lab4

Daniel Kouznetsov

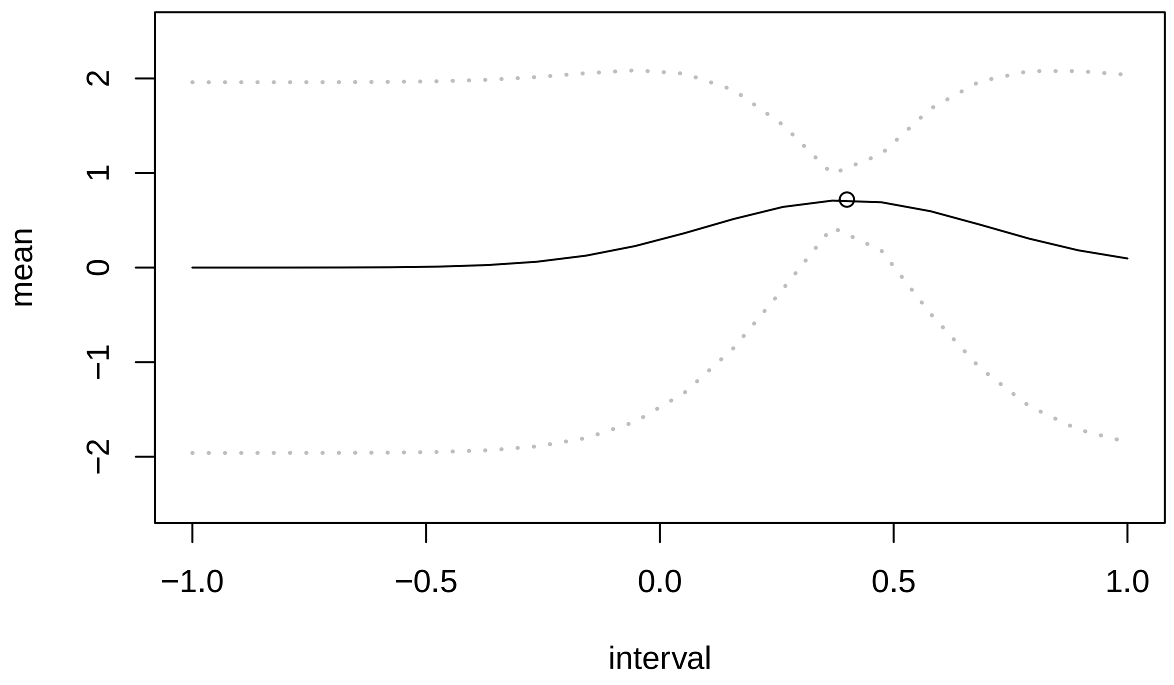
2022-10-07

Implementing GP Regression

```r
# SE Covariance function
# l determines smoothness
# sigmaF determines the variance
SquaredExpKernel <- function(x1, x2, sigmaF=1, l=0.3){
  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){
  # Covariance matrices
  KXX <- k(X, X)
  KXX_star <- k(X, XStar)
  Kstarstar <- k(XStar, XStar)

  size <- length(X)

  # transpose to get lower triangular
  L <- t(chol(KXX + (sigmaNoise^2) * diag(size)))

  Ly <- solve(L,y) # L\y
  alpha <- solve(t(L), Ly) # Lt\(L\y)

  predictiveMean <- t(KXX_star) %*% alpha

  V <- solve(L, KXX_star)
  predictiveVariance <- Kstarstar - t(V) %*% V

  return(list(mean=predictiveMean, var=predictiveVariance));
}
```

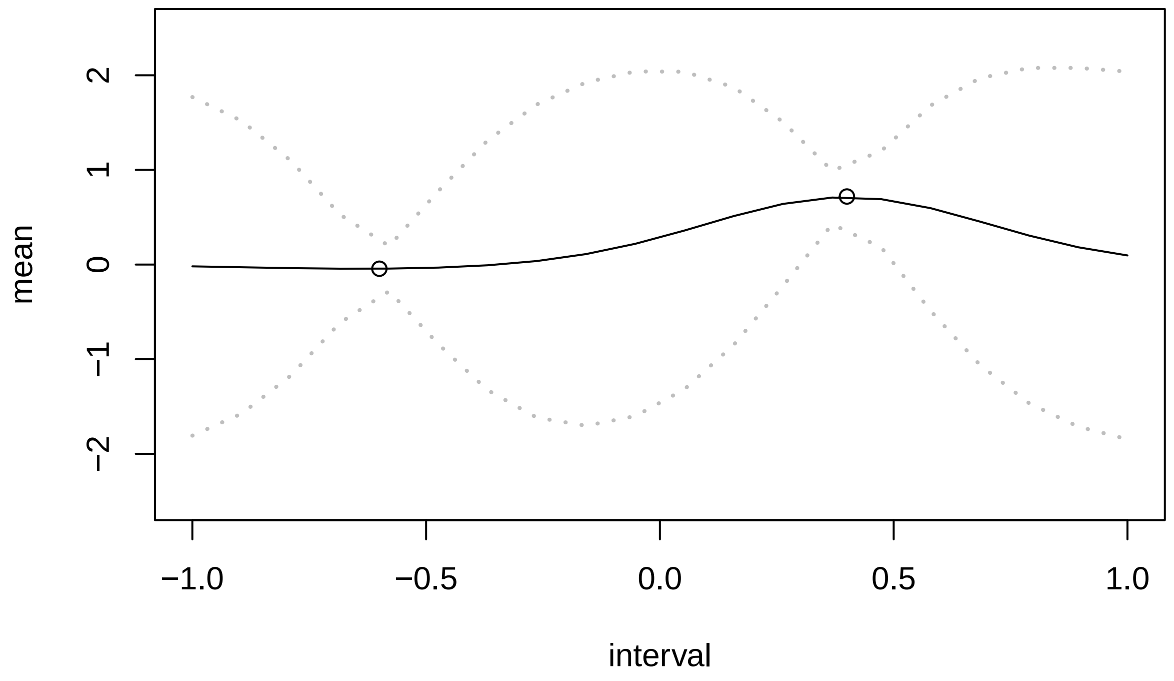
First plot

```r
sigmaN = 0.1
x = 0.4; y = 0.719
interval = seq(-1, 1, length=20)

posteriorSim = posteriorGP(x, y, interval, sigmaN, k=SquaredExpKernel)

plot(interval, posteriorSim$mean, type="l",
     ylim=c(min(-2.5), max(2.5)),
     xlab="interval", ylab="mean")

lines(interval, posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var)),
      col="gray", lty=21, lwd=2)
lines(interval, posteriorSim$mean + 1.96*sqrt(diag(posteriorSim$var)),
      col="gray", lty=21, lwd=2)
points(x, y)
```



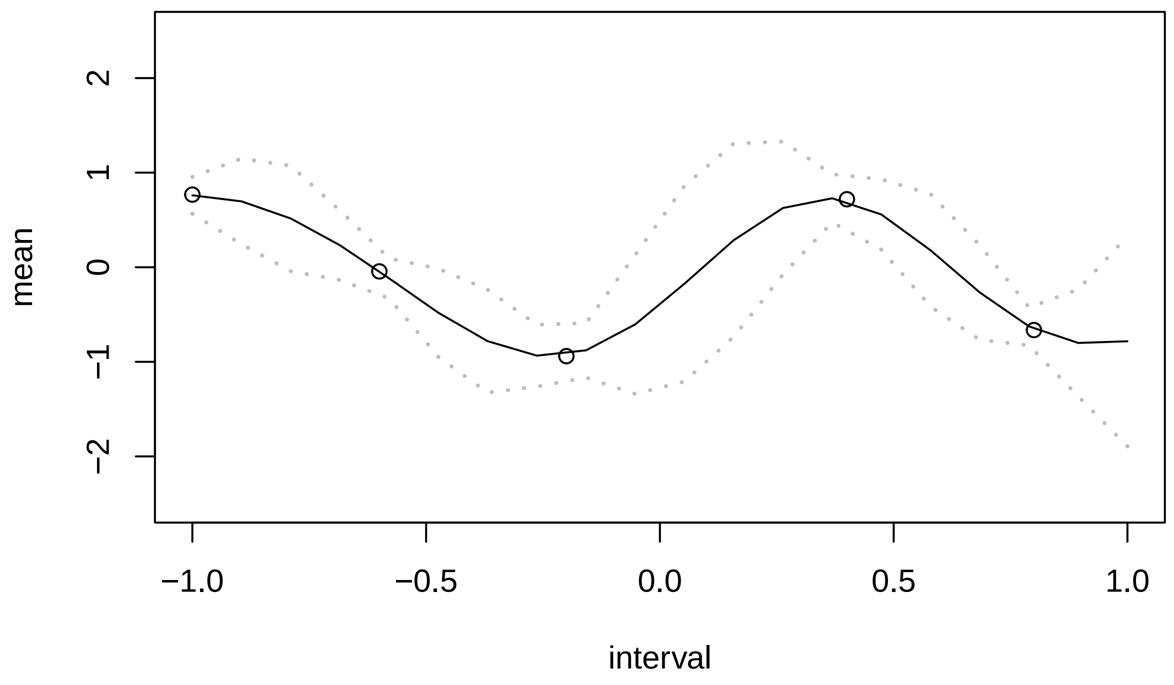
Second plot

```
x = c(0.4, -0.6); y = c(0.719, -0.044)

posteriorSim = posteriorGP(x, y, interval, sigmaN, k=SquaredExpKernel)

plot(interval, posteriorSim$mean, type="l",
     ylim=c(min(-2.5), max(2.5)),
     xlab="interval", ylab="mean")

lines(interval, posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var)),
      col="gray", lty=21, lwd=2)
lines(interval, posteriorSim$mean + 1.96*sqrt(diag(posteriorSim$var)),
      col="gray", lty=21, lwd=2)
points(x, y)
```



Third plot

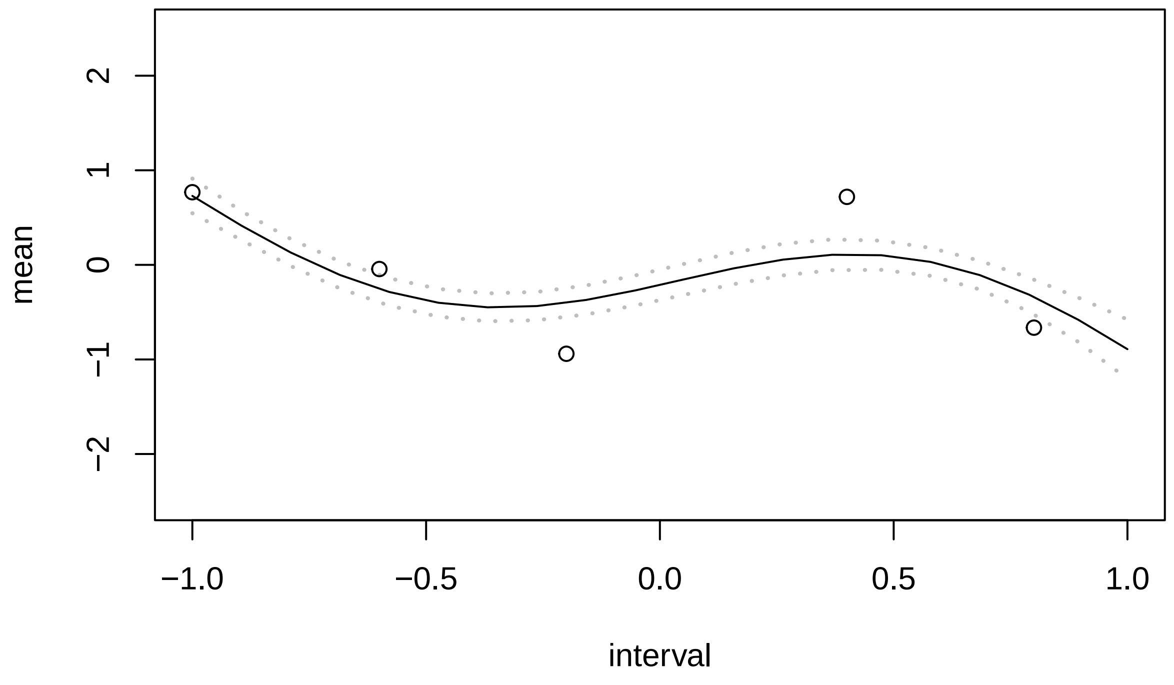
```r
x = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)

posteriorSim = posteriorGP(x, y, interval, sigmaN, k=SquaredExpKernel)

plot(interval, posteriorSim$mean, type="l",
     ylim=c(min(-2.5), max(2.5)),
     xlab="interval", ylab="mean")

lines(interval, posteriorSim$mean - 1.96*sqrt(diag(posteriorSim$var)),
      col="gray", lty=21, lwd=2)
lines(interval, posteriorSim$mean + 1.96*sqrt(diag(posteriorSim$var)),
      col="gray", lty=21, lwd=2)
points(x, y)
```



Fourth plot with different values for SE kernel

```r
# SE Covariance function
# l determines smoothness
# sigmaF determines the variance
SquaredExpKernel <- function(x1, x2, sigmaF=1, l=1){
  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)
}
```

```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)
```



Discussion 1, 2: For the first few plots our probability bands go down together with our known data which is to be expected, this is easily seen as the case with plot two when we add one more point as compared to the first one where we only have one.

3, 4, 5: Same principle as the first one for probability bands. Changing the hyperparameters change how our probability bands act. Using l = 1 instead of 0.3 gives us a much more smooth curve but this is at the cost of not being as accurate.

GP Regression with kernlab

Setting up sample data for temperatures

```
library(kernlab)

data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
                header=TRUE, sep=";")
time = seq(1, 2190, 5)
day = rep(seq(1, 365, 5), times=6)
data = data$temp[time]
```

Testing kernel and evaluating

```
X = c(1, 3, 4)
Xstar = c(2, 3, 4)
x = 1; xprim = 2

SquaredExpKernel <- function(sigmaF=1, l=0.3){
  rval <- 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])/l)^2)
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

SFunc <- SquaredExpKernel(sigmaF = 1, l = 2)
SFunc(x, xprim) # evaluating kernel
```

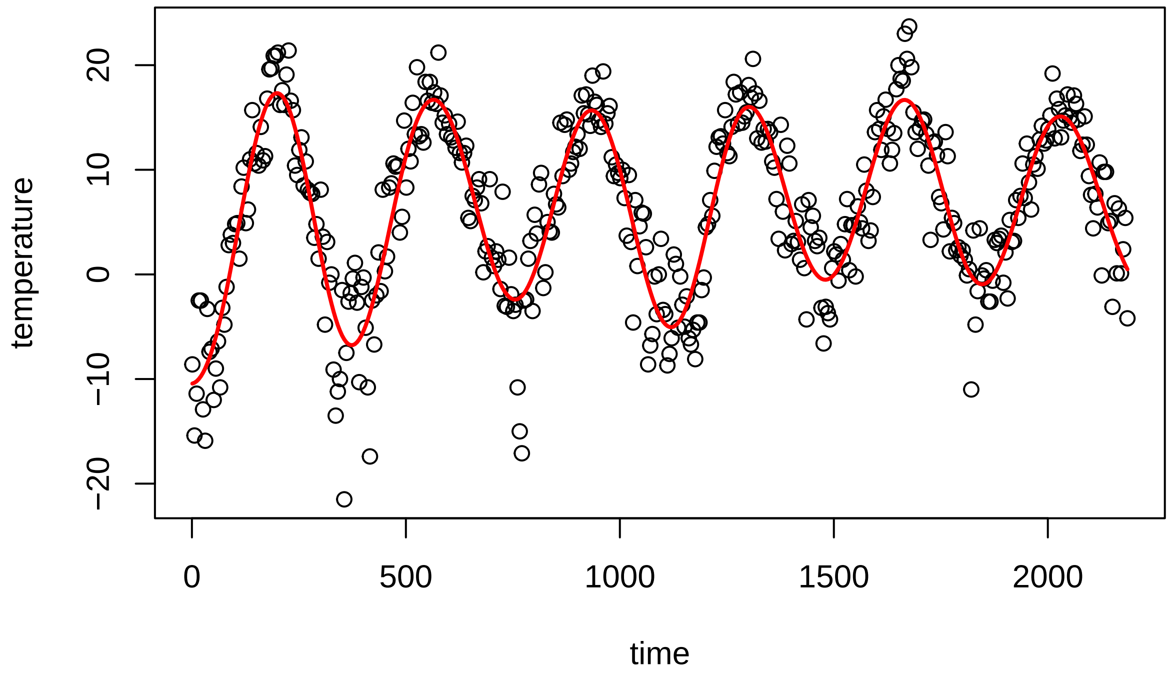
```
##           [,1]
## [1,] 0.8824969
```

```
# computing the covariance matrix K from kernel
kernelMatrix(kernel = SFunc, x= X, y = Xstar)
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.8824969 0.6065307 0.3246525
## [2,] 0.8824969 1.0000000 0.8824969
## [3,] 0.6065307 0.8824969 1.0000000
```

Mean regression for temperatures

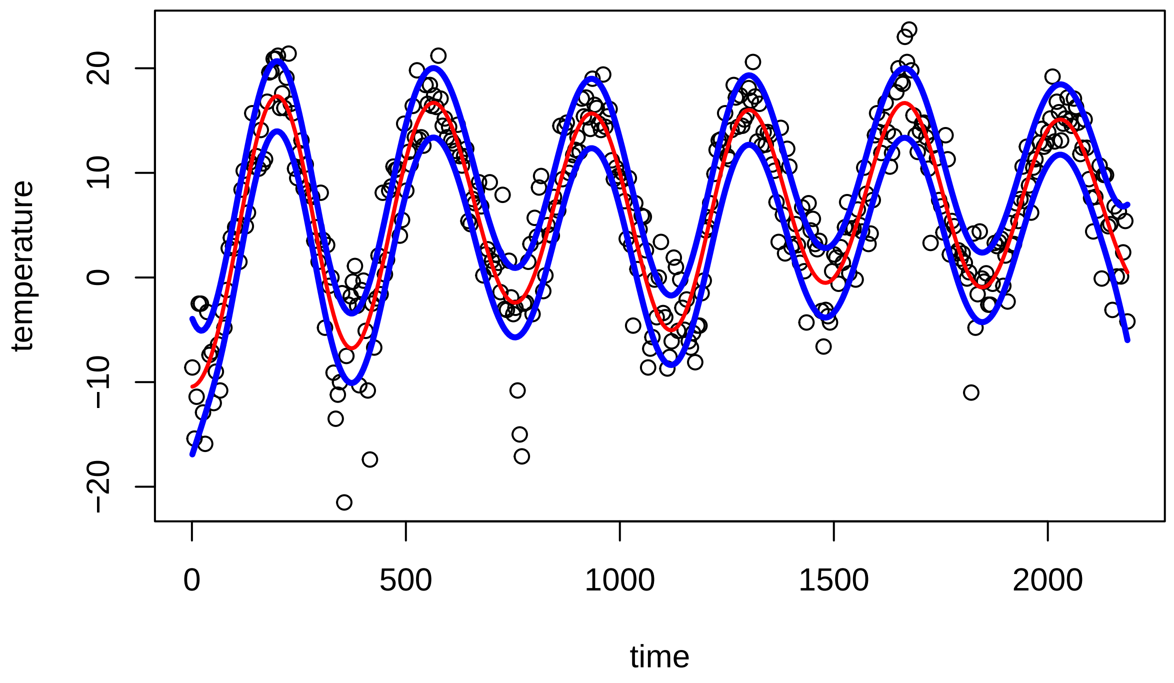
```
fit <- lm(data ~ time + time^2)
sigmaNoise <- sd(fit$residuals)

sigmaF <- 20; l <- 0.2
SFunc <- SquaredExpKernel(sigmaF = sigmaF, l = l)

GPfit <- gausspr(time, data, kernel=SFunc, var=sigmaNoise^2)
meanPred <- predict(GPfit, time)

plot(time, data, ylab="temperature")
lines(time, meanPred, col="red", lwd=2)
```



The mean follows our data pretty well!

Plot with 95% probability bands

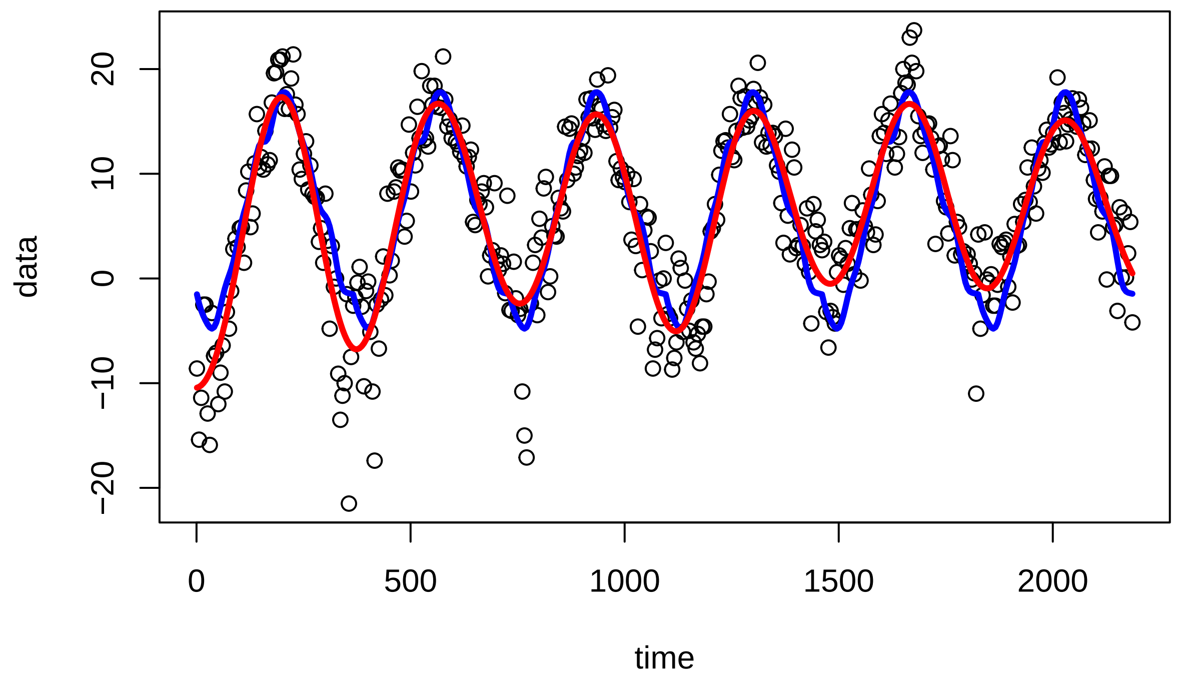
```r
posterior <- posteriorGP(X=scale(time), y=scale(data), XStar=scale(time),
                         sigmaNoise=sigmaNoise, k=SFunc)

# fix scaling
sqrtData <- sqrt(var(data))
meanPred <- posterior$mean * sqrtData + mean(data)

plot(time, data, ylab="temperature")
lines(time, meanPred, col="red", lwd=2)

lines(time, meanPred + 1.96*sqrt(diag(posterior$var)), col="blue", lwd=3)
lines(time, meanPred -1.96*sqrt(diag(posterior$var)), col="blue", lwd=3)
```



Here we see the same mean but with the addition of our 95% probability bands. We can clearly see that they miss some extremes in our data, for example the days where we have around -20 degrees, but other than that it fits our data pretty well!

Predictions by day

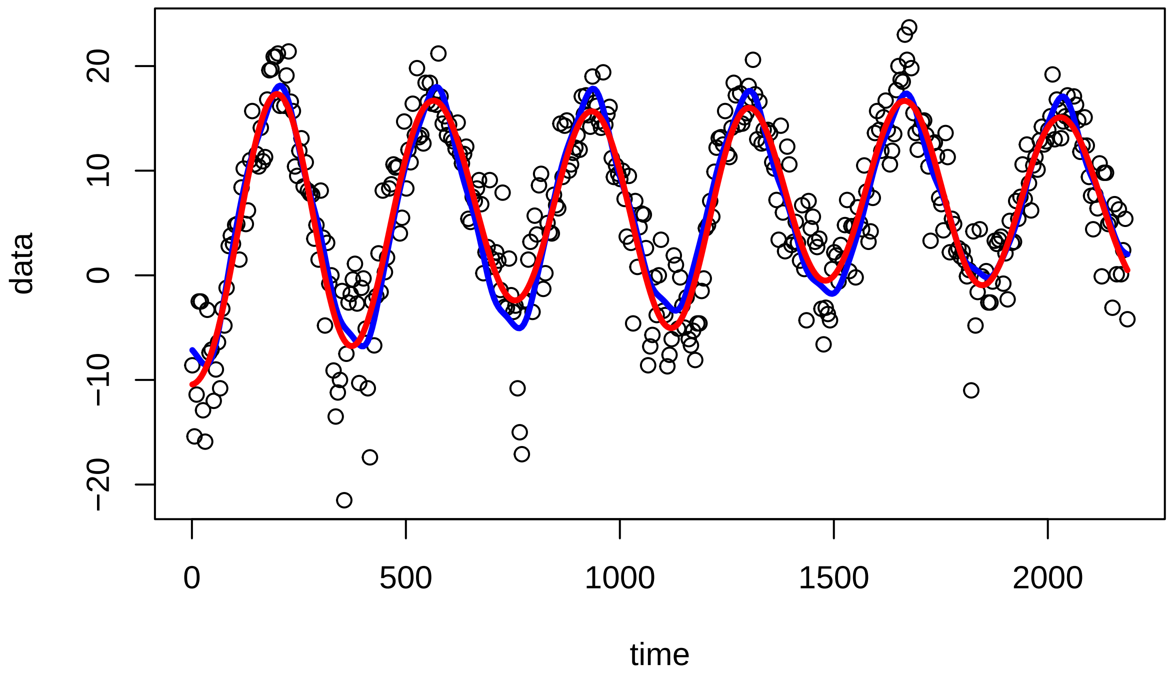
```
fit = lm(data ~ day + day^2)
sigmaNoise = sd(fit$residuals)
dayModel = gausspr(day, data, kernel=SFunc, var=sigmaNoise^2)
dayPred = predict(dayModel, day)

plot(time, data) # all time points on x-axis
lines(time, dayPred, col="blue", lwd=3)
lines(time, meanPred, col="red", lwd=3)
```



For this one we can see that it shows us the same line (blue) over and over. The red one should be more accurate over a longer span of time since it is being trained on more data, the blue line is only trained over data for one year whilst the red one is being trained on all of our data.

Using a locally periodic kernel

```
sigmaF = 20
l.1 = 1
l.2 = 10
d = 365/sd(time)

PeriodicKernel <- function(SigmaF, l.1, l.2, d){
  rval <-function(x1, x2=NULL){
    return(
      SigmaF^2 * exp(-(2* sin(pi*abs(x1 - x2)/d)^2) / (l.1)^2) *
        exp((-1/2) * (abs(x1-x2)/l.2)^2)
    )
  }
  class(rval) = "kernel"
  return(rval)
}


PKFunc <- PeriodicKernel(sigmaF, l.1, l.2, d)
PKFit = gausspr(time, data, kernel=PKFunc, var=sigmaNoise)

plot(time, data)
lines(time, predict(PKFit, time), col="blue", lwd=3)
lines(time, meanPred, col="red", lwd=3)
```



The blue line seems to be more prone to get affected by extreme data points and our red line seems to not be and lean more towards showing an accurate mean for having many data points in a cluster. Hard to tell which one is better though.

GP Classification with kernlab

```r
library(kernlab)
library(pracma) # AtmRay does not work for newer version of R
```

```
##
## Attaching package: 'pracma'

## The following objects are masked from 'package:kernlab':
##
##     cross, eig, size
```

```r
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)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
data.training = data[SelectTraining, ]

GPfit <- gausspr(fraud ~ varWave + skewWave, data.training)
```

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

```r
predict.training <- predict(GPfit, data.training)

cmatrix.training <- table(predict.training, data.training[,5])
cmatrix.training
```

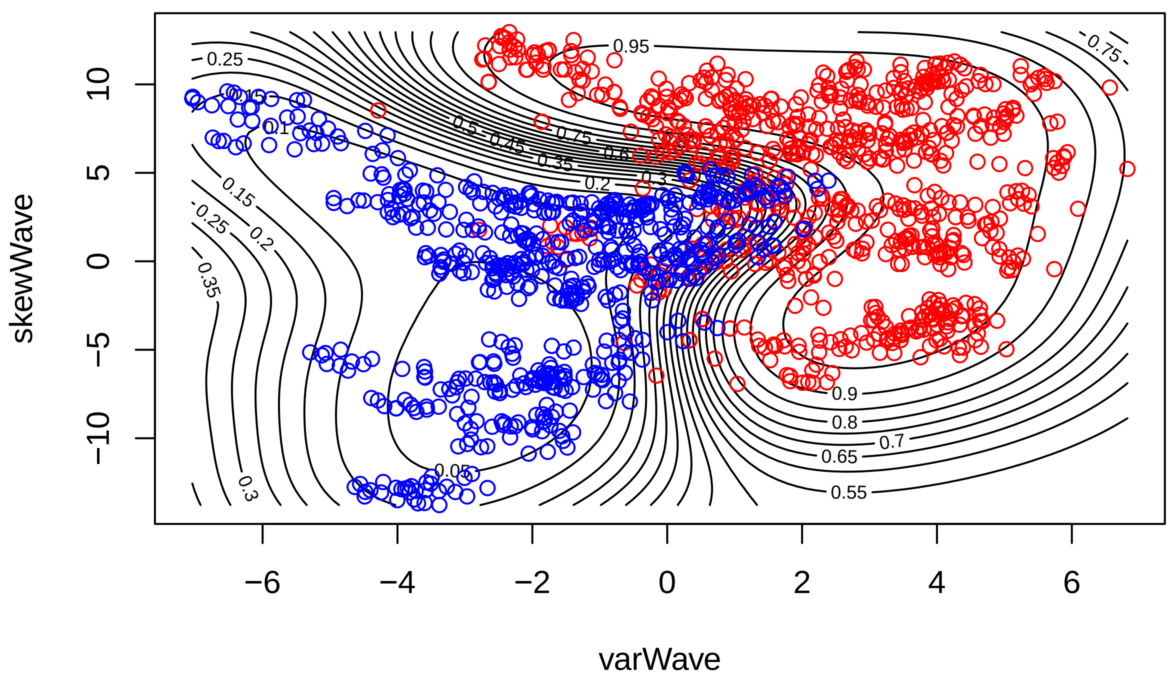
```
##
## predict.training   0   1
##                0 503  18
##                1  41 438
```

```r
accuracy.training <- sum((diag(cmatrix.training))) / sum(cmatrix.training)
accuracy.training
```

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

```r
x1 <- seq(min(data$varWave), max(data$varWave), length=100)
x2 <- seq(min(data$skewWave), max(data$skewWave), length=100)
gridPoints <- meshgrid(x1,x2)
gridPoints <- cbind(c(gridPoints$X), c(gridPoints$Y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(data)[1:2]

probPreds <- predict(GPfit, gridPoints, type="probabilities")
```

```
contour(x1, x2, matrix(probPreds[,1], 100, byrow = TRUE), 20,
        ylab = "skewWave", xlab = "varWave")

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



Predictions for the test set

```
# negative index means that it excludes does indexes from the vector
predict.test <- predict(GPfit, data[-SelectTraining,])
cmatrix.test <- table(predict.test, data[-SelectTraining, 5])
cmatrix.test
```

```
##
## predict.test   0   1
##            0 199   9
##            1  19 145
```

```
accuracy.test <- sum((diag(cmatrix.test))) / sum(cmatrix.test)
accuracy.test
```

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

Predictions when using all four covariates

```r
# new model with all covariates
GPfit <- gausspr(fraud ~ ., data.training)
```

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

```r
predict.final <- predict(GPfit, data[-SelectTraining,])
cmatrix.final <- table(predict.final, data[-SelectTraining, 5])
cmatrix.final
```

```
##
## predict.final   0   1
##             0 216   0
##             1   2 154
```

```r
accuracy.final <- sum((diag(cmatrix.final))) / sum(cmatrix.final)
accuracy.final
```

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

Discussion The accuracy is better for when were using four covariates instead of two. This is obvious because if the other variables depend on the one we are interested in of course we are going to get better results with more variables/data.