Lab 4: Gaussian Processes

Ravinder Reddy Atla

10/12/2021

2.1 Implementing GP Regression

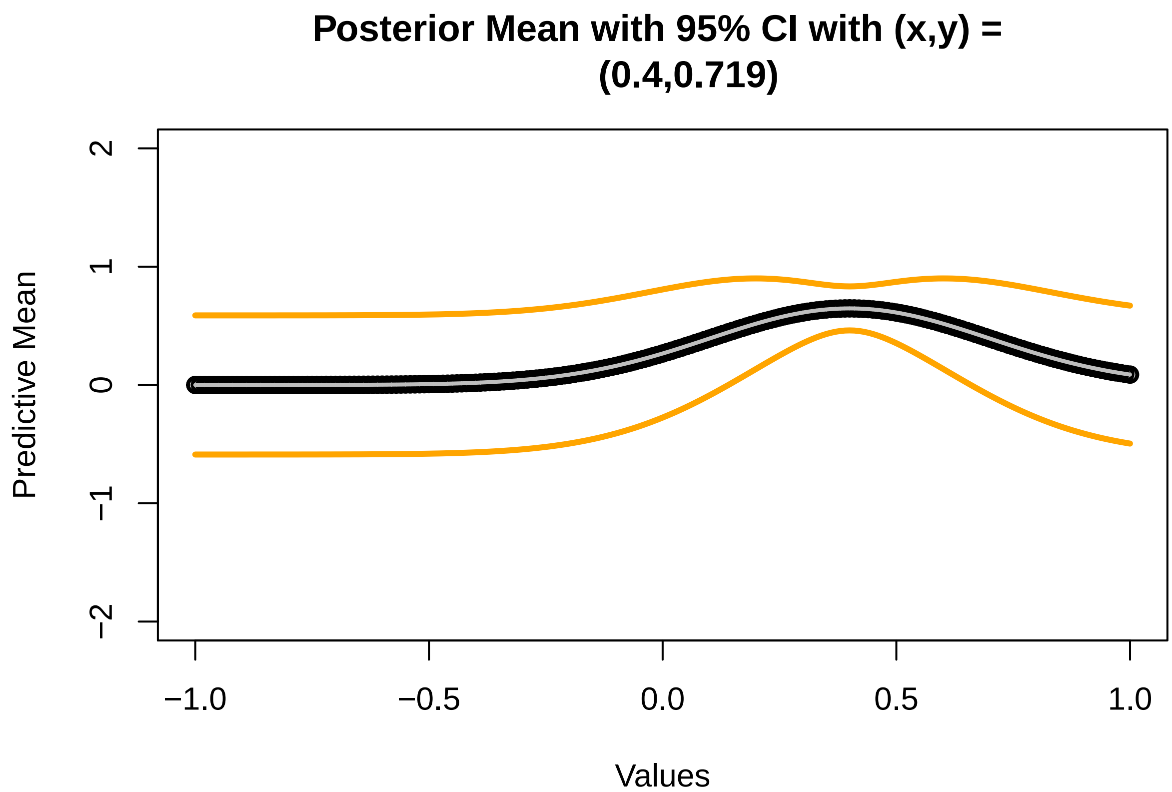
(1)

```
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, l = 0.3){
  n <- length(X)
  K <- SquaredExpKernel(X, X, l)
  L <- t(chol(K + (sigmaNoise^2 * diag(n))))

  alpha <- solve(t(L), solve(L, y))
  KStar <- SquaredExpKernel(X, XStar, l)
  f_pred_mean <- t(KStar) %*% alpha

  v <- solve(L, KStar)
  f_pred_var <- diag(SquaredExpKernel(XStar, XStar, l) - (t(v)%*%v))

  return(list('pred_mean' = f_pred_mean, 'pred_var' = f_pred_var))
}
```

(2)

```
x <- c(0.4)
y <- c(0.719)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star, sigmaNoise = sigma_n)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
```

```
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 2, main = 'Posterior Mean with 95% CI with (x,y) =
     (0.4,0.719)')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 2, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
```

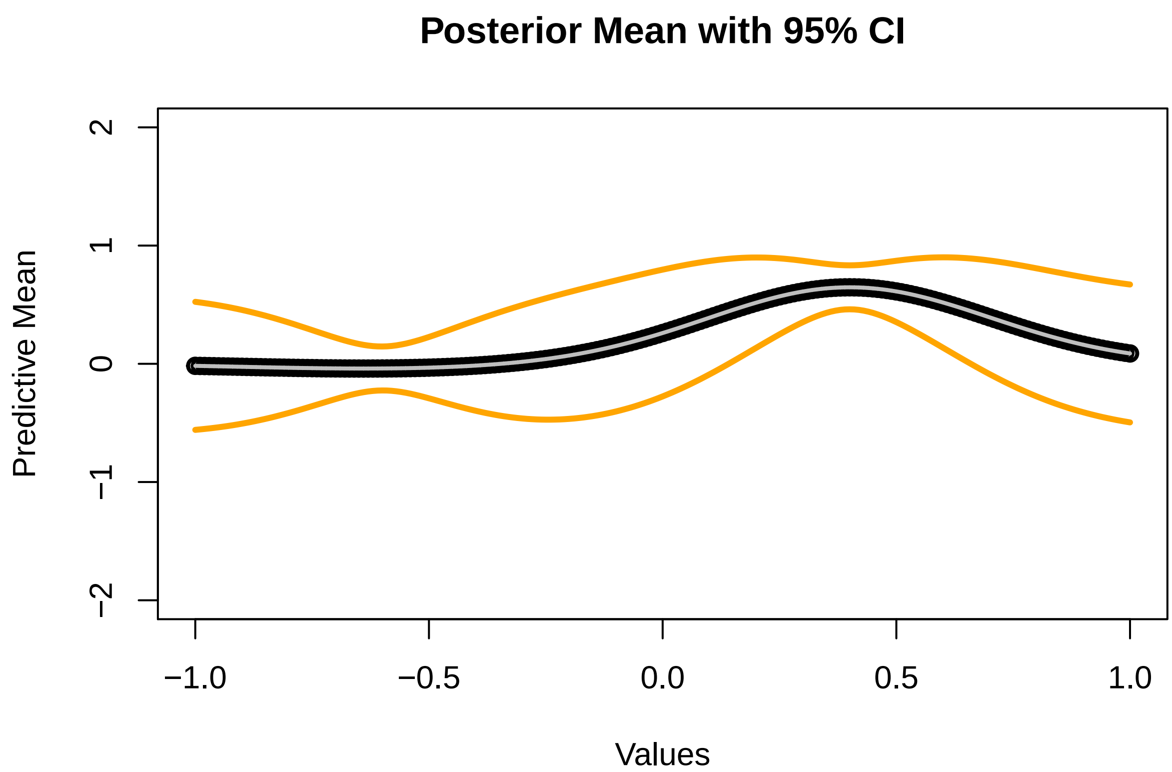


###

(3)

x	0.400	-0.600
y	0.719	-0.044

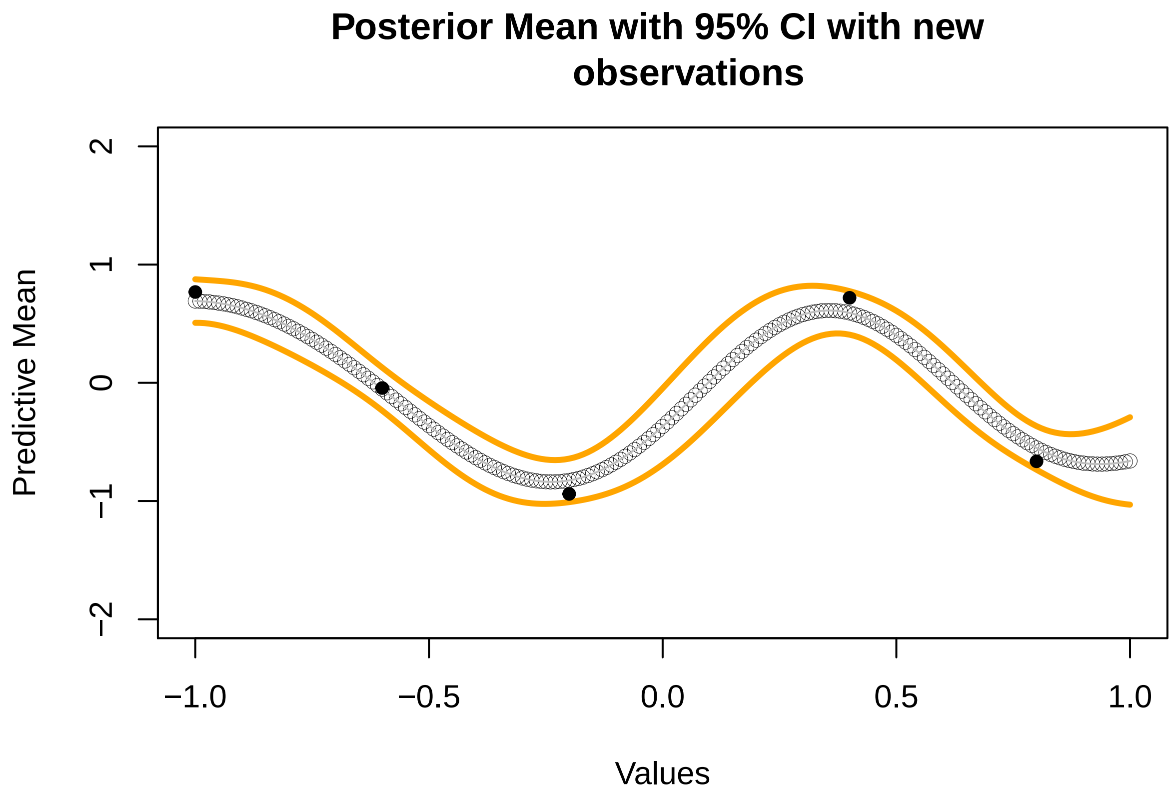
```
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star, sigmaNoise = sigma_n)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 2, main = 'Posterior Mean with 95% CI')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 2, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
```



(4)

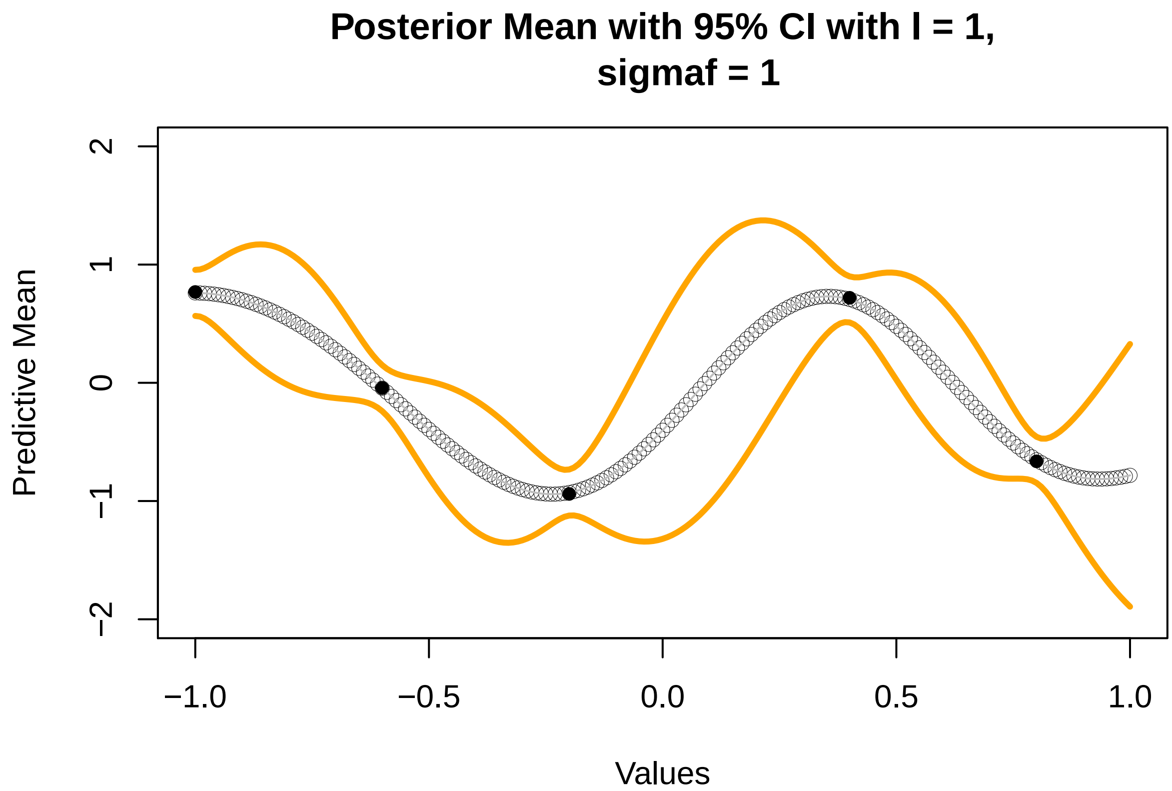
x	-1.000	-0.600	-0.20	0.400	0.800
y	0.768	-0.044	-0.94	0.719	-0.664

```
x <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star, sigmaNoise = sigma_n)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 0.2, main = 'Posterior Mean with 95% CI with new
     observations')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 1, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
points(x = x, y = y, pch = 20, lwd = 2)
```



(5)

```
x <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star,
                             sigmaNoise = sigma_n, l = 1)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 0.2, main = 'Posterior Mean with 95% CI with l = 1,
     sigmaf = 1')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 1, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
points(x = x, y = y, pch = 20, lwd = 2)
```



2.2 GP Regression with kernlab

```
library(kernlab)
```

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

time <- 1:nrow(temp_data)
day <- rep(1:365, 6)

time <- time[seq(1, length(time), 5)]
day <- day[seq(1, length(day), 5)]
```

(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)
#}
x1 = 1
x2 = 2

SquaredExpKernel <- function(sigmaF = 1, l = 0.3){
  SqExpKernel <- function(x1, x2){
```

```r
    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)
  }
  class(SqExpKernel) <- 'kernel'
  return(SqExpKernel)
}

sek_out <- SquaredExpKernel()
print(sek_out)
```

```
## function(x1, x2){
##     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)
##   }
## <bytecode: 0x7fcb1ea70118>
## <environment: 0x7fcb1f520a50>
## attr(,"class")
## [1] "kernel"
```

```r
X <- t(matrix(data = c(1,3,4),1))
X_star <- t(matrix(data = c(2,3,4),1))
cov_matrix <- kernelMatrix(sek_out, X, X_star)
print(cov_matrix)
```

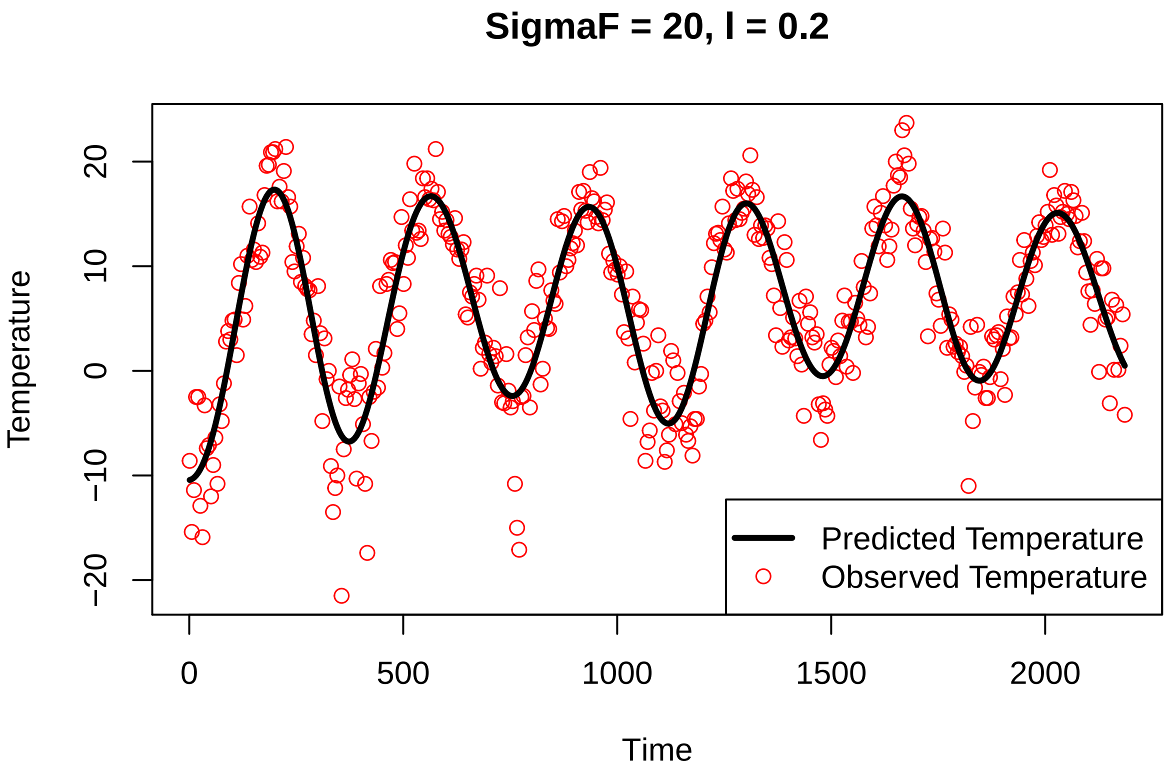
```
## An object of class "kernelMatrix"
##              [,1]         [,2]        [,3]
## [1,] 3.865920e-03 2.233631e-10 1.92875e-22
## [2,] 3.865920e-03 1.000000e+00 3.86592e-03
## [3,] 2.233631e-10 3.865920e-03 1.00000e+00
```

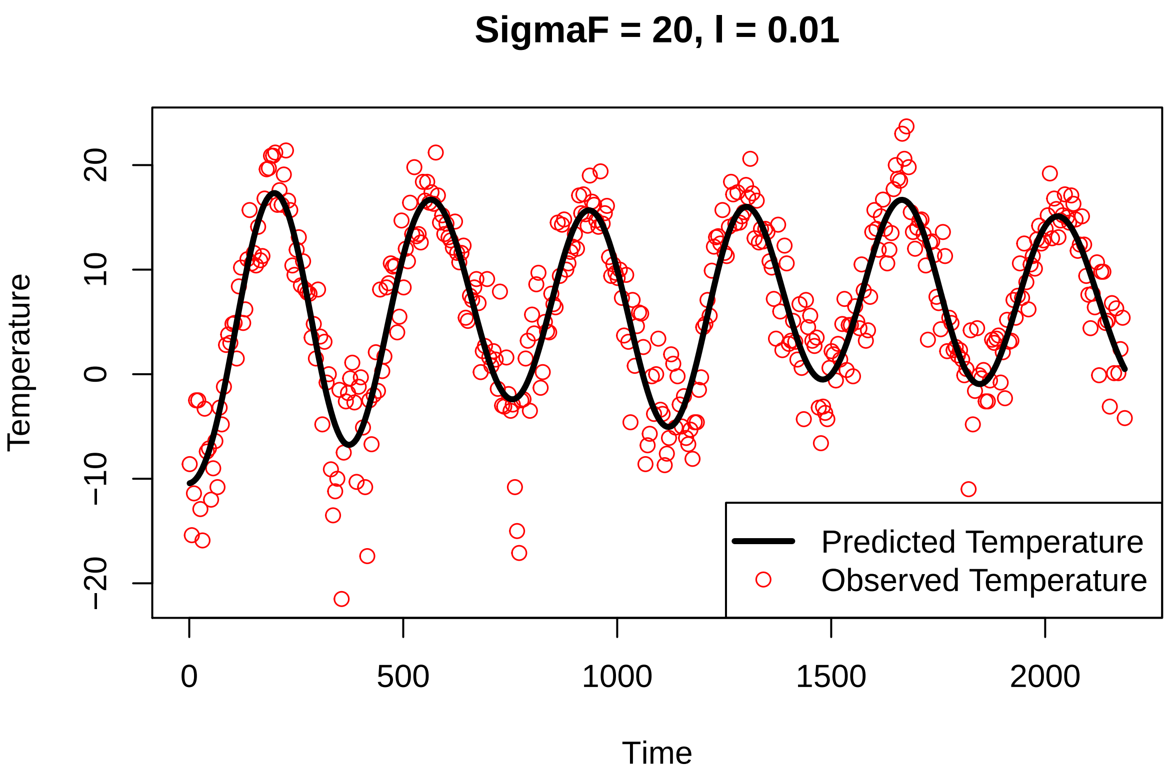
(2)

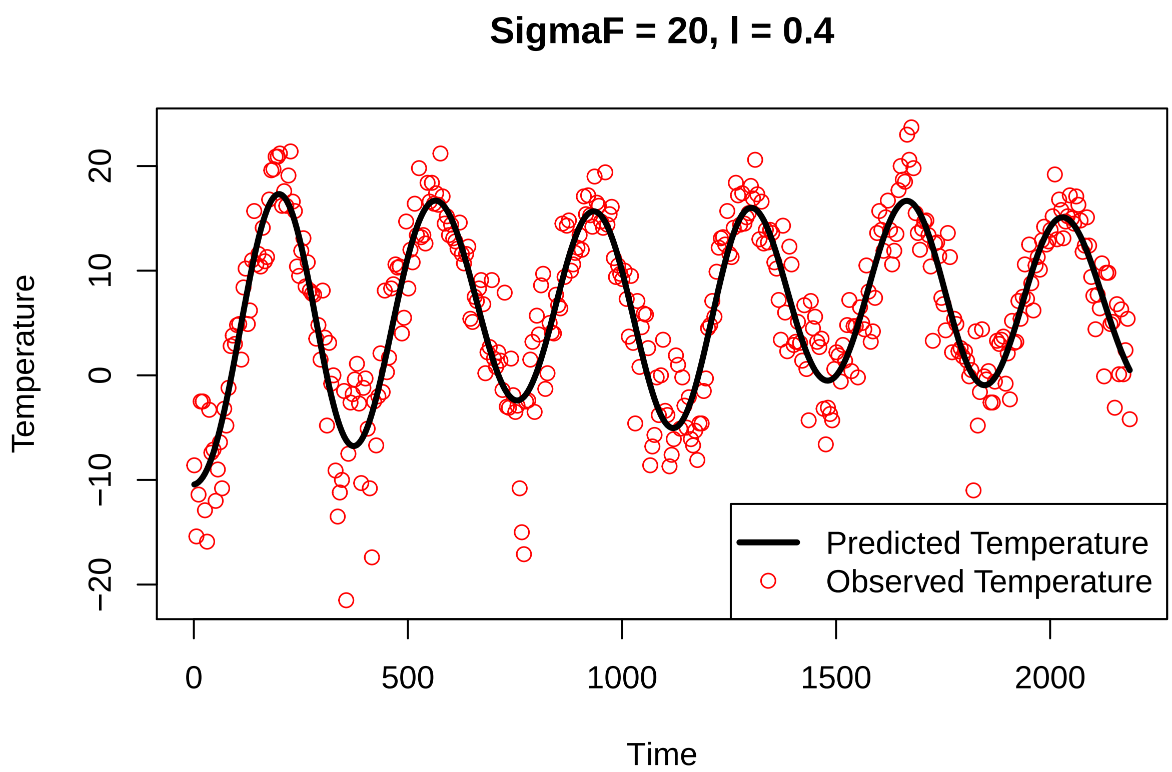
```r
temp_data_filter <- temp_data[seq(1, nrow(temp_data), 5), ]

regression_model <- lm(temp_data_filter$temp ~ time + time^2)
var_n <- var(regression_model$residuals)
```

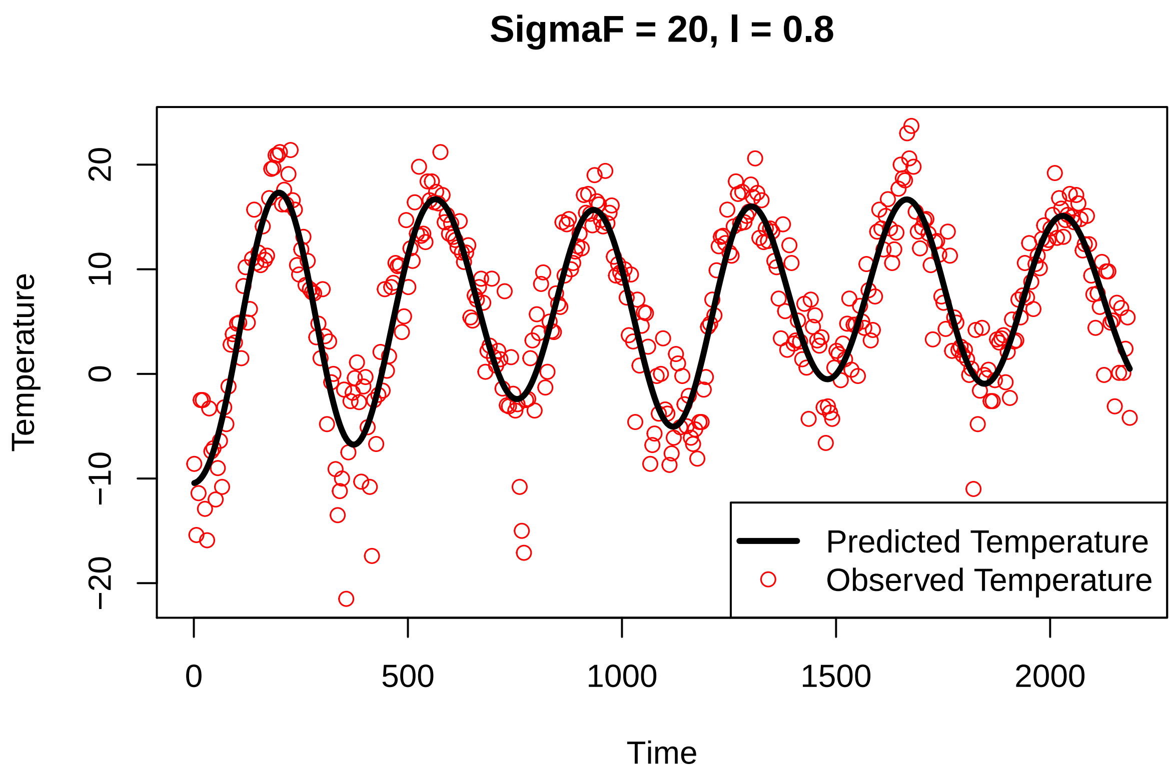
```r
l <- 0.2
sigma_f <- 20

gp_model <- gausspr(x = time, y = temp_data_filter$temp,
                kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                var = var_n)
```

```r
posterior_mean <- predict(gp_model, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.2')
lines(time, posterior_mean, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")
```







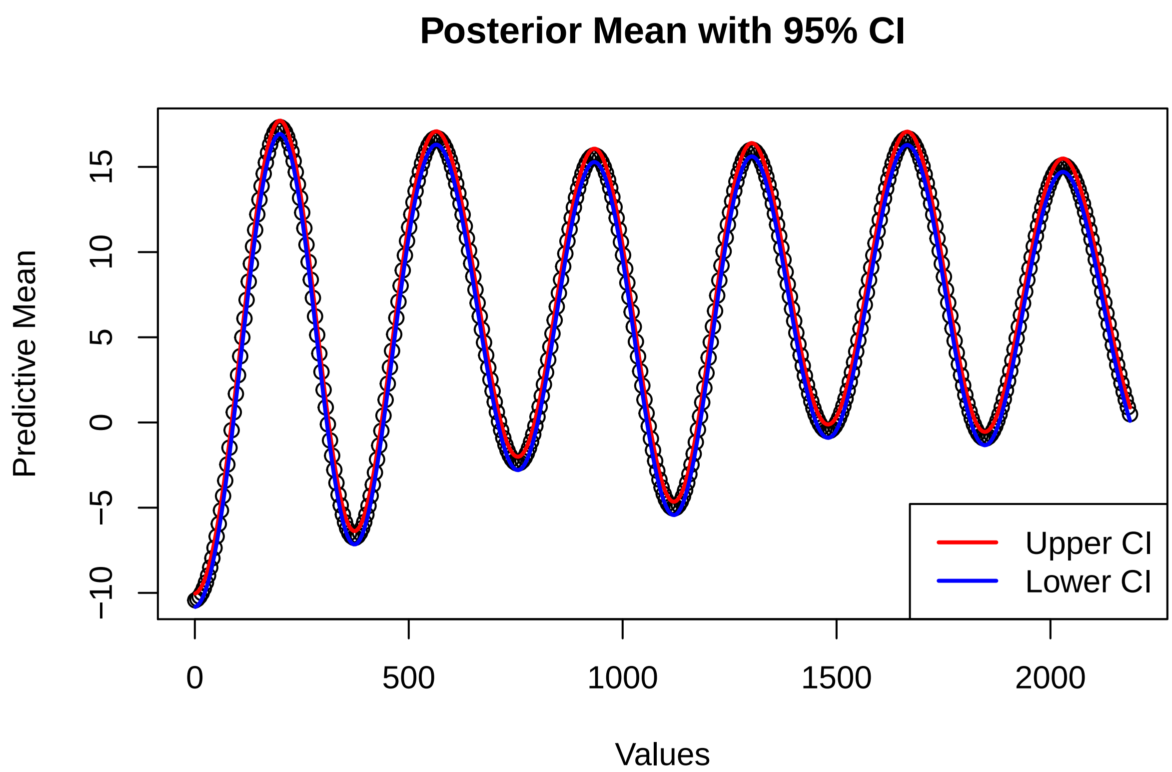


(3)

```
SquareExpKernel <- function(x1, x2, sigmaF = 20, l = 0.2){
  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)
}
posteriorGPVariance <- function(X, y, XStar, varNoise, l = 0.2){
  n <- length(X)
  K <- SquareExpKernel(X, X, l)
  L <- t(chol(K + (varNoise * diag(n))))

  alpha <- solve(t(L), solve(L, y))
  KStar <- SquareExpKernel(X, XStar, l)
  #f_pred_mean <- t(KStar) %*% alpha

  v <- solve(L, KStar)
  f_pred_var <- diag(SquareExpKernel(XStar, XStar, l) - (t(v)%*%v))

  return(f_pred_var)
}
```

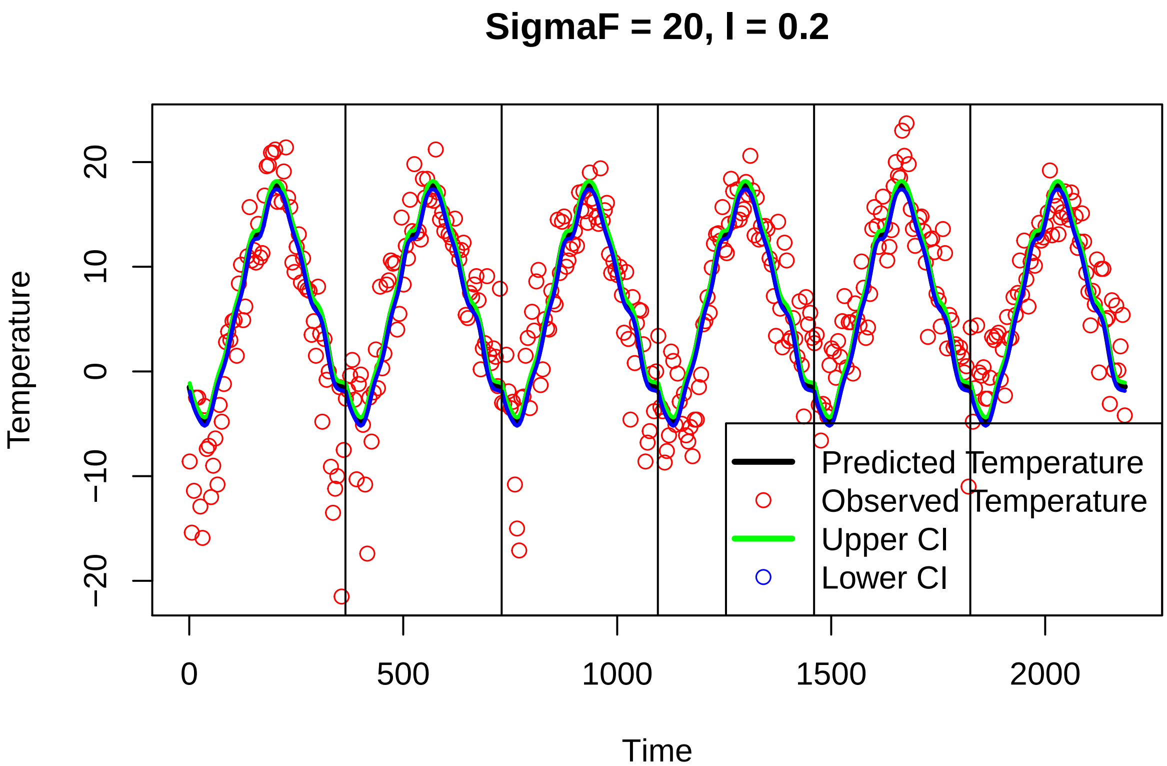
```
posterior_var <- posteriorGPVariance(X = time, y = temp_data_filter$temp,
                                     XStar = time, varNoise = var_n)

CI_hi <- posterior_mean + 1.96 * sqrt(posterior_var)
CI_lo <- posterior_mean - 1.96 * sqrt(posterior_var)

plot(x = time, y = posterior_mean,col = 'black',
     xlab = 'Values', ylab = 'Predictive Mean',
     lwd = 1, main = 'Posterior Mean with 95% CI')
#lines(x = x_star, y = posterior_out$pred_mean, lwd = 2, col = 'grey')
lines(x = time, y = CI_hi, lwd = 2, col = 'red')
lines(x = time, y = CI_lo, lwd = 2, col = 'blue')
legend("bottomright", legend = c("Upper CI", "Lower CI"), col=c("red","blue"), lty=c(1,1), lwd=c(2, 2))
```



(4)

```r
regression_model_day <- lm(temp_data_filter$temp ~ day + day^2)
var_n_day <- var(regression_model_day$residuals)

l <- 0.2
sigma_f <- 20

gp_model_day <- gausspr(x = day, y = temp_data_filter$temp,
                  kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                  var = var_n_day)

posterior_mean_day <- predict(gp_model_day, day)

posterior_var_day <- posteriorGPVariance(X = day, y = temp_data_filter$temp,
                                         XStar = day, varNoise = var_n_day)

CI_hi <- posterior_mean_day + 1.96 * sqrt(posterior_var_day)
CI_lo <- posterior_mean_day - 1.96 * sqrt(posterior_var_day)

plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.2')
lines(time, posterior_mean_day, lwd = 3, col = 'black')
lines(x = time, y = CI_hi, lwd = 2, col = 'green')
lines(x = time, y = CI_lo, lwd = 2, col = 'blue')
abline(v = c(365, 365*2, 365*3, 365*4, 365*5))
legend("bottomright", legend = c("Predicted Temperature",
                                 "Observed Temperature",
                                 'Upper CI','Lower CI'),
```

```
       col=c("black","red", 'green', 'blue'),
       lty=c(1,NA), pch=c(NA,1), lwd=c(3,0.8))
```

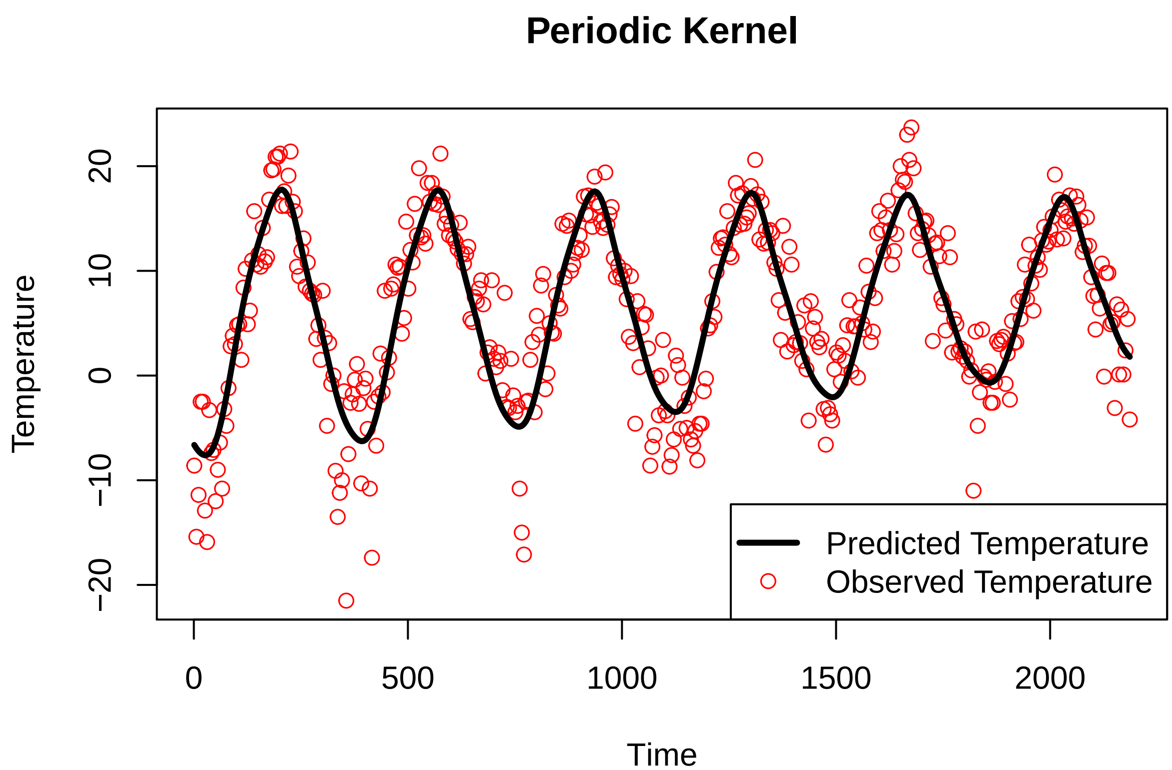


When the above plot using the day is compared with the one with time, the predictions obtained using day are following a similar distribution after each year. Whereas, predictions with the time seems to be changing a bit over the time.

(5)

```
PeriodicKernel <- function(sigmaF = 20, l1 = 1, l2 = 10, d = 365/sd(time)){
  PeriodKernel <- function(x1, x2){
    K <- sigmaF^2 * exp(-(2*(sin(pi*abs(x1 - x2)/d)^2))/l1^2) *
      exp((-0.5*(abs(x1-x2))^2)/l2^2)
    return(K)
  }
  class(PeriodKernel) <- 'kernel'
  return(PeriodKernel)
}
```

```
gp_model_period <- gausspr(x = time, y = temp_data_filter$temp,
                  kernel = PeriodicKernel(),
                  var = var_n)

posterior_mean_period <- predict(gp_model_period, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'Periodic Kernel')
lines(time, posterior_mean_period, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")
```



It looks like all the plots obtained above (excluding the experimented ones) are mostly similar. The periodic kernel and time based predictions are almost similar except at the start where periodic kernel starts above -10 unlike the other one. Whereas, the predictions obtained using the day variable with Squared exponential kernel are periodic over each year and also the least best of the three fitted models. It is quite hard to decide b/w 1st and 3rd model.

3. GP Classification with Kernlab

```
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(123)
index <- sample(1:nrow(data), size = 1000)

train_data <- data[index, ]
test_data <- data[-index, ]
```

```
library(AtmRay)
GPfit <- gausspr(fraud ~  varWave + skewWave, data = train_data)
```

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

```
GPfit
```

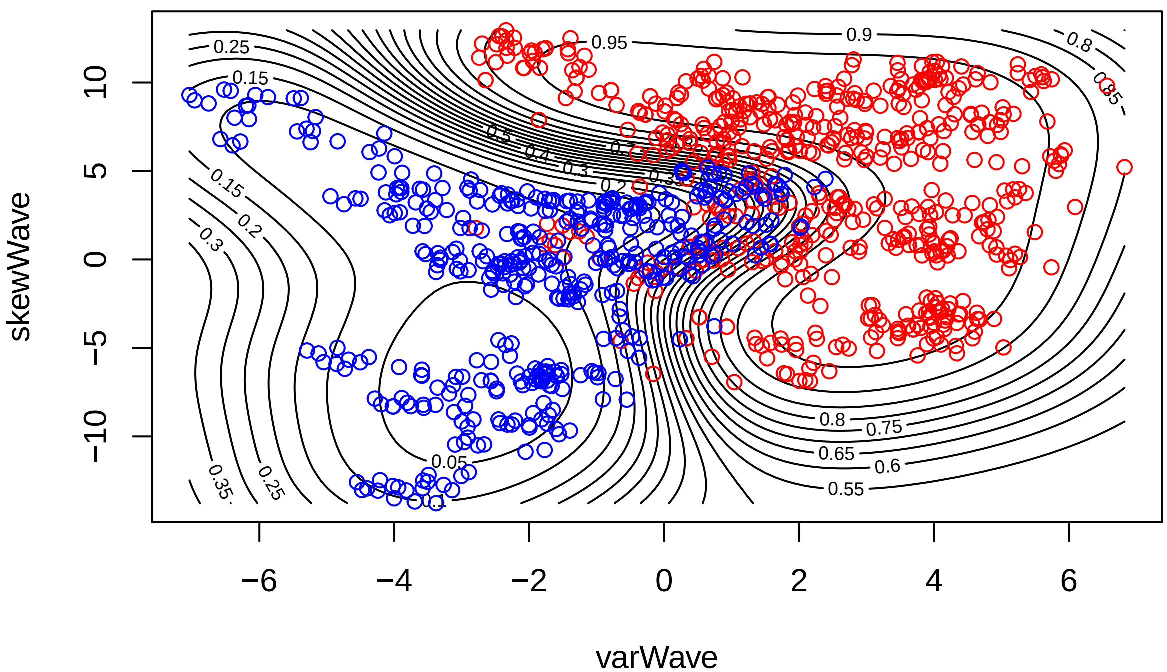
```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.39862659448122
##
## Number of training instances learned : 1000
```

```
## Train error : 0.077
```

```r
# predict on the training set
pred_values <- predict(GPfit,train_data[,1:2])
table(predict(GPfit,train_data[,1:2]), train_data[,5]) # confusion matrix
```

```
##
##       0   1
##   0 521  32
##   1  45 402
```

```r
# class probabilities
probPreds <- predict(GPfit, train_data[,1:2], type="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(train_data)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20,
        xlab = "varWave", ylab = "skewWave")
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")
```



(2) Predictions of Test Data

```r
conf_matrix <- table(predict(GPfit, test_data[,1:2]), test_data[,5])
conf_matrix
```

```
##
```

```
##       0   1
##   0 184   7
##   1  12 169
```

```r
accuracy <- sum(diag(conf_matrix))/sum(conf_matrix)
accuracy
```

```
## [1] 0.9489247
```

(3) Using All Covariates

```r
GPfit_comp <- gausspr(fraud ~ ., data = train_data)
```

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

```r
GPfit_comp
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.404435966489743
##
## Number of training instances learned : 1000
## Train error : 0
```

```r
# predict on the training set
conf_matrix_train_comp <- table(predict(GPfit_comp,train_data[,1:4]), train_data[,5]) # confusion matri
valid_accuracy_train_comp <- sum(diag(conf_matrix_train_comp))/sum(conf_matrix_train_comp)
print(valid_accuracy_train_comp)
```

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

```r
conf_matrix_comp <- table(predict(GPfit_comp, test_data[,1:4]), test_data[,5])
print(conf_matrix_comp)
```

```
##
##       0   1
##   0 195   0
##   1   1 176
```

```r
valid_accuracy_comp <- sum(diag(conf_matrix_comp))/sum(conf_matrix_comp)
print(valid_accuracy_comp)
```

```
## [1] 0.9973118
```

As observed from the results obtained above, the train and test accuracy of the model using two covariates (varWave and skewWave) is lower than that of the model's accuracy using all the covariates which is obvious to conclude as the second model is using more data. But, the difference between their accuracies very less indicating that fraud can be mostly detected using the two variables (varWave and skewWave).

Appendix:

```r
knitr::opts_chunk$set(echo = TRUE)

SquaredExpKernel <- function(x1, x2, sigmaF = 1, l = 0.3){
  n1 <- length(x1)
```

```r
  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, l = 0.3){
  n <- length(X)
  K <- SquaredExpKernel(X, X, l)
  L <- t(chol(K + (sigmaNoise^2 * diag(n))))

  alpha <- solve(t(L), solve(L, y))
  KStar <- SquaredExpKernel(X, XStar, l)
  f_pred_mean <- t(KStar) %*% alpha

  v <- solve(L, KStar)
  f_pred_var <- diag(SquaredExpKernel(XStar, XStar, l) - (t(v)%*%v))

  return(list('pred_mean' = f_pred_mean, 'pred_var' = f_pred_var))
}

x <- c(0.4)
y <- c(0.719)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star, sigmaNoise = sigma_n)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 2, main = 'Posterior Mean with 95% CI with (x,y) =
     (0.4,0.719)')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 2, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')

x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
knitr::kable(rbind(x,y))
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star, sigmaNoise = sigma_n)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)
```

```r
plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 2, main = 'Posterior Mean with 95% CI')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 2, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
x <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
knitr::kable(rbind(x,y))
x <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star, sigmaNoise = sigma_n)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 0.2, main = 'Posterior Mean with 95% CI with new
     observations')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 1, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
points(x = x, y = y, pch = 20, lwd = 2)

x <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
sigma_n <- 0.1
x_star <- seq(-1, 1, 0.01)

posterior_out <- posteriorGP(X = x, y = y, XStar = x_star,
                             sigmaNoise = sigma_n, l = 1)

CI_hi <- posterior_out$pred_mean + 1.96 * sqrt(posterior_out$pred_var)
CI_lo <- posterior_out$pred_mean - 1.96 * sqrt(posterior_out$pred_var)

plot(x = x_star, y = posterior_out$pred_mean,
     xlab = 'Values', ylab = 'Predictive Mean',
     ylim = c(-2,2), lwd = 0.2, main = 'Posterior Mean with 95% CI with l = 1,
     sigmaf = 1')
lines(x = x_star, y = posterior_out$pred_mean, lwd = 1, col = 'grey')
lines(x = x_star, y = CI_hi, lwd = 3, col = 'orange')
lines(x = x_star, y = CI_lo, lwd = 3, col = 'orange')
points(x = x, y = y, pch = 20, lwd = 2)
library(kernlab)
temp_data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTull

time <- 1:nrow(temp_data)
day <- rep(1:365, 6)
```

```r
time <- time[seq(1, length(time), 5)]
day <- day[seq(1, length(day), 5)]
#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)
#}
x1 = 1
x2 = 2

SquaredExpKernel <- function(sigmaF = 1, l = 0.3){
  SqExpKernel <- function(x1, x2){
    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)
  }
  class(SqExpKernel) <- 'kernel'
  return(SqExpKernel)
}

sek_out <- SquaredExpKernel()
print(sek_out)

X <- t(matrix(data = c(1,3,4),1))
X_star <- t(matrix(data = c(2,3,4),1))
cov_matrix <- kernelMatrix(sek_out, X, X_star)
print(cov_matrix)
temp_data_filter <- temp_data[seq(1, nrow(temp_data), 5), ]

regression_model <- lm(temp_data_filter$temp ~ time + time^2)
var_n <- var(regression_model$residuals)
l <- 0.2
sigma_f <- 20

gp_model <- gausspr(x = time, y = temp_data_filter$temp,
                 kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                 var = var_n)

posterior_mean <- predict(gp_model, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.2')
lines(time, posterior_mean, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")
l <- 0.01
sigma_f <- 20
```

```r
gp_model <- gausspr(x = time, y = temp_data_filter$temp,
                  kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                  var = var_n)

posterior_mean_sam <- predict(gp_model, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.01')
lines(time, posterior_mean, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")
l <- 0.4
sigma_f <- 20

gp_model <- gausspr(x = time, y = temp_data_filter$temp,
                  kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                  var = var_n)

posterior_mean_sam <- predict(gp_model, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.4')
lines(time, posterior_mean, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")
l <- 0.8
sigma_f <- 20

gp_model <- gausspr(x = time, y = temp_data_filter$temp,
                  kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                  var = var_n)

posterior_mean_sam <- predict(gp_model, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.8')
lines(time, posterior_mean, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")
SquareExpKernel <- function(x1, x2, sigmaF = 20, l = 0.2){
  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)
}
posteriorGPVariance <- function(X, y, XStar, varNoise, l = 0.2){
  n <- length(X)
  K <- SquareExpKernel(X, X, l)
  L <- t(chol(K + (varNoise * diag(n))))

  alpha <- solve(t(L), solve(L, y))
  KStar <- SquareExpKernel(X, XStar, l)
  #f_pred_mean <- t(KStar) %*% alpha

  v <- solve(L, KStar)
  f_pred_var <- diag(SquareExpKernel(XStar, XStar, l) - (t(v)%*%v))
```

```r
  return(f_pred_var)
}
posterior_var <- posteriorGPVariance(X = time, y = temp_data_filter$temp,
                                     XStar = time, varNoise = var_n)

CI_hi <- posterior_mean + 1.96 * sqrt(posterior_var)
CI_lo <- posterior_mean - 1.96 * sqrt(posterior_var)

plot(x = time, y = posterior_mean,col = 'black',
     xlab = 'Values', ylab = 'Predictive Mean',
     lwd = 1, main = 'Posterior Mean with 95% CI')
#lines(x = x_star, y = posterior_out$pred_mean, lwd = 2, col = 'grey')
lines(x = time, y = CI_hi, lwd = 2, col = 'red')
lines(x = time, y = CI_lo, lwd = 2, col = 'blue')
legend("bottomright", legend = c("Upper CI", "Lower CI"), col=c("red","blue"), lty=c(1,1), lwd=c(2, 2))
regression_model_day <- lm(temp_data_filter$temp ~ day + day^2)
var_n_day <- var(regression_model_day$residuals)

l <- 0.2
sigma_f <- 20

gp_model_day <- gausspr(x = day, y = temp_data_filter$temp,
                 kernel = SquaredExpKernel(sigmaF = sigma_f, l = l),
                 var = var_n_day)

posterior_mean_day <- predict(gp_model_day, day)

posterior_var_day <- posteriorGPVariance(X = day, y = temp_data_filter$temp,
                                     XStar = day, varNoise = var_n_day)

CI_hi <- posterior_mean_day + 1.96 * sqrt(posterior_var_day)
CI_lo <- posterior_mean_day - 1.96 * sqrt(posterior_var_day)

plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'SigmaF = 20, l = 0.2')
lines(time, posterior_mean_day, lwd = 3, col = 'black')
lines(x = time, y = CI_hi, lwd = 2, col = 'green')
lines(x = time, y = CI_lo, lwd = 2, col = 'blue')
abline(v = c(365, 365*2, 365*3, 365*4, 365*5))
legend("bottomright", legend = c("Predicted Temperature",
                                 "Observed Temperature",
                                 'Upper CI','Lower CI'),
       col=c("black","red", 'green', 'blue'),
       lty=c(1,NA), pch=c(NA,1), lwd=c(3,0.8))

PeriodicKernel <- function(sigmaF = 20, l1 = 1, l2 = 10, d = 365/sd(time)){
  PeriodKernel <- function(x1, x2){
    K <- sigmaF^2 * exp(-(2*(sin(pi*abs(x1 - x2)/d)^2))/l1^2) *
      exp((-0.5*(abs(x1-x2))^2)/l2^2)
    return(K)
  }
  class(PeriodKernel) <- 'kernel'
  return(PeriodKernel)
```

```r
}

gp_model_period <- gausspr(x = time, y = temp_data_filter$temp,
                 kernel = PeriodicKernel(),
                 var = var_n)

posterior_mean_period <- predict(gp_model_period, time)
plot(time, temp_data_filter$temp, col = 'red', lwd = 0.8,
     xlab = 'Time', ylab = 'Temperature', main = 'Periodic Kernel')
lines(time, posterior_mean_period, lwd = 3, col = 'black')
legend("bottomright", legend = c("Predicted Temperature", "Observed Temperature"), col=c("black","red")


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(123)
index <- sample(1:nrow(data), size = 1000)

train_data <- data[index, ]
test_data <- data[-index, ]
library(AtmRay)
GPfit <- gausspr(fraud ~  varWave + skewWave, data = train_data)
GPfit

# predict on the training set
pred_values <- predict(GPfit,train_data[,1:2])
table(predict(GPfit,train_data[,1:2]), train_data[,5]) # confusion matrix

# class probabilities
probPreds <- predict(GPfit, train_data[,1:2], type="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(train_data)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20,
        xlab = "varWave", ylab = "skewWave")
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")

conf_matrix <- table(predict(GPfit, test_data[,1:2]), test_data[,5])
conf_matrix
accuracy <- sum(diag(conf_matrix))/sum(conf_matrix)
accuracy
GPfit_comp <- gausspr(fraud ~ ., data = train_data)
GPfit_comp
```

```r
# predict on the training set
conf_matrix_train_comp <- table(predict(GPfit_comp,train_data[,1:4]), train_data[,5]) # confusion matri
valid_accuracy_train_comp <- sum(diag(conf_matrix_train_comp))/sum(conf_matrix_train_comp)
print(valid_accuracy_train_comp)

conf_matrix_comp <- table(predict(GPfit_comp, test_data[,1:4]), test_data[,5])
print(conf_matrix_comp)
valid_accuracy_comp <- sum(diag(conf_matrix_comp))/sum(conf_matrix_comp)
print(valid_accuracy_comp)
```