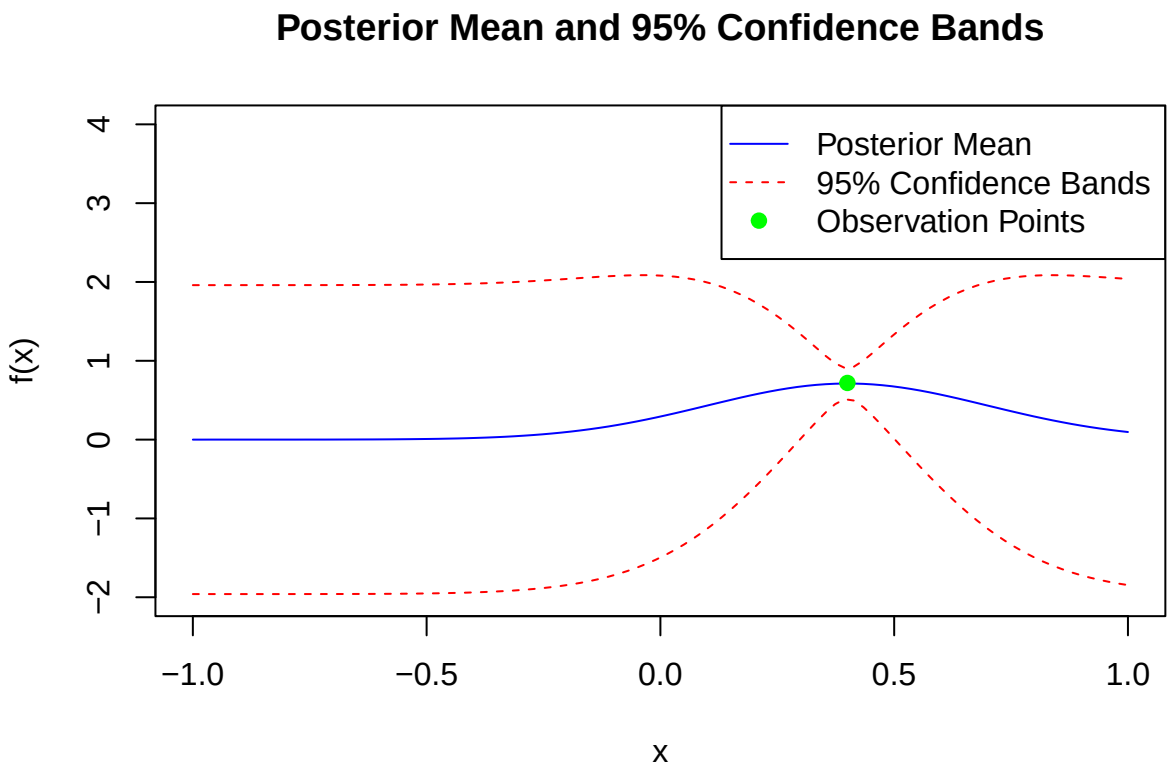
```
kernel_func <- function(x1,x2,sigma_f,l){
  #storing kernel values in matrix for all x points
  kernel_values <- matrix(NA,length(x1),length(x2))
  #calculating kerenel values in loop
  for(i in 1:length(x2)){
    kernel_values[,i] <- sigma_f^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(kernel_values)
}

gp_posterior <- function(X,y,Xstar,sigma_n,k_fn,...){
  k=k_fn(X,X,...)
  k_star=k_fn(X,Xstar,...)
  #cholesky
  L=t(chol(k+(sigma_n^2)*diag(length(X))))
  #alpha
  alp=solve(t(L),solve(L,y))
  #mean
  f_star = t(k_star) %*% alp
  v = solve(L,k_star)
  #variance
  v_fstar=k_fn(Xstar,Xstar,...)-t(v)%*%v
  return(list('mean'=f_star,'variance'=v_fstar))
}

plotPosterior <- function(X, y, XStar, sigma_n, k_fn, sigma_f, l) {
  # Call the posteriorGP function to get the posterior mean and variance
  posterior <- gp_posterior(X, y, XStar, sigma_n, k_fn,sigma_f, l)

  # Extract the posterior mean and variance
  fStar <- posterior$mean
  vStar <- posterior$variance

  # Calculate 95% confidence intervals
  lower_bound <- fStar - 1.96 * sqrt(diag(vStar))
  upper_bound <- fStar + 1.96 * sqrt(diag(vStar))
```

```
  # Plot the posterior mean
  plot(XStar, fStar, type = "l", col = "blue", xlab = "x",
       ylab = "f(x)", main = "Posterior Mean and 95% Confidence Bands",ylim=c(-2,4))

  # Plot the 95% confidence bands
  lines(XStar, lower_bound, col = "red", lty = 2)
  lines(XStar, upper_bound, col = "red", lty = 2)

  # Add the training points to the plot
  points(X, y, col = "green", pch = 19)

  # Add legend
  legend("topright", legend = c("Posterior Mean", "95% Confidence Bands", "Observation Points"),
         col = c("blue", "red", "green"), lty = c(1, 2, NA), pch = c(NA, NA, 19))
}
```

(2)

For the parameters $\sigma_f = 1$, $l = 0.3$ and $(x, y) = (0.4, 0.719)$, Plot of the posterior mean of f over the interval x from -1 to 1 and Plot for 95 % probability (pointwise) bands for f can be seen below.

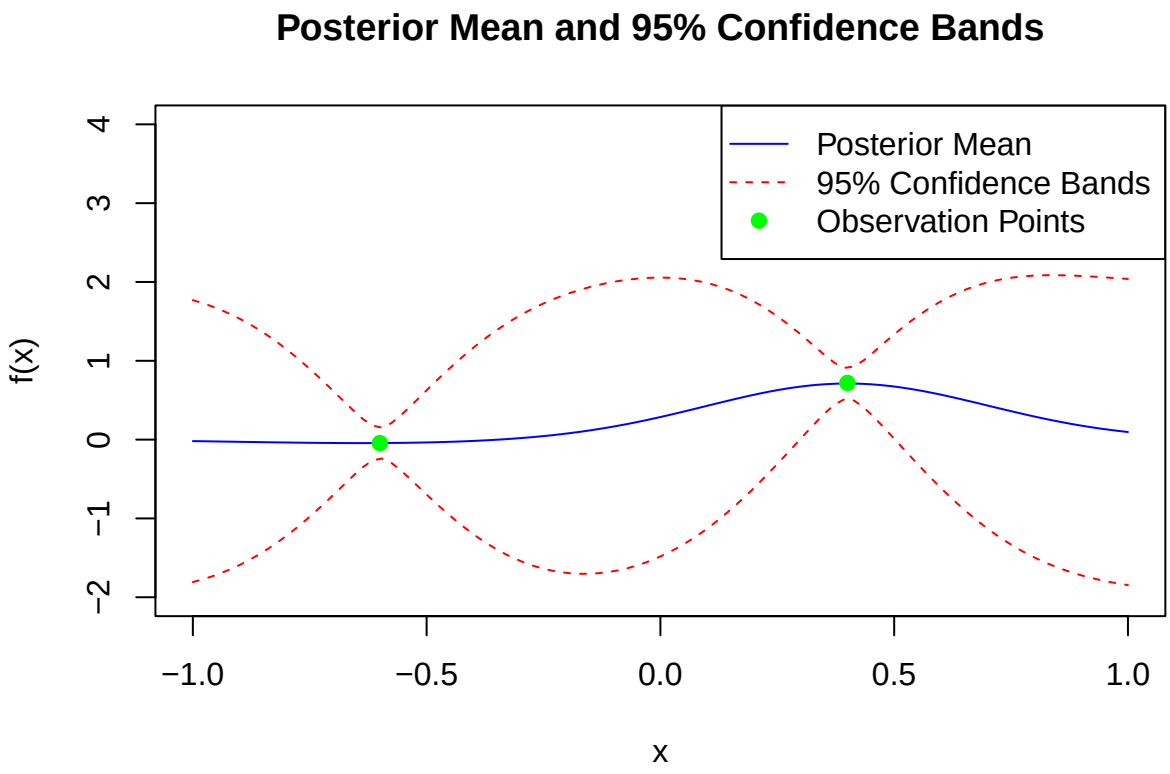
```
#question 2.1 (2)
# observation points
X <- c(0.4)
y <- c(0.719)

#inputs where the posterior distribution is evaluated (XStar)
XStar <- seq(-1, 1, length.out = 100)

# Set parameters
sigma_n <- 0.1
sigma_f <- 1
l <- 0.3

res <- gp_posterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)

# Call the plotPosterior function to plot the posterior mean and 95% probability bands
plotPosterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)
```



(3)

For the parameters $\sigma_f = 1$, $l = 0.3$ and $(x, y) = (0.4, 0.719) and (x, y) = (-0.6, -0.044)$, Plot of the posterior mean of f over the interval x from -1 to 1 and Plot for 95 % probability (pointwise) bands for f can be seen below.

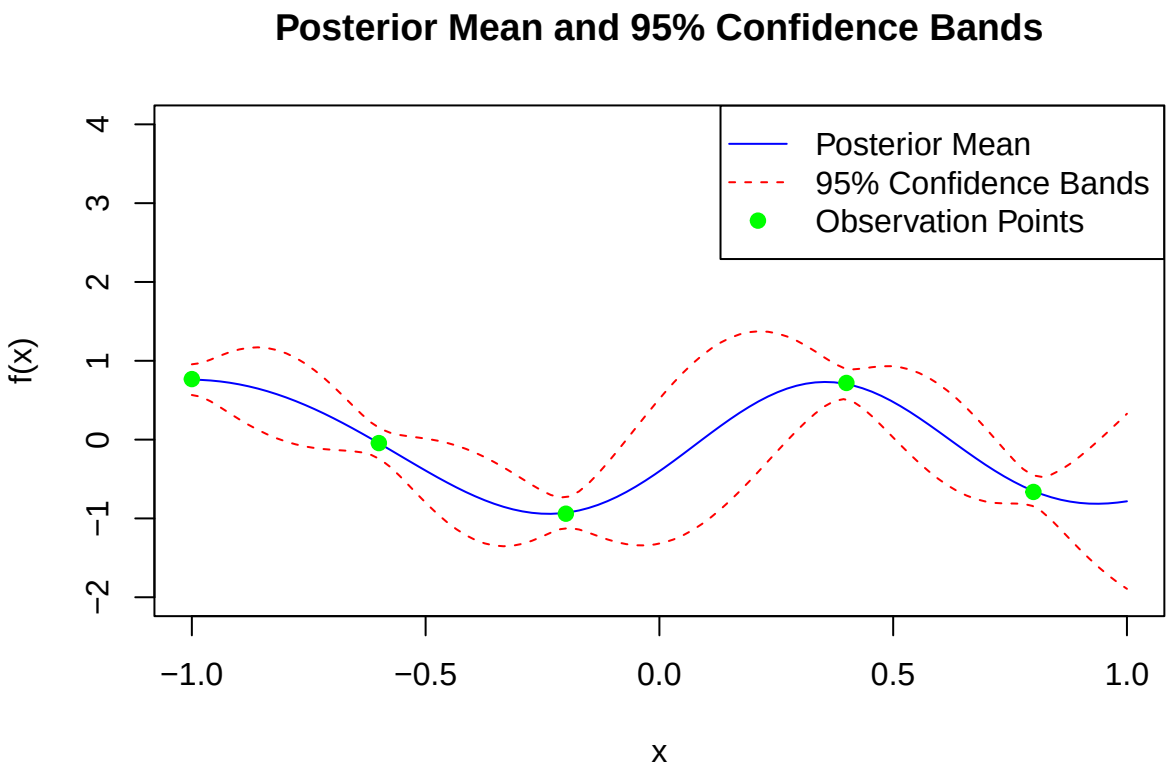
```
#question 2.1 (3)
X <- c(0.4,-0.6)
y <- c(0.719,-0.044)

res <- gp_posterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)

# Call the plotPosterior function to plot the posterior mean and 95% probability bands
plotPosterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)
```



(4)

For the parameters $\sigma_f = 1$, $l = 0.3$ and five points given in question, Plot of the posterior mean of f over the interval x from -1 to 1 and Plot for 95 % probability (pointwise) bands for f can be seen below.

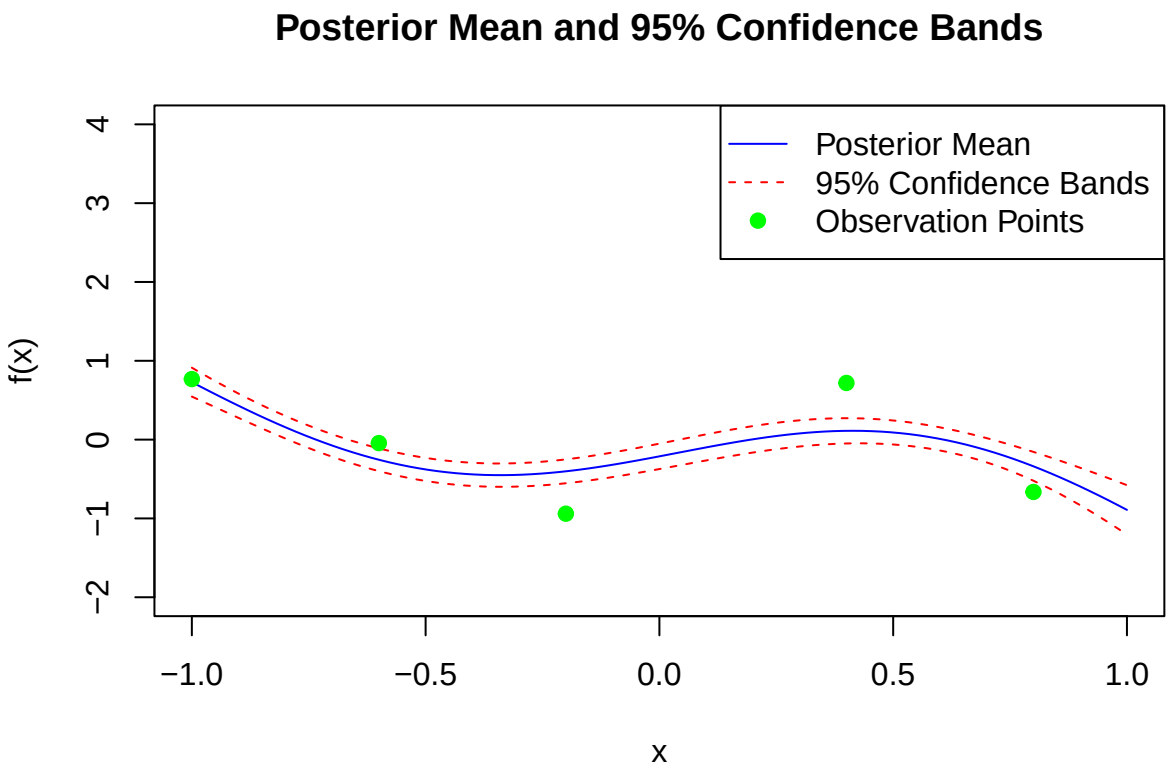
```
#question 2.1 (4)
X <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)

res <- gp_posterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)

# Call the plotPosterior function to plot the posterior mean and 95% probability bands
plotPosterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)
```



(5)

For the parameters $\sigma_f = 1$, $l = 1$ and five points given in question, Plot of the posterior mean of f over the interval x from -1 to 1 and Plot for 95 % probability (pointwise) bands for f can be seen below.

```
#question 2.1 (5)
sigma_f <- 1
l <- 1

res <- gp_posterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)

# Call the plotPosterior function to plot the posterior mean and 95% probability bands
plotPosterior(X, y, XStar, sigma_n, kernel_func, sigma_f, l)
```



The 'l' hyperparameter controls the complexity of the model. A smaller l value indicates more complicated model wherein the posterior mean function is more sensitive to small scale variations. That's why in the previous subquestion, we observed that the posterior mean function is oscillatory and more complex. This may lead to overfitting. Whereas when we increase l, the posterior mean function is no longer sensitive to small scale variations, rather correlation between nearby points decreases more slowly with distance, emphasizing longer-range dependencies, resulting in a smoother function.

Question 2.2

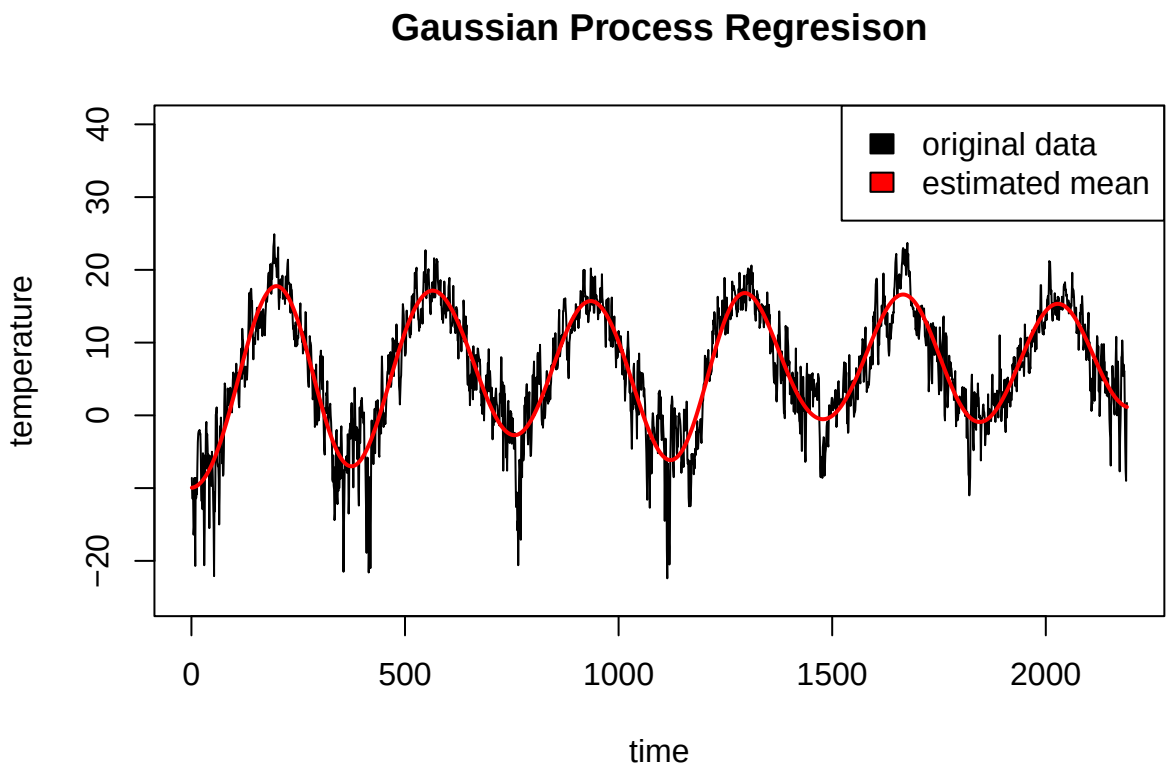
(1)

```
#question 2.2(1)
square_exp_KF <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
    r = outer(x,y,function(x1,x2){x1-x2});
    return(sigmaf^2*exp(-r^2/2/ell^2))
  }
  class(rval) <- "kernel"
  return(rval)
}

X<-1
```

```r
Xstar <- 2

kernelfunc <- square_exp_KF(sigmaf = 1, ell = 1)

kern_mat <- kernelMatrix(kernel = kernelfunc, x = X, y = Xstar)

#preparing data
temp_data <- read.csv("https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/master/GaussianProcess/Co
temp_data <- as.data.frame(temp_data)
temp_data$time <-  seq(1,nrow(temp_data),1)
temp_data$day <- rep(seq(1,365,1),6)
temp_data$time2 <- temp_data$time^2
```

(2)

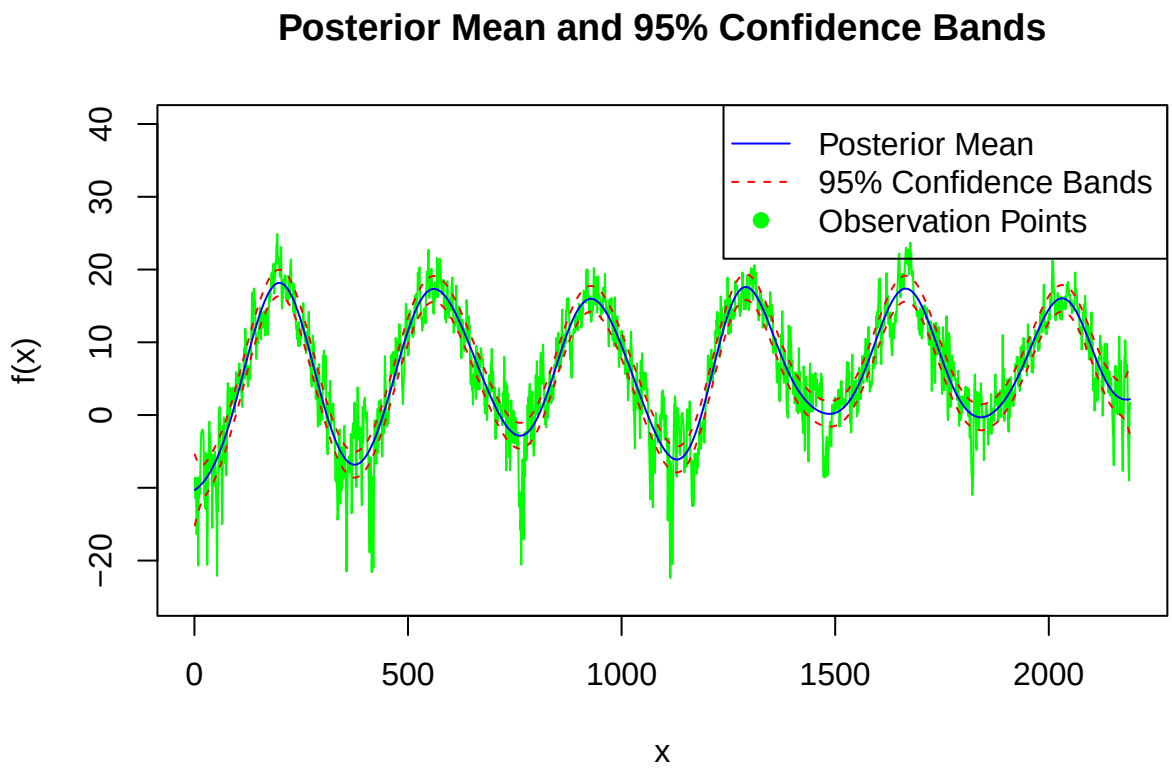
```r
#question 2.2(2)
#lm model
model_lm <- lm(formula = temp ~ time + time2,data = temp_data)
sigmanoise2 <- var(model_lm$residuals)


# set the kernel function
kernel <- square_exp_KF(20,0.2)
# Gaussian Process Regression
gp_model <- gausspr(x = temp_data$time,
                    y = temp_data$temp,
                    var = sigmanoise2,
                    data = temp_data,
                    kernel = kernel)
# Prediction
temp_pred <- predict(gp_model,temp_data[,'time'])
# Plot Figure
plot(temp_data$time, temp_data$temp,type = 'l', ylim = c(-25,40),
     xlab = 'time',ylab = 'temperature',main = 'Gaussian Process Regresison')
lines(temp_data$time,temp_pred,col = 'red', lwd = 2)
legend('topright',
       legend = c('original data','estimated mean'),
       fill = c('black','red'))
```



(3)

```r
#question 2.2(3)
# Scale X and X_star
X_scaled <- scale(temp_data$time)[,1]
y <- scale(temp_data$temp)[,1]
x_star_scaled <- scale(temp_data$time)[,1]
x <- kernel(x = X_scaled,y =x_star_scaled)

scaled_df=data.frame(time=X_scaled,time2=X_scaled^2,temp=y)
lm_model_scaled <- lm(formula = temp ~ time + time2,data = scaled_df)
sigmanoise2_scaled <- var(lm_model_scaled$residuals)

#posteriorGP function
result_223 <- gp_posterior(X_scaled,y,x_star_scaled,sqrt(sigmanoise2_scaled),kernel)

#plot
# Extract the posterior mean and variance
fStar <- (result_223$mean*sd(temp_data$temp))+mean(temp_data$temp)
vStar <- sqrt(diag(result_223$variance))*sd(temp_data$temp)

# Compute 95% CI
# Calculate 95% confidence intervals
lower_bound <- fStar - 1.96 * vStar
```

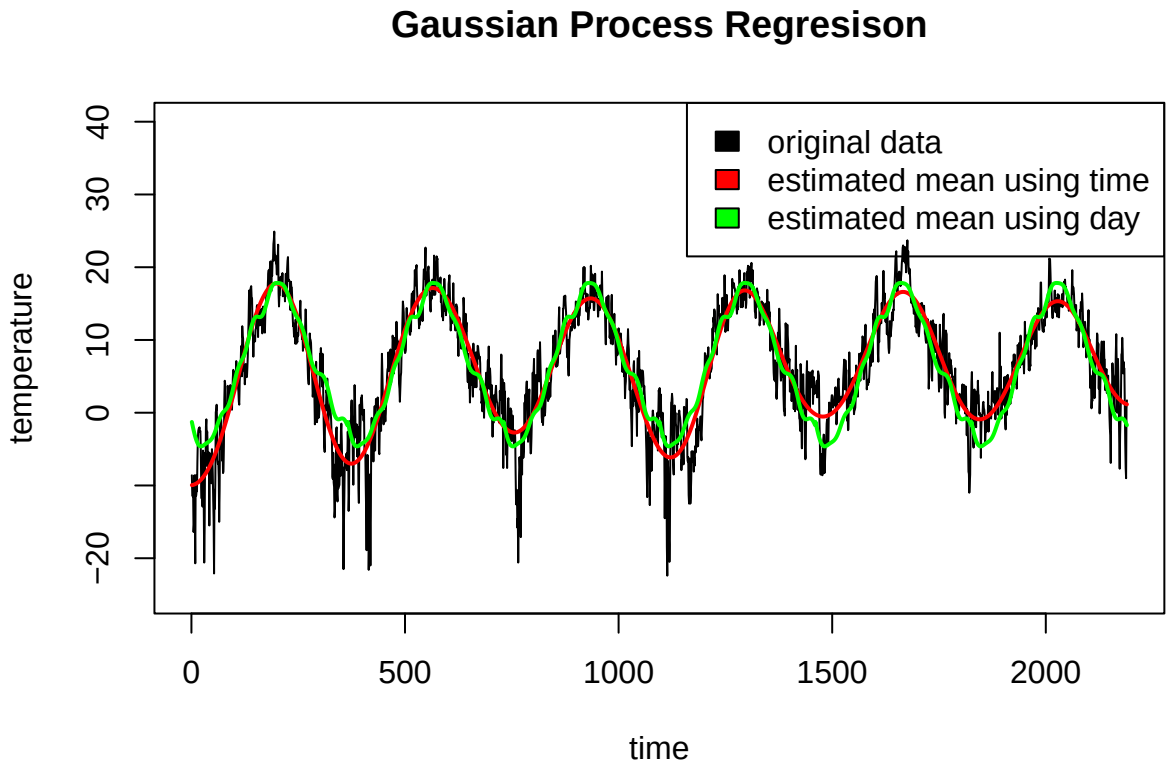
```r
upper_bound <- fStar + 1.96 * vStar

plot(temp_data$time, temp_data$temp, col = "green",type='l', xlab = "x",
     ylab = "f(x)", main = "Posterior Mean and 95% Confidence Bands",ylim=c(-25,40))
# Plot the posterior mean
lines(fStar, col = "blue")

# Plot the 95% confidence bands
lines(lower_bound, col = "red", lty = 2)
lines(upper_bound, col = "red", lty = 2)

# Add legend
legend("topright", legend = c("Posterior Mean", "95% Confidence Bands", "Observation Points"),
       col = c("blue", "red", "green"), lty = c(1, 2, NA), pch = c(NA, NA, 19))
```



(4)

```r
#question 2.2(4)
#lm model
temp_data$day2 <- temp_data$time^2
model_lm_day <- lm(formula = temp ~ day + day2,data = temp_data)
sigmanoise2_d <- var(model_lm_day$residuals)
```

```r
# set the kernel function
kernel <- square_exp_KF(20,0.2)
# Gaussian Process Regression
gp_model_day <- gausspr(x = temp_data$day,
                        y = temp_data$temp,
                        var = sigmanoise2_d,
                        data = temp_data,
                        kernel = kernel)
# Prediction
temp_pred_day <- predict(gp_model_day,temp_data[,'day'])
# Plot Figure
plot(temp_data$temp,type = 'l', ylim = c(-25,40),
     xlab = 'time',ylab = 'temperature',main = 'Gaussian Process Regresison')
lines(temp_pred,col = 'red', lwd = 2)
lines(temp_pred_day,col = 'green', lwd = 2)
legend('topright',
       legend = c('original data','estimated mean using time','estimated mean using day'),
       fill = c('black','red','green'))
```



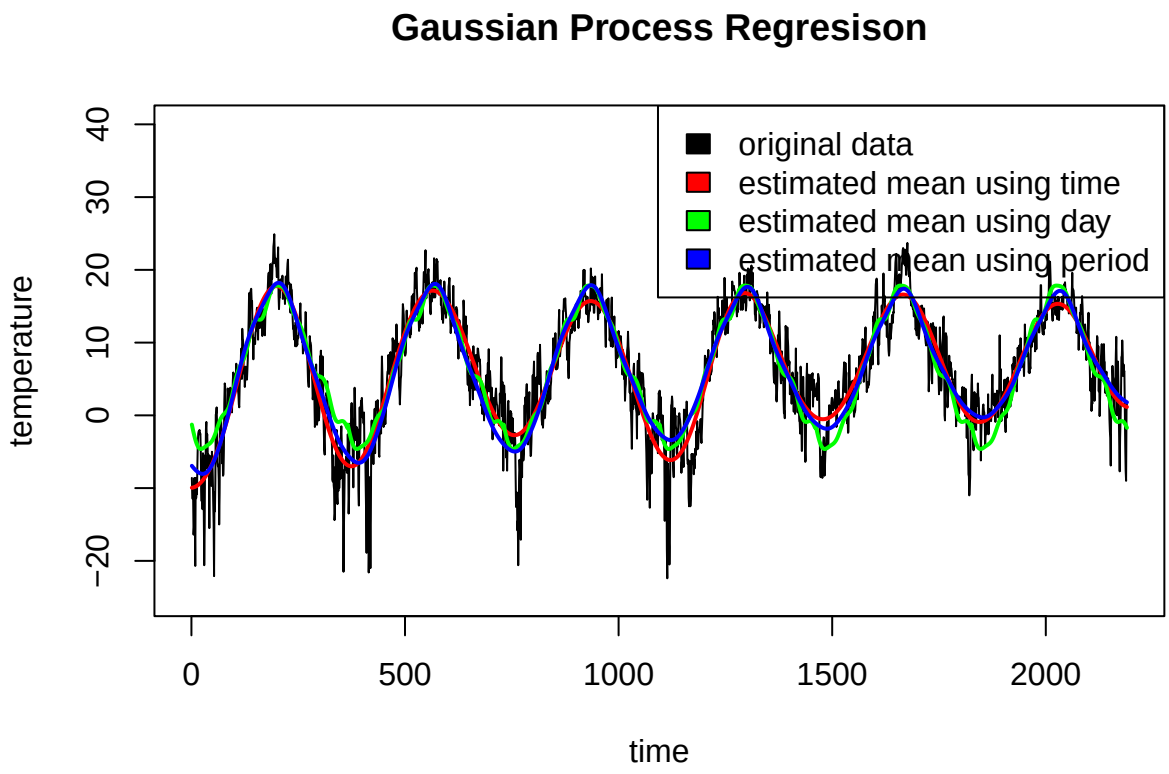
Compare the results of both models. What are the pros and cons of each model?

In time model, we have modelled time as sequence of values starting from 1 to 2190. Because of this, the model is able to capture the trend of the data. Whereas in the day model, we have modelled the day as a sequence of values from 1 to 365 for each year. So, although day 1 of 2010 is far off from day 1 of 2011-15, they are highly correlated due to our data setup. Because of this correlation, the day model is able to capture the periodicity. Pros and cons: As mentioned above, the time model is able to capture the short-term trend

in the data, but is not able to capture the periodicity/seasonality. But in the day model is able to capture the periodicity well, but comparatively it does not capture the short-term trend.

(5)

```
#question 2.2(5)
periodic_kernel <- function(sigmaf = 1, l1 = 1, l2,d)
{
  rval <- function(x, y) {
    r <- outer(x,y,function(x1,x2){abs(x1-x2)})
    return(sigmaf^2*exp(-(2*sin(pi*r/d)^2)/(l1^2))*exp(-1/2*(r^2/l2^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}

#intialize d
d <- 365/sd(temp_data$time)

# set the kernel function
kernel <- periodic_kernel(20,1,10,d)
# Gaussian Process Regression
gp_model_per <- gausspr(x = temp_data$time,
                        y = temp_data$temp,
                        var = sigmanoise2,
                        data = temp_data,
                        kernel = kernel)

# Prediction
temp_pred_per <- predict(gp_model_per,temp_data[,'time'])
# Plot Figure
plot(temp_data$temp,type = 'l', ylim = c(-25,40),
     xlab = 'time',ylab = 'temperature',main = 'Gaussian Process Regresison')
lines(temp_pred,col = 'red', lwd = 2)
lines(temp_pred_day,col = 'green', lwd = 2)
lines(temp_pred_per,col = 'blue', lwd = 2)
legend('topright',
       legend = c('original data','estimated mean using time',
                  'estimated mean using day','estimated mean using period'),
       fill = c('black','red','green','blue'))
```



Compare and discuss the results:

As shown above, the Time-Periodic Posterior mean is able to capture both the trend and the seasonality in the data, because the local periodic kernel is a combination of the squared exponential kernel(which capture the trend), and the periodic kernel(which capture the seasonality.)

Question 2.3

(1)

The plot, confusion matrix for the classifier and its training accuracy varWave and skewWave can be seen below.

```
#gausian classification
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[SelectTraining,]
selectest <-  setdiff(1:nrow(data), SelectTraining)
test <-  data[selectest,]
model <-  gausspr(fraud~varWave + skewWave, train)
```

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

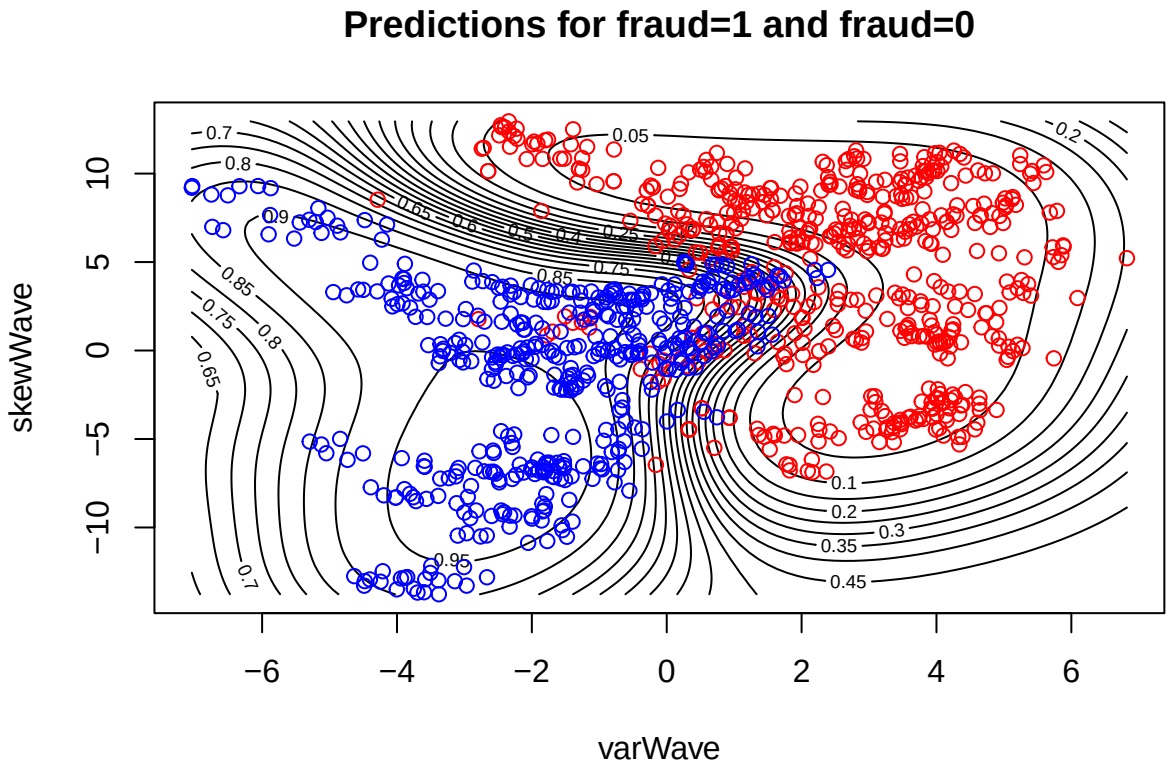
```
fraud_pred <- predict(model, train[,1:2])
tab_train <- table('prediction' = fraud_pred, 'true'=train[,'fraud'])
tab_train
```

```
##           true
## prediction   0   1
##          0 503  18
##          1  41 438
```

```
acc_train <- sum(diag(tab_train))/sum(tab_train)
cat('Training accuracy using varWave and skewWave:', acc_train)
```

```
## Training accuracy using varWave and skewWave: 0.941
```

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



(2)

The confusion matrix for the classifier and its test accuracy using varWave and skewWave can be seen below.

```
#2.3(2) test prediction
pred_test = predict(model, test)
tab_test <- table('prediction' = pred_test, 'true'=test[,'fraud'])
tab_test
```

```
##            true
## prediction   0   1
##          0 199   9
##          1  19 145
```

```
acc_test <- sum(diag(tab_test))/sum(tab_test)
cat('Test accuracy using varWave and skewWave:', acc_test)
```

```
## Test accuracy using varWave and skewWave: 0.9247312
```

(3)

The confusion matrix for the classifier and its test accuracy using all covariates can be seen below.

```
#2.3(3)
model_full <-  gausspr(fraud~varWave + skewWave+kurtWave+entropyWave, train)
```

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

```
test_pred_full <- predict(model_full, test[,1:4])
tab_full <- table('prediction' = test_pred_full, 'true'=test[,'fraud'])
tab_full
```

```
##           true
## prediction   0   1
##          0 216   0
##          1   2 154
```

```
acc_test_full <- sum(diag(tab_full))/sum(tab_full)
cat('Test accuracy using all covariates:', acc_test_full)
```

```
## Test accuracy using all covariates: 0.9946237
```

Compare the accuracy to the model with only two covariates: As seen in the contour plot, there are quite a lot data points which are near the decision boundary when only 2 covariates are considered for the classification. Some of the these points in the decision boundarywere getting missclassified. Whereas, when the covariates are increased to 4, these data points at the decision boundaries are correctly classified, as the 4 covarites are needed to capture the true decision boundary between the classes.