title: "154 - Lab 4" output: pdf_document date: "2022-12-05"

Here is my code for fitting a GPR.

```
rbf.k <- function(x,y,gamma)
  return(exp(-gamma*sum((x-y)^2)))

true.f <- function(x) {
  return(-(sin(x[1]*8) + x[2]^2 + x[1]*sqrt(x[2]) + 3 * dnorm(
x[1],.5,.2) * dnorm(x[2],.5,.2)))
}



gpreg <- function(x, y, gamma, sig, design) {
  # Evaluates mean and covariance of GP at grid of points on [
0,1]
  # Inputs:
  #   x, y: input and output values of data set
  #   lam: smoothing parameter in RBF kernel
  #   sig: error standard deviation of y
  #   design: grid of points to evaluate the GP
  # Returns:
  #   mean=posterior mean, vars=posterior variance, and design
=evaluation points
  n <- length(y)
  Sigma <- matrix(0,nrow=n+nrow(design), ncol=n+nrow(design))
  all <- rbind(x, design)
  for (i in 1:nrow(Sigma)) {
    for (j in i:nrow(Sigma))
      Sigma[i,j] <- rbf.k(all[i,], all[j,], gamma) -> Sigma[j,
i]
  }
  S11 <- Sigma[1:n, 1:n]
  S12 <- Sigma[1:n, (n+1):ncol(Sigma)]
  S21 <- Sigma[(n+1):ncol(Sigma), 1:n]
  S22 <- Sigma[(n+1):ncol(Sigma),(n+1):ncol(Sigma)]
  inv <- S21%*%solve(S11+sig^2*diag(n))
  mean <- inv%*%y
  cov <- S22-inv%*%S12
  vars <- diag(cov)
  return(list(mean=mean, vars=vars))
}
```

We are going to use Gaussian Process Optimization to try to minimize the function *true.f* above.

We are going to start with 10 evaluations of the function, but at random points (for fun). Our grid is going to be fairly coarse, because there are some matrix operations required, and the code is not optimized.

```
###Sample Usage
set.seed(4)
n <- 10
x <- cbind(runif(n),runif(n))
y <- c()
sig <- 0
for (i in 1:n)
  y[i] <- true.f(x[i,])
y <- y + rnorm(n, 0, sig)

design <- c()
grid <- seq(0,1,.03)
best.min <- 1e20
truth <- matrix(0,nrow=length(grid), ncol=length(grid))
for (i in 1:length(grid)) {
  for (j in 1:length(grid)) {
    design <- rbind(design, c(grid[i], grid[j]))
    truth[i,j] <- true.f(c(grid[i], grid[j]))
    if (truth[i,j] < best.min) {
      best.min <- truth[i,j]
      best.grid <- c(grid[i], grid[j])
    }
  }
}

gpfit <- gpreg(x, y, 10, 0.000, design)

image(grid,grid,matrix(gpfit$mean,nrow=sqrt(length(gpfit$mean)
), byrow = TRUE))
points(x, col='blue',pch=19)
image(grid,grid,truth)
min(y)-min(truth)
```

Above are the estimated function based on the 10 evaluations (with the location of the evaluations visualized), the true function, and the available improvement. Note that we need to choose a value of γ in fitting the GPR, where the kernel is

$$K(x, y) = e^{-\gamma(\|x-y\|^2)}$$

So large γ means flexible learner.

Let's compute the expected improvement (optimizing the possible improvement while not penalizing for evaluations that don't move the needle). Then we'll do the additional evauliation, and refit the GPR. It's possible to do fewer calculations in fitting the GRP (treating the old posterior as the new prior), but since the GRP fit is relatively inexpensive, we'll just refit the whole thing with the slightly larger sample size.

```
f_prime <- min(y)
ei <- c()
for (i in 1:length(gpfit$mean)) {
  if (gpfit$vars[i] > 0) {
    ei[i] <- (f_prime - gpfit$mean[i])*pnorm(f_prime,gpfit$mea
n[i], sqrt(gpfit$vars[i])) + gpfit$vars[i]*dnorm(f_prime,gpfit
$mean[i], sqrt(gpfit$vars[i]))
  } else {
    ei[i] <- 0
  }
}
image(grid,grid,matrix(ei,nrow=sqrt(length(gpfit$mean)), byrow
= TRUE))
best.ei <- design[which.max(ei),]

x <- rbind(x,best.ei)
y <- c(y, true.f(best.ei))
gpfit <- gpreg(x, y, 10, 0, design)

image(grid,grid,matrix(gpfit$mean,nrow=sqrt(length(gpfit$mean)
), byrow = TRUE))
points(x, col='blue',pch=19)
image(grid,grid,truth)
min(y)-min(truth)
```

We can run the last chunck repeatedly, each time adding the current best new evaulation point. It should find the bottom after 3 additional evaluations.

Tune a learner with a bivariate tuning parameter (so randomForest with mtry and maxnodes or svm with cost and gamma) on a data set of your choosing using Gaussian Process Optimization.