Lab 4 - Gaussian Processes

David Björelind

10/18/2020

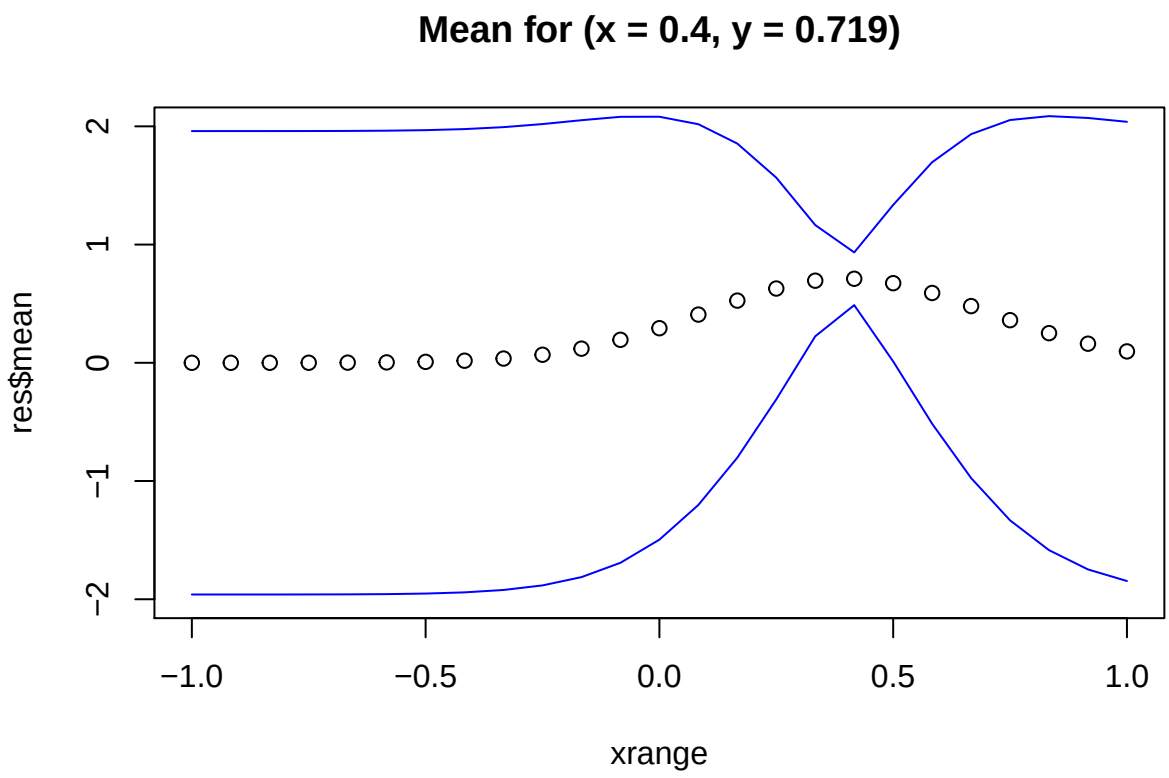
Assignment 1 - Implementing GP Regression

2.1) (1)

```
# Simulating from posterior distribution of f
posteriorGP = function(X, y, XStar, sigmaNoise, k, sigmaF, l){

  K = k(X,X, sigmaF^2, l)
  n = length(XStar)
  L = t(chol(K + sigmaNoise^2*diag(dim(K)[1])))
  kStar = k(X,XStar, sigmaF^2, l)
  alpha = solve(t(L), solve(L,y))

  FStar = t(kStar) %*% alpha
  v = solve(L, kStar)
  vf = k(XStar, XStar, sigmaF^2, l) - t(v)%*%v #+ sigmaNoise*diag(n) #Adding sigma for noise
  logmarglike = -t(y)%*%alpha/2 - sum(diag(L)) - n/2*log(2*pi)
  # Returns a vector with the posterior mean and variance
  return(list("mean" = FStar,"variance" = vf,"logmarglike" = logmarglike))
}
```

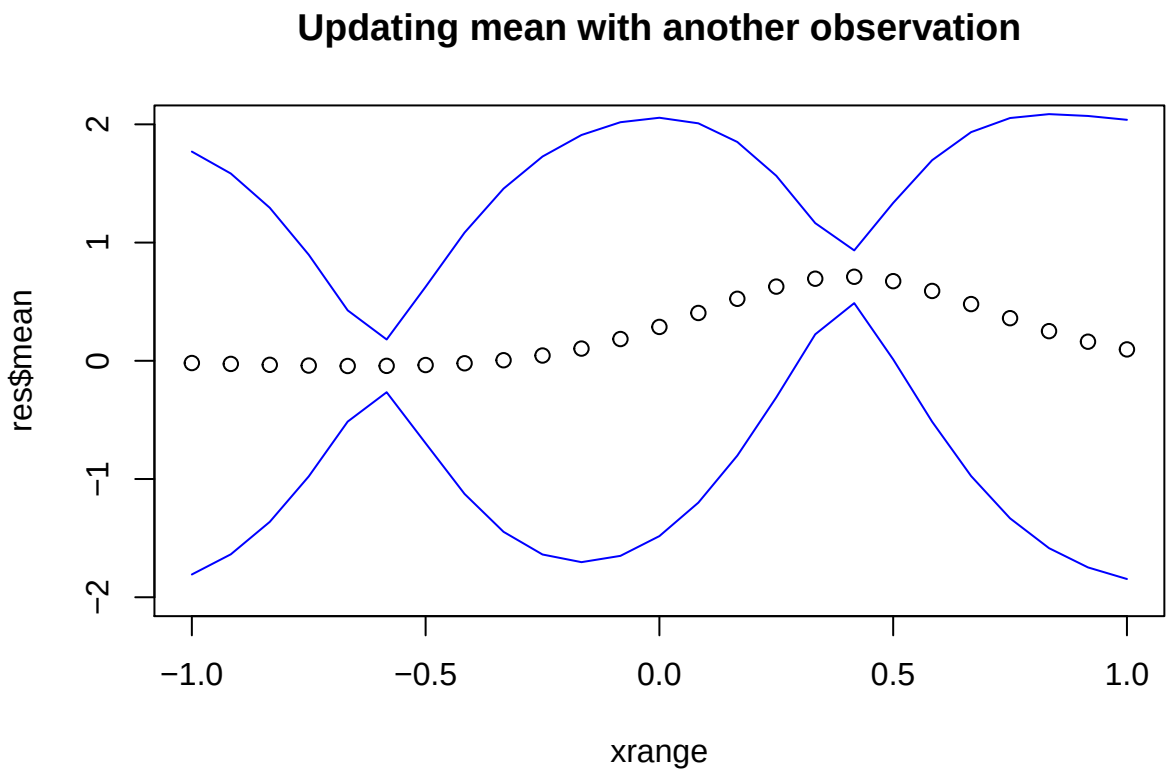
2.1) (2) GP Regression with kernlab

```
sigmaf = 1 # 1
sigman = 0.1 # 0.1
l = 0.3 # 0.3
x = 0.4
y = 0.719
xrange = seq(-1,1, length=25)

res = posteriorGP(x, y, xrange, sigman, SquaredExpKernel, sigmaF = sigmaf, l=l)

# (2) Plotting posterior mean and 95% interval bands
plot(x = xrange, y = res$mean, ylim = c(-2,2), main = "Mean for (x = 0.4, y = 0.719)")
lines(x = xrange, y = res$mean + sqrt(diag(res$variance))*1.96, col="blue", type="l")
lines(x = xrange, y = res$mean - sqrt(diag(res$variance))*1.96, col="blue",  type="l")
```



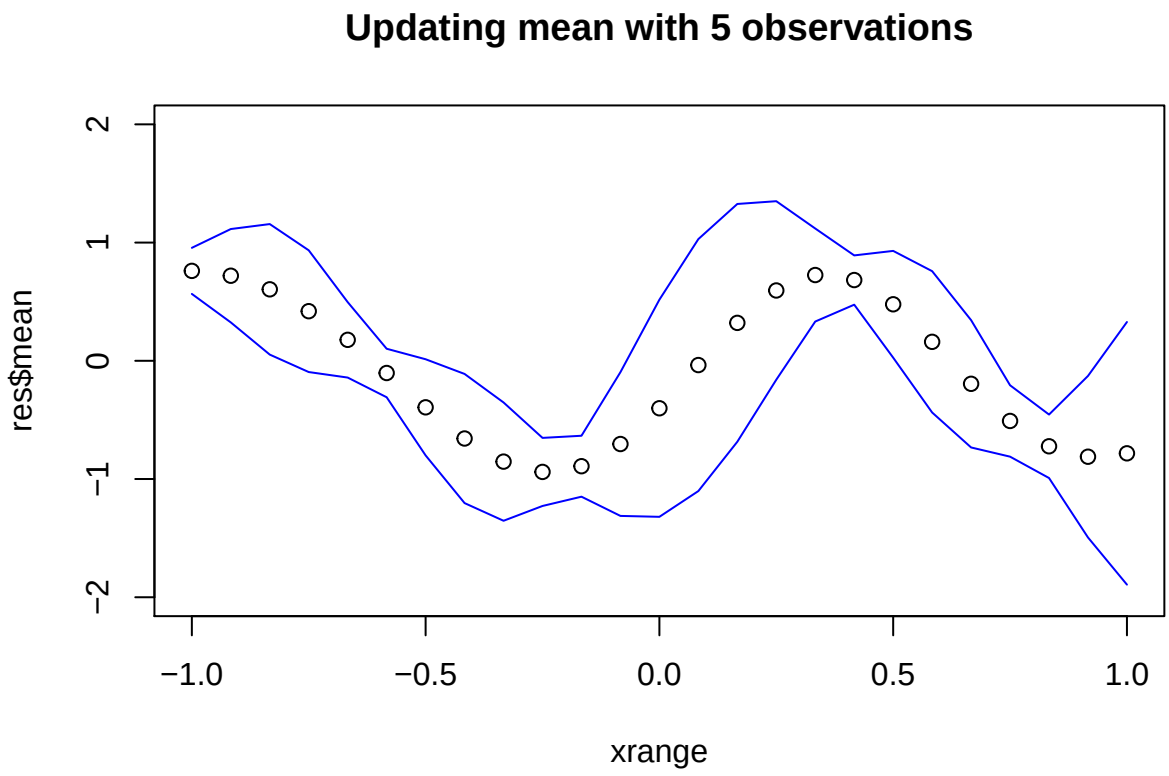
2.1) (3)

```
# (3)
x = c(0.4,-0.6)
y = c(0.719,-0.044)

res = posteriorGP(x, y, xrange, sigman, SquaredExpKernel, sigmaF = sigmaf, l=l)
plot(x = xrange, y = res$mean, ylim = c(-2,2), main = "Updating mean with another observation")
lines(x = xrange, y = res$mean + sqrt(diag(res$variance))*1.96, col="blue", type="l")
lines(x = xrange, y = res$mean - sqrt(diag(res$variance))*1.96, col="blue",  type="l")
```



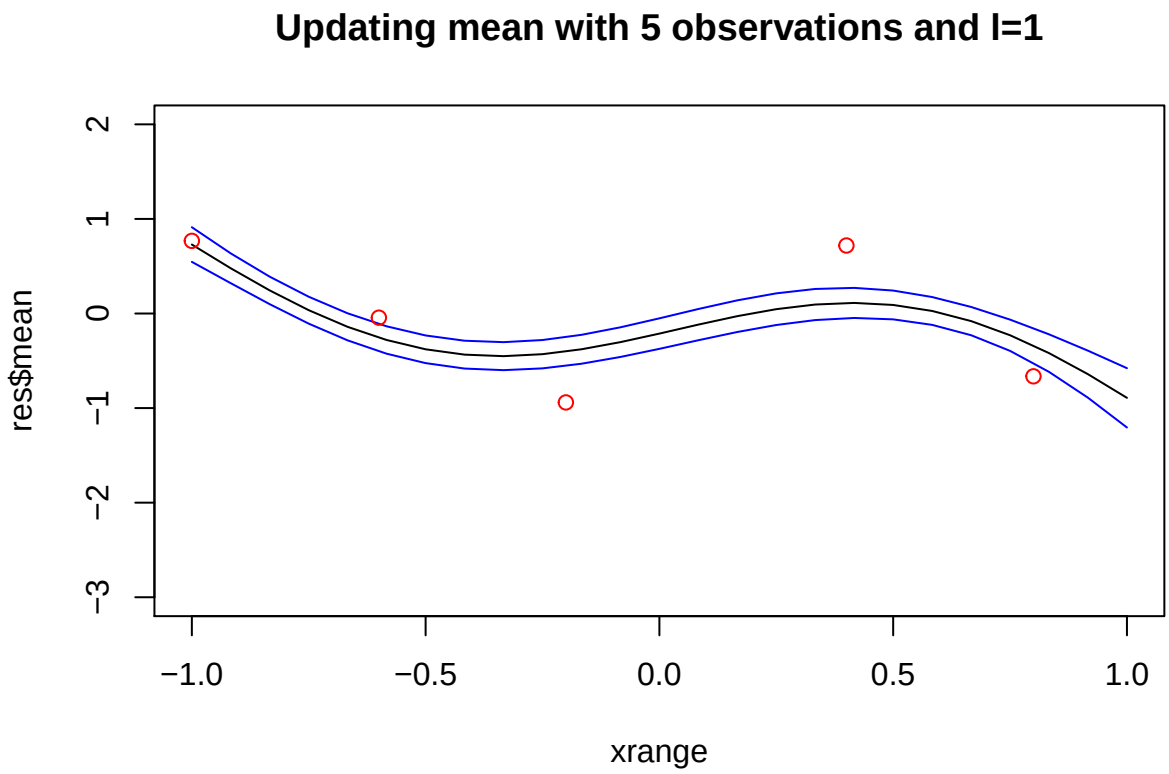
2.1) (4)

```
# (4)
x = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
sigmaf = 1
l = 0.3
res = posteriorGP(x, y, xrange, sigman, SquaredExpKernel, sigmaF = sigmaf, l=l)
plot(x = xrange, y = res$mean, ylim = c(-2,2), main = "Updating mean with 5 observations")
lines(x = xrange, y = res$mean + sqrt(diag(res$variance))*1.96, col="blue", type="l")
lines(x = xrange, y = res$mean - sqrt(diag(res$variance))*1.96, col="blue",  type="l")
```



2.1) (5)

```
# (5)
x = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
sigmaf = 1
l = 1
res = posteriorGP(x, y, xrange, sigman, SquaredExpKernel, sigmaF = sigmaf, l=l)

plot(x = xrange, y = res$mean, ylim = c(-3,2), main = "Updating mean with 5 observations and l=1", type=
lines(x = xrange, y = res$mean + sqrt(diag(res$variance))*1.96, col="blue", type="l")
lines(x = xrange, y = res$mean - sqrt(diag(res$variance))*1.96, col="blue",  type="l")
lines(x = x, y = y, col="red",  type="p")
```



When **l=1**, the produced function will be more smooth compared to (4). We also see that the bands created are much more narrow compared to plot produced in (4). This is underfitting of the data. A high value for **l** allows for a lower space between the probability bands. This means that we have to move further in other to get a change in correlation than with a small **l**.

Assignment 2 - GP Regression with kernlab

2.2 (1)

```
time = seq(from=1, to=2190, by=5)
temps = temps[time]
day = rep(seq(from=1, to=361, by=5), times=6)
# Data scaling
daymean = mean(day)
daysd = sd(day)
timemean = mean(time)
timesd = sd(time)
tempsmean = mean(temps)
tempssd = sd(temps)

day_s = scale(day)
time_s = scale(time)
temps_s = scale(temps)
```

2.2 (1)

```
SEKernel = function(x1, x2, ell = 1, sigmaf = 1){
  r = sqrt(sum((x1 - x2)^2))
  return(sigmaf^2*exp(-r^2/(2*ell^2)))
}

SEKernel(x1=1, x2=2, ell = l, sigmaf = sigmaf)
```

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

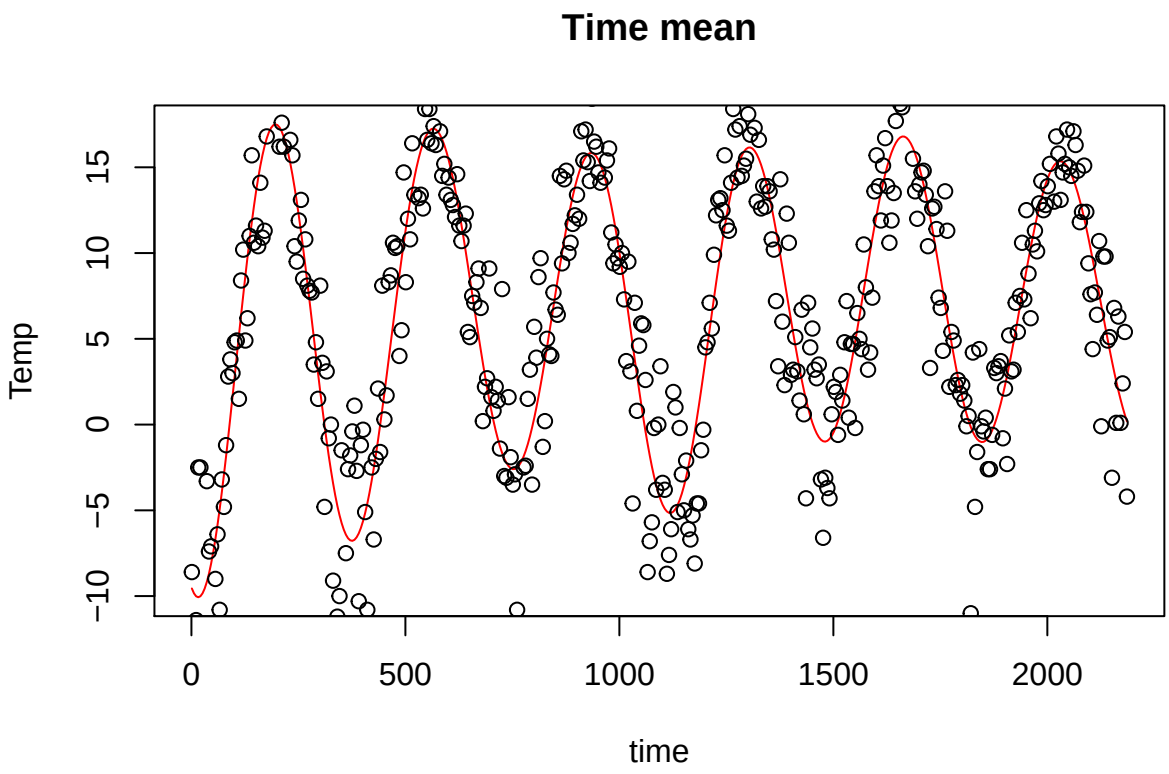
```
X = c(1,3,4)
XStar = c(2,3,4)
kernelMatrix(kernel = SEKernel, x=X, y=XStar)
```

```
## 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
```

```
SEKernelfunc = function(ell, sigmaf){
  kernel = function(x1, x2){
  r = sqrt(sum((x1 - x2)^2))
  return(sigmaf^2*exp(-r^2/(2*ell^2)))
  }
  class(kernel) <- "kernel"
  return(kernel)
}
```

2.2 (2)

```
ell = 0.3
sigmaf = 20

# Getting the error term for the GP
polyFit <- lm(temps_s ~  time_s + I(time_s^2))
sigma = sd(polyFit$residuals)
GPfit = gausspr(x=time_s, y=temps_s, kernel = SEKernelfunc(ell, sigmaf), var = sigma^2, type="regressio
meanpred = predict(GPfit, time_s)

# Plotting the means from time data
plot(x = time, y = meanpred*tempssd+tempsmean, col="red", type='l', ylab="Temp", main="Time mean")
lines(x = time, y = temps,  type="p")
```

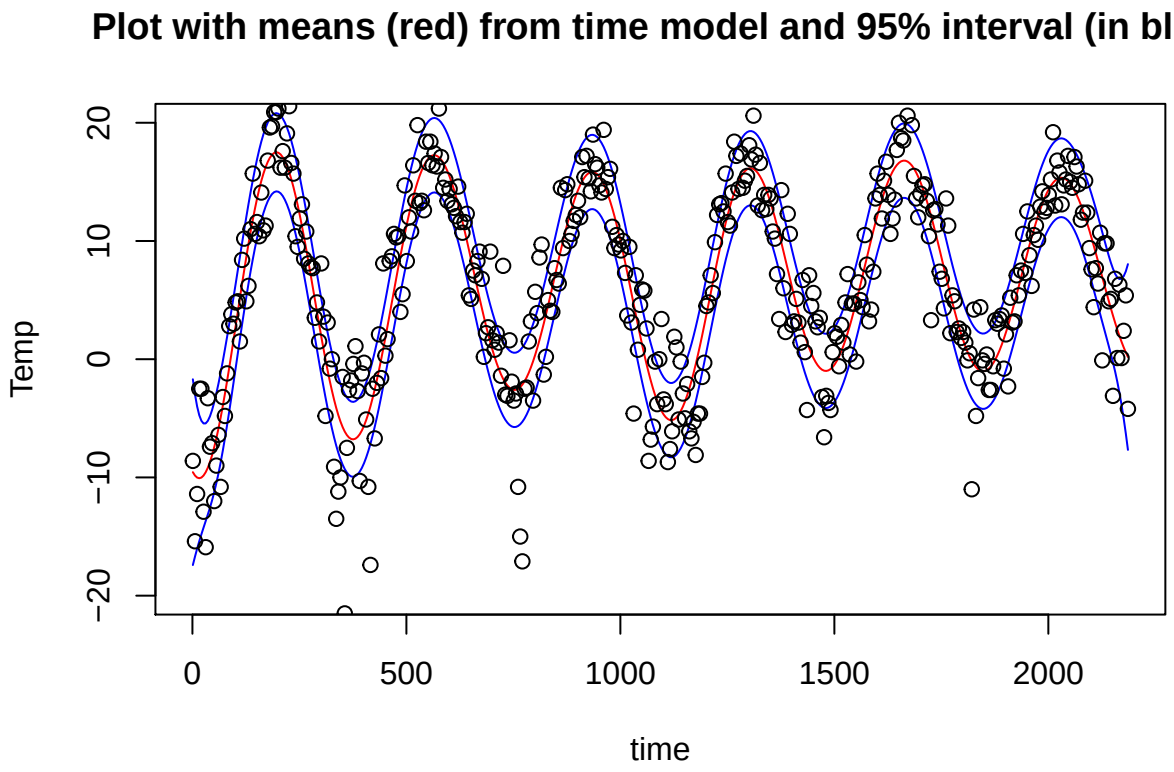


A higher value for ell increases smoothness. A higher value for sigmaf increases the possible ranges of the predicted means. A lower value gives lower covariance between prediction. This explains the difference in the highest and lowest values for different sigmafs. Judging from the mean we can seen it misses values in the highs and in the lows.

2.2 (3) Probability bands

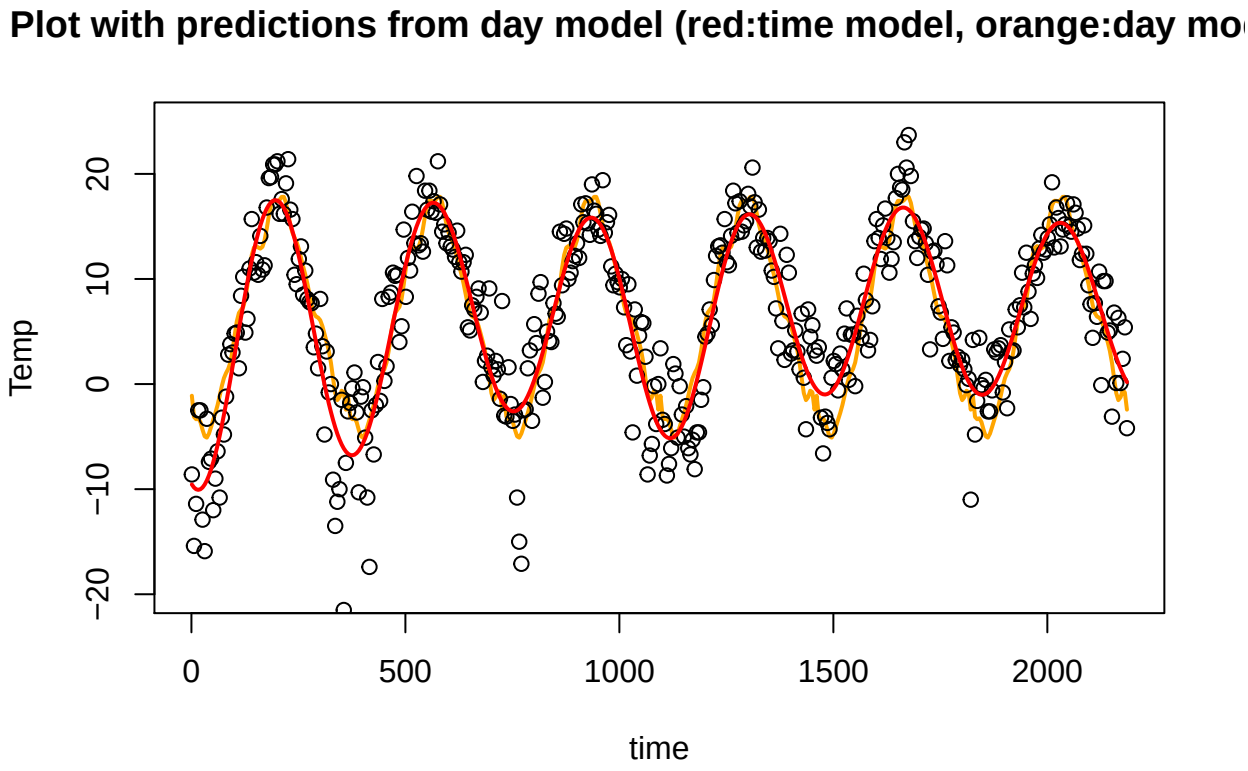
```
# Modified posteriorGP function
posteriorGP = function(X, y, XStar, sigmaNoise, k){
  K = kernelMatrix(k,X,X)
  n = length(XStar)
  L = t(chol(K + sigmaNoise^2*diag(dim(K)[1])))
  kStar = kernelMatrix(k,X,XStar)
  alpha = solve(t(L), solve(L,y))
  FStar = (t(kStar)) %*% alpha
  v = solve(L, kStar)
  vf = kernelMatrix(k,XStar, XStar) - t(v)%*%v #+ sigmaNoise^2*diag(n) #Adding sigma for noise
  logmarglike = -t(y)%*%alpha/2 - sum(diag(L)) - n/2*log(2*pi)

  # Returns a vector with the posterior mean and variance
  return(list("mean" = FStar,"variance" = vf,"logmarglike" = logmarglike))
}
sigmaf = 20
var = posteriorGP(X=time_s, y=temps_s, XStar=time_s, sigmaNoise=sigma, k=SEKernelfunc(ell, sigmaf))
```

```
plot(time, meanpred*tempssd+tempsmean, col="red", type='l', ylim=c(-20,20), main="Plot with means (red)
lines(x = time, y = meanpred*tempssd+tempsmean + sqrt(diag(var$variance))*1.96*tempssd, col="blue", typ
lines(x = time, y = meanpred*tempssd+tempsmean - sqrt(diag(var$variance))*1.96*tempssd, col="blue",  ty
lines(x = time, y = temps,  type="p")
```



2.2 (4) Day model

```
ell = 0.2
sigmaf = 20

# Getting the error term for the GP, using day instead of time
polyFit <- lm(temps_s ~  day_s + I(day_s^2))
sigma = sd(polyFit$residuals)
GPfit = gausspr(x=day_s, y=temps_s, kernel = SEKernelfunc(ell, sigmaf), var = sigma^2, type="regression
meanpred_day = predict(GPfit, day_s)

plot(time, meanpred_day*tempssd+tempsmean, col="orange", type='l', ylim=c(-20,25), main="Plot with predi
#lines(x = time, y = meanpred*tempssd+tempsmean + sqrt(diag(var$variance))*1.96*tempssd, col="blue", ty
#lines(x = time, y = meanpred*tempssd+tempsmean - sqrt(diag(var$variance))*1.96*tempssd, col="blue",  t
lines(x = time, y = temps,  type="p")
lines(x = time, y = meanpred*tempssd+tempsmean, col="red",  type="l", lwd=2)
```



Pros and cons of each model?

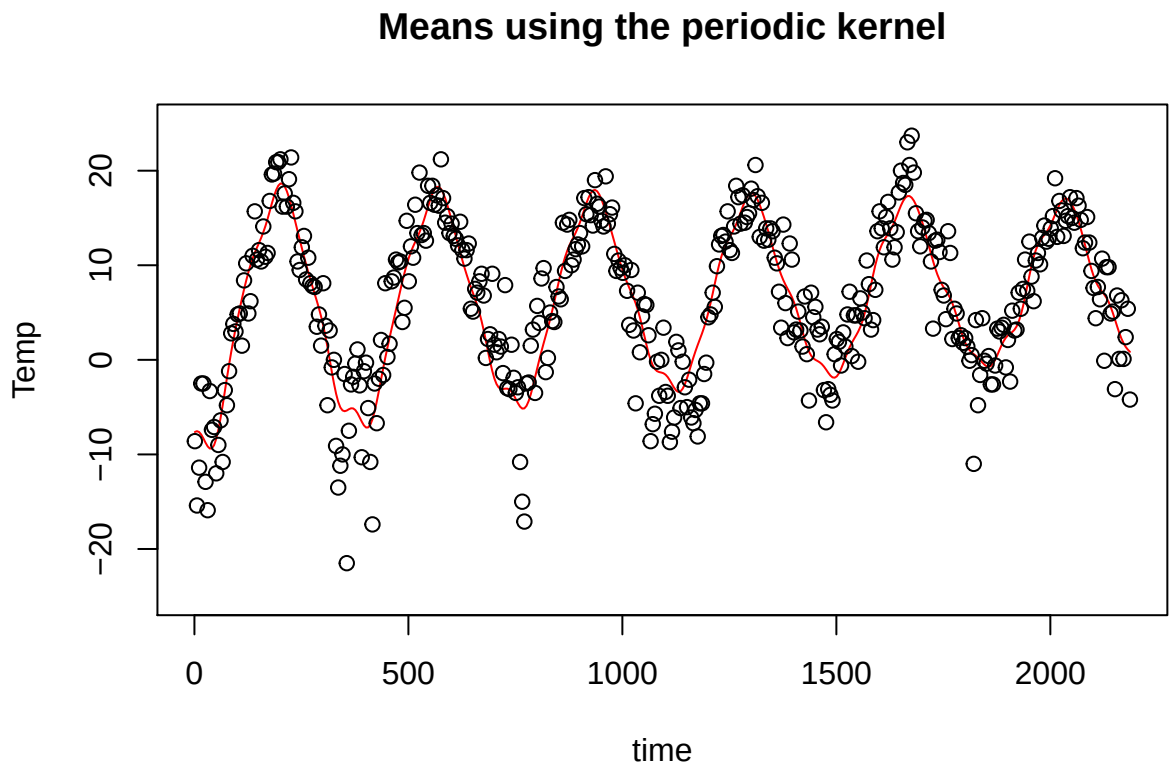
Days: Not smooth when changing year because the model cannot distinguish between different years.

Time: Produces smooth graphs and takes advantage of differences between years. This is seen in the graph where we seem to have an upwards trend.

2.2 (5)

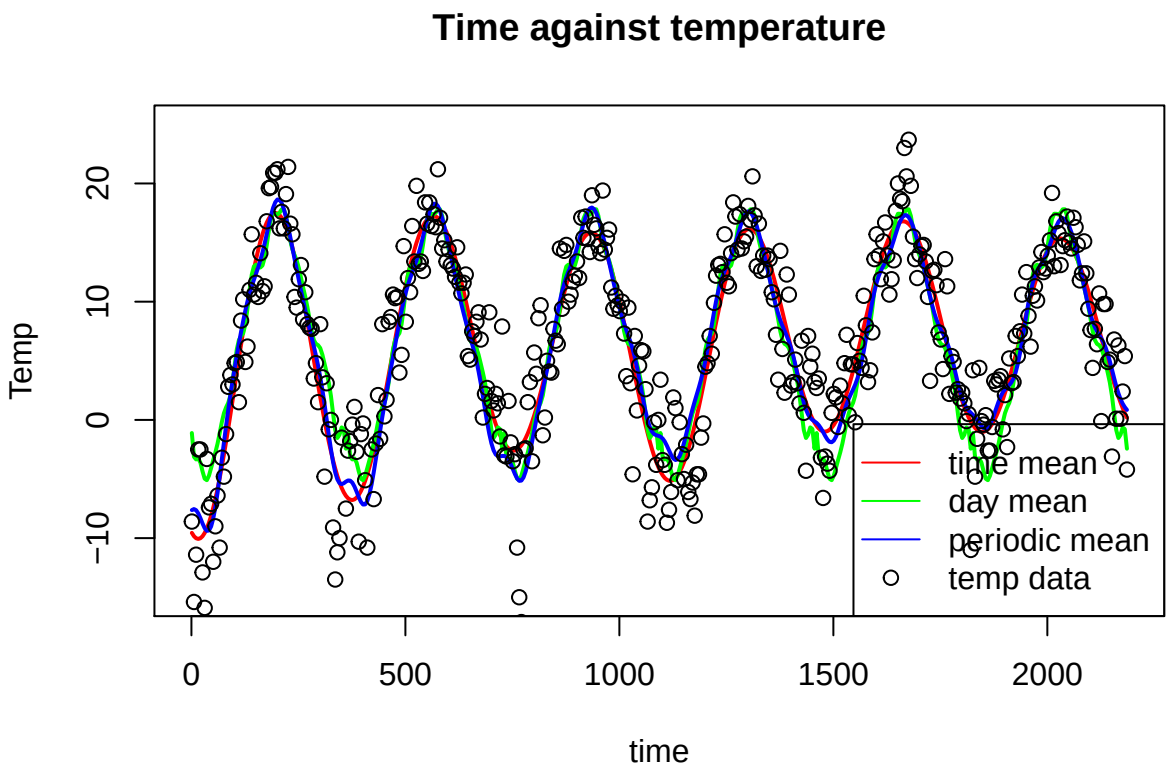
```
periodickernelfunc = function(sigmaf, l1, l2, d){
  periodicKernel = function(x1, x2){
  r = sqrt(sum((x1 - x2)^2))
  one = exp(-2*sin(pi*r/d)^2/l1^2)
  two = exp(-r^2/(2*l2^2))
  return(sigmaf^2*one*two)
  }
  class(periodicKernel) <- "kernel"
  return(periodicKernel)
}

# Hyperparameters
sigmaf = 20
l1 = 1
l2 = 10
d = 365/sd(time)
```

```r
# Getting the error term for the GP
polyFit <- lm(temps_s ~  time_s + I(time_s^2))
sigma = sd(polyFit$residuals)
GPfit = gausspr(x=time_s, y=temps_s, kernel = periodickernelfunc(sigmaf, l1, l2, d), var = sigma^2, typ
meanpred_5 = predict(GPfit, time_s)

# Plotting the means!
plot(x = time, y = meanpred_5*tempssd+tempsmean, col="red", type='l', ylim=c(-25,25), main = "Means usi
#lines(x = time, y = meanpred_5*tempssd+tempsmean + sqrt(diag(var$variance))*1.96*tempssd, col="blue",
#lines(x = time, y = meanpred_5*tempssd+tempsmean - sqrt(diag(var$variance))*1.96*tempssd, col="blue",
lines(x = time, y = temps,  type="p")
```



Plot of all the means

```r
plot(x = time, y = meanpred*tempssd+tempsmean, col="red", type='l', ylim=c(-15,25),main = "Time against
lines(x = time, y = meanpred_day*tempssd+tempsmean, col="green", type='l', lwd=2)
lines(x = time, y = meanpred_5*tempssd+tempsmean, col="blue", type='l', lwd=2)
lines(x = time, y = temps, col="black", type='p')
legend("bottomright", legend=c("time mean", "day mean", "periodic mean", "temp data"), lwd=c(1,1,1,NA),
```



We can see that the periodic kernel produces produces more sharp "valleys" and more even "tops" compared to the more smooth Squared Exponential kernel. This can be explained by the use of the periodic kernel. The model now gives higher correlation for data points years forward and backwards. This is due to the periodic nature of the data, which is translated into our model.

Day model captures better extreme values than time model does. One thing in common is that the tops and bottom has the same shape, which is not true for the periodic model. We also see, for the day model, that all the years are repeating. It does not know which year it is on. This makes it so that the model does not capture differences in different years, like the time model does. It can be interpreted as the model does a prediction on a certain date and has access to 6 data point for that date.

Assignment 3 - GP with classifications

2.3 (1)

```
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)

train_data = data[SelectTraining,]
test_data = data[-SelectTraining,]
```

```
GPfitfraud <- gausspr(fraud ~  varWave + skewWave, data=train_data)
```

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

```
GPfitfraud
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.42415004757028
##
## Number of training instances learned : 1000
## Train error : 0.059
```

```
# Predict on the training set
pred = predict(GPfitfraud,train_data)
# Confusion matrix
CM=table(pred, data[SelectTraining,5])
CM
```

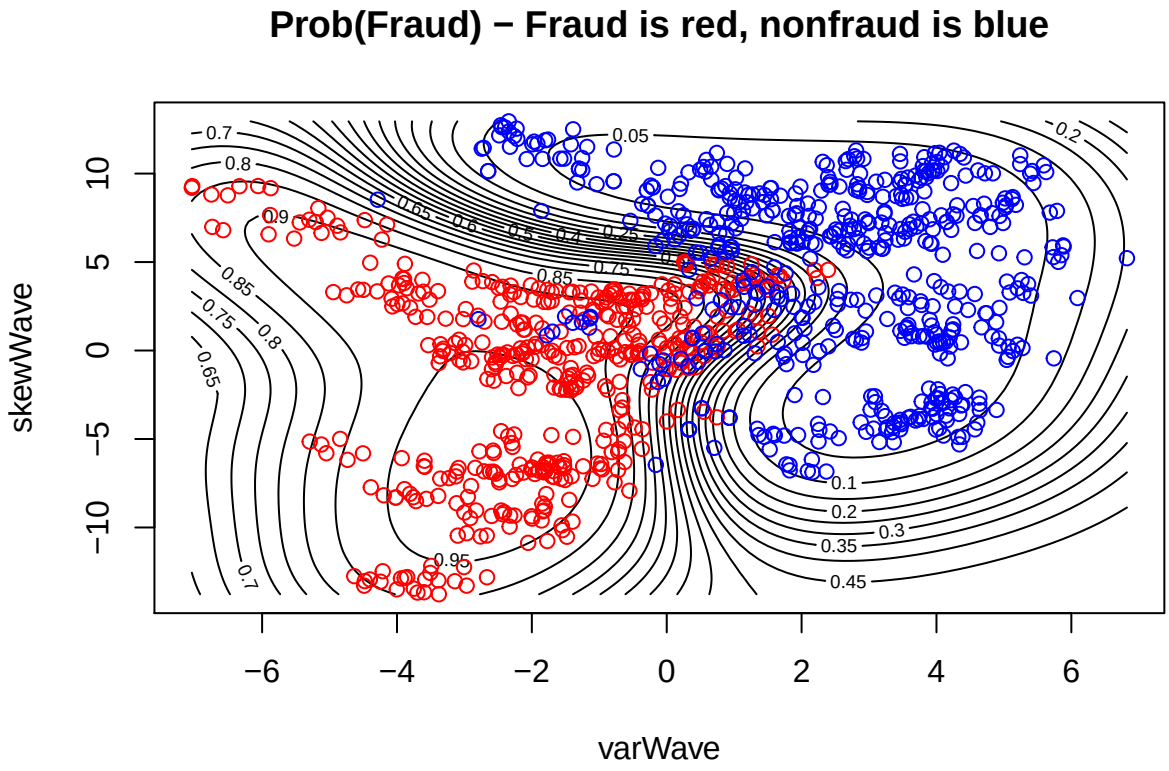
```
##
## pred   0   1
##    0 503  18
##    1  41 438
```

```
# accuracy
sum(diag(CM))/sum(CM)
```

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

Plotting the data

```
probPreds <- predict(GPfitfraud, test_data, type="probabilities")
varWave <- seq(min(data[,1]),max(data[,1]),length=100)
skewWave <- seq(min(data[,2]),max(data[,2]),length=100)
gridPoints <- meshgrid(varWave, skewWave)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

# Plotting for Prob(fraud)
contour(varWave, skewWave, matrix(probPreds[,2], 100, byrow = TRUE), 20, xlab = "varWave", ylab = "skew
points(train_data[train_data$fraud==1,1], train_data[train_data$fraud==1,2],col="red")
points(train_data[train_data$fraud==0,1], train_data[train_data$fraud==0,2],col="blue")
```



2.3 (2)

```
# Predict on the test set
pred = predict(GPfitfraud,test_data)
# Confusion matrix (test)
conf = table(pred, data[-SelectTraining,5])
print(conf)
```

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

```
# Computing accuracy
sum(diag(conf))/sum(conf)
```

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

2.3 (3)

```
GPfitfraud <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=train_data)
```

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

```
GPfitfraud
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.438275644976386
##
## Number of training instances learned : 1000
## Train error : 0.003
```

```
# Predict on the test set
pred_new = predict(GPfitfraud,test_data)
# Confusion matrix (test)
conf_new = table(pred_new, data[-SelectTraining,5])
print(conf_new)
```

```
##
## pred_new   0   1
##        0 216   0
##        1   2 154
```

```
# Computing accuracy
sum(diag(conf_new))/sum(conf_new)
```

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

We can see that accuracy is much higher when using all four covariates when predicting fraud. Only two cases were incorrectly classified! A reason might be that the model has access to more datapoints for each prediction made, which makes it better.