AML LAB 4

Harshavardhan Subramanian

15/10/2020

2.1 Implementing GP Regression

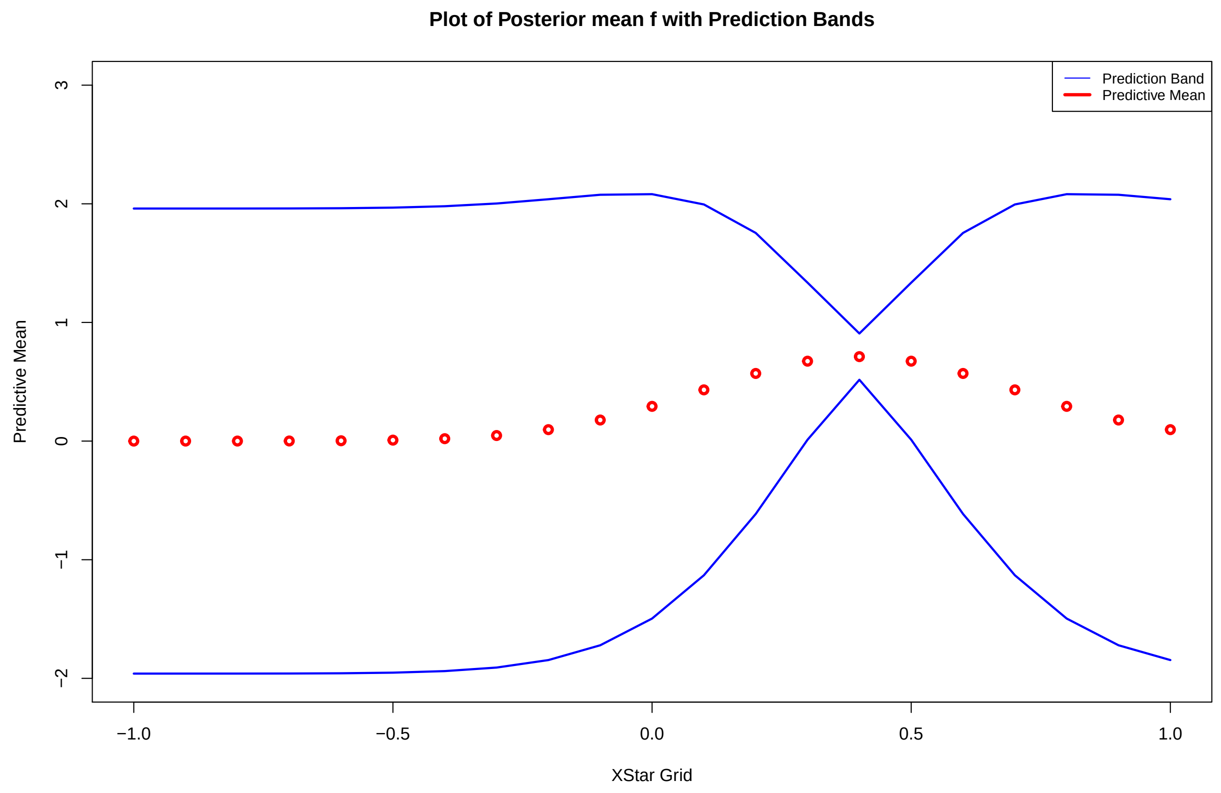
2.1.1

```r
library(AtmRay)
library(kernlab)
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=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)
{
  K <- k(X,X,sigmaF=1,l=0.3)
  KStar <- k(X,XStar,sigmaF=1,l=0.3)
  KStar2 <- k(XStar,XStar,sigmaF=1,l=0.3)
  L <- t(chol(K + sigmaNoise^2 * diag(length(X))))
  alpha <- solve(t(L),solve(L,Y))

  # Pred Mean
  f_star <- t(KStar) %*% alpha

  # Pred Var
  v <- solve(L,KStar)
  pred_var <- KStar2 - t(v) %*% v
  #logp <- -(t(Y) %*% alpha) / 2 - sum(log(L)) - length(X) * log(2 * pi) / 2
  return(list("F_mean" = f_star,"F_variance" = pred_var))

}
```

2.1.2

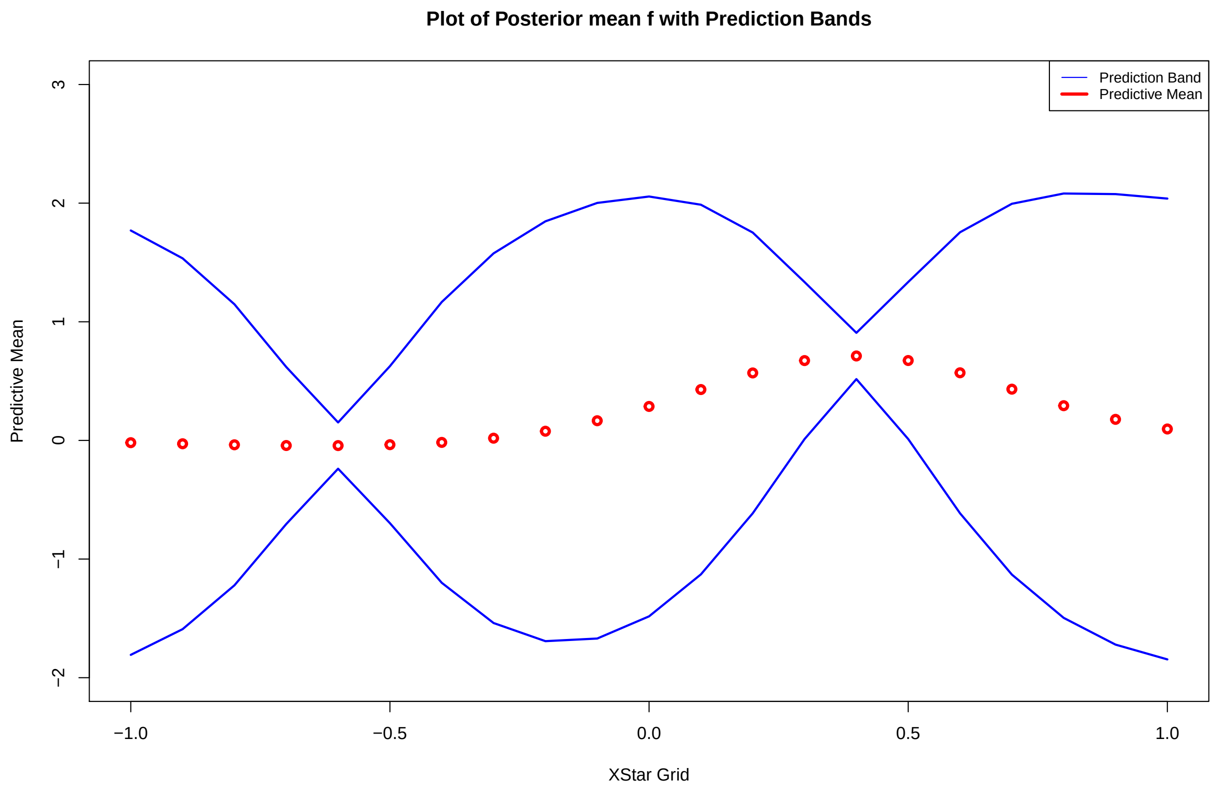
```r
XStargrid <- seq(-1,1,0.1)
#rnorm(21,0,1)

sim_post <- posteriorGP(X = 0.4, Y = 0.719, XStar = XStargrid, sigmaNoise = 0.1, k = SquaredExpKernel)

plot(XStargrid,sim_post$F_mean, col = "red", lwd = 3, ylim = c(-2,3), xlab = "XStar Grid", ylab = "Predi
lines(XStargrid, sim_post$F_mean - 1.96*sqrt(diag(sim_post$F_variance)), col = "blue", lwd = 2)
lines(XStargrid, sim_post$F_mean + 1.96*sqrt(diag(sim_post$F_variance)), col = "blue", lwd = 2)
```

```r
legend("topright",
       legend = c("Prediction Band", "Predictive Mean"),
       col = c("blue","red"),
       lty = c(1,1), lwd = c(1,3), cex = 0.8)
```



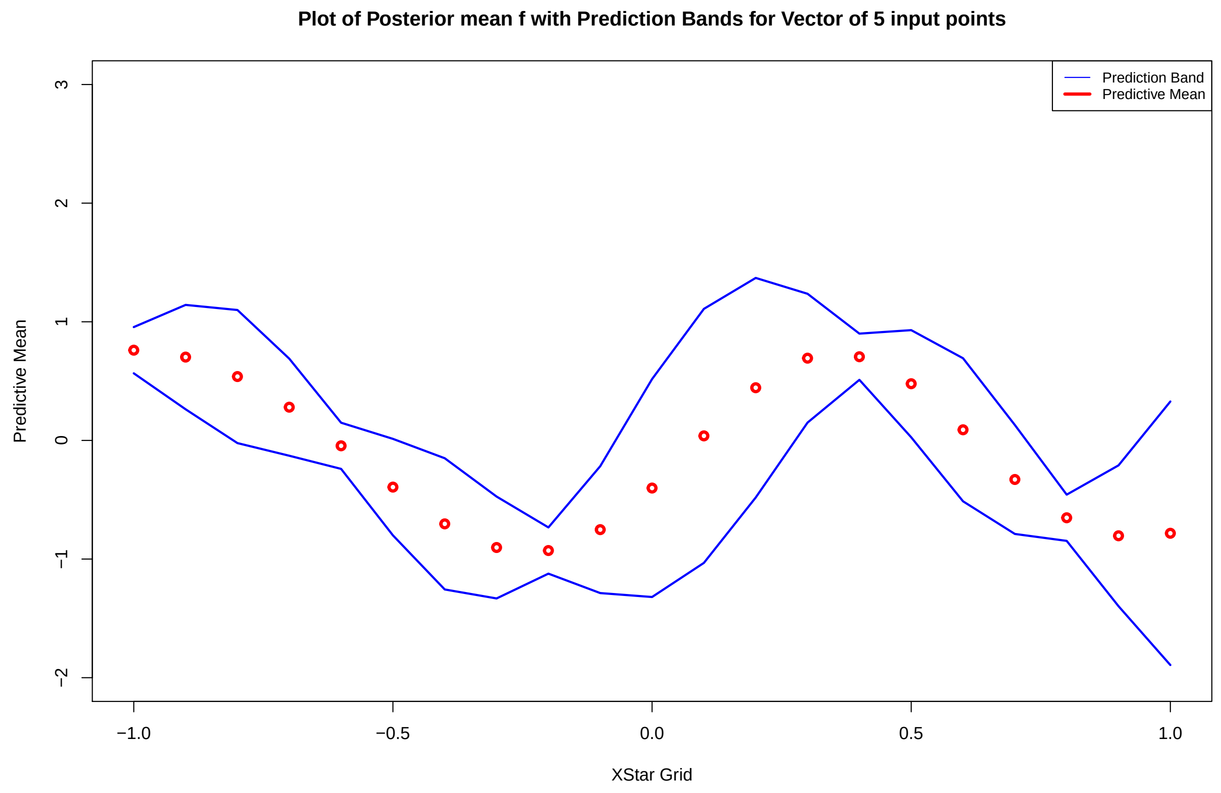
2.1.3

```r
X <- c(0.4,-0.60)
Y <- c(0.719,-0.044)
sim_post2 <- posteriorGP(X , Y , XStar = XStargrid, sigmaNoise = 0.1, k = SquaredExpKernel)
plot(XStargrid,sim_post2$F_mean, col = "red", lwd = 3, ylim = c(-2,3),xlab = "XStar Grid", ylab = "Pred
lines(XStargrid, sim_post2$F_mean - 1.96*sqrt(diag(sim_post2$F_variance)), col = "blue", lwd = 2)
lines(XStargrid, sim_post2$F_mean + 1.96*sqrt(diag(sim_post2$F_variance)), col = "blue", lwd = 2)
legend("topright",
       legend = c("Prediction Band", "Predictive Mean"),
       col = c("blue","red"),
       lty = c(1,1), lwd = c(1,3), cex = 0.8)
```



2.1.4

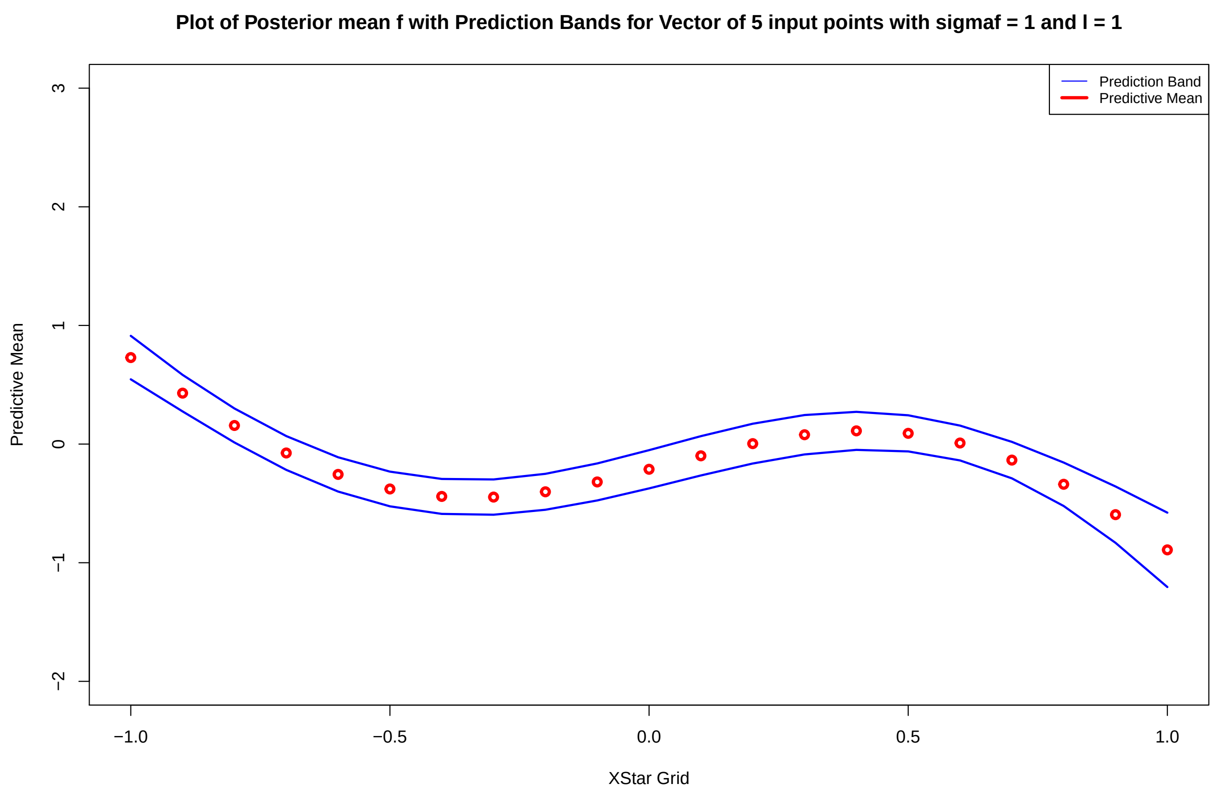
```
X <- c(-1.0,-0.6,-0.2,0.4,0.8)
Y <- c(0.768,-0.044,-0.940,0.719,-0.664)
sim_post3 <- posteriorGP(X , Y , XStar = XStargrid, sigmaNoise = 0.1, k = SquaredExpKernel)
par(mfrow = c(1,1))
#dim(sim_post3$F_variance)
plot(XStargrid,sim_post3$F_mean, col = "red", lwd = 3, ylim = c(-2,3),xlab = "XStar Grid", ylab = "Pred
lines(XStargrid, sim_post3$F_mean - 1.96*sqrt(diag(sim_post3$F_variance)), col = "blue", lwd = 2)
lines(XStargrid, sim_post3$F_mean + 1.96*sqrt(diag(sim_post3$F_variance)), col = "blue", lwd = 2)
legend("topright",
       legend = c("Prediction Band", "Predictive Mean"),
       col = c("blue","red"),
       lty = c(1,1), lwd = c(1,3), cex = 0.8)
```



2.1.5

```
posteriorGP2 <- function(X,Y,XStar,sigmaNoise,k)
{
  K <- k(X,X,sigmaF=1,l=1)
  KStar <- k(X,XStar,sigmaF=1,l=1)
  KStar2 <- k(XStar,XStar,sigmaF=1,l=1)
  L <- t(chol(K + sigmaNoise^2 * diag(length(X))))
  alpha <- solve(t(L),solve(L,Y))

  # Pred Mean
  f_star <- t(KStar) %*% alpha

  # Pred Var
  v <- solve(L,KStar)
  pred_var <- KStar2 - t(v) %*% v
  #logp <- -(t(Y) %*% alpha) / 2 - sum(log(L)) - length(X) * log(2 * pi) / 2
  return(list("F_mean" = f_star,"F_variance" = pred_var))

}

sim_post4 <- posteriorGP2(X , Y , XStar = XStargrid, sigmaNoise = 0.1, k = SquaredExpKernel)
par(mfrow = c(1,1))
plot(XStargrid,sim_post4$F_mean, col = "red", lwd = 3, ylim = c(-2,3),xlab = "XStar Grid", ylab = "Predi
lines(XStargrid, sim_post4$F_mean - 1.96*sqrt(diag(sim_post4$F_variance)), col = "blue", lwd = 2)
lines(XStargrid, sim_post4$F_mean + 1.96*sqrt(diag(sim_post4$F_variance)), col = "blue", lwd = 2)
```

```r
legend("topright",
       legend = c("Prediction Band", "Predictive Mean"),
       col = c("blue","red"),
       lty = c(1,1), lwd = c(1,3), cex = 0.8)
```



Comparing the model 4 and model 5 with different hyper parameters, it is evident that in the model 4, the Predictive mean curve is not so smooth since the smoothing parameter l is very less, it can also be seen that Predictive bands evaluated at the points equal to X* star values converges to a single point sinc the prediction at thta point is the point itself. Whereas in model 5, the smoothing parameter l has been increased to 1 which makes the Predictive mean very smooth and the Prediction band region is similar to all the points irrespective of the points closer to evaluation points or not.

2.2 GP Regression with kernlab

2.2.1

```r
TempTulling <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")
temp <- TempTulling$temp
time <- 1:length(temp)
#date <- TempTulling$date
day <- rep(1:365,6)
time <- rep(seq(1,time[length(time)],5))
temp <- temp[time]
day <- rep(seq(1,365,5),6)
library(kernlab)
```

```
Square_exp_kern <- function(sigmaf,l)
{
  rval <- function(x, y = NULL) {
    r <- sqrt(crossprod(x-y));
    return(sigmaf^2*exp(-r^2 / (2 * l^2)))
  }
  class(rval) <- "kernel"
  return(rval)
}


SEKFunc <-  Square_exp_kern(sigmaf = 1, l = 1) # SEKFunc is a kernel FUNCTION.
#SEKFunc(1,2) # Evaluating the kernel in x=1, x'=2.
# Computing the whole covariance matrix K from the kernel.
cat("The Covariance Matrix for the given Input Vectors \n")
```

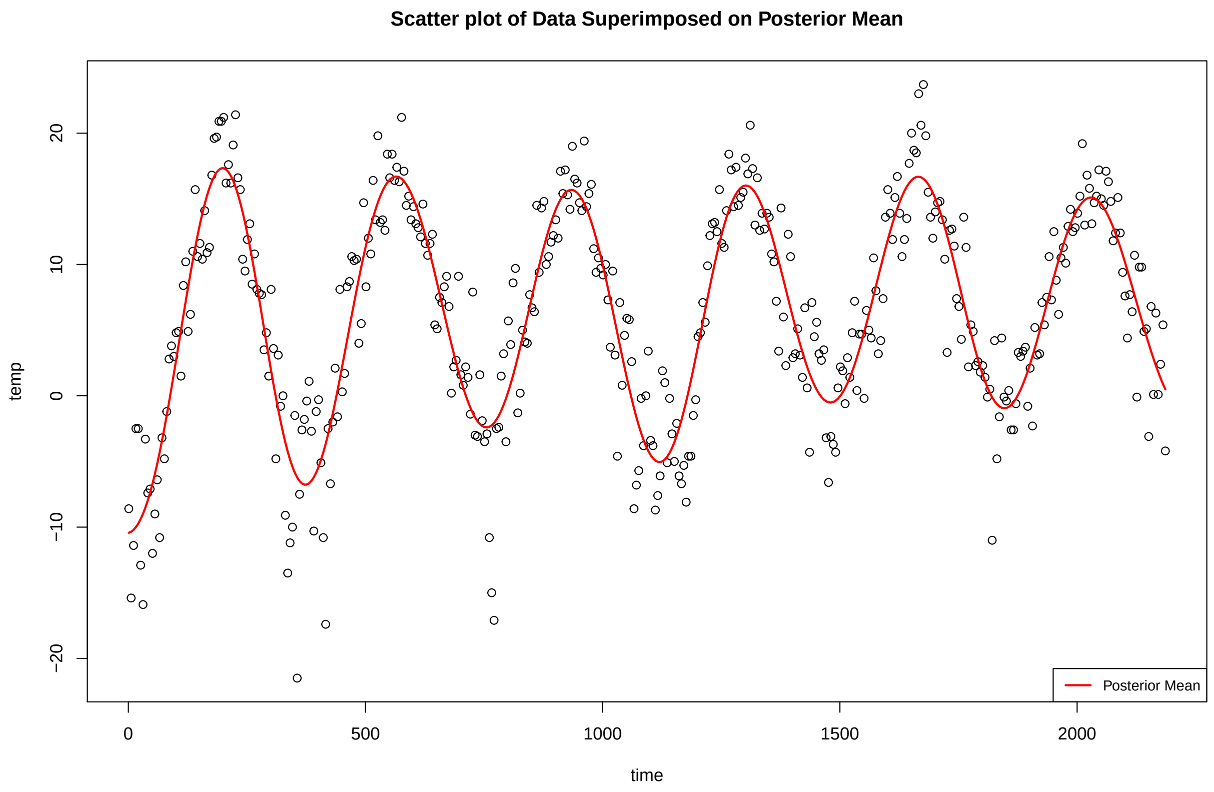
```
## The Covariance Matrix for the given Input Vectors
```

```
kernelMatrix(kernel = SEKFunc, x = c(1,3,4), y = c(2,3,4)) # So this is K(X,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
```

2.2.2

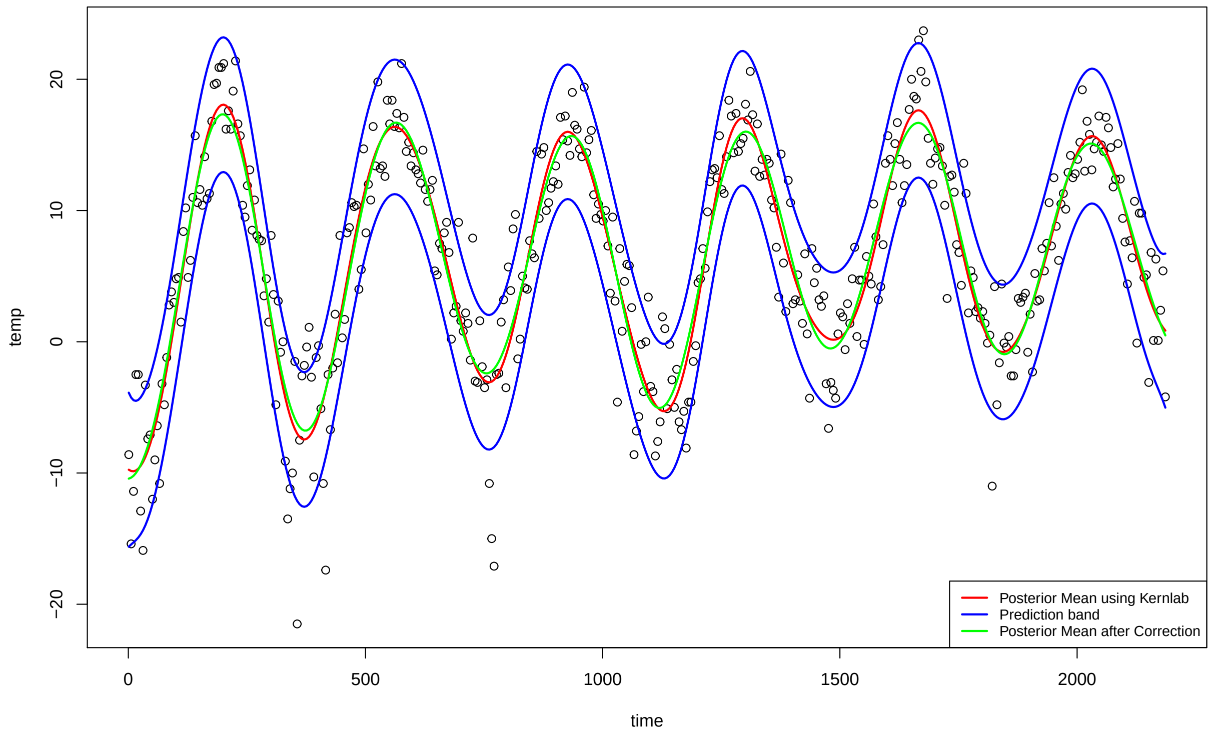
```
polyFit <- lm(temp ~  time + I(time^2) + I(time^3))
sigmaNoise = sd(polyFit$residuals)
#plot(time,temp)


SEkernel <- Square_exp_kern(sigmaf = 20,l = 0.2) # Note the reparametrization.
GPfit <- gausspr(time,temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, time) # Predicting the training data.
plot(time,temp,main = "Scatter plot of Data Superimposed on Posterior Mean")
lines(time, meanPred, col="red", lwd = 2)
legend("bottomright",
       legend = c("Posterior Mean"),
       col = c("red"),
       lty = c(1), lwd = c(2), cex = 0.8)
```



2.2.3

```
posteriorGP3 <- function(X,Y,XStar,sigmaNoise,k)
{
  K <- k(X,X,sigmaF=20,l=0.2)
  KStar <- k(X,XStar,sigmaF=20,l=0.2)
  KStar2 <- k(XStar,XStar,sigmaF=20,l=0.2)
  L <- t(chol(K + sigmaNoise^2 * diag(length(X))))
  alpha <- solve(t(L),solve(L,Y))

  # Pred Mean
  f_star <- t(KStar) %*% alpha

  # Pred Var
  v <- solve(L,KStar)
  pred_var <- KStar2 - t(v) %*% v
  #logp <- -(t(Y) %*% alpha) / 2 - sum(log(L)) - length(X) * log(2 * pi) / 2
  return(list("F_mean" = f_star,"F_variance" = pred_var))



}
temp_scale <- scale(temp)
time_scale <- scale(time)
polyFit <- lm(temp_scale ~  time_scale + I(time_scale^2) + I(time_scale^3))
sigmaNoise <- sd(polyFit$residuals)
```

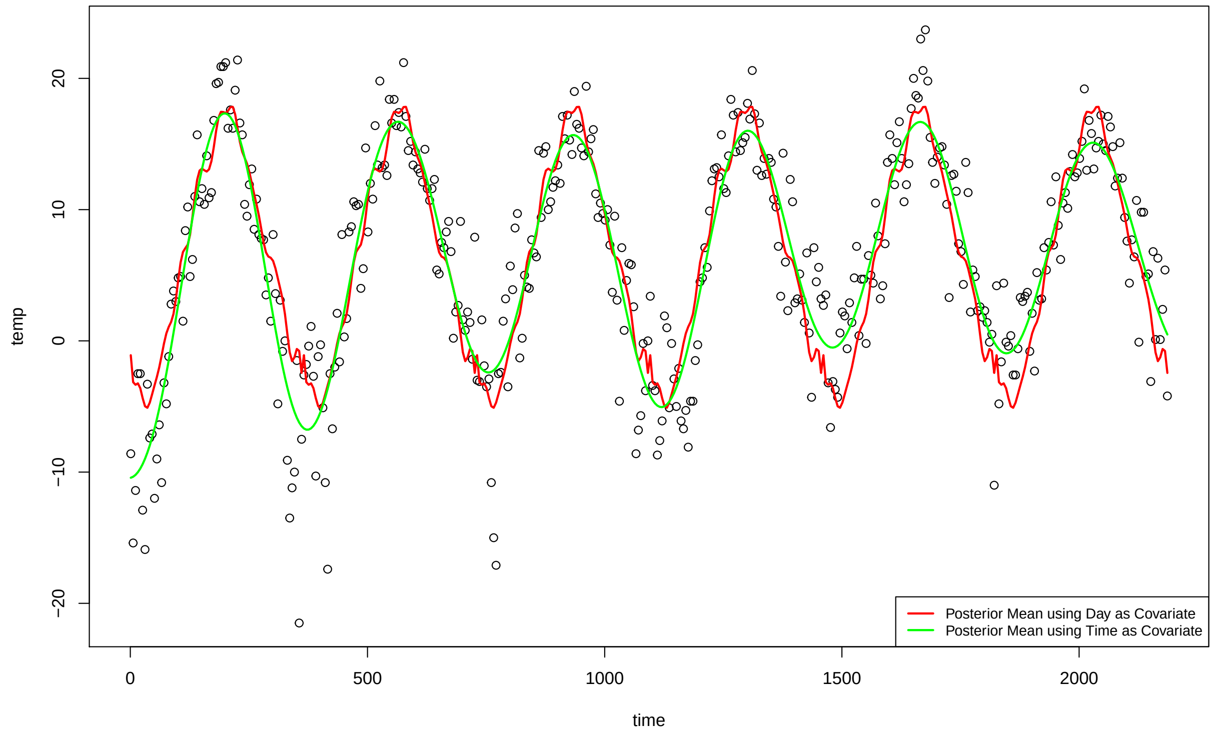
```
sim_GP <- posteriorGP3(X = time_scale, Y = temp_scale, XStar = time_scale, sigmaNoise, k = SquaredExpKe
pred_meanGP1 <- sim_GP$F_mean
pred_meanGP1 <- pred_meanGP1 * sd(temp) + mean(temp)
pred_varGP <- sim_GP$F_variance
pred_varGP <- pred_varGP  * sd(temp) + mean(temp)
#dim(pred_varGP)

plot(time,temp)
lines(time, pred_meanGP1, col="red", lwd = 2)
lines(time, pred_meanGP1 - 1.96*sqrt(diag(pred_varGP)), col = "blue", lwd = 2)
lines(time, pred_meanGP1 + 1.96*sqrt(diag(pred_varGP)), col = "blue", lwd = 2)
lines(time, meanPred, col="green", lwd = 2)
legend("bottomright",
       legend = c("Posterior Mean using Kernlab", "Prediction band","Posterior Mean after Correction"),
       col = c("red","blue","green"),
       lty = c(1,1,1), lwd = c(2,2,2), cex = 0.8)
```



2.2.4

```
day_scale <- scale(day)
#length(day_scale)
polyFit <- lm(temp_scale ~  day_scale + I(day_scale^2) + I(day_scale^3))
sigmaNoise <- sd(polyFit$residuals)
sim_GP <- posteriorGP3(X = day_scale, Y = temp_scale, XStar = day_scale, sigmaNoise, k = SquaredExpKern
pred_meanGP2 <- sim_GP$F_mean
```

```r
pred_meanGP2 <- pred_meanGP2 * sd(temp) + mean(temp)
pred_varGP <- sim_GP$F_variance
pred_varGP <- pred_varGP  * sd(temp) + mean(temp)

plot(time,temp)
lines(time,pred_meanGP2, col="red", lwd = 2)
lines(time, meanPred, col="green", lwd = 2)
legend("bottomright",
       legend = c("Posterior Mean using Day as Covariate","Posterior Mean using Time as Covariate"),
       col = c("red","green"),
       lty = c(1,1), lwd = c(2,2), cex = 0.8)
```

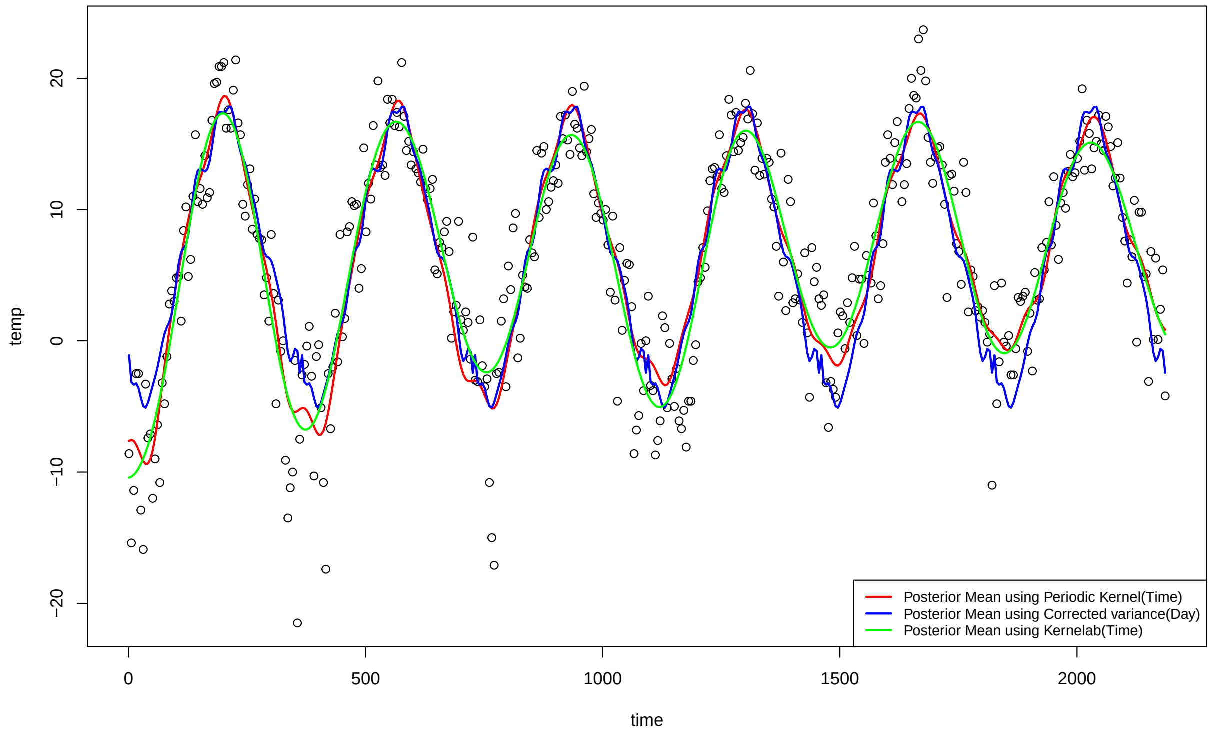


Comparing the results of both models, that is using Time as Covariate and Day as Covariate, we can see that the Posterior Mean using Day as Covariate predicts the data points better than model using Time as Covariate. The model with Time covariate has Predictive mean which looks very smooth and has a trend changing every year around the same days. Where as the model with Day as covariate has a constant seasonality which predicts almost the same for every year. Hence in terms of selecting a covariate, Day has prediction than Time. Pros of Time Covariate model is making better predictions of immediate future data whereas for Day Covariate model makes better long term prediction.

Cons of Time Covariate model is, it underfits the data by smoothening the Predictive mean curve where as for Day Covariate model,the Prediction mean curve attempts to overfit the data at few instances.

2.2.5

```r
Cust_Square_exp_kern <- function(x1,x2,sigmaf,l1,l2,d)
{
  rval <- function(x, y = NULL) {
    r <- sqrt(crossprod(x-y));
    return(sigmaf^2* exp((-2 * (sin(pi*r/d))^2)/l1^2) * exp((-0.5*r^2)/l2^2))
  }
  class(rval) <- "kernel"
  return(rval)
}

temp_scale <- scale(temp)
time_scale <- scale(time)
polyFit <- lm(temp_scale ~  time_scale + I(time_scale^2) + I(time_scale^3))
sigmaNoise <- sd(polyFit$residuals)
#d <- 365 / sd(time_scale)

#SEkernel <- Cust_Square_exp_kern(sigmaf = 20,l1 = 1, l2 = 10, d) # Note the reparametrization.
GPfit <- gausspr(time_scale,temp_scale, kernel = Cust_Square_exp_kern,
                 kpar = list(sigmaf=20,l1=1,l2=10,d=365/sd(time)),
                 var = sigmaNoise^2)
meanPred_gen <- predict(GPfit, time_scale) # Predicting the training data.
meanPred_gen <- meanPred_gen * sd(temp) + mean(temp)
plot(time,temp)
lines(time, meanPred_gen, col="red", lwd = 2)
lines(time,pred_meanGP2, col="blue", lwd = 2)
lines(time, meanPred, col="green", lwd = 2)
legend("bottomright",
       legend = c("Posterior Mean using Periodic Kernel(Time)","Posterior Mean using Corrected variance
       col = c("red","blue","green"),
       lty = c(1,1,1), lwd = c(2,2,2), cex = 0.8)
```



Comparing all the 3 models, from the plot it is evident that, Day Covariate model has better Prediction curve than the other two. In the other two models, there is a clear trend for same time of future years. The temporal dependency with respect to Day model is better captured, that is similar prediction for same day of every year where as the other two models has prediction dependent on immediate previous day values rather than same day in previous years.

2.3 GP Classification with kernlab

2.3.1

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

tr_data <- data[SelectTraining,]
te_data <- data[-SelectTraining,]
X_tr <- tr_data[,1:2] # Simulating some data.
Y_tr <- tr_data[,5]
#l <- 1

polyFit <- lm(Y_tr ~  X_tr[,1] + X_tr[,2])
sigmaNoise = sd(polyFit$residuals)
```

```
#plot(age,logWage)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
GPfit <- gausspr(X_tr,Y_tr, var = sigmaNoise^2)
```

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

```
meanPred <- predict(GPfit, X_tr, type = "response") # Predicting the training data.
cat("The confusion matrix for Train data: ")
```

```
## The confusion matrix for Train data:
```

```
cm <- table(meanPred,Y_tr)
cm
```

```
##          Y_tr
## meanPred   0   1
##        0 503  18
##        1  41 438
```

```
cat("Acurracy of Classification for Train data: ")
```

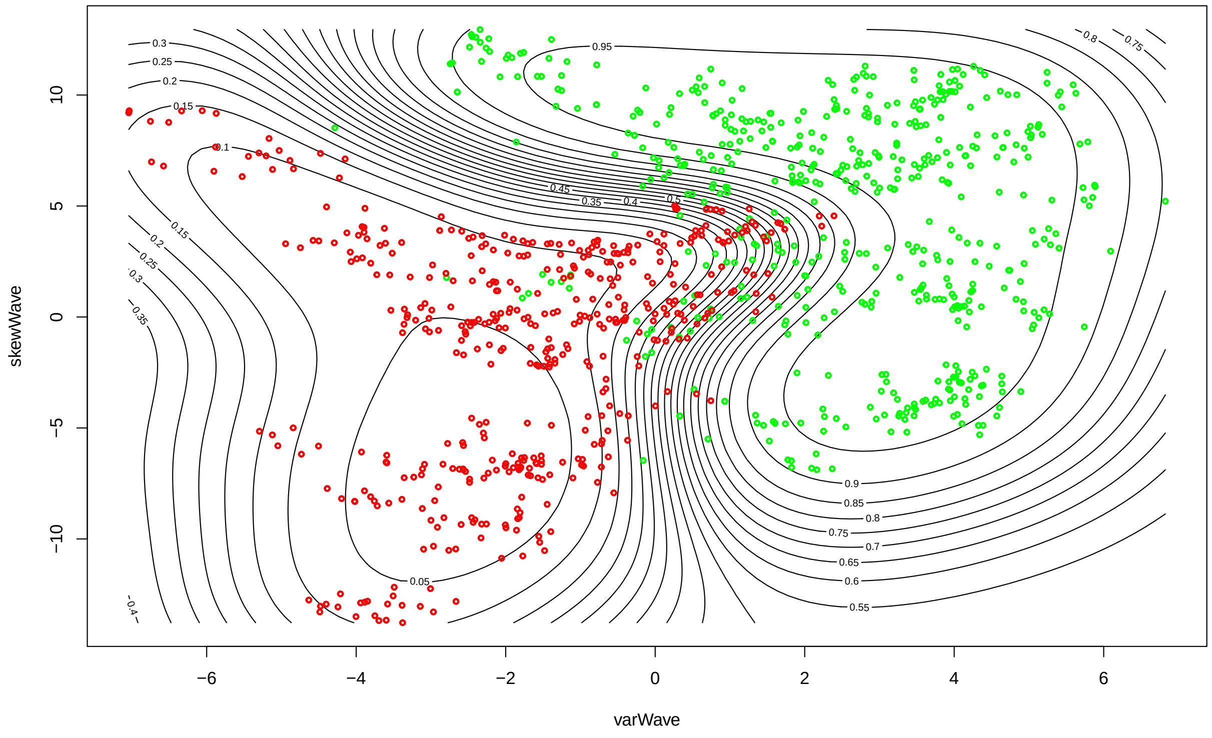
```
## Acurracy of Classification for Train data:
```

```
acc <- sum(diag(cm)) / sum(cm)
acc
```

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

```
x1 <- seq(min(X_tr[,1]),max(X_tr[,1]),length=100)
x2 <- seq(min(X_tr[,2]),max(X_tr[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(X_tr)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")
contour(x1,x2,
        matrix(probPreds[,1],nrow=100,ncol=100,byrow = TRUE),
        20,
        xlab = "varWave",
        ylab = "skewWave", main = "Contour plot of Prediction Probabilities")
points(tr_data[tr_data[,5]==0,1],tr_data[tr_data[,5]==0,2],col="green",lwd='2',cex=0.6)
points(tr_data[tr_data[,5]==1,1],tr_data[tr_data[,5]==1,2],col="red",lwd='2',cex=0.6)
```

Contour plot of Prediction Probabilities



2.3.2

```
X_te <- te_data[,1:2]
Y_te <- te_data[,5]
meanPredTe <- predict(GPfit, X_te, type = "response")
cm_te <- table(meanPredTe,Y_te)
cat("\n The confusion matrix for Test data: ")
```

```
##
##  The confusion matrix for Test data:
```

```
cm_te
```

```
##           Y_te
## meanPredTe   0   1
##          0 199   9
##          1  19 145
```

```
acc_te <- sum(diag(cm_te)) / sum(cm_te)
cat("\n Acurracy of Classification for Test data: ")
```

```
##
##  Acurracy of Classification for Test data:
```

```
acc_te
```

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

2.3.3

```
X_tr <- tr_data[,1:4] # Simulating some data.
Y_tr <- tr_data[,5]
#l <- 1

polyFit <- lm(Y_tr ~  X_tr[,1] + X_tr[,2] + X_tr[,3] + X_tr[,4])
sigmaNoise = sd(polyFit$residuals)
GPfitall <- gausspr(X_tr,Y_tr, var = sigmaNoise^2)
```

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

```
X_te <- te_data[,1:4]
Y_te <- te_data[,5]
meanPredTe <- predict(GPfitall, X_te, type = "response")
cat("\n The confusion matrix for Test data trained on all 4 Covariates: ")
```

```
##
##  The confusion matrix for Test data trained on all 4 Covariates:
```

```
cm_te <- table(meanPredTe,Y_te)
cm_te
```

```
##           Y_te
## meanPredTe   0   1
##          0 216   0
##          1   2 154
```

```
cat("\n Acurracy of Classification for Test data trained on all 4 Covariates: ")
```

```
##
##  Acurracy of Classification for Test data trained on all 4 Covariates:
```

```
acc_te <- sum(diag(cm_te)) / sum(cm_te)
acc_te
```

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