Advanced Machine Learning

Lab 4: Gaussian Processes

Jakob Berggren

2023-10-30

Question 1: Implementing GP Regression.

The first exercise is to write your own implementation for the Gaussian Process Regression model:

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

To do this, the function *posteriorGP* was implemented with the help of Algorithm 2.1 in Rasmussen and Willams' book on Gaussian Processes for Machine Learning, and the lab instructions:

Question 1.1

Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function 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 σ_n.*
- *k: Covariance function or kernel.*

This resulted in code as follows:

```
posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...) {

  # Covariance matrices:
  K <- k(X, X, ...)
  KStar <- k(X, XStar, ...)

  L <- t(chol(K + (sigmaNoise ** 2) * diag(length(X)))) # Cholesky decomposition

  alpha <- solve(t(L), solve(L, y))
  fstar <- t(KStar) %*% alpha # Predictive mean

  v <- solve(L, KStar)
  V <- k(XStar, XStar, ...) - t(v) %*% v # Predictive variance

  return(list(predMean = fstar, predVar = V))
}
```

Here, k is the Squared Exponential Kernel, or the Covariance Function, which was implemented as so:

```
# Covariance function
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)
}
```

Question 1.2

Next, experiments using the implementation from Q1.1 was run according to instructions:

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 $\sigma_n = 0.1$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 % probability pointwise bands for f.

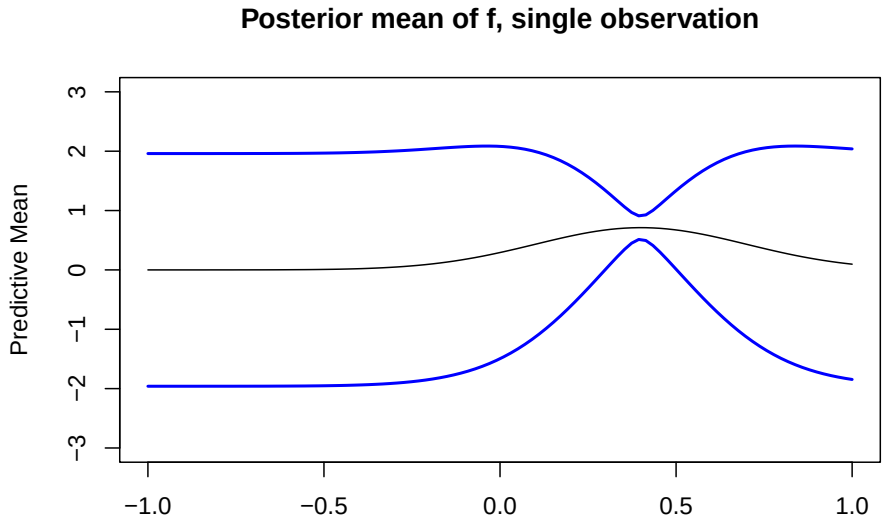
```
# Prior hyperparameters
sigmaF <- 1
l <- 0.3

observations <- data.frame(x = 0.4, y = 0.719) # single observation: (x, y) = (0.4, 0.719)

sigmaNoise <- 0.1
interval <- seq(-1, 1, length = 100) # Posterior mean of f over the interval [-1, 1]

postSim <- posteriorGP(X = observations$x,
                       y = observations$y,
                       interval,
                       sigmaNoise,
                       k = SquaredExpKernel,
                       sigmaF,
                       l)

plot(interval,
     postSim$predMean,
     type = "l",
     ylim = c(-3, 3),
     xlab = "",
     ylab = "Predictive Mean",
     main = "Posterior mean of f, single observation")
lines(interval,
      postSim$predMean - 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
lines(interval,
      postSim$predMean + 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
```



As the plot shows, the probability bands is considerably tighter around the observed point used as prior.

Question 1.3

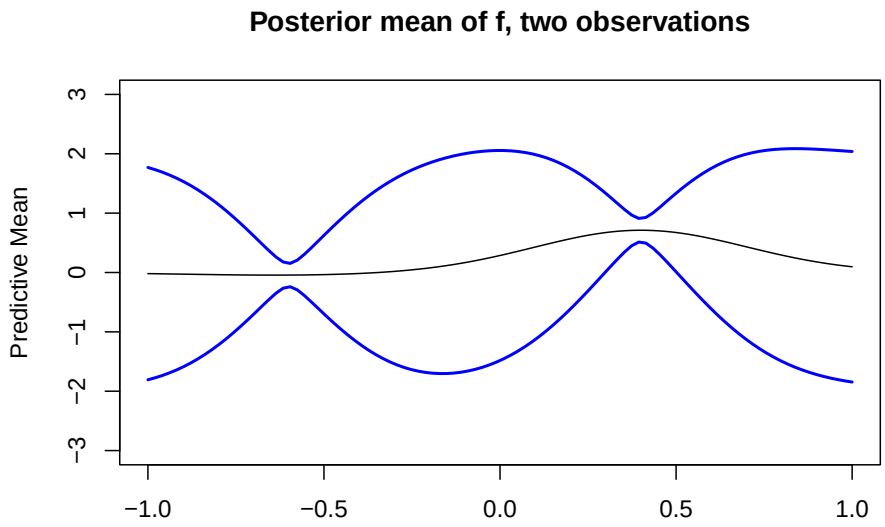
Next, we add another observation to the prior:

Update your posterior from (2) with another observation: $(x, y) = (-0.6, -0.044)$.

```
observation <- data.frame( x = c(0.4, -0.6), y = c(0.719, -0.044))

postSim <- posteriorGP(X = observation$x,
                       y = observation$y,
                       interval, sigmaNoise,
                       k = SquaredExpKernel,
                       sigmaF,
                       l)

plot(interval,
     postSim$predMean,
     type = "l",
     ylim = c(-3, 3),
     xlab = "",
     ylab = "Predictive Mean",
     main = "Posterior mean of f, two observations")
lines(interval,
      postSim$predMean - 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
lines(interval,
      postSim$predMean + 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
```



The outcome is very similar to the previous experiment. We can see that the probability bands is considerably tighter around the observed points in the prior.

Question 1.4

Continuing the experiments, we are now computing the posterior distribution of f using all five data points in the following table:

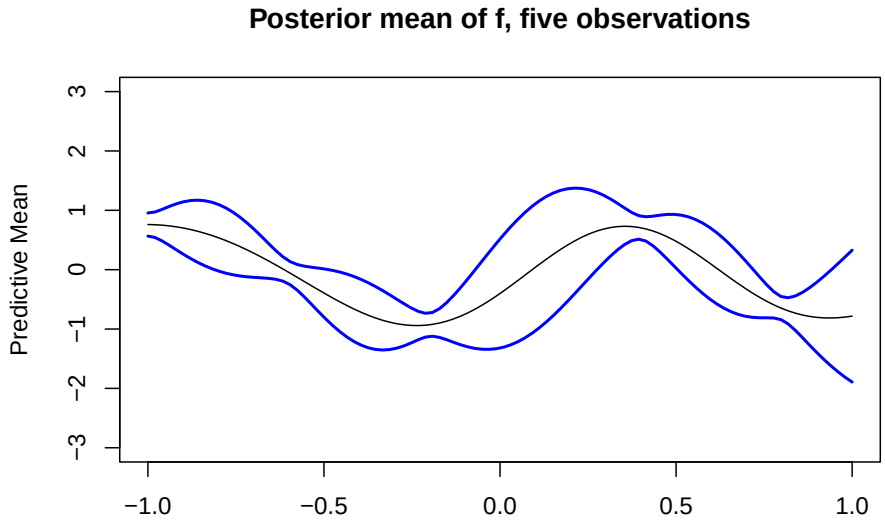
x	-1.0	-0.6	-0.2	0.4	0.8
y	0.768	-0.044	-0.940	0.719	-0.664

Again, using very similar code to the previous experiments, just updating the observations data frame.

```
observations <- data.frame(x = c(-1.0, -0.6, -0.2, 0.4, 0.8),
                           y = c(0.768, -0.044, -0.940, 0.719, -0.664))

postSim <- posteriorGP(X = observations$x,
                       y = observations$y,
                       interval,
                       sigmaNoise,
                       k = SquaredExpKernel,
                       sigmaF,
                       l)

plot(interval,
     postSim$predMean,
     type = "l",
     ylim = c(-3, 3),
     xlab = "",
     ylab = "Predictive Mean",
     main = "Posterior mean of f, five observations")
lines(interval,
      postSim$predMean - 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
lines(interval,
      postSim$predMean + 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
```



The same pattern can be seen here, where the confidence bands are much tighter around the observations. One can also see that the confidence bands are tighter over all which is a result of more observations.

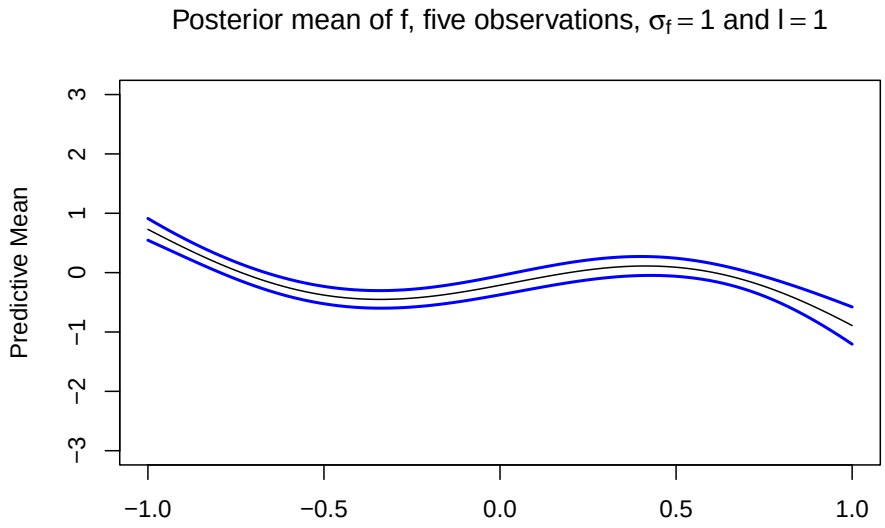
Question 1.5

Here, the experiment from 1.4 is repeated, but using the hyperparameters $\sigma_f = 1$ and $l = 1$.

```r
# New hyperparameters
sigmaF <- 1
l <- 1

postSim <- posteriorGP(X = observations$x,
                       y = observations$y,
                       interval,
                       sigmaNoise,
                       k = SquaredExpKernel,
                       sigmaF,
                       l)

plot(interval,
     postSim$predMean,
     type = "l",
     ylim = c(-3, 3),
     xlab = "",
     ylab = "Predictive Mean",
     main = expression(paste("Posterior mean of f, five observations, ",
                             sigma[f] == 1,
                             " and ", l == 1)))
lines(interval,
      postSim$predMean - 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
lines(interval,
      postSim$predMean + 1.96 * sqrt(diag(postSim$predVar)),
      col = "blue",
      lwd = 2)
```



The parameter l is also referred to as the *smoothing factor*. This is very evident in the plot, where the confidence bands have been smoothed out, and are no longer tighter or more loose dependent on whether they are near an observation or not.

Question 2: GP Regression with kernlab

Question 2 is about exploring GP regression using the R package *kernlab*. The dataset for this exercise is the daily mean temperature in Stockholm during the period January 1, 2010 - December 31, 2015. The leap year day is removed.

First, the dataset is read.

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

Next, the data is structured according to the lab instructions.

```
# time = 1, 2, ..., 365 x 6, only use every fifth observation
time <- seq(from = 1, to = 365 * 6, by = 5)
# day = 1, 2, . . ., 365, 1, 2, . . ., 365, only use every fifth observation
day <- rep(seq(from = 1, to = 365, by = 5), times = 6)

temperatures <- data$temp[time] # Temperatures for each (fifth) day.
```

Question 2.1

Question 2.1 simply asks to familiarize with the functions *gausspr* and *kernelMatrix* in the package *kernlab*:

define your own square exponential kernel function (with parameters l and $sigma_f$), 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 $X_* = (2, 3, 4)^T$.

First, a new square exponential kernel function which only takes l and σ as parameters is defined. The inner function is the same as in Question 1.

```
# Define your own square exponential kernel function with parameters l and sigmaf.
SE <- function(l = 1, sigmaf = 1) {
  r <- function (X, XStar) {
    n1 <- length(X)
    n2 <- length(XStar)
    K <- matrix(NA, n1, n2)
    for (i in 1:n2) {
      K[, i] <- sigmaf ^ 2 * exp(-0.5 * ((X - XStar[i]) / l) ^ 2)
    }
    return(K)
  }
  class(r) <- 'kernel' # Return as class kernel.
  return (r)
}

X <- matrix(c(1,3,4)) # Simulating some data.
Xstar <- matrix(c(2,3,4))

# Own SE kernel
SEkernel <- SE(l = 1, sigmaf = 1)
SEkernel(1,2) # Just a test - evaluating the kernel in the points x=1 and x'=2.
```

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

```
# Computing the whole covariance matrix K from the kernel. Just a test.
kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(X,Xstar).
```

```
## 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
```

Question 2.2

Next, we 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 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

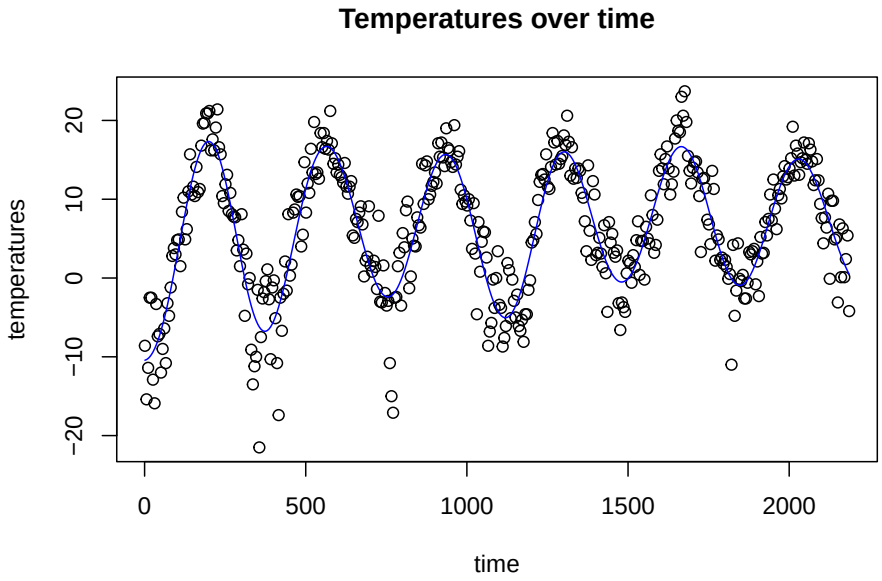
```
# Let sigmaN be the residual variance from a simple quadratic regression fit
lmfit <- lm(temperatures ~ time + time^2)
sigmaN <- sd(lmfit$residuals)

# Estimate the above Gaussian process regression model using
# the squared exponential function from (1) using sigmaF = 20 and l = 0.2.

GP <- gausspr(time, temperatures, kernel=SE(l=0.2, sigmaf=20), var=sigmaN^2)

# Use the predict function in R to compute the posterior mean
# at every data point in the training dataset.
postMean <- predict(GP, newdata=time)

# Make a scatterplot of the data and superimpose the posterior mean of f as a curve
plot(time, temperatures, main="Temperatures over time")
lines(time, postMean, type="l", col="blue")
```



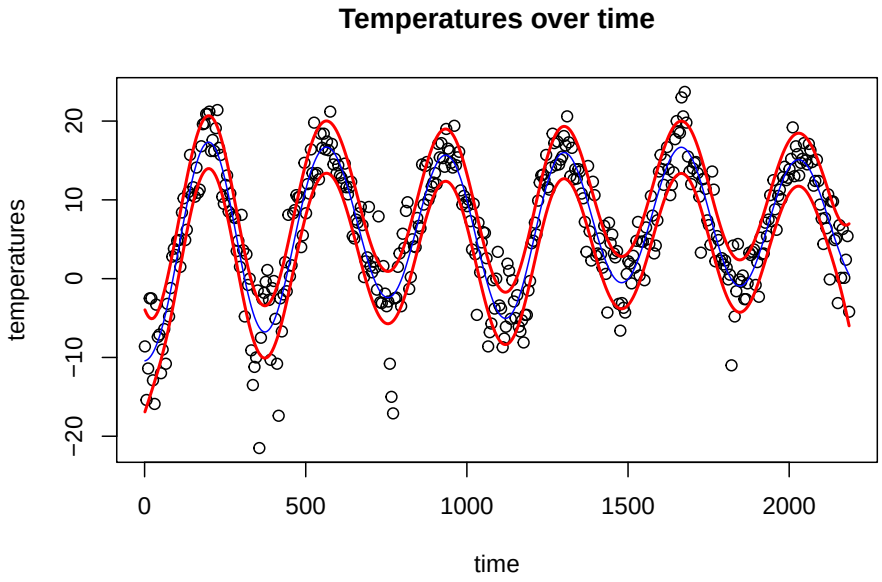
The blue line shows the Posterior Mean. As one can see, the data is fitted quite well.

Question 2.3

Next, we will compute the posterior variance of f and plot the probability bands. Also, superimpose these bands on the figure with the bands obtained in Question 2.2.

```
# Do your own computations for the posterior variance of f
post <- posteriorGP(scale(time),
                    scale(temperatures),
                    scale(time), sigmaN,
                    k = SquaredExpKernel,
                    20, 0.2)

plot(time, temperatures, main="Temperatures over time")
# plot the 95 % probability (pointwise) bands for f
lines(time,
      post$predMean * sqrt(var(temperatures)) + mean(temperatures) - 1.96 *
        sqrt(diag(post$predVar)),
      col = "red",
      lwd = 2)
lines(time,
      post$predMean * sqrt(var(temperatures)) + mean(temperatures) + 1.96 *
        sqrt(diag(post$predVar)),
      col = "red",
      lwd = 2)
# Superimpose with Posterior mean from Q2.2
lines(time, postMean, type="l", col="blue")
```



The 95% probability bands are here plotted in red together with the posterior mean from question 2.2. The probability bands are able to capture the variance in the data quite well.

Question 2.4

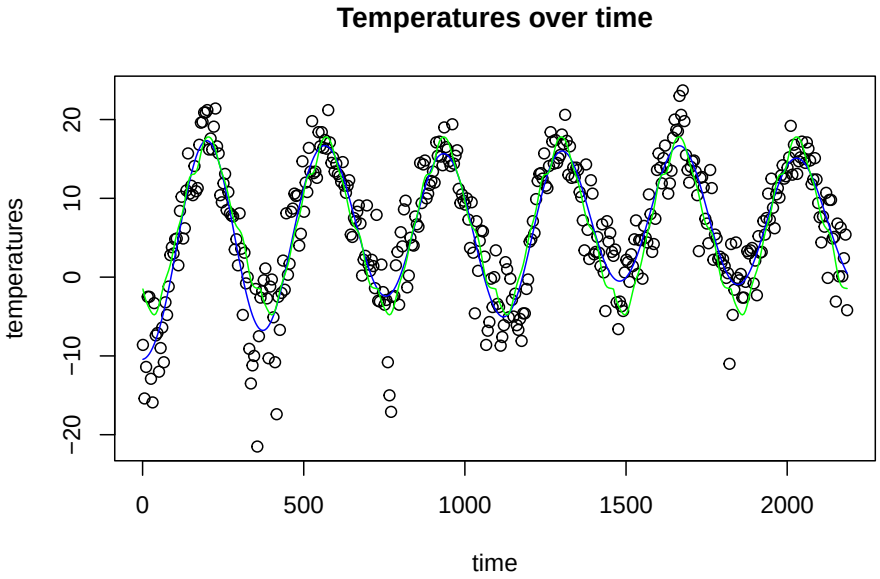
Now consider the model:

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

Estimate the model using 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 Question 2.2. Compare the results from each model.

```
# Estimate the model using the squared exponential function with sigmaf = 20 and l = 0.2.
GP <- gausspr(day, temperatures, kernel=SE(l = 0.2, sigmaf = 20), var = sigmaN^2)
postMean2 <- predict(GP, newdata=day)

plot(time, temperatures, main="Temperatures over time")
# Superimpose the posterior mean from this model on the posterior mean from model in (2).
lines(time, postMean, type="l", col="blue")
lines(time, postMean2, type="l", col="green")
```



Here the posterior gained from the new model is plotted in green together with the previous posterior in blue. The two models fit the data similarly and quite well, with only slight variations. One can see that the the later (green) posterior is able to fit the data a bit better later in time, while the first (blue) posterior is better fitted in early time periods.

Question 2.5

Implement the following extension to the squared exponential kernel (locally periodic kernel).

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

Estimate the GP model using the time variable with this kernel and hyperparamters $\sigma_f = 20$, $l_1 = 1$, $l_2 = 10$ and $d = 365/sd(time)$. Compare the fit to the previous two models with $\sigma_f = 20$ and $l = 0.2$

First, implement the Periodic Kernel according to the formula.

```
# Implement the periodic kernel.
PeriodicKernel <- function(sigmaf, l1, l2, d) {
  r <- function(X, XStar) {
    t1 <- -((2 * sin(pi * abs(X - XStar)^2 / d)) / l1^2)
    t2 <- -(0.5 * (abs(X - XStar)^2) / l2^2)
    return(sigmaf^2 * exp(t1) * exp(t2))
  }
```

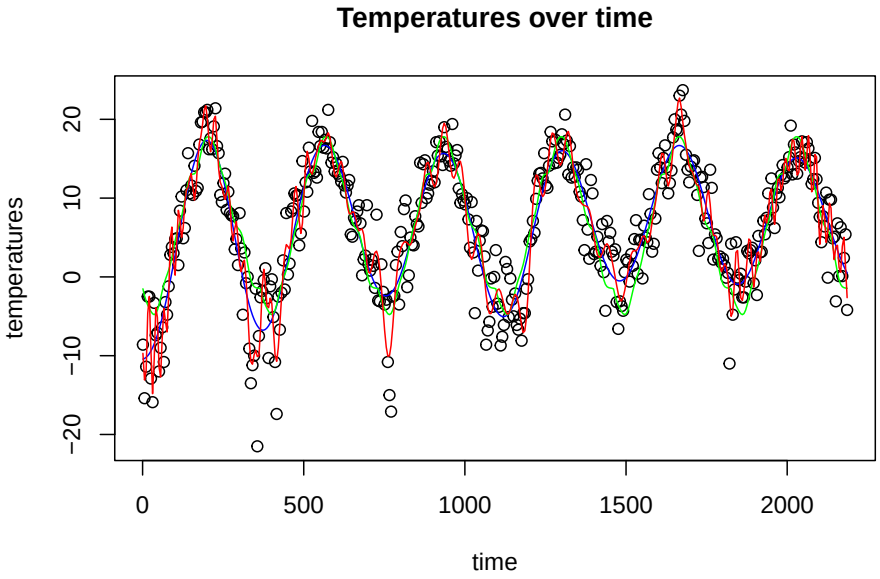
```
  class(r) <- 'kernel' # Return as class kernel.
  return (r)
}
```

This is then used to estimate a GP model, and predict the Posterior Mean. Finally, results are plotted together with previous experiments.

```
# Estimate the GP model using the time variable with this kernel
GPPeriodic <- gausspr(time,
              temperatures,
              kernel=PeriodicKernel(sigmaf = 20, l1 = 1, l2 = 10, d = 365/sd(time)),
              var = sigmaN^2)

postMeanPeriodic <- predict(GPPeriodic, newdata = time)

# Compare the fit to the previous two models.
plot(time, temperatures, main="Temperatures over time")
lines(time, postMean, type="l", col="blue")
lines(time, postMean2, type="l", col="green")
lines(time, postMeanPeriodic, type="l", col="red")
```



It is clear that the Periodic Kernel is able to fit the data better than the Squared Exponential kernel. More Data points are captured, even some that could be considered as outliers. It is also clear that the Periodic Kernel is a lot more "jerky" as supposed to "smooth".

The Periodic Kernel can (as the name suggests) capture periodic patterns which is why it is better at modelling the underlying data.

Question 3: GP Classification with kernlab.

In Question 3, GP Classification is used to classify banknote fraud data.

First, the dataset is read.

```
# Read 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])
```

1000 observations is used as training data, and the rest as test.

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

Question 3.1

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). Compute the confusion matrix for the classifier and its accuracy.

First, fit a GP classification model on the trianing data.

```
# Fit a Gaussian process classification model for fraud on the training data.
# Start only using covariates varWave and skewWave in the model.
GPClass <- gausspr(fraud ~ varWave + skewWave, data = train) # Default kernel & parameters
```

Next, predictions are made and the confusion matrix is calculated.

```
fraudPrediction <- predict(GPClass, newdata=train)
table(fraudPrediction, train$fraud) # Confusion Matrix
```

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

```
sum(diag(table(fraudPrediction, train$fraud))) /
  sum(table(fraudPrediction, train$fraud)) # Accuracy
```

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

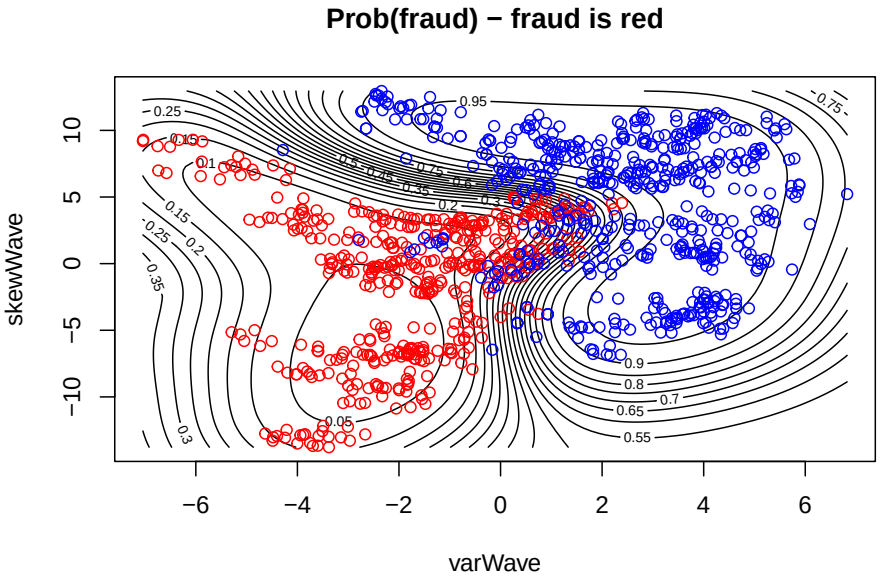
As the confusion matrix shows, most predictions are correct. The accuracy is 94.1%.

Now we will make a contour plot over the results. Here function "meshgrid" from R package AtmRay is used.

```
# class probabilities
probPreds <- predict(GPClass, train, type="probabilities")
x1 <- seq(min(train$varWave), max(train$varWave), length = 100)
x2 <- seq(min(train$skewWave), max(train$skewWave), length = 100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

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



Question 3.2

Make predictions for the test set using the estimated model from Question 3.1. Compute the accuracy.

```
# Make predictions for the test set.
fraudPredictionTest <- predict(GPClass, newdata=test)

# Compute the confusion matrix and the classifier accuracy
table(fraudPredictionTest, test$fraud) # Confusion Matrix Test
```

```
##
## fraudPredictionTest   0   1
##                   0 199   9
##                   1  19 145
```

```
sum(diag(table(fraudPredictionTest, test$fraud))) /
  sum(table(fraudPredictionTest, test$fraud)) # Accuracy Test
```

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

As expected, the accuracy is slightly lower, but still quite high at 92.5%.

Question 3.3

Now 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
# Train model using all four covariates.
GPClass2 <- gausspr(fraud ~ ., data = train)
```

```r
# Make predictions for the test set.
fraudPredictionTest2 <- predict(GPClass2, newdata=test)

# Confusion Matrix and Accuracy
table(fraudPredictionTest2, test$fraud) # Confusion Matrix Test
```

```
##
## fraudPredictionTest2   0   1
##                    0 216   0
##                    1   2 154
```

```r
sum(diag(table(fraudPredictionTest2, test$fraud))) /
  sum(table(fraudPredictionTest2, test$fraud)) # Accuracy Test
```

```
## [1] 0.9946237
```

Using all four covariates significantly improves the model accuracy. Almost all data points have been classified correctly, with a high accuracy of 99.5%. This is also as expected, since we add more information to the model, and thus the accuracy should increase.