AML_Lab4_Hyungyum_Kim

Josh Hyungyum Kim

2018 - 10 - 15

1. Implementing GP Regression

1-1. Write your own code

For the first exercise, the following Gaussian process regression model will be implemented:

$$y = f(x) + \epsilon, \quad \epsilon \sim N(0, \sigma_n^2), \quad f \sim GP(0, k(x, x'))$$

The function is called posteriorGP and has 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. $X_\star$.
- hyperParam: Vector with two elements, σ_f and l.
- sigmaNoise: Noise standard deviation σ_n.

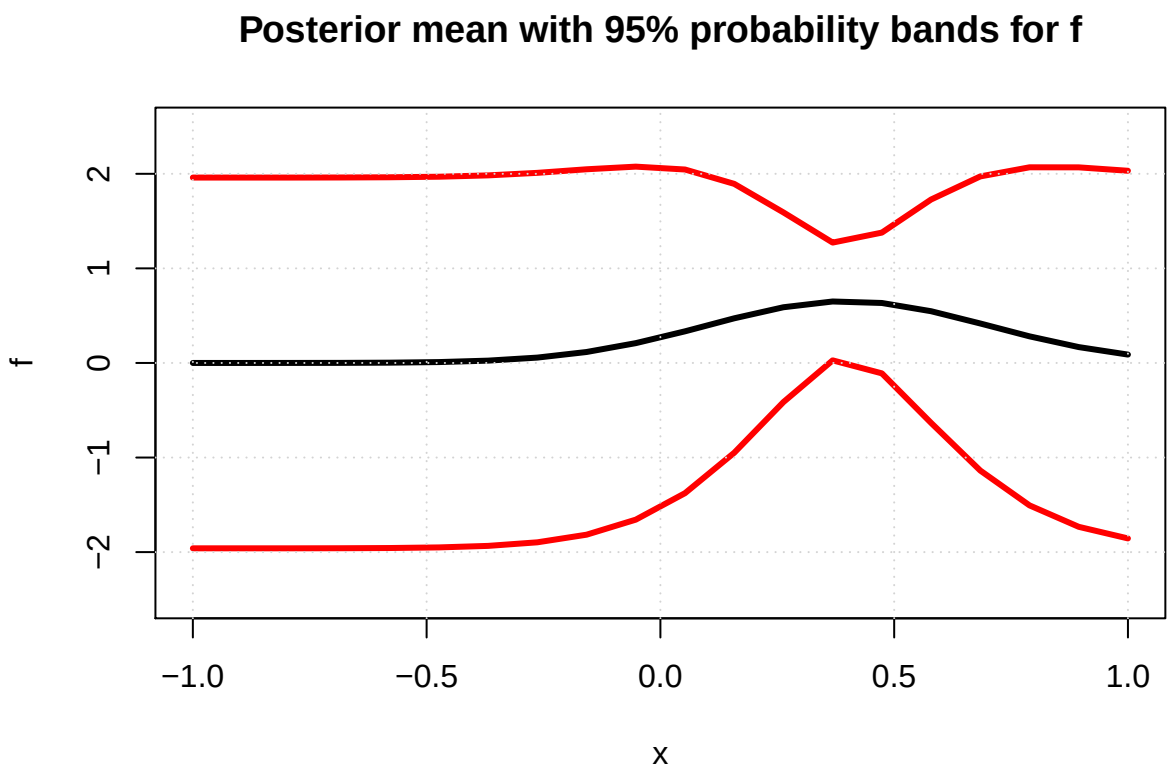
```
# load required pacakages
require(kernlab)
require(AtmRay)

# 1.

# 1-1.

# Setting up the separated kernel function
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
posteriorGP <- function(X, y, XStar, hyperParam, sigmaNoise){

  covMat <- SquaredExpKernel(X, X, sigmaF=hyperParam[1], l=hyperParam[2])
  L <- t(chol(covMat + diag(sigmaNoise, nrow=length(X)))) # Lower triangle
  alpha <- solve(t(L), solve(L,y))

  # Gaussian process posterior mean
  k_star <- SquaredExpKernel(X, XStar, sigmaF=hyperParam[1], l=hyperParam[2])
  post_mean <- t(k_star) %*% alpha

  # Gaussian process posterior cov_mat
  v <- solve(L, k_star)
```

```r
  post_var <- SquaredExpKernel(XStar, XStar, sigmaF=hyperParam[1], l=hyperParam[2]) - t(v)%*%v

  # Returning mean and variance
  res <- list(Post_mean=as.vector(post_mean), Post_var=as.vector(diag(post_var)))
  return(res)
}
```

1-2. Update prior with a single observation

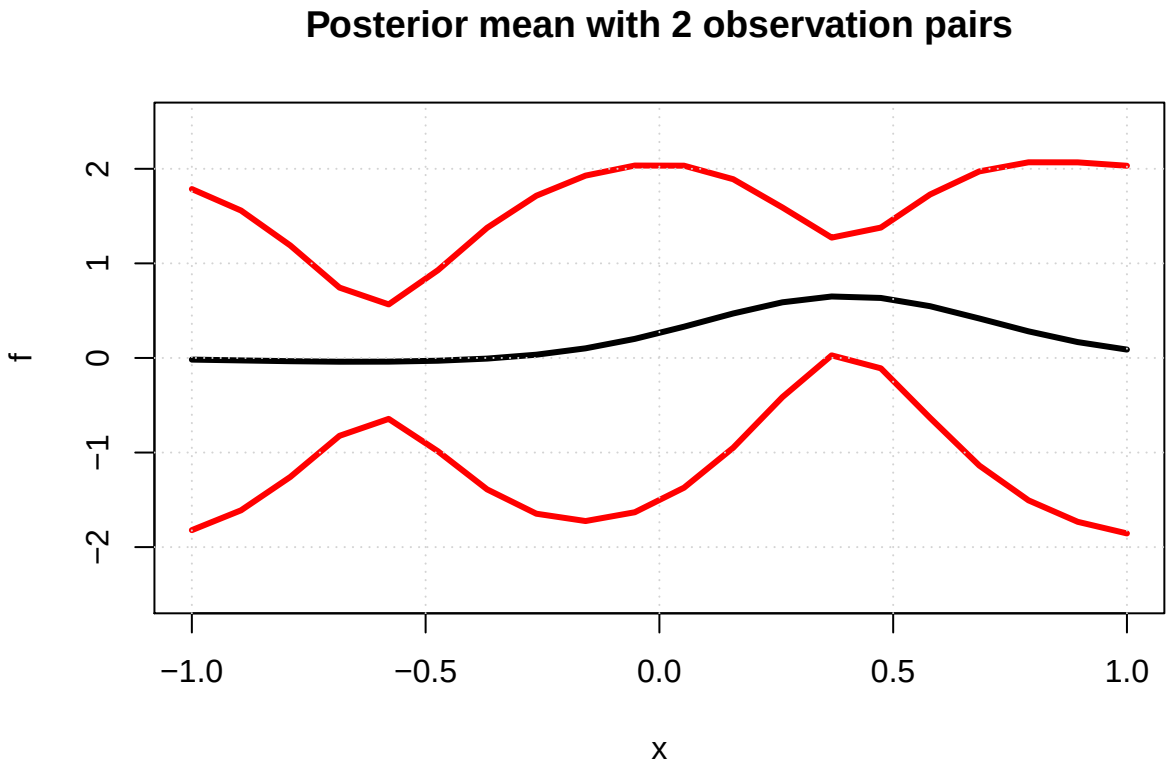
Here, one observation pair $(x, y) = (0.4, 0.719)$ and hyperparameters $\sigma_f = 1$ and $l = 0.3$ are used for 20 points in interval $x \in [-1, 1]$.

```r
# 1-2.

# Run the function with given conditions
xGrid <- seq(-1,1,length=20)
gpsim1 <- posteriorGP(X=0.4, y=0.719, XStar = xGrid, hyperParam = c(1,0.3), sigmaNoise = 0.1)

# Posterior mean plot with 95% probability bands for f.
plot(xGrid, gpsim1[[1]], type="l", ylim=c(-2.5,2.5), lwd=3, xlab="x", ylab="f",
     main="Posterior mean with 95% probability bands for f")
lines(xGrid, gpsim1[[1]]+1.96*sqrt(gpsim1[[2]]), col = "red", lwd = 3)
lines(xGrid, gpsim1[[1]]-1.96*sqrt(gpsim1[[2]]), col = "red", lwd = 3)
grid()
```



1-3. Update prior with two observation pairs

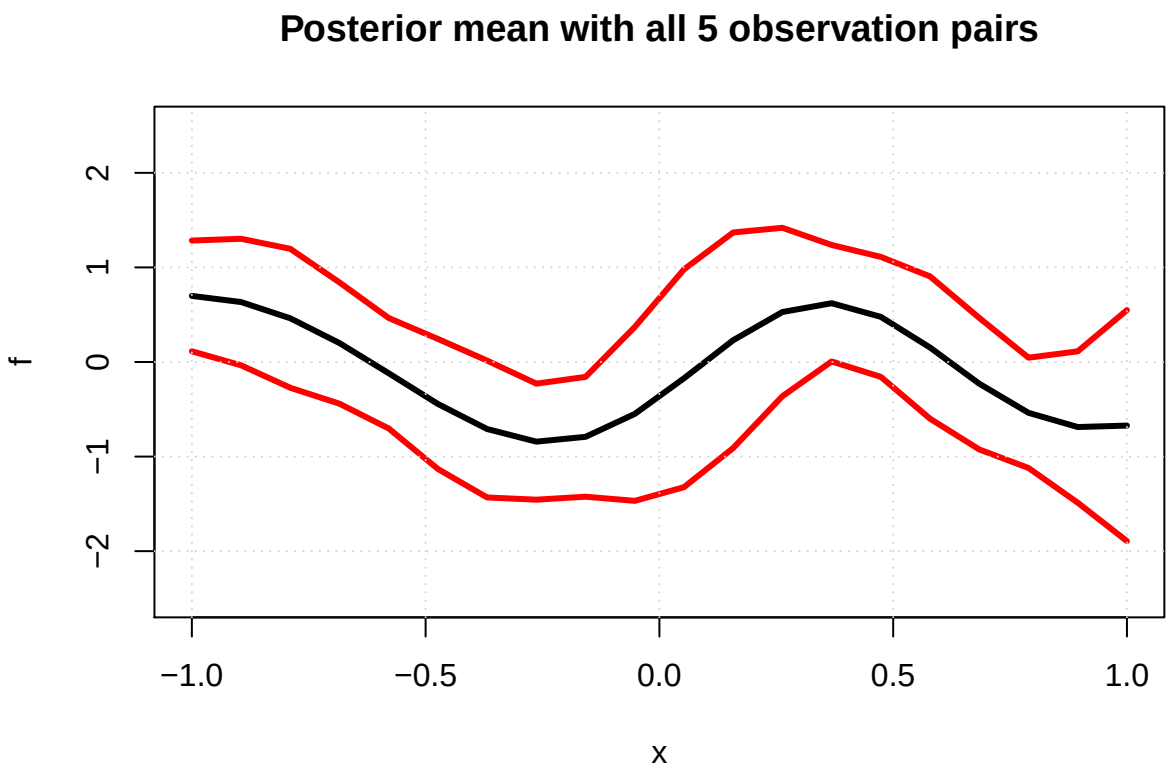
```
# 1-3.

# Run the function with given conditions
xGrid <- seq(-1,1,length=20)
X <- c(0.4, -0.6)
y <- c(0.719, -0.044)
gpsim2 <- posteriorGP(X=X, y=y, XStar = xGrid, hyperParam = c(1,0.3), sigmaNoise = 0.1)

# Posterior mean plot with 95% probability bands for f.
plot(xGrid, gpsim2[[1]], type="l", ylim=c(-2.5,2.5), lwd=3, xlab="x", ylab="f",
     main="Posterior mean with 2 observation pairs")
lines(xGrid, gpsim2[[1]]+1.96*sqrt(gpsim2[[2]]), col = "red", lwd = 3)
lines(xGrid, gpsim2[[1]]-1.96*sqrt(gpsim2[[2]]), col = "red", lwd = 3)
grid()
```



1-4. Update prior with all five observation pairs

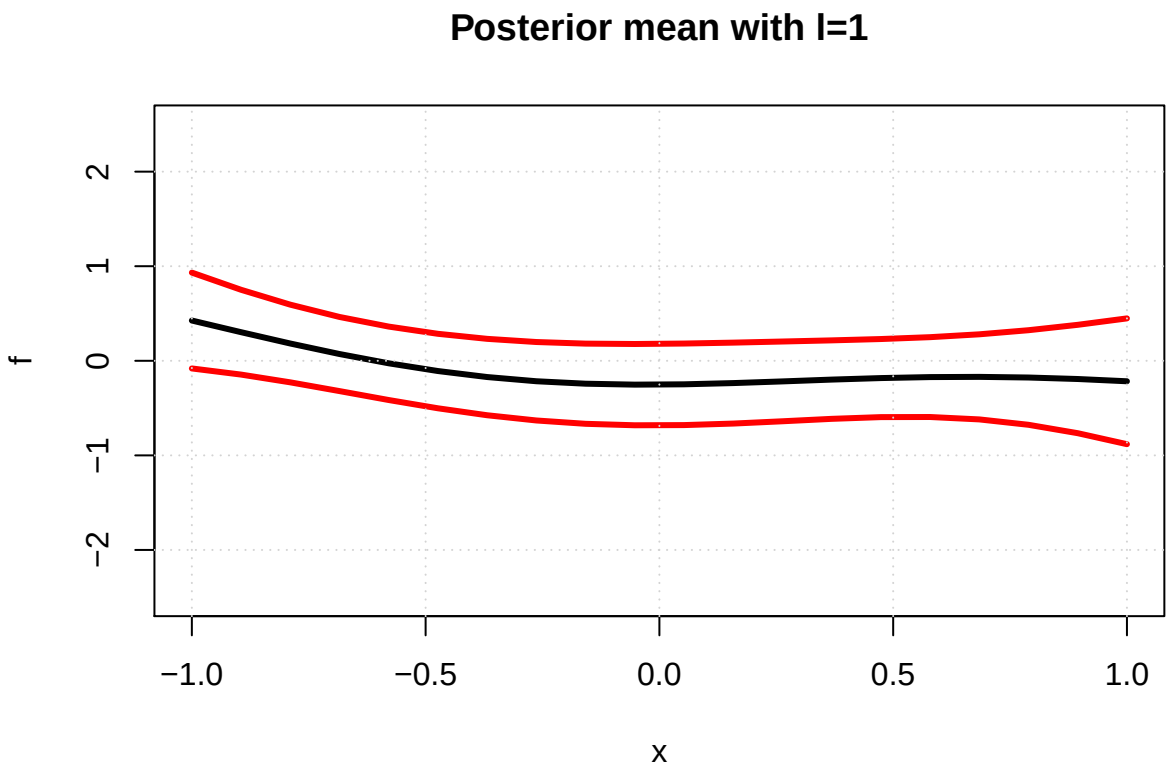
```
# 1-4.

# Run the function with given conditions
xGrid <- seq(-1,1,length=20)
X <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
gpsim3 <- posteriorGP(X=X, y=y, XStar = xGrid, hyperParam = c(1,0.3), sigmaNoise = 0.1)
```

```r
# Posterior mean plot with 95% probability bands for f.
plot(xGrid, gpsim3[[1]], type="l", ylim=c(-2.5,2.5), lwd=3, xlab="x", ylab="f",
     main="Posterior mean with all 5 observation pairs")
lines(xGrid, gpsim3[[1]]+1.96*sqrt(gpsim3[[2]]), col = "red", lwd = 3)
lines(xGrid, gpsim3[[1]]-1.96*sqrt(gpsim3[[2]]), col = "red", lwd = 3)
grid()
```



1-5. Repeat 1-4 with hyperparameters $\sigma_f = 1$ and $l = 1$

```r
# 1-5.

# Run the function with given conditions
xGrid <- seq(-1,1,length=20)
X <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
gpsim3 <- posteriorGP(X=X, y=y, XStar = xGrid, hyperParam = c(1,1), sigmaNoise = 0.1)

# Posterior mean plot with 95% probability bands for f.
plot(xGrid, gpsim3[[1]], type="l", ylim=c(-2.5,2.5), lwd=3, xlab="x", ylab="f",
     main="Posterior mean with l=1")
lines(xGrid, gpsim3[[1]]+1.96*sqrt(gpsim3[[2]]), col = "red", lwd = 3)
lines(xGrid, gpsim3[[1]]-1.96*sqrt(gpsim3[[2]]), col = "red", lwd = 3)
grid()
```



Above plot shows the result of Gaussian process when the value of parameter l(ell) = 1. Compared to the previous step 1-4. it is obvious that the posterior mean plot is much more smoothed than before $l = 0.3$. This result makes sense because relatively high value of l allows the kernel to yield high correlation of inputs.

2. GP Regression with kernlab

For this exercise, data of daily mean temperature in Stockholm (Tullinge) is used.

```
# 2.

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

# create new variables
temp$time <- seq(1, 2190, 1)
temp$day <- rep(seq(1, 365, 1), 6)

# Remove observations except every fifth
temp <- temp[temp$day%%5==1,]
row.names(temp) <- NULL

head(temp)
```

```
##       date  temp time day
## 1 01/01/10  -8.6    1   1
## 2 06/01/10 -15.4    6   6
## 3 11/01/10 -11.4   11  11
## 4 16/01/10  -2.5   16  16
## 5 21/01/10  -2.5   21  21
## 6 26/01/10 -12.9   26  26
```

2-1.

Custom Squared Exponential Kernel function was defined above for Exercise1.

```
# 2-1.

# ?gausspr
# ?kernelMatrix

# Evaluate point x=1, x'=2
list("Squared_Exponential_Kernel(1,2)"=SquaredExpKernel(1, 2, 1, 0.3))
```

```
## $`Squared_Exponential_Kernel(1,2)`
##            [,1]
## [1,] 0.00386592
```

```
# Cov_mat of X=(1,3,4) and X_star=(2,3,4)
list("KernelMatrix(X,X_star)"=kernelMatrix(SquaredExpKernel, x=c(1,3,4), y=c(2,3,4)))
```

```
## $`KernelMatrix(X,X_star)`
## An object of class "kernelMatrix"
##              [,1]         [,2]        [,3]
## [1,] 3.865920e-03 2.233631e-10 1.92875e-22
## [2,] 3.865920e-03 1.000000e+00 3.86592e-03
## [3,] 2.233631e-10 3.865920e-03 1.00000e+00
```

2-2.

$$temp = f(time) + \epsilon, \quad \epsilon \sim N(0, \sigma_n^2), \quad f \sim GP(0, k(time, time'))$$

```
# 2-2.

SEK <- function(sigmaF = 20, l = 0.2){

  SquaredExpKernel <- 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(SquaredExpKernel) <- "kernel"
  return(SquaredExpKernel)
}

# Simple quadratic model with lm
quad_reg <- lm(temp ~ I(time)+I(time^2), data = temp)
summary(quad_reg)
```

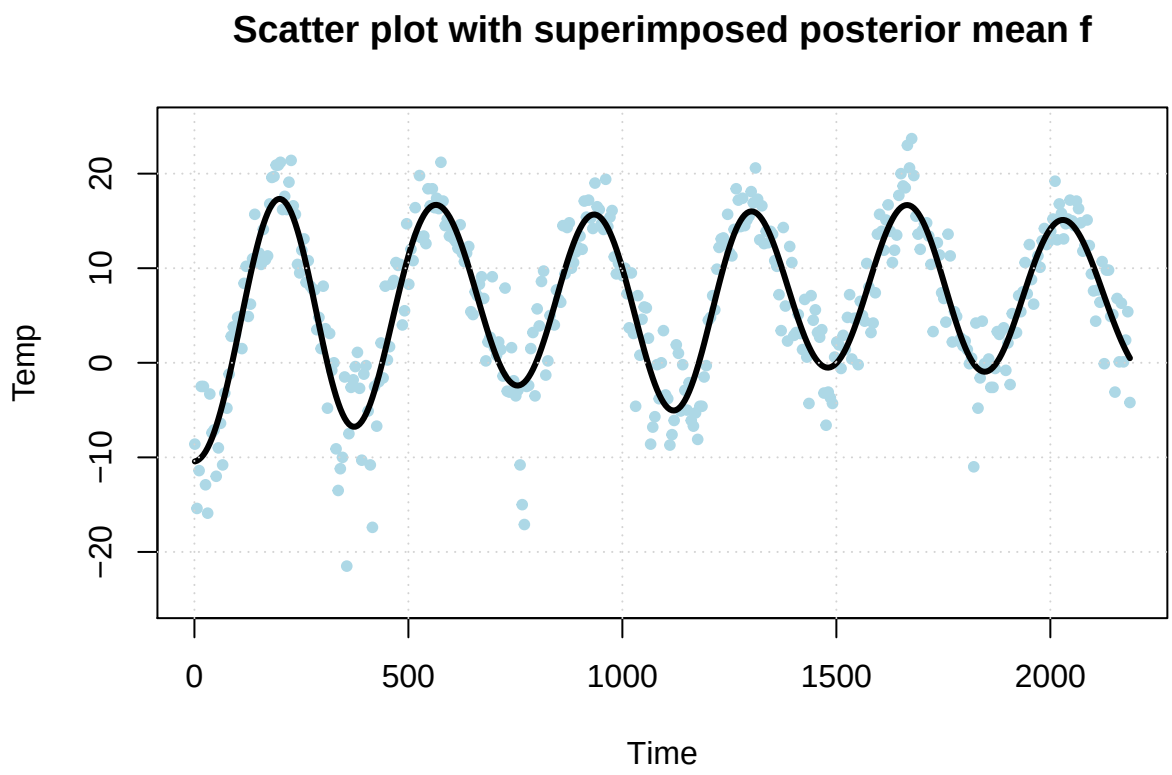
```
##
## Call:
## lm(formula = temp ~ I(time) + I(time^2), data = temp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.1009  -5.6479   0.0386   6.4933  17.3806
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.892e+00  1.172e+00   2.468   0.0140 *
## I(time)      5.319e-03  2.474e-03   2.149   0.0321 *
## I(time^2)   -1.453e-06  1.095e-06  -1.327   0.1853
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.195 on 435 degrees of freedom
## Multiple R-squared:  0.03054,    Adjusted R-squared:  0.02608
## F-statistic: 6.851 on 2 and 435 DF,  p-value: 0.001177
```

```
# sigma_n
sigma_n=sd(quad_reg$residuals)
list("Sigma_n"=sigma_n)
```

```
## $Sigma_n
## [1] 8.176288
```

```
# variable  setting
Temp  <- temp$temp
Time <- temp$time
```

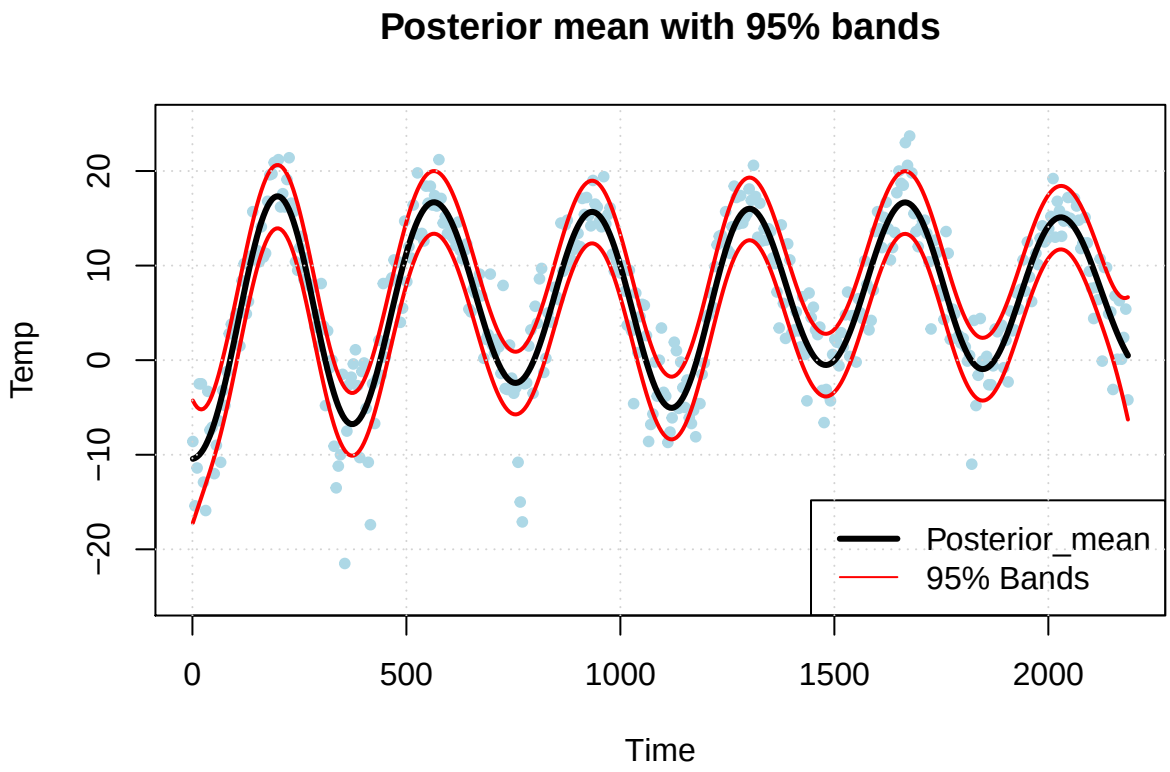
```r
# Gaussian  process fit and plot
gpfit1 <- gausspr(Time, Temp, kernel=SEK, kpar=list(sigmaF=20, l=0.2), var=sigma_n^2)
meanPred <- predict(gpfit1, Time)

plot(Time, Temp, pch=20, col="lightblue", ylim=c(-25,25),
     main="Scatter plot with superimposed posterior mean f")
lines(Time, meanPred, lwd = 3, type = "l")
grid()
```



2-3.

```r
# 2-3.

# Using posteriorGP function to have 95% bands
post_run <- posteriorGP(X=scale(Time), y=Temp, XStar=scale(Time),
                        hyperParam=c(20,0.2), sigmaNoise=sigma_n^2)

# superimposed plot
plot(Time, Temp, pch=20, col="lightblue",
     ylim=c(-25,25), main="Posterior mean with 95% bands")
lines(Time, meanPred, lwd = 3, type = "l", ylab="Temp")
lines(Time, post_run[[1]]+1.96*sqrt(post_run[[2]]), col="red", lwd=2)
lines(Time, post_run[[1]]-1.96*sqrt(post_run[[2]]), col="red", lwd=2)
legend("bottomright", legend=c("Posterior_mean", "95% Bands"),
       col=c("black","red"), lwd=c(3,1))
```

```
grid()
```



2-4.

$$temp = f(day) + \epsilon, \quad \epsilon \sim N(0, \sigma_n^2), \quad f \sim GP(0, k(day, day'))$$

```
# 2-4.

# Simple quadratic model with lm
quad_day <- lm(temp ~ I(day)+I(day^2), data = temp)
summary(quad_day)
```

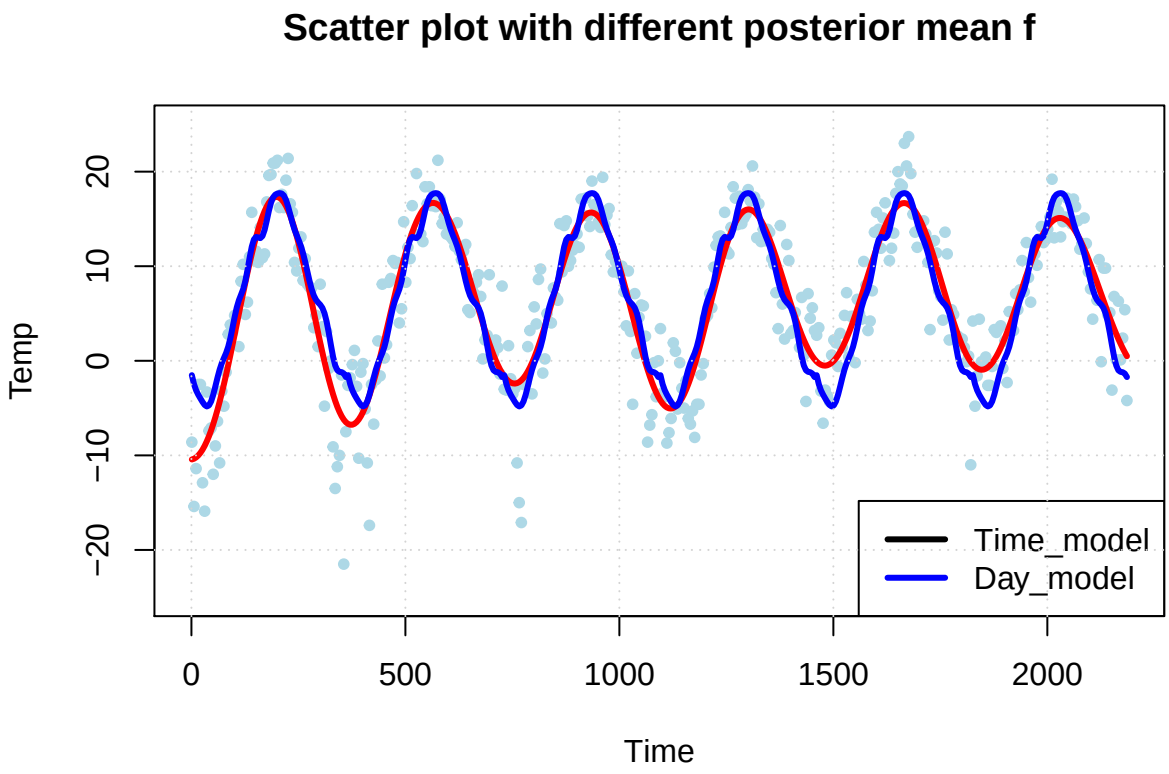
```
##
## Call:
## lm(formula = temp ~ I(day) + I(day^2), data = temp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.2715  -2.6197   0.3812   2.9483  15.3191
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.137e+01  6.860e-01  -16.58   <2e-16 ***
## I(day)       2.525e-01  8.758e-03   28.84   <2e-16 ***
## I(day^2)    -6.373e-04  2.341e-05  -27.22   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.863 on 435 degrees of freedom
## Multiple R-squared:  0.6586, Adjusted R-squared:  0.657
## F-statistic: 419.5 on 2 and 435 DF,  p-value: < 2.2e-16
```

```
# sigma_day
sigma_day=sd(quad_day$residuals)
list("Sigma_day"=sigma_day)
```

```
## $Sigma_day
## [1] 4.852199
```

```
# Variable setting
Day <- temp$day

# Gaussian  process fit and plot
gpfit2 <- gausspr(Day, Temp, kernel=SEK, kpar=list(sigmaF=20, l=0.2), var=sigma_day^2)
meanPred2 <- predict(gpfit2, Day)

plot(Time, Temp, pch=20, col="lightblue", ylim=c(-25,25),
     main="Scatter plot with different posterior mean f")
lines(Time, meanPred, lwd = 3, type = "l", col="red")
lines(Time, meanPred2, lwd = 3, type = "l", col="blue")
legend("bottomright", legend=c("Time_model", "Day_model"),
       col=c("black","blue"), lwd=c(3,3))
grid()
```

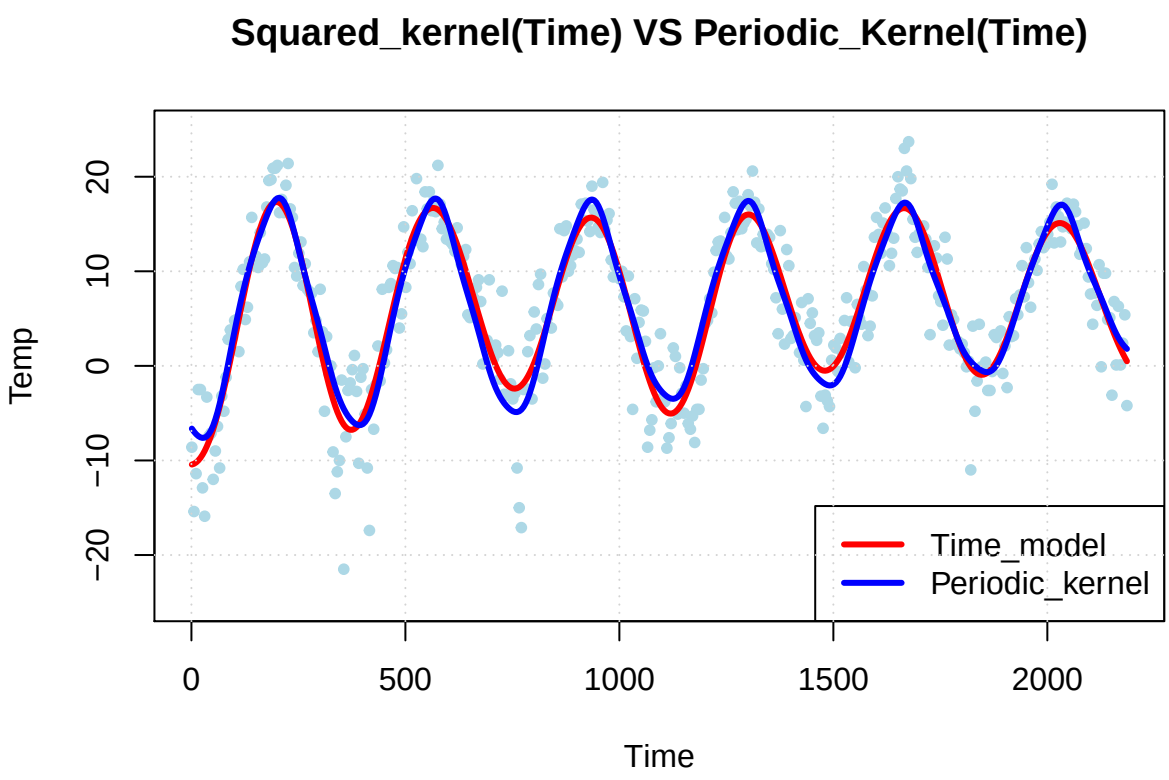


Model with prediction variable *Time* shows more smooth posterior mean curve than model with prediction variable *Day*. Furthermore, the former shows the overall trend of temperature overtime for given period while the latter only shows the seasonal trend of 365 days since variables 1,5,...,365 are duplicated 6 times. Briefly, it is better to use *Time* model to see changes overtime and better to use *Day* model to see general seasonal trend within a year.

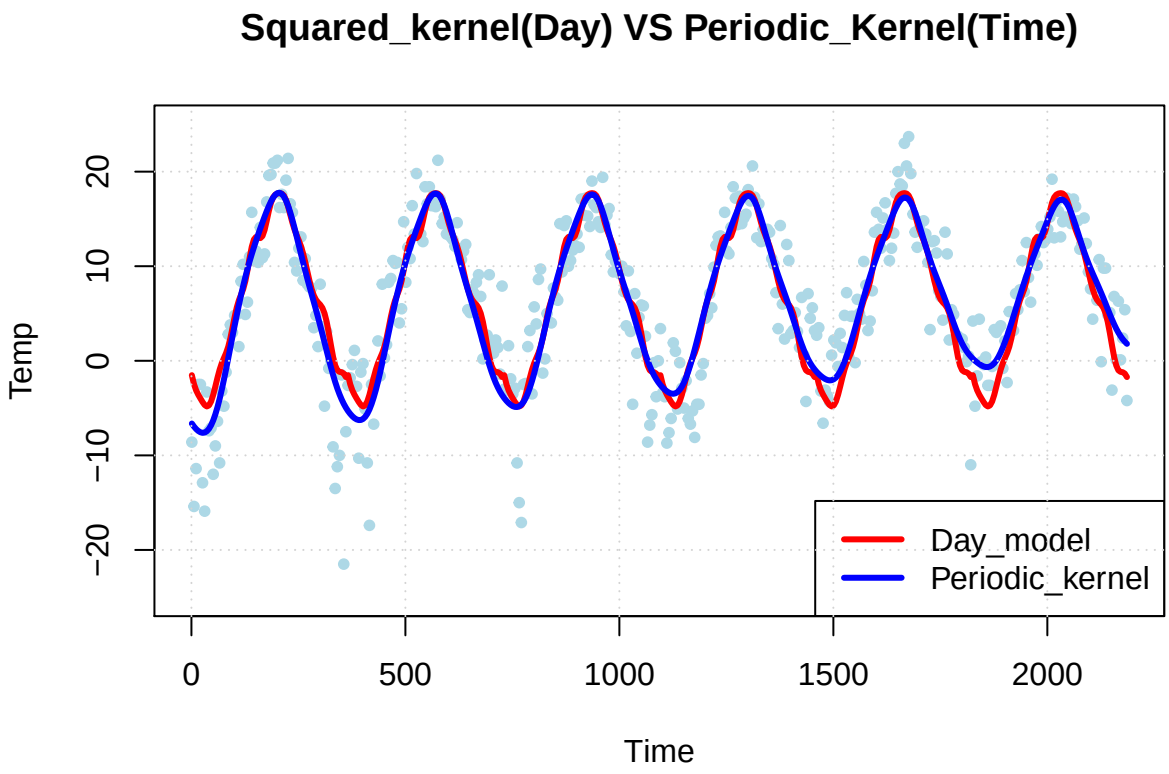
2-5.

```
# 2-5.

# Variable setting
d <- 365/sd(Time)

# Periodic kernel
periodic_kernel <- function(sigmaF=20, l1=1, l2=10, d){
  p_kern <- function(x1, x2){
    r <- sqrt(crossprod(x1-x2))
    return(sigmaF^2*exp(-2*sin((pi*r)/d)^2/(l1^2))*exp(-0.5*(r/l2)^2))
  }
  class(p_kern) <- "kernel"
  return(p_kern)
}

# Gaussian  process fit and plot
gpfit3 <- gausspr(Time, Temp, kernel=periodic_kernel, kpar=list(sigmaF=20, l1=1, l2=10, d=d), var=sigma
meanPred3 <- predict(gpfit3, Time)
```

```
plot(Time, Temp, pch=20, col="lightblue", ylim=c(-25,25),
     main="Squared_kernel(Time) VS Periodic_Kernel(Time)")
lines(Time, meanPred, lwd = 3, type = "l", col="red")
lines(Time, meanPred3, lwd = 3, type = "l", col="blue")
legend("bottomright", legend=c("Time_model", "Periodic_kernel"),
       col=c("red","blue"), lwd=c(3,3))
grid()
```




```
plot(Time, Temp, pch=20, col="lightblue", ylim=c(-25,25),
     main="Squared_kernel(Day) VS Periodic_Kernel(Time)")
lines(Time, meanPred2, lwd = 3, type = "l", col="red")
lines(Time, meanPred3, lwd = 3, type = "l", col="blue")
legend("bottomright", legend=c("Day_model", "Periodic_kernel"),
       col=c("red","blue"), lwd=c(3,3))
grid()
```



Based on the above two plots it seems the model with periodic kernel captures more within year seasonality than the model with squared kernel with variable $Time$ and also shows more overall trend of temperature than the model with squared kernel with variable Day. Since periodic kernel has one more length scale which controls the correlation between the same day in different years, this result looks reasonable.

3. GP Classification with kernlab

```
# 3.

# read data from github
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])
head(data)
```

```
##   varWave skewWave kurtWave entropyWave fraud
## 1 3.62160   8.6661  -2.8073    -0.44699     0
## 2 4.54590   8.1674  -2.4586    -1.46210     0
## 3 3.86600  -2.6383   1.9242     0.10645     0
## 4 3.45660   9.5228  -4.0112    -3.59440     0
## 5 0.32924  -4.4552   4.5718    -0.98880     0
## 6 4.36840   9.6718  -3.9606    -3.16250     0
```

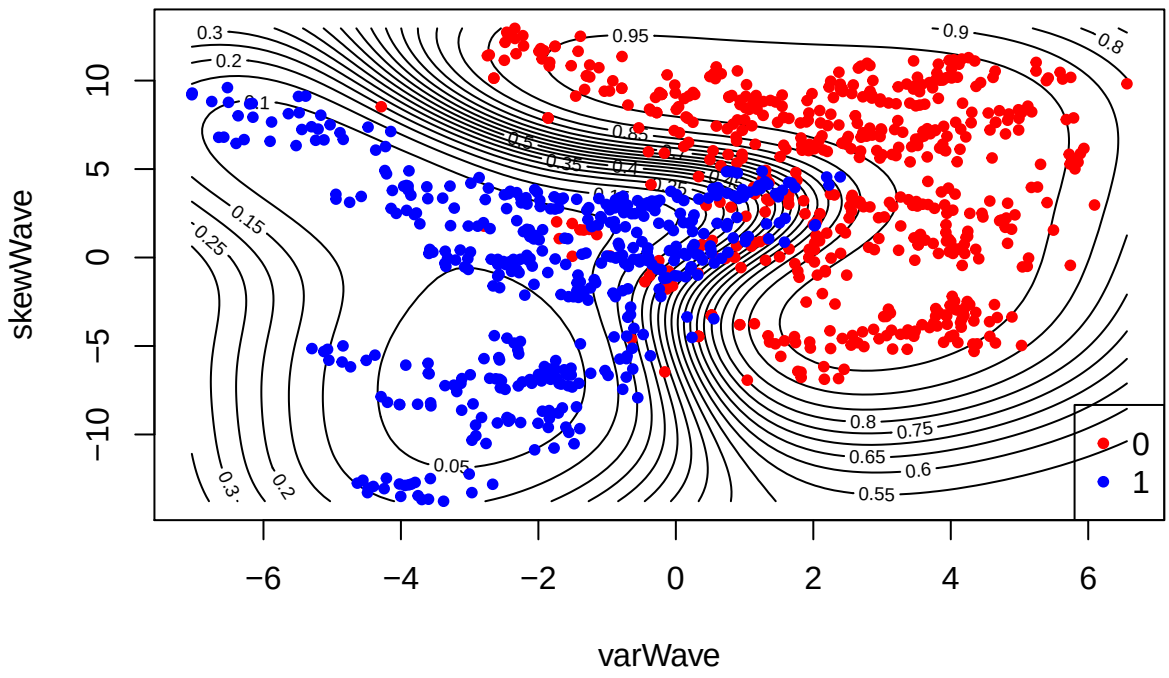
```
# data description
# http://archive.ics.uci.edu/ml/datasets/banknote+authentication

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

3-1.

```
# 3-1.

# GP fit with the given condition
gpcfit1 <- gausspr(fraud ~ varWave + skewWave, data=train)
```

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

```
# class probabilities
probPreds <- predict(gpcfit1, train[,1:2], type="probabilities")
x1 <- seq(min(train[,1]),max(train[,1]),length=100)
x2 <- seq(min(train[,2]),max(train[,2]),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(gpcfit1, gridPoints, type="probabilities")


# Plotting for Prob(0)
contour(x1,x2,matrix(probPreds[,1],100,byrow=TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prediction probability contour plot')
points(train[train[,5]=='0',1], train[train[,5]=='0',2],col="red",pch=20)
points(train[train[,5]=='1',1], train[train[,5]=='1',2],col="blue",pch=20)
legend("bottomright", legend=c("0","1"), pch=c(20,20), col=c("red","blue"))
```




```
# predict on the training set
pred1 <- predict(gpcfit1, train[,1:2])
conf_mat1 <- table(pred1, train$fraud) # confusion matrix
acc1 <- (conf_mat1[1,1] + conf_mat1[2,2]) / sum(conf_mat1)

list(Confusion_matrix=conf_mat1, Accuracy=acc1)
```

```
## $Confusion_matrix
##
## pred1   0   1
##     0 512  24
##     1  44 420
##
## $Accuracy
## [1] 0.932
```

3-2.

```
# 3-2.

# predict on the training set
pred2 <- predict(gpcfit1, test[,1:2])
conf_mat2 <- table(pred2, test$fraud) # confusion matrix
acc2 <- (conf_mat2[1,1] + conf_mat2[2,2]) / sum(conf_mat2)

list(Confusion_matrix=conf_mat2, Accuracy=acc2)
```

```
## $Confusion_matrix
##
## pred2   0   1
##     0 191   9
##     1  15 157
##
## $Accuracy
## [1] 0.9354839
```

3-3.

```
# 3-3.

# GP fit with all variables
gpcfit2 <- gausspr(fraud ~ ., data=train)
```

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

```
# predict on the training set
pred3 <- predict(gpcfit2, test[,1:4])
conf_mat3 <- table(pred3, test$fraud) # confusion matrix
acc3 <- (conf_mat3[1,1] + conf_mat3[2,2]) / sum(conf_mat3)

list(Confusion_matrix=conf_mat3, Accuracy=acc3)
```

```
## $Confusion_matrix
##
## pred3   0   1
##     0 205   0
##     1   1 166
##
## $Accuracy
## [1] 0.9973118
```

Here, the accuracy is 0.9973118 which are very close to 1 and actually there was only one misclassification. The result with only two covariates was not bad since the accuracy was 0.9354839, but obviously result gets better when all covariates are considered.