732A96: Advanced Machine Learning

LAB 4: GAUSSIAN PROCESSES

Mohammed Bakheet - mohba508
Mudith Chathuranga Silva - kogsi273
Martin Svensson - marsv079
Nour Qweder - nouqw898

17 October 2020

Contents

Statement of Contribution

For this lab the group had a long discussion of the questions after all group members had finished the lab. Then the group submission was created using the structure from one member and merged text from all labs and included part of the discussions.

PART 2.1. Implementing GP Regression.

1) Simulating from the posterior distribution of f using the squared exponential kernel.

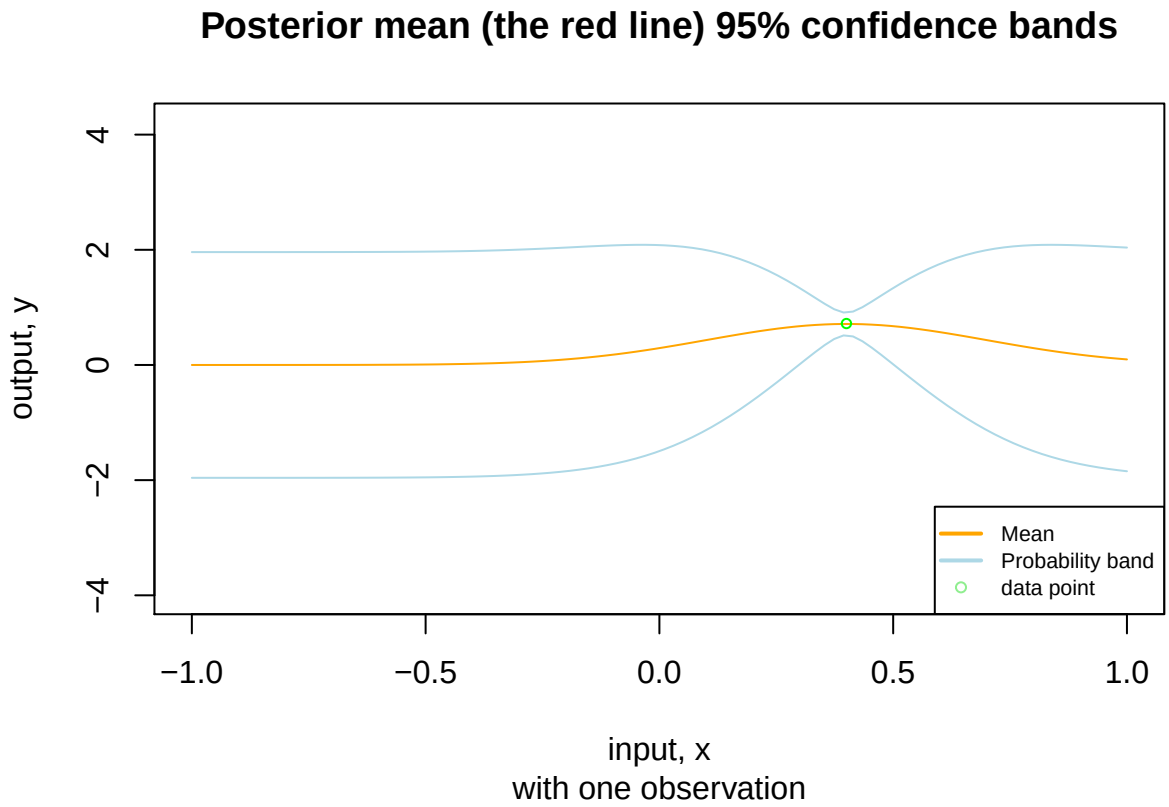
$y = f(x) + \epsilon$
with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$
and $f \sim \mathcal{GP}(0, k(x, x'))$

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 /bb means the vector x that solves the equation A x = b (see p. xvii in the book). This is implemented in R with the help of the function solve.

2) Update this prior

Plot is showing the posterior mean and 95% probability intervals, with one prior observation.

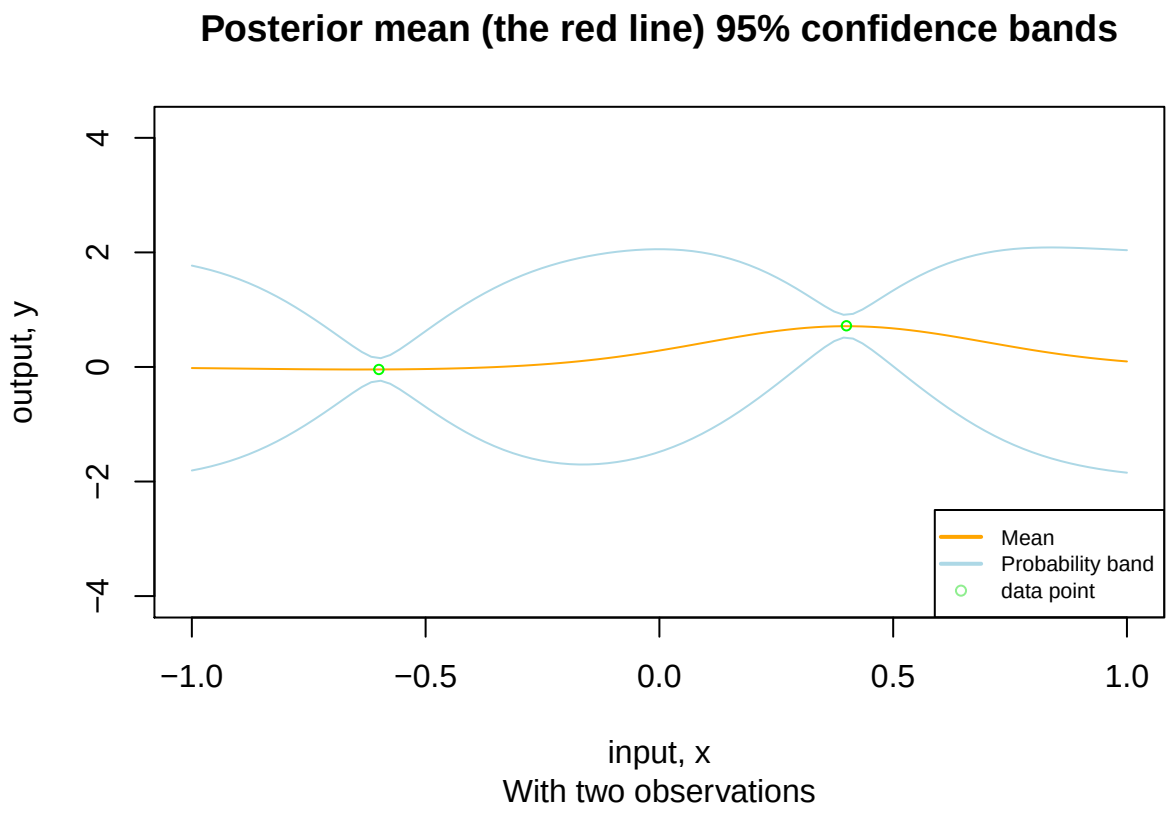
Evaluating the function on *(x,y) = (0.4,0.719)*



By using $\sigma_f = 1$ and $l = 0.3$, it can be seen from the previous plot, after put in one data point, the fit lines has to go near this datapoint, it sort of restriction on where the fit function has to lie in a parameter space.

3) Update your posterior from (2) with another observation:

Here there are two prior observations, showing the function of pointwise probability bands.

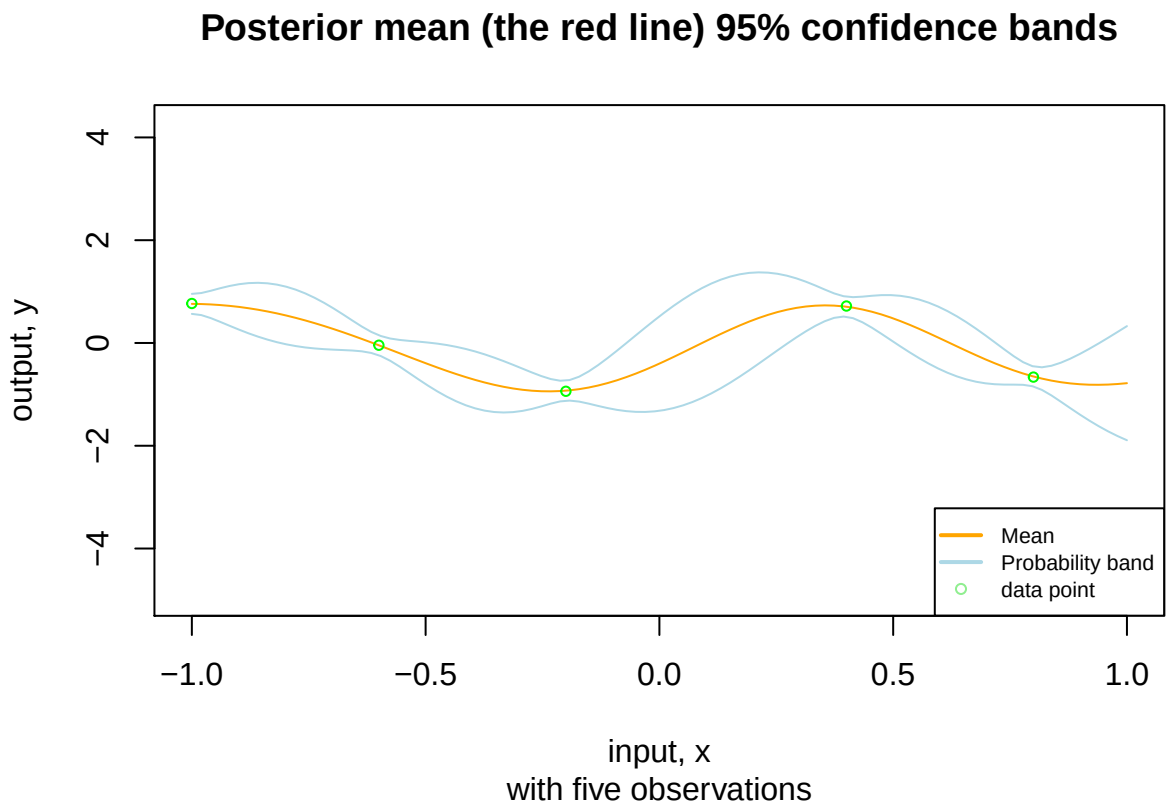


After adding two datapoints, it can be noticed that two datapoints are restricting the fit line of our data.

Table 1: observations

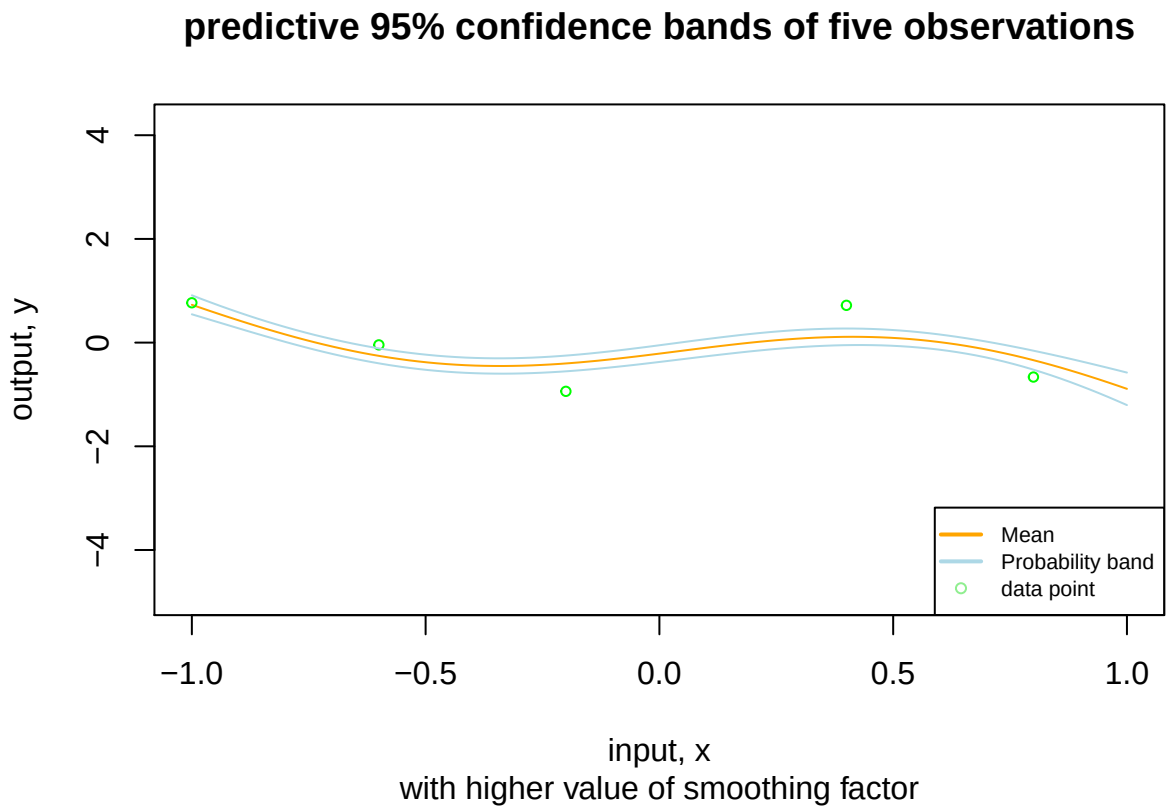
x	0.400	-0.600	-1.000	-0.20	0.800
y	0.719	-0.044	0.768	-0.94	-0.664

4) Compute the posterior distribution of f using all the five data points in the table below. Plot the posterior mean of f. Plot also 95 probability (pointwise) bands for f.



To compare with the previous two steps, as the number of observations increases the probability interval is smoother as well. It can still be seen that the edge without any close observation is very uncertain (to the right). However, the fit function does a much better job. Realize this is only five observations (that is not very much data). which indicates, by adding more observations we start to approximate a fit function with a very little data from Gaussian Process (prior GP).

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



In this time we are updating hyperparameters by using the same value of $\sigma_f = 1$ but the higher value of `smoothing factor` $l = 1$, It can be seen from the plot above that the posterior function is more fluid, so is the confidence interval becomes more accurate at the evaluated points, whereas when using a smaller value of l the confidence band is higher around all values other than the ones our function was evaluated on. So, by increasing the value of l, this will decrease the value of the exponential due to its division by l and taking the negative value of that. Hence, the value of `posteriorGP` function will increase, and this will produce a smoother function. On the contrary, when we use a smaller value of l, this will result in a noisy function, as it can be remarked from the plot above while using a value of 0.3 for l. When we used a value of 0.3 for l and the evaluation was done on one and two points respectively, the confidence band was narrower around the point/s of evaluation, and it was high for the rest of our function values.
as a recap for this part, which indicates that our assumption of GP prior does not fit this chunk of data points.

PART 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.
Create the variable time which records the day number since the start of the dataset (i.e., time= 1, 2, . . ., $365 \times 6 = 2190$). 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.

Setting parameters:

1) Familiarize yourself with the functions gausspr and kernelMatrix

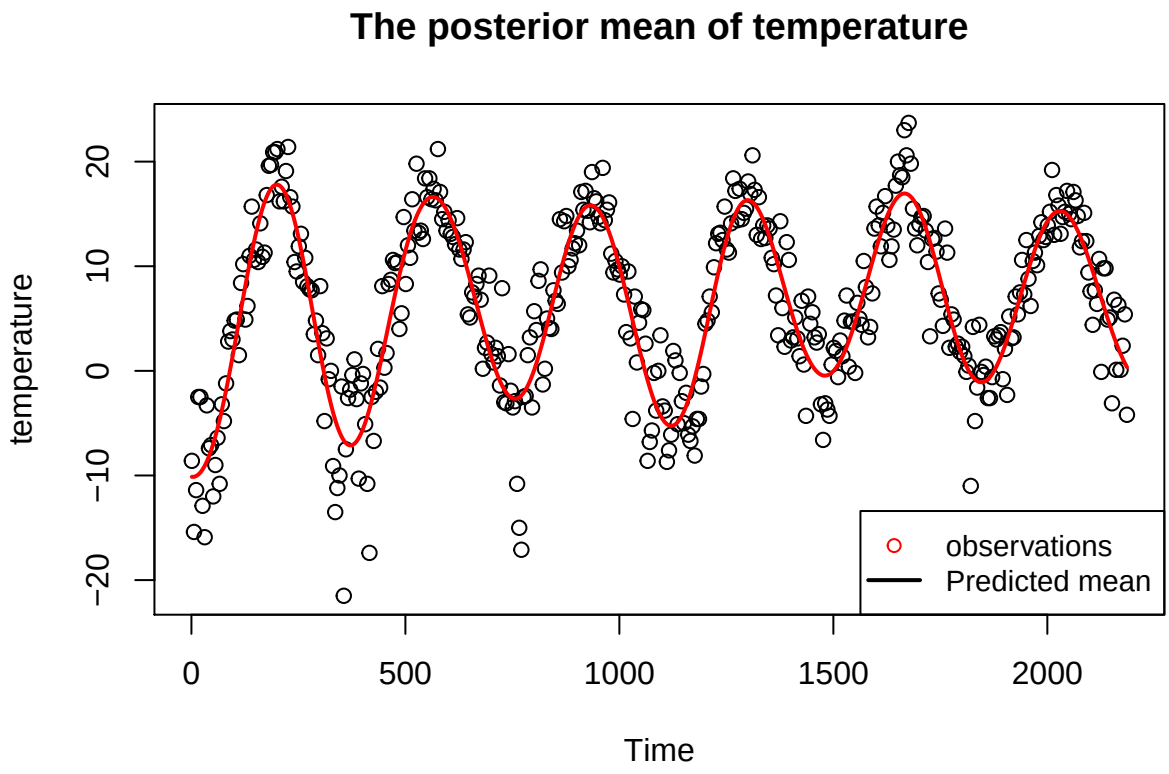
```
## The function evaluation at the point (1,2) is:
## 0.6065307
```

The covariance matrix for `X =` (1, 3, 4) and X_star(2, 3, 4) is:

0.6065307	0.1353353	0.0111090
0.6065307	1.0000000	0.6065307
0.1353353	0.6065307	1.0000000

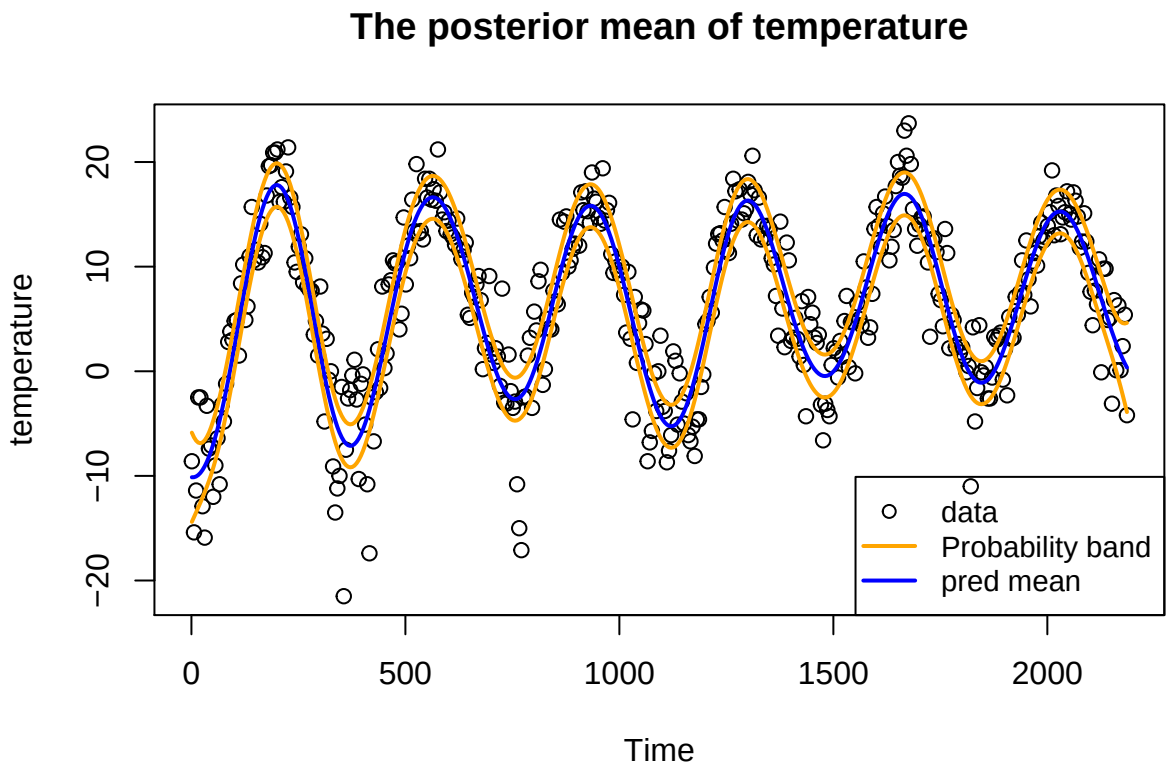
2.2.2) Model using time as input

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



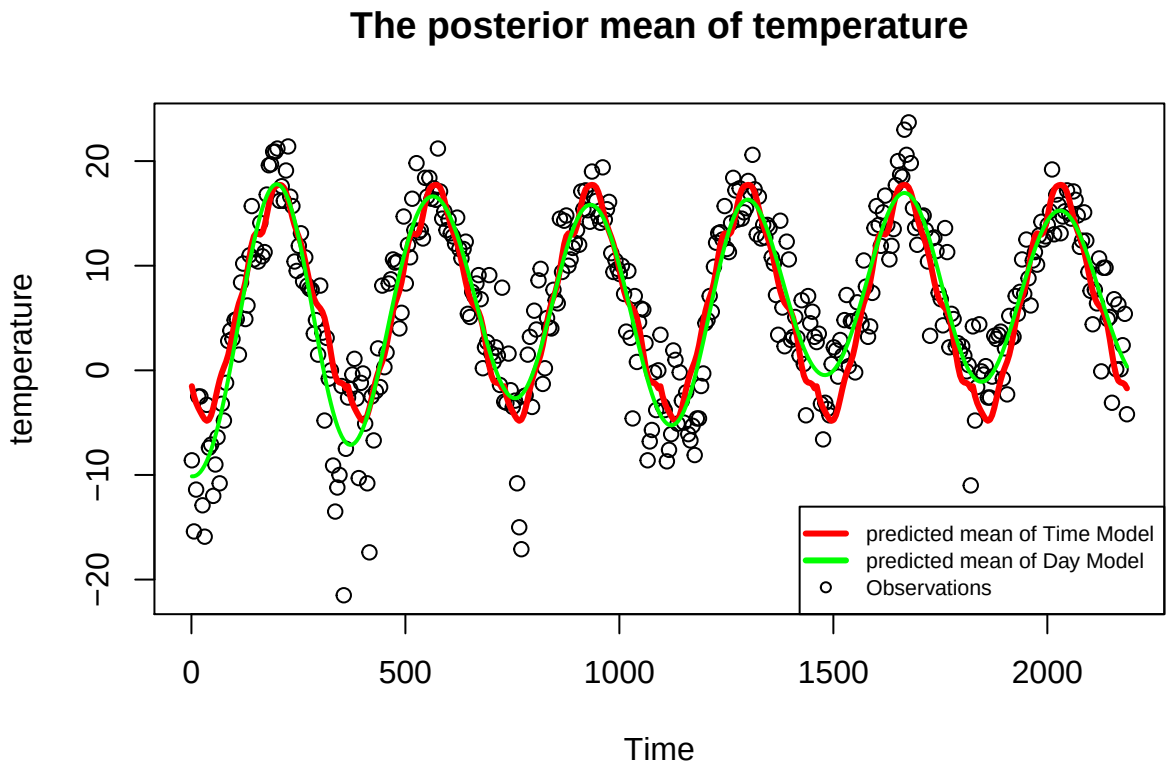
Fitted a second-degree polynomial, it can be seen from the previous figure, the model provides an approximately reasonable model.

2.2.3) Gaussian Process and kernlab Posterior Variance.



Fitted a second-degree polynomial, it can be seen from the previous figure, the model provides an approximately reasonable model, but not as flexible as in 2.2.2. A potential reason for using this instead would be that it appears to be a little bit strong at grasping the outliers.

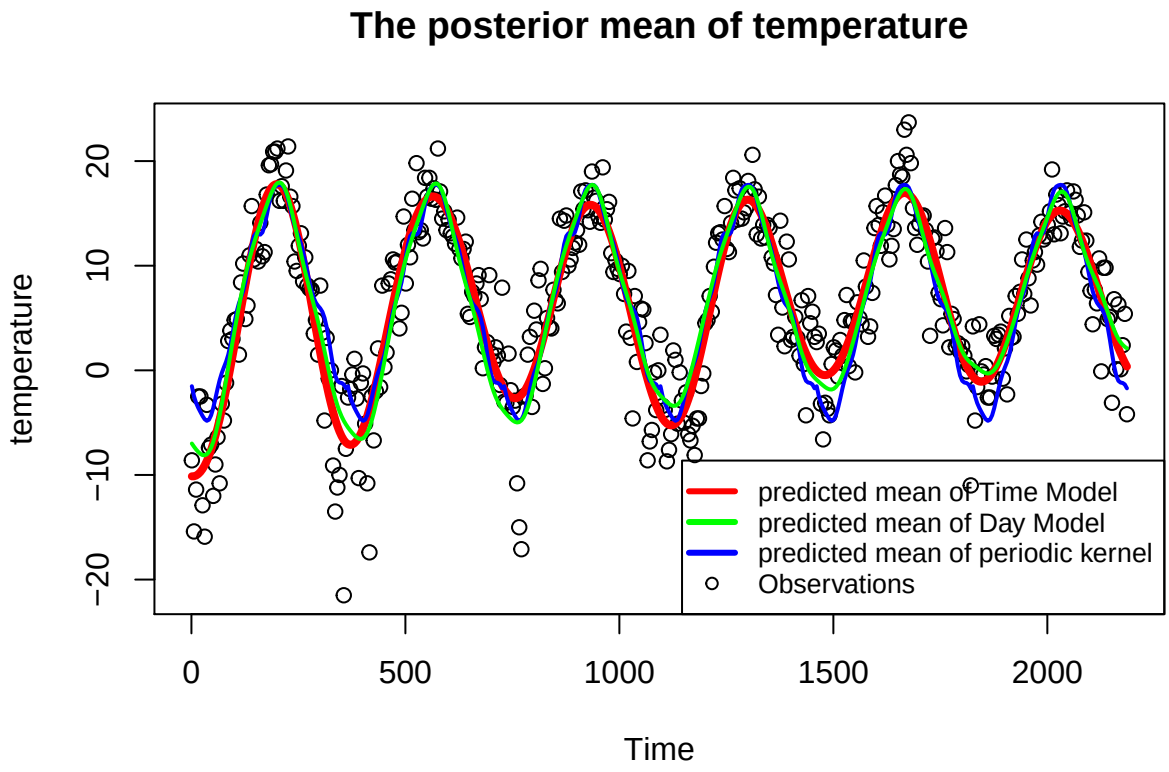
2.2.4) Date Model Gaussian Process.



The prediction band for this model (using the day variable) is narrower than when using the time variable. And that's because when the time variable is used the model is using an increasing value of time with an interval of five. Whereas, when the day is used, the model here has an idea of which day it should predict, due to the repetition of one day in many years, so, the day variable gives the model a better idea of what to predict given a specific day. And that's why the prediction band of the second model is more certain than the first model.

Nonetheless, using the day variables results in a less smooth fit to the data compared to time model, and that's because the day variable considers the seasonality of temperature ruing the entire period. Even though the l value is small, which should assure a smoother fit, but since the day variable doesn't consider the trend and only seasonality (maybe the trend during a single year to be more accurate). The day model is repeating for all years, but non-smooth within each year.

2.2.5) Time Model Gaussian Process and Periodic Kernel.



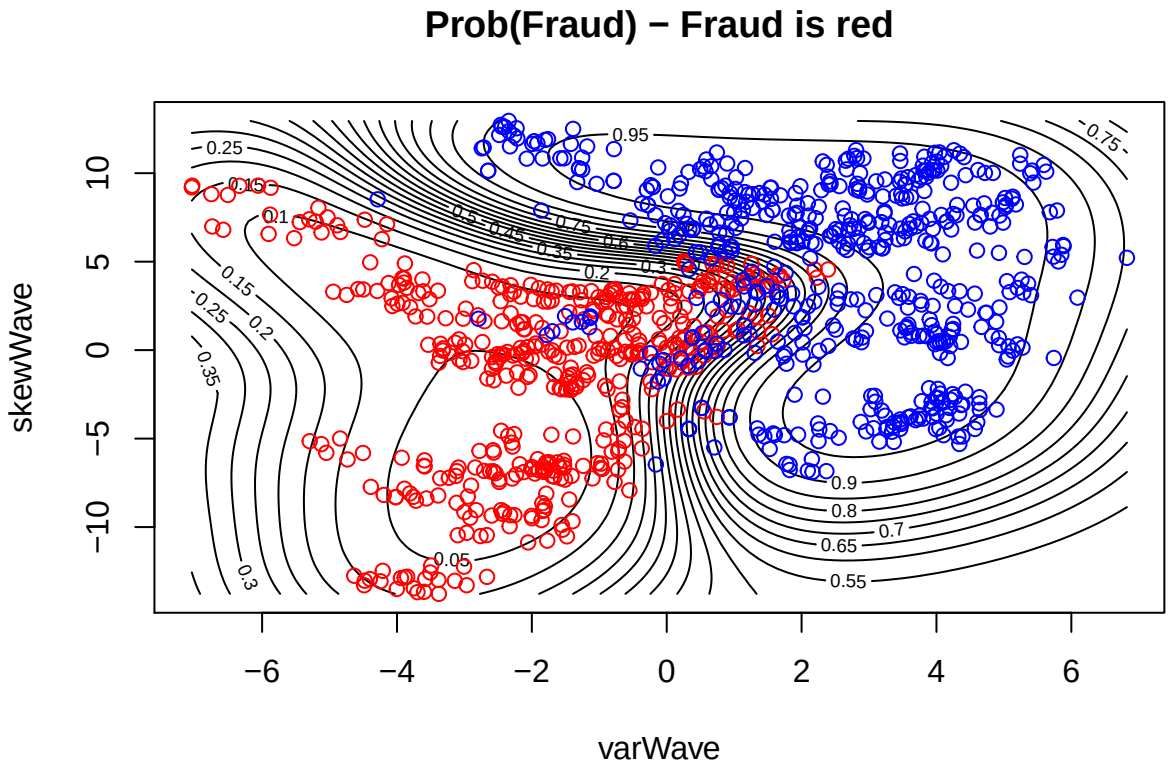
Posterior Mean of f for `day` model seems to repeat the trend for each year since we use the number of `days` per `year` for each observation. Periodic kernel `Posterior-means` appears to be an acceptable fitting line since it covers all seasonal changes and also follows the global trend of winter which can be seen it getting warmer than the past year.

PART 2.3 CLASSIFICATION with kernlab

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.

2.3.1) GP Classification

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




```
## Accuracy rate (training set):
## 0.941
```

	0	1
0	503	18
1	41	438

2.3.2) GP Classification Accuracy using 'varWave' and 'skewWav'

```
## Accuracy rate (training set):
## 0.924731182795699
```

	0	1
0	199	9
1	19	145

By looking to the accuracy rate in this case which is pretty much the same as the accuracy on the training data. Not great, not terrible.

2.3.3) GP Classification Accuracy using all features

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

```
## Accuracy rate (training set):
## 0.994623655913978
```

	0	1
0	216	0
1	2	154

For this case of testing, the test data accuracy is now 0.9946237 which is significantly better than the previous model, that intimates just one wrongly predicted out of 372 samples. In this case, overfitting not likely since test data. As a conclusion, the accuracy of the model is approximately powerful compared to the model when all functions were adopted to train the model. In general, both models have roughly 90% + accuracy, which is relatively reliable accuracy.

Reference

[1] Advanced Machine Learning Course Materials

APPENDIX

```
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(dplyr)
library(kernlab)
library(AtmRay)

## k:  Gaussian process posterior covariance
# Covariance function from helpfile [1]

#################### 2.1.1

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, K, sigmaNoise, xStar,...){

    #' Function for simulating from a posterior distribution.
    #'
    #' Returns simulated values from the posterior distribution.
    #' @param x Vector of training inputs.
    #' @param y Vector of training targets/outputs.
    #' @param XStar Vector of inputs where the posterior distribution is evalueated, i.e. X_star
    #' @param sigmaNoise Noise standard deviation sigma_n.
    #' @param k Covariance function or kernel.
    #' @param ... Input to kernel method.

  K_XX <- K(X,X,...)
  L= t(chol(K_XX + sigmaNoise^2 * diag(length(X))))
  ## predictive mean eq (2.25)

  alpha= solve(t(L) , solve(L,y))
  K_Star<- K(X,xStar,...)
  fStar = t(K_Star) %*% alpha

  ## predictive variance eq. (2.26)
  v=solve(L, K_Star)

  #print(K_XX)
  # returnthe posterior mean and variance of f
  v_fStar <- K(xStar,xStar,...) - t(v) %*% v
```

```
  llik <- -1/2 * t(y) %*% alpha - sum(log(diag(L))) - (length(X))/2 * log(2*pi)
  return(list(post_mean = fStar , varf = diag(v_fStar)))

}



####################

plotGP<- function(XStar, X, y, mean, sd, title, sub_title){

quantile <- qnorm(0.975)
upper_band = mean + quantile * sd
lower_band = mean - quantile *  sd

plot(XStar,
       posterior$post_mean,
       type='l',
       ylim=c(min(mean) - 4,max(mean) + 3.5), col="orange", main = title,ylab = "output, y", xlab = "inp
  points(X, y, col="green", cex=0.65)
  lines(XStar, upper_band, col='light blue')
  lines(XStar, lower_band, col='light blue')
  legend('bottomright',
         legend = c('Mean', 'Probability band', "data point"),
         pch=c(NA, NA, 1),
         lwd=c(2, 2, NA),
         cex = 0.7,
         col = c('orange', 'light blue',"light green"))
}




#################### 2.1.2
sigmaF <- 1
l <- 0.3
XStar <- seq(-1, 1, length.out = 100)
y <- c(0.719)
X <- c(0.4)
sigmaN <- 0.1


posterior <- posteriorGP(X = X,
                         y = y,
                         xStar = XStar,
                         sigmaF, l,
                         sigmaNoise = sigmaN,
                         K = SquaredExpKernel)

#length(XStar) ##100
#length( posterior$post_mean)
p2.1.2<- plotGP(XStar, X, y, posterior$post_mean, sqrt(posterior$varf), title = "Posterior mean (the re
```

```
########################## 2.1.3

sigmaF <- 1
l <- 0.3
XStar <- seq(-1, 1, length.out = 100)
y <- c(0.719, -0.044)
X <- c(0.4, -0.6)
sigmaN <- 0.1

posterior <- posteriorGP(X = X,
                         y = y,
                          xStar = XStar,
                         sigmaF, l,
                         sigmaNoise = sigmaN,
                         K = SquaredExpKernel)

#length(XStar) ##100
#length( posterior$post_mean)

p2.1.3<- plotGP(XStar, X, y, posterior$post_mean, sqrt(posterior$varf), title = "Posterior mean (the re


############################ 2.1.4
y <- c( 0.719, -0.044, 0.768, -0.940, -0.664)
X <- c( 0.4, -0.6, -1, -0.2, 0.8)
df<- data.frame(x=X , y=y)
knitr::kable(t(df), caption = "observations",  format = "latex")


y <- c(0.719, -0.044, 0.768, -0.940, -0.664)
X <- c(0.4, -0.6, -1, -0.2, 0.8)

posterior <- posteriorGP(X = X,
                         y = y,
                          xStar = XStar,
                         sigmaF, l,
                         sigmaNoise = sigmaN,
                         K = SquaredExpKernel)

#length(XStar) ##100
#length( posterior$post_mean)

p2.1.4<- plotGP(XStar, X, y, posterior$post_mean, sqrt(posterior$varf), title = "Posterior mean (the re

#################### 2.1.5
sigmaF <- 1
l <- 1

XStar <- seq(-1, 1, length.out = 100)
y <- c(0.719, -0.044, 0.768, -0.940, -0.664)
X <- c(0.4, -0.6, -1, -0.2, 0.8)

posterior <- posteriorGP(X = X,
```

```
                          y = y,
                          xStar = XStar,
                          sigmaF, l,
                          sigmaNoise = sigmaN,
                          K = SquaredExpKernel)

p2.1.5<- plotGP(XStar, X, y, posterior$post_mean, sqrt(posterior$varf), title = "predictive 95% confide

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

time<- c(1:2190) #time = c(1:nrow(temp_df))
day<- rep(1:365,6)

## Now we are sampling from the previous datpoints by picking up a spicific positions
samples<- seq(from = 1, to = 2186, by = 5)
sim_time<- time[samples]
sim_day<- day[samples]
sim_temp=temp_df$temp[sim_time]



####### 2.2.1

library(lubridate)
library(kernlab)
library(AtmRay)

squareExponential<- function(sigmaF = 1, ell = 1)
{
  ker <- SquaredExpKernel <- function(x, y = NULL) {
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x-y[i])/ell)^2)
    }
    return(K)
  }
  class(ker) <- "kernel"
  return(ker)
}
x1 = c(1)
x2 = c(2)

X = as.vector(c(1, 3, 4))
X_star = as.vector(c(2, 3, 4))

#Evaluating the function
square_exp_Func = squareExponential(sigmaF = 1, ell = 1)



K = kernelMatrix(kernel = squareExponential() , x = X, y = X_star)
```

```r
cat('The function evaluation at the point (1,2) is: ',square_exp_Func(x1, x2), sep = "\n")
knitr::kable(K, format = "latex" )
####### 2.2.2
library(kernlab)

# Fitting a polynomial due to characteristics of the data
fit <- lm(formula = sim_temp ~ sim_day + I(sim_day^2))
#summary(fit)
sigma_noice = sd(fit$residuals)

#fit = lm(temp ~ time+time^2)

sigmaF=20
## smoothing factor
l=0.2
SEkernel <- squareExponential(ell = l, sigmaF = sigmaF)
GPfit <- gausspr(sim_time, sim_temp, kernel = SEkernel, var = sigma_noice^2)
pred_mean <- predict(GPfit, sim_time) # Predicting the training data. To plot the fit.


plot(sim_time, sim_temp,
     ylab='temperature',
     xlab='Time',
     type="p",
     pch=1,
     main="The posterior mean of temperature" )
lines(sim_time,
      pred_mean,
      type="l",
      lwd=2,
      col="red")
legend( "bottomright",legend = c('observations', 'Predicted mean'),
              col = c('red','black'),
       lwd = c(NA,2),
       pch = c(1,NA),
       cex = 0.9,
       xpd=T)

####### 2.2.3
posterior = posteriorGP(scale(sim_time),
                        scale(sim_temp),
                        scale(sim_time),
                        sigmaNoise = sigma_noice,
                        K = SquaredExpKernel,
                        sigmaF=20,
                        l=0.2)
post_var = posterior$varf
post_mean = posterior$post_mean


quantile <- qnorm(0.975)
#quantile <- qnorm(0.025)
upper_band = pred_mean + quantile*sqrt(post_var)
```

```r
lower_band = pred_mean - quantile*sqrt(post_var)

plot(sim_time, sim_temp,
     ylab='temperature', xlab='Time',
     type="p",
     main="The posterior mean of temperature" )
lines(sim_time, upper_band, lwd=2, col="orange")
lines(sim_time,
      pred_mean,type="l",
      lwd=2,
      col="blue")
lines(sim_time, lower_band, lwd=2, col="orange")

legend( "bottomright",legend = c('data', 'Probability band', "pred mean"),
       col = c("black",'orange','blue'),
       lwd = c(NA,2,2),
       pch = c(1,NA,NA),
       cex = 0.9,
       xpd=T)



####### 2.2.4
## Consider now the following model: temp = f(day) + epsilon with epsilon ~ N(0, sigmaN^2) and f~GP(0,
## Estimate the model using the squared exponential function with sigmaF=20 and l=0.2. Superimpose the
## from this model on the posterior mean from the model in (2). Note that this plot should also have ti
## horizontal axis. Compare the results of both models. What are the pros and cons of each model?


sigmaF = 20
l = 0.2

SEkernel2_4 <- squareExponential(ell = l, sigmaF = sigmaF)
GPfit2_4 <- gausspr(sim_day,
                    sim_temp,
                    kernel = SEkernel2_4,
                    var = sigma_noice^2)
pred_mean2_4 <- predict(GPfit2_4, sim_day) # Predicting the training data. To plot the fit.
plot(sim_time,
     sim_temp,
     type= "p",
     ylab='temperature', xlab='Time',
     main="The posterior mean of temperature" )

lines(sim_time,
      pred_mean2_4,
      type="l",
      lwd=3,
      col="red")

lines(sim_time,
      pred_mean,
      type="l",
```

```
      lwd=2,
      col="green")


legend('bottomright',
       legend = c('predicted mean of Time Model',
                  'predicted mean of Day Model',
                  'Observations'),

       col = c('red','green','black'),
       lwd = c(3,3,NA),
       pch = c(NA,NA,1),
       cex = 0.7,
       xpd=T)


####### 2.2.5

## Finally, implement a generalization of the periodic kernel given in the lectures. Note that Note tha
## length scales here, and `2 controls the correlation between the same day in different years. Estimat
## time variable with this kernel and hyperparameters sigmaF = 20, l1 = 1, l2 = 10 and d = 365/sd(time)
## The reason for the rather strange period here is that kernlab standardizes the inputs to have standa
## Compare the fit to the previous two models (with sigmaF = 20 and l = 0.2). Discuss the results.

periodic_Kernel <- function(sigmaF=1,l1=1, l2=10, d){
  periodic = function(x, xStar) {
    diff = abs(x-xStar)
    t1 <- exp(-(2 * sin(pi * diff / d)^2) / l1^2)
    t2 <- exp(-0.5 * diff^2 / l2^2)
    return(sigmaF^2 * t1 * t2)
  }
  class(periodic)='kernel'
  return(periodic)
}

sigmaF=20
l1=1
l2=10
d=365/sd(sim_time)


SEkernel2_5 <- periodic_Kernel(l1, l2,d = d, sigmaF = sigmaF)
GPfit2_5 <- gausspr(sim_time,
                    sim_temp,
                    kernel = SEkernel2_5,
                    var = sigma_noice^2)

pred_mean2_5 <- predict(GPfit2_5, sim_time) # Predicting the training data. To plot the fit.
plot(sim_time,
     sim_temp,
     type="p",
     ylab='temperature', xlab='Time',
     main="The posterior mean of temperature" )
```

```r
lines(sim_time,
      pred_mean,
      type="l",
      lwd=4,
      col="red")
lines(sim_time,
      pred_mean2_4,
      type="l",
      lwd=2,
     col="blue")
lines(sim_time,
      pred_mean2_5,
      type="l",
      lwd=2,
      col="green")
legend('bottomright',
       legend = c('predicted mean of Time Model',
                  'predicted mean of Day Model',
                  'predicted mean of periodic kernel',
                  'Observations'),

       col = c('red','green',"blue", "black"),
       lwd = c(3,3,3,NA),
       pch = c(NA,NA,NA,1),
       cex = 0.8,
       xpd=T)


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)
id <- sample(1:dim(data)[1], size = 1000,replace = FALSE)
training<-data[id,]
testing <-data[-id,]
####### 2.3.1)
library(kernlab)
#Use the R package kernlab to fit a Gaussian process classification model for fraud on the training dat
## and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contour
## over a suitable grid of values of varWave and skewWave. Overlay the training data for fraud=1 (as bl
## (as red points). You can reuse code from the file KernLabDemo.R available on course website. Compute
## for the classifier and its accuracy.

model_3.1 <- gausspr(fraud ~ varWave + skewWave, data=training) #

# predict on the training set
pred_3.1 = predict(model_3.1,newdata= training)
confuse_mat=table(pred_3.1, training[,5]) # confusion matrix


# class probabilities
```

```
probPreds_3.1 <- predict(model_3.1,newdata= training, type="probabilities")
x1 <- seq(min(training$varWave),max(training$varWave),length=100)
x2 <- seq(min(training$skewWave),max(training$skewWave),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(training[,c("varWave", "skewWave")])
probPreds_3.1 <- predict(model_3.1, gridPoints, type="probabilities")

# Plotting for Prob(frud)
contour(x1,x2,matrix(probPreds_3.1[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main

points(training[training[,5]==1,1],training[training[,5]==1,2],col="red")
points(training[training[,5]==0,1],training[training[,5]==0,2],col="blue")




acc_rate = sum(diag(confuse_mat))/sum(confuse_mat)
cat(paste("Accuracy rate (training set): ", acc_rate, sep = "\n"))
knitr::kable(confuse_mat, format = "latex")



####### 2.3.2)


# predict on the training set
pred_3.2 = predict(model_3.1,newdata= testing)
confuse_mat_test=table(pred_3.2, testing[,5]) # confusion matrix
acc_rate_test = sum(diag(confuse_mat_test))/sum(confuse_mat_test)
cat(paste("Accuracy rate (training set): ", acc_rate_test, sep = "\n"))
knitr::kable(confuse_mat_test, format = "latex")




####### 2.3.3)


model_3.3 <- gausspr(fraud ~ ., data=training) # mode = 54
#summary(model_3.3)


# predict on the training set
pred_3.3 = predict(model_3.3,newdata= testing)
confuse_mat_test3.3=table(pred_3.3, testing[,5]) # confusion matrix
acc_rate_test3.3 = sum(diag(confuse_mat_test3.3))/sum(confuse_mat_test3.3)
cat(paste("Accuracy rate (training set): ", acc_rate_test3.3, sep = "\n"))
knitr::kable(confuse_mat_test3.3, format = "latex", digits = 6)

# Creates code appendix
```

```
# from https://yihui.name/en/2018/09/code-appendix/
```