Lab 4 - Gaussian Processes

William Bergekrans

2021-10

Part 1 - Implementing GP Regression

The GP regression model is:

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

Define the kernel and posteriorGP function

Now I create the squared exponential kernel and a function for simulating from the posterior distribution of f. It is assumed that the prior mean of f is zero for all x, the created functions look as follows:

```
# Squared exponential kernel function
# This function is copied from the course website.
SE_kernel <- 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)
}

# Function for simulation from the posterior distribution of f
# x: Vector of training input
# y: Vector of training targets/ outputs
# xStar: Vector of inputs where the posterior distribution is evaluated.
# sigmaNoise: Noise standard deviation
# K: Covariance function or kernel.
#
# Returns a vector with the posterior mean and variance of f.
posteriorGP <- function(x, y, xStar, sigmaNoise, K, ...) {
  # Get the squared noise diagonal matrix. dimension x*x.
  s2_matrix = (sigmaNoise^2)* diag(1, length(x))

  # Calculate L. Transpose it to get the correct dimensions.
  L = t(chol(K(x, x, ...) + s2_matrix))

  # predictive mean
  alpha = solve(t(L), solve(L, y))
  f_star= t(K(x, xStar, ...)) %*% alpha

  # predictive variance
  v = solve(L, K(x, xStar, ...))
```

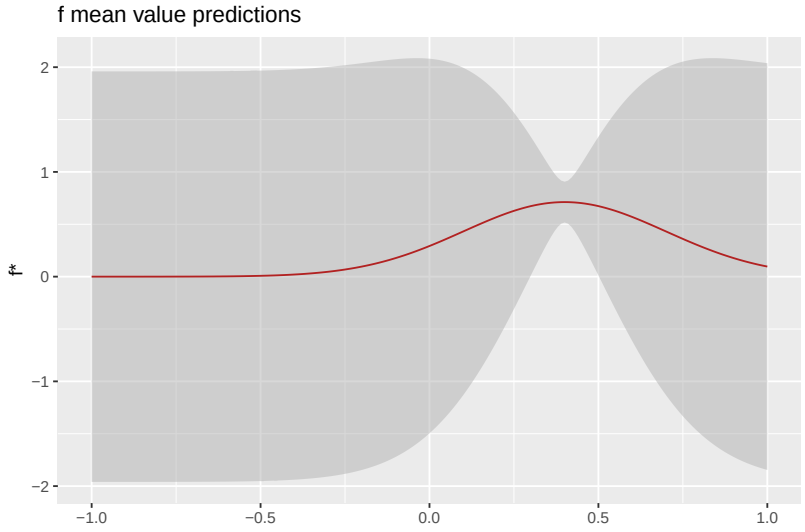
```
  Vf = K(xStar, xStar, ...) - (t(v)%*%v)

  return(list("fStar" = f_star, "vf"= Vf))
}
```

Posterior Mean for f

Now the function for simulation from the posterior distribution shall be used to plot the posterior mean for f over the interval [-1,1]. The prior is updated with the point (0.4, 0.719). The parameters used are $\sigma_n = 0.1$, $\sigma_f = 0.1$ and $l = 0.3$.

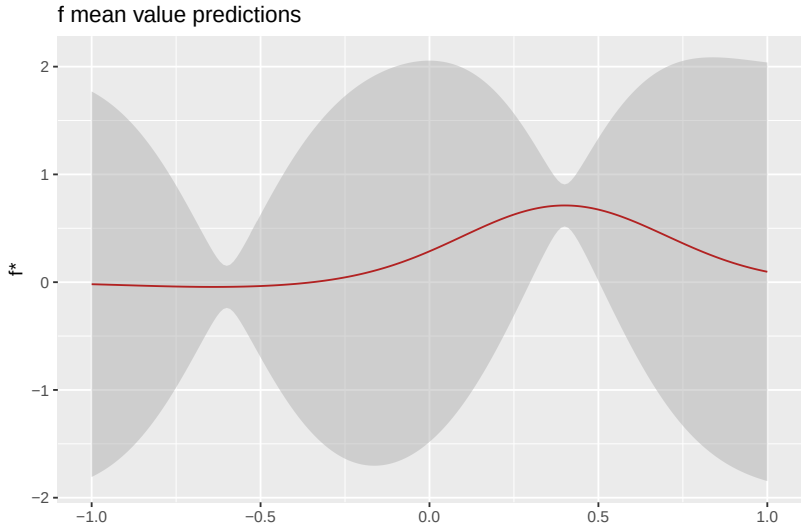
The predictive intervals for f_* is $mean(f_*) \pm 1.96 * \sqrt{(var(f_*))}$.

In the following plot the simulations are done with a single training point for the prior, which is [0.4, 0.719].



As the predictions bands shows the simulations reflect the uncertainty in our data/ prior. We only have information about one point and therefore the predictive bands get very wide for points far away from 0.4. However, the kernel makes sure that the function will be quite smooth and the prediction band is relatively small for values close to 0.4.

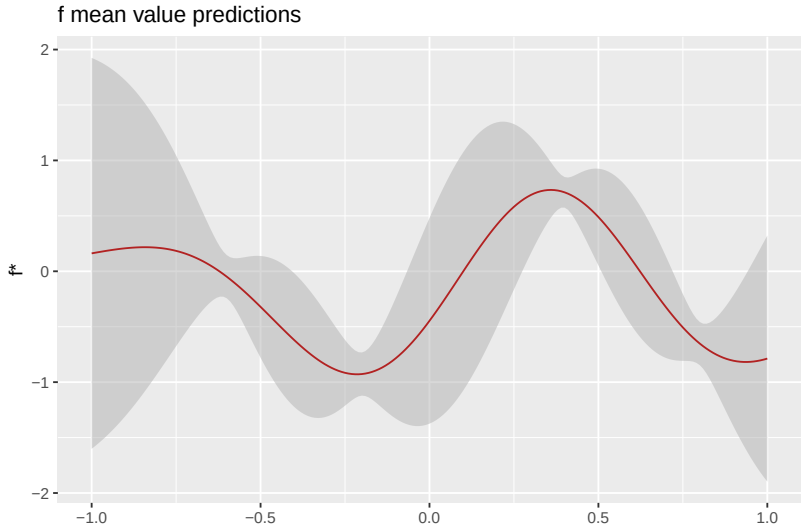
In the following graph another observation is added to the prior. Now I also use the point [-0.6, -0.044].



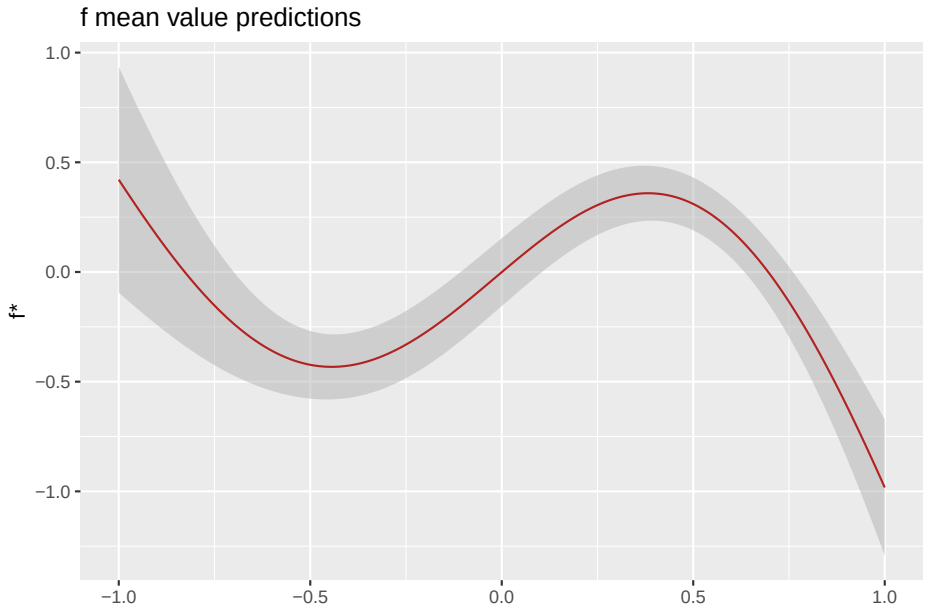
Only by adding one point both the mean value and prediction band get a very different shape. Even though there are only two points the prediction band is quite narrow for all points.

I believe this is because the kernel forces the distribution to be this way, as to reach the smoothness goals etc, which are implied by the hyper parameters.

In the next plot the simulations are made with 5 points in the prior distribution.



For the following simulations the length scale (l) is set to 1, previously it was 3.



A smaller L makes the function less smooth. This is expected as L is the length scale, what L does it that points need to be close together in order to be highly correlated. In this case where L is smaller than the previous graphs the function is no longer as restricted (two points close to each other does not have to be very similar anymore), which creates a less smooth graph.

The code for part 1 is:

```
simulate <- function(x, y, sigmaNoise, K, ...) {
  xSeq = seq(-1, 1, 0.01)
  # Simulate from the posterior of f.
  pGP =
    posteriorGP(x, y, xSeq, 0.1, K, ...)

  # Confidence interval
  ci_min = pGP$fStar + 1.96 * sqrt(diag(pGP$vf))
  ci_max = pGP$fStar - 1.96 * sqrt(diag(pGP$vf))

  ggplot(as.data.frame(pGP), aes(x = xSeq, y = fStar)) +
    geom_ribbon(aes(ymin = ci_min, ymax = ci_max),
                fill = "grey70",
                alpha = 0.5) +
    geom_line(color = "firebrick") +
    ggtitle("f mean value predictions") +
    xlab(" ") +
    ylab("f*")
}

simulate(c(0.4), c(0.719), 0.1, SE_kernel)

simulate(c(0.4, -0.6), c(0.719, -0.044), 0.1, SE_kernel)
simulate(c(0.4, -0.6, -0.2, 0.4, 0.8), c(0.719, -0.044, -0.940, 0.719, -0.664), 0.1, SE_kernel)
simulate(c(0.4, -0.6, -0.2, 0.4, 0.8), c(0.719, -0.044, -0.940, 0.719, -0.664), 0.1, SE_kernel, l=1)
```

GP Regression with kernlab

Part 1

In this part the package kernlab in R will be used for Gaussian Process Regression. The data set consist of temperature measurements in Stockholm over a 5 year period.To make computations less demanding only every 6th observation is used. For every observation two new parameters are created, time and day. time is the number of days since the first observation and day is the number of the since the start of the year (when the measurement was taken).

The squared exponential kernel defined in part 1 of this lab is reused. This kernel is used with the kernelMatrix-function to get the covariance matrix for X and X_*. The matrix $K(X, X_*)$ is as follows:

	1	3	4
2	0.6065307	0.1353353	0.0111090
3	0.6065307	1.0000000	0.6065307
4	0.1353353	0.6065307	1.0000000

The kernel test for the two points returned the covariance 0.6065307.

Part 2

Now I consider the following model:

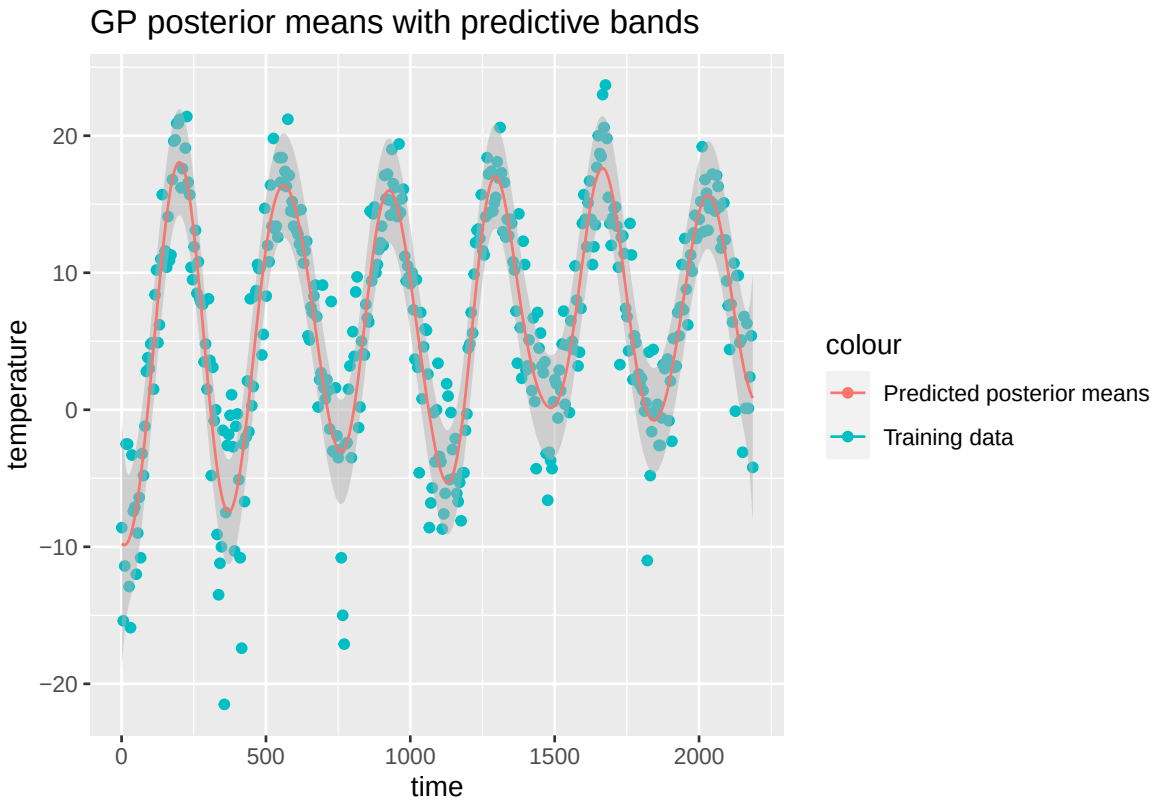
$$temp = f(time) + \epsilon \text{ with } \epsilon \sim N(0, \sigma_n^2) \text{ and } f \sim GP(0, k(time, time'))$$

σ_n^2 is the residual variance from a simple quadratic regression fit. I shall now estimate the GP regression using the Squared Exponential function that I defined above. σ_n^2 is calculated by taking the standard deviation of all the residuals from the lm-fit. When calculating the regression fit (with lm) temperature was the target and time the feature. For calculating the Gaussian Proccess fit $\sigma_f = 20$ and $l = 0.2$. This fit was used to predict values for every time in our training set. The result is as follows:

Compared to different values of σ_f and l this GP seem to fit the data well.

Part 3

Now I shall also compute the prediction bands of f. For this the variance is calculated using the posteriorGP function written in the first part of this lab. In order to get the same results as kernlab the temperatures are scaled before the function is run. The result is then unscaled using the standard deviation and mean of the original temperature measurements. The posterior band is the calculated with $mean(f_*) \pm 1.96 * \sqrt{var(f_*)}$. The result is as follows (grey areas is the posterior band):

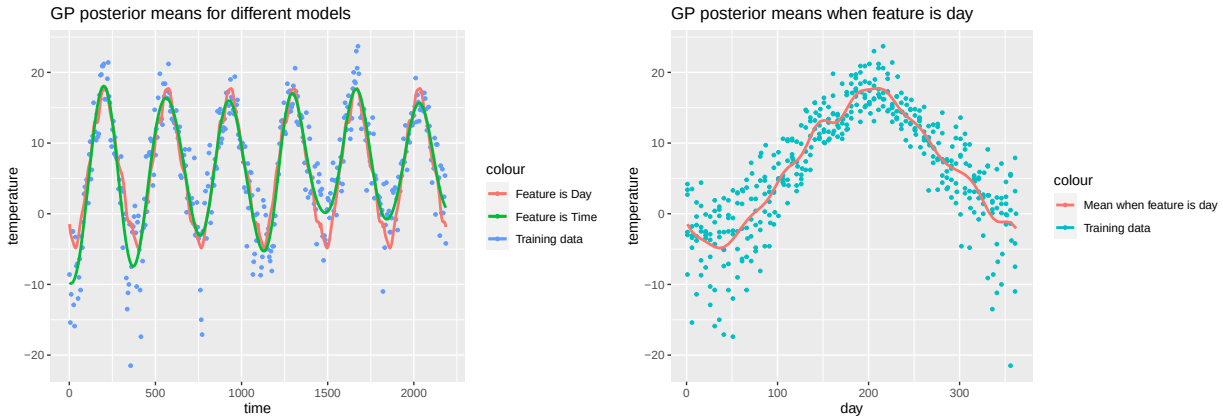


I believe this result is realistic because only a few data points fall outside the predictive band. By definition 95% of the observations should fall within the band, and this seem reasonable.

Part 4

Now I consider a new model that is based on the variable day instead of time:

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

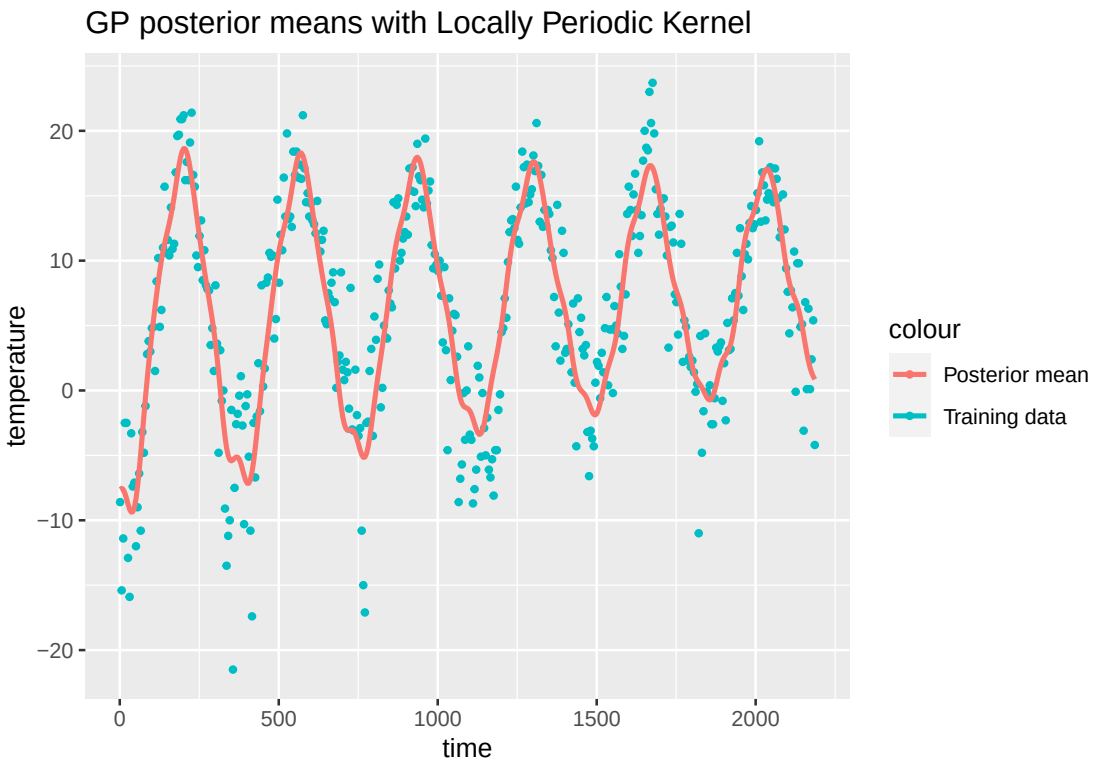


Depending on what model is used the resulting posterior means look quite different. Then the feature is time f is much smoother compared to when the feature is day. The model based on day seem to vary much more locally, which makes sense because the model has more data points for every 5th day. The problem with this model is that it is not good at accounting for differences over time, the temperature differences are higher over the year in the beginning compared to the last year in the measurements. The model where the feature is time can take this into account even though it may not be as good to spot smaller trends over a year.

As made obvious by the right graph the mean graph has the same shape for every year when day is the feature. The data sets the two models try to fit are therefore very different and it is not weird that the results are quite different.

Part 5

Now a locally periodic kernel is implemented. This kernel is then used to fit a GP regression model. Prediction on the time data are made using this model and the result is as follows:



This function is less smooth compared to the earlier models and is therefore more adaptable to local data. This model does manage to adapt over time similar to the model based purely on the time feature. The advantage the LPK model has over that one is that it can take into account trends that happen inside a given year. For example accelerations and deceleration of temperature changes.

The chosen kernel can obviously have a big impact on the generated GP model. By making the kernel take into account both time and date we can get a model in between the two previous. By only looking at the visuals the LPK model looks like the best option, however to be sure I would need to compare the models on some test data.

Code:

```
# Read temperature and relevant date.
temps <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")

# Time and day variables
time <- seq(1,2190,5)
day <- strptime(temps$date, format="%d/%m/%y")
day <- day$yday +1
day <- day[seq(1,2190,5)] # Keep every 5th observation

temps <- data.frame(
  "temperature" = temps$temp[seq(1,2190,5)],
  "day" = day,
  "time" = time
)

SE_kernlab <- function(sigmaf = 1, ell = 1)
{
  rval <-  function(x, y=NULL, sigmaF=sigmaf,l=ell){
    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)
}

# Test the defined kernel with X = 1, Xstar = 2
kernel_func <- SE_kernlab()
# kernel_func(1, 2)

# Calculate the covariance for our X and XStar.
X <- c(1,3,4)
Xstar <- c(2,3,4)
mat <- kernelMatrix(kernel = kernel_func, x = X, y = Xstar)
rownames(mat) <- Xstar
kable(mat, row.names = TRUE, col.names=X)
# Estimate sigma2_n
lm_fit <- lm(scale(temperature) ~ scale(time) + scale(time^2), temps)
```

```
sigma_n <- sd(lm_fit$residuals)

# Estimate the GP
GPfit <- gausspr(temps$time, temps$temperature, kernel = SE_kernlab, kpar =list(sigmaf = 20, ell=0.2), v

# predict
pred <- predict(GPfit, temps$time)

ggplot(temps, aes(x=time, y=temperature, color="Training data")) +
  geom_point(size=1) +
  geom_line(aes(y=pred, color="Predicted posterior means")) +
  ggtitle("GP posterior means on the training data")
postGP <- posteriorGP(scale(temps$time), scale(temps$temperature), scale(temps$time),
                      sigma_n, SE_kernel, l=0.2, sigmaF = 20)

pred_min <- postGP$fStar*sd(temps$temperature) + mean(temps$temperature)-1.96*sqrt((diag(postGP$vf)))*s

pred_max <- postGP$fStar*sd(temps$temperature) + mean(temps$temperature)+1.96*sqrt((diag(postGP$vf)))*s

ggplot(temps, aes(x=time, y=temperature, color="Training data")) +
  geom_point(size=1.5) +
  geom_ribbon(aes(ymin = pred_min, ymax = pred_max, color="Predictive band"),
              fill = "grey70",
              alpha = 0.5, colour = NA) +
  geom_line(aes(y=pred, color="Predicted posterior means")) +
  ggtitle("GP posterior means with predictive bands")

lm_fit <- lm(temperature ~ day + (day^2), temps)
sigma_n_day <- sd(lm_fit$residuals)

# Estimate the GP
GPfit <- gausspr(temps$day, temps$temperature, kernel = SE_kernlab, kpar =list(sigmaf = 20, ell=0.2), va

# predict
pred_day <- predict(GPfit, temps$day)

ggplot(temps, aes(x=time, y=temperature, color="Training data")) +
  geom_point(size=1) +
  geom_line(aes(y=pred_day, color="Feature is Day"), size=1) +
  geom_line(aes(y=pred, color="Feature is Time"), size=1) +
  ggtitle("GP posterior means for different models")

ggplot(temps, aes(x=day, y=temperature, color="Training data")) +
  geom_point(size=1) +
  geom_line(aes(y=pred_day, color="Mean when feature is day"), size=1) +
  ggtitle("GP posterior means when feature is day")
lpk_kernel <- function(sigmaf = 1, ell = c(1,1), d)
{
  rval <-  function(x, y=NULL, sigmaF=sigmaf,l=ell, de=d){
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
```

```
      K[,i] <- sigmaF^2 * exp(-2*((sin(pi*abs(x-y[i])/de)^2)/(l[1]^2))) * exp(-0.5*(abs(x-y[i])^2/(l[2]
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

kern <- lpk_kernel(sigmaf = 20, ell=c(1,10), d=365/sd(temps$time))

GPfit <- gausspr(temps$time, temps$temperature, kernel = kern, var = sigma_n)

pred_time <- predict(GPfit, temps$time)

ggplot(temps, aes(x=time, y=temperature, color="Training data")) +
  geom_point(size=1) +
  geom_line(aes(y=pred_time, color="Posterior mean"), size=1) +
  ggtitle("GP posterior means with Locally Periodic Kernel")
```

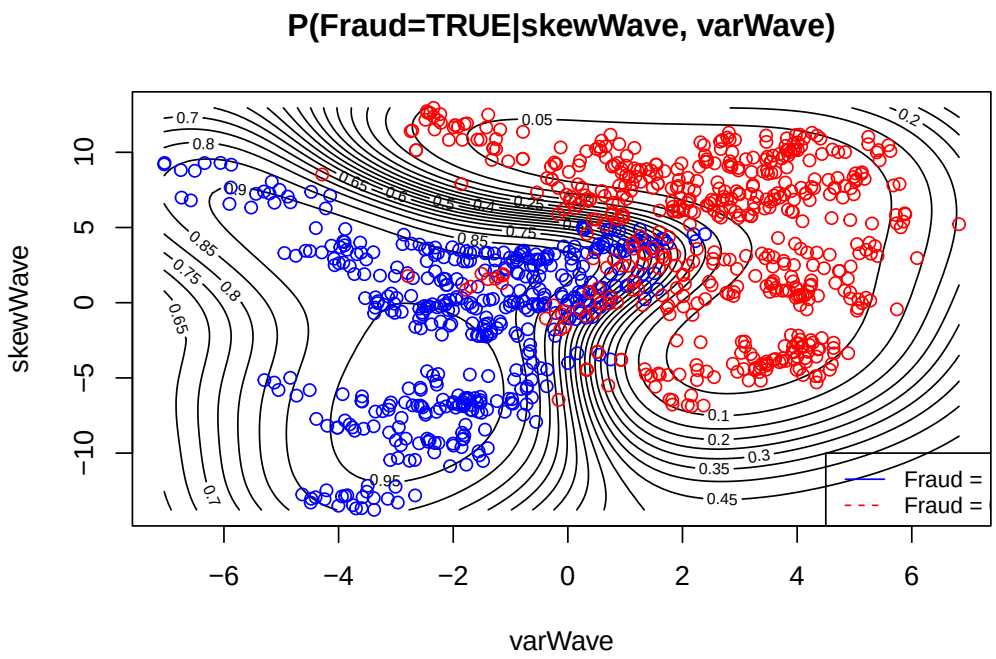
GP Classification with kernlab

In this part a new data set is used, which consist of banknote fraud data.

Part 1

First, 1000 observations are taken randomly from the data to be used for training. The rest of the data will be used for testing. A Gaussian process classification model is fit using the training data. The target for this model is fraud and the features are skewWave and varWave. The fitted GP model is used to predict over a grid of points in order to get prediction probabilities for predicting fraud as 1. These prediction probabilities are used to plot a contour graph where the data points are added.

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



The contours seem to match the training data set well. There are some red dots which will be misclassified and some blue as well. However, most data points are very likely to receive the correct classification which indicates that the model is well fitted to the training data. It does not look to be over or underfitted.

The confusion matrix for the classifier is:

	0	1
0	503	18
1	41	438

The accuracy is 0.941.

If this accuracy is good enough depends entirely on the domain. But it is quite good!

Part 2

The confusion matrix for the classifier on the test set is:

	0	1
0	199	9
1	19	145

The accuracy is 0.9247.

This GP model has a small drop of accuracy on the test set but this is expected. If the model was over or underfitted the accuracy on the test set would probably even worse. I believe that for its circumstances this model is performing well and has a good fit.

Part 3

Now a new GP model is fitted where all available features are used. This model is then evaluated on the test set.

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

The confusion matrix for the classifier on the test set with a model trained on all available features is:

	0	1
0	216	0
1	2	154

The accuracy is 0.9946.

The classifier that is trained on all features performs very well, it actually performs better on the test set than the first model does on its own training set. To only misclassify two points in the test set is exceptional. Compared to the first model this performs better, the two extra features should therefor have information that further divide fraud into two groups and should be included in the model.

Code:

```
# Import bank data
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)
training <- data[SelectTraining,]
test <- data[-SelectTraining,]

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

x1 <- seq(min(training[,1]),max(training[,1]),length=100)
x2 <- seq(min(training[,2]),max(training[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main = '
points(training[training[,5]==1,1],training[training[,5]==1 ,2],col="blue")
points(training[training[,5]==0,1],training[training[,5]==0 ,2],col="red")
legend(4.5, -10, legend=c("Fraud = 1", "Fraud = 0"),
       col=c("blue", "red"),lty = 1:2, cex=0.8)
# Confusion matrix

cfm <- table(predict(GPfit,training[,1:2]), training[,5])

accuracy <- sum(diag(cfm))/sum(cfm)

# Part 2
```

```
cfm_test <- table(predict(GPfit,test[,1:2]), test[,5])
accuracy_test <- sum(diag(cfm_test))/sum(cfm_test)

# Part 3
GPfit_complete <- gausspr(fraud ~  varWave + skewWave + kurtWave + entropyWave, data=training)
cfm_complete <- table(predict(GPfit_complete,test[,1:4]), test[,5])
accuracy_complete <- sum(diag(cfm_complete))/sum(cfm_complete)
```