732A96 Advanced Machine Learning

LAB 4: GAUSSIAN PROCESSES

Johannes Hedström

STIMA
Department of Computer and Information Science
Linköpings universitet

2024-10-11

Contents

```r
library(knitr)
library(ggplot2)
library(mvtnorm)
library(ggplot2)
library(lubridate)
library(kernlab)
library(pracma)
```

1 1 Implementing GP Regression.

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

$$y = f(x) + \epsilon \text{ with } \epsilon \sim N(0, \sigma_n^2) \text{ and } f \sim GP(0, k(x, x^{'}))$$

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/b means the vector x that solves the equation Ax = b (see p. xvii in the book). This is implemented in R with the help of the function solve. Here is what you need to do: * (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$*$. * sigmaNoise: Noise standard deviation n. * k: Covariance function or kernel. That is, the kernel should be a separate function (see the file GaussianProcesses.R on the course web page).

- (2) Now, let the prior hyperparameters be $\sigma_n = 1$ and $\ell = 0.3$. 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 % probability (pointwise) bands for f.
- (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 % probability (pointwise) bands for f. Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations.
- (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 % probability (pointwise) bands for f.

```r
x=c(-1.0, -0.6, -0.2, 0.4, 0.8)
y=c( 0.768 ,-0.044, -0.940, 0.719, -0.664)

df <- data.frame(x,y)

knitr::kable(t(df))
```

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

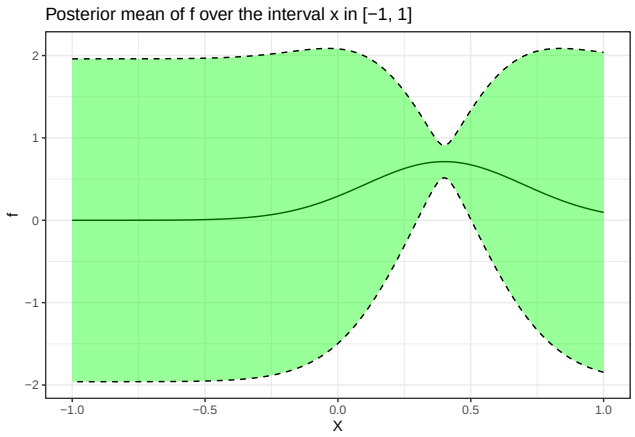
(5) Repeat (4), this time with hyperparameters $\sigma_f = 1$ and $\ell = 1$. Compare the results.

1.1 1

```
# Covariance function from joses code
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)
}


posteriorGP <- function(x,y,XStar,sigmaNoise,k,l=3, sigmaF=1){

  #2 Cholesky on K(X,X) // Transpose to get lower triangle
  K <- k(x,x,sigmaF,l)
  L <- t( chol(K+ diag(sigmaNoise^2,nrow(K))))

  #4 Solving out a
  a <- solve(t(L),solve(L,y))

  # predictive mean
  f_star <- t(k(x,XStar,sigmaF,l)) %*% a

  #6 Solving v
  v <- solve(L,k(x,XStar,sigmaF,l))

  # Predictive variance
  Var_f <- k(XStar,XStar,sigmaF,l) - t(v) %*% v

  #8 # diag to ge the variances
  return(list('f'=f_star,'Var'=diag(Var_f)))

}
```

1.2 2

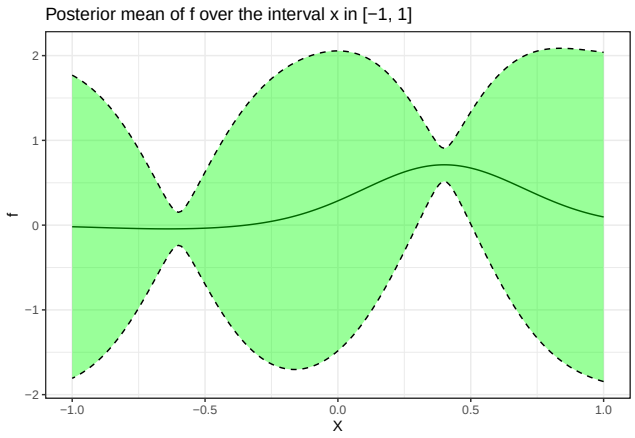
```r
# setting our values
x1 = 0.4
y1 = 0.719
k=SquaredExpKernel
l=0.3
sigmaNoise <- 0.1
XStar <- seq(-1,1,by=0.01)# grid from -1 to 1

# running the function
GP <- posteriorGP(x1,y1,XStar,sigmaNoise,k=SquaredExpKernel,l=l)

# data frame with the values so its easier to plot
GP_df <- data.frame('f'=GP$f,'upper'=GP$f+1.96*sqrt(GP$Var),
                    'lower'=GP$f-1.96*sqrt(GP$Var),'X'=XStar )

# plot
ggplot(GP_df, aes(x=X, y=f)) + geom_line() +
  geom_ribbon(aes(min=lower,max=upper), alpha=0.4, fill='green', color='black', linetype='dashed')+
  theme_bw() + ggtitle('Posterior mean of f over the interval x in [-1, 1]')
```



1.3 3

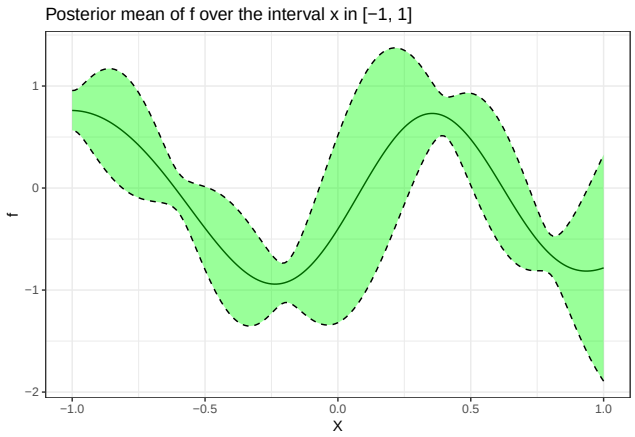
```r
x1 = c(0.4 ,-0.6)
y1 = c(0.719,-0.044)
k=SquaredExpKernel
l=0.3
sigmaNoise <- 0.1
XStar <- seq(-1,1,by=0.01)# grid from -1 to 1

# running the function
GP <- posteriorGP(x1,y1,XStar,sigmaNoise,k=SquaredExpKernel,l=l)
```

```r
# data frame with the values so its easier to plot
GP_df <- data.frame('f'=GP$f,'upper'=GP$f+1.96*sqrt(GP$Var),
                    'lower'=GP$f-1.96*sqrt(GP$Var),'X'=XStar )

# plot
ggplot(GP_df, aes(x=X, y=f)) + geom_line() +
  geom_ribbon(aes(min=lower,max=upper), alpha=0.4, fill='green', color='black', linetype='dashed')+
  theme_bw() + ggtitle('Posterior mean of f over the interval x in [-1, 1]')
```



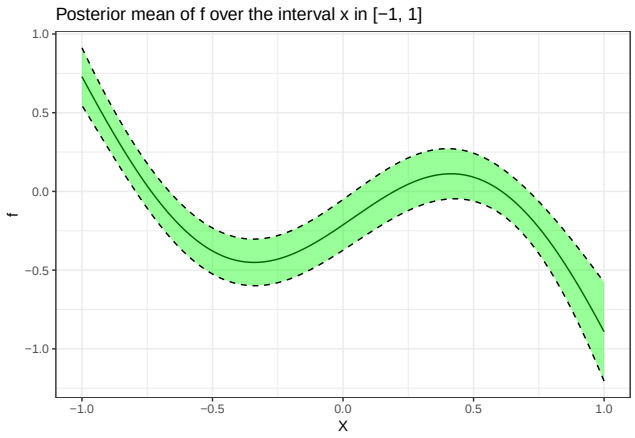
1.4 4

```r
x=c(-1.0, -0.6, -0.2, 0.4, 0.8)
y=c( 0.768 ,-0.044, -0.940, 0.719, -0.664)

df <- data.frame(x,y)
k=SquaredExpKernel
l=0.3
sigmaNoise <- 0.1
XStar <- seq(-1,1,by=0.01)# grid from -1 to 1

# running the function
GP <- posteriorGP(df[,1],df[,2],XStar,sigmaNoise,k=SquaredExpKernel,l=l)

# data frame with the values so its easier to plot
GP_df <- data.frame('f'=GP$f,'upper'=GP$f+1.96*sqrt(GP$Var),
                    'lower'=GP$f-1.96*sqrt(GP$Var),'X'=XStar )

# plot
ggplot(GP_df, aes(x=X, y=f)) + geom_line() +
  geom_ribbon(aes(min=lower,max=upper), alpha=0.4, fill='green', color='black', linetype='dashed')+
  theme_bw() + ggtitle('Posterior mean of f over the interval x in [-1, 1]')
```



1.5 5

```
l=1
sigmaNoise <- 0.1
XStar <- seq(-1,1,by=0.01)# grid from -1 to 1

# running the function
GP <- posteriorGP(df[,1],df[,2],XStar,sigmaNoise,k=SquaredExpKernel,l=l)

# data frame with the values so its easier to plot
GP_df <- data.frame('f'=GP$f,'upper'=GP$f+1.96*sqrt(GP$Var),
                    'lower'=GP$f-1.96*sqrt(GP$Var),'X'=XStar )

# plot
ggplot(GP_df, aes(x=X, y=f)) + geom_line() +
  geom_ribbon(aes(min=lower,max=upper), alpha=0.4, fill='green', color='black', linetype='dashed')+
  theme_bw() + ggtitle('Posterior mean of f over the interval x in [-1, 1]')
```



2 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. You can read the dataset with the command: read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/ Code/TempTullinge.csv", header=TRUE, sep=";") 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 3 KernLabDemo.R available on the course website. You will need to understand it. Now, 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 $X_* = (2, 3, 4)^T$.

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

time <- 1:nrow(tempdata)# time variable

day <- rep(1:365,2190/365) # creating the day variable

fifth_index <- seq(1,length(day),5) # index to chose every fifth value

time <- time[fifth_index]
day <- day[fifth_index]
```

```
# function that returns a kernel function
# so we can evaluate the kernal at inputs

sekf <- function(sigmaF,ell){

  # The kernel from lecture 10, slide 4
  SEK <- function(x1,x2){

      r <- abs(x1-x2) # from lecture 10
    return( sigmaF^2*exp(-(r^2 /(2*ell^2)) ))
  }

  class(SEK) <- "kernel" # class to kernel as kernlab expects this
  return(SEK)
}


evalkernel <- sekf(1,1) # sigmaF = 1 and ell = 2
evalkernel(1,2) # eval x=1, x*=2
```

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

```
X <- c(1, 3, 4) # X
X_star <- c(2,3,4) # X_Star

covm <- kernelMatrix(evalkernel,X,X_star) # Covariance (X,X*)

knitr::kable(covm, caption='Covariance matrix for (X,X*)')
```

Table 2: Covariance matrix for (X,X*)

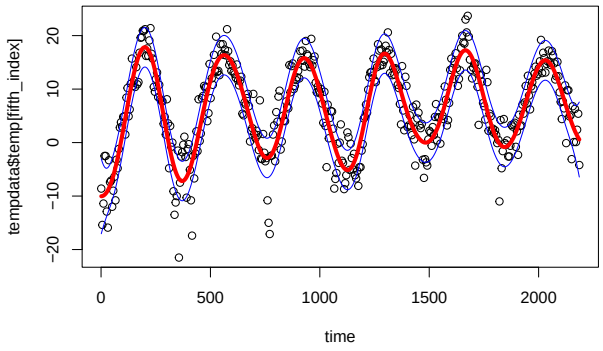
0.6065307	0.1353353	0.0111090
0.6065307	1.0000000	0.6065307
0.1353353	0.6065307	1.0000000

- (2) Consider the following model:

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

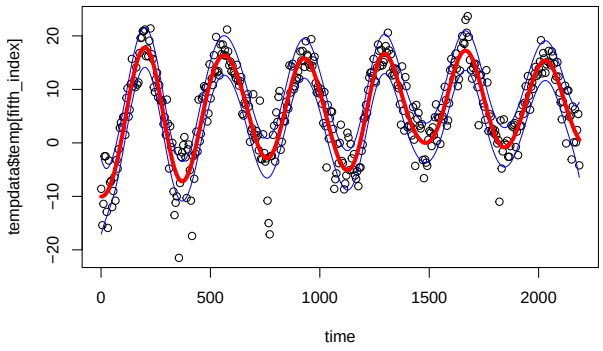
let σ_n^2 be the residual variance from a simple quadratic regression fit (using the lm function in R). Estimate the above Gaussian process regression model using the gausspr function with the squared exponential function from (1) with σ_f= 20 and $\ell = 100$ (use the option scaled=FALSE in the gausspr function, otherwise these σ_f and ℓ values are not suitable). 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). Plot also the 95 % probability (pointwise) bands for f. Play around with different values on σ_f and ℓ (no need to write this in the report though).

```
# Getting the residuals from simple quadratic model
residuals <- lm(tempdata$temp[fifth_index] ~ time + I(time**2))$residuals

GPfit <- gausspr(time,tempdata$temp[fifth_index],kernel=sekf(20,100),
                 scaled=FALSE,var = var(residuals), variance.model = TRUE)

meanPred <- predict(GPfit, time)
plot(time,tempdata$temp[fifth_index])
lines(time, meanPred, col="red", lwd = 4)
lines(time, meanPred+1.96*predict(GPfit,time, type="sdeviation"),col="blue")
lines(time, meanPred-1.96*predict(GPfit,time, type="sdeviation"),col="blue")
```



- (3) Repeat the previous exercise, but now use Algorithm 2.1 on page 19 of Rasmussen and Willams' book to compute the posterior mean and variance of f.

```
gpfit <-posteriorGP(x=time,y=tempdata$temp[fifth_index],XStar=time,
                    sigmaNoise =sd(residuals),k=SquaredExpKernel,l=100,sigmaF=20)


plot(time,tempdata$temp[fifth_index])
lines(time, gpfit$f, col="red", lwd = 4)
lines(time, gpfit$f+1.96* sqrt(gpfit$Var),col="blue")
lines(time,gpfit$f-1.96*sqrt(gpfit$Var),col="blue")
```



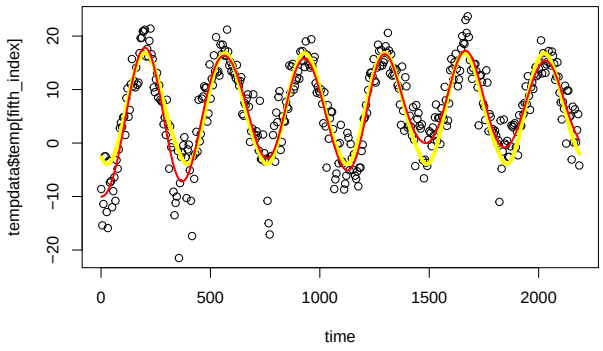
- (4) Consider now the following model:

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

Estimate the model using the gausspr function with the squared exponential function from (1) with $\sigma_f = 20$ and $\ell = 100$ (use the option scaled=FALSE in the gausspr function, otherwise these σ_f and ℓ values are not

suitable). 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?

```
# Getting the residuals from simple quadratic model
residuals <- lm(tempdata$temp[fifth_index] ~ day + I(day**2))$residuals

GPfit4 <- gausspr(day,tempdata$temp[fifth_index],kernel=sekf(20,100),
                  scaled=FALSE,var = var(residuals), variance.model = TRUE)

meanPred4 <- predict(GPfit4, day)
plot(time,tempdata$temp[fifth_index])
lines(time, meanPred4, col="yellow", lwd = 4)
lines(time, meanPred,col="red",lwd = 2)
```



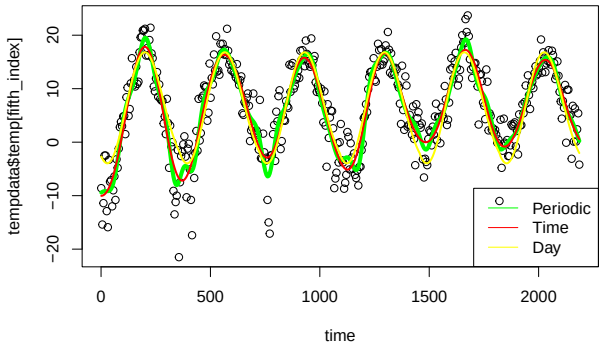
It seems like the model that uses the days(yellow) have a yearly season that stays the same throughout the time period this is because the variable is repeating 1:365 over and over again, the model that uses time as the variable instead changes over time as that variable is a sequence from 1 to 2190.

- (5) Finally, implement the following extension of the squared exponential kernel with a periodic kernel (a.k.a. locally periodic kernel):

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

Note that we have two different length scales in the kernel. Intuitively, l_1 controls the correlation between two days in the same year, 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 $\sigma_f = 20$, $l_1 = 1$, $l_2 = 100$ and d $= 365$. Use the gausspr function with the option scaled=FALSE, otherwise these f , l_1 and l_2 values are not suitable. Compare the fit to the previous two models (with $\sigma_f = 20$ and $\ell = 100$). Discuss the results.

```
lpk <- function(sigmaF,ell1, ell2){

  # The kernel from lab
  sekpk <- function(x1,x2){

      r <- abs(x1-x2) # difference

    return( sigmaF^2*exp(-(2*sin((pi*r)/365)^2)/ell1 )*exp(-0.5*(r^2/ell2^2)))
  }

  class(sekpk) <- "kernel" # class to kernel as kernlab expects this
  return(sekpk)
}



GPfit5 <- gausspr(time,tempdata$temp[fifth_index],kernel=lpk(20,1,100),
                  scaled=FALSE,var = var(residuals), variance.model = TRUE)

meanPred5 <- predict(GPfit5, time)
plot(time,tempdata$temp[fifth_index])
lines(time, meanPred5, col="green", lwd = 4)
#lines(time, meanPred+1.96*predict(GPfit5,time, type="sdeviation"),col="blue")
#lines(time, meanPred-1.96*predict(GPfit5,time, type="sdeviation"),col="blue")
lines(time, meanPred, col="red", lwd = 2)
lines(time, meanPred4, col="yellow", lwd = 2)
# adding a legend so its easier to understand the plot
legend('bottomright', legend=c('Periodic', 'Time', 'Day'),
       lwd=c(1,1,1),col=c('green','red','yellow'))
```



The model that uses the periodic kernel fits the data a bit better, showing less smoothing as its not as rounded as the other two.

3 3 GP Regression with kernlab.

Download the banknote fraud data: 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]) You can read about this dataset here. Choose 1000 observations as training data using the following command (i.e., use the vector SelectTraining to subset the training observations): set.seed(111); SelectTraining <- sample(1:dim(data)1, size = 1000, replace = FALSE)

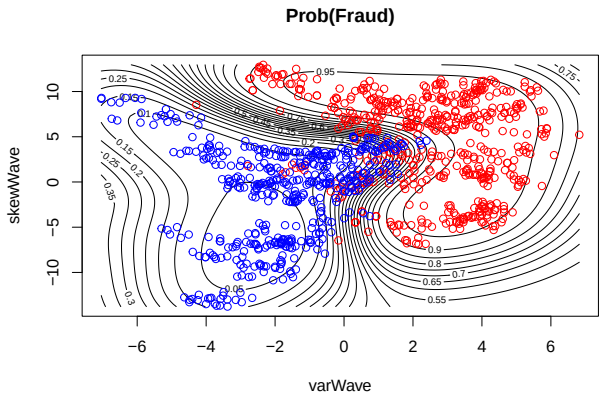
```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud.
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000,replace = FALSE)

trainingd <- data[SelectTraining,]
test <- data[-SelectTraining,]
```

- (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.

```
GPfitfraud <- gausspr(fraud ~  varWave + skewWave , data=trainingd) # running the gausspr on training da
```

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

```
x1 <- seq(min(trainingd[,1]),max(trainingd[,1]),length=100)
x2 <- seq(min(trainingd[,2]),max(trainingd[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$X), c(gridPoints$Y))

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

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main = 'P
points(trainingd[trainingd[,5]==0,1],trainingd[trainingd[,5]==0,2],col="red")
points(trainingd[trainingd[,5]==1,1],trainingd[trainingd[,5]==1,2],col="blue")
```




```
fraudpred <- predict(GPfitfraud,trainingd[,1:2])

table(fraudpred, trainingd[,5]) # confusion matrix
```

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

```
acc <- mean(fraudpred==trainingd[,5])# accuracy

print(acc)
```

```
## [1] 0.941
```

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

```
fraudpstest <- predict(GPfitfraud,test[,1:2])

acct <- mean(fraudpstest==test[,5])# accuracy

print(acct)
```

```
## [1] 0.9247312
```

The accuracy on the test data is a bit lower than on the training data which is expected most of the time, but its still very high. The model isn't over fitting that much.

- (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
GPfitfraudall <- gausspr(fraud ~  . , data=trainingd) # running the gausspr on training data
```

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

```r
fraudpstest3 <- predict(GPfitfraudall,test[,1:4])

acct3 <- mean(fraudpstest3==test[,5])# accuracy


df <- data.frame('1'=acct,'2'=acct3)

colnames(df)<- c('Test accuracy small model','Test accuracy large model')

knitr::kable(df,caption='Difference in test accuracy' )
```

Table 3: Difference in test accuracy

Test accuracy small model	Test accuracy large model
0.9247312	0.9946237

When using all variables we get a really high test accuracy compared to with only using two variables, the large, more complex model is clearly better at making predictions on the test data. The added information from the other variables makes the large model being able to classify better.