Advanced Machine Learning

Lab 3

Rasmus Holm

2017-10-01

```
library(kernlab)
library(AtmRay)
```

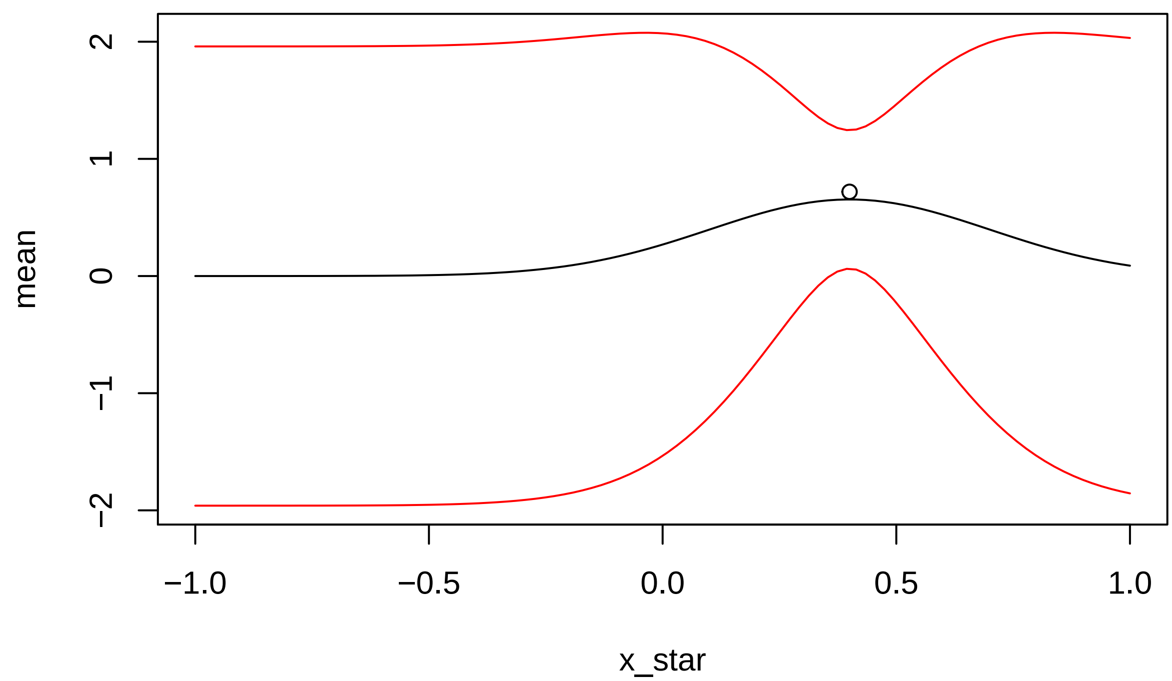
1)

a)

```
squared_exp_kernel <- function(sigma, l){
    function(x1, x2) {
        n1 <- length(x1)
        n2 <- length(x2)
        K <- matrix(NA, n1, n2)

        for (i in 1:n2){
            K[, i] <- sigma^2 * exp(-0.5 * ((x1 - x2[i]) / l)^2)
        }

        K
    }
}

posterior_gp <- function(x_new, x, y, noise, kernel) {
    Kxx <- kernel(x, x)
    Kxs <- kernel(x, x_new)
    Kss <- kernel(x_new, x_new)

    L <- t(chol(Kxx + diag(noise, nrow(Kxx), ncol(Kxx))))
    alpha <- solve(t(L), solve(L, y))
    mean <- t(Kxs) %*% alpha
    v <- solve(L, Kxs)
    covariance <- Kss - t(v) %*% v
    list(mean=mean, variance=covariance)
}

plot_gp <- function(posterior, x_star) {
    mean <- posterior$mean
    lower_band <- mean - 1.96 * sqrt(diag(posterior$variance))
    upper_band <- mean + 1.96 * sqrt(diag(posterior$variance))

    plot(x_star, mean, type="l", ylim=c(min(lower_band), max(upper_band)))
    lines(x_star, lower_band, col="red")
    lines(x_star, upper_band, col="red")
}
```

b)

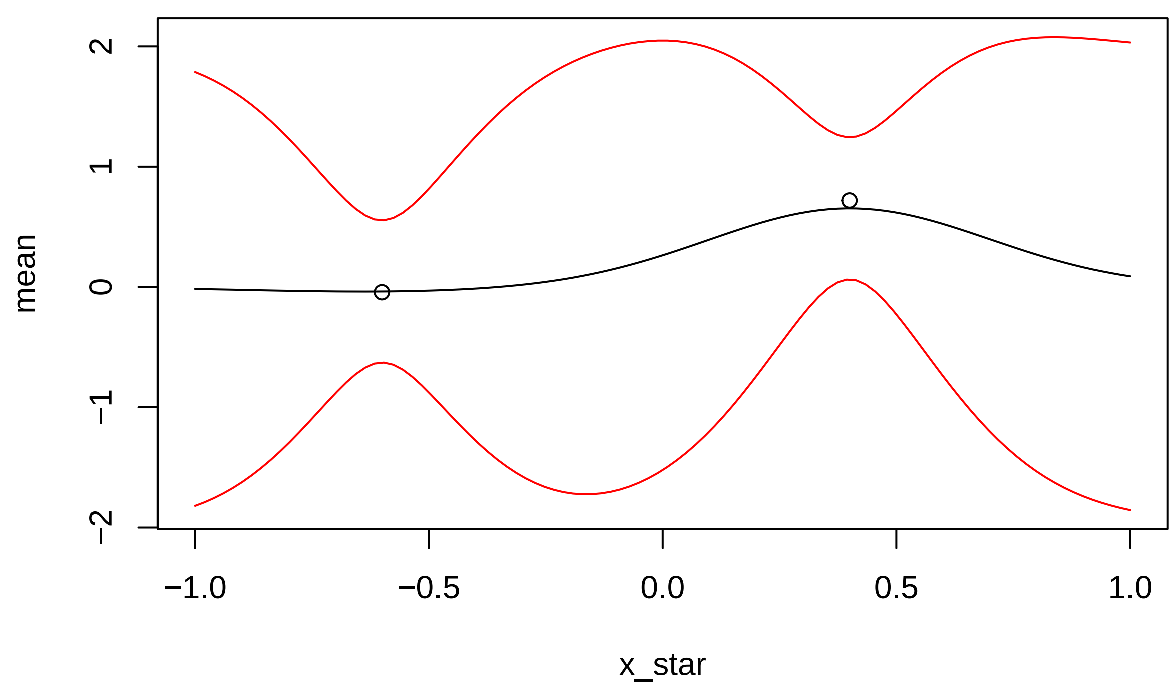
```
kernel <- squared_exp_kernel(1, 0.3)
x_star <- seq(-1, 1, length=100)
x <- c(0.4)
y <- c(0.719)
noise <- 0.1

pgp <- posterior_gp(x_star, x, y, noise, kernel)

plot_gp(pgp, x_star)
points(x, y)
```



c)

```
kernel <- squared_exp_kernel(1, 0.3)
x_star <- seq(-1, 1, length=100)
x <- c(0.4)
y <- c(0.719)
noise <- 0.1

x_new <- c(-0.6)
y_new <- c(-0.044)
```

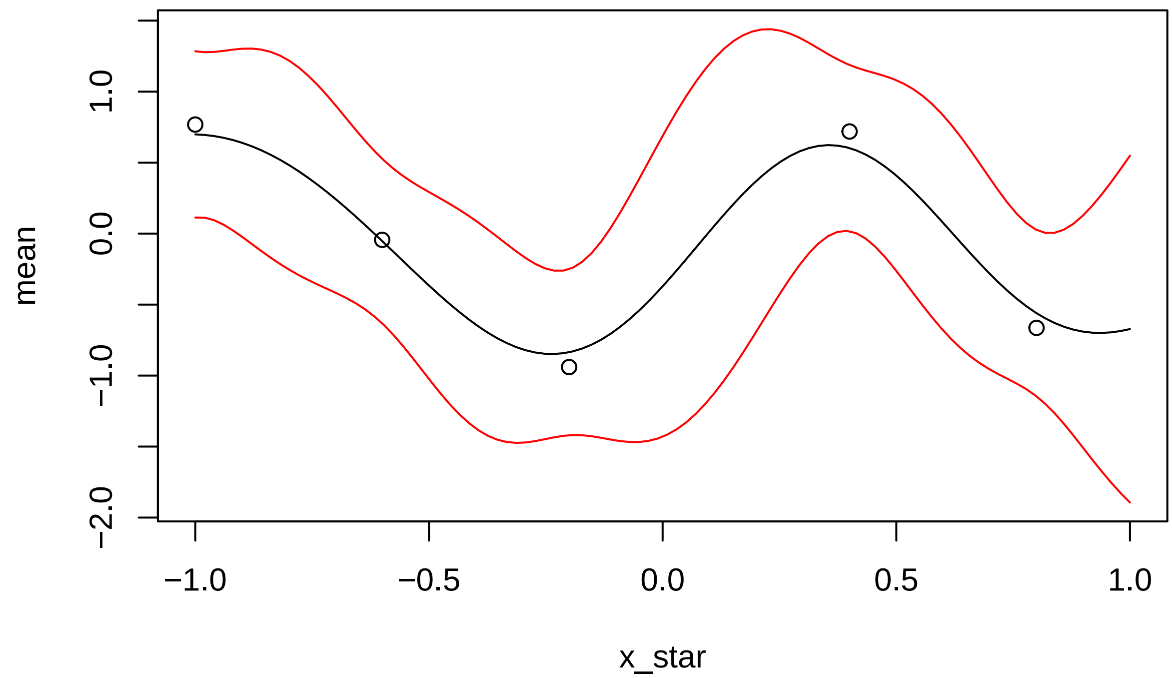
```r
prior_mean <- pgp$mean
prior_covariance <- pgp$variance

## Should it be noise here?
posterior_mean <- prior_mean +
    kernel(x_star, x_new) %*% solve(kernel(x_new, x_new) + diag(noise, length(x_new), length(x_new))) %*
posterior_covariance <- prior_covariance -
    kernel(x_star, x_new) %*% solve(kernel(x_new, x_new) + diag(noise, length(x_new), length(x_new))) %*

mean <- posterior_mean
lower_band <- mean - 1.96 * sqrt(diag(posterior_covariance))
upper_band <- mean + 1.96 * sqrt(diag(posterior_covariance))

plot(x_star, mean, type="l", ylim=c(min(lower_band), max(upper_band)))
points(c(x, x_new), c(y, y_new))
lines(x_star, lower_band, col="red")
lines(x_star, upper_band, col="red")
```



d)

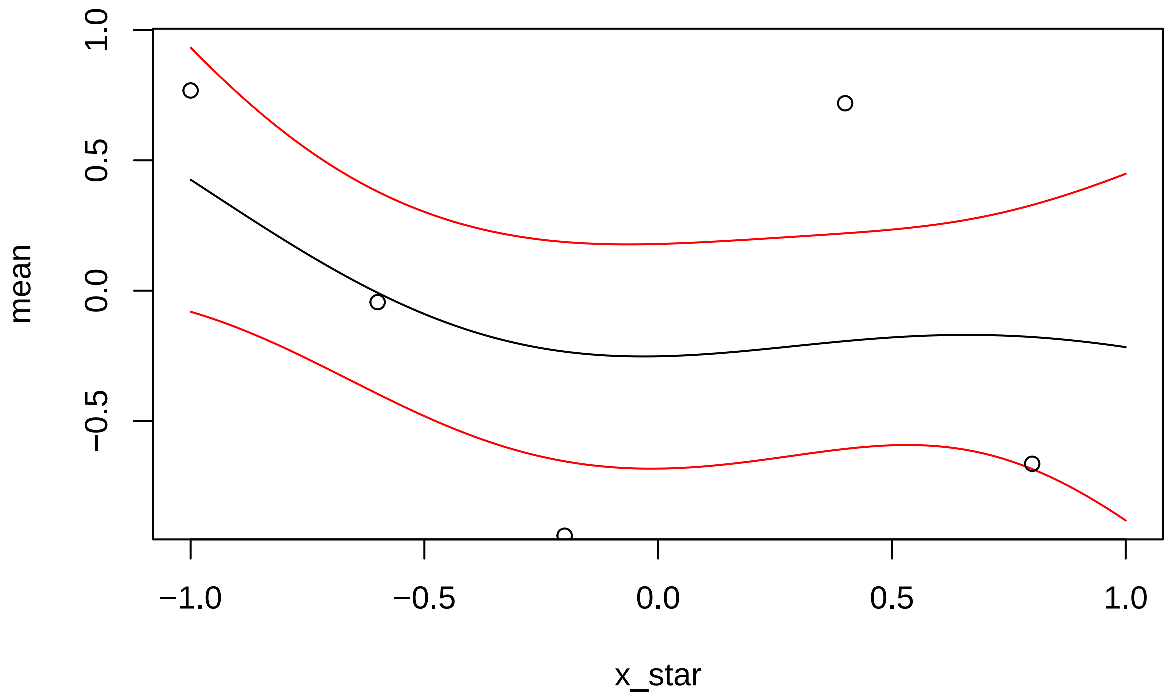
```r
kernel <- squared_exp_kernel(1, 0.3)
x_star <- seq(-1, 1, length=100)
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
```

```
noise <- 0.1

pgp <- posterior_gp(x_star, x, y, noise, kernel)

plot_gp(pgp, x_star)
points(x, y)
```



e)

```
kernel <- squared_exp_kernel(1, 1)
x_star <- seq(-1, 1, length=100)
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
noise <- 0.1

pgp <- posterior_gp(x_star, x, y, noise, kernel)

plot_gp(pgp, x_star)
points(x, y)
```



1.0
0.5
0.0
−0.5
mean
−1.0
−0.5
0.0
0.5
1.0
x_star

2)

```r
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
                 header=TRUE, sep=";")
data$time<- 1:nrow(data)
data$day <- 0:(nrow(data) - 1) %% 365 + 1

thinned_data <- data[(data$time - 1) %% 5 == 0, ]
```

```r
single_squared_exp_kernel <- function(sigma, l) {
    f <- function(x1, x2) {
        sigma^2 * exp(-0.5 * ((x1 - x2) / l)^2)
    }
    class(f) <- "kernel"
    f
}

single_periodic_kernel <- function(sigma, l1, l2, d) {
    f <- function(x1, x2) {
        sigma^2 *
            exp(-2 * sin(pi * abs(x1 - x2) / d) / l1^2) *
            exp(- (1 / 2) * ((x1 - x2) / l)^2)
    }
    class(f) <- "kernel"
    f
}
```

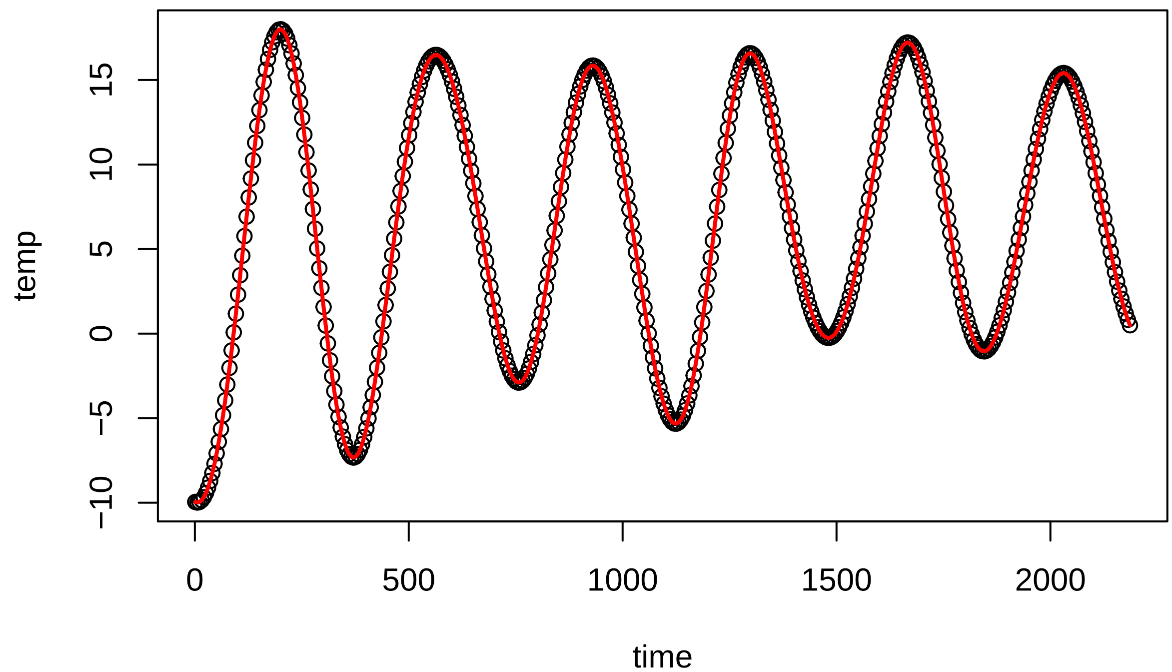
a)

```r
x <- c(1, 3, 4)
x_star <- c(2, 3, 4)

kernel <- single_squared_exp_kernel(1, 1)
kernelMatrix(kernel, x, x_star)
#> 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
```

b)

```r
kernel <- single_squared_exp_kernel(20, 0.2)

lm_fit <- lm(temp ~ poly(time, 2), thinned_data)
sigma <- sd(resid(lm_fit))

gp_fit <- gausspr(temp ~ time, thinned_data, kernel=kernel, var=sigma)
predicted <- predict(gp_fit, thinned_data)
```

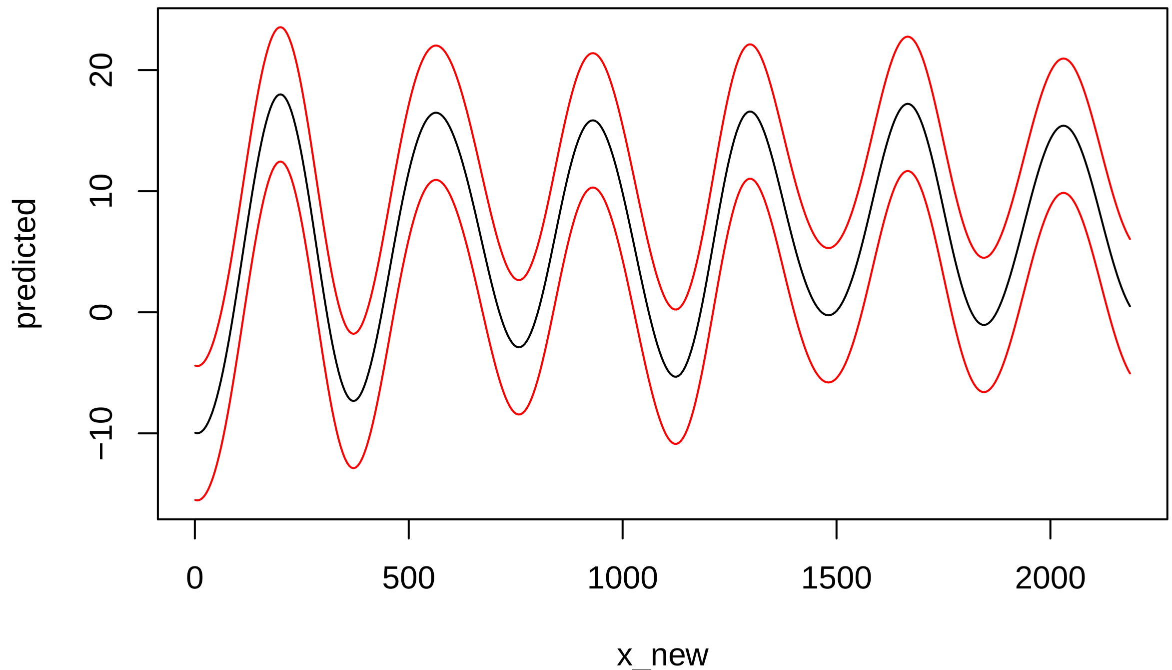
```r
plot(thinned_data$time, predicted, type="p", xlab="time", ylab="temp")
lines(thinned_data$time, predicted, col="red", lwd=2)
```



c)

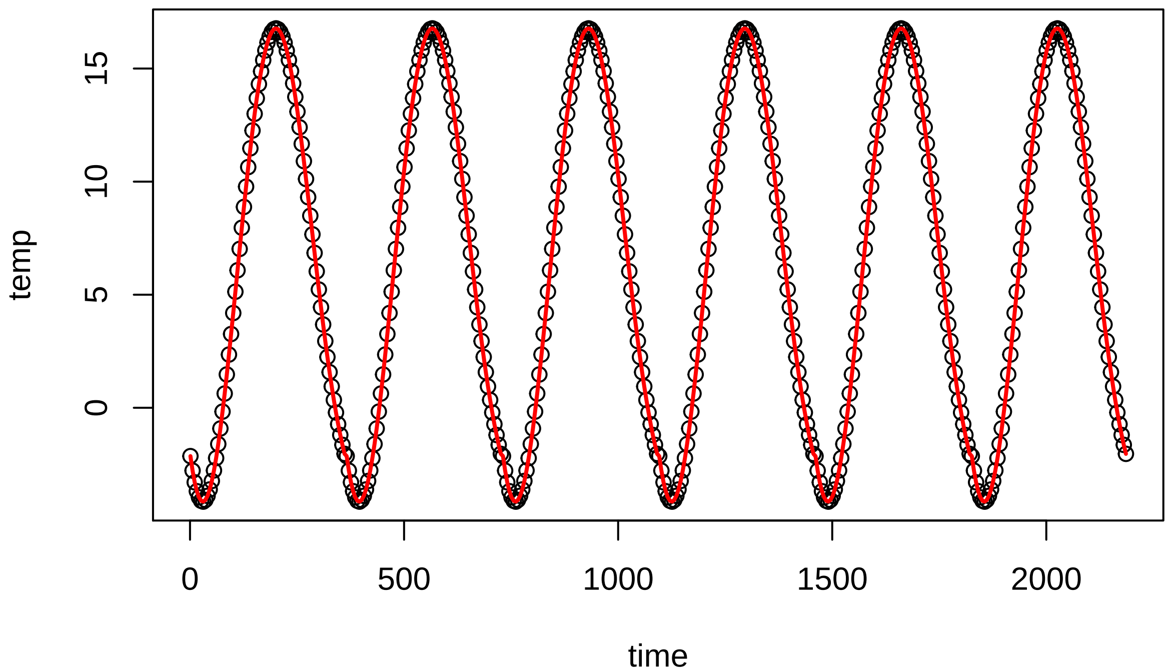
```r
prediction <- function(x_new, x, y, kernel, noise) {
    K <- kernel(x_new, x)
    S <- solve(kernel(x, x) + diag(noise, length(x), length(x)))
    pred <- K %*% S %*% y
    sigma <- kernel(x_new, x_new) - K %*% S %*% t(K)
    list(pred=pred, variance=sigma)
}

kernel <- squared_exp_kernel(20, 0.2)
x_new <- thinned_data$time
x <- thinned_data$time
y <- thinned_data$temp

preds <- prediction(x_new, x, y, kernel, sigma)

mean <- predicted
lower_band <- mean - 1.96 * sqrt(diag(preds$variance))
upper_band <- mean + 1.96 * sqrt(diag(preds$variance))
```

```r
plot(x_new, predicted, ylim=c(min(lower_band), max(upper_band)), type="l")
lines(x_new, lower_band, col="red")
lines(x_new, upper_band, col="red")
```



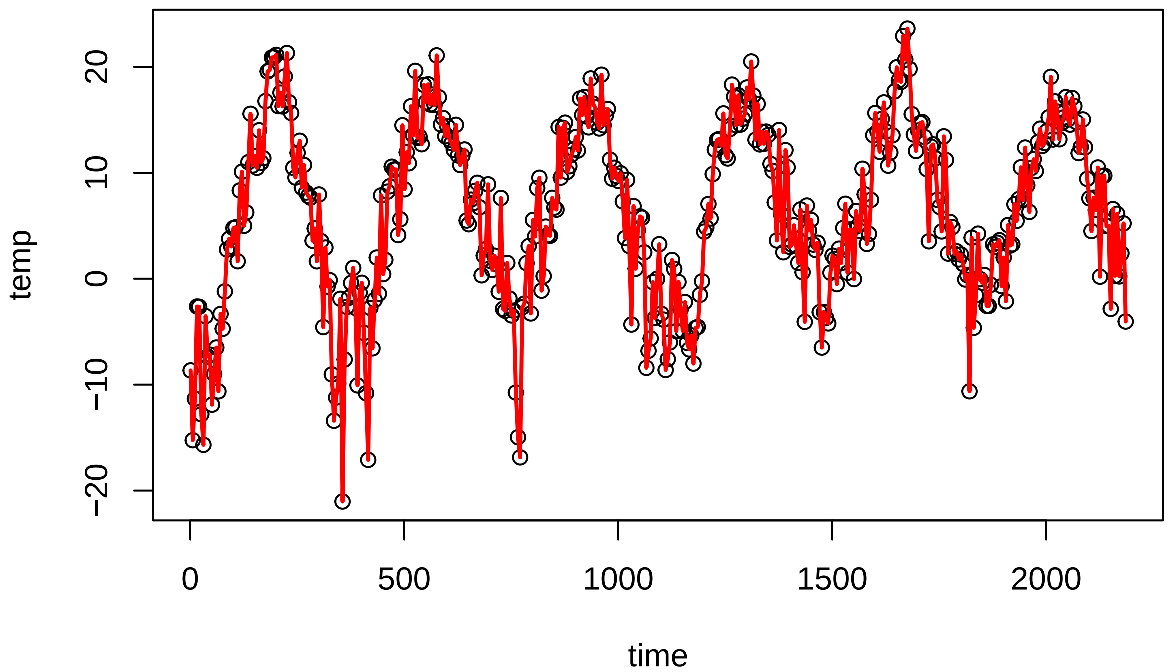
d)

```r
kernel <- single_squared_exp_kernel(20, 1.2)

gp_fit <- gausspr(temp ~ day, thinned_data, kernel=kernel)
predicted <- predict(gp_fit, thinned_data)

plot(thinned_data$time, predicted, type="p", xlab="time", ylab="temp")
lines(thinned_data$time, predicted, col="red", lwd=2)
```



e)

```
kernel <- single_periodic_kernel(20, 1, 10, 365 / sd(thinned_data$time))

gp_fit <- gausspr(temp ~ time, thinned_data, kernel=kernel)
predicted <- predict(gp_fit, thinned_data)

plot(thinned_data$time, predicted, type="p", xlab="time", ylab="temp")
lines(thinned_data$time, predicted, col="red", lwd=2)
```



temp
−20
−10
0
10
20
0
500
1000
1500
2000
time

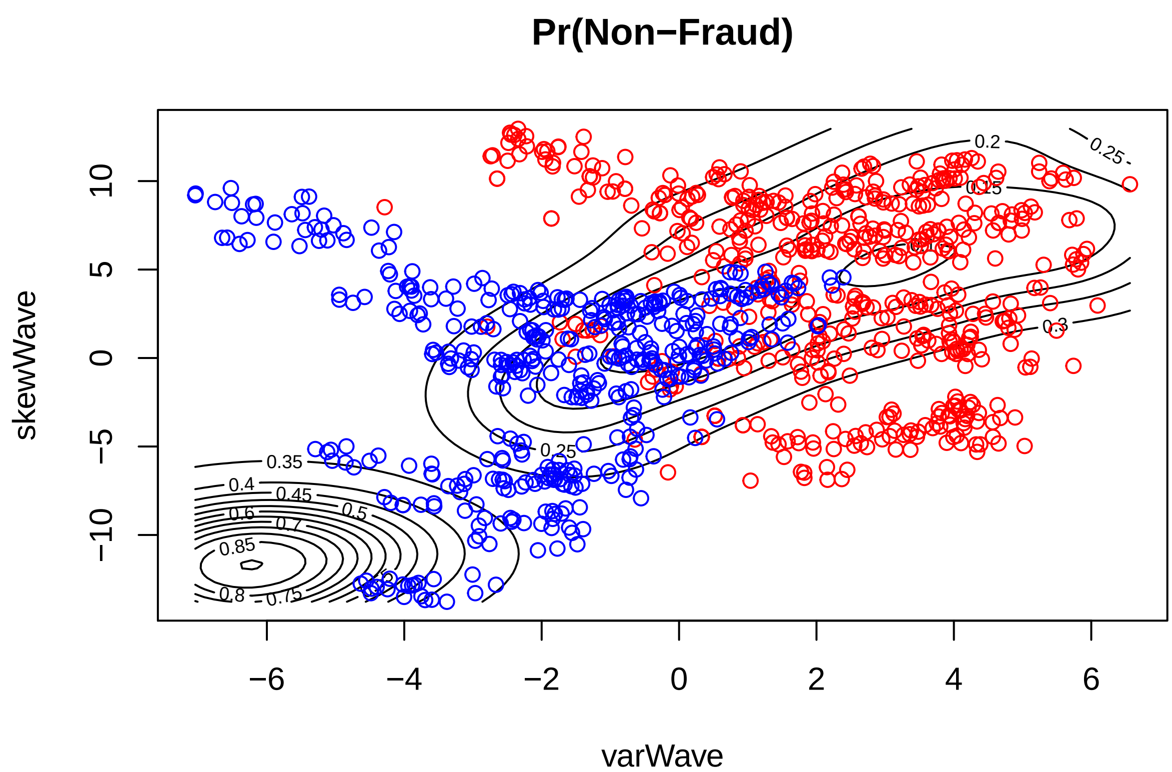
3)

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

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

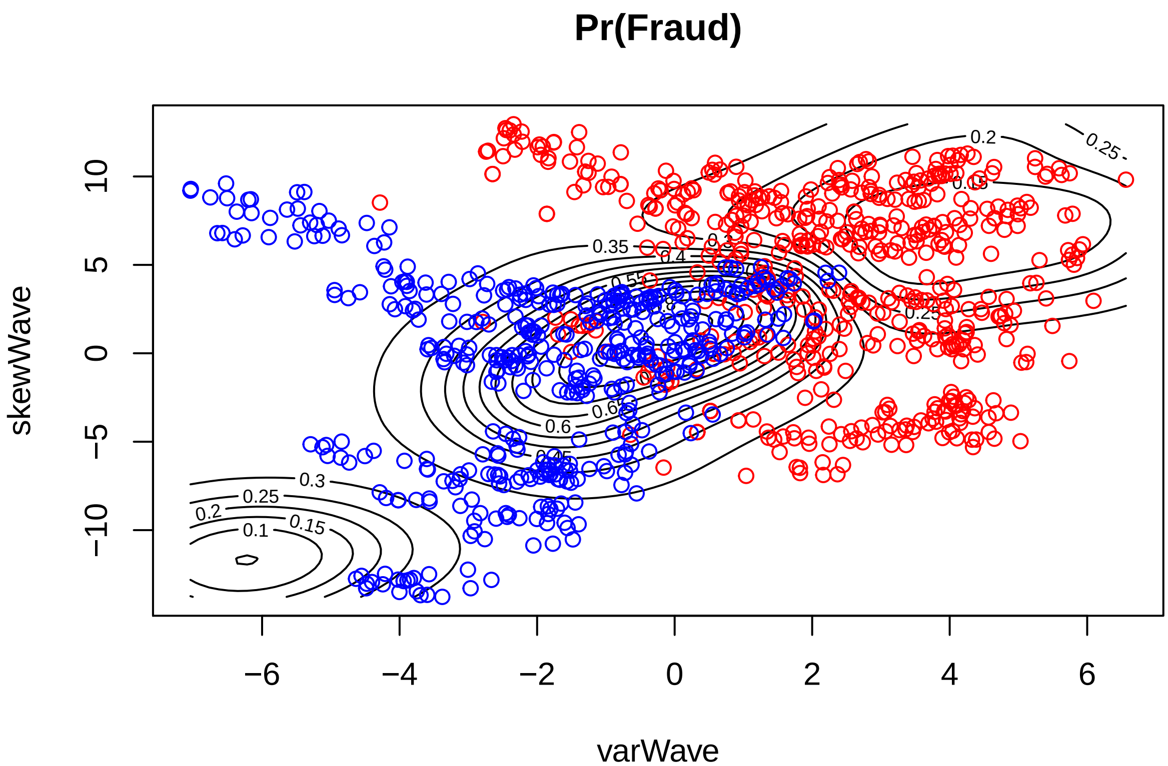
a)

```
gp_fit <- gausspr(fraud ~ varWave + skewWave, data=train)
#> Using automatic sigma estimation (sigest) for RBF or laplace kernel
train_predictions <- predict(gp_fit, train)
train_tbl <- table(train_predictions, train$fraud)
train_acc <- sum(diag(train_tbl)) / sum(train_tbl)
train_tbl
#>
#> train_predictions   0   1
#>                 0 512  24
#>                 1  44 420
train_acc
#> [1] 0.932
```

```
x1 <- seq(min(train$varWave), max(train$varWave), length=100)
x2 <- seq(min(train$skewWave), max(train$skewWave), length=100)
grid_points <- meshgrid(x1, x2)
grid_points <- cbind(c(grid_points$x), c(grid_points$y))
grid_points <- data.frame(gridPoints)
names(grid_points) <- c("varWave", "skewWave")
predicted_probs <- predict(gp_fit, grid_points, type="probabilities")

## Plotting for Prob(Non-Fraud)
contour(x1, x2, matrix(probPreds[, 1],100), 20,
        xlab = "varWave", ylab="skewWave",
        main = 'Pr(Non-Fraud)')
points(train$varWave[train$fraud == 0], train$skewWave[train$fraud == 0], col="red")
points(train$varWave[train$fraud == 1], train$skewWave[train$fraud == 1], col="blue")
```




```r
## Plotting for Prob(Fraud)
contour(x1, x2, matrix(probPreds[, 2],100), 20,
        xlab = "varWave", ylab="skewWave",
        main = 'Pr(Fraud)')
points(train$varWave[train$fraud == 0], train$skewWave[train$fraud == 0], col="red")
points(train$varWave[train$fraud == 1], train$skewWave[train$fraud == 1], col="blue")
```



b)

```
test_predictions <- predict(gp_fit, test)
test_tbl <- table(test_predictions, test$fraud)
test_acc <- sum(diag(test_tbl)) / sum(test_tbl)
test_tbl
#>
#> test_predictions   0   1
#>                0 191   9
#>                1  15 157
test_acc
#> [1] 0.9354839
```

c)

```
gp_fit <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data=train)
#> Using automatic sigma estimation (sigest) for RBF or laplace kernel
train_predictions <- predict(gp_fit, train)
train_tbl <- table(train_predictions, train$fraud)
train_acc <- sum(diag(train_tbl)) / sum(train_tbl)
train_tbl
#>
#> train_predictions   0   1
```

```
#>                0 552   0
#>                1   4 444
train_acc
#> [1] 0.996
```

```
test_predictions <- predict(gp_fit, test)
test_tbl <- table(test_predictions, test$fraud)
test_acc <- sum(diag(test_tbl)) / sum(test_tbl)
test_tbl
#>
#> test_predictions   0   1
#>                0 205   0
#>                1   1 166
test_acc
#> [1] 0.9973118
```