Lab4_AML

Andreas C Charitos[andch552]

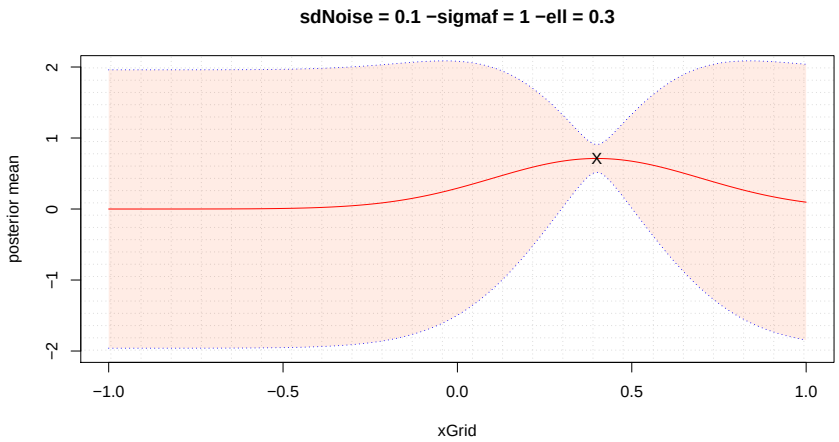
10/14/2019

Contents

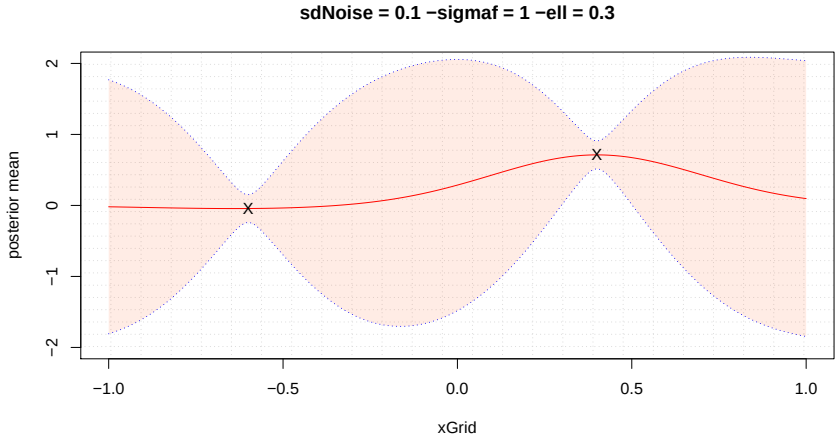
Assignment 2.1-Implement GP Regression

1

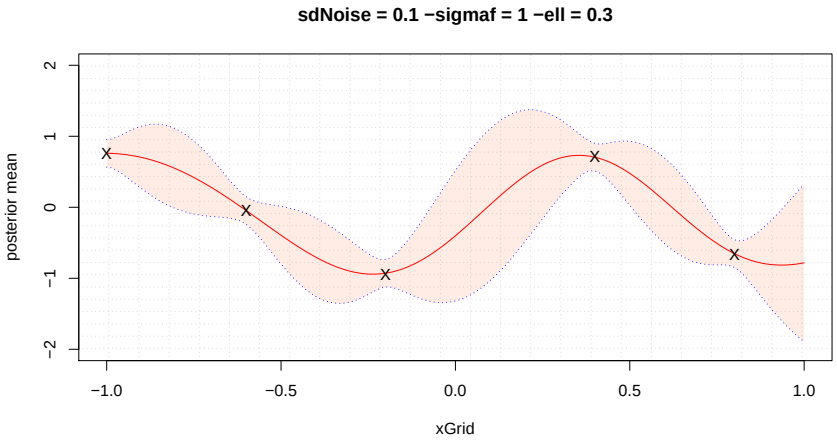
2



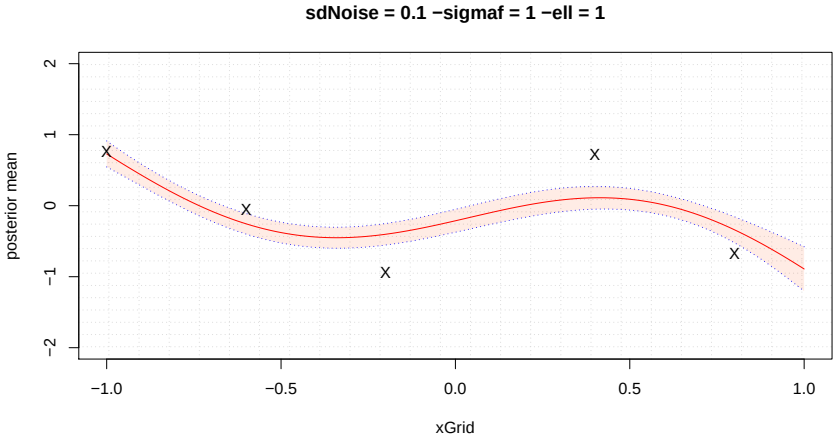
3



4



5



As we can see from the resulting plots as the parameter l increases the predictions mean line doesn't croses the points to predict so well as the ll with smaller value. The parameter l is refered to as the lengthscale and describes how smooth a function is. Small lengthscale value means that function values can change quickly, large values characterize functions that change only slowly. Lengthscale also determines how far we can reliably extrapolate from the training data.So if we have a large scale we are considering more points arround the estimated point and this makes the prediction mean curve more smooth but on the other hand of we set a small value for l we are considering fewer points arround the estimate point and this resulting a prediction mean curve less smoother.In conclusion, we can say that l controls the smoothness of the predictive mean.

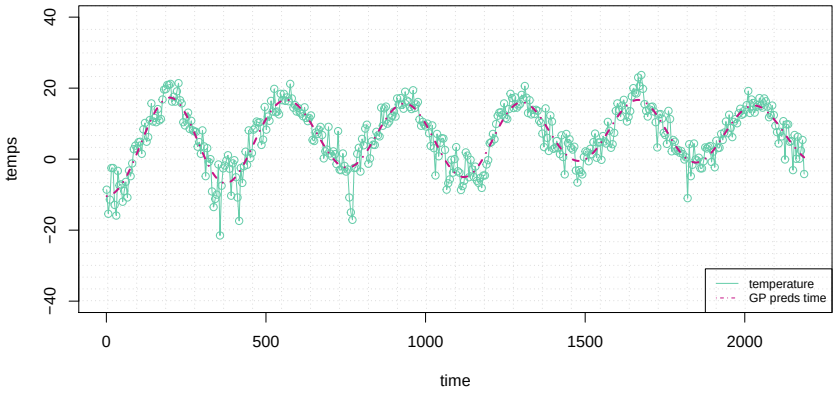
Assignment 2.2-GP Regression with kernlab

1

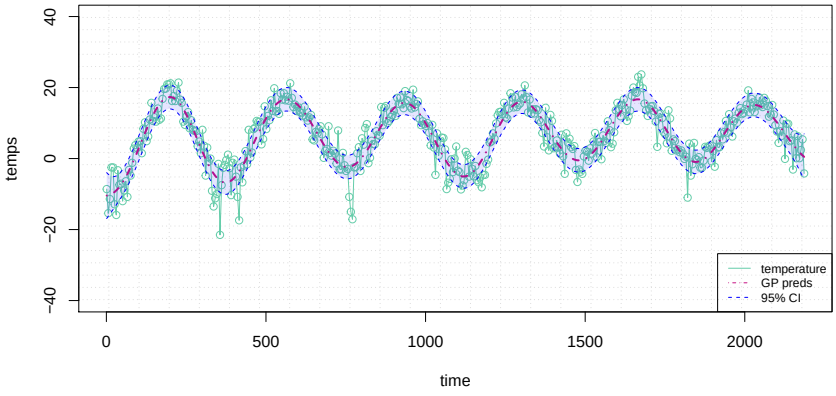
The result evaluating the kernel in $x = 1, x' = 2$ is 2.4261226 and the result for the input vectors $X = (1, 3, 4)^T, X^* = (2, 3, 4)^T$ is given below.

V1	V2	V3
2.4261226	0.5413411	0.044436
2.4261226	4.0000000	2.426123
0.5413411	2.4261226	4.000000

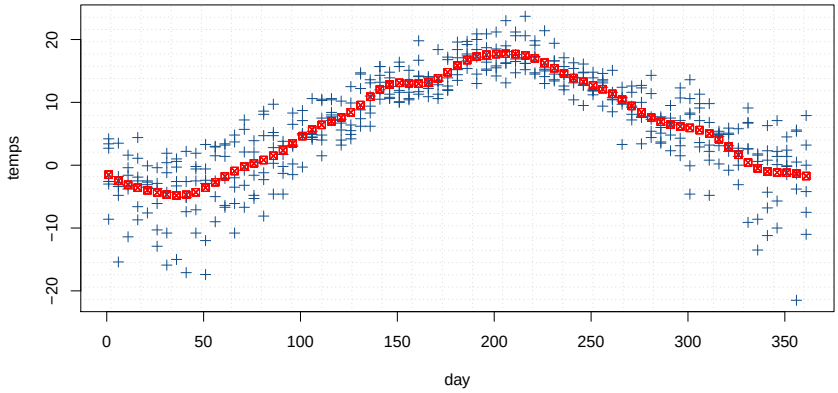
2

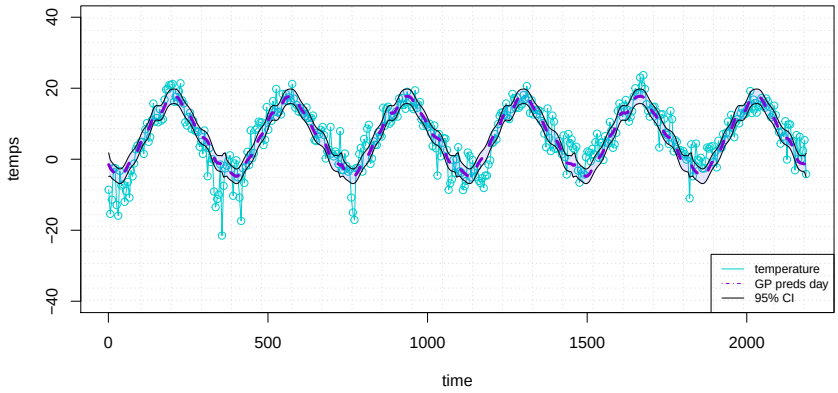


3

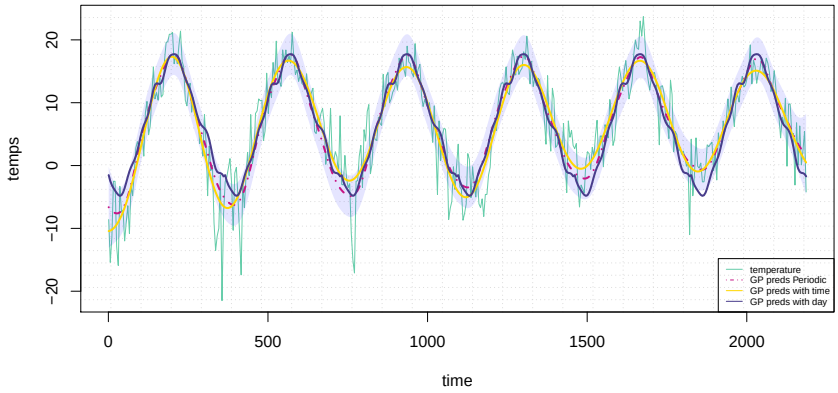


4





5



As we can see from the plots the resulting posterior mean using the time variable is more smooth compared

with one with the day variable.According to that result is more preferable to use the day as the covariate because the posterior mean matches the pattern of the data better.Regarding the posterior mean using the periodic kernel the results seems to be better compared with the previous 2 models.This is reasonable since the periodic kernel as its name suggests it has the ability to capture seasonal patterns in the data which is something that totally applies to our situation sinc we are working with temperature signals that have seasonality inside.

Assignment 2.3-Classification with kernlab

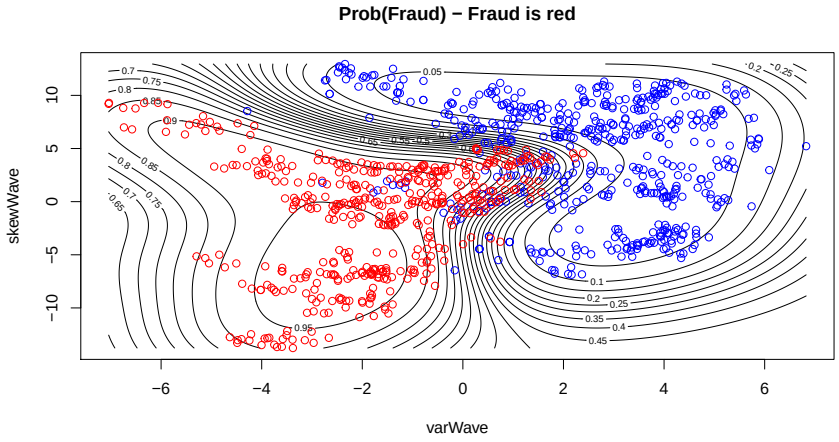
1

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

```
## The confusion matrix for train data is :
```

```
##     preds
## true   0   1
##    0 503  41
##    1  18 438
```

The train accuracy with covariates varWave and skewWave is 0.941



2

```
## The confusion matrix for test data is :
```

```
##     preds
## true   0   1
##    0 199  19
##    1   9 145
```

The test accuracy with all the covariates is : 0.925

3

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

```
## The confusion matrix for tesr data is :
```

```
##     preds
## true   0   1
##    0 216   2
##    1   0 154
```

	accuracy
Accuracy 2 covs	0.925
Accuracy all	0.995

As we can see from the table above the accuracy improves as the number of the covariates increases.

References

Kernel cookbook link

Covariance functions link

Apprendix

```
## ----global_options, R.options=knitr::opts_chunk$set(warning=FALSE, message=FALSE,echo=F,knitr.table.


## ------------------------------------------------------------------------

# Libraries ---------------------------------------------------------------
library(kernlab)
library(AtmRay)
library(ggplot2)
# ------------------------------------------------------------------------



## ------------------------------------------------------------------------

# 2.1-Implementing GP -----------------------------------------------------
# 1 -----------------
# Define helper functions

SquaredExpKernel <- function(x1,x2,sigmaF,l){
  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,...){
  ########################################################################
  #==================== Gasussian Process ============================#
  # inputs :  X -training inputs, y -target inputs, XStar -test inputs #
  #           sigmaNoise -noise sd, k -covariance matrix of kernel     #
  # outputs : fStar -posterior mean, vfStar -posterior variance,       #
  #           logMarg -log marginal likelihood                         #
  ########################################################################
  n=length(X)
  sigmaSqNoise=sigmaNOise^2
  K=k(X,X,...)
  d=dim(K)[1]
  L=t(chol(K+sigmaSqNoise*diag(d)))
  # predictive mean
  a=solve(t(L),solve(L,y))
  kStar=k(X,XStar,...)
  fStar=t(kStar)%*%a
  v=solve(L,kStar)
  vfStar=k(XStar,XStar,...)-t(v)%*%v
  #logMargLik= (-0.5)*(t(y)%*%solve(K+sigmaSqNoise%*%diag(d)))
  #%*%y-0.5*log(abs(K+sigmaSqNoise%*%diag(d)))-(n/2)*log(2*pi)
```

```
  return(list(fStar=fStar,vfStar=vfStar))

}

# ------------------



## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

# 2 -----------------

plot_func=function(xGrid,GPresults,sdNoise,sigmaf,ell,trX,try){
  plot(xGrid,GPresults$fStar,ylim=c(-2,2),col = "red",type="l",
       ylab="posterior mean",panel.first=grid(25,25))
  diagElem=diag(GPresults$vfStar)
  U=GPresults$fStar + 1.96*sqrt(diagElem)
  L=GPresults$fStar - 1.96*sqrt(diagElem)
  lines(xGrid,U,lty=3,col="blue")
  lines(xGrid,L,lty=3,col="blue")
  polygon(x = c(xGrid, rev(xGrid)),
        y=c(L,rev(U)),col =  adjustcolor("orangered", alpha.f = 0.10),
        border=NA )

  points(trX,try,col="black",pch="X")
  title(paste("sdNoise =" ,sdNoise, "-sigmaf =",sigmaf,"-ell =",ell ))
}

xGrid=seq(-1,1,by=0.01)

trX=0.4; try=0.719 ; sdNoise=0.1 ; sigmaf=1; ell=0.3

results=posteriorGP(trX,try,xGrid,sdNoise,SquaredExpKernel,sigmaf,ell)

plot_func(xGrid,results,sdNoise,sigmaf,ell,trX,try)

# --------------------



## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

# 3 -----------------

xGrid=seq(-1,1,by=0.01)

trX=c(0.4,-0.6) ; try=c(0.719,-0.044) ; sdNoise=0.1 ; sigmaf=1; ell=0.3

results1=posteriorGP(trX,try,xGrid,sdNoise,SquaredExpKernel,sigmaf,ell)

plot_func(xGrid,results1,sdNoise,sigmaf,ell,trX,try)

# --------------------
```

```r
## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

# 4 -----------------

xGrid=seq(-1,1,by=0.01)

trX=c(-1.0,-0.6,-0.2,0.4,0.8) ; try=c(0.768,-0.044,-0.940,0.719,-0.664)
sdNoise=0.1 ; sigmaf=1; ell=0.3

results2=posteriorGP(trX,try,xGrid,sdNoise,SquaredExpKernel,sigmaf,ell)

plot_func(xGrid,results2,sdNoise,sigmaf,ell,trX,try)

# -------------------



## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------
# 5 -----------------

xGrid=seq(-1,1,by=0.01)

trX=c(-1.0,-0.6,-0.2,0.4,0.8) ; try=c(0.768,-0.044,-0.940,0.719,-0.664)
sdNoise=0.1 ; sigmaf=1; ell=1

results3=posteriorGP(trX,try,xGrid,sdNoise,SquaredExpKernel,sigmaf,ell)

plot_func(xGrid,results3,sdNoise,sigmaf,ell,trX,try)

# ------------------------------------------------------------------------



## ------------------------------------------------------------------------
# 2.2-GP Regression with kernlab ------------------------------------------

# 1 -----------------
temps=read.csv(
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv",
  header=TRUE,sep=";")

temps$time=1:dim(temps)[1] ; temps$day=1:365
temps.new= temps[seq(1, nrow(temps), 5), ]


SEkernel=function(sigmaf = 1,ell=1){
  rval <- function(x, y = NULL) {
    if (!is(x, "vector"))
      stop("x must be a vector")
    if (!is(y, "vector") && !is.null(y))
      stop("y must a vector")
```

```
    if (is(x, "vector") && is.null(y)) {
      return(1)
    }
    if (is(x, "vector") && is(y, "vector")) {
      if (!length(x) == length(y))
        stop("number of dimension must be the same on both data points")
      r2=crossprod(x-y)
      res=(sigmaf^2)*exp(-1*(r2/(2*(ell^2))))

      return(res)

    }
  }
  #class(rval)<-"kernel"
  #return(rval)
  return(new("kernel", .Data = rval, kpar = list(sigmaf = sigmaf,ell=ell)))
}

SEkernFunc = SEkernel(sigmaf = 2, ell = 1) # MaternFunc is a kernel FUNCTION
S1=SEkernFunc(1,2) # Evaluating the kernel in x=1, x'=2



## ----fig.align="center"--------------------------------------------------
# Computing the whole covariance matrix K from the kernel.
X=c(1,3,4) ; Xstar=c(2,3,4) # input vectors
K=kernelMatrix(kernel =SEkernel(2,1), x = X, y = Xstar) # So this is K(X,Xstar)

knitr::kable(as.data.frame(K))

# -------------------


## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------
# 2 -----------------
# Estimating the noise variance from a second degree polynomial fit
polyFit= lm(temp ~  time + I(time^2),data=temps.new )
sigmaNoise_time= sd(polyFit$residuals)

# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)
#GPfit=gausspr(temps.new$temps, temps.new$time, kernel = rbfdot,
                              #kpar = list(sigma=20), var = sigmaNoise^2)
GPfit=gausspr(temps.new$time, temps.new$temp, kernel = SEkernel,
              kpar = list(sigmaf =20,ell=0.2), var = sigmaNoise_time^2)
meanPredsTime=predict(GPfit, temps.new$time) # Predicting the training data. To plot the fit.
plot(y=temps.new$temp,x=temps.new$time,type="o",
     col="mediumaquamarine",ylab="temps",xlab="time",panel.first=grid(25.25),
     ylim=c(-40,40))
lines(x=temps.new$time,y=meanPredsTime,
      col="mediumvioletred", lwd = 2,lty=4)
legend("bottomright",legend=c("temperature","GP preds time"),
       col=c("mediumaquamarine","mediumvioletred"),lty=c(1,4),cex=0.75)
```

```r
## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

# 3 -----------------

GPfit_time=posteriorGP(scale(temps.new$time),scale(temps.new$temp),
                       scale(temps.new$time),sigmaNoise_time,SquaredExpKernel,20,0.2)

plot(y=temps.new$temp,x=temps.new$time,type="o",
     col="mediumaquamarine",ylim=c(-40,40),ylab="temps",xlab="time",
     panel.first=grid(25,25))
lines(x=temps.new$time,y=meanPredsTime, col="mediumvioletred", lwd = 2,lty=4)
# L= GPfit_time$fStar - 1.96*sqrt(diag(GPfit_time$vfStar))
# U= GPfit_time$fStar + 1.96*sqrt(diag(GPfit_time$vfStar))
L= meanPredsTime - 1.96*sqrt(diag(GPfit_time$vfStar))
U= meanPredsTime + 1.96*sqrt(diag(GPfit_time$vfStar))
lines(temps.new$time, L, col = "blue", lwd = 1,lty=2)
lines(temps.new$time, U, col = "blue", lwd = 1,lty=2)
polygon(x = c(temps.new$time, rev(temps.new$time)),
        y=c(L,rev(U)),col =  adjustcolor("blue", alpha.f = 0.10),
        border=NA )
legend("bottomright",legend=c("temperature","GP preds","95% CI"),
       col=c("mediumaquamarine","mediumvioletred","blue"),lty=c(1,4,2),
       cex=0.75)

# -------------------



## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

# 4 -----------------
polyFit= lm(temp ~  day + I(day^2),data=temps.new)
sigmaNoise_day= sd(polyFit$residuals)

GPfit_day=posteriorGP(scale(temps.new$day),temps.new$temp,scale(temps.new$day),
                      sigmaNoise_day,SquaredExpKernel,20,0.2)

GPfit_day1=gausspr(temps.new$day, temps.new$temp, kernel = SEkernel,
                   kpar = list(sigmaf =20,ell=0.2), var = sigmaNoise_day^2)

meanPredsDay=predict(GPfit_day1, temps.new$day)

plot(y=temps.new$temp,x=temps.new$day,col="dodgerblue4",pch=3,
     ylab="temps",xlab="day",panel.first=grid(25,25))
points(y=meanPredsDay,x=temps.new$day,col="red",pch=7)



## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

plot(y=temps.new$temp,x=temps.new$time,type="o",col="darkturquoise",ylim=c(-40,40),
     ylab="temps",xlab="time",panel.first=grid(25,25))
```

```r
lines(x=temps.new$time,y=meanPredsDay, col="darkviolet", lwd = 3,lty=2)
#lines(x=temps.new$time,y=GPfit_day$fStar, col="gray", lwd = 2,lty=1)
# L= GPfit_day$fStar - 1.96*sqrt(diag(GPfit_day$vfStar))
# U= GPfit_day$fStar + 1.96*sqrt(diag(GPfit_day$vfStar))
L= meanPredsDay - 1.96*sqrt(diag(GPfit_day$vfStar))
U= meanPredsDay + 1.96*sqrt(diag(GPfit_day$vfStar))
lines(temps.new$time, L, col = "black", lwd = 1,lty=1)
lines(temps.new$time, U, col = "black", lwd = 1,lty=1)
polygon(x = c(temps.new$time, rev(temps.new$time)),
        y=c(L,rev(U)),col =  adjustcolor("blue", alpha.f = 0.10),
        border=NA )
legend("bottomright",legend=c("temperature","GP preds day","95% CI"),
       col=c("darkturquoise","darkviolet","black"),lty=c(1,4,1),cex = 0.75)

# --------------------



## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

# 5 -----------------

perKernel=function(sigmaf,ell1,ell2,d){
  rval <- function(x, y = NULL) {
    if (!is(x, "vector"))
      stop("x must be a vector")
    if (!is(y, "vector") && !is.null(y))
      stop("y must a vector")
    if (is(x, "vector") && is.null(y)) {
      return(1)
    }
    if (is(x, "vector") && is(y, "vector")) {
      if (!length(x) == length(y))
        stop("number of dimension must be the same on both data points")
      r2=crossprod(x-y)
      p1=(-2)*sin((pi*abs(x-y))/d)^2
      res=(sigmaf^2)*exp(p1/ell1^2)*exp(-r2/(2*ell2^2))
      return(res)
    }
  }
  return(new("kernel", .Data = rval, kpar = list(sigmaf = sigmaf,ell1=ell1,ell2,d)))
}




## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------

sigmaf=20 ; ell1=1 ; ell2=10 ; dd=365/sd(temps.new$time)

GPfit_time_per=gausspr(temps.new$time, temps.new$temp, kernel = perKernel,
                       kpar = list(sigmaf =sigmaf,ell1=ell1,ell2=ell2,d=dd), var = sigmaNoise_time^2)
```

```r
GPfit_time_=posteriorGP(scale(temps.new$time),scale(temps.new$temp),scale(temps.new$time),
                        sigmaNoise_time,SquaredExpKernel,20,0.2)

meanPredPer=predict(GPfit_time_per, temps.new$time) # Predicting the training data. To plot the fit.

plot(y=temps.new$temp,x=temps.new$time,type="l",col="mediumaquamarine",
     ylab="temps",xlab="time",panel.first=grid(25,25))
L= meanPredPer - 1.96*sqrt(diag(GPfit_time_$vfStar))
U= meanPredPer + 1.96*sqrt(diag(GPfit_time_$vfStar))
polygon(x = c(temps.new$time, rev(temps.new$time)),
        y=c(L,rev(U)),col =  adjustcolor("blue", alpha.f = 0.10),
        border=NA )
lines(x=temps.new$time,y=meanPredPer, col="mediumvioletred", lwd = 2,lty=4)
lines(x=temps.new$time,y=meanPredsTime, col="gold", lwd = 2,lty=1)
lines(x=temps.new$time,y=meanPredsDay, col="darkslateblue", lwd = 2,lty=1)
legend("bottomright",legend=c("temperature","GP preds Periodic","GP preds with time",
                              "GP preds with day"),
       col=c("mediumaquamarine","mediumvioletred","gold","darkslateblue"),lty=c(1,4,1,1),
       cex = 0.55)

# ------------------------------------------------------------------------



## ------------------------------------------------------------------------

# 2.3-GP Classification with kernlab ------------------------------------------

# 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)
trainData=data[SelectTraining,]
testData=data[-SelectTraining,]

GPfraud=gausspr(fraud~varWave+skewWave,data=trainData)
# predict on the training set
covariates_indx=which(colnames(trainData)%in%c("varWave","skewWave"))
trainPreds=predict(GPfraud,trainData[,covariates_indx])
cat("The confusion matrix for train data is :\n")
table( trainData$fraud,trainPreds,dnn = c("true","preds")) # confusion matr



## ------------------------------------------------------------------------
trainAccuracy=sum(trainPreds==trainData$fraud)/length(trainData$fraud)
#trainAccuracy
```

```
## ----fig.width=9,fig.height=5,fig.align="center",out.width = '70%'-------
# class probabilities
probPreds = predict(GPfraud,trainData[,covariates_indx] , type="probabilities")
x1 = seq(min(trainData$varWave),max(trainData$varWave),length=200)
x2 = seq(min(trainData$skewWave),max(trainData$skewWave),length=200)
gridPoints = meshgrid(x1, x2)
gridPoints = cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(trainData)[covariates_indx]
probPreds <- predict(GPfraud, gridPoints, type="probabilities")

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,2],200,byrow = TRUE), 20,
        xlab = "varWave", ylab = "skewWave", main = 'Prob(Fraud) - Fraud is red')
points(trainData[trainData[,5]=='0',1],trainData[trainData[,5]=='0',2],col="blue")
points(trainData[trainData[,5]=='1',1],trainData[trainData[,5]=='1',2],col="red")



## ------------------------------------------------------------------------

# 2 -----------------

testPreds=predict(GPfraud,testData[,covariates_indx])
cat("The confusion matrix for test data is :\n")
table(testData$fraud,testPreds, dnn = c("true","preds")) # confusion matrix
testAccuracy=sum(testPreds==testData$fraud)/length(testData$fraud)
testAccuracy=round(testAccuracy,3)
#testAccuracy



## ------------------------------------------------------------------------

# 3 -----------------
GPfraudAll=gausspr(fraud~.,data=trainData)


testPredsAll=predict(GPfraudAll,testData[,-5])
cat("The confusion matrix for tesr data is :\n")
table(testData$fraud,testPredsAll,dnn=c("true","preds"))

# ------------------------------------------------------------------------



## ----fig.align="center"--------------------------------------------------
testAccuracyAll=sum(testPredsAll==testData$fraud)/length(testData$fraud)
testAccuracyAll=round(testAccuracyAll,3)

Dt=as.data.frame(cbind(c("Accuracy 2 covs","Accuracy all"),
                       c(testAccuracy,testAccuracyAll)))
```

```r
colnames(Dt)=c("","accuracy")
knitr::kable(Dt)

# ------------------------------------------------------------------------------



## ----code = readLines(knitr::purl("~/Courses/Advanced ML/Labs/Lab4/LAB4_AML.Rmd",documentation = 1)),
## NA
```