---

title: "Lab 7"

author: "Feiyi Ma"

output: pdf_document

date: "11:59PM April 22, 2021"

We will get some experience with speeding up R code using C++ via the `Rcpp` package.

First, clear the workspace and load the `Rcpp` package.

```{r}
pacman::p_load(Rcpp)
```

Create a variable `n` to be 10 and a vaiable `Nvec` to be 100 initially. Create a random vector via `rnorm` `Nvec` times and load it into a `Nvec` x `n` dimensional matrix.

```{r}
n=10

Nvec =100

X = matrix(data=rnorm(Nvec*n),nrow= Nvec)

head(X)
```

Write a function `all_angles` that measures the angle between each of the pairs of vectors. You should measure the vector on a scale of 0 to 180 degrees with negative angles coerced to be positive.

```{r}
```

```
angles = function(u,v){
  (acos(sum(u*v)/sqrt(sum(u^2)*sum(v^2))))*(180/pi)
}
all_angles = function(x){
  A = matrix(NA, nrow= nrow(X), ncol= nrow(X))
  for(i in 1:(nrow(X)-1)){
    for(j in (i+1):nrow(X)){
      A[i,j]=angles(X[i,], X[j,])
    }
  }
  A
}
all_angles(X)
```

Plot the density of these angles.

```{r}
pacman::p_load(ggplot2)
ggplot(data.frame(angles =c(all_angles(X)))) + aes(x=angles)+geom_density()
#two matrix othogonal to each other (most of time)
```

Write an Rcpp function `all_angles_cpp` that does the same thing. Use an IDE if ou want, but write it below in-line.

```{r}
cppFunction(
  "
```

```
NumericMatrix all_angles_cpp(NumericMatrix X) {
  int n = X.nrow();
  int p = X.ncol();
  NumericMatrix A(n, n);
  std::fill(A.begin(), A.end(), NA_REAL);
  for (int i_1 = 0; i_1 < (n - 1); i_1++){
    for (int i_2 = i_1 + 1; i_2 < n; i_2++){
      double sum_sqd_u = 0;
      double sum_sqd_v = 0;
      double sum_u_v = 0;
      for (int j = 0; j < p; j++){
        sum_sqd_u +=pow(X(i_1, j), 2);
        sum_sqd_v +=pow(X(i_2, j), 2);
        sum_u_v += X(i_1, j) * X(i_2, j);
      }
      A(i_1, i_2) = (acos(sum_u_v/sqrt(sum_sqd_u*sum_sqd_v)))*(180/M_PI);
    }
  }
  return A;
}
  "
)
all_angles_cpp(X)
```

Test the time difference between these functions for `n = 1000` and `Nvec = 100, 500, 1000, 5000`. Store the results in a matrix.

```{r}
```

```
pacman::p_load(microbenchmark)

n=1000

Nvec =100

X = matrix(data=rnorm(Nvec*n),nrow= Nvec)

microbenchmark(all_angles(X),all_angles_cpp(X),times=10)
```

Plot the divergence of performance (in log seconds) over nvec using a line geometry. Use two different colors for the R and CPP functions. Make sure there's a color legend on your plot.

```{r}
pacman::p_load(microbenchmark)

Nvecs = c(100,500,1000,5000)

res = data.frame(
  Nvec = Nvecs,
  time_for_base_R = NA,
  time_for_cpp = NA
)

for(i in 1:length(Nvecs)){
  X = matrix(data = rnorm(Nvecs[i]*n), nrow= Nvecs[i])
  res$time_for_base_R[i] = all_angles(X)
  res$time_for_cpp[i] = all_angles_cpp(X)
}

ggplot(res) +
  geom_line(aes(x = Nvec, y = time_for_base_R),col="red")+
  geom_line(aes(x = Nvec, y = time_for_cpp),col="yellow")

```

Let `Nvec = 10000` and vary `n` to be 10, 100, 1000. Plot the density of angles for all three values of `n` on one plot using color to signify `n`. Make sure you have a color legend. This is not easy.

```{r}
Nvecs = 10000
n= c(10,100,1000)
```

Write an R function `nth_fibonnaci` that finds the nth Fibonnaci number via recursion but allows you to specify the starting number. For instance, if the sequency started at 1, you get the familiar 1, 1, 2, 3, 5, etc. But if it started at 0.01, you would get 0.01, 0.01, 0.02, 0.03, 0.05, etc.

```{r}

```

Write an Rcpp function `nth_fibonnaci_cpp` that does the same thing. Use an IDE if ou want, but write it below in-line.

```{r}
cppFunction(
  "
nth_fibonnaci

"
)
```

Time the difference in these functions for n = 100, 200, ..., 1500 while starting the sequence at the smallest possible floating point value in R. Store the results in a matrix.

```{r}

#TO-DO

```

Plot the divergence of performance (in log seconds) over n using a line geometry. Use two different colors for the R and CPP functions. Make sure there's a color legend on your plot.

```{r}

#TO-DO

```

Data Wrangling / Munging / Carpentry

Throughout this assignment you can use either the `tidyverse` package suite or `data.table` to answer but not base R. You can mix `data.table` with `magrittr` piping if you wish but don't go back and forth between `tbl_df`'s and `data.table` objects.

```{r}

pacman::p_load(tidyverse, magrittr, data.table)

```

Load the `storms` dataset from the `dplyr` package and investigate it using `str` and `summary` and `head`. Which two columns should be converted to type factor? Do so below.

```{r}

data(storms)

str(storms)

summary(storms)

head(storms)

```

Reorder the columns so name is first, status is second, category is third and the rest are the same.

```{r}
storms%>%
  select(name,status,category,everything())
```

Find a subset of the data of storms only in the 1970's.

```{r}
storms %>%
  filter(year>=1970 & year <=1979)
```

Find a subset of the data of storm observations only with category 4 and above and wind speed 100MPH and above.

```{r}
storms %>%
  filter(category>=4 & wind >=100)
```

Create a new feature `wind_speed_per_unit_pressure`.

```{r}
storms%>%
 mutate(wind_speed_per_unit_pressure = wind/pressure)
```

Create a new feature: `average_diameter` which averages the two diameter metrics. If one is missing, then use the value of the one that is present. If both are missing, leave missing.

```{r}
storms%>%
  rowwise()%>%
  arrange(desc(year)) %>%
 mutate(average_diameter = mean(c(ts_diameter,hu_diameter),na.rm=TRUE))
#mean function take the whole function, not row by row

```

For each storm, summarize the maximum wind speed. "Summarize" means create a new dataframe with only the summary metrics you care about.

```{r}
storms%>%
  group_by(name) %>%
  summarise(max_wind_speed = max(wind, na.rm=TRUE))
```

Order your dataset by maximum wind speed storm but within the rows of storm show the observations in time order from early to late.

```{r}
storms%>%
  group_by(name) %>%
  mutate(max_wind_by_storm = max(wind, na.rm=TRUE))%>%
```

```
  select(name,max_wind_by_storm,everything()) %>%
  arrange(desc(max_wind_by_storm),year, month, day, hour)
```

Find the strongest storm by wind speed per year.

```{r}
storms%>%
  group_by(year) %>%
  arrange(year,desc(wind))%>%
  slice(1) %>%
  select(name,year,wind)
```

For each named storm, find its maximum category, wind speed, pressure and diameters. Do not allow the max to be NA (unless all the measurements for that storm were NA).

```{r}
storms%>%
  group_by(name) %>%
  arrange(desc(category))%>%
  slice(1) %>%
  arrange(desc(wind))%>%
  slice(1) %>%
  arrange(desc(pressure))%>%
  slice(1) %>%
  arrange(desc(ts_diameter))%>%
```

```
  slice(1) %>%
  arrange(desc(hu_diameter))%>%
  slice(1) %>%
  select(name,category,wind,pressure,ts_diameter,hu_diameter)
```

For each year in the dataset, tally the number of storms. "Tally" is a fancy word for "count the number of". Plot the number of storms by year. Any pattern?

```{r}
storms2 = storms%>%
  group_by(year) %>%
summarize(num_storms = n_distinct(name))

ggplot(storms2) +
  aes(x=factor(year), y=storms2$num_storms) +
  geom_point()
```

For each year in the dataset, tally the storms by category.

```{r}
storms%>%
  group_by(year) %>%
  arrange(sort(category))%>%
  arrange(sort(year))%>%
```

```
  select(name,category,year)
```

For each year in the dataset, find the maximum wind speed per status level.

```{r}
storms%>%
  group_by(year) %>%
  arrange(sort(status),desc(wind))%>%
    slice(1)%>%
  arrange(year)%>%
  select(name,status,year,wind)
```

For each storm, summarize its average location in latitude / longitude coordinates.

```{r}
storms%>%
  group_by(name) %>%
  summarise(ave_loca_lat = ave(lat, na.rm=TRUE),ave_loca_long = ave(long, na.rm=TRUE) )
```

For each storm, summarize its duration in number of hours (to the nearest 6hr increment).

```{r}
storms%>%
  group_by(name) %>%
  summarise(duration = add(hour) )
```

For storm in a category, create a variable `storm_number` that enumerates the storms 1, 2, ... (in date order).

```{r}
storms%>%
  rowwise()%>%
  group_by(category) %>%
 mutate(storm_number = mean(c(ts_diameter,hu_diameter),na.rm=TRUE))
```

Convert year, month, day, hour into the variable `timestamp` using the `lubridate` package. Although the new package `clock` just came out, `lubridate` still seems to be standard. Next year I'll probably switch the class to be using `clock`.

```{r}
pacman::p_load(lubridate)
storms%>%
  mutate(timestamp = ymd_h(paste(year, month, day,hour,sep="-")))
```

Using the `lubridate` package, create new variables `day_of_week` which is a factor with levels "Sunday", "Monday", ... "Saturday" and `week_of_year` which is integer 1, 2, ..., 52.

```{r}
storms3 = storms%>%
  mutate(day_of_week = wday(paste(year, month, day,hour,sep="-"),label = TRUE))%>%
  mutate(week_of_year = week(ymd(paste(year, month, day))))
  storms3
```

For each storm, summarize the day in which is started in the following format "Friday, June 27, 1975".

```{r}
storms4=storms3%>%
  mutate(day_information = mdy(paste(month, day,year,sep = "-")))%>%
  relocate(day_information,.before = name)
 storms4%>%
   relocate(day_of_week,.before = day_information)

```

Create a new factor variable `decile_windspeed` by binning wind speed into 10 bins.

```{r}
decile_windspeed
```

Create a new data frame `serious_storms` which are category 3 and above hurricanes.

```{r}

storms$category = ifelse(storms$category>=3,"serious_storms", storms$category)
storms$category = as.factor(storms$category)
storms%>%
  select(name, category,year)

```

In `serious_storms`, merge the variables lat and long together into `lat_long` with values `lat / long` as a string.

```{r}
storms$category = ifelse(storms$category>=3,"serious_storms", storms$category)
storms$category = as.factor(storms$category)
```

Let's return now to the original storms data frame. For each category, find the average wind speed, pressure and diameters (do not count the NA's in your averaging).

```{r}
storms%>%
  group_by(category)%>%
  summarise(ave_wind_speed = ave(wind, na.rm=TRUE),ave_pressure = ave(pressure, na.rm=TRUE) ,ave_diameter = ave(diamonds,na.rm=TRUE))
```

For each named storm, find its maximum category, wind speed, pressure and diameters (do not allow the max to be NA) and the number of readings (i.e. observations).

```{r}
storms%>%
  group_by(name)%>%
  summarise(max_category = max(category, na.rm=TRUE),max_wind_speed = max(wind, na.rm=TRUE),max_pressure = max(pressure, na.rm=TRUE) ,max_diameter = max(diamonds,na.rm=TRUE))
```

Calculate the distance from each storm observation to Miami in a new variable `distance_to_miami`. This is very challenging. You will need a function that computes distances from two sets of latitude / longitude coordinates.

```{r}
MIAMI_LAT_LONG_COORDS = c(1,25.7617, -80.1918)
storm_location= storms%>%
  group_by(name)%>%
  slice(1)%>%
  select(name,lat,long)
 storm_place = rbind(data=storm_location)
storm_place[,2]=storm_place[,2]+ 25.7617
storm_place[,3]=storm_place[,3] - 80.1918
distance_to_miami=storm_place
distance_to_miami
```

For each storm observation, use the function from the previous question to calculate the distance it moved since the previous observation.

```{r}
distance_to_miami%>%
  mutate(distance_moved = add(sqrt(lat*lat + long*long)))
```

For each storm, find the total distance it moved over its observations and its total displacement. "Distance" is a scalar quantity that refers to "how much ground an object has covered" during its motion. "Displacement" is a vector quantity that refers to "how far out of place an object is"; it is the object's overall change in position.

```{r}
```

#TO-DO

```

For each storm observation, calculate the average speed the storm moved in location.

```{r}

#TO-DO

```

For each storm, calculate its average ground speed (how fast its eye is moving which is different from windspeed around the eye).

```{r}

#TO-DO

```

Is there a relationship between average ground speed and maximum category attained? Use a dataframe summary (not a regression).

```{r}

#TO-DO

```

Now we want to transition to building real design matrices for prediction. This is more in tune with what happens in the real world. Large data dump and you convert it into X and y how you see fit.

Suppose we wish to predict the following: given the first three readings of a storm, can you predict its maximum wind speed? Identify the `y` and identify which features you need $x_1, ... x_p$ and build that matrix with `dplyr` functions. This is not easy, but it is what it's all about. Feel free to "featurize" as creatively as you would like. You aren't going to overfit if you only build a few features relative to the total 198 storms.
```

```{r}
#TO-DO
```

Fit your model. Validate it.

```{r}
#TO-DO
```

Assess your level of success at this endeavor.

#TO-DO

The Forward Stepwise Procedure for Probability Estimation Models

Set a seed and load the `adult` dataset and remove missingness and randomize the order.

```{r}
set.seed(1)
pacman::p_load_gh("coatless/ucidata")
data(adult)
adult = na.omit(adult)
adult = adult[sample(1 : nrow(adult)), ]
```

Copy from the previous lab all cleanups you did to this dataset.

```{r}
K = 5
test_prop = 1 / K
train_indices = sample(1 : nrow(adult), round((1 - test_prop) * nrow(adult)))
adult_train = adult[train_indices, ]
y_train = adult_train$income
X_train = adult_train
X_train$income = NULL
test_indices = setdiff(1 : nrow(adult), train_indices)
adult_test = adult[test_indices, ]
y_test = adult_test$income
X_test = adult_test
X_test$income = NULL
```

We will be doing model selection. We will split the dataset into 3 distinct subsets. Set the size of our splits here. For simplicitiy, all three splits will be identically sized. We are making it small so the stepwise algorithm can compute quickly. If you have a faster machine, feel free to increase this.

```{r}
Nsplitsize = 1000
```

Now create the following variables: `Xtrain`, `ytrain`, `Xselect`, `yselect`, `Xtest`, `ytest` with `Nsplitsize` observations. Binarize the y values.

```{r}
Xtrain = adult[1 : Nsplitsize, ]
```

```
Xtrain$income = NULL
ytrain = ifelse(adult[1 : Nsplitsize, "income"] == ">50K", 1, 0)
Xselect = adult[(Nsplitsize + 1) : (2 * Nsplitsize), ]
Xselect$income = NULL
yselect = ifelse(adult[(Nsplitsize + 1) : (2 * Nsplitsize), "income"] ==">50K", 1, 0)
Xtest = adult[(2 * Nsplitsize + 1) : (3 * Nsplitsize), ]
Xtest$income = NULL
ytest = ifelse(adult[(2 * Nsplitsize + 1) : (3 * Nsplitsize), "income"] == ">50K", 1, 0)
```

Fit a vanilla logistic regression on the training set.

```{r}
logistic_mod = glm(ytrain ~ ., Xtrain, family = "binomial")
```

and report the log scoring rule, the Brier scoring rule.

```{r}
#TO-DO
```

We will be doing model selection using a basis of linear features consisting of all first-order interactions of the 14 raw features (this will include square terms as squares are interactions with oneself).

Create a model matrix from the training data containing all these features. Make sure it has an intercept column too (the one vector is usually an important feature). Cast it as a data frame so we can use it more easily for modeling later on. We're going to need those model matrices (as data frames) for both the select and test sets. So make them here too (copy-paste). Make sure their dimensions are sensible.

```{r}
#TO-DO
dim(Xmm_train)
dim(Xmm_select)
dim(Xmm_test)
```

Write code that will fit a model stepwise. You can refer to the chunk in the practice lecture. Use the negative Brier score to do the selection. The negative of the Brier score is always positive and lower means better making this metric kind of like s_e so the picture will be the same as the canonical U-shape for oos performance.

Run the code and hit "stop" when you begin to the see the Brier score degrade appreciably oos. Be patient as it will wobble.

```{r}
pacman::p_load(Matrix)
p_plus_one = ncol(Xmm_train)
predictor_by_iteration = c() #keep a growing list of predictors by iteration
in_sample_brier_by_iteration = c() #keep a growing list of briers by iteration
oos_brier_by_iteration = c() #keep a growing list of briers by iteration
i = 1
repeat {
  #TO-DO
  #wrap glm and predict calls with use suppressWarnings() so the console is clean during run

  if (i > Nsplitsize || i > p_plus_one){
    break
  }
}
```

```

Plot the in-sample and oos (select set) Brier score by $p$. Does this look like what's expected?

```{r}
#TO-DO
```
```