Lab4_IndividualReport

varsi146

2023-10-18

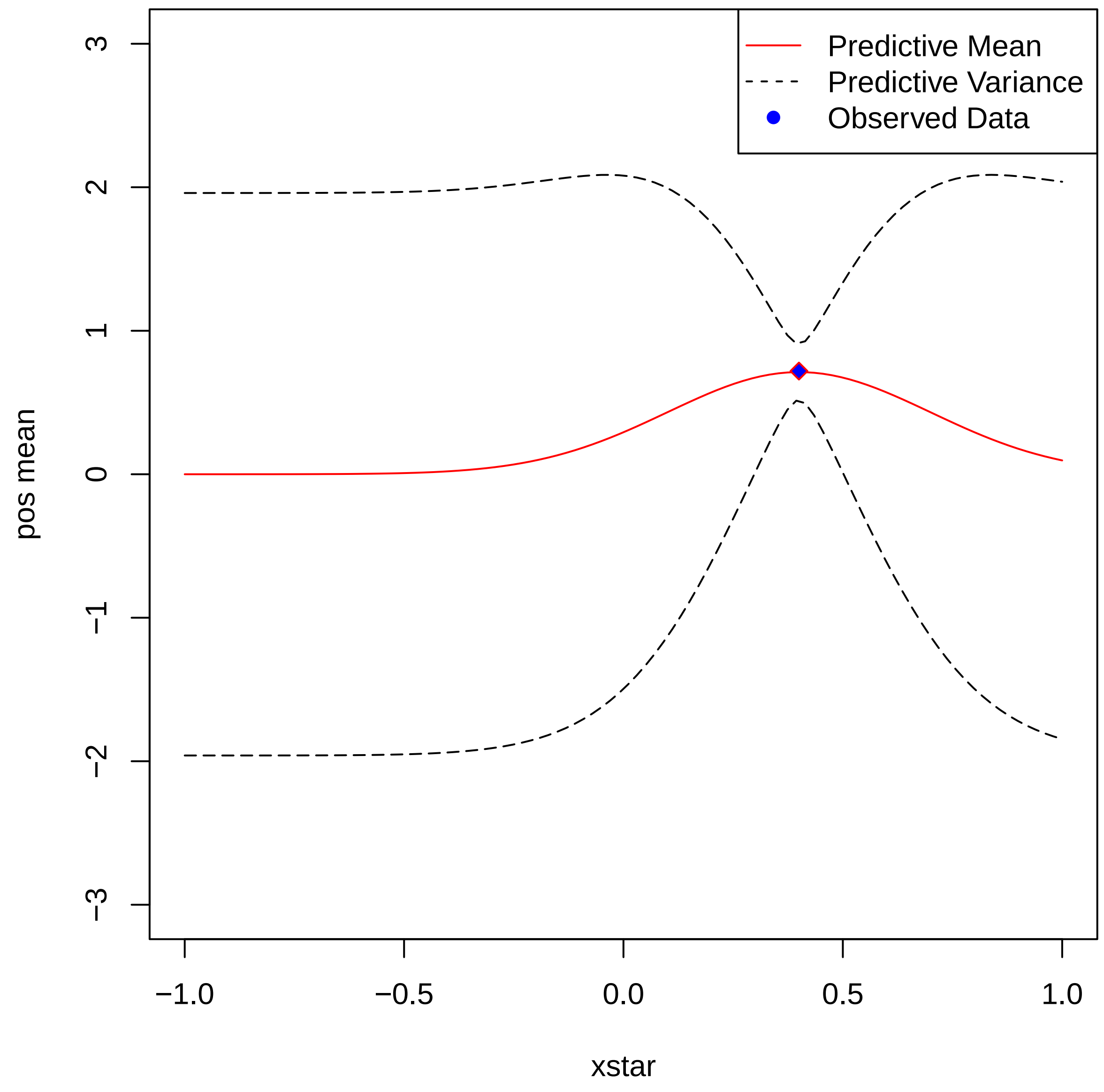
Question 2.1: Implementing GP Regression

Sub Question 1

```r
# Implementing GP Regression
# Algorithm 2.1

# Squared Exponential kernel or otherwise called
# Covariance 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)
}

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))))

  alph <- solve(t(L), solve(L,y))

  # Predictive Mean
  fStar <- t(Kstar) %*% alph

  # Predictive Variance
  v <- solve(L, Kstar)
  var_fStar <- k(XStar, XStar)- t(v) %*% v

  # Return this when required
  # log_marg_ll <- (-0.5*(t(y) %*% alph)) -sum(log(diag(L))) - (0.5*n*log(2*pi))
  return(list('fStar' = fStar, 'var_fStar' = var_fStar))
}
```

Sub Question 2

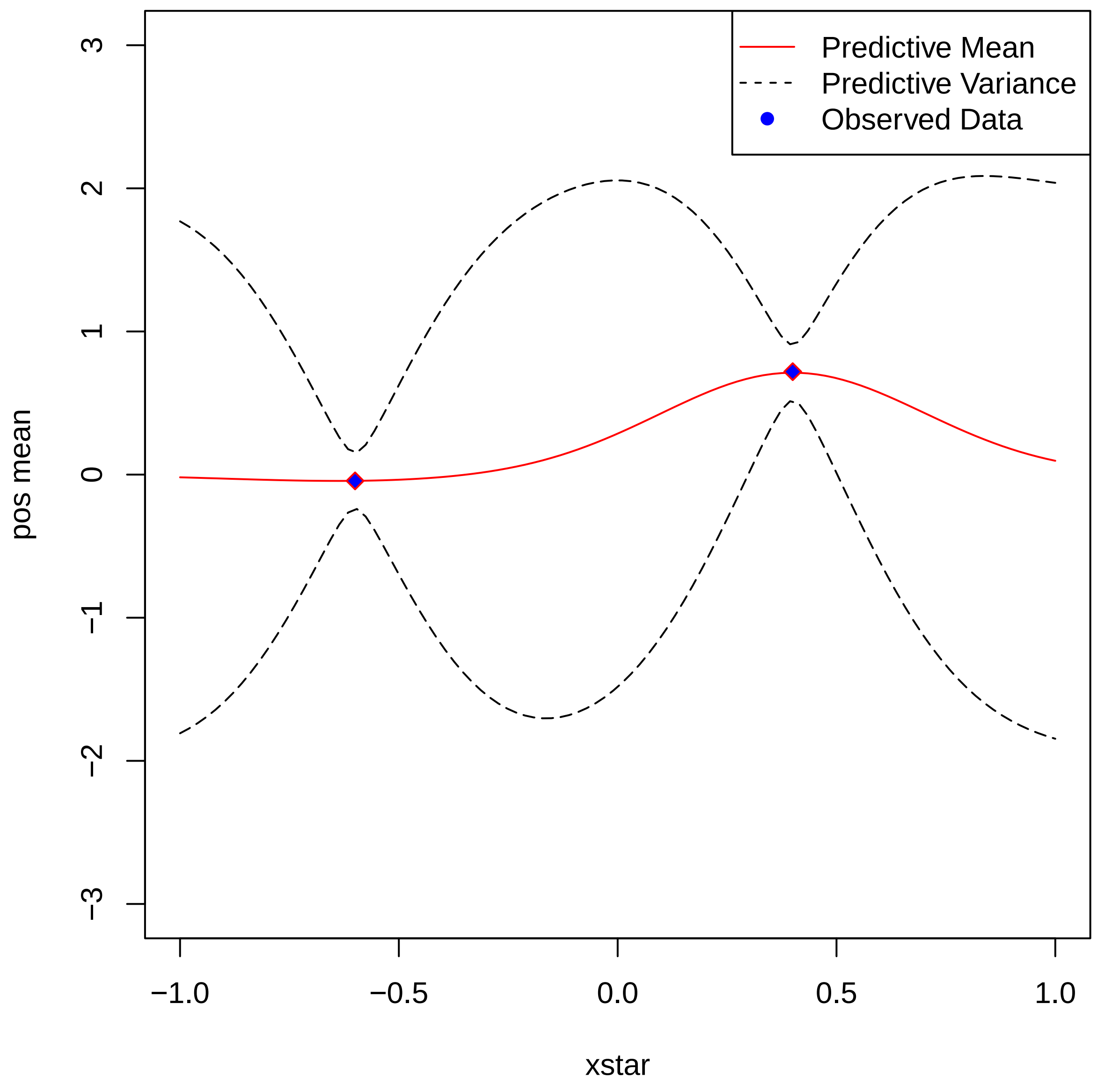
```r
first_data <- c(0.4 ,0.719)
xstar = seq(-1,1,length=100)
sigmaNoise=0.1
k = SquaredExpKernel
post_GP <- posteriorGP(first_data[1], first_data[2], xstar, sigmaNoise, k)

mean_PostPred <- post_GP$fStar
var_PostPred <- post_GP$var_fStar

plot(xstar,mean_PostPred,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred + 1.96*sqrt(diag(var_PostPred)), lty = 5)
lines(xstar, mean_PostPred - 1.96*sqrt(diag(var_PostPred)), lty = 5)
points(first_data[1], first_data[2], pch = 23, col = 'red',  bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
```



Sub Question 3

```
x_2 <- c(0.4, -0.6)
y_2 <- c(0.719, -0.044)
post_GP2 <- posteriorGP(x_2, y_2, xstar, sigmaNoise, k)

mean_PostPred2 <- post_GP2$fStar
var_PostPred2 <- post_GP2$var_fStar
```

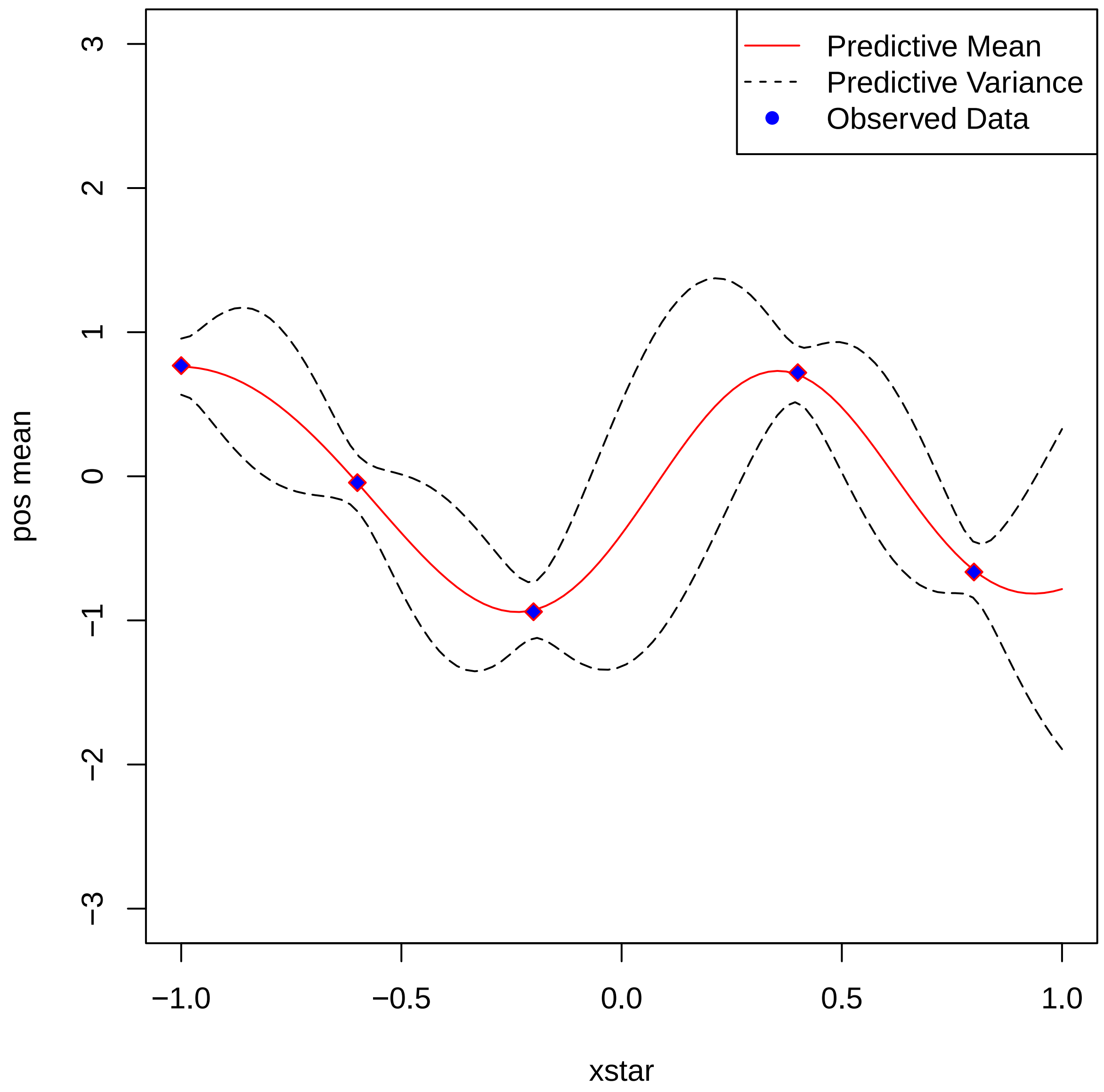
```
plot(xstar,mean_PostPred2,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred2 + 1.96*sqrt(diag(var_PostPred2)), lty = 5)
lines(xstar, mean_PostPred2 - 1.96*sqrt(diag(var_PostPred2)), lty = 5)
points(x_2, y_2, pch = 23, col = 'red', bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
```



Sub Question 4

From the CIs in the below plot, we can observe that the model is more certain at points where we have observed the prior, i.e. (-1.0, -0.6, -0.2, 0.4, 0.8). Hence the variance reduces at these points.

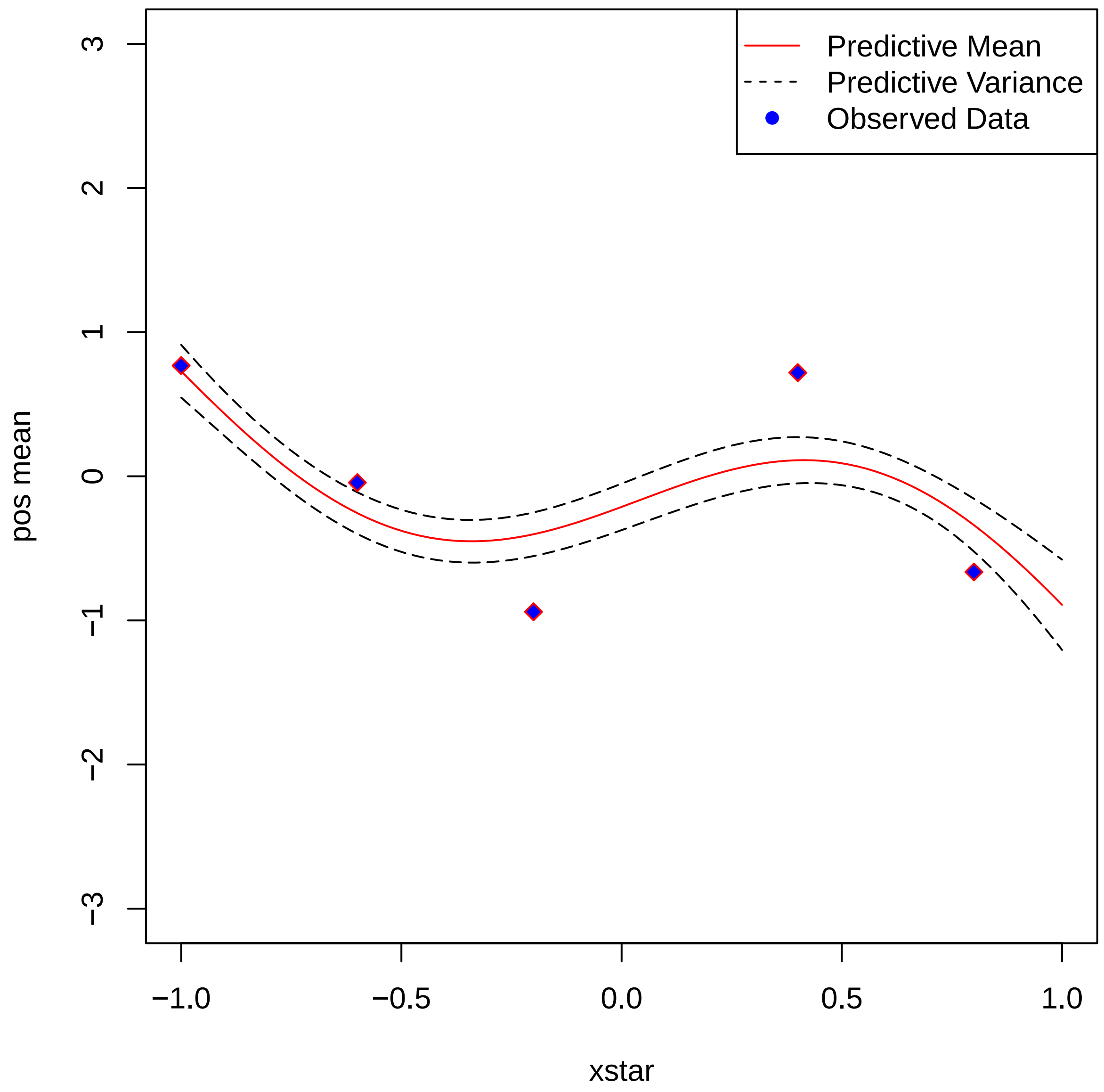
```r
x_3 <- c(-1, -0.6, -0.2, 0.4, 0.8)
y_3 <- c(0.768, -0.044, -0.940, 0.719, -0.664)

post_GP3 <- posteriorGP(x_3, y_3, xstar, sigmaNoise, k)

mean_PostPred3 <- post_GP3$fStar
var_PostPred3 <- post_GP3$var_fStar

plot(xstar,mean_PostPred3,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred3 + 1.96*sqrt(diag(var_PostPred3)), lty = 5)
lines(xstar, mean_PostPred3 - 1.96*sqrt(diag(var_PostPred3)), lty = 5)
points(x_3, y_3, pch = 23, col = 'red', bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
```



Sub Question 5

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.

```r
x_4 <- c(-1, -0.6, -0.2, 0.4, 0.8)
y_4 <- c(0.768, -0.044, -0.940, 0.719, -0.664)

post_GP4 <- posteriorGP(x_4, y_4, xstar, sigmaNoise, k, sigmaF = 1, l = 1)

mean_PostPred4 <- post_GP4$fStar
var_PostPred4 <- post_GP4$var_fStar

plot(xstar,mean_PostPred4,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred4 + 1.96*sqrt(diag(var_PostPred4)), lty = 5)
lines(xstar, mean_PostPred4 - 1.96*sqrt(diag(var_PostPred4)), lty = 5)
points(x_4, y_4, pch = 23, col = 'red', bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
```



Question 2.2: GP Regression with kernlab

Sub Question 1

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.

train_data <- data
train_data$time <- c(1:nrow(data))
train_data$time2 <- train_data$time**2
```

```r
train_data$day <- rep(c(1:365), 6)

SEkernel <- function(l = 1, sigmaF = 1) {
  kernval <- function(x, x_prime = NULL) {
    res <- matrix(0, nrow = length(x), ncol = length(x_prime))
    for (i in 1:length(x)) {
      for (j in 1:length(x_prime)) {
        res[i, j] <- (x[i] - x_prime[j])^2
      }
    }
    res <- (sigmaF^2) * exp(-res / (2 * l^2))
    return(res)
  }
  class(kernval) <- 'kernel'
  return(kernval)
}


SEkernel_func <- SEkernel(1, 1)
# Evaluating the kernel at x=1 and x' = 2
cat("Kernel evaluation at x = 1 and x' = 2:",SEkernel_func(x = 1, x_prime = 2))
```

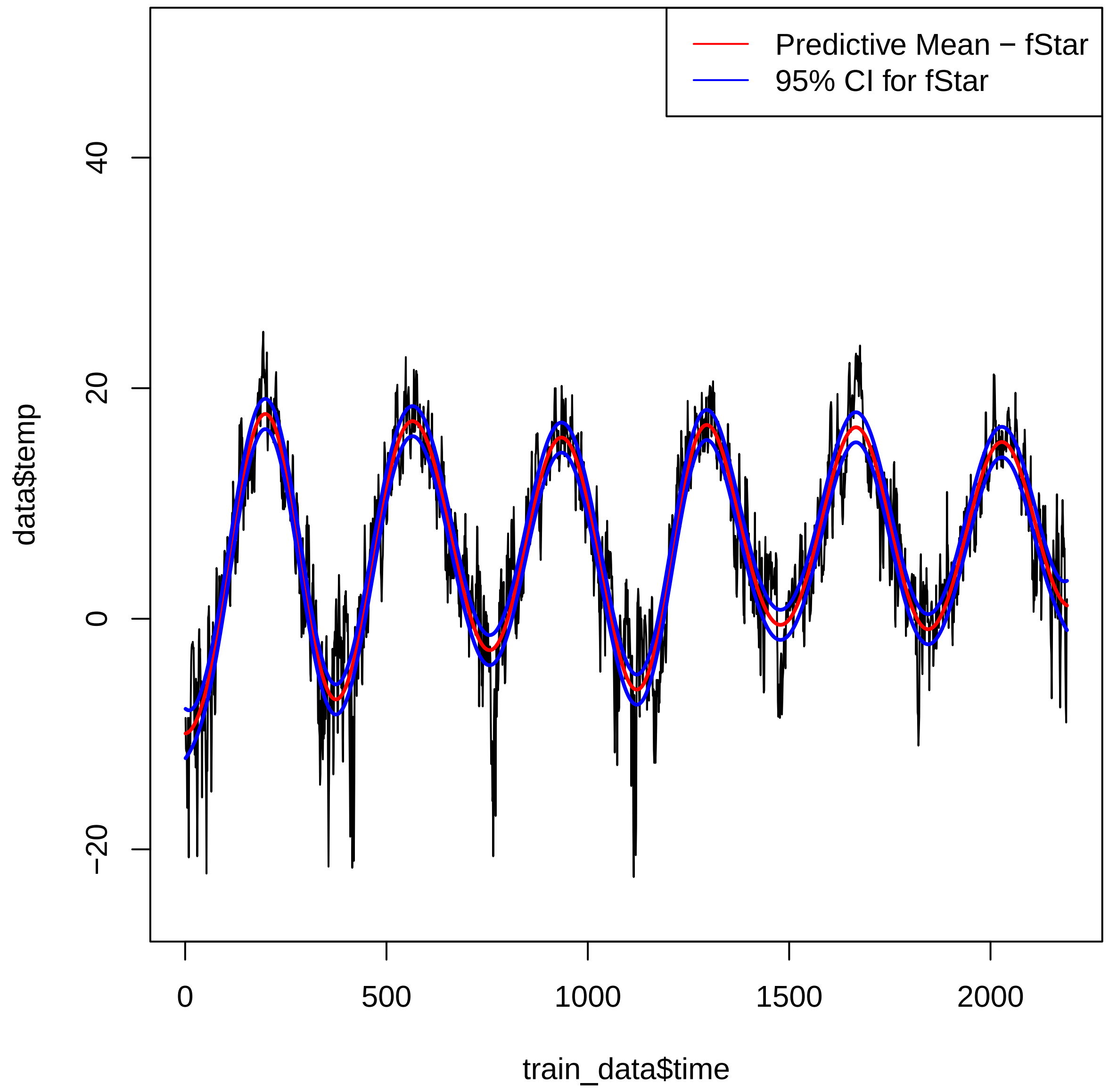
```
## Kernel evaluation at x = 1 and x’ = 2: 0.6065307
```

```r
# Computing covariance matrix K(X, Xstar)
kernelMatrix(kernel = SEkernel_func, x = c(1,3,4), y = c(2, 3, 4))
```

```
## 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
```

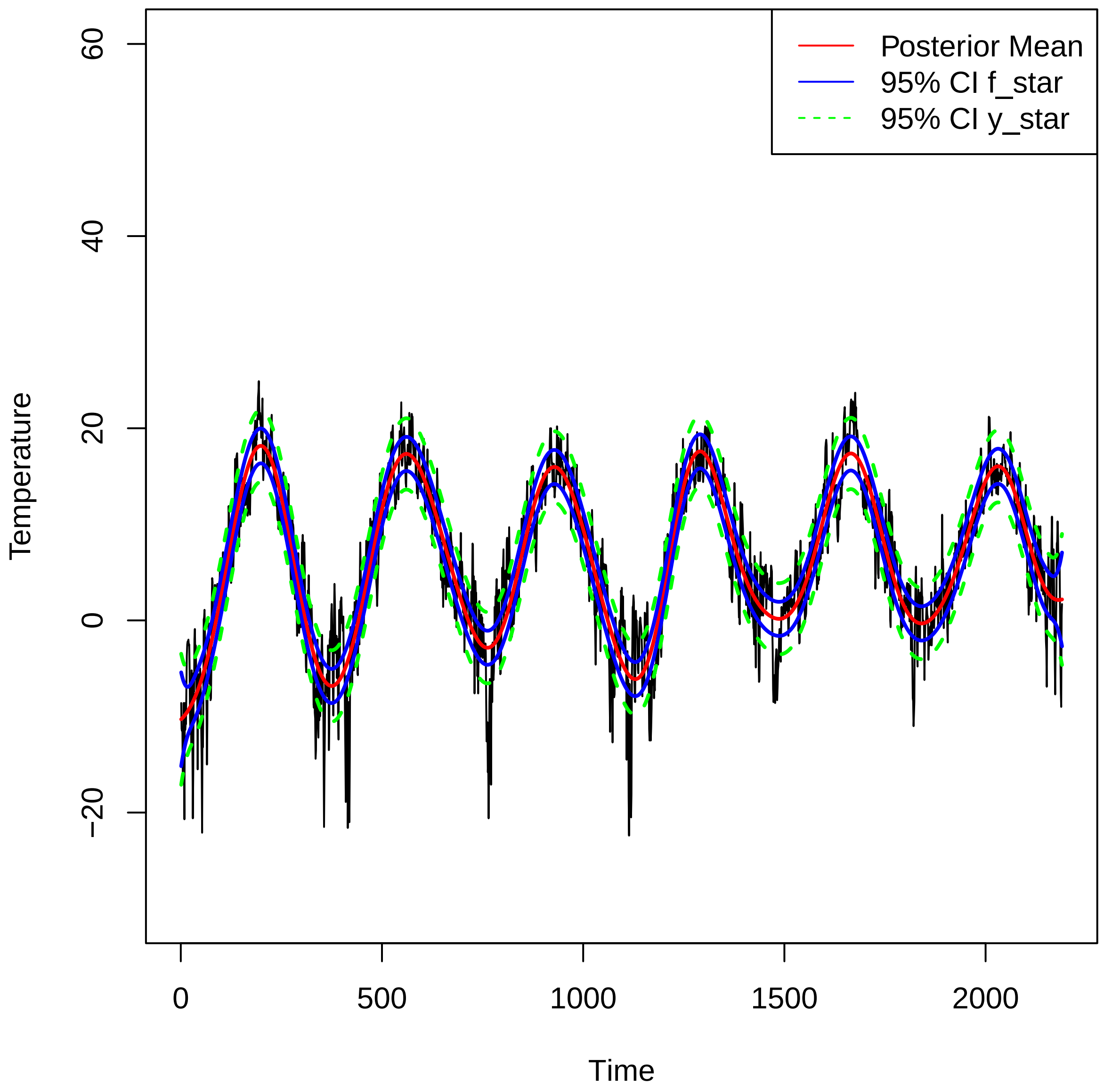
Sub Question 2

```r
lm_fit <- lm(temp~time+time2, data = train_data)
sigma2_resid <- var(lm_fit$residuals)
se_kern <- SEkernel(l = 0.2, sigmaF = 20)

GPfit <- gausspr(train_data$time, train_data$temp,
                 kernel = se_kern,
                 variance.model = TRUE,
                 var = sigma2_resid)

meanPred  <- predict(GPfit, train_data$time)
sdPred <- predict(GPfit,train_data$time, type="sdeviation")

plot(train_data$time, data$temp, type = 'l',ylim = c(-25, 50))
lines(train_data$time, meanPred, col = 'red', lwd=2.0)
# We use 0.1 because of a bug in the kernlab
# Using 1.96 makes the CI bands to be too wide
lines(train_data$time, meanPred+0.1*sdPred,col="blue", lwd = 2.0)
lines(train_data$time, meanPred-0.1*sdPred,col="blue", lwd = 2.0)
legend("topright", legend = c("Predictive Mean - fStar", "95% CI for fStar"),
       col = c("red",'blue'), lty = c(1, 1))
```



Sub Question 3

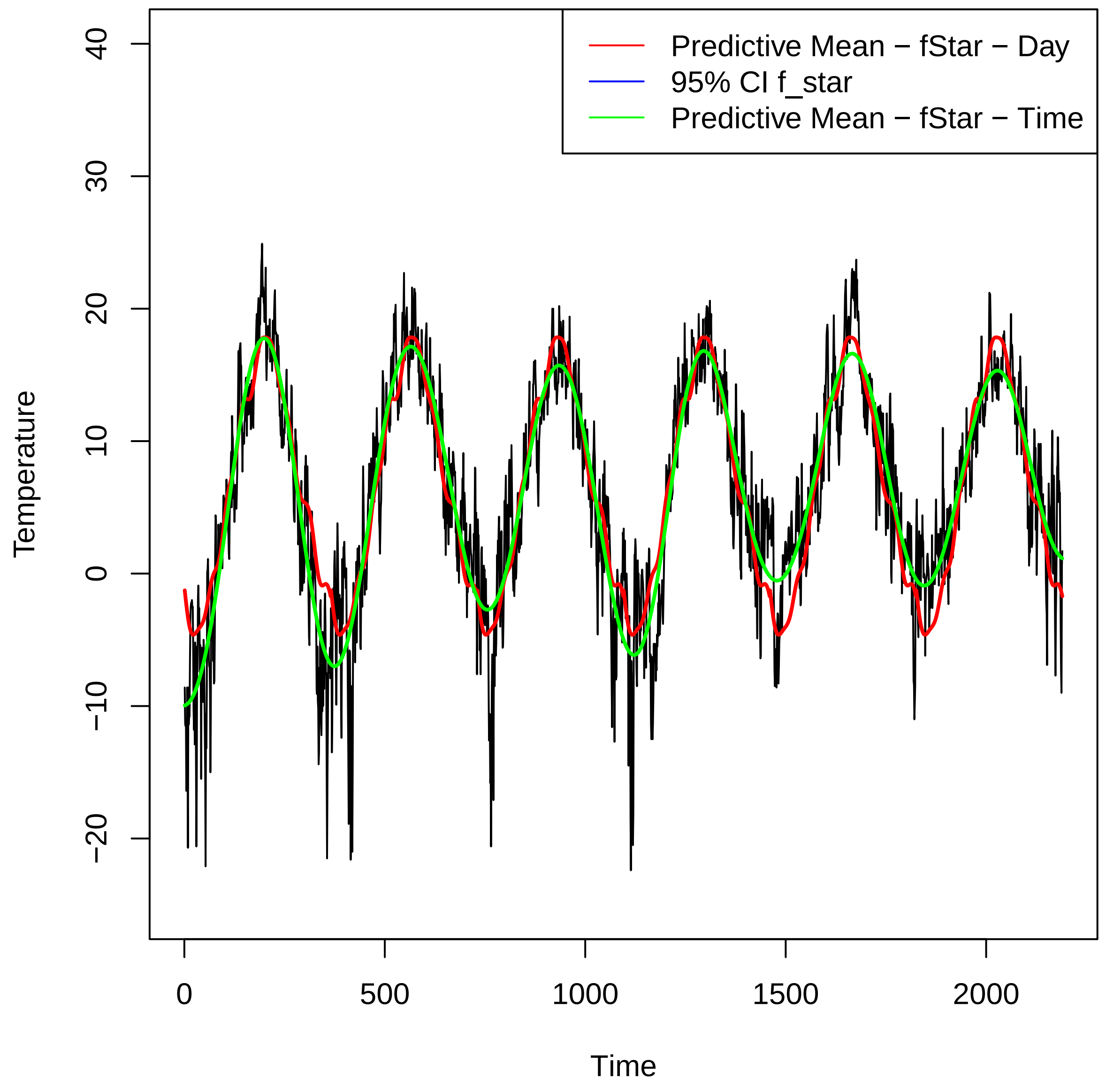
```r
train_data_scaled <- train_data
train_data_scaled$temp <- scale(train_data$temp)[,1]
train_data_scaled$time <- scale(train_data$time)[,1]
train_data_scaled$time2 <- scale(train_data$time2)[,1]

lm_scaled <- lm(formula = temp~time+time2, data = train_data_scaled)

sigma2_scale <- var(lm_scaled$residuals)

post_GP_temp <- posteriorGP(X = train_data_scaled$time,
                            y = train_data_scaled$temp,
                            XStar = train_data_scaled$time,
                            sigmaNoise = sqrt(sigma2_scale), k = se_kern)

unscaled_post_mean_temp <-(post_GP_temp$fStar*sd(train_data$temp))+mean(train_data$temp)
unscaled_post_var_temp<- sqrt(diag(post_GP_temp$var_fStar))*sd(train_data$temp)


plot(train_data$time, train_data$temp, type = 'l', ylim = c(-30, 60), xlab = 'Time',
     ylab = 'Temperature')
lines(unscaled_post_mean_temp, col = 'red', lwd = 2.0)
lines(train_data$time,  unscaled_post_mean_temp + 1.96*(unscaled_post_var_temp),
      col = 'blue', lwd = 2.0)
lines(unscaled_post_mean_temp - 1.96*(unscaled_post_var_temp),
      col = 'blue', lwd = 2.0)
lines(unscaled_post_mean_temp+ 1.96*(unscaled_post_var_temp + sqrt(sigma2_scale)), col = 'green', lwd =
lines(unscaled_post_mean_temp- 1.96*(unscaled_post_var_temp + sqrt(sigma2_scale)), col = 'green', lwd =
legend("topright", legend = c("Posterior Mean", "95% CI f_star", "95% CI y_star"),
       col = c("red", "blue", "green"), lty = c(1, 1, 2))
```



Sub Question 4

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.

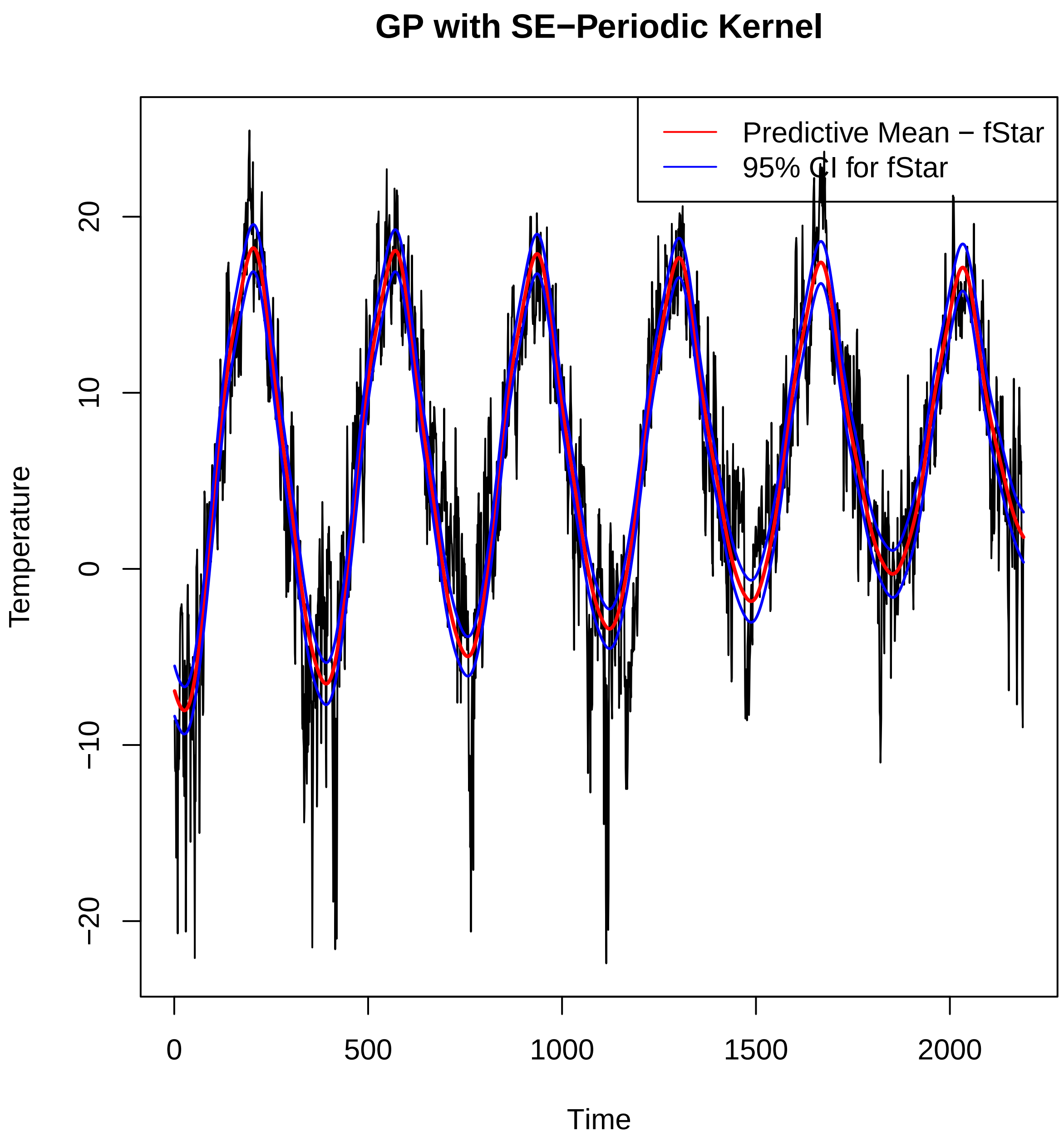
```r
lm_fit2 <- lm(temp~day+(day**2), data = train_data)
sigma2_resid2 <- var(lm_fit2$residuals)

GPfit_2 <- gausspr(train_data$day, train_data$temp,
                 kernel = se_kern,
                 variance.model = TRUE,
                 var = sigma2_resid2)

meanPred2  <- predict(GPfit_2, train_data$day)
sdPred2 <- predict(GPfit_2,train_data$day, type="sdeviation")

plot(train_data$time, data$temp, type = 'l', ylim = c(-25, 40), xlab = 'Time', ylab = 'Temperature')
lines(train_data$time, meanPred2, col = 'red', lwd=2.0)
# We use 0.1 because of a bug in the kernlab
# Using 1.96 makes the CI bands to be too wide
# lines(train_data$time, meanPred2+0.1*sdPred2,col="blue", lwd = 2.0)
# lines(train_data$time, meanPred2-0.1*sdPred2,col="blue", lwd = 2.0)
lines(train_data$time, meanPred, col = 'green', lwd = 2.0)
legend("topright", legend = c("Predictive Mean - fStar - Day",
                              "95% CI f_star",
                              "Predictive Mean - fStar - Time"),
       col = c("red", "blue", 'green'),
       lty = c(1, 1, 1))
```



Sub Question 5

Compare and discuss the results: As shown below, 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.)

```r
SE_periodic_kernel <- function(l1 = 1, l2 = 1, sigmaF = 1, d = 1){
  kern_val <- function(x, x_prime){
    res <- matrix(0, nrow = length(x), ncol = length(x_prime))
    for (i in 1:length(x)) {
      for (j in 1:length(x_prime)) {
        res[i, j] <- abs(x[i] - x_prime[j])
      }
    }
    return((sigmaF**2)*exp(-(2*sin((pi*res)/d)**2)/(l1**2))*exp(-(res**2)/(2*l2**2)))
  }
  class(kern_val) <- 'kernel'
  return(kern_val)
}
se_per_kern <- SE_periodic_kernel(l1 = 1, l2 = 10, sigmaF = 20, d = 365/sd(train_data$time))

GP_fit3 <- gausspr(train_data$time, train_data$temp,
                   kernel = se_per_kern,
                   variance.model = TRUE,
                   var = sigma2_resid)

meanPred3  <- predict(GP_fit3, train_data$time)
sdPred3 <- predict(GP_fit3,train_data$time, type="sdeviation")

plot(train_data$time, train_data$temp, type = 'l', xlab = 'Time',
     ylab = 'Temperature', main = 'GP with SE-Periodic Kernel')
lines(train_data$time, meanPred3, col = 'red', lwd=2.0)
lines(train_data$time, meanPred3+0.1*sdPred3,col="blue", lwd = 1.5)
lines(train_data$time, meanPred3-0.1*sdPred3,col="blue", lwd = 1.5)
legend("topright", legend = c("Predictive Mean - fStar", "95% CI for fStar"),
       col = c("red",'blue'), lty = c(1, 1))
```



Question 2.3 GP Classification with kernlab

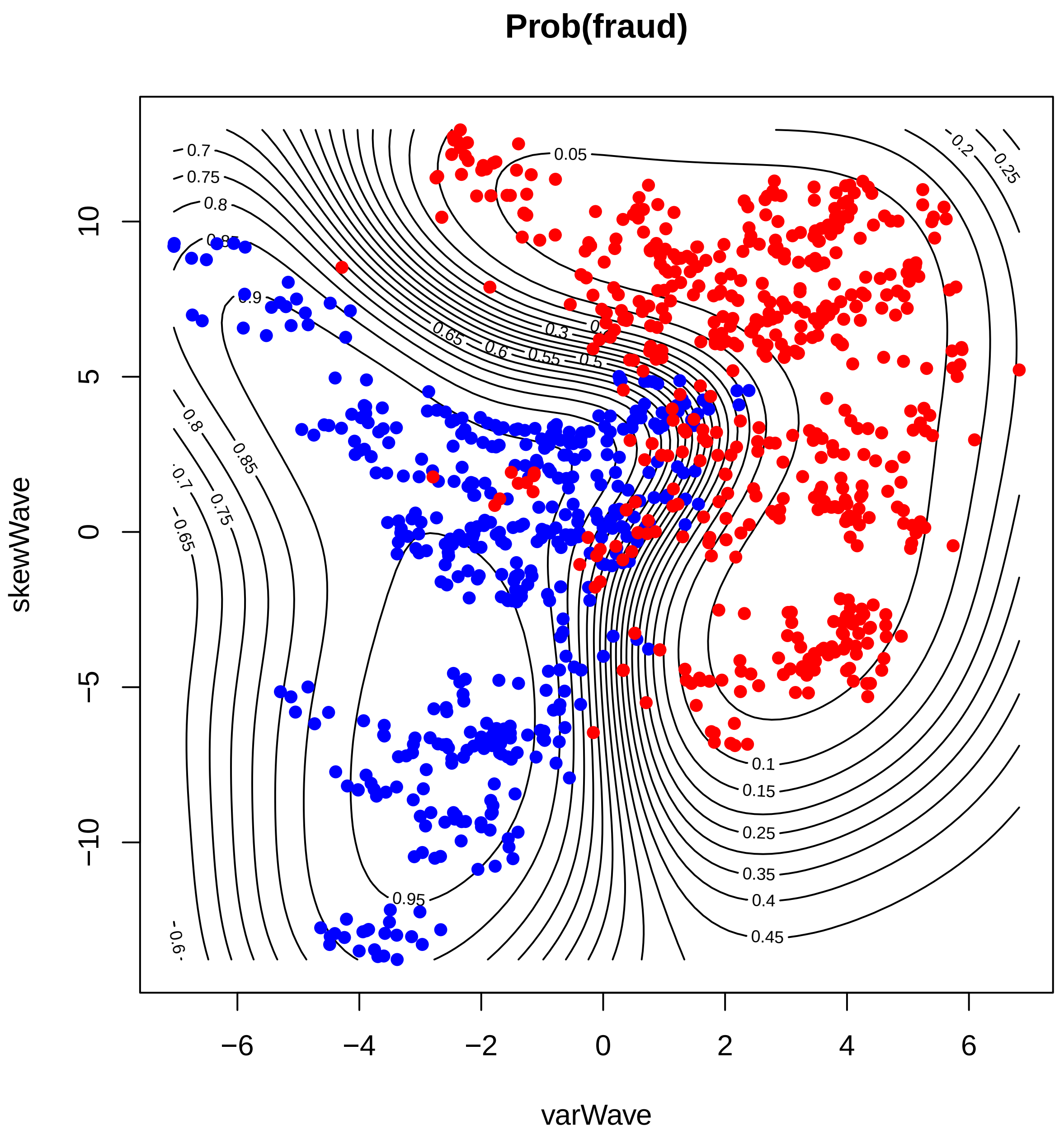
Sub Question 1

```
data2 <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFrau
names(data2) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data2[,5] <- as.factor(data2[,5])

set.seed(111)
```

```r
SelectTraining <- sample(1:dim(data2)[1], size = 1000,
                                        replace = FALSE)

train_data1 <- data2[SelectTraining, ]
test_data1 <- data2[-SelectTraining, ]

train_data1 <- train_data1[,c(1,2,5)]
test_data1 <- test_data1[,c(1,2,5)]

GPfitFraud <- gausspr(fraud~varWave+skewWave, data = train_data1)
```

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

```r
train_pred <- predict(object = GPfitFraud, train_data1[,1:2])
conf_mat_train <- table(train_data1$fraud, train_pred)
train_acc <- sum(diag(conf_mat_train))/sum(conf_mat_train)

probPreds <- predict(GPfitFraud, train_data1[,1:2], type = 'probabilities')

x1 <- seq(min(train_data1[,1]),max(train_data1[,1]),length=100)
x2 <- seq(min(train_data1[,2]),max(train_data1[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train_data1)[1:2]

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

# Plotting for Prob(fraud = 1)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud)')
points(train_data1[train_data1[,3]==1,1],train_data1[train_data1[,3]==1,2],col="blue", pch = 16)
points(train_data1[train_data1[,3]==0,1],train_data1[train_data1[,3]==0,2],col="red", pch = 16)
```



```
print('The confusion matrix for the classifier is:')
```

```
## [1] "The confusion matrix for the classifier is:"
```

```
conf_mat_train
```

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

```
cat('The resulting accuracy is:', train_acc)
```

```
## The resulting accuracy is: 0.941
```

Sub Question 2

```
test_pred <- predict(object = GPfitFraud, test_data1[,1:2])
conf_mat_test <- table(test_data1$fraud, test_pred)
test_acc <- sum(diag(conf_mat_test))/sum(conf_mat_test)

print('The confusion matrix for the classifier is:')
```

```
## [1] "The confusion matrix for the classifier is:"
```

```
conf_mat_test
```

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

```
cat('The resulting accuracy is:', test_acc)
```

```
## The resulting accuracy is: 0.9247312
```

Sub Question 3

```
train_data2 <- data2[SelectTraining, ]
test_data2 <- data2[-SelectTraining,]

GPfitFraud2 <- gausspr(fraud~., data = train_data2)
```

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

```
train_pred2 <- predict(object = GPfitFraud2, train_data2[,1:4])

conf_mat_train2 <- table(train_data2$fraud, train_pred2)
train_acc2 <- sum(diag(conf_mat_train2))/sum(conf_mat_train2)

test_pred2 <- predict(object = GPfitFraud2, test_data2[,1:4])
conf_mat_test2 <- table(test_data2$fraud, test_pred2)
test_acc2 <- sum(diag(conf_mat_test2))/sum(conf_mat_test2)

print('The confusion matrix for the classifier is:')
```

```
## [1] "The confusion matrix for the classifier is:"
```

```r
conf_mat_test2
```

```
##    test_pred2
##       0   1
##   0 216   2
##   1   0 154
```

```r
cat('The resulting accuracy is:', test_acc2)
```

```
## The resulting accuracy is: 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 boundary were 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.

Some notes on GP:

ell=1 is too smooth, and sigmaNoise=1 implies that the points are not trustworthy as function values. So, the GP does not really try to go through them and the probability bands are wide. Setting sigmaNoise=.1 reduces the probability bands drastically, as the points are now trustworthy. However, the GP is still too smooth to fit the data well.

ell=0.3 is quite flexible and, thus, the GP goes through the points with sigmaNoise=0.1. However, with sigmaNoise=1, the GP does not see any reason to go through the points, since they are very noisy versions of the function values. Hence the probability bands are wide too.

See also p. 21 in the book by Rasmussen and Williams.

Effect of $\sigma_n^2 = 0$ the noise variance:

The posterior covariance is zero at the training points because the functions must go through them, i.e. there is no uncertainty due to this being a noisy-free problem.

Effect of $\sigma_n^2 = 0$ the noise variance on covariance:

The posterior covariance is not monotone decreasing with the distance because it is constrained by the fact of being zero in the training points.

Effect of σ_f^2 on smoothness and overall variance of covariance function:

Changing sigma2f will have not effect on the relative covariance between points on the curve, i.e. will not affect the smoothness. But lowering sigma2f makes the whole covariance curve lower. This means that the variance k(0,0) is lower and has the effect of giving a probability distribution over curves which is tighter (lower variance) around the mean function of the GP. This means that simulated curves from the GP will be less variable.

Optimizing parameters from a Bayesian Perspective:

1. The marginal likelihood can be used to select optimal hyperparameters, and also the noise variance. We can optimize the log marginal likelihood with respect to the hyperparameters. In Gaussian Process Regression the marginal likelihood is availble in closed form (a formula).
2. We can use sampling methods (e.g. MCMC) to sample from the marginal posterior of the hyperparameters We need a prior p(theta) for the hyperparameter and then Bayes rule gives the marginal posterior p(theta|data) propto p(data | theta)*p(theta) where p(data | theta) is the marginal likelihood (f has been integrated out).

Bayesian Perspective when noise variance is unknown:

If the noise variance is unknown, we can treat like any of the kernel hyperparameters and infer the noise variance jointly with the length scale and the prior variance sigma_f

```r
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
library(AtmRay)
# Implementing GP Regression
# Algorithm 2.1

# Squared Exponential kernel or otherwise called
# Covariance 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)
}

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))))

  alph <- solve(t(L), solve(L,y))

  # Predictive Mean
  fStar <- t(Kstar) %*% alph

  # Predictive Variance
  v <- solve(L, Kstar)
  var_fStar <- k(XStar, XStar)- t(v) %*% v

  # Return this when required
  # log_marg_ll <- (-0.5*(t(y) %*% alph)) -sum(log(diag(L))) - (0.5*n*log(2*pi))
  return(list('fStar' = fStar, 'var_fStar' = var_fStar))
}
first_data <- c(0.4 ,0.719)
xstar = seq(-1,1,length=100)
sigmaNoise=0.1
k = SquaredExpKernel
post_GP <- posteriorGP(first_data[1], first_data[2], xstar, sigmaNoise, k)

mean_PostPred <- post_GP$fStar
var_PostPred <- post_GP$var_fStar

plot(xstar,mean_PostPred,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred + 1.96*sqrt(diag(var_PostPred)), lty = 5)
lines(xstar, mean_PostPred - 1.96*sqrt(diag(var_PostPred)), lty = 5)
points(first_data[1], first_data[2], pch = 23, col = 'red',  bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
x_2 <- c(0.4, -0.6)
y_2 <- c(0.719, -0.044)
post_GP2 <- posteriorGP(x_2, y_2, xstar, sigmaNoise, k)

mean_PostPred2 <- post_GP2$fStar
var_PostPred2 <- post_GP2$var_fStar

plot(xstar,mean_PostPred2,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred2 + 1.96*sqrt(diag(var_PostPred2)), lty = 5)
lines(xstar, mean_PostPred2 - 1.96*sqrt(diag(var_PostPred2)), lty = 5)
points(x_2, y_2, pch = 23, col = 'red', bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
x_3 <- c(-1, -0.6, -0.2, 0.4, 0.8)
y_3 <- c(0.768, -0.044, -0.940, 0.719, -0.664)

post_GP3 <- posteriorGP(x_3, y_3, xstar, sigmaNoise, k)

mean_PostPred3 <- post_GP3$fStar
var_PostPred3 <- post_GP3$var_fStar

plot(xstar,mean_PostPred3,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred3 + 1.96*sqrt(diag(var_PostPred3)), lty = 5)
lines(xstar, mean_PostPred3 - 1.96*sqrt(diag(var_PostPred3)), lty = 5)
points(x_3, y_3, pch = 23, col = 'red', bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))
x_4 <- c(-1, -0.6, -0.2, 0.4, 0.8)
y_4 <- c(0.768, -0.044, -0.940, 0.719, -0.664)

post_GP4 <- posteriorGP(x_4, y_4, xstar, sigmaNoise, k, sigmaF = 1, l = 1)

mean_PostPred4 <- post_GP4$fStar
var_PostPred4 <- post_GP4$var_fStar

plot(xstar,mean_PostPred4,col='red',ylab='pos mean',type='l', ylim = c(-3,3))
lines(xstar, mean_PostPred4 + 1.96*sqrt(diag(var_PostPred4)), lty = 5)
lines(xstar, mean_PostPred4 - 1.96*sqrt(diag(var_PostPred4)), lty = 5)
points(x_4, y_4, pch = 23, col = 'red', bg = 'blue')
legend("topright", legend = c("Predictive Mean", "Predictive Variance", "Observed Data"),
       col = c("red", "black",'blue'), lty = c(1, 2, NA), pch = c(NA, NA, 16))

# Generating the next x based on maximum uncertainty

upper <- mean_PostPred + 1.96*sqrt(diag(var_PostPred))
lower <- mean_PostPred - 1.96*sqrt(diag(var_PostPred))

uncertainty <- which.max(upper - lower)
```

```r
next_x_seq <- seq(10, 20, 0.1)
sigmaN <- 6
for (i in 1:20) {
  y_vals2 <- Yfun(X)
  xData <- X
  yData <- Yfun(X)
  post_GP_unc <- posteriorGP(X = xData, y = yData,XStar = next_x_seq, sigmaNoise=sigmaN, k = SquaredExpl
  mean_PostPred <- post_GP_unc$fStar
  var_PostPred <- post_GP_unc$var_fStar

  upper <- mean_PostPred + 1.96*sqrt(diag(var_PostPred)+sigmaN^2)
  lower <- mean_PostPred - 1.96*sqrt(diag(var_PostPred)+sigmaN^2)

  X <- c(X, next_x_seq[which.max(upper-lower)])
}

# To plot the covariance of f for x belongs to [-1,1] and x' = 0
cov_x0 <- var_PostPred4[,which(xstar==0)]
plot(xstar, cov_x0, ylim = c(-.1, 1))
cov_x0[which(xstar== -.4)]
# Calculate covariance at multiple xstar points
cov_xstar <- diag(var_PostPred4)

# Grid Search for one parameter:

sigmaSeq <- seq(0.1, 10, 0.1)
log_Lik <- c()

for (i in 1:length(sigmaSeq)) {
  post_GP <- posteriorGP(X = scale(time),
                         y = scale(temp),
                         XStar = scale(time),
                         sigmaNoise = sigmaSeq[i], k = SEkernel_func)
  log_Lik <- c(log_Lik, post_GP$logMarg)
}
sigmaSeq[which.max(log_Lik)]

# NOTE To use optim and optimize one parameter modify the posteriorGP function to
# return only the log marginal likelihood
# Example usage: posteriorGP2 is modified version of posteriorGP as per above
# comment.
optim_val <- optim(par = 0.20, fn = posteriorGP2,
                   X = scale(time),
                   y = scale(temp),
                   XStar = scale(time),
                   k = SEkernel_func, method = 'L-BFGS-B',
                   lower = c(.Machine$double.eps),control=list(fnscale=-1))

# NOTE To use optim and perform grid search to optimize two hyperparameters at once
# use the following set-up:

SEKernel2 <- function(par=c(20,0.2),x1,x2){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- (par[1]^2)*exp(-0.5*( (x1-x2[i])/par[2])^2 )
  }
  return(K)
}

LM <- function(par=c(20,0.2),X,y,k,sigmaNoise){
  n <- length(y)
  L <- t(chol(k(par,X,X)+((sigmaNoise^2)*diag(n))))
  a <- solve(t(L),solve(L,y))
  logmar <- -0.5*(t(y)%*%a)-sum(log(diag(L)))-(n/2)*log(2*pi)
  return(logmar)
}

bestLM<-LM(par=c(20,0.2),X=scale(time),
           y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit) # Grid search
bestLM
besti<-20
bestj<-0.2
for(i in seq(1,50,1)){
  for(j in seq(0.1,10,0.1)){
    aux<-LM(par=c(i,j),X=scale(time),y=scale(temp),
            k=SEKernel2,sigmaNoise=sigmaNoiseFit)
    if(bestLM<aux){
      bestLM<-aux
      besti<-i
      bestj<-j
    }
  }
}
bestLM
besti
bestj

foo<-optim(par = c(1,0.1), fn = LM, X=scale(time),
           y=scale(temp),k=SEKernel2,sigmaNoise=sigmaNoiseFit, method="L-BFGS-B",
           lower = c(.Machine$double.eps, .Machine$double.eps),
           control=list(fnscale=-1)) # Alternatively, one can use optim

foo$value
foo$par[1]
foo$par[2]

# To perform hyperparameter search by optimizing validation accuracy:

sigmaSeq <- seq(0.1, 10, 0.1)
acc <- c()
train_acc <- c()
# Grid Search
for (i in 1:length(sigmaSeq)) {
  GPfitFraud <- gausspr(fraud~., data = train_data1,
```

```r
                        kernel = 'rbfdot',
                        kpar = list(sigma = sigmaSeq[i]))
  valid_pred <- predict(object = GPfitFraud, valid_data1[,1:4])
  conf_mat_valid <- table(valid_data1$fraud, valid_pred)
  train_acc <- c(train_acc, sum(diag(conf_mat_valid))/sum(conf_mat_valid))
}

# Optim:
acc_func <- function(par){
  GPfitFraud <- gausspr(fraud~., data = train_data1,
                        kernel = 'rbfdot',
                        kpar = list(sigma = par[1]))
  valid_pred <- predict(object = GPfitFraud, valid_data1[,1:4])
  conf_mat_valid <- table(valid_data1$fraud, valid_pred)
  valid_acc <- sum(diag(conf_mat_valid))/sum(conf_mat_valid)
  return(valid_acc)
}

optim(par = c(1),
      fn = acc_func,
      method = 'L-BFGS-B',
      lower = c(.Machine$double.eps),
      control=list(fnscale=-1))

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)
}

zGrid = seq(0.01,1,by=0.01)

matern_kern1 <- Matern32(sigmaf = 1, ell = 0.5)
matern_kern2 <- Matern32(sigmaf = 0.5, ell = 0.5)

# For computing covariance as k(0,z)
cov_vals1 <- c()
cov_vals2 <- c()

for (i in zGrid) {
  cov_vals1 <- c(cov_vals1, matern_kern1(x = 0, y = i))
  cov_vals2 <- c(cov_vals2, matern_kern2(x = 0, y = i))
}

# As per GP Lecture 1: Slide 21
# For posterior mean without Algorithm 2.1, refer same slide
# Posterior Variance without Algorithm 2.1
n <- length(distance)
Kss <- kernelMatrix(kernel = matern_kern_gp2, x = xs, y = xs)
```

```r
Kxx <- kernelMatrix(kernel = matern_kern_gp2, x = distance, y = distance)
Kxs <- kernelMatrix(kernel = matern_kern_gp2, x = distance, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs) # Covariance matrix of fStar.

# Prediction intervals for yStar.
lines(xs, postMean2 - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "blue")
lines(xs, postMean2 + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "blue")

# Simulating from a GP

SimGP <- function(m = 0,K,x,nSim,...){
  n <- length(x)
  if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(x)
  covMat <- K(x,x,...)
  f <- rmvnorm(nSim, mean = meanVector, sigma = covMat)
  return(f)
}

sigmaF <- 1
l <- 0.2
nSim <- 5
fSim <- SimGP(m=0, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l)
plot(xGrid, fSim[1,], type="l", ylim = c(-3,3), lwd = 2)
cols <- rainbow(5)
if(nSim>1){
  for (i in 2:nSim) {
    lines(xGrid, fSim[i,], type="l", col = cols[i], lty = 2, lwd = 2)
  }
}

# Computing Corr(f(0), f(0.1)) given some hyperparameters
# NOTE SquaredExpKernel can be any kernel
SquaredExpKernel(x1 = 0, x2 = 0.1, sigmaF = 1, l = 1)

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

train_data <- data
train_data$time <- c(1:nrow(data))
train_data$time2 <- train_data$time**2
train_data$day <- rep(c(1:365), 6)

SEkernel <- function(l = 1, sigmaF = 1) {
  kernval <- function(x, x_prime = NULL) {
    res <- matrix(0, nrow = length(x), ncol = length(x_prime))
    for (i in 1:length(x)) {
      for (j in 1:length(x_prime)) {
        res[i, j] <- (x[i] - x_prime[j])^2
      }
    }
    res <- (sigmaF^2) * exp(-res / (2 * l^2))
    return(res)
  }
  class(kernval) <- 'kernel'
  return(kernval)
}


SEkernel_func <- SEkernel(1, 1)
# Evaluating the kernel at x=1 and x' = 2
cat("Kernel evaluation at x = 1 and x' = 2:",SEkernel_func(x = 1, x_prime = 2))

# Computing covariance matrix K(X, Xstar)
kernelMatrix(kernel = SEkernel_func, x = c(1,3,4), y = c(2, 3, 4))
lm_fit <- lm(temp~time+time2, data = train_data)
sigma2_resid <- var(lm_fit$residuals)
se_kern <- SEkernel(l = 0.2, sigmaF = 20)

GPfit <- gausspr(train_data$time, train_data$temp,
                 kernel = se_kern,
                 variance.model = TRUE,
                 var = sigma2_resid)

meanPred  <- predict(GPfit, train_data$time)
sdPred <- predict(GPfit,train_data$time, type="sdeviation")

plot(train_data$time, data$temp, type = 'l',ylim = c(-25, 50))
lines(train_data$time, meanPred, col = 'red', lwd=2.0)
# We use 0.1 because of a bug in the kernlab
# Using 1.96 makes the CI bands to be too wide
lines(train_data$time, meanPred+0.1*sdPred,col="blue", lwd = 2.0)
lines(train_data$time, meanPred-0.1*sdPred,col="blue", lwd = 2.0)
legend("topright", legend = c("Predictive Mean - fStar", "95% CI for fStar"),
       col = c("red",'blue'), lty = c(1, 1))

train_data_scaled <- train_data
train_data_scaled$temp <- scale(train_data$temp)[,1]
train_data_scaled$time <- scale(train_data$time)[,1]
train_data_scaled$time2 <- scale(train_data$time2)[,1]

lm_scaled <- lm(formula = temp~time+time2, data = train_data_scaled)

sigma2_scale <- var(lm_scaled$residuals)

post_GP_temp <- posteriorGP(X = train_data_scaled$time,
                            y = train_data_scaled$temp,
                            XStar = train_data_scaled$time,
                            sigmaNoise = sqrt(sigma2_scale), k = se_kern)

unscaled_post_mean_temp <-(post_GP_temp$fStar*sd(train_data$temp))+mean(train_data$temp)
unscaled_post_var_temp<- sqrt(diag(post_GP_temp$var_fStar))*sd(train_data$temp)


plot(train_data$time, train_data$temp, type = 'l', ylim = c(-30, 60), xlab = 'Time',
     ylab = 'Temperature')
lines(unscaled_post_mean_temp, col = 'red', lwd = 2.0)
lines(train_data$time,  unscaled_post_mean_temp + 1.96*(unscaled_post_var_temp),
      col = 'blue', lwd = 2.0)
lines(unscaled_post_mean_temp - 1.96*(unscaled_post_var_temp),
      col = 'blue', lwd = 2.0)
lines(unscaled_post_mean_temp+ 1.96*(unscaled_post_var_temp + sqrt(sigma2_scale)), col = 'green', lwd =
lines(unscaled_post_mean_temp- 1.96*(unscaled_post_var_temp + sqrt(sigma2_scale)), col = 'green', lwd =
legend("topright", legend = c("Posterior Mean", "95% CI f_star", "95% CI y_star"),
       col = c("red", "blue", "green"), lty = c(1, 1, 2))

lm_fit2 <- lm(temp~day+(day**2), data = train_data)
sigma2_resid2 <- var(lm_fit2$residuals)

GPfit_2 <- gausspr(train_data$day, train_data$temp,
                 kernel = se_kern,
                 variance.model = TRUE,
                 var = sigma2_resid2)

meanPred2  <- predict(GPfit_2, train_data$day)
sdPred2 <- predict(GPfit_2,train_data$day, type="sdeviation")

plot(train_data$time, data$temp, type = 'l', ylim = c(-25, 40), xlab = 'Time', ylab = 'Temperature')
lines(train_data$time, meanPred2, col = 'red', lwd=2.0)
# We use 0.1 because of a bug in the kernlab
# Using 1.96 makes the CI bands to be too wide
# lines(train_data$time, meanPred2+0.1*sdPred2,col="blue", lwd = 2.0)
# lines(train_data$time, meanPred2-0.1*sdPred2,col="blue", lwd = 2.0)
lines(train_data$time, meanPred, col = 'green', lwd = 2.0)
legend("topright", legend = c("Predictive Mean - fStar - Day",
                              "95% CI f_star",
                              "Predictive Mean - fStar - Time"),
       col = c("red", "blue", 'green'),
       lty = c(1, 1, 1))
SE_periodic_kernel <- function(l1 = 1, l2 = 1, sigmaF = 1, d = 1){
  kern_val <- function(x, x_prime){
    res <- matrix(0, nrow = length(x), ncol = length(x_prime))
    for (i in 1:length(x)) {
      for (j in 1:length(x_prime)) {
        res[i, j] <- abs(x[i] - x_prime[j])
      }
    }
    return((sigmaF**2)*exp(-(2*sin((pi*res)/d)**2)/(l1**2))*exp(-(res**2)/(2*l2**2)))
  }
  class(kern_val) <- 'kernel'
  return(kern_val)
}
se_per_kern <- SE_periodic_kernel(l1 = 1, l2 = 10, sigmaF = 20, d = 365/sd(train_data$time))

GP_fit3 <- gausspr(train_data$time, train_data$temp,
                   kernel = se_per_kern,
                   variance.model = TRUE,
                   var = sigma2_resid)

meanPred3  <- predict(GP_fit3, train_data$time)
sdPred3 <- predict(GP_fit3,train_data$time, type="sdeviation")
```

```r
plot(train_data$time, train_data$temp, type = 'l', xlab = 'Time',
     ylab = 'Temperature', main = 'GP with SE-Periodic Kernel')
lines(train_data$time, meanPred3, col = 'red', lwd=2.0)
lines(train_data$time, meanPred3+0.1*sdPred3,col="blue", lwd = 1.5)
lines(train_data$time, meanPred3-0.1*sdPred3,col="blue", lwd = 1.5)
legend("topright", legend = c("Predictive Mean - fStar", "95% CI for fStar"),
       col = c("red",'blue'), lty = c(1, 1))
data2 <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFrau
names(data2) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data2[,5] <- as.factor(data2[,5])

set.seed(111)
SelectTraining <- sample(1:dim(data2)[1], size = 1000,
                                        replace = FALSE)

train_data1 <- data2[SelectTraining, ]
test_data1 <- data2[-SelectTraining, ]

train_data1 <- train_data1[,c(1,2,5)]
test_data1 <- test_data1[,c(1,2,5)]

GPfitFraud <- gausspr(fraud~varWave+skewWave, data = train_data1)

train_pred <- predict(object = GPfitFraud, train_data1[,1:2])
conf_mat_train <- table(train_data1$fraud, train_pred)
train_acc <- sum(diag(conf_mat_train))/sum(conf_mat_train)

probPreds <- predict(GPfitFraud, train_data1[,1:2], type = 'probabilities')

x1 <- seq(min(train_data1[,1]),max(train_data1[,1]),length=100)
x2 <- seq(min(train_data1[,2]),max(train_data1[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train_data1)[1:2]

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

# Plotting for Prob(fraud = 1)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud)')
points(train_data1[train_data1[,3]==1,1],train_data1[train_data1[,3]==1,2],col="blue", pch = 16)
points(train_data1[train_data1[,3]==0,1],train_data1[train_data1[,3]==0,2],col="red", pch = 16)

print('The confusion matrix for the classifier is:')
conf_mat_train

cat('The resulting accuracy is:', train_acc)
test_pred <- predict(object = GPfitFraud, test_data1[,1:2])
conf_mat_test <- table(test_data1$fraud, test_pred)
test_acc <- sum(diag(conf_mat_test))/sum(conf_mat_test)

print('The confusion matrix for the classifier is:')
```

```r
conf_mat_test

cat('The resulting accuracy is:', test_acc)
train_data2 <- data2[SelectTraining, ]
test_data2 <- data2[-SelectTraining,]

GPfitFraud2 <- gausspr(fraud~., data = train_data2)
train_pred2 <- predict(object = GPfitFraud2, train_data2[,1:4])

conf_mat_train2 <- table(train_data2$fraud, train_pred2)
train_acc2 <- sum(diag(conf_mat_train2))/sum(conf_mat_train2)

test_pred2 <- predict(object = GPfitFraud2, test_data2[,1:4])
conf_mat_test2 <- table(test_data2$fraud, test_pred2)
test_acc2 <- sum(diag(conf_mat_test2))/sum(conf_mat_test2)

print('The confusion matrix for the classifier is:')
conf_mat_test2

cat('The resulting accuracy is:', test_acc2)
```