AML_lab4

Zijie Feng

2019/10/9

2.1 Implementing GP Regression

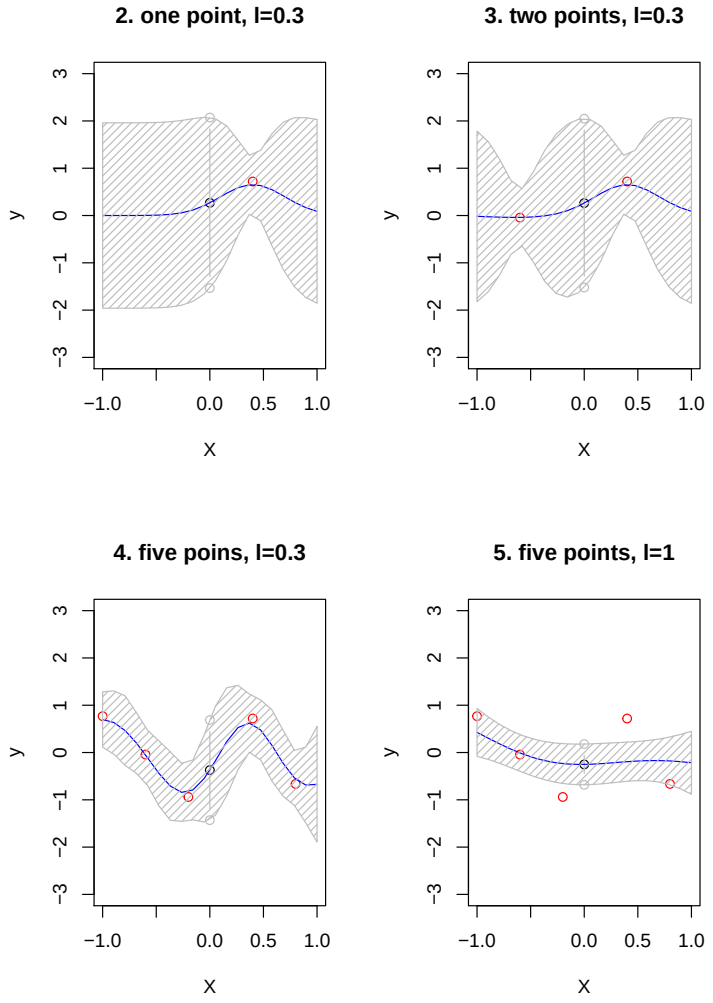
Simulation Code

Write your own code for simulating from the posterior distribution of f using the squared exponential kernel.

```
########### lab4q1 ######################
rm(list=ls())

# covariance function of Gaussian Process Regression
SquaredExpKernel <- function(x1, x2, sigmaF,l){
  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. XStar.
# hyperParam: Vector with two elements sigma_f and l.
# sigmaNoise: Noise standard deviation sigma_n

posteriorGP <- function(X, y, hyperParam, sigmaNoise, XStar){
  N <- length(X)
  K <- SquaredExpKernel(X, X, sigmaF=hyperParam[1],l=hyperParam[2])
  L <- t( chol( K+sigmaNoise*diag(N) ) )
  a <- solve(t(L))%*% (solve(L)%*%y)          # y = K  %*% a
  kStar <- SquaredExpKernel(X, XStar,hyperParam[1],hyperParam[2])
  FStar <- t(kStar)%*%a                       # f = k' %*% a

  v <- solve(L) %*% kStar
  Vf <- SquaredExpKernel(XStar,XStar,hyperParam[1],hyperParam[2])-t(v)%*%v

  logp <- -1/2%*%t(y)%*%a - sum(log(diag(L)))-N/2*log(2*pi)

  # return a vector with the posterior mean and variance of f
  return(list( mean=FStar, var=Vf, logllk=logp ))
}
```

Compute the posterior mean and varaince of f

Update the prior (red points) with $\sigma_n = 0.1$ and comparing the results



It is obvious that the 95% confident interval of $X_* = 0$ is more narrow when more points are considered.

$$k(x, x') = cov(f(x), f(x')) = \sigma_f^2 \exp\{-\frac{|x - x'|^2}{2\ell^2}\}$$

The length of scale ℓ also affects the confident interval evidently. When ℓ is larger, it would make the convariances and varainces of $f(x)$ close to σ_f^2.

$$K = \begin{pmatrix} \sigma_f^2 & \cdots & \sigma_f^2 \\ \vdots & \ddots & \vdots \\ \sigma_f^2 & \cdots & \sigma_f^2 \end{pmatrix}$$

The solution of $y = K\alpha$ will tend to a horizontal line at 0 (very smooth), so do the estiamtes for posterior mean $f = k_*\alpha$. In addition, the confident interval (variance) of posterior mean will smaller afterwards when ℓ is larger, and variable σ_f controls the minimal width of confident interval.

2.2 GP Regression with `kernlab`

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.

(1)

Define your own square exponential kernel function (with parameters ℓ(ell) and σ_f(sigmaf)), 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 $X_* = (2, 3, 4)^T$.

```
se <- function(sigmaf, ell) {
  rval <- function(x, y = NULL) {
    r = sqrt(crossprod(x-y))
    return( sigmaf^2*exp(-(r^2)/(2*ell^2)) )     #sequared exponential error
  }
  class(rval) <- "kernel"
  return(rval)
}

sek = se(sigmaf = 2, ell = 1) # a kernel FUNCTION

sek(1,2)
```

```
##          [,1]
## [1,] 2.426123
```

```
X <- c(1,3,4)
XStar <- c(2,3,4)
K <- kernelMatrix(kernel = sek, x = X, y = XStar)
K
```

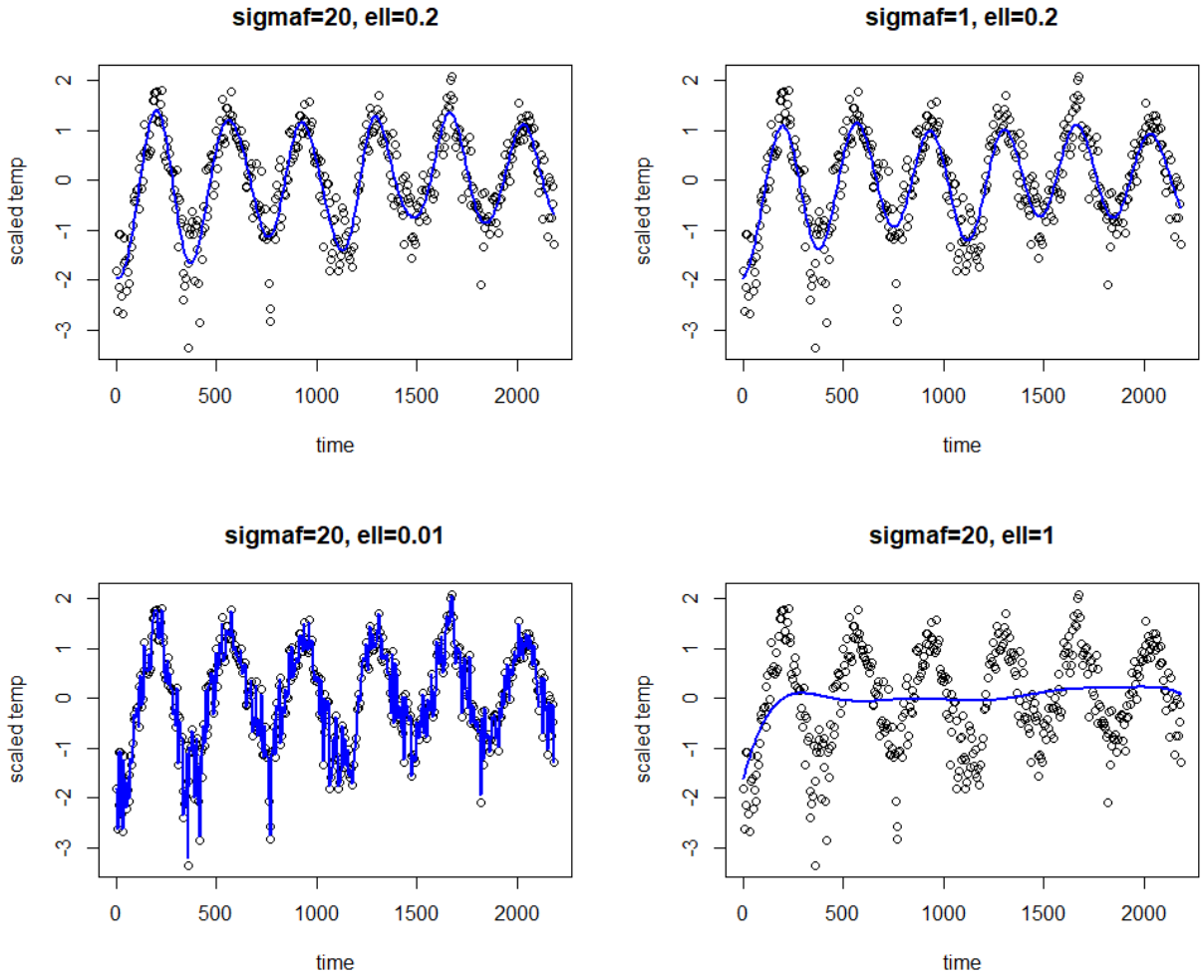
```
## An object of class "kernelMatrix"
##           [,1]      [,2]       [,3]
## [1,] 2.4261226 0.5413411 0.04443599
## [2,] 2.4261226 4.0000000 2.42612264
## [3,] 0.5413411 2.4261226 4.00000000
```

(2)

Consider first the following model:

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

Let σ_n be the residual variance from a simple quadratic regression fit (using the lm function in R).



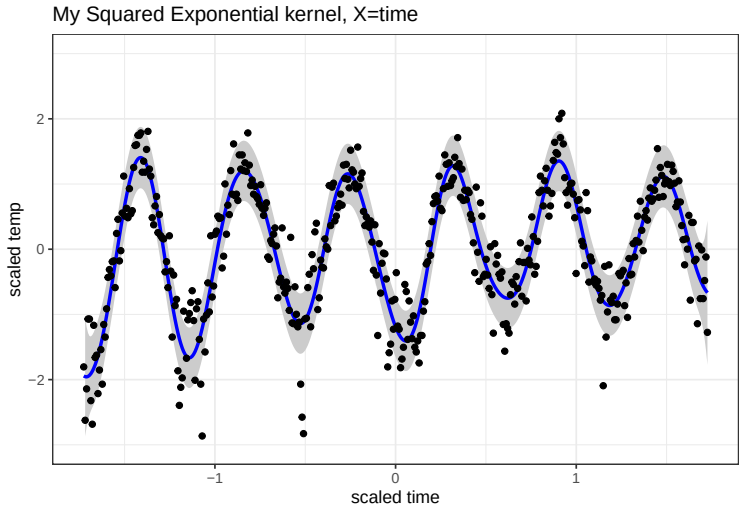
From left to right, top to bottom, the parameters `kpar` of these plots are (20,0.2), (1,0.2), (20,0.01) and (20,1). It seems that the larger σ_f is, the rougher/more detailed the model fit. Kernel parameter ℓ controls the influence of r in squared exponential kernel. The larger ℓ is, the smoother the plot is.

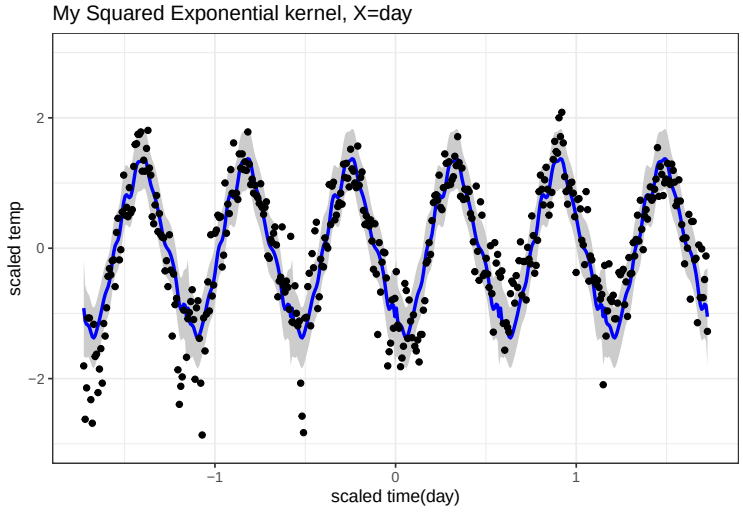
(3) & (4) & (5)

Consider now the following model:

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

Estimate the model using the squared exponential function with $\sigma_f = 20$ and $\ell = 0.2$. 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.

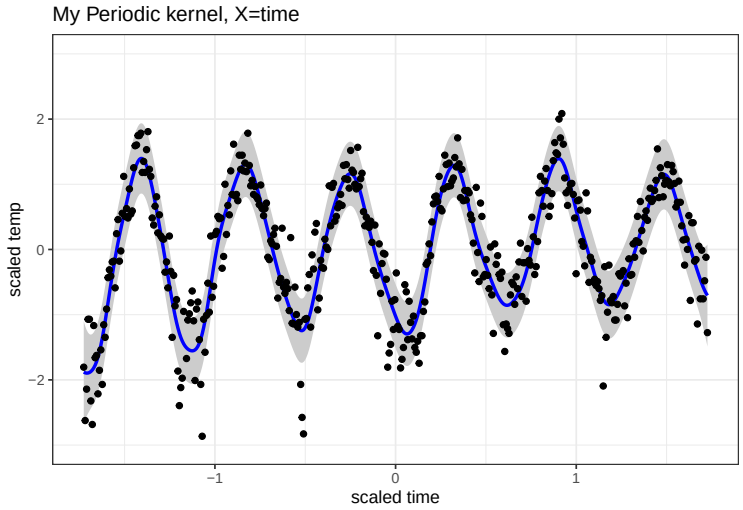




Finally, implement a generalization of the periodic kernel given in the lectures:

$$k(x, x') = \sigma_f^2 \exp\{-\frac{2\sin^2(\pi|x - x'|/d)}{\ell_1^2}\} \exp\{-\frac{1}{2}\frac{|x - x'|^2}{\ell_2^2}\}$$

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 $\sigma_f = 20$, $\ell_1 = 1$, $\ell_2 = 10$ 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 $\ell = 0.2$). Discuss the results.



	uncovered points
Q3	102
Q4	115
Q5	80

it seems that considering $X = time$ is better to fit the temperature than cosidering $X = day$. Although both of them interpret the patterns of $y = temp$, the plot with $X = time$ has smoother path and better confident interval, which misses 102 points. On contrast, the confident interval with $X = day$ misses 115 points and its plot repeats periodically.

However, the model with $X = day$ provides smaller σ_N (0.96<0.99) and store more information from real data (e.g. fluctuation), compared with the first model. Its posterior confident interval would be affected by the density of original data too much.

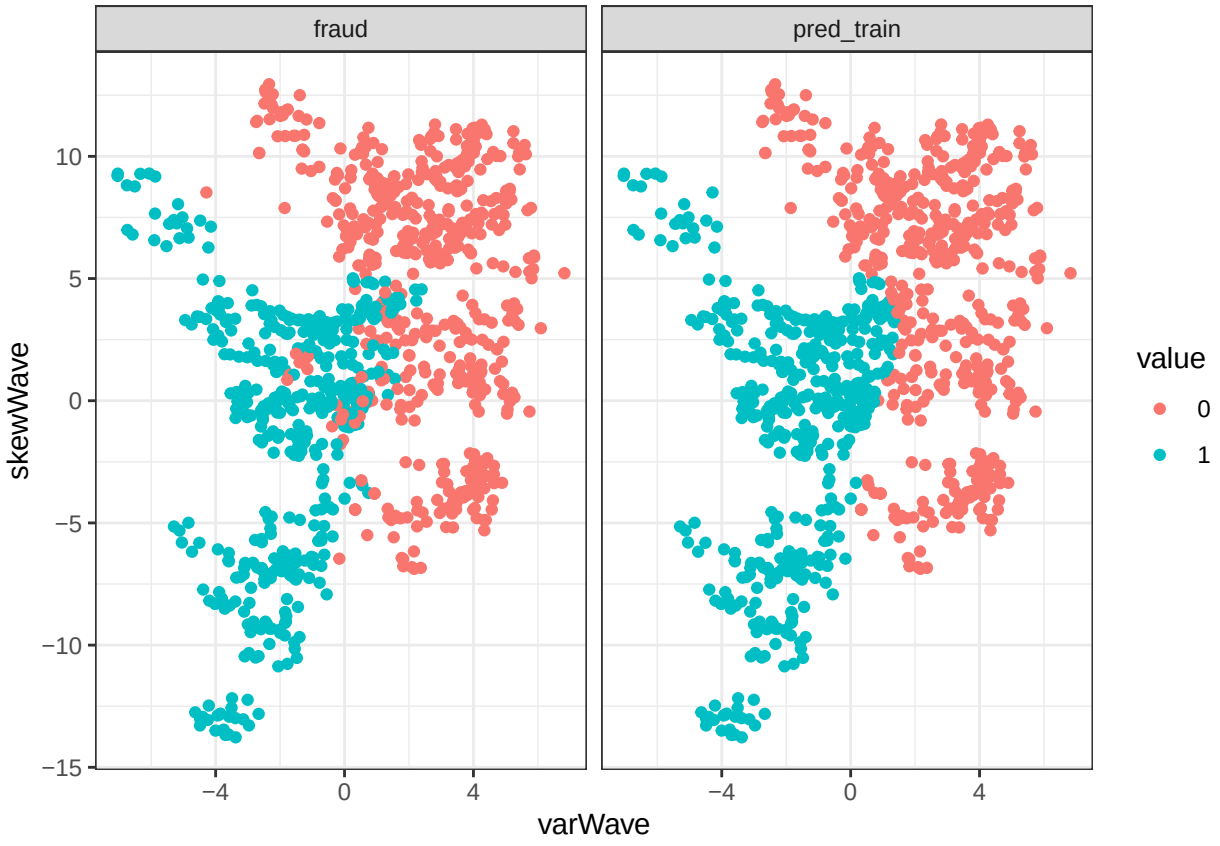
In addition, the preiodic kernel with $X = time$ works the best in all models. There are only 82 points outsides the confident interval. It has the benefits from both the first and the second models. It is smooth enough and its confident interval shows the density of data.

2.3 GP Classification with `kernlab`

(1)

Choose 1000 observations as training data. 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). Compute the confusion matrix for the classifier and its accuracy.

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



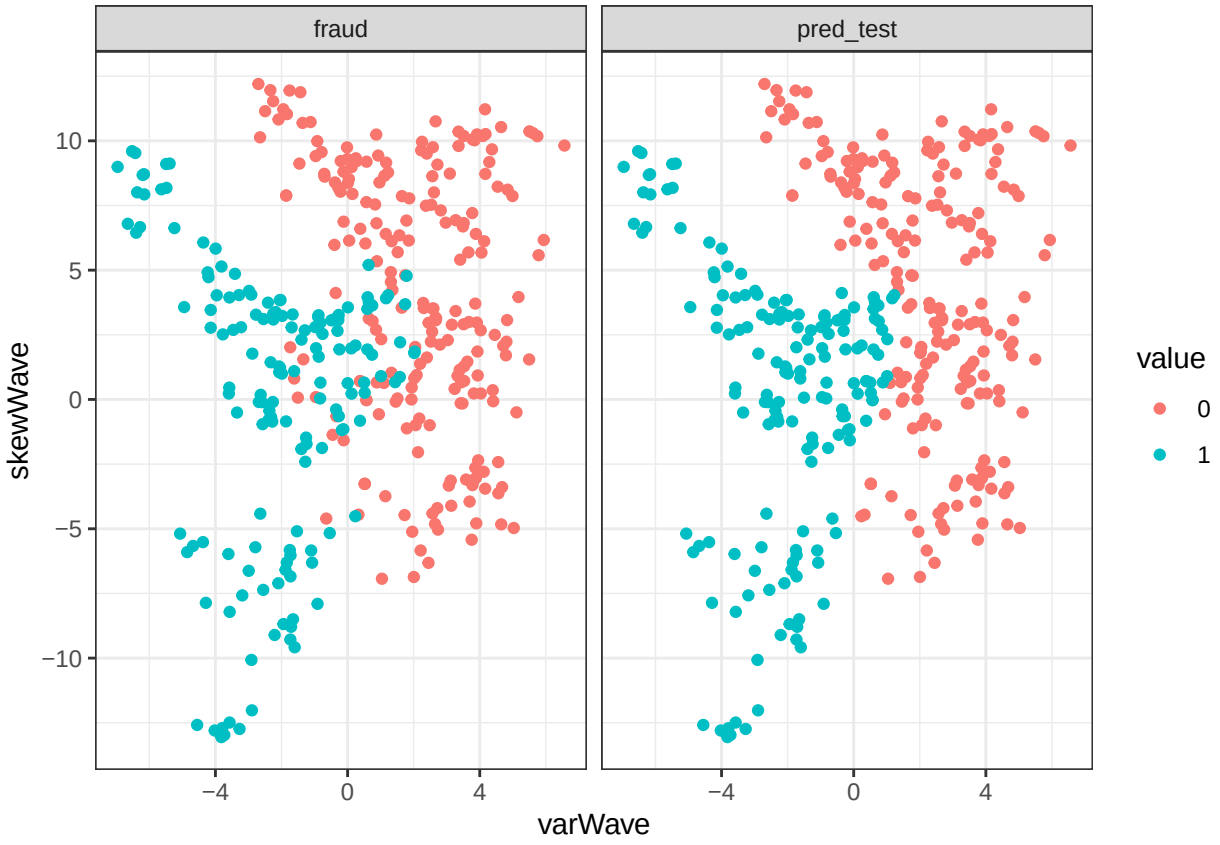

```
## the confusion matrix of train data is
```

	0	1
0	503	18
1	41	438

```
## Accuracy of train data is 0.941
```

(2)

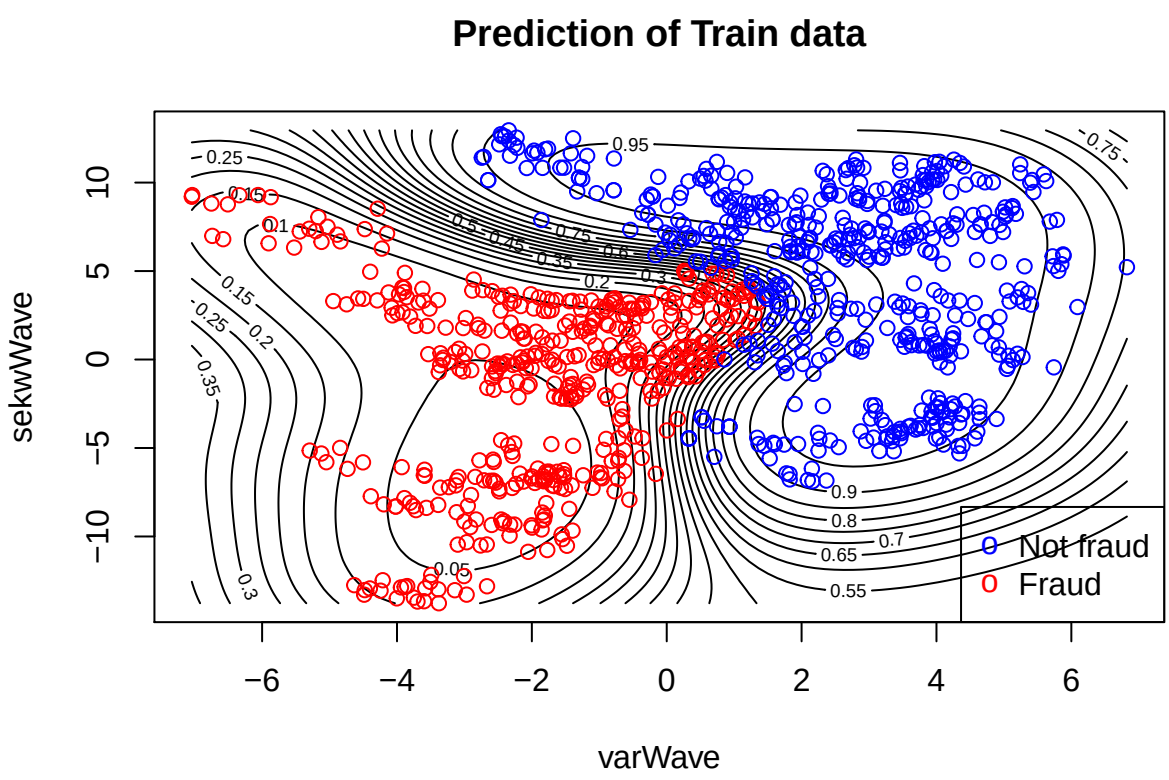
Using the estimated model from (1), make predictions for the test set. Compute the accuracy.




```
## the confusion matrix of train data is
```

	0	1
0	199	9
1	19	145

```
## Accuracy of test data is 0.9247312
```


Prediction of Train data
sekwWave
varWave
Not fraud
Fraud

(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.

```
mdl = gausspr(fraud ~ ., data = train)
pred_all <- predict(mdl, test)
```

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

```
## the confusion matrix of test data is
```

	0	1
0	216	0
1	2	154

```
## Accuracy of test data is 0.9946237
```

It is obvious that the Gaussian process classification model considering all variables has better preformance than the one only considering the covariates *varWave* and *skewWave*.

Appendix

```
############# lab4 ########################
knitr::opts_chunk$set(echo = FALSE, out.width="60%", warning = F)
library(kernlab)
library(ggplot2)
library(kableExtra)
library(gridExtra)
library(AtmRay)
########### lab4q1 ######################
rm(list=ls())

# covariance function of Gaussian Process Regression
SquaredExpKernel <- function(x1, x2, sigmaF,l){
  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. XStar.
# hyperParam: Vector with two elements sigma_f and l.
# sigmaNoise: Noise standard deviation sigma_n

posteriorGP <- function(X, y, hyperParam, sigmaNoise, XStar){
  N <- length(X)
  K <- SquaredExpKernel(X, X, sigmaF=hyperParam[1],l=hyperParam[2])
  L <- t( chol( K+sigmaNoise*diag(N) ) )
  a <- solve(t(L))%*% (solve(L)%*%y)          # y = K  %*% a
  kStar <- SquaredExpKernel(X, XStar,hyperParam[1],hyperParam[2])
  FStar <- t(kStar)%*%a                       # f = k' %*% a

  v <- solve(L) %*% kStar
  Vf <- SquaredExpKernel(XStar,XStar,hyperParam[1],hyperParam[2])-t(v)%*%v

  logp <- -1/2%*%t(y)%*%a - sum(log(diag(L)))-N/2*log(2*pi)

  # return a vector with the posterior mean and variance of f
  return(list( mean=FStar, var=Vf, logllk=logp ))
}
par(mfrow=c(1,2))

hyperParam <- c(1, 0.3)
sigmaNoise <- 0.1
XStar <- 0

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

```
prior <- posteriorGP(X, y, hyperParam, sigmaNoise, XStar)
pp <- posteriorGP(X,y, hyperParam, sigmaNoise, xgrid)

plot(X, y, xlim=c(-1,1), ylim=c(-3,3), col="Red", main="2. one point, l=0.3")
points(XStar, prior$mean, col="Black")
lines(c(XStar,XStar), y=c(prior$mean-1.96*sqrt(prior$var),
                          prior$mean+1.96*sqrt(prior$var)),col="Grey",type="b" )
lines(xgrid, pp$mean, col="blue")
polygon(x = c(rev(xgrid), xgrid),
        y = c(rev(pp$mean+1.96*sqrt(diag(pp$var))), pp$mean-1.96*sqrt(diag(pp$var))),
        col = "grey",
        border = "grey",
        density = c(20))

X <- c(-0.6, 0.4)
y <- c(-0.044, 0.719)
prior <- posteriorGP(X, y, hyperParam, sigmaNoise, XStar)
pp <- posteriorGP(X,y, hyperParam, sigmaNoise, xgrid)

plot(X, y, xlim=c(-1,1), ylim=c(-3,3), col="Red", main="3. two points, l=0.3")
points(XStar, prior$mean, col="Black")
lines(c(XStar,XStar), y=c(prior$mean-1.96*sqrt(prior$var),prior$mean+1.96*sqrt(prior$var)),col="Grey",ty
lines(xgrid, pp$mean, col="blue")
polygon(x = c(rev(xgrid), xgrid),
        y = c(rev(pp$mean+1.96*sqrt(diag(pp$var))), pp$mean-1.96*sqrt(diag(pp$var))),
        col = "grey",
        border = "grey",
        density = c(20))
par(mfrow=c(1,2))

hyperParam <- c(1, 0.3)
sigmaNoise <- 0.1
X <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.94, 0.719, -0.664)
prior <- posteriorGP(X, y, hyperParam, sigmaNoise, XStar)
pp <- posteriorGP(X,y, hyperParam, sigmaNoise, xgrid)

plot(X, y, xlim=c(-1,1), ylim=c(-3,3), col="Red", main="4. five poins, l=0.3")
points(XStar, prior$mean, col="Black")
lines(c(XStar,XStar), y=c(prior$mean-1.96*sqrt(prior$var),prior$mean+1.96*sqrt(prior$var)),col="Grey",ty
lines(xgrid, pp$mean, col="blue")
polygon(x = c(rev(xgrid), xgrid),
        y = c(rev(pp$mean+1.96*sqrt(diag(pp$var))), pp$mean-1.96*sqrt(diag(pp$var))),
        col = "grey",
        border = "grey",
        density = c(20))

hyperParam <- c(1, 1)
sigmaNoise <- 0.1
X <- c(-1, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.94, 0.719, -0.664)
XStar <- 0
prior <- posteriorGP(X, y, hyperParam, sigmaNoise, XStar)
```

```r
pp <- posteriorGP(X,y, hyperParam, sigmaNoise, xgrid)

plot(X, y, xlim=c(-1,1), ylim=c(-3,3), col="Red", main="5. five points, l=1")
points(XStar, prior$mean, col="Black")
lines(c(XStar,XStar), y=c(prior$mean-1.96*sqrt(prior$var),prior$mean+1.96*sqrt(prior$var)),col="Grey",ty
lines(xgrid, pp$mean, col="blue")
polygon(x = c(rev(xgrid), xgrid),
        y = c(rev(pp$mean+1.96*sqrt(diag(pp$var))), pp$mean-1.96*sqrt(diag(pp$var))),
        col = "grey",
        border = "grey",
        density = c(20))

########### lab4q2 ######################

rm(list=ls())
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
GaussianProcess/Code/TempTullinge.csv", header=TRUE, sep=";")

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

idx <- seq(1, 2186, 5)
data0 <- data[idx,]
day0 <- day[idx]
time0 <- time[idx]
se <- function(sigmaf, ell) {
  rval <- function(x, y = NULL) {
    r = sqrt(crossprod(x-y))
    return( sigmaf^2*exp(-(r^2)/(2*ell^2)) )     #sequared exponential error
  }
  class(rval) <- "kernel"
  return(rval)
}

sek = se(sigmaf = 2, ell = 1) # a kernel FUNCTION

sek(1,2)

X <- c(1,3,4)
XStar <- c(2,3,4)
K <- kernelMatrix(kernel = sek, x = X, y = XStar)
K
y <- scale(data0[,2])
X <- scale(time0)

# the residual variance from a simple quadratic regression fit
df <- data.frame(X,y)
lmfit <- lm(y~X+I(X)^2, df)
sigmaNoise <- sd(lmfit$residuals)
sigmaf <- 20
ell <- 0.2

GPfit <- gausspr(X, y,
```

```
                kernel=se, kpar=list(sigmaf=sigmaf, ell=ell),
                var=sigmaNoise, variance.model=TRUE, scaled=F)
meanPred <- predict(GPfit, X)

# plot(X, y, xlab="time", ylab="scaled temp", main=paste0("sigmaf=",sigmaf,", ell=",ell))
# lines(X, meanPred, col="blue", lwd = 2)
knitr::include_graphics("20_02.jpg")
knitr::include_graphics("1_02.jpg")

knitr::include_graphics("20_001.jpg")
knitr::include_graphics("20_1.jpg")
N <- length(y)
sek <- se(sigmaf, ell)
K <- kernelMatrix(sek, X)
L <- t( chol( K+sigmaNoise*diag(N) ) )

# X is a scaled arithmetic sequence from 1 to 2186
# we can just use X as XStar

XStar <- X
kStar <- kernelMatrix(sek, X, XStar)
v <- solve(L) %*% kStar
vf <- kernelMatrix(sek, XStar, XStar) - t(v)%*%v

aa <- sd(data0[,2])
bb <- mean(data0[,2])

dat <- data.frame(x=X, y=meanPred, obs=y,
                  ymin=meanPred-1.96*sqrt(diag(vf)),
                  ymax=meanPred+1.96*sqrt(diag(vf)))
ggplot(dat) +
  geom_ribbon(aes(x=x,y=y, ymin=ymin, ymax=ymax), fill="grey80") +
  geom_line(aes(x=x,y=y), size=1, color="blue") +
  geom_point(aes(x=x,y=obs)) +
  scale_y_continuous(lim=c(-3,3), name="scaled temp") + xlab("scaled time")+
  theme_bw()+
  ggtitle("My Squared Exponential kernel, X=time")

out1 <- sum(dat$obs<dat$ymin) + sum(dat$obs>dat$ymax)
y <- scale(data0[,2])
X <- scale(day0)

# the residual variance from a simple quadratic regression fit
df <- data.frame(X,y)
lmfit <- lm(y~X+I(X)^2, df)
sigmaNoised <- sd(lmfit$residuals)
sigmaf <- 20
ell <- 0.2

GPfit <- gausspr(X, y,
                kernel=se, kpar=list(sigmaf=sigmaf, ell=ell),
                var=sigmaNoised, variance.model=TRUE, scaled=F)
meanPred <- predict(GPfit, X)
```

```
N <- length(y)
sek <- se(sigmaf, ell)
K <- kernelMatrix(sek, X)
L <- t( chol( K+sigmaNoised*diag(N) ) )

XStar <- X
kStar <- kernelMatrix(sek, X, XStar)
v <- solve(L) %*% kStar
vf <- kernelMatrix(sek, XStar, XStar) - t(v)%*%v

dat <- data.frame(x=scale(time0), y=meanPred, obs=y,
                  ymin=meanPred-1.96*sqrt(diag(vf)),
                  ymax=meanPred+1.96*sqrt(diag(vf)))
ggplot(dat) +
  geom_ribbon(aes(x=x,y=y, ymin=ymin, ymax=ymax), fill="grey80") + # var
  geom_line(aes(x=x,y=y), size=1, color="blue") + # mean
  geom_point(aes(x=x,y=obs)) +  # observed data
  scale_y_continuous(lim=c(-3,3), name="scaled temp") + xlab("scaled time(day)")+
  theme_bw()+
  ggtitle("My Squared Exponential kernel, X=day")

out2 <- sum(dat$obs<dat$ymin) + sum(dat$obs>dat$ymax)
ped <- function(sigmaf, ell1, ell2, d) {
  rval <- function(x, y = NULL) {
    r = sqrt(crossprod(x-y))
    d =
    return( sigmaf^2*exp(-2*(sin(pi*r/d)^2)/(ell1^2))*exp(-r^2)/(2*ell2^2) )
  }
  class(rval) <- "kernel"
  return(rval)
}
y <- scale(data0[,2])
X <- scale(time0)
GPfit <- gausspr(X, y, kernel=ped,
                 kpar=list(sigmaf=20, ell1=1, ell2=10,d=365/sd(time0)),
                 var=sigmaNoise, scaled=F)
meanPred <- predict(GPfit, X)

N <- length(y)
pedk <- ped(20,1,10,365/sd(time0))
K <- kernelMatrix(pedk, X)
L <- t( chol( K+sigmaNoise*diag(N) ) )

XStar <- X
kStar <- kernelMatrix(pedk, X, XStar)
v <- solve(L) %*% kStar
vf <- kernelMatrix(pedk, XStar, XStar) - t(v)%*%v

dat <- data.frame(x=scale(time0), y=meanPred, obs=y,
                  ymin=meanPred-1.96*sqrt(diag(vf)),
                  ymax=meanPred+1.96*sqrt(diag(vf)))
ggplot(dat) +
  geom_ribbon(aes(x=x,y=y, ymin=ymin, ymax=ymax), fill="grey80") + # Var
```

```
  geom_line(aes(x=x,y=y), size=1, color="blue") + #MEAN
  geom_point(aes(x=x,y=obs)) +  #OBSERVED DATA
  scale_y_continuous(lim=c(-3,3), name="scaled temp") + xlab("scaled time") +
  theme_bw()+
  ggtitle("My Periodic kernel, X=time")

out3 <- sum(dat$obs<dat$ymin) + sum(dat$obs>dat$ymax)
out <- data.frame(`Q3`=out1, Q4=out2, Q5=out3)
rownames(out) <- "uncovered points"
kable(t(out)) %>%
  kable_styling(latex_options="basic")
########### lab4q3 ######################

rm(list=ls())
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
GaussianProcess/Code/banknoteFraud.csv", header=FALSE, sep=",")
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])

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

mdl = gausspr(fraud ~ varWave + skewWave, data = train)
pred_train <- predict(mdl, train)

dt1 <- cbind(train[,c(1,2,5)], pred_train)
dt11 <- reshape2::melt(dt1, id.vars=c("varWave","skewWave"))

ggplot(dt11) +
  geom_point(aes(x=varWave, y=skewWave, color=value))+
  facet_wrap(~variable)+
  theme_bw()

cat("the confusion matrix of train data is")
conf <- table(pred_train, real=train$fraud)
kable(conf) %>%
  kable_styling(latex_options="basic")
acc <- sum(diag(conf))/sum(conf)
cat("Accuracy of train data is",acc)
pred_test <- predict(mdl, test)
dt1 <- cbind(test[,c(1,2,5)], pred_test)
dt11 <- reshape2::melt(dt1, id.vars=c("varWave","skewWave"))

ggplot(dt11) +
  geom_point(aes(x=varWave, y=skewWave, color=value))+
  facet_wrap(~variable)+
  theme_bw()

cat("the confusion matrix of train data is")
conf <- table(pred_test, real=test$fraud)
kable(conf) %>%
```

```
  kable_styling(latex_options="basic")
acc <- (conf[1,1]+conf[2,2])/sum(conf)
cat("Accuracy of test data is",acc)
# init suitable grid of values for varWave and skewWave
# therefore, check min and max values of these two variables
x1 <- seq(min(train$varWave),max(train$varWave),length=100)
x2 <- seq(min(train$skewWave),max(train$skewWave),length=100)

# Creates 2D matrices for accessing images and 2D matrices
gridPoints <- meshgrid(x1, x2)# output of mashgrid:
# list of every of the 100 points repeated 10 times
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y)) #output
# x1 and x2 in just 2 columns with length 10000

# transform to make new predictions
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]
# Question2 ----------------------------------
# Question2 ====================== why another prediction ???
# Question2 ##############################################
probPreds <- predict(mdl, gridPoints, type="probabilities")

# Plotting for Prob
contour(x = x1, y = x2,
        z = matrix(probPreds[,1],100,byrow = TRUE),
        nlevel = 20,
        xlab = "varWave", ylab = "sekwWave",
        main = 'Prediction of Train data')
points(x = train$varWave[pred_train == 1],
       y = train$skewWave[pred_train == 1], col = "red")
points(x = train$varWave[pred_train == 0],
       y = train$skewWave[pred_train == 0], col = "blue")
legend("bottomright",
       pch = c("o","o"),
       legend = c("Not fraud", "Fraud"),
       col = c("blue", "red"))
mdl = gausspr(fraud ~ ., data = train)
pred_all <- predict(mdl, test)
mdl = gausspr(fraud~.,data=train)
pred_all <- predict(mdl, test)
cat("the confusion matrix of test data is")
conf <- table(pred_all, real_test=test$fraud)
kable(conf) %>%
  kable_styling(latex_options="basic")
acc <- (conf[1,1]+conf[2,2])/sum(conf)
cat("Accuracy of test data is",acc,"\n")
```