title: "Lab 8"

author: "Asher KAtz"

output: pdf_document

More Bagging Practice

Write a function `random_bagged_ols` which takes as its arguments `X` and `y` with further arguments `num_ols_models` defaulted to 100 and `mtry` defaulted to NULL which then gets set within the function to be 50% of available features at random. This argument builds an OLS on a bootstrap sample of the data and uses only `mtry < p` of the available features. The function then returns all the `lm` models as a list with size `num_ols_models`.

```{r}
#TO-DO
#####
randomForest(data.frame(x = x_train), y = y_train, ntree = 1, replace = FALSE, sampsize = n, nodesize = nodeSize)
```

For masters students: show that bagged ols does better than just ols out of sample. The diamonds data is likely a good data set to demo this on. You may have to add a few interactions.

More RF Practice

Load up the Boston Housing Data and separate into `X` and `y`.

```{r}
#TO-DO
pacman::p_load(MASS)
data(Boston)
boston = data.table(Boston)
y = boston[, c('medv') ]
X = boston[, !c('medv')]
y
```

X

```

Similar to lab 1, write a function that takes a matrix and punches holes (i.e. sets entries equal to `NA`) randomly with an argument `prob_missing`.

```{r}
#TO-DO
hole_puncher = function(mat, prob_missing){

  for ( i in 1:(nrow(mat)-1) ){
    for ( j in 1:(ncol(mat)-1) ){
      if ( runif(1) < prob_missing ){
  # runif(1) creates a random probability between 1 and 0. 0.3 is 30%
      mat[i,j] = NA
      }
    }
  }
return(mat)
}
n = 50
# 2500 entries, 1250 of them 0's, 625 of them 1's, 625 of them 2's
# rep is repeat(a value, x many times)
R = matrix( sample( c( rep(0, 1250), rep(1, 625), rep(2, 625) ) ), nrow = n, ncol = n )
hole_puncher(R, 0.3)
```

Create a matrix `Xmiss` which is `X` but has missingness with probability of 10%.

```{r}
#TO-DO
Xm = as.matrix(X)
```
```

```
head(Xm)

ncol(Xm)

nrow(Xm)

Xmiss = hole_puncher(Xm, 0.1)

head(Xmiss)
```

Use a random forest modeling procedure to iteratively fill in the `NA`'s by predicting each feature of X using every other feature of X. You need to start by filling in the holes to use RF. So fill them in with the average of the feature.

```{r}
#####

Ximps = list()

t = 1

repeat {

  for (j in 1 : p){

    Ximps[[t]][, j] =

  }

  t = t + 1

  stop = randomForest(data.frame(x = Xmiss), y = y)#stop condition if Ximps[[t]] - Ximps[[t - 1]] is close together

  if (stop){

    break

  }

}
```

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}
#TO-DO
data(storms)
str(storms)
summary(storms)
head(storms)
storms = data.table(storms) # teh storm is a year, a month, a day, an hour, a current position, catagory, a windspeed, nor pressure nor a force diameter
storms[, status:= factor(status)] # it is blank_storm NAME
storms[, name:= factor(name)]
storms
```

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

```{r}
#TO-DO
setcolorder(storms, c("name", "status", "category")) # name first then status...
storms
```

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

```{r}
#TO-DO
storms[year>= 1970 & year <= 1979] # no commans, simple range of the data
```

```

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

```{r}
#TO-DO
storms[category>= 4 & wind >= 100]
```

Create a new feature `wind_speed_per_unit_pressure`.

```{r}
#TO-DO
storms[,wind_speed_per_unit_pressure := wind/pressure ] # new features can be declared in the second comma
storms
```

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}
#TO-DO
pacman::p_load(ggplot2, data.table)

diamonds = data.table(diamonds)

#get pmean function from someone who wrote it already (this function should be standard in R)
source("https://raw.githubusercontent.com/tanaylab/tgutil/master/R/utils.R")
diamonds[, avg_dim := pmean(x, y, z, na.rm = TRUE)]
diamonds
```
```

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

```{r}

#TO-DO

storms[, .(max_wind_speed = max(wind)), by = name] # Show me the name of the max wind speed storms ... this only the max of each storm, we need to order it as well \/

distinct(storms[, max_wind_by_year := max(wind), by = year][wind == max_wind_by_year, .(year, name, wind)])[, .(year, name)]

#distinct max wind of year, .() means subset, inlcude only year, name, wind and sorted by year. Now return the name of the max of the year
```

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}

#TO-DO

#order from the storm with the highest windspeed, and each storm should be in it's time order.

storms[ , storm_max_wind_speed := max(wind), by = name] # Order storm max wind speed column by name

storms

setcolorder(storms, c("storm_max_wind_speed", "name")) # order the row differently... 1st col is storm max windspeed, 2nd is name

storms[order(-storm_max_wind_speed, year, month, day, hour)] # have these 4 col in descending order

```

Find the strongest storm by wind speed per year.

```{r}

#TO-DO

storms[order(-wind), .SD[1,name], by = year][order(year)] # for every year going up, sort every windspeed going down, by year ascending... and grab the max wind of the year
```

```
#storms[, max_wind_speed := max(wind), by = year]

#storms[order(year, -max_wind_speed)]
```

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}
#TO-DO
storms[,.(
        max_catagory = max(category, na.rm = TRUE),
        max_wind_speed = max(wind, na.rm = TRUE),
        max_pressure = max(pressure, na.rm = TRUE),
        max_diameter = max(hurricane_force_diameter, na.rm = TRUE)
        ),
      by = name] # not this :=, because we are not changing the underlying data

```

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}
num_storms_by_year = storms[,.(num_storms = uniqueN(name)), by = year]
num_storms_by_year[,year := factor(year)]
num_storms_by_year
pacman::p_load(ggplot2)
ggplot(num_storms_by_year)+
  geom_point(aes(x = year, y = num_storms))
```

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

```{r}

#TO-DO

storms[order(year, category),,] # display year and stormcat in ascending

storms[order(year, category),.(num_storms = uniqueN(name)), by = .(year, category)] #  ... and show me a SS which is num storms... by year ... by catagory


```


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


```{r}

#TO-DO

storms[order(wind), .SD[.N, wind], by = .(year, status)][order(year, status)]


```


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


```{r}

#TO-DO

# use the pmean fucntion, similiar to the other problem on slack

#####
```


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


```{r}

#TO-DO

storms[order(name), .(duration_hr = .N*6 -6) , by = name]

```

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

```{r}
#TO-DO
#####
storms[, .(storm_number = uniqueN(name) )]
```

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}
#TO-DO
#?lubridate
#####

pacman::p_load(lubridate, data.table)
storms = data.table(storms)
storms[, timestamp := ymd_h(paste(year,month,day, hour, sep = "-"))]
storms
```

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}
#TO-DO
#####
storms[, dow := wday(timestamp)]
storms
```

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

```{r}
#TO-DO
#####
storms[, timestamp := ymd_h(paste(dow,day, hour, sp = "-"))]
```

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

```{r}
#TO-DO
storms[, decile_windspeed := factor(ntile(wind, 10)), ]
```

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

```{r}
#TO-DO
serious_storms = storms[category >= 3]
serious_storms
```

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

```{r}
#TO-DO
#create a new variable and use the paste function
#####
```

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}

#TO-DO

#skip cause dublicate

#gave the answr on slack essentially

```

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}

#TO-DO

#skip cause dublicate

```

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(25.7617, -80.1918)

# #TO-DO

 dist = function(lats1, longs1, lats2, longs2){

lats3 = lats2 - lats1

longs3 = longs2 - longs1

return( sqrt( (lats3)^2 + (longs3)^2 ))

}

# dist_miami = function(lat, long){

#   n = length(lats)

#   dist(lats, longs, rep(MIAMI_LAT_LONG_COORDS[1],n), rep(MIAMI_LAT_LONG_COORDS[2], n))

#   #go to the internet, find the answer, post on slack

# }

#

# storms[,distance_to_miami := dist_miami(lat, long),]
```

```
#
#
custom_hav_dist = function(lat1, lon1, lat2, lon2) {
  R <- 3958.756
  Radian_factor <- 0.0174533
  lat_1 <- (90-lat1)*Radian_factor
  lat_2 <- (90-lat2)*Radian_factor
  diff_long <-(lon1-lon2)*Radian_factor

  distance_in_miles <- 6371*acos((cos(lat_1)*cos(lat_2))+
                (sin(lat_1)*sin(lat_2)*cos(diff_long)))
  rm(lat1, lon1, lat2, lon2)
  return(distance_in_miles)
}
storms[, distance_to_maimi := custom_hav_dist(lat, long, previous_latitude, previous_longitude)]
storms

#PS: To calculate distances in miles, substitute R in function (6371) with 3958.756 (and for nautical miles, use 3440.065).
```

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

```{r}
#TO-DO
storms[, origination_year := year]
storms[name == "Zeta" & year == 2006, origination_year := 2005]
storms[, previous_latitude := lag(lat), by =.(name, origination_year)]
storms[, previous_longitude := lag(long), by =.(name, origination_year)]
storms[, distance_since_previous_observation := custom_hav_dist(lat, long, previous_latitude, previous_longitude)]
storms
```

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

storms[, first_latitude := lag(lat[2]), by =.(name, origination_year)]
storms[, first_longitude := lag(long[2]), by =.(name, origination_year)]
storms[, last_latitude := lag(lat[.N]), by =.(name, origination_year)]
storms[, last_longitude := lag(long[.N]), by =.(name, origination_year)]
storms[, total_displacement := custom_hav_dist(first_latitude, first_longitude, last_latitude, last_longitude)]
storms
storms[, distance_since_previous_observation := custom_hav_dist(lat, long, previous_latitude, previous_longitude)]
storms
```
```

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}
#####
```

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
#the y is going to be max wind speed
pacman::p_load(data.table(), tdiyverse)
storms = data.table(storms)

last_period = 10

storms[, max_wind_speed := max(wind), by = .(name, year)]
first3 = storms[,.SD[1:last_period], by = .(name, year)]
Xy = first3[,.(
  last_status = .SD[last_period,status],
  max_category = max(category),
  max_wind_speed_thus_far = max(wind),
  min_pressure = min(pressure),
  max_pressure = max(pressure),
  y = max_wind_speed
```
```

```
  ),
  by = .(name, year)
]
Xy = Xy[,.SD[1], by = .(name, year)]
Xy[,max_category := factor(max_category, ordered = FALSE)]
Xy
table(Xy$max_category, Xy$last_status)
```

```

Fit your model. Validate it.

```{r}
#TO-DO
lm_mod = lm(y~. -name, Xy)
coef(lm_mod)
summary(lm_mod)
```

Assess your level of success at this endeavor.

Not so well, although parts are not so clear. For example, our standard error for wind_speed was only 24%, which is ok

More data munging with table joins

```{r}
pacman::p_load(tidyverse, magrittr, data.table)
```
```

We will be using the `storms` dataset from the `dplyr` package. Filter this dataset on all storms that have no missing measurements for the two diameter variables, "ts_diameter" and "hu_diameter". Zeroes count as missing as well.

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

From this subset, create a data frame that only has storm name, observation period number for each storm (i.e., 1, 2, ..., T) and the "ts_diameter" and "hu_diameter" metrics.

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

Create a data frame in long format with columns "diameter" for the measurement and "diameter_type" which will be categorical taking on the values "hu" or "ts".

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

Using this long-formatted data frame, use a line plot to illustrate both "ts_diameter" and "hu_diameter" metrics by observation period for four random storms using a 2x2 faceting. The two diameters should appear in two different colors and there should be an appropriate legend.

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

In this next first part of this lab, we will be joining three datasets in an effort to make a design matrix that predicts if a bill will be paid on time. Clean up and load up the three files. Then I'll rename a few features and then we can examine the data frames:

```{r}
rm(list = ls())
pacman::p_load(tidyverse, magrittr, data.table, R.utils)
bills = fread("https://github.com/kapelner/QC_MATH_342W_Spring_2021/raw/master/labs/bills_dataset/bills.csv.bz2")
payments = fread("https://github.com/kapelner/QC_MATH_342W_Spring_2021/raw/master/labs/bills_dataset/payments.csv.bz2")
discounts = fread("https://github.com/kapelner/QC_MATH_342W_Spring_2021/raw/master/labs/bills_dataset/discounts.csv.bz2")
setnames(bills, "amount", "tot_amount")
setnames(payments, "amount", "paid_amount")
head(bills)
head(payments)
head(discounts)
```

The unit we care about is the bill. The y metric we care about will be "paid in full" which is 1 if the company paid their total amount (we will generate this y metric later).

Since this is the response, we would like to construct the very best design matrix in order to predict y.

I will create the basic steps for you guys. First, join the three datasets in an intelligent way. You will need to examine the datasets beforehand.

```{r}
bills_with_payments = merge(bills, payments, all.x = TRUE, by.x = "id", by.y = "bill_id")
bills_with_payments[, id.y := NULL]
bills_with_payments_with_discounts = merge(bills_with_payments, discounts, all.x = TRUE, by.x = "discount_id", by.y = "id")
colnames(bills_with_payments_with_discounts)
bills_with_payments_with_discounts
```

```

Now create the binary response metric `paid_in_full` as the last column and create the beginnings of a design matrix `bills_data`. Ensure the unit / observation is bill i.e. each row should be one bill!

```{r}
#TO-DO
table(bills_data$paid_in_full, useNA = "always")

bills_with_payments_with_discounts[, total_paid := sum(paid_amount, na.rm = TRUE), by = id]
#ideally should be cumamalitve sum
bills_with_payments_with_discounts[, paid_bill := total_paid >= tot_amount, by = id]
bills_paid = bills_with_payments_with_discounts[, .(paid_bill = any(paid_bill)), by = id]
table(bills_paid$paid_bill)
```

How should you add features from transformations (called "featurization")? What data type(s) should they be? Make some features below if you think of any useful ones. Name the columns appropriately so another data scientist can easily understand what information is in your variables.

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

Now let's do this exercise. Let's retain 25% of our data for test.

```{r}
K = 4
test_indices = sample(1 : nrow(bills_data), round(nrow(bills_data) / K))
train_indices = setdiff(1 : nrow(bills_data), test_indices)
bills_data_test = bills_data[test_indices, ]
```
```

bills_data_train = bills_data[train_indices,]

```

Now try to build a classification tree model for `paid_in_full` with the features (use the `Xy` parameter in `YARF`). If you cannot get `YARF` to install, use the package `rpart` (the standard R tree package) instead. You will need to install it and read through some documentation to find the correct syntax.

Warning: this data is highly anonymized and there is likely zero signal! So don't expect to get predictive accuracy. The value of the exercise is in the practice. I think this exercise (with the joining exercise above) may be one of the most useful exercises in the entire semester.

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

For those of you who installed `YARF`, what are the number of nodes and depth of the tree?

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

For those of you who installed `YARF`, print out an image of the tree.

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