Homework 1

STAT6306

Installing and loading packages

```
packages = c('biglm','foreach','doParallel')
for(package in packages){
  if(!require(package,character.only=TRUE,quietly=TRUE)){
    install.packages(package,repos='http://cran.us.r-project.org')
    require(package,character.only=TRUE,quietly=TRUE)
  }
}
```

Out-of-memory Least Squares via biglm

An important part of being a data scientist is getting comfortable experimenting via simulation. Simulation is the process of generating data from a known distribution. Then, the performance of any procedure(s) can be evaluated/explored precisely due to knowing the truth (which would be unknowable for a data set). Let's first experiment with fitting the least squares procedure to simulated data.

```
set.seed(1)
n = 2000
p = 500
X = matrix(rnorm(n*p),nrow=n,ncol=p)
X[,1] = 1
format(object.size(X),units='auto')#memory used by X
```

```
## [1] "7.6 Mb"
```

```
b = rep(0,p)
b[1:5] = 25
b_0 = 0
Xdf = data.frame(X)
Y = b_0 + X %*% b + rnorm(n)
hatBeta = coef(lm(Y~X-1)) #Here, the [-1] ignores the intercept
```

Part a

Report the first 5 entries in $\hat{\beta}$ (that is, hatBeta in the above code) using lm on all the data simultaneously

```
#SOLUTION
```

Part b

Though this is practically speaking a small problem (see the 'object.size' above), let's pretend it is too large to fit in memory and hence directly using lm is infeasible. The first step is to get the feature matrix into chunks on the hard drive:

```
#Using out-of-core technique
write.table(X[1:500,],file='Xchunk1.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(X[501:1000,],file='Xchunk2.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(X[1001:1500,],file='Xchunk3.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(X[1501:2000,],file='Xchunk4.txt',sep=',',row.names=F,col.names=names(Xdf))
write.table(Y[1:500],file='Ychunk1.txt',sep=',',row.names=F,col.names=F)
write.table(Y[501:1000],file='Ychunk2.txt',sep=',',row.names=F,col.names=F)
write.table(Y[1001:1500],file='Ychunk3.txt',sep=',',row.names=F,col.names=F)
write.table(Y[1501:2000],file='Ychunk4.txt',sep=',',row.names=F,col.names=F)
```

Now, we can read in each chunk individually and update the least squares solution using biglm. This will alleviate the need for reading all of X into memory at the same time. The below code is only partially completed. Complete the procedure in the natural way on the remaining chunks.

```
# Chunk 1
Xchunk = read.table(file='Xchunk1.txt',sep=',',header=T)
Ychunk = scan(file='Ychunk1.txt',sep=',')
form = as.formula(paste('Ychunk ~ -1 + ',paste(names(Xchunk),collapse=' + '),collapse=''))
out.biglm = biglm(formula = form,data=Xchunk)
hatBeta[1:5]
```

```
##       X1       X2       X3       X4       X5
## 24.99160 24.98724 24.96734 25.05268 25.04096
```

```
coef(out.biglm)[1:5]
```

```
##       X1       X2       X3       X4       X5
## 25.08815 24.48229 26.09057 26.84305 24.64633
```

```
# Chunk 2
Xchunk = read.table(file='Xchunk2.txt',sep=',',header=T)
Ychunk = scan(file='Ychunk2.txt',sep=',')
out.biglm = update(out.biglm,moredata=Xchunk)
hatBeta[1:5]
```

```
##       X1       X2       X3       X4       X5
## 24.99160 24.98724 24.96734 25.05268 25.04096
```

```
coef(out.biglm)[1:5]
```

```
##       X1       X2       X3       X4       X5
## 24.96665 25.00382 24.97887 25.01876 25.08741
```

```
# Chunk 3
Xchunk = read.table(file='Xchunk3.txt',sep=',',header=T)
Ychunk = scan(file='Ychunk3.txt',sep=',')
out.biglm = update(out.biglm,moredata=Xchunk)
hatBeta[1:5]
```

```
##       X1       X2       X3       X4       X5
## 24.99160 24.98724 24.96734 25.05268 25.04096
```

```
coef(out.biglm)[1:5]
```

```
##       X1       X2       X3       X4       X5
## 24.98077 25.00492 24.97527 25.02329 25.08050
```

```
## Can you figure out the final steps? Have we updated on all of the chunks?
```

```
print(hatBeta[1:5])
```

```
##       X1       X2       X3       X4       X5
## 24.99160 24.98724 24.96734 25.05268 25.04096
```

```
print(coef(out.biglm)[1:5])
```

```
##       X1       X2       X3       X4       X5
## 24.98077 25.00492 24.97527 25.02329 25.08050
```

Solution to part b:

Compare the first 5 entries in $\hat{\beta}$ formed by this method with the entries in (a)

Cross Validation and Parallelism

First, let's explore parallelism in R. Parallelism can be used to speed up computations by using multiple processors/CPUs at the same time. It is especially useful if we need to run a lot of computations at the same time that don't depend on eachother (this is often called "trivially" or "embarrassingly" parallel)

Note: The -1 in the "nCores" below is if using a GUI and hence needing system resources for other than processing. Usually, parallel processing is meant to be run in "batch" mode (R CMD BATCH myRscript.r &) though we won't be running in batch for this assignment.

```
nCores  = detectCores(all.tests = FALSE, logical = TRUE) - 1
cat('My work station has ',nCores,' cores \n')
```

```
## My work station has  3  cores
```

```
workers = makeCluster(nCores)
registerDoParallel(workers)

tmp  = rep(.1,nCores*10)
wait = function(tmp_i) Sys.sleep(tmp_i)
system.time(sapply(tmp,wait))#single processor
```

```
##    user  system elapsed
##   0.003   0.001   3.112
```

```
system.time(foreach(tmp_i = tmp) %dopar% {wait(tmp_i)})#multi-processor
```

```
##    user  system elapsed
##   0.123   0.003   1.176
```

Solution:

Comment on the relative sizes of nCores vs. the system.time with and without parallelism (note that if you happen to have a 2 core or fewer work station, you won't see any difference. If you have 2 cores, try and eliminate the '-1' from the code above)

Cross-Validation

Now, we want to implement our own CV and apply it the simulated data from the previous question.

```
K        = 10# Do K fold CV
folds    = sample(rep(1:K, length.out = n))
CVoutput = rep(0,K)

for(k in 1:K){
  validIndex  = which(folds == k)
  YhatValid   = X[validIndex,]%*%coef(lm(Y~X-1,subset=-validIndex))
  CVoutput[k] = mean( (YhatValid-Y[validIndex])**2  )
}
cat('CV estimate of the risk: ',mean(CVoutput),'\n')
```

```
## CV estimate of the risk:  1.369106
```

```
cat('Standard error of CV estimate of the risk: ',sd(CVoutput),'\n')
```

```
## Standard error of CV estimate of the risk:  0.1356844
```

Solution:

What could the standard error of CV estimate be used for?

Cross-validation in Parallel

CV runs well in parallel, so let's try that out. First, let's define a function:

```
cvF = function(k){
  validIndex  = which(folds == k)
  YhatValid   = X[validIndex,]%*%coef(lm(Y~X-1,subset=-validIndex))
  return(mean( (YhatValid-Y[validIndex])**2  ))
}
```

Now, we want to extend the above parallel code to CV

```
K        = 10# Do K fold CV
folds    = sample(rep(1:K, length.out = n))
startTime = proc.time()[3]
CVoutputParallel = foreach(k = 1:K) %dopar%{cvF(k)}
endTime = proc.time()[3]
cat('CV time: ',endTime-startTime,'\n')
```

```
## CV time:  3.372
```

```
cat('CV estimate of the risk: ',mean(CVoutput),'\n')
```

```
## CV estimate of the risk:  1.369106
```

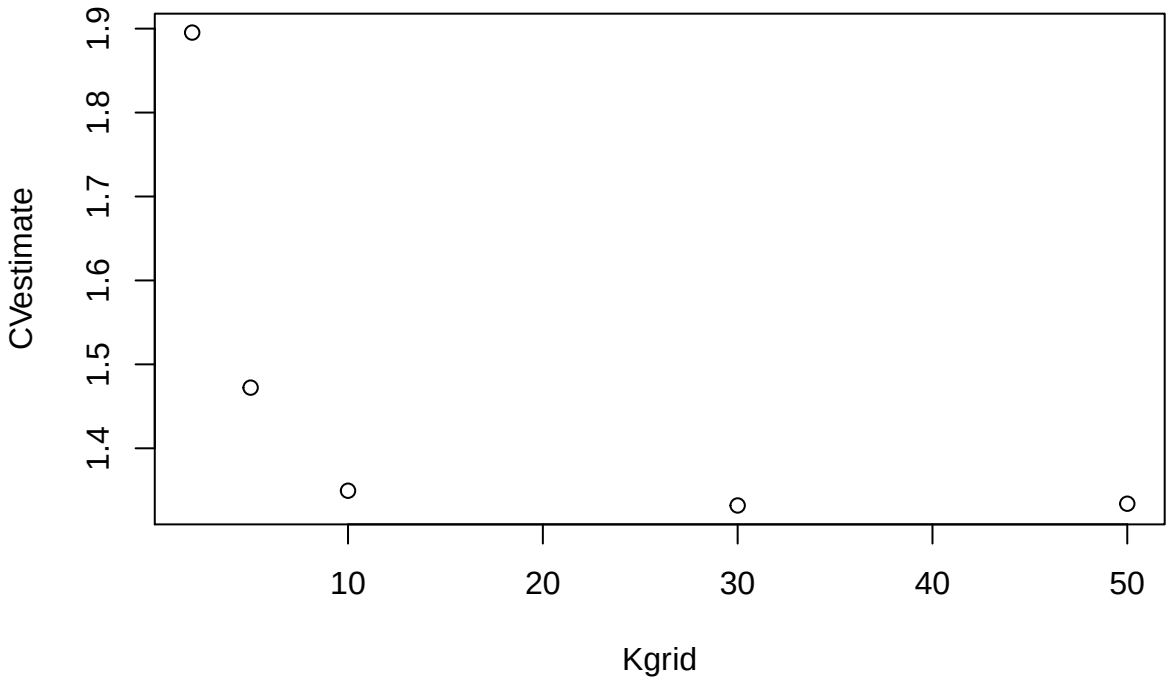
Solution:

The above code can be made to not run in parallel by replacing "dopar" with "do". Compare the time for CV with and without parallelism.

Solution

Also, make a plot of the CV estimate of the risk for a variety of K values (note that LOOCV would be of interest, but would take awhile on this problem):

```
Kgrid = c(2,5,10,30,50)
CVestimate = rep(0,length(Kgrid))
Kiter = 0
for(K in Kgrid){
  Kiter = Kiter + 1
  folds    = sample(rep(1:K, length.out = n))
  CVestimate[Kiter] = mean(unlist(foreach(k = 1:K) %dopar%{cvF(k)}))
}
plot(Kgrid,CVestimate)
```



Solution

Possibly tricky question: For which K is the K-Fold CV the best estimate of the risk?

Cross-validation in Caret

The "caret" package can do some commonly done machine learning tasks for you. In particular, create the folds:

```
 if(!require(caret)) install.packages("caret", dependencies = c("Depends", "Suggests"));require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
#See https://cran.r-project.org/web/packages/caret/vignettes/caret.pdf
K = 10
folds = createFolds(Y, k = K, list = TRUE, returnTrain = FALSE)
```

Solution

How does this 'folds' object differ from the 'folds' object defined for the cvF function? (be specific!)

Forward Selection

Using the same simulated X and Y generated in the previous problem, use forward selection and AIC to estimate the a constrained version of the least squares problem

```
if(!require(leaps)){install.packages('leaps',repos='http://cran.us.r-project.org');require(leaps)}
```

```
## Loading required package: leaps
```

Solution

Compare the first 5 entries of the coefficient vector estimated via forward selection to the first 5 entries of the coefficient vector found using lm (or biglm). How many nonzero entries do each have?