Parallel Computing for Random Forest

Introduction

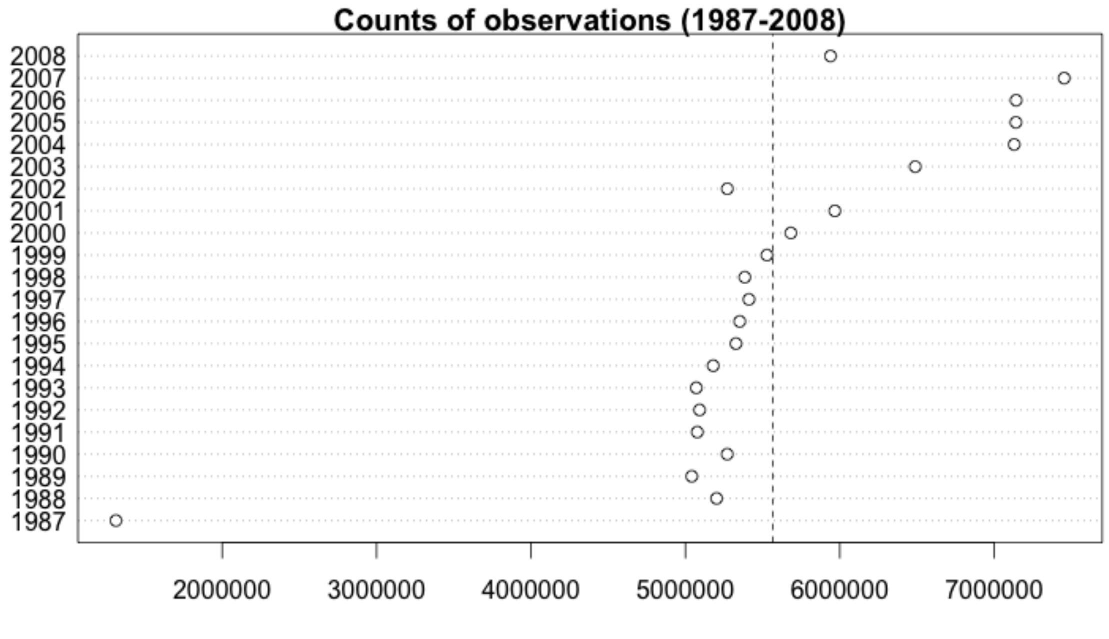
Parallel computing is a form of computation in which many calculations are carried out simultaneously. Running several computations at the exact same time could take advantage of multiple cores or CPUs on a single system and thus speed up the computations. However, regardless of the number of cores in our machine, R will only use 1 core on the default build. In this assignment, we explored to use “parallel” package to use the cores in my MacBook Air with 4 cores and 4G of memory.

Since Random Forests use bootstrapping and randomness, we can easily parallelizing the process and then use combine() function to obtain the entire forests. We develop a parRandomForest() to construct a committee of classification trees generated from bootstrap samples processed by the cores in our laptop. In other words, we fork the random forest computation to the two or three cores in our laptop and keep one core for mastering purpose. The parLapply() in parallel package significantly speedups computations for the random forests. The constraint of system memory in parallel computing will also be discussed in this essay.

Data Structure and Sampling

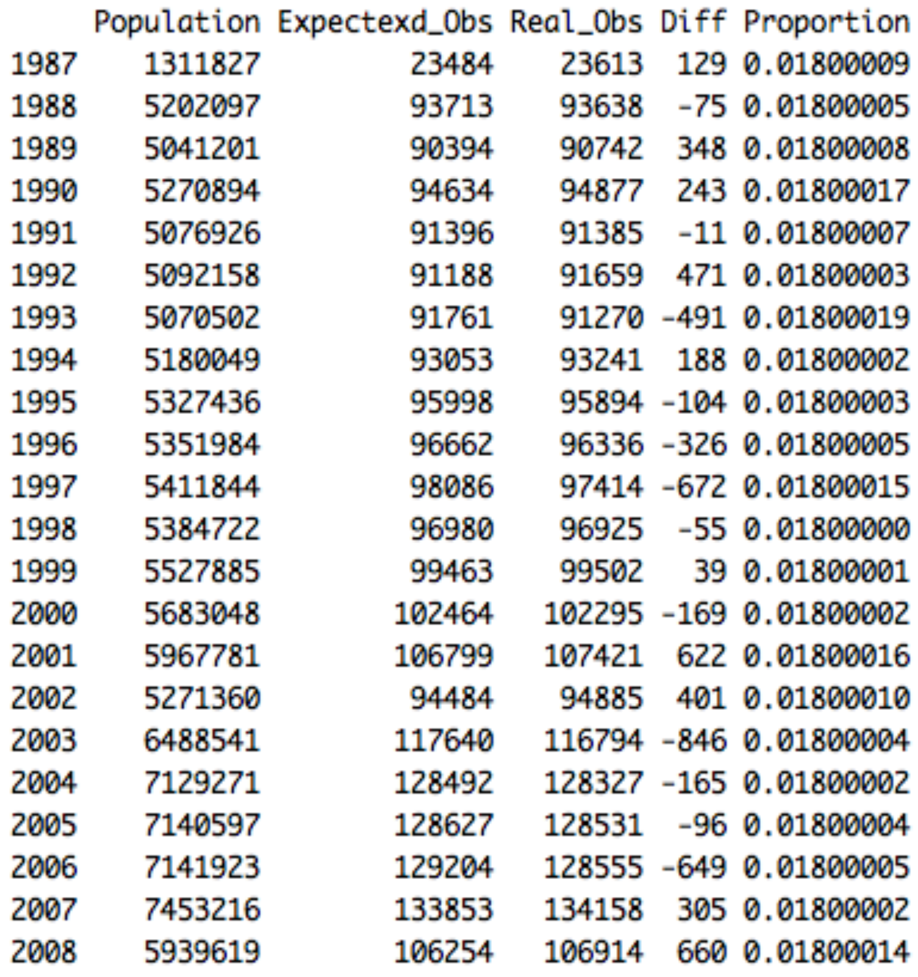
(I) Data Introduction

The data sets are constructed by 22 csv files (totally 12 Gigabytes), which contain domestic airline flights summaries as well as arrival and departure performance each year from 1987 to 2008. We have 29 columns in each data file for different years and have particular interests in predicting the 15th column for arrival delay (ARR_DELAY) using selected variables.



(II) Sampling Methods

To obtain a random sample of the observations, we could either take a equal samples of n = 100, 000 from each year or do the sampling proportionally based on the size of population in each year. For consistency of the data structure as the population, the latter approach is adopted in obtaining the final samples. We randomly take 1.8% of the observations from the population for each year. Following is the information about the original populations and the counts of observations obtained by sampling proportionally.



```
     Population Expectexd_Obs Real_Obs Diff Proportion
1987    1311827         23484    23613  129 0.01800009
1988    5202097         93713    93638  -75 0.01800005
1989    5041201         90394    90742  348 0.01800008
1990    5270894         94634    94877  243 0.01800017
1991    5076926         91396    91385  -11 0.01800007
1992    5092158         91188    91659  471 0.01800003
1993    5070502         91761    91270 -491 0.01800019
1994    5180049         93053    93241  188 0.01800002
1995    5327436         95998    95894 -104 0.01800003
1996    5351984         96662    96336 -326 0.01800005
1997    5411844         98086    97414 -672 0.01800015
1998    5384722         96980    96925  -55 0.01800000
1999    5527885         99463    99502   39 0.01800001
2000    5683048        102464   102295 -169 0.01800002
2001    5967781        106799   107421  622 0.01800016
2002    5271360         94484    94885  401 0.01800010
2003    6488541        117640   116794 -846 0.01800004
2004    7129271        128492   128327 -165 0.01800002
2005    7140597        128627   128531  -96 0.01800004
2006    7141923        129204   128555 -649 0.01800005
2007    7453216        133853   134158  305 0.01800002
2008    5939619        106254   106914  660 0.01800014
```

We have 23613 observations for 1987 and 106914 observations for 2008 after sampling. That gives us an average 100,000 observations for each year and a final sampling dataset with total 2.2 million observations for all years.

The sampling is conducted in two ways: Perl language and R language. A small trunk of Perl code is very efficient in dealing with the proportionally sampling. We generate a rand() each time when we read a new line. If the random number is less than 0.018, we put it aside and save it into another file in the future;

otherwise, we discuss the line. This is extraordinarily fast and effortless in sampling from the 12 GB data:

```
system(“cat [12]*.csv | perl -n -e 'print if (rand() < .018)' > airport_perl.out")
```

Another way is to stream the data into R and then grab lines randomly. We open a connection from Unix pipe and read blocks of data into R. Each time, we sample a proportion of random lines and use cat() to write it into a separate file. However, this is much slower.

```
rsample.cat<-function(B,prop,con){
  if(prop*B<1) stop ("sampling size is too small for each reading block")
  B=as.integer(B)
  index=1:B
  #data constructing
  bag=c()
  #reading blocks
  while(TRUE){
    txt=readLines(con,n=B)
    if (length(txt)==0)
      break
    #do random sampling
    sample.index<-sample(index,prop*B)
    bag<-rbind(bag,txt[sample.index])
    rm(txt,sample.index)
    gc(verbose=FALSE)
  }
  cat(bag,file="airport_R.out",sep="\n")
}
```

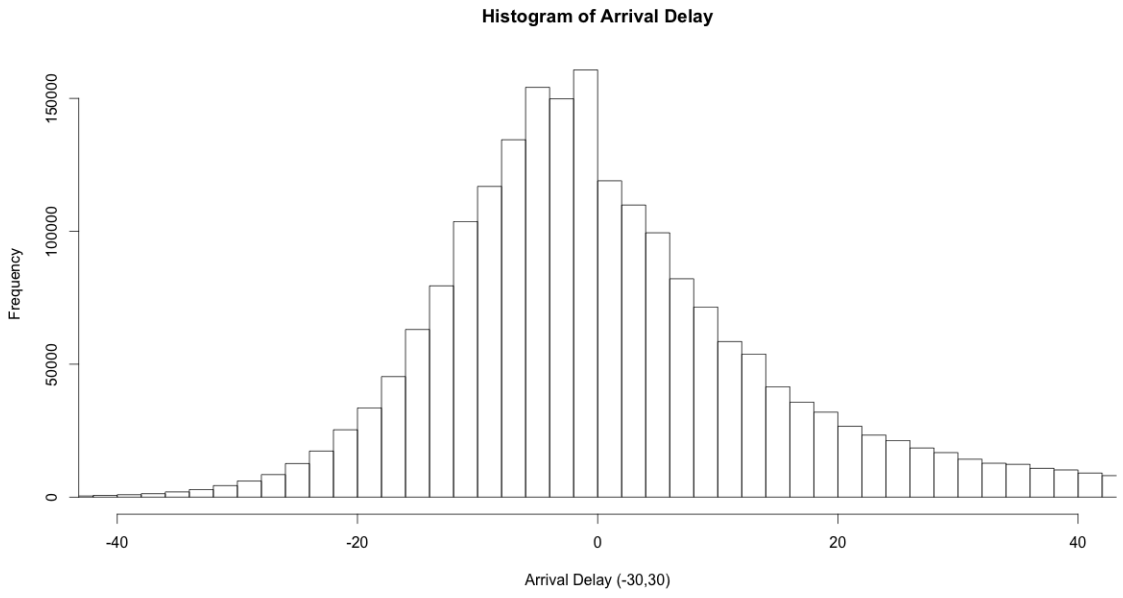
The small piece of Perl code took only 50 seconds to dram samples for all files from 1987 to 2008. However, the R code took 100 seconds to draw samples from the 2008.csv. Therefore, 40 mins are expected to finish the sampling from all files. That is, 48 times as much as the time needed for the Perl. If we are not allowed to conduct the sampling using other languages, we can parallelize the sampling process to all four cores in the macbook and the time may be shorten to 10mins.

However, parallelize the sampling to a cluster of machines or servers is not suggested due to the time needed for data transferring. We can obtain the subset samples on the master machine or from the database, compress them and then transfer to the machines needed for future computation. Also, we delete 1/3 of the variables which we are not interested in. The final subset rda file goes from 62.3M to 18.2M. If we process the data in R, we can find a significant memory saving in RAN around 500M while training the random forest. As a side note, the data has been randomly partitioned into two sets, with 70% for the training data and 30% for the test data.

Exploratory Analysis

(I) Arrival Delay

We have particular interests in predicting the arrival delay. Most of the flights are around 0 which falls into the “on time” category. We would like to construct a categorical responsive variable in our classification, specifically "Very Early","Early","On Time","Late" and "Very Late", using the combination of quantile() and cut() in R.



NAs are detected in the arrival delay. Some flights are either diverted or cancelled and the numbers of this abnormal cases match the numbers of NAs. We delete the NAs because we don’t need them for the prediction of arrival delay.

(2) Causes of Delay (excluded)

As for the predictors, we don’t want to include the last five causes of delay for two reasons: first, the causes of delay are recorded starting from 6/2003 and we lack information to train the our classifiers if we decide the include data before 2003; secondly, these five metrics are recorded in unit of minutes which is not meaningful for our prediction. We won’t know exactly how long the delay would be before the delay actually happened. Also, setting up categorical variables for these five causes would not help because we can’t specify the possibility of the events such as bad weather or security delay in our classifier.

Five causes of delay.

# CarrierDelay	Carrier Delay, in Minutes
# WeatherDelay	Weather Delay, in Minutes
# NASDelay	National Air System Delay, in Minutes
# SecurityDelay	Security Delay, in Minutes
# LateAircraftDelay	Late Aircraft Delay, in Minutes

(3) Timing Variables and Carriers
Month, DayOfWeek, CRSDepTime are included because they offers seasonal information. For example, we expect more visitors flying to Las Vegas on Friday at night. Flights are well schedule during this period and delay seldom happens. Different carriers have different operation philosophy and arrangement in different airports and thus serve as an important predictor.

(5) Origin and Destination
The R package with randomForest has a constraint on the levels of categorical variables. it can't handle factors with more than 32 levels. One possible solution is to create a new variable which groups airports into hubs or non-hub.

We can obtain the list of hubs in United States from Wikipedia and use regular expressions to import the list into R. We then grouping the origin and destination in to the binary category and observations fall into the following table.

```
#              Dest.Hub
# Origin.Hub      Hub     Non-Hub
# Hub          845316     567523
# Non-Hub      568767     223023
```

Classifier: Random Forest

In random forests, we grow trees from bootstrapped samples of the data, and obtain the predictions from a committee of the individual trees using a simple majority vote criterion. The procedures involve 1) Bootstrap samples; 2) At each split, randomly sample a subset of variables; 3) Grow trees and let them vote.

(I) Intuition behind the Voting
Voting in random forests have been shown to be very successful in improving the prediction accuracy given the difference of members in the committee. We explore the basic intuition behind majority vote.

Suppose we have 5 completely independent classifiers with a 70% accuracy for each. If we constitute a committee of these five independent classifiers, our accuracy will be improved to 83.69% accuracy. The aggregated probability are calculated based on the binominal theorem:

$$0.7^5 + 5 \times 0.7^4 \times 0.3^1 + 10 \times 0.7^3 \times 0.3^2$$

(II) Optimal Number of Trees for Voting
Random forest uses boostraping (picking a sub-sample with replacement rather than use all of the observations) and randomness (picking a subset of variables rather than all of the predictors). To decide the optimal number of trees, we have to consider the numbers of predictors we pick up and the number observations.

We have 8 predictors which are relevant small to the hundreds of trees growed. Features of some variables could be missed in one tree but should not be missed in others given enough trials of bootstraping. Due to the constraint of computation power, we only try 300 trees in this assignment

(III) Parallel Computing
Running several computations at the exact same time could take advantage of multiple cores or CPUs on a single system. However, R will only use 1 core on the default build regardless of the number of cores in our machine. Therefore, we can use a R package called "parallel" to make folks on our machine who share the same objects in the workspace but process tasks simultaneously.

Since Random Forests use bootstrapping and randomness, we can easily parallelizing the process and then use combine() function to obtain the entire forests. We develop a parRandomForest() to construct a committee of classification trees generated from bootstrap samples processed by the cores in our laptop. In other words, we fork the random forest computation to other cores in our laptop and keep one core for mastering purpose.

```
parRandomForest <- function(xx, ..., ntree = 300, mc = (detectCores()-1))
{ cl <- makeCluster(mc,type = "FORK")
  clusterEvalQ(cl, library(randomForest))
  rfwrap <- function(ntree, xx, ...) randomForest(x=xx, ntree=ntree, ...)
  rfpar <- parLapply(cl, rep(ceiling(ntree/mc), mc), rfwrap, xx=xx,...)
  stopCluster(cl)
  do.call(combine, rfpar)
}
```

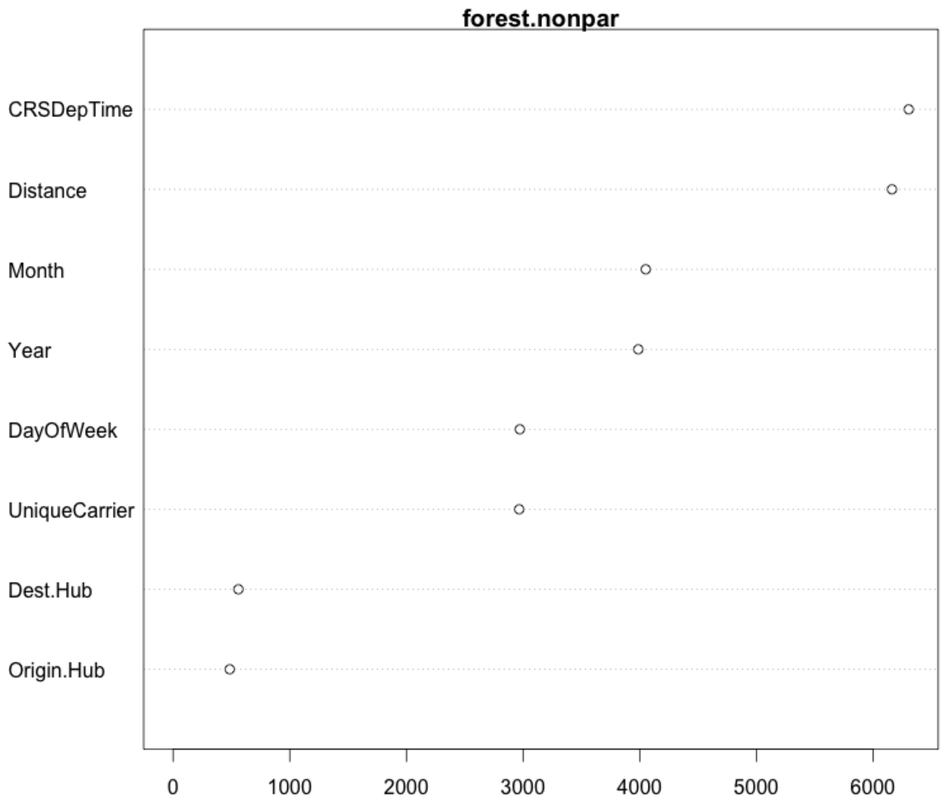
Results shows that the the parLapply() in parallel package significantly speedups computations for the random forests. **In the case of growing 99 trees, our elapsed time is shorten from 305.388 seconds to 192.898 seconds. In the case of growing 300 trees, we have improvement from 1842.200 seconds to 1011.312 seconds.** More trees and larger bootstrapping sample size would increase the time in feeding classifiers due to the memory constraints of the personal laptop. We will discuss it in the next session.

(IV) Memory Constraints and Bootstrapping Size
Due to the limitation in the system memory, bootstrapping size has to be limited to successfully grow trees in R. I am using a quad-core Macbook Air with 4G of system memory. Limiting bootstrapping size to 50000 help me to limit the memory consumption around 1G for each core and thus two cores could be utilized relevant efficiently without waiting for samples for a long time. If the number of observations are large, but the number of trees is too small, then some observations will be predicted only one or even no any times which may give us a bad prediction. We may want to grow more trees to compensate that.

(V) Randomization of Variables
In the package "randomForest", the default for "mtry" option in is sqrt(p), which specify the number of variables randomly sampled as candidates at each split. If we set "mtry=length(predictors)", we will have bagging (bootstrapping aggregation) because we are not hiding any variables. The performance and computation speed should be in between the two cases above.

We can expect that the bagging have worse performance because each trees should look very similar when we don't randomize the variables. Hiding half of the variables increase the performance but has less combinations than the default settings for sqrt(p).

(VI) Importance of Variables
To tell what variables are important in our classification, we draw a variable importance plots which show the mean decrease accuracy. The decrease in accuracy would tell us how important is the variable if we get rid of it. In every tree grown in the forest, we set aside the OOB observations and count the number of votes resulting in correct class. We then randomly permute the values of variables in the OOB observations and run the trees again. The accuracy normally decreases as we are using the cutted version of trees.



CRSDepTime and Distance seems to be the most important predictors in our random forest classifier. Other timing variables and carrier are also important factors in the arrival delay. Notice that the importance of destination and origin are far behind other predictors. This is an alert that splitting the airports into binary category is too simple. We should explore other list of hubs with more categories such as large hub, medium hub, small hub, etc. Origin and destination airports offer far more information than that: state, visitor flows, facility and etc. Dividing them into multiple categories may improve the results.