Advanced Machine Learning Lab 4

Simge Cinar

2024-10-13

Question 1

(1)

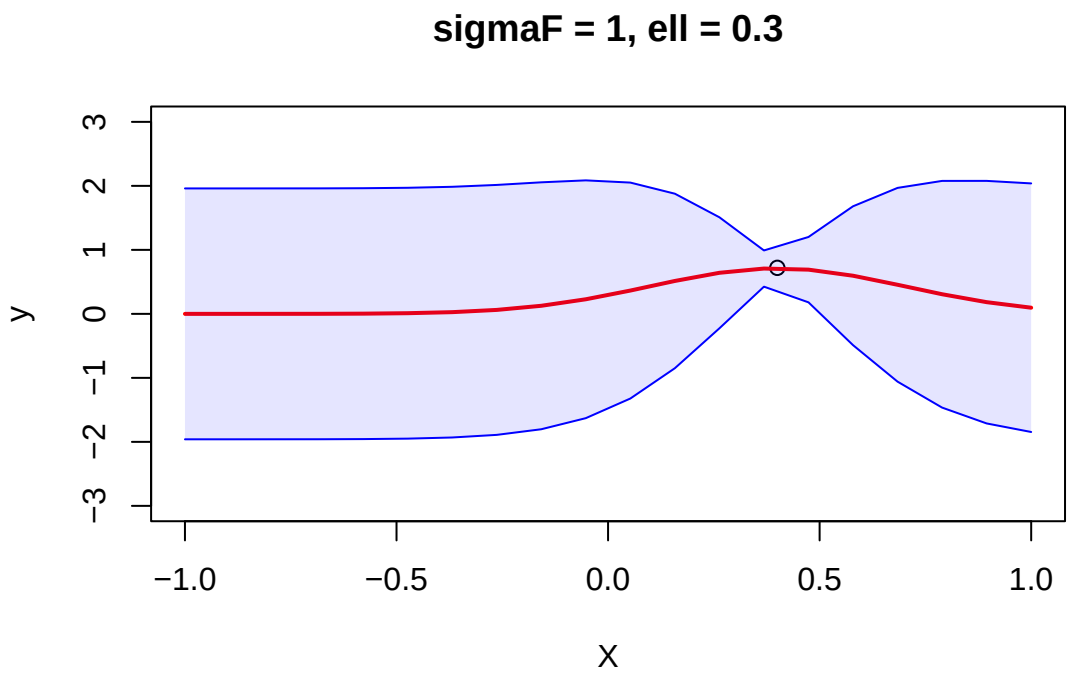
I implemented the posterior distribution of f using squared exponential kernel which returns a vector with posterior mean and variance of f.

```
# Covariance function from the course web page
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)
}

# 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 sigma_n.
# k: Covariance function or kernel. That is, the kernel should be a separate function

posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...){
  # '...' allows passing additional parameters to the kernel function

  n <- length(X)
  K <- k(X, X, ...)
  KStar <- k(X, XStar, ...)

  # Take the transpose since the algorithm in the book is a lower triangular matrix, whereas the R func
  L <- t(chol(K+ sigmaNoise^2 * diag(n)))
  alpha <- solve(t(L), solve(L,y))
  fStar <- t(KStar) %*% alpha
  v <- solve(L, KStar)
  V_fStar <- k(XStar,XStar, ...) - t(v) %*% v
  logp <- -1/2 %*% t(y) %*% alpha - sum(log(diag(L)))- (n/2)*log(2*pi)
  return(list(mean = fStar, cov = V_fStar, logLikelihood = logp))
}
```

(2)

Prior hyperparameters are $\sigma_f = 1$ and $\ell = 0.3$. Updated this prior with a single observation, (x, y) = (0.4, 0.719) and assumed that $\sigma_n = 0.1$.

```
X <- 0.4
y <- 0.719
XStar <- seq(-1,1, length.out = 20)
sigmaNoise <- 0.1

result_12 <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = 1, l = 0.3)

plot(X,y, xlim=c(-1,1), ylim = c(-3,3), main = "sigmaF = 1, ell = 0.3")
lines(XStar, result_12$mean, col="red", lwd = 2)
lines(XStar, result_12$mean - 1.96*sqrt(diag(result_12$cov)), col="blue")
lines(XStar, result_12$mean + 1.96*sqrt(diag(result_12$cov)), col="blue")

polygon(c(XStar, rev(XStar)),
        c(result_12$mean + 1.96*sqrt(diag(result_12$cov)),
          rev(result_12$mean - 1.96*sqrt(diag(result_12$cov)))), col=rgb(0, 0, 1, 0.1), border=NA)
```

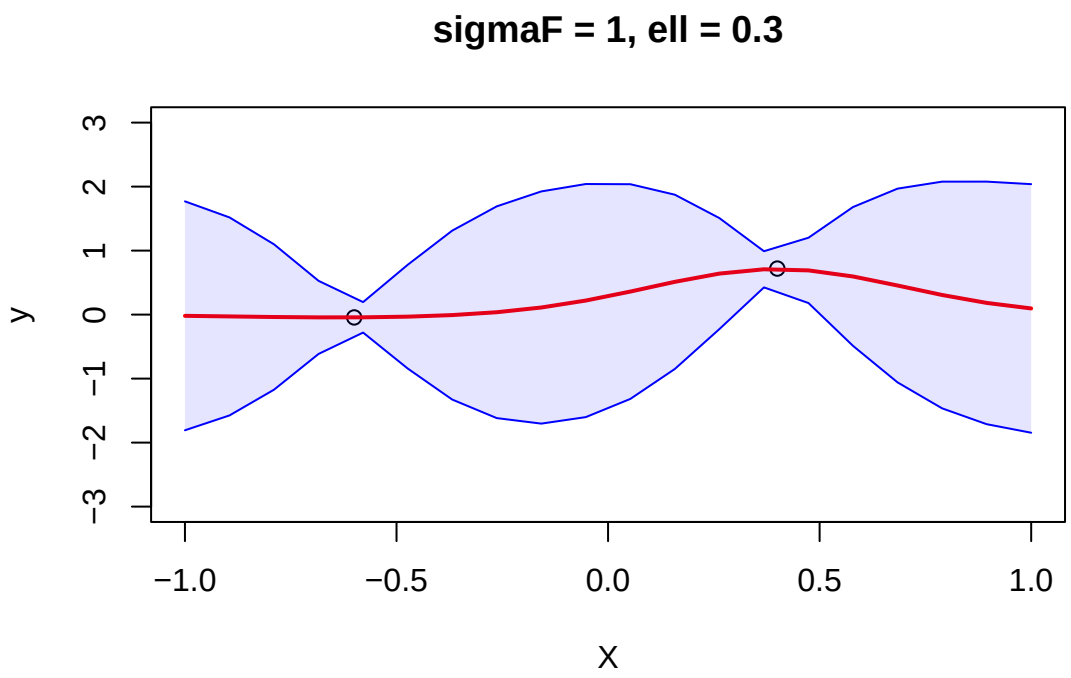


(3)

We have 2 observations x = (0.4, -0.6) and y = (0.719, -0.044) with same hyperparameters

```
X <- c(0.4, -0.6)
y <- c(0.719, -0.044)
XStar <- seq(-1,1, length.out = 20)
sigmaNoise <- 0.1


result_13 <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = 1, l = 0.3)

plot(X,y, xlim=c(-1,1), ylim = c(-3,3), main = "sigmaF = 1, ell = 0.3")
```

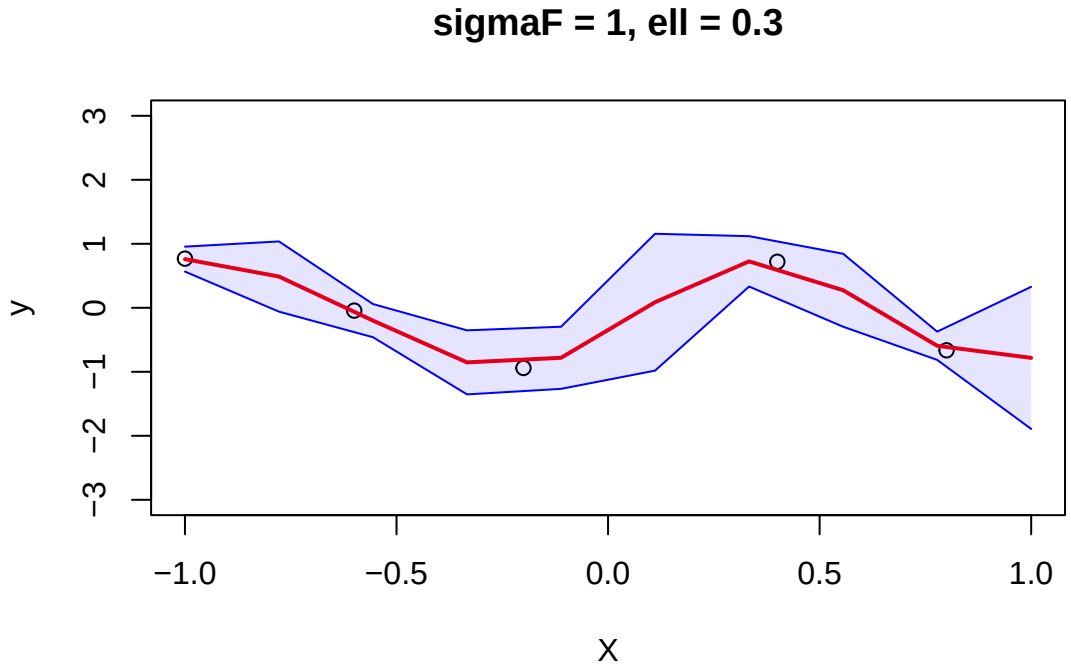
```
lines(XStar, result_13$mean, col="red", lwd = 2)
lines(XStar, result_13$mean - 1.96*sqrt(diag(result_13$cov)), col="blue")
lines(XStar, result_13$mean + 1.96*sqrt(diag(result_13$cov)), col="blue")

polygon(c(XStar, rev(XStar)),
        c(result_13$mean + 1.96*sqrt(diag(result_13$cov)),
          rev(result_13$mean - 1.96*sqrt(diag(result_13$cov)))), col=rgb(0, 0, 1, 0.1), border=NA)
```



(4)

We have 5 observations x = (-1, -0.6, -0.2, 0.4, 0.8) and y = (0.768, -0.044, -0.940, 0.719, -0.664) with same hyperparameters

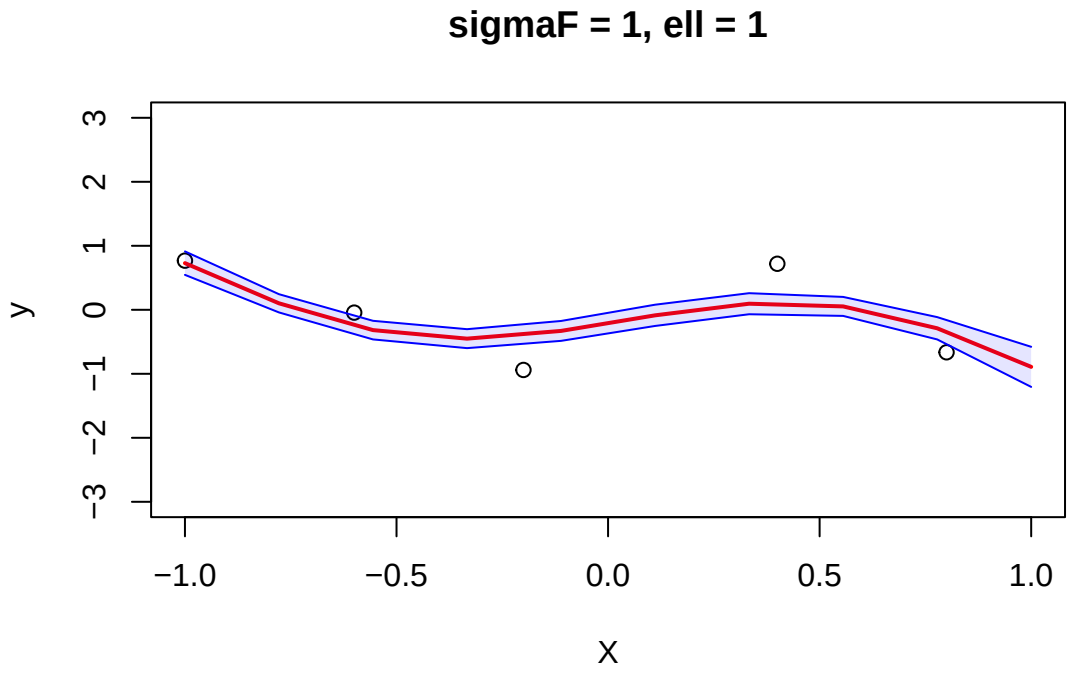
```
X <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
XStar <- seq(-1,1, length.out = 10)
sigmaNoise <- 0.1

result_14 <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = 1, l = 0.3)

plot(X,y, xlim=c(-1,1), ylim = c(-3,3), main = "sigmaF = 1, ell = 0.3")
lines(XStar, result_14$mean, col="red", lwd = 2)
lines(XStar, result_14$mean - 1.96*sqrt(diag(result_14$cov)), col="blue")
lines(XStar, result_14$mean + 1.96*sqrt(diag(result_14$cov)), col="blue")

polygon(c(XStar, rev(XStar)),
        c(result_14$mean + 1.96*sqrt(diag(result_14$cov)),
          rev(result_14$mean - 1.96*sqrt(diag(result_14$cov)))), col=rgb(0, 0, 1, 0.1), border=NA)
```



(5)

We have the previous 5 observations but we changed the hyperparameter ℓ from 0.3 to 1. The 95% confidence interval is more narrow compared to part (4) and don't contain most of the data points. When we increase ℓ, the function becomes more smooth and this might cause underfitting (but we cannot say this exactly without validation data)

```
X <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
XStar <- seq(-1,1, length.out = 10)
sigmaNoise <- 0.1


result_15 <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = 1, l = 1)

plot(X,y, xlim=c(-1,1), ylim = c(-3,3), main = "sigmaF = 1, ell = 1")
lines(XStar, result_15$mean, col="red", lwd = 2)
lines(XStar, result_15$mean - 1.96*sqrt(diag(result_15$cov)), col="blue")
lines(XStar, result_15$mean + 1.96*sqrt(diag(result_15$cov)), col="blue")

polygon(c(XStar, rev(XStar)),
        c(result_15$mean + 1.96*sqrt(diag(result_15$cov)),
          rev(result_15$mean - 1.96*sqrt(diag(result_15$cov)))), col=rgb(0, 0, 1, 0.1), border=NA)
```



Question 2

(1)

I did some data manipulation and created new variables as requested in the description. I created a function for square exponential kernel function with parameters ℓ and σ_f

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

df$date <- as.Date(df$date, format = "%d/%m/%y") # Convert to date format

# Adding new variables time and day
df$time <- seq(1, nrow(df), by = 1)
df$day <- as.numeric(df$date - as.Date(paste0(format(df$date, "%Y"), "-01-01"))) + 1

df_subsample <- df[seq(1, nrow(df), by = 5), ]
head(df_subsample)
```

```
##          date  temp time day
## 1  2010-01-01  -8.6    1   1
## 6  2010-01-06 -15.4    6   6
## 11 2010-01-11 -11.4   11  11
## 16 2010-01-16  -2.5   16  16
## 21 2010-01-21  -2.5   21  21
## 26 2010-01-26 -12.9   26  26
```

```
squared_exp <- function(sigmaf, ell){
  rval <- function(x, y) {
    r = sqrt(crossprod(x-y));
    return(sigmaf^2 * exp(-(r^2) / (2*ell^2)))
  }
  class(rval) <- "kernel"
  return(rval)
```

```
}

squared_exp_func = squared_exp(sigmaf = 1, ell = 1) # squared_exp_func is a kernel FUNCTION.
cat("Kernel function at x=1, x'=2:\n")
```

```
## Kernel function at x=1, x'=2:
```

```
squared_exp_func(1,2) # Evaluating the kernel at x=1, x'=2
```

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

```
X <- c(1,3,4)
Xstar <- c(2,3,4)
cat("Covariance matrix K(X, XStar):\n")
```

```
## Covariance matrix K(X, XStar):
```

```
#kernelMatrix(kernel = squared_exp_func, x = X, y = Xstar) # alternative 1
kernelMatrix(kernel = squared_exp(1,1), x = X, y = Xstar) # alternative 2
```

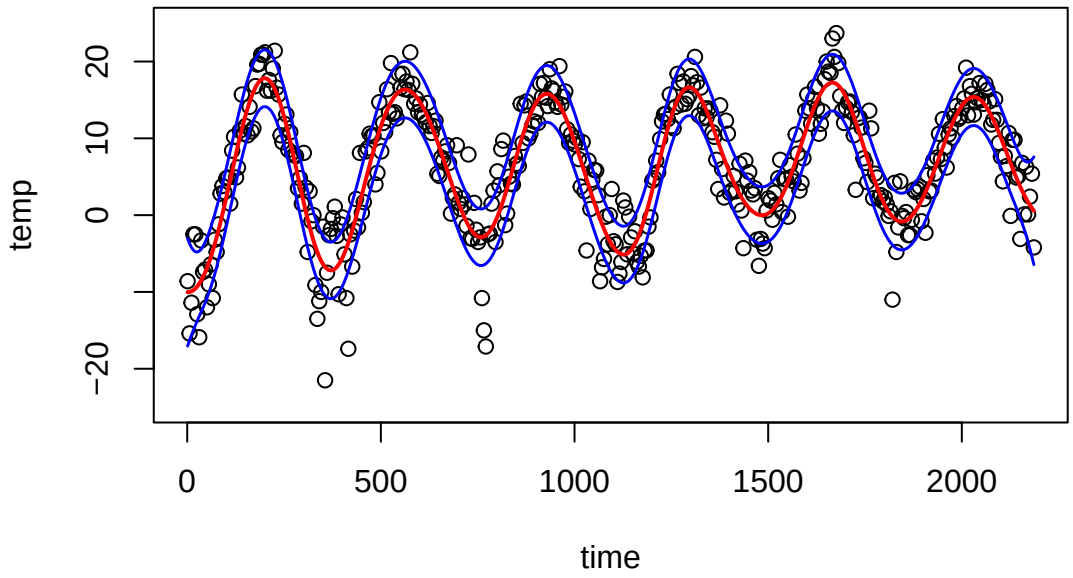
```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

(2)

```
temp_df <- df_subsample$temp
time_df <- df_subsample$time

polyFit <- lm(temp_df ~  time_df + I(time_df^2))
sigmaNoise_22 = sd(polyFit$residuals)
cat("Residual variance:", sigmaNoise_22^2)
```

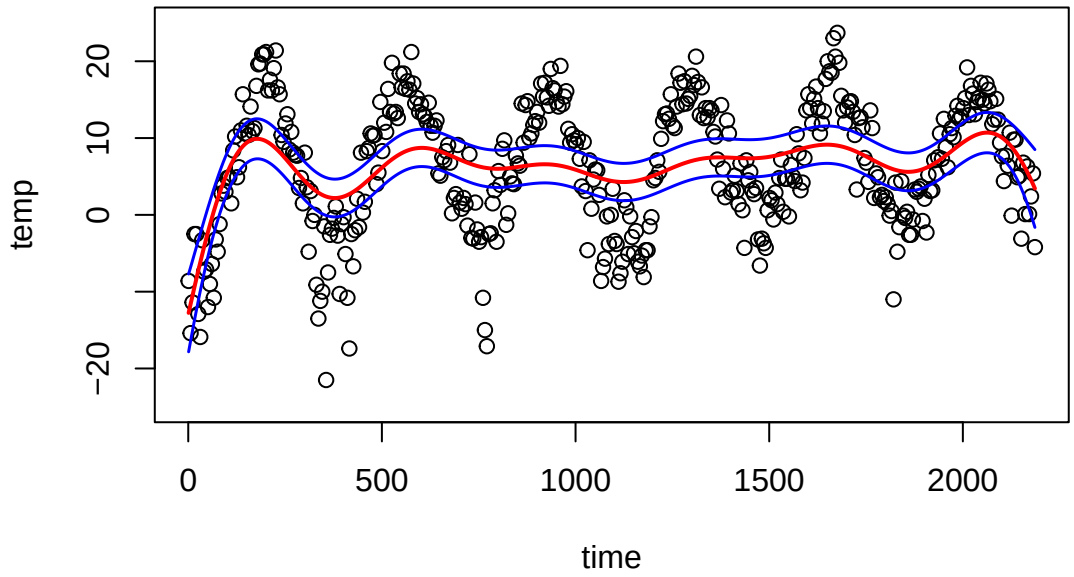
```
## Residual variance: 66.85169
```

```
GPfit_22 <- gausspr(time_df, temp_df,
                    kernel = squared_exp,
                    kpar = list(sigmaf = 20, ell = 100),
                    var = sigmaNoise_22^2,
                    scaled=FALSE, variance.model = TRUE)

meanPred_22 <- predict(GPfit_22, time_df) # Predicting the training data. To plot the fit.

plot(time_df, temp_df, xlab = "time", ylab = "temp", ylim = c(-25, 25))
lines(time_df, meanPred_22, col="red", lwd = 2)
lines(time_df, meanPred_22 + 1.96*predict(GPfit_22, time_df, type="sdeviation"),col="blue", lwd = 1.5)
lines(time_df, meanPred_22 - 1.96*predict(GPfit_22, time_df, type="sdeviation"),col="blue", lwd = 1.5)
```



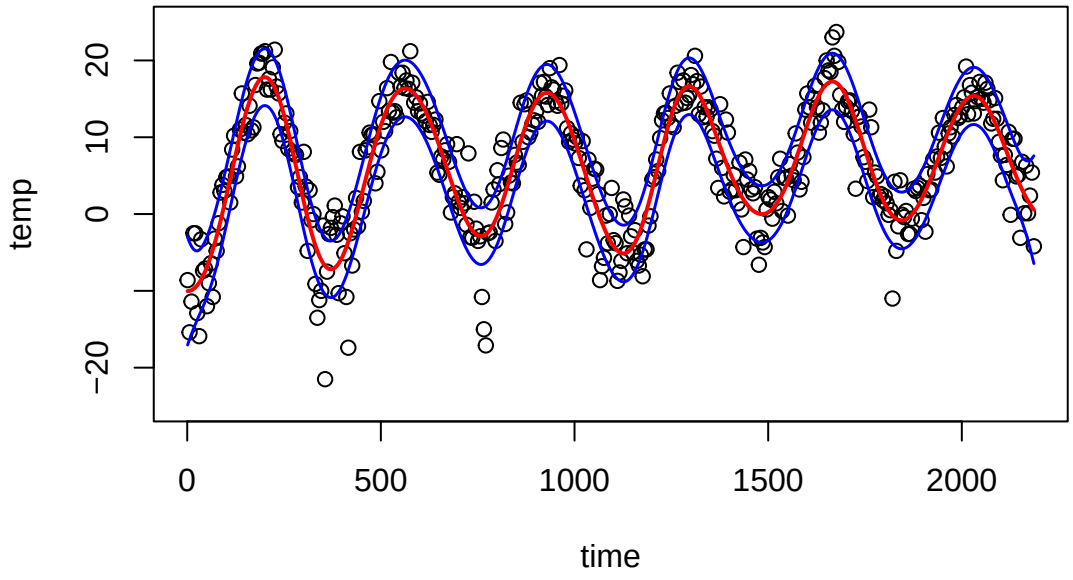
When I increase the ℓ parameter from 100 to 250, the function is more smooth and probably model is underfitting as it can be seen below.

```
GPfit_22_1 <- gausspr(time_df, temp_df,
                      kernel = squared_exp,
                      kpar = list(sigmaf = 20, ell = 250),
                      var = sigmaNoise_22^2,
                      scaled=FALSE, variance.model = TRUE)

meanPred_22_1 <- predict(GPfit_22_1, time_df)

plot(time_df, temp_df, xlab = "time", ylab = "temp", ylim = c(-25, 25))
lines(time_df, meanPred_22_1, col="red", lwd = 2)
lines(time_df, meanPred_22_1 + 1.96*predict(GPfit_22_1, time_df, type="sdeviation"),col="blue", lwd = 1
lines(time_df, meanPred_22_1 - 1.96*predict(GPfit_22_1, time_df, type="sdeviation"),col="blue", lwd = 1
```



(3)

```
result_23 <- posteriorGP(X = time_df,
                         y = temp_df,
                         XStar = time_df,
                         sigmaNoise = sigmaNoise_22,
                         k = SquaredExpKernel,
                         sigmaF = 20, l = 100)

posterior_mean <- result_23$mean
posterior_lower <- result_23$mean - 1.96*sqrt(diag(result_23$cov))
posterior_upper <- result_23$mean + 1.96*sqrt(diag(result_23$cov))

plot(time_df, temp_df, ylim = c(-25, 25), xlab = 'time', ylab = 'temp')
lines(time_df,result_23$mean, col="red", lwd = 2)
lines(time_df, result_23$mean - 1.96*sqrt(diag(result_23$cov)), col="blue", lwd = 1.5)
lines(time_df, result_23$mean + 1.96*sqrt(diag(result_23$cov)), col="blue", lwd = 1.5)
```



(4)

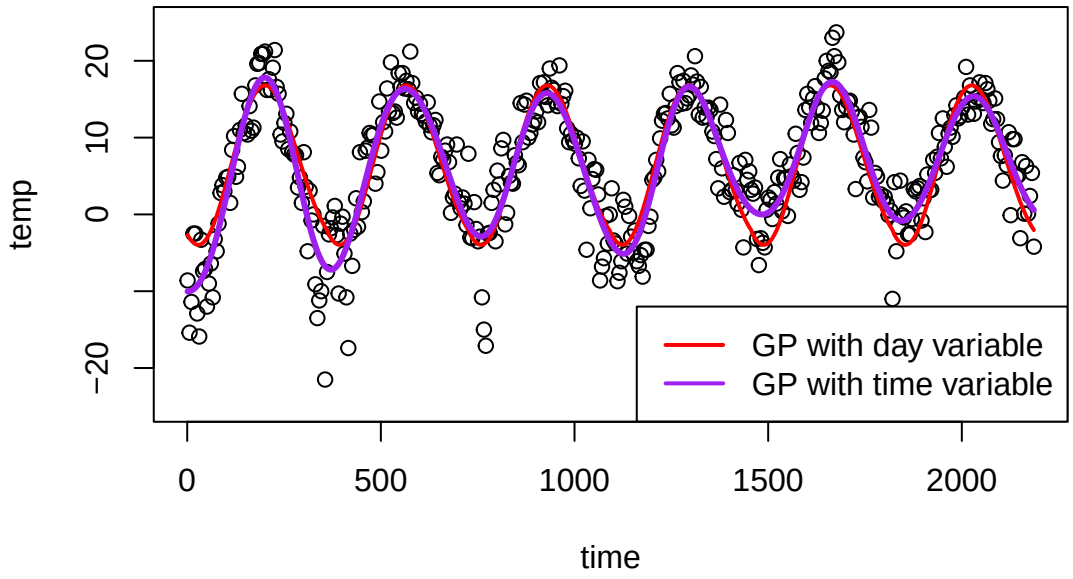
The graph shows that the model with the time variable fits better, particularly at the beginning. However, using the time variable is more computationally expensive compared to the day variable. The advantage of using time variable is its ability to detect long-term trend patterns, which the day variable cannot capture.

```
day_df <- df_subsample$day

polyFit <- lm(temp_df ~ day_df + I(day_df^2))
sigmaNoise_24 = sd(polyFit$residuals)
cat("Residual variance:", sigmaNoise_24^2)
```

```
## Residual variance: 23.53375
```

```
GPfit_24 <- gausspr(day_df, temp_df,
                    kernel = squared_exp,
                    kpar = list(sigmaf = 20, ell = 100),
                    var = sigmaNoise_24^2,
                    scaled=FALSE, variance.model = TRUE)
```

```r
meanPred_24 <- predict(GPfit_24, day_df) # Predicting the training data. To plot the fit.

plot(time_df, temp_df, xlab = "time", ylab = "temp", ylim = c(-25,25))
lines(time_df, meanPred_24, col="red", lwd = 2)
lines(time_df, meanPred_22, col="purple", lwd = 3)
legend("bottomright",
       legend = c("GP with day variable", "GP with time variable"),
       col = c("red", "purple"), lty = 1, lwd = 2)
```



(5)

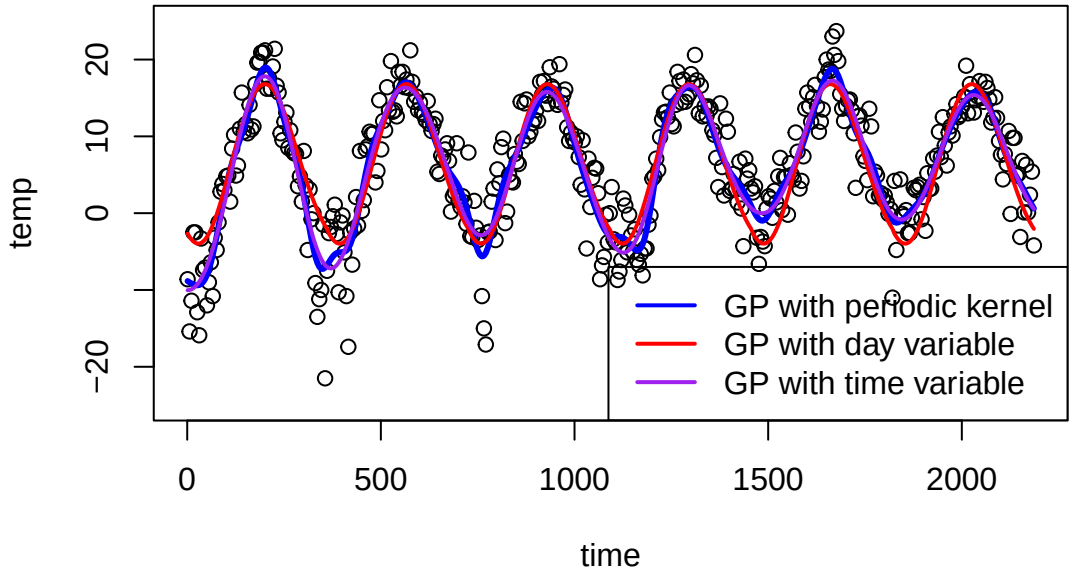
Using periodic kernel is useful capture periodic behavior and long-term trends over time.We can say that ℓ_1 controls the smoothness of day-to-day variations in one year whereas ℓ_2 controls the smoothness of the long-term trend. We set $\ell_1 = 1$, this will cause the model to be more sensitive to small changes in one year. Also setting $\ell_2 = 100$ will capture the seasonal pattern more in the model since temperatures on the same day in different years will be highly correlated.

When we look at the blue line below, it can be observed that the model can adapt to small changes more than using time variable (purple line) even though we used periodic kernel. Furthermore it generalizes less compared to day variable (red line) as well

```r
squared_exp_periodic <- function(sigmaf, ell1, ell2, d)
{
  rval <- function(x, y = NULL) {
    r = sqrt(crossprod(x-y));
    return(sigmaf^2 * exp(-2* ((sin(pi*r/d))^2) / ell1^2) * exp(-(1/2) * r^2/ell2^2))
  }
  class(rval) <- "kernel"
  return(rval)
}
```

```r
GPfit_25 <- gausspr(time_df, temp_df,
                    kernel = squared_exp_periodic,
                    kpar = list(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365),
                    var = sigmaNoise_22^2, # this is the noise from the model 2.2
                    scaled=FALSE, variance.model = TRUE)
```

```r
meanPred_25 <- predict(GPfit_25, time_df) # Predicting the training data. To plot the fit.

plot(time_df, temp_df, xlab = "time", ylab = "temp", ylim = c(-25,25))
lines(time_df, meanPred_25, col="blue", lwd = 3)
lines(time_df, meanPred_24, col="red", lwd = 2)
lines(time_df, meanPred_22, col="purple", lwd = 2)
legend("bottomright",
       legend = c("GP with periodic kernel","GP with day variable", "GP with time variable"),
       col = c("blue","red", "purple"), lty = 1, lwd = 2)
```



Question 3

```r
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])
head(data)
```

```
##   varWave skewWave kurtWave entropyWave fraud
## 1 3.62160   8.6661  -2.8073    -0.44699     0
## 2 4.54590   8.1674  -2.4586    -1.46210     0
## 3 3.86600  -2.6383   1.9242     0.10645     0
## 4 3.45660   9.5228  -4.0112    -3.59440     0
## 5 0.32924  -4.4552   4.5718    -0.98880     0
## 6 4.36840   9.6718  -3.9606    -3.16250     0
```

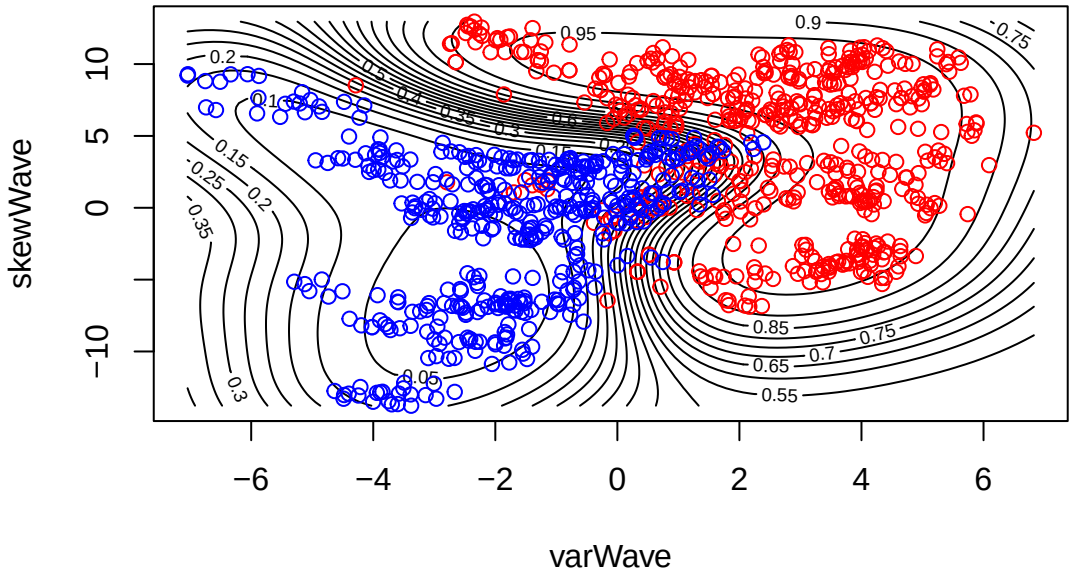
```r
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)

data_train <- data[SelectTraining,]
data_test <- data[-SelectTraining,]
```

(1)

```
# The model
GPfit_31 <- gausspr(fraud ~  varWave + skewWave, data=data_train)
```

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

```
# getting class probabilities, the code below is from KernLabDemo.R
probPreds <- predict(GPfit_31, data_train[,1:2], type="probabilities")
x1 <- seq(min(data_train[,1]),max(data_train[,1]),length=100)
x2 <- seq(min(data_train[,2]),max(data_train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$X), c(gridPoints$Y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(data_train)[1:2]
probPreds <- predict(GPfit_31, gridPoints, type="probabilities")

contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave")
points(data_train[data_train[,5]== 0, 1], data_train[data_train[,5]==0, 2],col="red")
points(data_train[data_train[,5]== 1, 1], data_train[data_train[,5]==1, 2],col="blue")
```




```
# predict on the training set
preds_train_31 <- predict(GPfit_31, data_train[,1:2])
conf_31 <- table(preds_train_31, data_train[,5]) # confusion matrix

cat("Confusion matrix of training data:\n")
```

```
## Confusion matrix of training data:
```

```
conf_31
```

```
##
## preds_train_31   0   1
##              0 503  18
##              1  41 438
```

```r
acc_31 <- sum(diag(conf_31)) / sum(conf_31)
cat("Accuracy of training data:", acc_31)
```

```
## Accuracy of training data: 0.941
```

(2)

```r
# predict on the test set
preds_test_32 <- predict(GPfit_31, data_test[,1:2])
conf_32 <- table(preds_test_32, data_test[,5]) # confusion matrix

cat("Confusion matrix of test data:\n")
```

```
## Confusion matrix of test data:
```

```r
conf_32
```

```
##
## preds_test_32   0   1
##             0 199   9
##             1  19 145
```

```r
acc_32 <- sum(diag(conf_32)) / sum(conf_32)
cat("Accuracy of test data:", acc_32)
```

```
## Accuracy of test data: 0.9247312
```

(3)

Accuracy is higher on the model with all variables compared to model with only 2 variables. Even though this model is more complex, it classifies better since it has more information.

```r
# The model with all variables
GPfit_33 <- gausspr(fraud ~ varWave + skewWave +  kurtWave + entropyWave, data=data_train)
```

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

```r
# predict on the test set
preds_test_33 <- predict(GPfit_33, data_test[,1:4])
conf_33 <- table(preds_test_33, data_test[,5]) # confusion matrix

cat("Confusion matrix of test data:\n")
```

```
## Confusion matrix of test data:
```

```r
conf_33
```

```
##
## preds_test_33   0   1
##             0 216   0
##             1   2 154
```

```r
acc_33 <- sum(diag(conf_33)) / sum(conf_33)
cat("Accuracy of test data:", acc_33)
```

```
## Accuracy of test data: 0.9946237
```