Advanced Machine Learning (732A96) Lab4

Anubhav Dikshit(anudi287)

16 October, 2019

Contents

Questions

Questions 1: Implementing GP Regression

This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \epsilon \quad \epsilon \sim N(0, \sigma_n^2) \; and \; f \sim GP(0, k(x, x'))$$

You must implement Algorithm 2.1 on page 19 of Rasmussen and Willams' book. The algorithm uses the Cholesky decomposition (chol in R) to attain numerical stability. Note that L in the algorithm is a lower triangular matrix, whereas the R function returns an upper triangular matrix. So, you need to transpose the output of the R function. In the algorithm, the notation A slash b means the vector x that solves the equation Ax = b. This is implemented in R with the help of the function solve.

1) Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it posteriorGP) should return a vector with the posterior mean and variance of f, both evaluated at a set of x-values (X_*). You can assume that the prior mean of f is zero for all x. The function should have the following inputs:

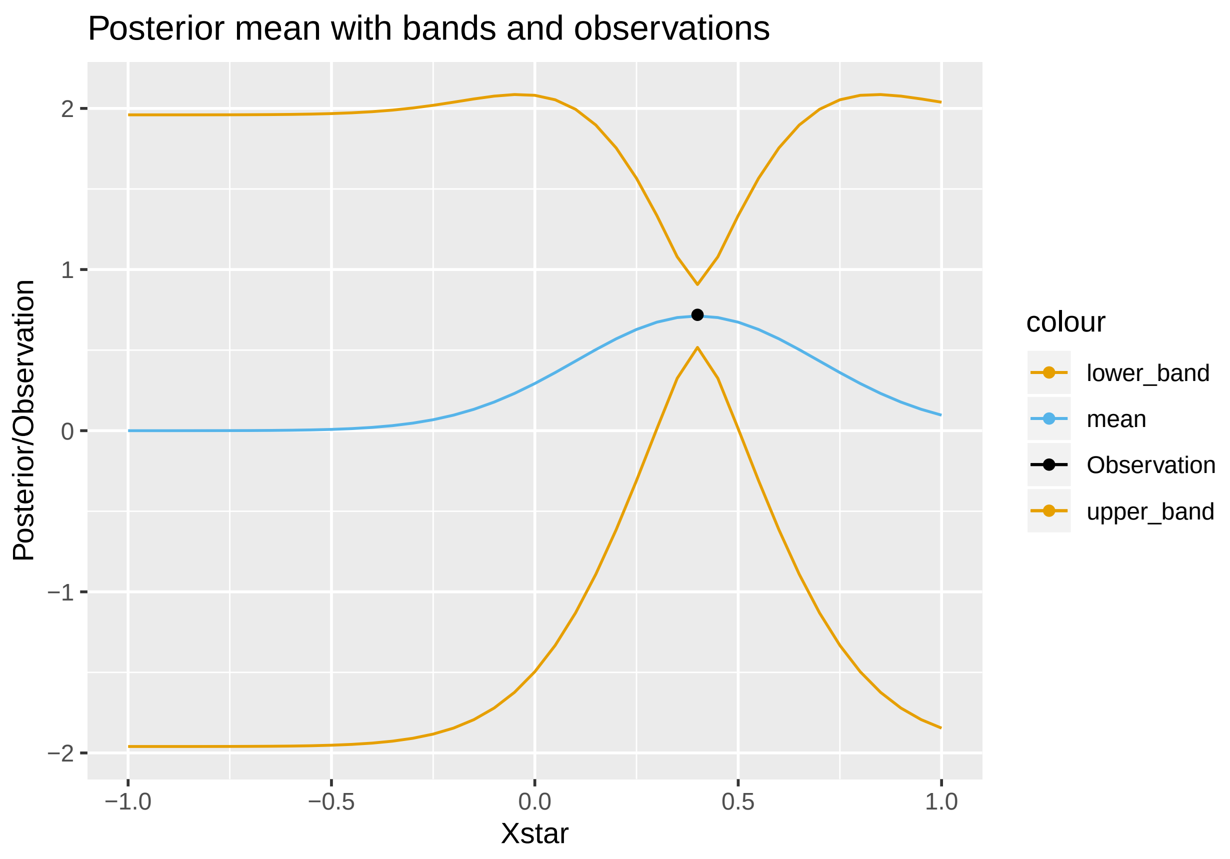
X: Vector of training inputs.
y: Vector of training targets/outputs.
XStar: Vector of inputs where the posterior distribution is evaluated, i.e. X_*
hyperParam: Vector with two elements, σ_f and l
sigmaNoise: Noise standard deviation σ_n.

```r
set.seed(111)

square_exp_kernel = function(x1, x2, sigmaF, l) {
  n1 = length(x1)
  n2 = length(x2)
  K = matrix(NA, n1, n2)
  for(i in 1:n2) {
    for(j in 1:n1) {
      K[j,i] = sigmaF^2 * exp(-0.5 * ( (x2[i] - x1[j])/l)^2 )
    }
  }
  return(K)
}

posteriorGP_square_exp_kernel <- function(x,y,XStar,hyperParam, sigmaNoise){
  sigmaF <- hyperParam[1]
  l <- hyperParam[2]
  n = length(x)
  L = t(chol(square_exp_kernel(x,x, sigmaF, l) + sigmaNoise^2 * diag(n)))
  KStar = square_exp_kernel(x, XStar, sigmaF, l)

  # Predictive Mean
  alpha = solve(t(L), solve(L, y))
  fStarBar = t(KStar) %*% alpha

  # Predictive Variance
  v = solve(L, KStar)
  V = square_exp_kernel(XStar, XStar,sigmaF, l) - t(v) %*% v
  return(data.frame(mean = fStarBar, variance = diag(V)))
}

Visualize = function(X, y, XStar, res) {
  mean = res$mean
  sd = sqrt(res$variance)
  plot(XStar, mean, type="l", ylim=c(-3, 4),
       xlab="x", ylab="y", main="Posterior Gaussian Process")
  lines(XStar, mean - 1.96 * sd, type="l", col="blue")
  lines(XStar, mean + 1.96 * sd, type="l", col="blue")
  points(X, y, pch=19)
  legend("topright",
         legend=c("posterior mean", "probability band", "observations"),
         col=c("black", "blue", "black"), lty=c(1,1,0), pch=c(-1,-1,19))
}


plot_gaussian <- function(x,y,Xstar,res){
mu <- res$mean
sd <- sqrt(res$variance)
df <- data.frame(XStar=XStar,
                 mu = mu,
                 upper_band = mu + 1.96 * sd,
                 lower_band = mu - 1.96 * sd)

df2 <- data.frame(x=x,y=y)

plot1 <- ggplot(data=df, aes(x=XStar)) +
  geom_line(aes(y=mu, color="mean")) +
  geom_line(aes(y=upper_band, color="upper_band")) +
  geom_line(aes(y=lower_band, color="lower_band")) +
  geom_point(data=df2, aes(x=x, y=y, color="Observation")) +
  ggtitle("Posterior mean with bands and observations") +
  ylab("Posterior/Observation") + xlab("Xstar") +
       scale_colour_manual(values = c("#E69F00", "#56B4E9", "#000000", "#E69F00"))

print(plot1)

}
```

2) Now, let the prior hyperparameters be $\sigma_f = 1$ and $l = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that $\sigma_n = 0.1$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 percent probability (pointwise) bands for f.

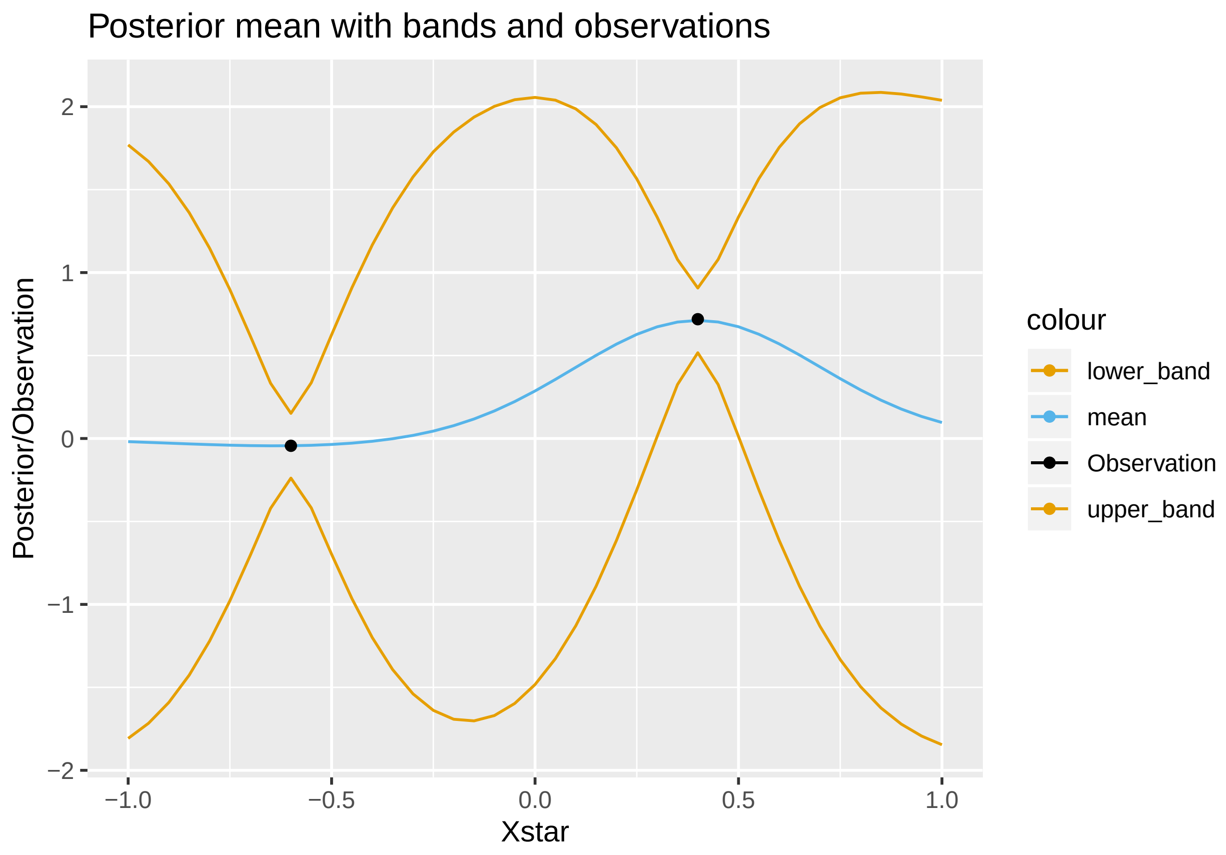
```r
set.seed(111)
XStar = seq(-1, 1, 0.05)

x2 = c(0.4)
y2 = c(0.719)
# sigmaf and l
hyperParam = c(1,0.3)
res = posteriorGP_square_exp_kernel(x=x2, y=y2, XStar=XStar,
                                    hyperParam=hyperParam, sigmaNoise = 0.1)
```

```
plot_gaussian(x2, y2, XStar, res)
```



3) Update your posterior from (2) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95

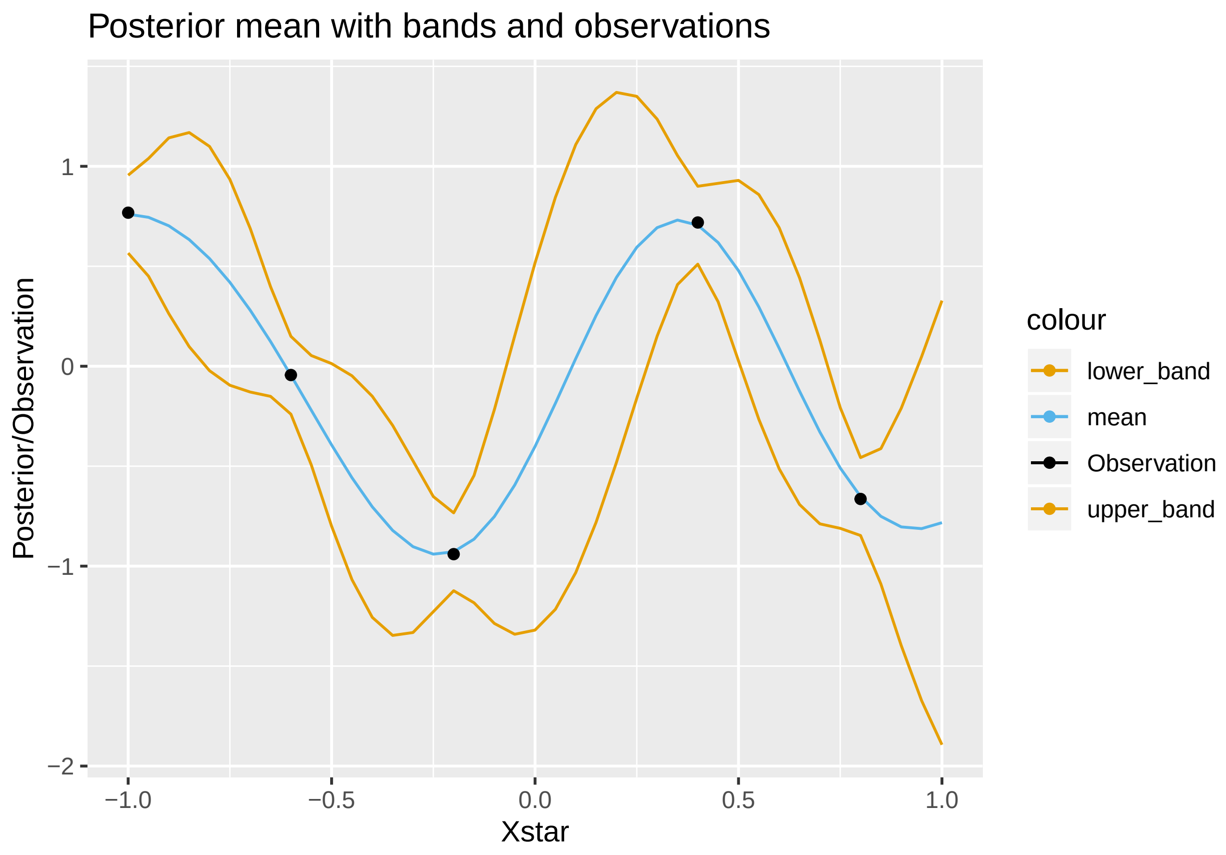
```
set.seed(111)
x3 = c(0.4, -0.6)
y3 = c(0.719, -0.044)
hyperParam=c(1,0.3)
res = posteriorGP_square_exp_kernel(x3, y3, XStar, sigmaNoise = 0.1,
                                    hyperParam=hyperParam)
plot_gaussian(x3, y3, XStar, res)
```



4) Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95

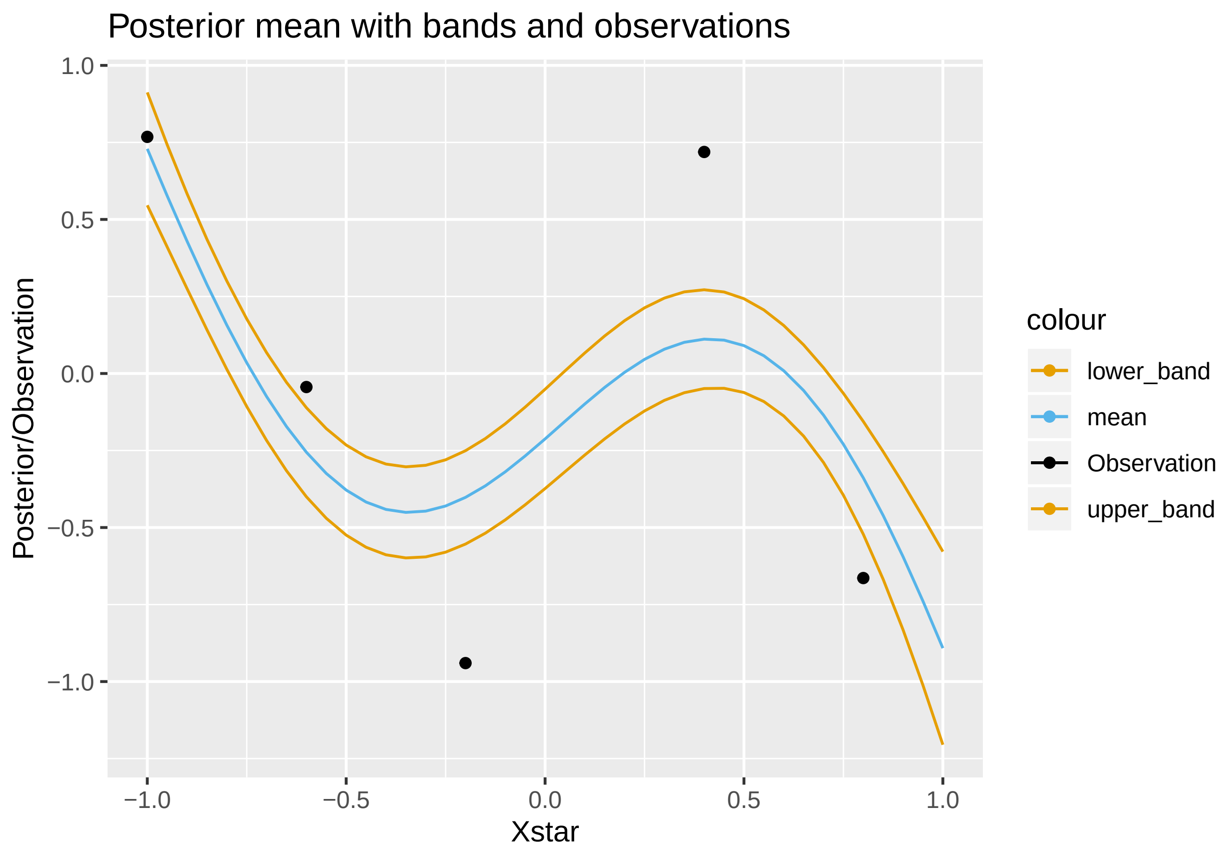
x	y
-1.0	0.768
-0.6	-0.044
-0.2	-0.940
0.4	0.719
0.8	-0.664

```
set.seed(111)
x4 = c(-1,-0.6,-0.2,0.4,0.8)
y4 = c(0.768, -0.044, -0.940, 0.719, -0.664)
hyperParam=c(1,0.3)
res = posteriorGP_square_exp_kernel(x4, y4, XStar, sigmaNoise = 0.1,
                                    hyperParam=hyperParam)
plot_gaussian(x4, y4, XStar, res)
```



5) Repeat (4), this time with hyperparameters $\sigma_f = 1$ and $l = 1$. Compare the results.

```
set.seed(111)
hyperParam = c(1,1)
res = posteriorGP_square_exp_kernel(x4, y4, XStar, sigmaNoise = 0.1,
                                    hyperParam=hyperParam)
plot_gaussian(x4, y4, XStar, res)
```



Analysis: The function is forced to be too smooth, thus the model fails to capture the any of the data points to a point where even the prediciton bands miss the data points which implies the $sigma_f$ was too low.

Questions 2: GP Regression with kernlab

In this exercise, you will work with the daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. We have removed the leap year day February 29, 2012 to make things simpler.

Create the variable *time* which records the day number since the start of the dataset (i.e., time= 1, 2,. . . ., 365 × 6 = 2190). Also, create the variable *day* that records the day number since the start of each year (i.e., day= 1, 2,. . . ..,365, 1, 2,. . . ..,365). Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the data and use only every fifth observation. This means that your time and day variables are now *time*= 1, 6,11,. . . ., 2186 and *day*= 1, 6, 11,. . . .., 361, 1, 6, 11,. . . ., 361.

1) Familiarize yourself with the functions gausspr and kernelMatrix in kernlab. Do ?gausspr and read the input arguments and the output. Also, go through the file KernLabDemo.R available on the course website. You will need to understand it. Now, define your own square exponential kernel function (with parameters l and σ_f), evaluate it in the point x = 1, x' = 2, and use the kernelMatrix function to compute the covariance matrix $K(X, X_*)$ for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

```
set.seed(111)

tulling_data = read.csv("TempTullinge.csv", header=TRUE, sep=";")
tulling_data$date <- as.Date(tulling_data$date, format = "%d/%m/%y")
tulling_data$time = seq(1, 2190, 1)
```

```
tulling_data$time2 = tulling_data$time^2
tulling_data$day = rep(seq(1, 365, 1), 6)

#sample
tulling_sample_data = tulling_data[seq(1, NROW(tulling_data), 5),]


squared_exp_kernel = function(sigmaF = 1, ell = 1)
{
  SquaredExpKernel <- function(x, y = NULL) {
    n1 = length(x)
    n2 = length(y)
    K = matrix(NA, n1, n2)
    for(i in 1:n2) {
      for(j in 1:n1) {
        K[j,i] = sigmaF^2 * exp(-0.5 * ( (x[i] - y[j]) / ell)^2 )
      }
    }
    return(K)
  }
  class(SquaredExpKernel) <- "kernel"
  return(SquaredExpKernel)
}


k = squared_exp_kernel()

k(1,2)
```

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

```
kernelMatrix(k, c(1,3,4), 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
```

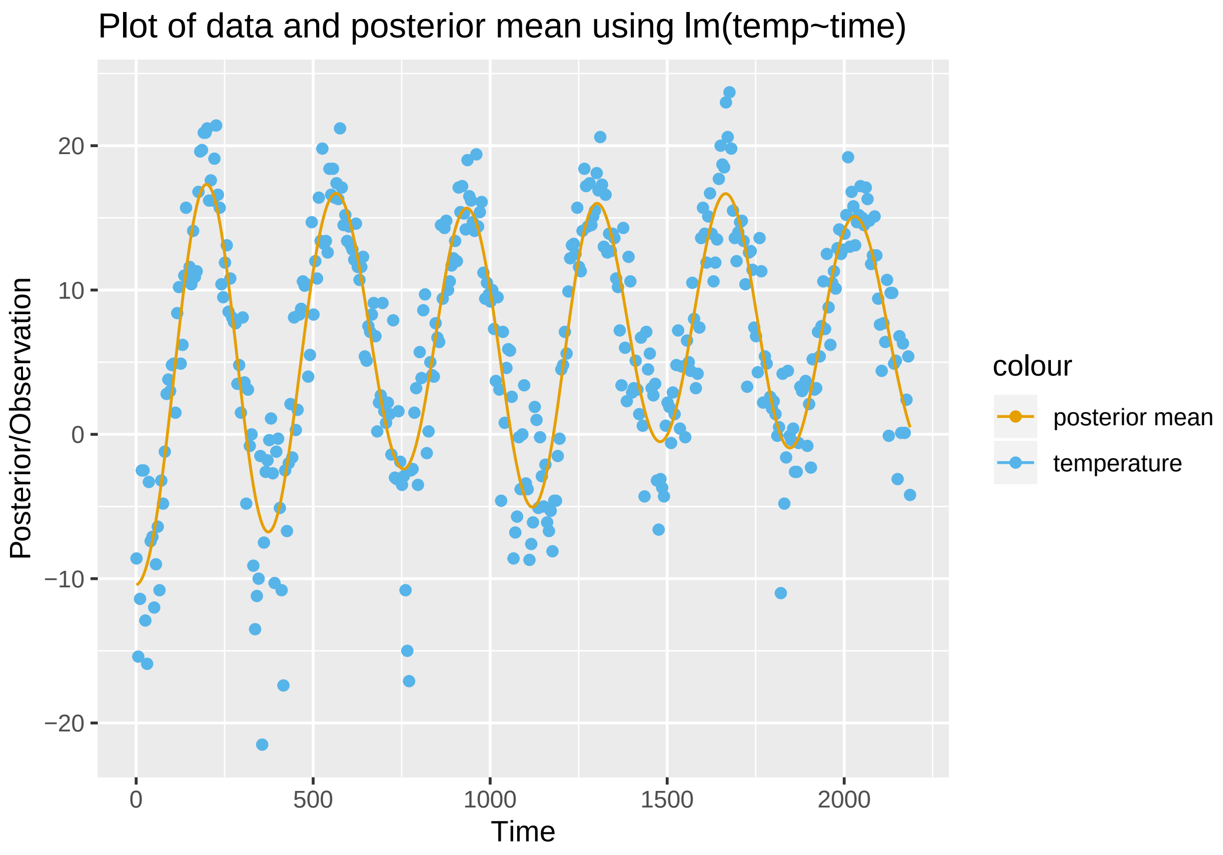
2) Consider first the following model:

$$temp = f(time) + \epsilon \quad with\ \epsilon \sim N(0, \sigma_n^2)\ and\ f \sim GP(0, k(time, time'))$$

Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the lm function in R). Estimate the above Gaussian process regression model using the squared exponential function from (1) with $\sigma_f = 20$ and $l = 0.2$. Use the predict function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use type="l" in the plot function). Play around with different values on f and l (no need to write this in the report though).

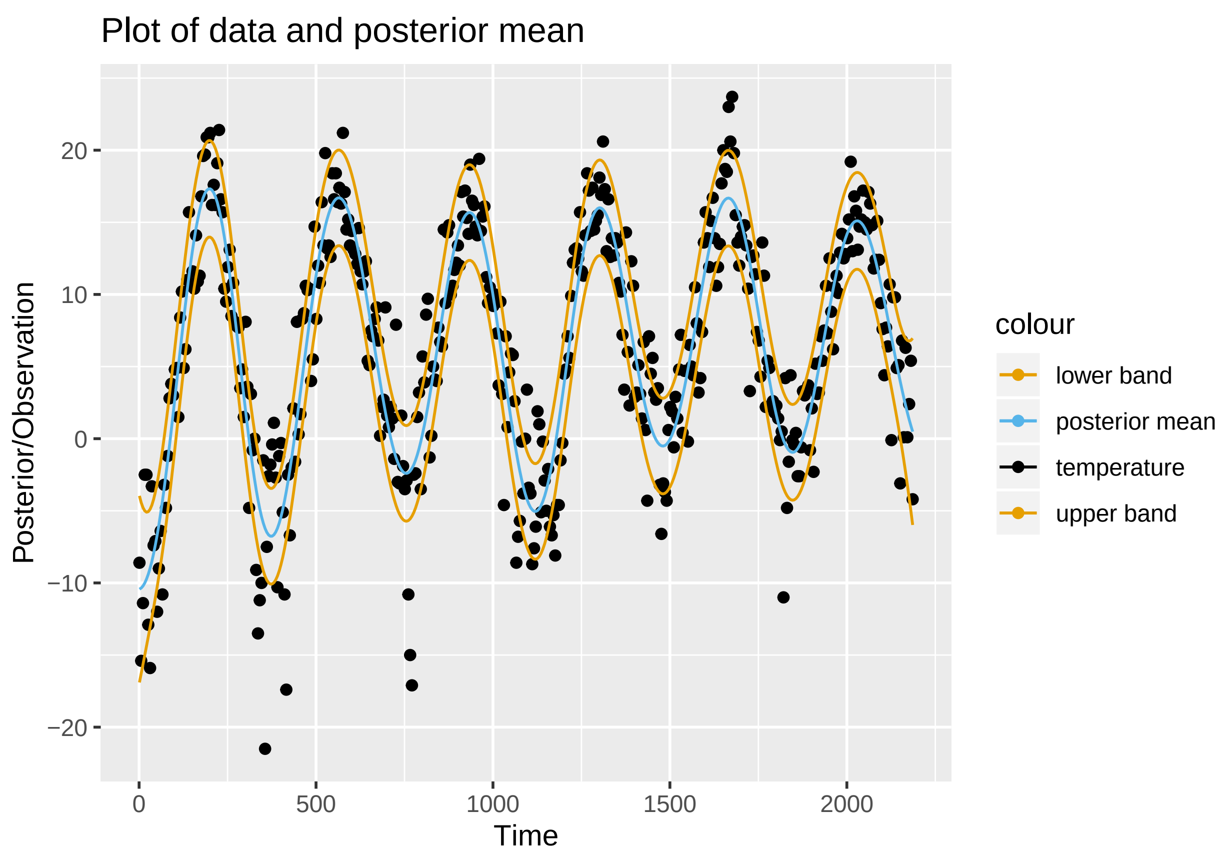
```
set.seed(111)
```

```r
lm_model = lm(temp ~ time + time2, data = tulling_sample_data)
sigma_n = sd(lm_model$residuals)

# Estimate Gaussian Process
GP_time = gausspr(temp ~ time, data = tulling_sample_data,
                  kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                  var = sigma_n^2)

posterior_mean = predict(GP_time, tulling_sample_data)

df <- data.frame(posterior_mean=posterior_mean, tulling_sample_data)

ggplot(data=df, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean")) +
  ggtitle("Plot of data and posterior mean using lm(temp~time)")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#000000", "#E69F00"))
```



3) kernlab can compute the posterior variance of f, but it seems to be a bug in the code. So, do your own computations for the posterior variance of f and plot the 95% probability (pointwise) bands for f. Superimpose these bands on the figure with the posterior mean that you obtained in (2).

```r
set.seed(111)
X = scale(tulling_sample_data$time)
XStar = scale(X)

KStarStar = kernelMatrix(kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                         x = XStar, y = XStar)

KStar = kernelMatrix(kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                     x = X, y = XStar)

K = kernelMatrix(kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                 x = X, y = XStar)

V = diag(KStarStar - t(KStar) %*% solve(K + sigma_n^2 * diag(length(X)), KStar))

df$upper_band <- df$posterior_mean + 1.96 * sqrt(V)
df$lower_band <- df$posterior_mean - 1.96 * sqrt(V)

ggplot(data=df, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean")) +
    geom_line(aes(y=upper_band, color="upper band")) +
    geom_line(aes(y=lower_band, color="lower band")) +
  ggtitle("Plot of data and posterior mean")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#000000", "#E69F00"))
```

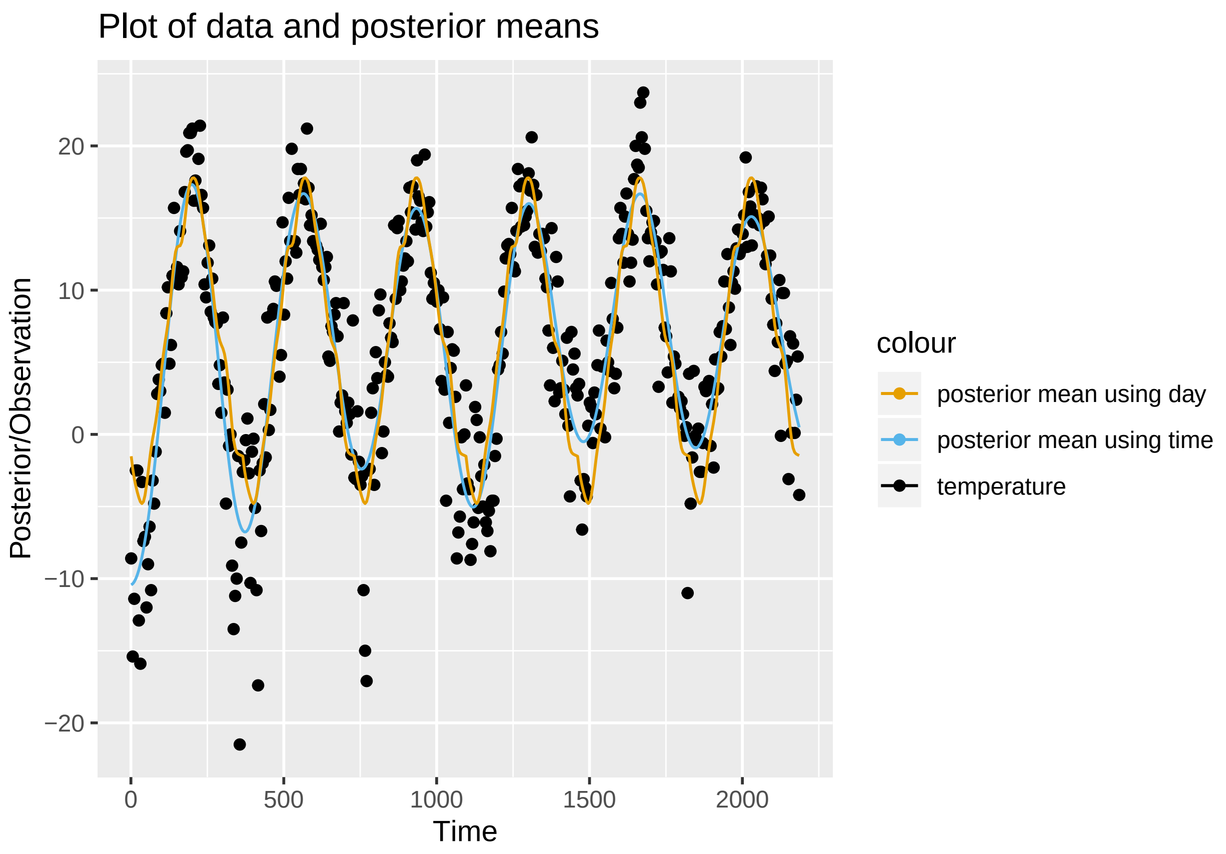


4) Consider now the following model:

$$temp = f(day) + \epsilon \quad with\ \epsilon \sim N(0, \sigma_n^2)\ and\ f \sim GP(0, k(day, day'))$$

Estimate the model using the squared exponential function with $\sigma_f = 20$ and $l = 0.2$. Superimpose the posterior mean from this model on the posterior mean from the model in (2). Note that this plot should also have the time variable on the horizontal axis. Compare the results of both models. What are the pros and cons of each model?

```r
set.seed(111)
lm_model = lm(temp ~ day, data = tulling_sample_data)
sigma_n = sd(lm_model$residuals)

# Estimate Gaussian Process
k = squared_exp_kernel(sigmaF = 20, ell = 0.2)
GP_day = gausspr(temp ~ day, data = tulling_sample_data,
                 kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                 var = sigma_n^2)

posterior_mean_day = predict(GP_day, tulling_sample_data)

df2 <- cbind(df,posterior_mean_day=posterior_mean_day)

ggplot(data=df2, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean using time")) +
  geom_line(aes(y=posterior_mean_day, color="posterior mean using day")) +
  ggtitle("Plot of data and posterior means")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#000000", "#E69F00"))
```



Analysis: The pro of using periodic kernel is that it helps in getting a better fit, especially if the periodic is very unique (recession every decade). Using time as kernel is the simplist approach and this may work very well with noisy data. Using Day as kernel is like a compromise between day and periodic kernel.

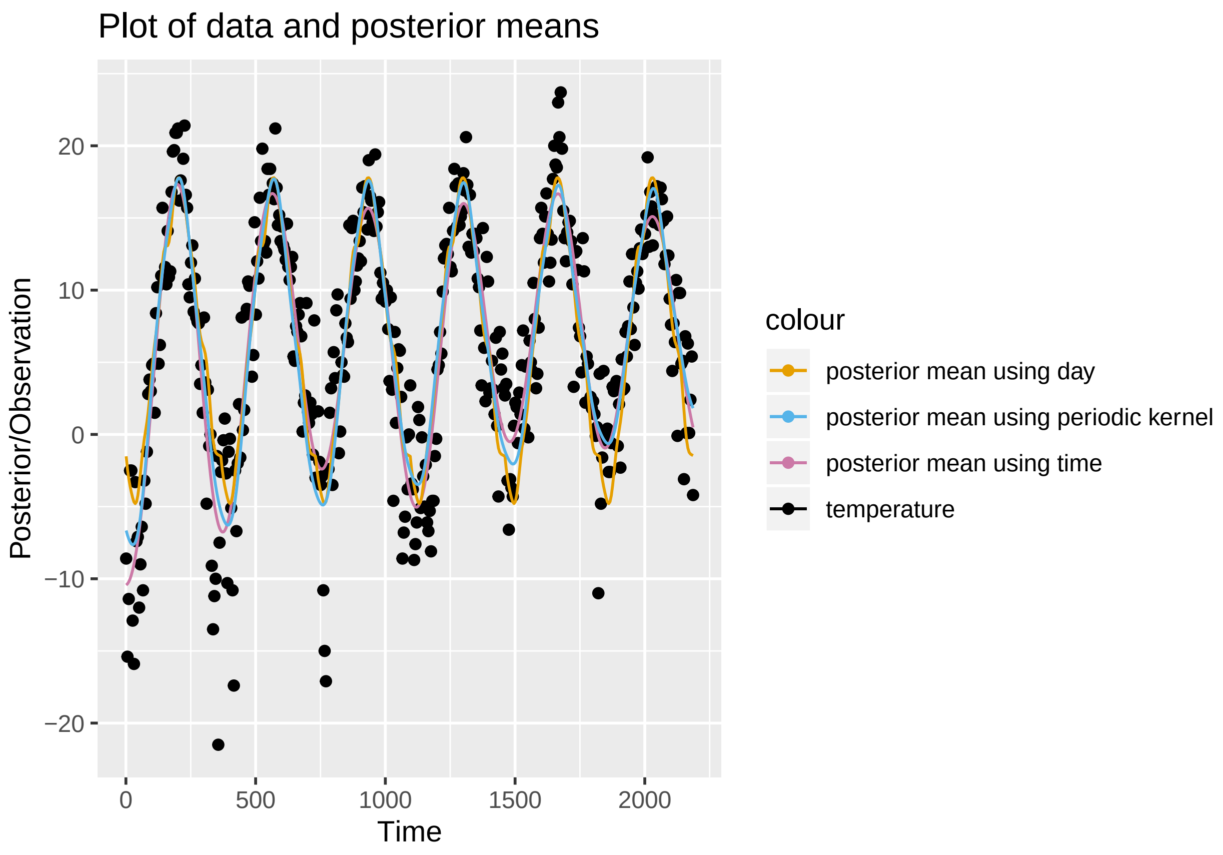
The cons of periodic kernel is that these can be hard to tune since there are more hyper-parameters especially with complex periodic function. The con of using time as the kernel is that this makes very little assumptions about the data so the fit will be worse in complex patterns. The con using day as the periodic is if the underlying assumption of periodic is not yearly then this kernel will be bad.

5) Consider now the following model:

$$k(x,x') = \sigma_f^2 exp((-\frac{2sin^2(\pi\frac{|x-x'|}{d}))}{l_1^2}exp(-\frac{1}{2}\frac{|x-x'|^2}{l_2^2})$$

Note that we have two different length scales here, and l_2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20$, $l_1 = 1$, $l_2 = 10$ and $d = 365/sd(time)$. The reason for the rather strange period here is that kernlab standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20$ and $l = 0.2$). Discuss the results.

```r
set.seed(111)

periodic_kernel = function(sigmaF = 1, l1 = 1, l2 = 1, d)
{
  periodickernel <- function(x, y) {
    K = sigmaF^2 *
      exp(-2 * sin(pi * abs(y - x) / d)^2 / l1^2 ) *
      exp(-0.5 * (y - x)^2 / l2^2 )
    return(K)
  }
  class(periodickernel) <- "kernel"
  return(periodickernel)
}

GP_periodic = gausspr(temp ~ time, data = tulling_sample_data,
                     kernel = periodic_kernel(sigmaF = 20, l1 = 1, l2 = 10,
                                      d=365/sd(tulling_sample_data$time)),
                     var = sigma_n^2)

posterior_mean_periodic = predict(GP_periodic, tulling_sample_data)

df3 <- cbind(df2,posterior_mean_periodic=posterior_mean_periodic)

ggplot(data=df3, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean using time")) +
  geom_line(aes(y=posterior_mean_day, color="posterior mean using day")) +
  geom_line(aes(y=posterior_mean_periodic, color="posterior mean using periodic kernel")) +
  ggtitle("Plot of data and posterior means")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#CC79A7", "#000000"))
```



Analysis: Almost very close to previous approaches, this is since all our kernels are based on the distance and they account for the distance in days rather well.

Questions 3: GP Classification with kernlab

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

1) Use the R package kernlab to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave. Overlay the training data for fraud = 1 (as blue points) and fraud = 0 (as red points). You can reuse code from the file KernLabDemo.R available on the course website. Compute the confusion matrix for the classifier and its accuracy.

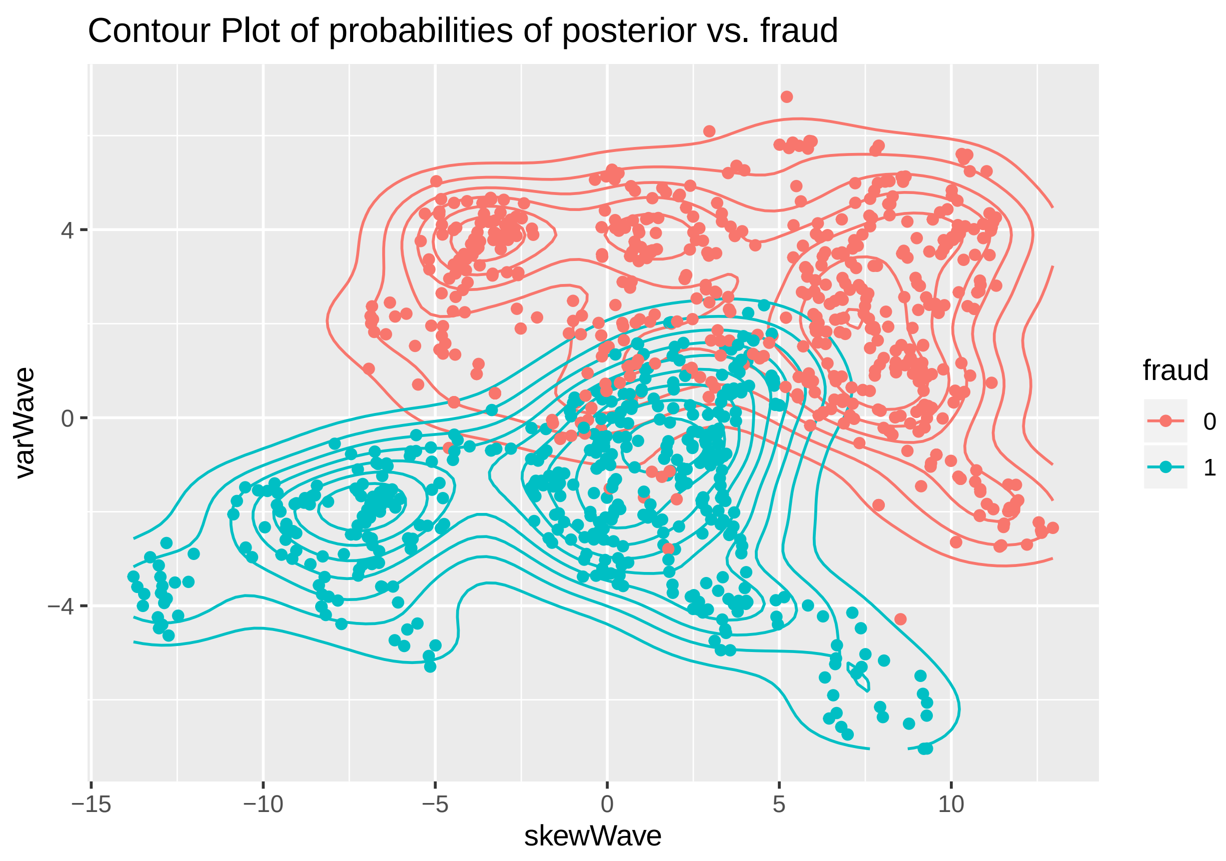
```r
set.seed(111)
fraud_data <- read.csv("BankFraud.csv")
names(fraud_data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")

SelectTraining <- sample(1:NROW(fraud_data), size = 1000, replace = FALSE)
train = fraud_data[SelectTraining,]
test = fraud_data[-SelectTraining,]
train$fraud <- as.factor(train$fraud)
test$fraud <- as.factor(test$fraud)

fit_model = gausspr(fraud ~ varWave + skewWave, data = train)
```

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

```
#create a copy
train2 <- train

# Class predictions on train data
train2$probability_train = predict(fit_model, train2, type="probabilities")[,2]
train2$predictions_class = ifelse(train2$probability_train > 0.5, 1, 0)

# Class predictions on test data
probability_test = predict(fit_model, test, type="probabilities")
test$predictions_class = ifelse(probability_test[,2] > 0.5, 1, 0)

train2$predictions_class <- as.factor(train2$predictions_class)
confusionMatrix(train2$fraud, train2$predictions_class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 506  39
##          1  27 428
##
##                Accuracy : 0.934
##                  95% CI : (0.9168, 0.9486)
##     No Information Rate : 0.533
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.8672
##
##  Mcnemar's Test P-Value : 0.1757
##
##             Sensitivity : 0.9493
##             Specificity : 0.9165
##          Pos Pred Value : 0.9284
##          Neg Pred Value : 0.9407
##              Prevalence : 0.5330
##          Detection Rate : 0.5060
##    Detection Prevalence : 0.5450
##       Balanced Accuracy : 0.9329
##
##        'Positive' Class : 0
##
```

```
# contour plot
ggplot(data=train2, aes(x=skewWave, y=varWave, z=probability_train)) +
  geom_density_2d(aes(color=fraud)) +
  geom_point(aes(color=fraud)) +
  ggtitle("Contour Plot of probabilities of posterior vs. fraud")
```



2) Using the estimated model from (1), make predictions for the test set. Compute the accuracy.

```
test$predictions_class <- as.factor(test$predictions_class)
confusionMatrix(test$fraud, test$predictions_class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 198  18
##          1  10 145
##
##                Accuracy : 0.9245
##                  95% CI : (0.8928, 0.9493)
##     No Information Rate : 0.5606
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.846
##
##  Mcnemar's Test P-Value : 0.1859
##
##             Sensitivity : 0.9519
##             Specificity : 0.8896
##          Pos Pred Value : 0.9167
##          Neg Pred Value : 0.9355
##              Prevalence : 0.5606
##          Detection Rate : 0.5337
##    Detection Prevalence : 0.5822
##       Balanced Accuracy : 0.9207
##
##        'Positive' Class : 0
##
```

3) Train a model using all four covariates. Make predictions on the test set and compare the accuracy to the model with only two covariates.

```r
fit_model_four_var = gausspr(fraud ~., data = train)
```

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

```r
# Class predictions on test data
probability_test_four_cov = predict(fit_model_four_var, test, type="probabilities")
test$predictions_class_four_cov = ifelse(probability_test_four_cov[,2] > 0.5, 1, 0)

test$predictions_class_four_cov <- as.factor(test$predictions_class_four_cov)
confusionMatrix(test$fraud, test$predictions_class_four_cov)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 214   2
##          1   0 155
##
##                Accuracy : 0.9946
##                  95% CI : (0.9807, 0.9993)
##     No Information Rate : 0.5768
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.9889
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9873
##          Pos Pred Value : 0.9907
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5768
##          Detection Rate : 0.5768
##    Detection Prevalence : 0.5822
##       Balanced Accuracy : 0.9936
##
##        'Positive' Class : 0
##
```

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
options(scipen=999)

library("tidyverse") #ggplot and dplyr
library("gridExtra") # combine plots
library("knitr") # for pdf
library("kernlab") # Gaussian Regression
library("mvtnorm") # multi dimensional normal distribution
library("caret") # confusion matrix
library("e1071") # confusion matrix and accuracy

# The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442",
                "#0072B2", "#D55E00", "#CC79A7")
set.seed(111)
set.seed(111)

square_exp_kernel = function(x1, x2, sigmaF, l) {
  n1 = length(x1)
  n2 = length(x2)
  K = matrix(NA, n1, n2)
  for(i in 1:n2) {
    for(j in 1:n1) {
      K[j,i] = sigmaF^2 * exp(-0.5 * ( (x2[i] - x1[j])/l)^2 )
    }
  }
  return(K)
}

posteriorGP_square_exp_kernel <- function(x,y,XStar,hyperParam, sigmaNoise){
  sigmaF <- hyperParam[1]
  l <- hyperParam[2]
  n = length(x)
  L = t(chol(square_exp_kernel(x,x, sigmaF, l) + sigmaNoise^2 * diag(n)))
  KStar = square_exp_kernel(x, XStar, sigmaF, l)

  # Predictive Mean
  alpha = solve(t(L), solve(L, y))
  fStarBar = t(KStar) %*% alpha

  # Predictive Variance
  v = solve(L, KStar)
  V = square_exp_kernel(XStar, XStar,sigmaF, l) - t(v) %*% v

  return(data.frame(mean = fStarBar, variance = diag(V)))
}

Visualize = function(X, y, XStar, res) {
  mean = res$mean
  sd = sqrt(res$variance)
  plot(XStar, mean, type="l", ylim=c(-3, 4),
       xlab="x", ylab="y", main="Posterior Gaussian Process")
  lines(XStar, mean - 1.96 * sd, type="l", col="blue")
  lines(XStar, mean + 1.96 * sd, type="l", col="blue")
  points(X, y, pch=19)
  legend("topright",
         legend=c("posterior mean", "probability band", "observations"),
         col=c("black", "blue", "black"), lty=c(1,1,0), pch=c(-1,-1,19))
}


plot_gaussian <- function(x,y,Xstar,res){
mu <- res$mean
sd <- sqrt(res$variance)
df <- data.frame(XStar=XStar,
                 mu = mu,
                 upper_band = mu + 1.96 * sd,
                 lower_band = mu - 1.96 * sd)

df2 <- data.frame(x=x,y=y)

plot1 <- ggplot(data=df, aes(x=XStar)) +
  geom_line(aes(y=mu, color="mean")) +
  geom_line(aes(y=upper_band, color="upper_band")) +
  geom_line(aes(y=lower_band, color="lower_band")) +
  geom_point(data=df2, aes(x=x, y=y, color="Observation")) +
  ggtitle("Posterior mean with bands and observations") +
  ylab("Posterior/Observation") + xlab("Xstar") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9", "#000000", "#E69F00"))

print(plot1)

}

set.seed(111)
XStar = seq(-1, 1, 0.05)

x2 = c(0.4)
y2 = c(0.719)
# sigmaf and l
hyperParam = c(1,0.3)
res = posteriorGP_square_exp_kernel(x=x2, y=y2, XStar=XStar,
                                    hyperParam=hyperParam, sigmaNoise = 0.1)

plot_gaussian(x2, y2, XStar, res)
set.seed(111)
x3 = c(0.4, -0.6)
y3 = c(0.719, -0.044)
hyperParam=c(1,0.3)
res = posteriorGP_square_exp_kernel(x3, y3, XStar, sigmaNoise = 0.1,
                                    hyperParam=hyperParam)
plot_gaussian(x3, y3, XStar, res)
x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
```

```r
kable(data.frame(x=x,y=y))
set.seed(111)
x4 = c(-1,-0.6,-0.2,0.4,0.8)
y4 = c(0.768, -0.044, -0.940, 0.719, -0.664)
hyperParam=c(1,0.3)
res = posteriorGP_square_exp_kernel(x4, y4, XStar, sigmaNoise = 0.1,
                                    hyperParam=hyperParam)
plot_gaussian(x4, y4, XStar, res)

set.seed(111)
hyperParam = c(1,1)
res = posteriorGP_square_exp_kernel(x4, y4, XStar, sigmaNoise = 0.1,
                                    hyperParam=hyperParam)
plot_gaussian(x4, y4, XStar, res)
set.seed(111)

tulling_data = read.csv("TempTullinge.csv", header=TRUE, sep=";")
tulling_data$date <- as.Date(tulling_data$date, format = "%d/%m/%y")
tulling_data$time = seq(1, 2190, 1)
tulling_data$time2 = tulling_data$time^2
tulling_data$day = rep(seq(1, 365, 1), 6)

#sample
tulling_sample_data = tulling_data[seq(1, NROW(tulling_data), 5),]


squared_exp_kernel = function(sigmaF = 1, ell = 1)
{
  SquaredExpKernel <- function(x, y = NULL) {
    n1 = length(x)
    n2 = length(y)
    K = matrix(NA, n1, n2)
    for(i in 1:n2) {
      for(j in 1:n1) {
        K[j,i] = sigmaF^2 * exp(-0.5 * ( (x[i] - y[j]) / ell)^2 )
      }
    }
    return(K)
  }
  class(SquaredExpKernel) <- "kernel"
  return(SquaredExpKernel)
}


k = squared_exp_kernel()

k(1,2)

kernelMatrix(k, c(1,3,4), c(2,3,4))

set.seed(111)

lm_model = lm(temp ~ time + time2, data = tulling_sample_data)
```

```r
sigma_n = sd(lm_model$residuals)

# Estimate Gaussian Process
GP_time = gausspr(temp ~ time, data = tulling_sample_data,
                  kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                  var = sigma_n^2)

posterior_mean = predict(GP_time, tulling_sample_data)

df <- data.frame(posterior_mean=posterior_mean, tulling_sample_data)

ggplot(data=df, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean")) +
  ggtitle("Plot of data and posterior mean using lm(temp~time)")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#000000", "#E69F00"))

set.seed(111)
X = scale(tulling_sample_data$time)
XStar = scale(X)

KStarStar = kernelMatrix(kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                         x = XStar, y = XStar)

KStar = kernelMatrix(kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                     x = X, y = XStar)

K = kernelMatrix(kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                 x = X, y = XStar)

V = diag(KStarStar - t(KStar) %*% solve(K + sigma_n^2 * diag(length(X)), KStar))

df$upper_band <- df$posterior_mean + 1.96 * sqrt(V)
df$lower_band <- df$posterior_mean - 1.96 * sqrt(V)

ggplot(data=df, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean")) +
    geom_line(aes(y=upper_band, color="upper band")) +
    geom_line(aes(y=lower_band, color="lower band")) +
  ggtitle("Plot of data and posterior mean")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#000000", "#E69F00"))

set.seed(111)
lm_model = lm(temp ~ day, data = tulling_sample_data)
sigma_n = sd(lm_model$residuals)

# Estimate Gaussian Process
k = squared_exp_kernel(sigmaF = 20, ell = 0.2)
```

```r
GP_day = gausspr(temp ~ day, data = tulling_sample_data,
                 kernel = squared_exp_kernel(sigmaF = 20, ell = 0.2),
                 var = sigma_n^2)

posterior_mean_day = predict(GP_day, tulling_sample_data)

df2 <- cbind(df,posterior_mean_day=posterior_mean_day)

ggplot(data=df2, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean using time")) +
  geom_line(aes(y=posterior_mean_day, color="posterior mean using day")) +
  ggtitle("Plot of data and posterior means")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#000000", "#E69F00"))

set.seed(111)

periodic_kernel = function(sigmaF = 1, l1 = 1, l2 = 1, d)
{
  periodickernel <- function(x, y) {
    K = sigmaF^2 *
      exp(-2 * sin(pi * abs(y - x) / d)^2 / l1^2 ) *
      exp(-0.5 * (y - x)^2 / l2^2 )
    return(K)
  }
  class(periodickernel) <- "kernel"
  return(periodickernel)
}

GP_periodic = gausspr(temp ~ time, data = tulling_sample_data,
                      kernel = periodic_kernel(sigmaF = 20, l1 = 1, l2 = 10,
                                         d=365/sd(tulling_sample_data$time)),
                      var = sigma_n^2)

posterior_mean_periodic = predict(GP_periodic, tulling_sample_data)

df3 <- cbind(df2,posterior_mean_periodic=posterior_mean_periodic)

ggplot(data=df3, aes(x=time)) +
  geom_point(aes(y=temp, color="temperature")) +
  geom_line(aes(y=posterior_mean, color="posterior mean using time")) +
  geom_line(aes(y=posterior_mean_day, color="posterior mean using day")) +
  geom_line(aes(y=posterior_mean_periodic, color="posterior mean using periodic kernel")) +
  ggtitle("Plot of data and posterior means")  +
  ylab("Posterior/Observation") + xlab("Time") +
        scale_colour_manual(values = c("#E69F00", "#56B4E9",
                                       "#CC79A7", "#000000"))

set.seed(111)
fraud_data <- read.csv("BankFraud.csv")
names(fraud_data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
```

```r
SelectTraining <- sample(1:NROW(fraud_data), size = 1000, replace = FALSE)
train = fraud_data[SelectTraining,]
test = fraud_data[-SelectTraining,]
train$fraud <- as.factor(train$fraud)
test$fraud <- as.factor(test$fraud)

fit_model = gausspr(fraud ~ varWave + skewWave, data = train)

#create a copy
train2 <- train

# Class predictions on train data
train2$probability_train = predict(fit_model, train2, type="probabilities")[,2]
train2$predictions_class = ifelse(train2$probability_train > 0.5, 1, 0)

# Class predictions on test data
probability_test = predict(fit_model, test, type="probabilities")
test$predictions_class = ifelse(probability_test[,2] > 0.5, 1, 0)

train2$predictions_class <- as.factor(train2$predictions_class)
confusionMatrix(train2$fraud, train2$predictions_class)

# contour plot
ggplot(data=train2, aes(x=skewWave, y=varWave, z=probability_train)) +
  geom_density_2d(aes(color=fraud)) +
  geom_point(aes(color=fraud)) +
  ggtitle("Contour Plot of probabilities of posterior vs. fraud")



test$predictions_class <- as.factor(test$predictions_class)
confusionMatrix(test$fraud, test$predictions_class)


fit_model_four_var = gausspr(fraud ~., data = train)

# Class predictions on test data
probability_test_four_cov = predict(fit_model_four_var, test, type="probabilities")
test$predictions_class_four_cov = ifelse(probability_test_four_cov[,2] > 0.5, 1, 0)

test$predictions_class_four_cov <- as.factor(test$predictions_class_four_cov)
confusionMatrix(test$fraud, test$predictions_class_four_cov)
```