Vikgu708
Viktor Gustafsson
2020-10-16

TDDE15 Lab 4

2.1.1

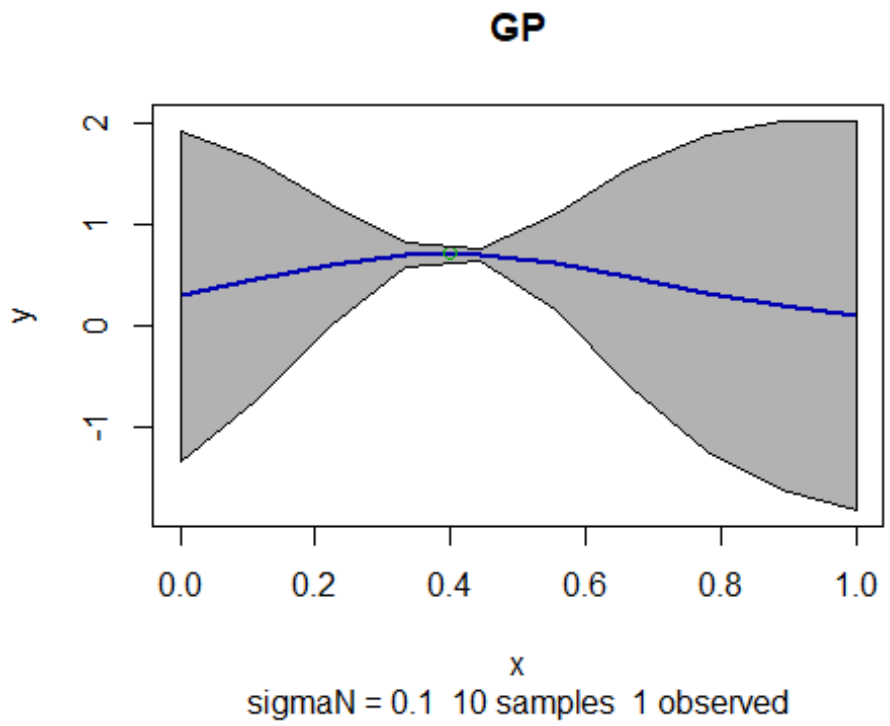
```
kernelMaker = function(sigmaF=1,l=0.3){

expKernel <- function(x1,x2){
  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)
}

return(expKernel)
}

posteriorGP = function (X, y, Xstar, sigmaNoise, k){
  kstar = k(X,Xstar)
  L = chol(k(X,X)+sigmaNoise^2*diag(length(X)))
  L = t(L)
  alpha = solve(t(L), solve(L,y))
  fMean = t(kstar) %*% alpha
  v = solve(L,kstar)
  fVar = k(Xstar, Xstar)-t(v)%*%v
  logMarginalLike = -0.5*t(y)%*%alpha-sum(sum(log(L)))-length(y)/2*log(2*pi
)
  return (list(fmean=fMean, fvar=fVar, logLike=logMarginalLike))
}

plotGP = function(fmean, fvar, Xstar, X, y){
  upperConf = (fmean+diag(fvar)*1.96)[,1]
  lowerConf = (fmean-diag(fvar)*1.96)[,1]
  subheading = sprintf("sigmaN = %g  %i samples  %i observed",
                       sigmaNoise, length(Xstar),length(X))
  plot(x=Xstar, y=fmean, col="blue", type="l", lwd=2,
       ylim=c(min(lowerConf),max(upperConf)),
       sub=subheading, main="GP", ylab="y", xlab="x") +
      points(x=X, y=y, col="green")

  polygon(x = c(rev(Xstar), Xstar),
          y = c(rev(upperConf), lowerConf),
          col = rgb(0, 0, 0, 0.3))


}

par(mfrow=c(1,1))
```

Vikgu708
Viktor Gustafsson
2020-10-16

```
#par(mfrow = c(2,2))
```

2.1.2

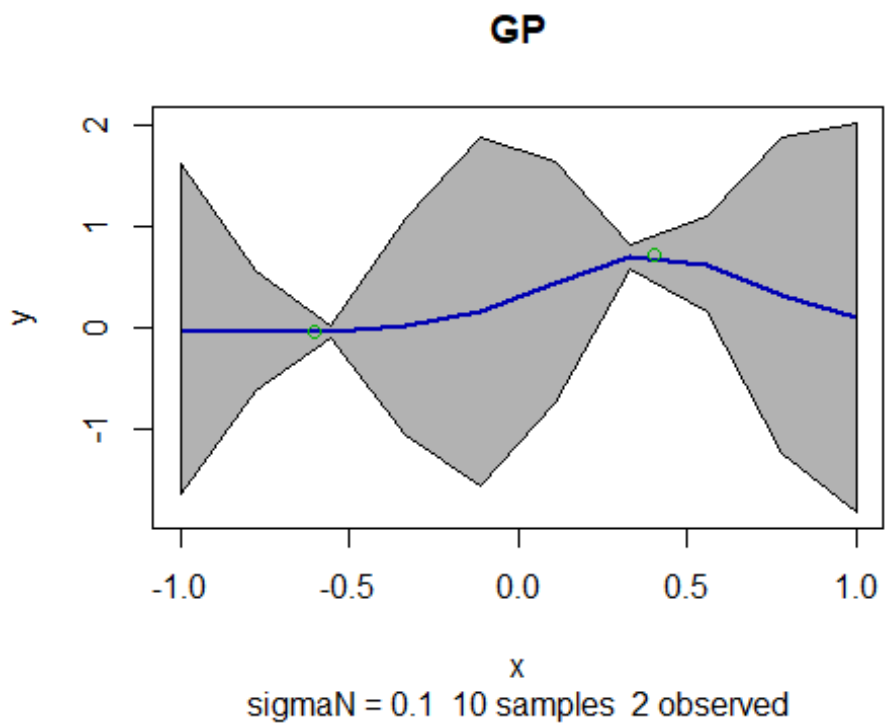
```
X = c(0.4)
y = c(0.719)
sigmaNoise = 0.1
Xstar = seq(0,1,length=10)

gpParam = posteriorGP(X, y, Xstar, sigmaNoise, kernelMaker())
plotGP(gpParam$fmean, gpParam$fvar, Xstar, X, y)
```



2.1.3

```
X = c(0.4, -0.6)
y = c(0.719, -0.044)
sigmaNoise = 0.1
Xstar = seq(-1,1,length=10)

gpParam = posteriorGP(X, y, Xstar, sigmaNoise, kernelMaker())
plotGP(gpParam$fmean, gpParam$fvar, Xstar, X, y)
```

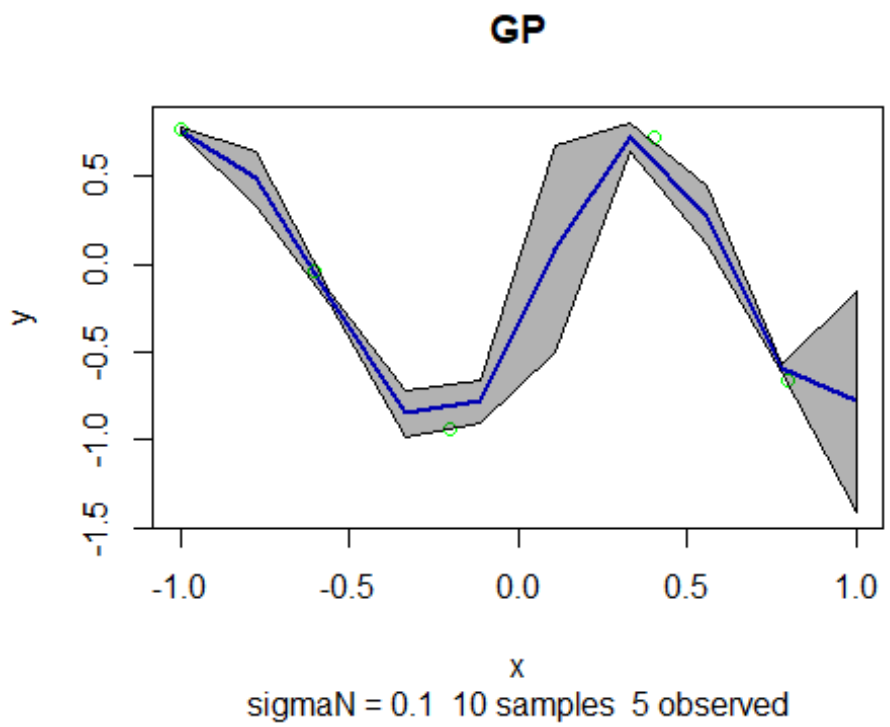
Vikgu708
Viktor Gustafsson
2020-10-16



2.1.4

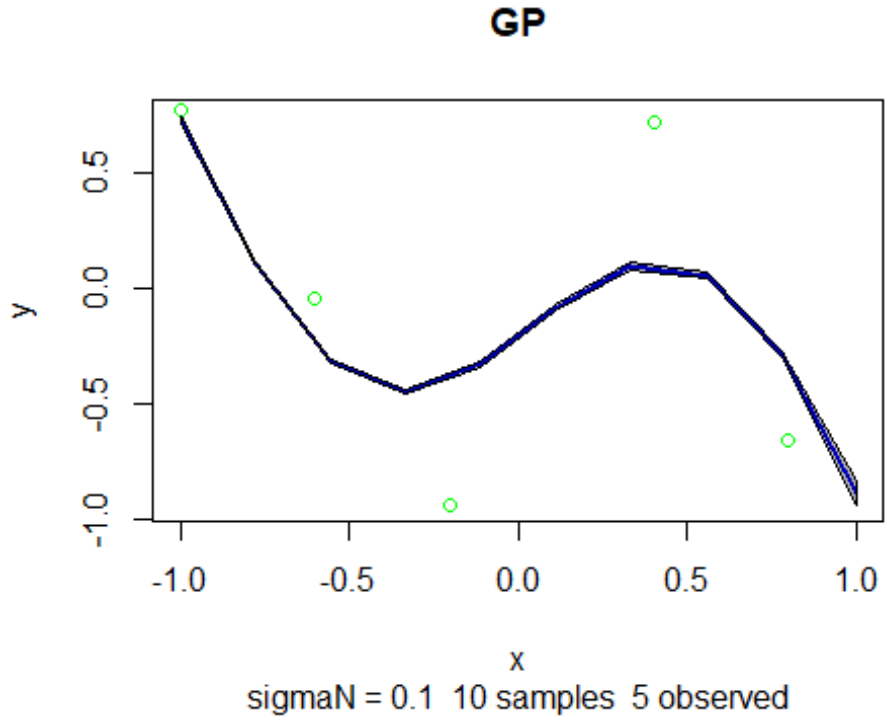
```
X = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
sigmaNoise = 0.1
Xstar = seq(-1,1,length=10)
gpParam = posteriorGP(X, y, Xstar, sigmaNoise, kernelMaker())
plotGP(gpParam$fmean, gpParam$fvar, Xstar, X, y)
```

Vikgu708
Viktor Gustafsson
2020-10-16



2.1.5

```
gpParam = posteriorGP(X, y, Xstar, sigmaNoise, kernelMaker(sigmaF=1,l=1))
plotGP(gpParam$fmean, gpParam$fvar, Xstar, X, y)
```



Vikgu708
Viktor Gustafsson
2020-10-16

A higher L means the covariance will be higher for points further away compared to a smaller value of L. This means that the function that increasing l restricts the functions variance within a given range. This is why the line fits the data poorly, as the data changes faster than the function is able to vary as the function is restricted with high covariance between points, and this means the function cant fit this data that varies in y quickly for relative small changes in x.

2.2 Help functions and preprocessing

```
library(kernlab)

par(mfrow = c(1, 1))

kernelWrapPeriod = function(sigmaF, l1, l2, d) {
  expKernel <- function(x1, x2) {
    n1 <- length(x1)
    n2 <- length(x2)
    K <- matrix(NA, n1, n2)
    for (i in 1:n2) {
      K[, i] <-
        sigmaF ^ 2 * exp(-2 * (sin(pi * abs(x1 - x2[i]) / d) / l1) ^ 2) *
          exp(-0.5 * (abs(x1 - x2[i]) / l2) ^ 2)
    }
    return(K)
  }
  class(expKernel) <- 'kernel'

  return(expKernel)
}

kernelWrap = function(sigmaF, l) {
  expKernel <- function(x1, x2) {
    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)
  }
  class(expKernel) <- 'kernel'

  return(expKernel)
}

plotGP = function(fmean, fvar, Xstar, X, y) {
  subheading = sprintf("sigmaN = %g  %i samples  %i observed",
                       sigmaNoise,
                       length(Xstar),
                       length(X))

  if (!is.null(fvar)) {
    upperConf = (fmean + diag(fvar) * 1.96)[, 1]
```

```
    lowerConf = (fmean - diag(fvar) * 1.96)[, 1]
    plot(
      x = Xstar,
      y = fmean,
      col = "blue",
      type = "l",
      ylim = c(min(lowerConf), max(upperConf)),
      lwd = 2,
      sub = subheading,
      main = "GP",
      ylab = "temp",
      xlab = "time"
    ) +
      lines(x = X, y = y, col = "black")

    polygon(
      x = c(rev(Xstar), Xstar),
      y = c(rev(upperConf), lowerConf),
      col = rgb(0, 0, 0, 0.3)
    )
  } else {
    plot(
      x = Xstar,
      y = fmean,
      col = "blue",
      type = "l",
      lwd = 2,
      sub = subheading,
      main = "GP",
      ylab = "temp",
      xlab = "time"
    ) +
      lines(x = X, y = y, col = "black")
  }

}

getGPvar = function (X, y, Xstar, sigmaNoise, k) {
  y = scale(y)
  X = scale(X)
  Xstar = scale(Xstar)
  kstar = k(X, Xstar)
  L = chol(k(X, X) + sigmaNoise ^ 2 * diag(length(X)))
  L = t(L)
  v = solve(L, kstar)
  fVar = k(Xstar, Xstar) - t(v) %*% v
  return (fVar)
}

# preprocessing

data = read.csv(
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
```

```
Code/TempTullinge.csv",
  header = TRUE,
  sep = ";"
)

dates = as.Date(data$date, "%d/%m/%y")
rel_days = sapply(dates, function(x) {
  as.numeric(x - dates[1]) %% 365 + 1
})
abs_days = sapply(dates, function(x) {
  as.numeric(x - dates[1]) + 1
})
data$day = rel_days
data$time = abs_days

small.data = c()
small.data$temp = data$temp[seq(1, length(data$temp), by = 5)]
small.data$day = rel_days[seq(1, length(rel_days), by = 5)]
small.data$time = abs_days[seq(1, length(abs_days), by = 5)]
```

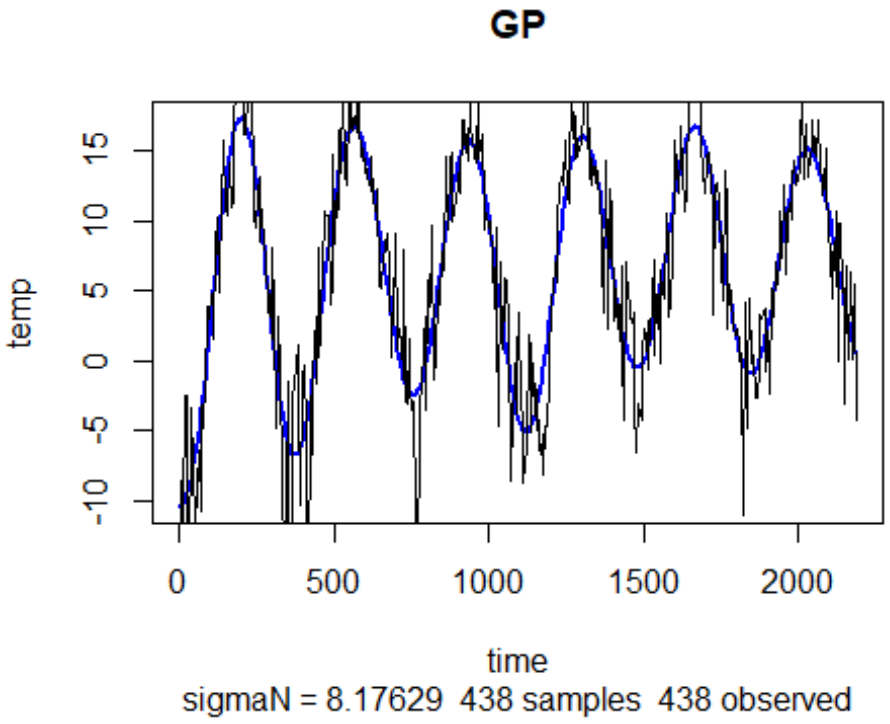
2.2.1

```
X = c(1, 3, 4)
Xstar = c(2, 3, 4)
kernelMatrix(kernel = kernelWrap(sigmaF = 20, l = 0.2),
             x = X,
             y = Xstar)

## An object of class "kernelMatrix"
##              [,1]         [,2]         [,3]
## [1,] 1.490661e-03 7.714999e-20 5.545373e-47
## [2,] 1.490661e-03 4.000000e+02 1.490661e-03
## [3,] 7.714999e-20 1.490661e-03 4.000000e+02
```

2.2.2

```
polyFit <-
  lm(small.data$temp ~  small.data$time + I(small.data$time ^ 2))
sigmaNoise = sd(polyFit$residuals)

GPfit <- gausspr(
  small.data$time,
  small.data$temp,
  kpar = list(sigmaF = 20, l = 0.2),
```

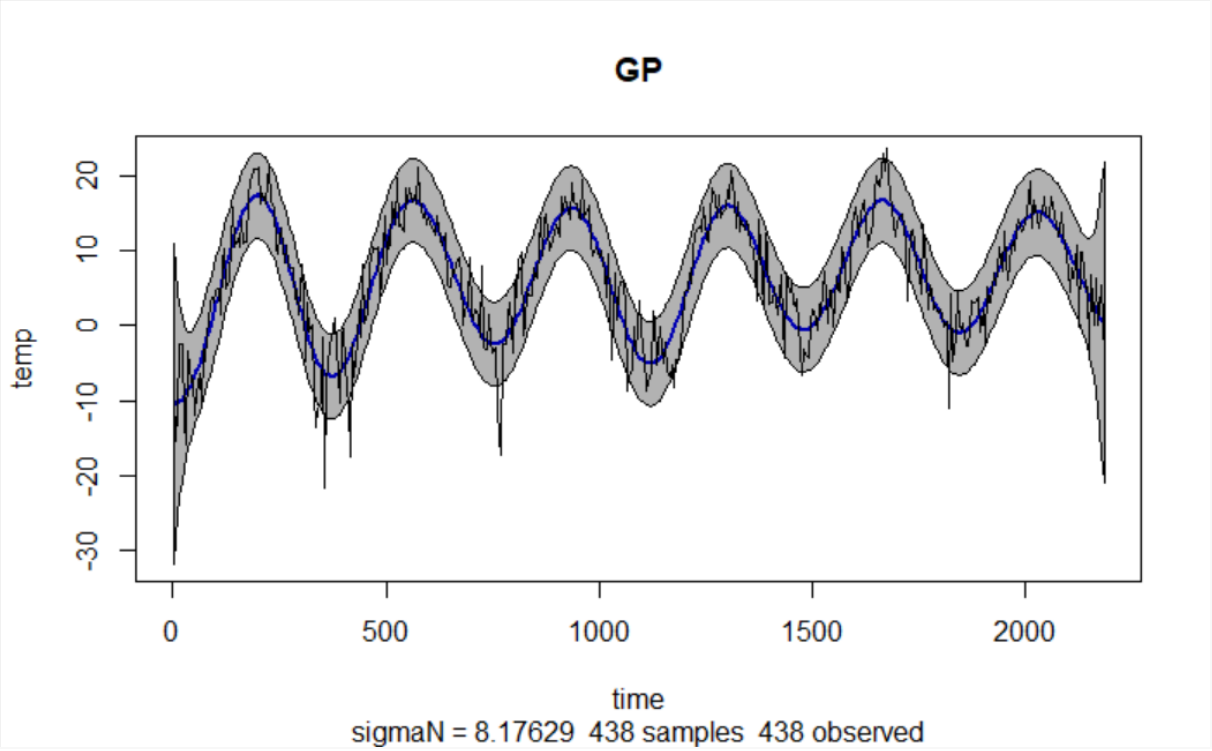
```
  kernel = kernelWrap,
  var = sigmaNoise ^ 2
)

fstar = predict(GPfit, small.data$time)
plotGP(fstar, NULL, small.data$time, small.data$time, small.data$temp)
```



2.2.3

```
fvar = getGPvar(
  small.data$time,
  small.data$temp,
  small.data$time,
  sigmaNoise,
  kernelWrap(sigmaF = 20, l = 0.2)
)
plotGP(fstar, fvar, small.data$time, small.data$time, small.data$temp)
```

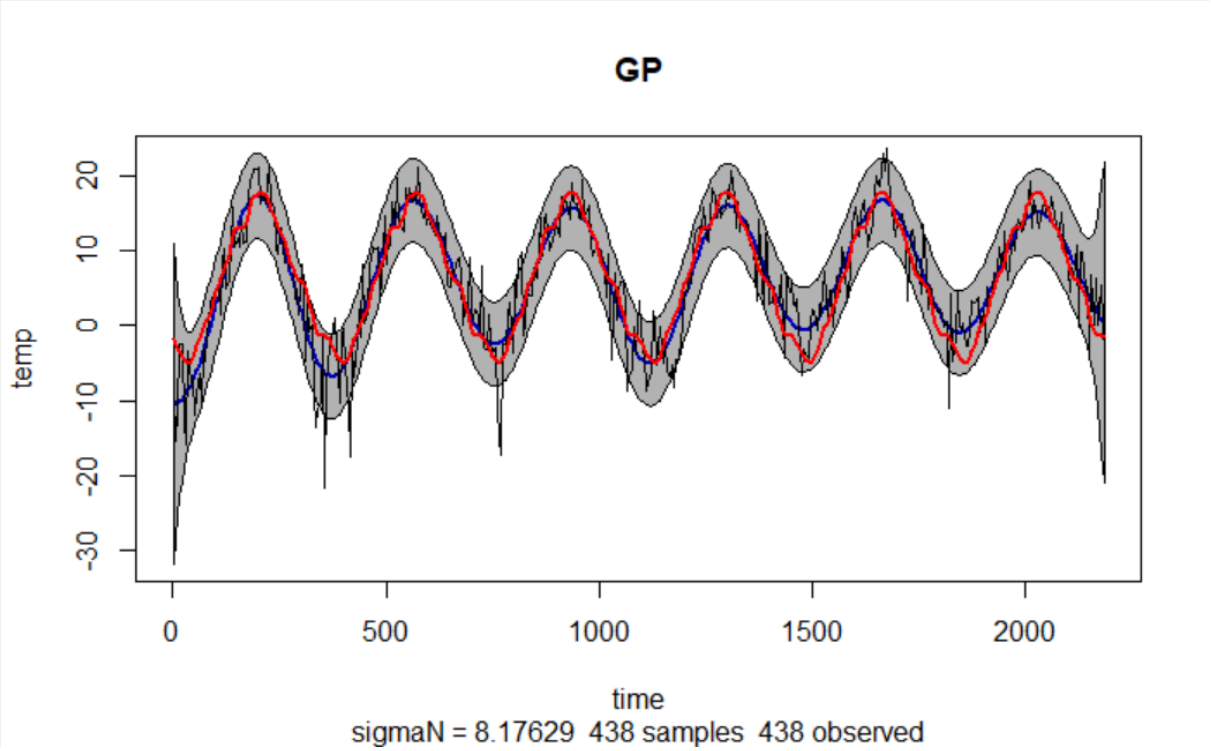
Vikgu708
Viktor Gustafsson
2020-10-16



2.2.4

```
day.polyFit <-
  lm(small.data$temp ~  small.data$day + I(small.data$day ^ 2))
day.sigmaNoise = sd(day.polyFit$residuals)

day.GPfit <- gausspr(
  small.data$day,
  small.data$temp,
  kpar = list(sigmaF = 20, l = 0.2),
  kernel = kernelWrap,
  var = day.sigmaNoise ^ 2
)

day.fstar = predict(day.GPfit, small.data$day)
lines(x = small.data$time,
      day.fstar,
      col = "red",
      lwd = 2)
```

Vikgu708
Viktor Gustafsson
2020-10-16

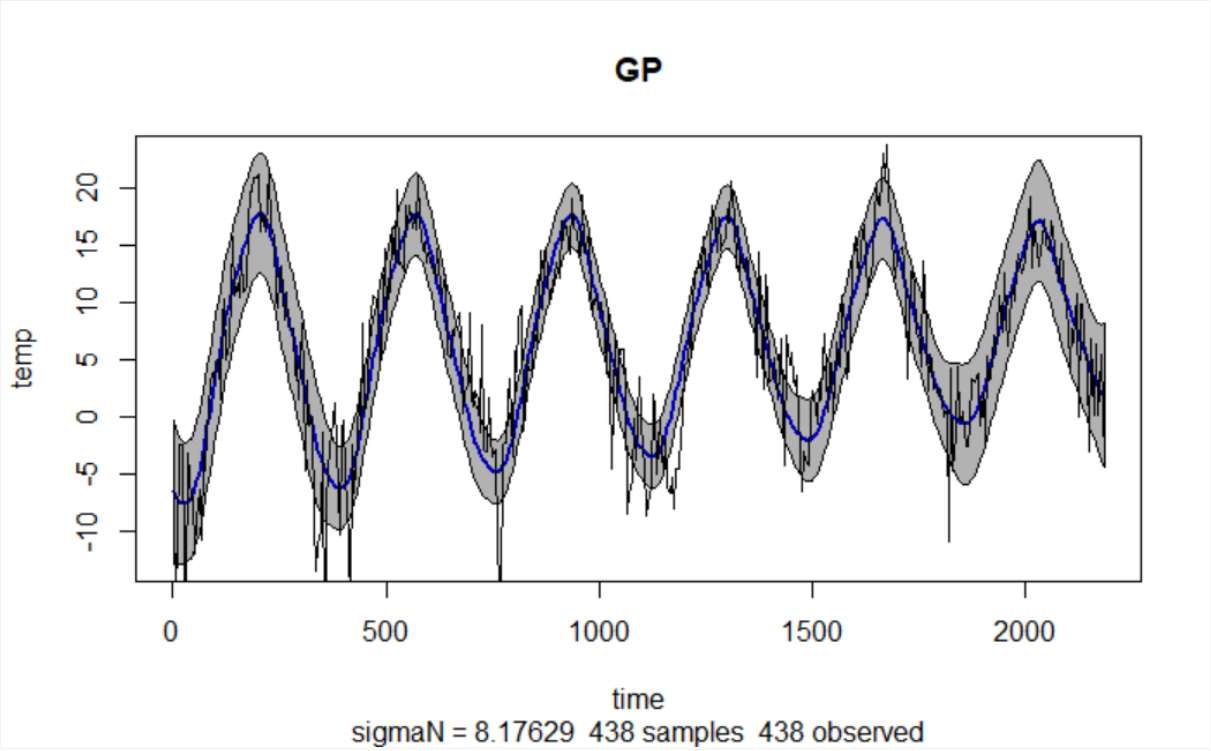


When using relative days durings the year, the model cannot adjust for long term trends over several years. This because the truncation of the datapoints over 365 days makes the model only able to learn the same model for all years. This is why this model predicts the same temperature for each year. It is therefor not a good model for temperature prediction, as there are trends over years for temperature.

2.2.5

```
period.GPfit <- gausspr(
  small.data$time,
  small.data$temp,
  kpar = list(
    sigmaF = 20,
    l1 = 1,
    l2 = 10,
    d = 365 / sd(small.data$time)
  ),
  kernel = kernelWrapPeriod,
  var = sigmaNoise ^ 2
)

period.fstar = predict(period.GPfit, small.data$time)
period.fvar = getGPvar(
  small.data$time,
  small.data$temp,
  small.data$time,
  sigmaNoise,
  kernelWrapPeriod(
```

```
    sigmaF = 20,
    l1 = 1,
    l2 = 10,
    d = 365 / sd(small.data$time)
  )
)

plotGP(period.fstar,
       period.fvar,
       small.data$time,
       small.data$time,
       small.data$temp)
```



This model seems to correctly find the correlation both over absolute time and correlation between each relative day during the year. This is probably because the periodic kernel function uses both relative and absolute time for the correlation, as it is a combination of the purely periodic kernel (which gives similar result to using relative days as inputs) and the squared exponential kernel (that gives the same result as using time as input).

2.3 Helper functions and preprocessing

```
library(AtmRay)
```

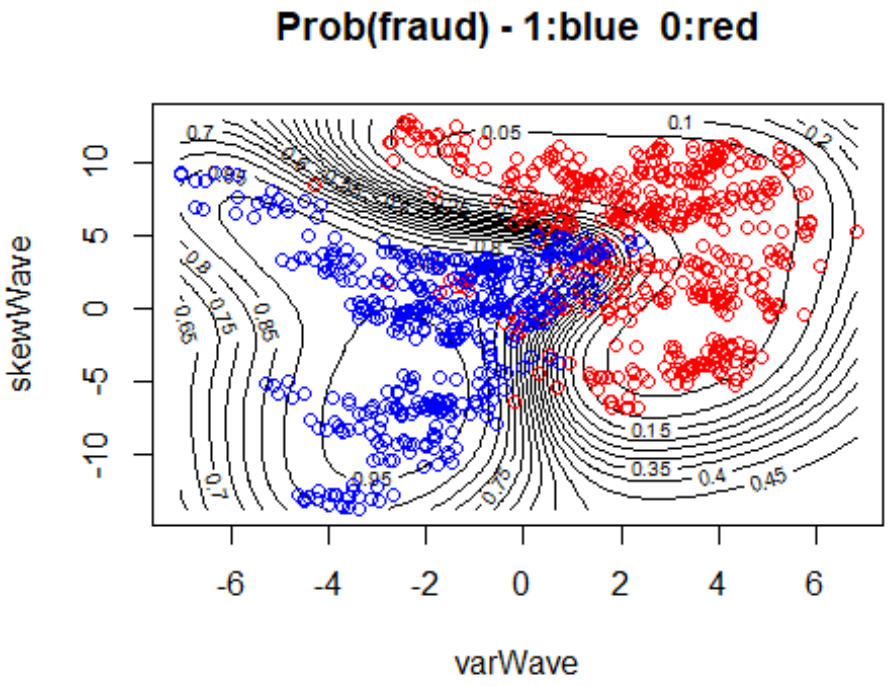
Vikgu708
Viktor Gustafsson
2020-10-16

```r
library(kernlab)

# plots cov1, cov2 against target with contours given a trained GP modell G
Pfit
makeGridPlot = function(GPfit,
                        targetName,
                        cov1Name,
                        cov2Name,
                        data) {
  cov1Col = which(names(data) == cov1Name)
  cov2Col = which(names(data) == cov2Name)
  targetCol = which(names(data) == targetName)

  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) <- c(names(data)[1], names(data)[2])

  probPreds <- predict(GPfit, gridPoints, type = "probabilities")
  contour(
    x1,
    x2,
    matrix(probPreds[, 2], 100, byrow = TRUE),
    20,
    xlab = cov1Name,
    ylab = cov2Name,
    main = sprintf('Prob(%s) - 1:blue  0:red', targetName)
  )

  points(data[data[, targetCol] == "0", cov1Col], data[data[, targetCol] ==
"0", cov2Col], col =
           "red")
  points(data[data[, targetCol] == "1", cov1Col], data[data[, targetCol] ==
"1", cov2Col], col =
           "blue")
}

#Prints accuracy and confusion tables given predictions and targets
printAccAndConf = function(label, preds, real) {
  conf = table(preds, real)
  acc = sum(diag(conf)) / sum(conf)
  print(sprintf("%s Confusion table", label))
  print(conf)
  print(sprintf("%s Accuracy: %g", label, acc))
}
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])

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

2.3.1

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

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

makeGridPlot(GPfit, "fraud", "varWave", "skewWave", train)
```




```
# conf matrix
train_predict = predict(GPfit, train)
printAccAndConf("Train", train_predict, train$fraud)

## [1] "Train Confusion table"
##      real
## preds   0   1
##     0 503  18
##     1  41 438
## [1] "Train Accuracy: 0.941"
```

Vikgu708
Viktor Gustafsson
2020-10-16

2.3.2

```
test_predict = predict(GPfit, test)
printAccAndConf("Test", test_predict, test$fraud)
```

```
## [1] "Test Confusion table"
##      real
## preds   0   1
##     0 199   9
##     1  19 145
## [1] "Test Accuracy: 0.924731"
```

2.3.3

```
GPfitAll <- gausspr(fraud ~  ., data = train)
```

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

```
all_test_predict = predict(GPfitAll, test)
printAccAndConf("Test", all_test_predict, test$fraud)
```

```
## [1] "Test Confusion table"
##      real
## preds   0   1
##     0 216   0
##     1   2 154
## [1] "Test Accuracy: 0.994624"
```

There is a clear performance improvement with using all four variables as the test accuracy increases to near perfect predictions, with only 2 false negatives and zero false positives for the entire test set. This is an increase of about 7 percent units from the prediction with only two covariates. This is quite a large increase in accuracy, considering how accurate the model is on the test set with two covariates.