Getting, Processing, and Analyzing Github Data for Project 1

Andrew Leroux

11 October, 2017

This document provides all the code used to get the data, and perform the analysis for my Project 1. Note that although the code is all here, and can be run with only one minor change, due to the time it takes to scrape the data, and the file size of the data once scraped/processed, **the only code that is actually executed locally** is code after the section titled "Analyzing the data" subsection "Extract summary features of the data".

Getting the data

The data was retrieved from Github. The data was retrieved over the 72 hour period period 10/02/2017-10/04/2017. All data retrieval and analyses were performed in *R* (add package + R citation). At a high level, the retrieval procedure followed the two steps below

- Use the *gh* package available (add package citation) to find all repos associated with the class which were created on or after 12/01/2007
- Use the *gh* package to search the repos discovered in the previous step for a file called "run_analysis.R". Then scrape the data using the url structure implied by the location of the "run_analysis.R" file using the *readlines* function

More detailed information on each of these steps is provided below.

Finding Repo Names

Searching for repos associated with the getting and cleaning data class results in over 30,000 search results. However, the github API will only report up to 1,000 search results. To get around this, we searched repos by date of creation considering periods of two weeks. The creation dates examined spanned the dates 12/01/2007-10/02/2017. The justification for the start date was based on the start of the creation date of the class as well as some manual exploration which suggested the search results turned up zero entries for reasonable creation dates beyond this period. Note that using a 2 week moving window we never hit the 1,000 search result limit, implying that our window was sufficiently high resolution for this data. In another application (or for the class going forward), this window may need to shrink in order to capture all repos created during the search window.

The exact repo search was performed using the query:

"GET /search/repositories?q=getting+and+cleaning+data+**repo_date**&per_page=100",

where **repo_date** is of the form "created:2007-11-31..2007-12-15" to get repos created anytime during the two week period 12/01/2007-12/14/2007.

We note that one limitation of our procedure is that any repos created prior to 12/01/2007 would not be included in our analysis. However, since this date is prior to the creation of the class, in order for someone to be missed using this procedure, they would've had to rename or re-purpose an existing repo. We believe this to be unlikely, but note that it is a possibility. Also, note that our code does not allow for repository names to have hypes in them. These individuals were excluded from our analysis. These individuals number **INCLUDE NUMBER**.

The code below shows exactly how this procedure was down. Note that to re-run this code, a user will need to supply their own github token (the fourth line of the code below). Note: due to the time it takes to scrape the data, this code is not executed locally.

```
library(dplyr); library(gh);library(lubridate)

token <- readLines("../AdvDataScience_Project1/github_token.txt")[1]

### Only 1000 results per page (the max).
### Search by range of dates created -- 1 week periods. Should be able to grab them all
date_start <- ymd("2007-12-01")        ## start date
day_inc    <- 14                       ## increment days by 14 at a time
dates <- c(); i <- 1
while(date_start < Sys.Date() - (day_inc+1)) {
        dates[[i]] <- c(rep(date_start,2) %m+% c(days(-1),days(day_inc+1)))
        date_start <- date_start + days(day_inc + 1)
        i <- i + 1
}
rm(list=c("date_start","i","day_inc"))

## Loop over date ranges and extract:
##   repos - vector of repo names
##   branch - vector of branch names (not everyone leaves default "master")
##   score - github search scoring variable
repos <- branch <- score <- date_repo <- c()
for(i in 1:length(dates)){
        gh_date        <- paste("created:", paste(dates[[i]], collapse=".."), sep="")
        repo_len_start <- length(repos)

        ## max search result is 100 per page, with up to 10 pages
        ## loop over 10 pages
        for(k in 1:10){
                gh_get <- paste("GET /search/repositories?q=getting+and+cleaning+data+",
                                gh_date, "&per_page=100", sep="")
                x     <- try(gh(gh_get, page=k, .token=token))

                if("try-error" %in% class(x)) break

                repos  <- c(repos, vapply(x[[3]], "[[", character(1), "full_name"))
                branch <- c(branch, vapply(x[[3]], "[[", character(1), "default_branch"))
                score  <- c(score, vapply(x[[3]], "[[", numeric(1), "score"))

        }
        delta_repos <- length(repos) - repo_len_start     ## keep track of which
        date_repo   <- c(date_repo, rep(i, delta_repos))  ## dates correspond to which repo

        Sys.sleep(60)                           ## add to prevent exceeding API rate limit
        print(dates[i]);print(length(repos))    ## optional print statement
}
rm(list=c("delta_repos","gh_get","x","repo_len_start","gh_date","i"))
```

Scraping the Data

To scrape the data, we looped over the repos found in the previous step and searched their (entire) repository for a file called "run_analysis.R". To do so we used the following search query in the **gh()** function: "GET /search/code?q=repo:**repo_name**+extension:r" where **repo_name** is the repository name. This will search the entire repository for any .R files.

We then used regular expressions (*gregexpr*) to find whether any of these .R files matched "run_analysis.r" by using the *to.lower* function in R to allow for various capitalized letters and still match. Finally, we scraped the "run_analysis.R" file using the *readlines* function on the appropriate url based on the file name, repository name and branch name.

We note that our code does impose some limitations. First and foremost, there were some files that we attempted to read, but were found to be non- UTF8 files. We did not try to parse these files at all and they were excluded from the analysis. From manual inspection, these tended to be individuals who copied and pasted R console output into a .R script and used that as their final project. So these are users who, sometimes, were able to complete the task. We estimate the number of these individuals to be **INCLUDE NUMBER**.

The code below shows exactly how this procedure was down. Note: due to the time it takes to scrape the data, this code is not executed locally.

```
## create empty list to hold students' code
## each element of this list will be a character vector with each entry
## corresponding to a line of code
code <- list()
for(i in 1:length(repos)){
        code[[i]] <- NA
}

## loop over recovered repos to get run_analysis.R
for(i in 1:length(repos)){
        repo   <- repos[i]

        ## search for all .r files
        string <- paste0("GET /search/code?q=repo:", repo,"+extension:r")
        res    <- try(gh::gh(string, .token=token))

        if("try-error" %in% class(res)) next

        ## check to see if we found any .r files
        ## if none exist, exit this iteration of the loop
        path <- try(res[[3]][[1]]$path)
        if("try-error" %in% class(path)) next

        has_code <- FALSE
        ## loop over the .r files found to see if any of them are correspond to
        ## "run_analysis.R", ignoring capitalization.
        for(k in 1:length(res[[3]])){
                path_cur      <- res[[3]][[k]]$path
                file_name_inx <- c(gregexpr("/[aA-zZ]+?.[rR]",path_cur)[[1]][1],
                                   nchar(path_cur))
                file_name     <- substr(path_cur, file_name_inx[1]+1, file_name_inx[2])

                if(tolower(file_name) != "run_analysis.r") next
```

```
                ## exit the loop once we find "run_analysis.R"
                has_code <- TRUE
                break
        }

        ## exit this iteration of the loop if no "run_analysis.R" file found
        if(!has_code) next

        ## create the url for the code, incorporating the branch name for the rep
        code.url <- file.path("https://raw.githubusercontent.com",repo, branch[i], path_cur)
        ## change white space to %20 as required to be a valid url
        code.url <- gsub(" ", "%20", code.url)

        ## read in the code as a character vector, with one line of the .r file = one element
        ## and trim white space
        code[[i]] <- try(gsub("\\\\", "", trimws(readLines(code.url))))

        ## randomly choose the wait time to check the next repo -- needed to prevent "abuse"
        ## flagging
        Sys.sleep(sample(4:10,size=1))
        print(i) ## optional print statement
}
```

Processing the Data

Extracting Features of the Data

Note: due to file size limitations for Github, this cannot be compile locally so this code will not run when the markdown document is compiled.

```
## write a function get_data
## which takes as it's only arguement, a character vector
## corresponding to a students' code
##
## will return a data frame with number of rows equal to the number of
## rows of a students' .R file and elements:
##  - line:            numeric, which line in the .R file this row corresponds to
##  - blank_line:      locigal variable, is this a blank line? If so, TRUE, ow FALSE
##  - comment_line:    locigal variable, is this line commented out? If so, TRUE, ow FALSE
##  - assignment:      logical variable, does this line assign some value to a new object?
##                     If so, TRUE
##  - assignment_name: name of the variable which is assigned a value. NA if
##                     assignment = FALSE
##  - named_functions: matrix where each row corresponds to the approriate row of the code,
##                     each column denotes a named function used, in order of occurence.
##                     Filled in with NAs as appropriate
##  - subset_functions: matrix where each row corresponds to the approriate row of the code,
##                     each column denotes a base subset function used ($, [, [[), in order
##                     of occurence. Filled in with NAs as appropriate.
get_data <- function(x){
        if(length(x) == 1 & !is.finite(x[1])) return(NA)
        if(length(x) == 0) return(NA)
```

```
## number of lines in the .R file code
## number of lines of comments only
## number of blank lines
n_lines         <- length(x)
comment_line    <- grepl("^#", x)
blank_line      <- try(nchar(x)==0)

## this error checking will catch non utf-8 encoded files
## no attempt is made to parse these files
if("try-error" %in% class(blank_line)) return(NA)

## total number of characters in each line
n_char_tot      <- vapply(x, nchar, numeric(1))

## check for whether a line of code involves the assignment of a variable
## note: this will not catch uses of the assign() function
is_assign <- grepl("^[a-zA-Z]+[0-9]* *<- *", x) | grepl("^[a-zA-Z]+[0-9]* *= *", x)

## get the name of the variable which is assigned to
assign_name_loc <-  gregexpr("^[a-zA-Z]+[0-9]*(\\([a-zA-Z]+[0-9]*\\))? *([<-]|[=])",x)
assign_names <- vapply(1:n_lines, function(y){
        tmp <- assign_name_loc[y][1]
        ## match fail if length is greater than 1
        if(!(tmp[1] %in% 1) | length(tmp[1]) > 1) return(NA_character_)

        name_only <- gregexpr("^[a-zA-Z]+[0-9]*(\\([a-zA-Z]+[0-9]*\\))?",x[y])
        trimws(substr(x[y], 1, attributes(name_only[[1]])$match.length))
},character(1))

assignment_object <- ifelse(grepl("\\(",assign_names),0,1)
assignment_object[is.na(assign_names)] <- NA


## get names of top-level functions used
## note: this will not capture fns used in side apply type fns

## create a duplicate of "x" which replaces [ with white space
x_wo_sub <- gsub("\\[", "       ", x)

## get the locations of named functions
fn_patt <- "[A-Za-z]+[0-9]*.{0,1}[A-Za-z]+[0-9]*\\("
named_fn_loc <- gregexpr(paste("(^",fn_patt,")",
                               "|([,|=|-| |^]{1} *[^\"]\\[*",
                               fn_patt,")", sep=""),
                         x_wo_sub)

## subsetting includes $ and [
subset_fn_loc <- gregexpr("( *\\[+ *[A-Za-z]+[0-9]*)|(\\$[A-Za-z]+)", x)


## store the names of named functions and subset functions in matrices
## get dimensions for these matrices
max_named_func <- max(vapply(named_fn_loc,length, numeric(1)))
```

```
        max_sub_func   <- max(vapply(subset_fn_loc,length, numeric(1)))

        mat_named_func <- matrix(NA, ncol=max_named_func, nrow=n_lines)
        mat_sub_func   <- matrix(NA, ncol=max_sub_func, nrow=n_lines)

        ## fill in the matrices using the location of named/subset functions
        ## found previously
        for(n in 1:n_lines){
                tmp     <- named_fn_loc[n][[1]]
                tmp_sub <- subset_fn_loc[n][[1]]

                if(tmp[1] != -1){
                        for(k in 1:length(tmp)){
                                mat_named_func[n,k] <- substr(x_wo_sub[n], tmp[k],
                                                              tmp[k]+
                                                              attributes(tmp)$match.length[k]-2)
                        }
                }
                if(tmp_sub[1] != -1){
                        for(k in 1:length(tmp_sub)){
                                mat_sub_func[n,k] <- substr(x[n], tmp_sub[k], tmp_sub[k])
                        }
                }
        }

        ## return object is a data frame with all these features
        ret <- data.frame("line" = c(1:n_lines),
                          "blank_line" = blank_line,
                          "comment_line" = comment_line,
                          "n_characters" = n_char_tot,
                          "assignment" = is_assign,
                          "assignment_name" = assign_names,
                          "assignment_object" = assignment_object,
                          "named_functions" = I(mat_named_func),
                          "subset_functions" = I(mat_sub_func),
                          stringsAsFactors = FALSE)

        ret
}
```

Create final tidy dataset by merging elements of the list of processed data. Does not evaluate due to Github size limitations.

```
## apply our get_data function to the
## list of subjects' code
processed_list <- sapply(code, get_data)

## get max number of columns for named and subset funcitons, respectively
## so we can create matrices of equal dimension across subjects
max_fns <- vapply(seq_along(processed_list), function(x){
        if(!is.finite(processed_list[[x]][[1]])) return(c(NA,NA))
        c(ncol(processed_list[[x]]$named_functions),ncol(processed_list[[x]]$subset_functions))
}, numeric(2))
max_named <- max(max_fns[1,], na.rm=TRUE)
```

```
max_subset<- max(max_fns[2,], na.rm=TRUE)



## create our final, tidy data matrix
data       <- c()
named_mat  <- c()
subset_mat <- c()

for(i in 1:length(code)){
        if(is.na(code[[i]][1]) | is.na(processed_list[[i]])[1]) next

        tmp  <- data.frame("repo"=repos[i],
                           "date"=date_repo[i],
                           "score"=score[i],
                           processed_list[[i]],
                           stringsAsFactors = FALSE)
        data <- rbind.data.frame(data, tmp[,-c(11:12)])


        d_named <- max_named - ncol(tmp[,11])
        d_subset <- max_subset - ncol(tmp[,12])
        if(d_named > 0) for(j in 1:d_named)   tmp[,11] <- cbind(tmp[,11], NA)
        if(d_subset > 0) for(j in 1:d_subset) tmp[,12] <- cbind(tmp[,12], NA)

        named_mat <- rbind.data.frame(named_mat, unclass(tmp[,c(11)]),
                                      stringsAsFactors = FALSE)
        subset_mat <- rbind.data.frame(subset_mat, unclass(tmp[,c(12)]),
                                       stringsAsFactors = FALSE)

        print(i) ## optional print statement
}

data$raw_named_functions <- I(as.matrix(named_mat))
## account for some minor incorrect string grabs
data$named_functions     <- I(trimws(gsub("^[[:punct:]]", "", trimws(as.matrix(named_mat)))))
data$subset_functions    <- I(trimws(as.matrix(subset_mat)))
```

Analyzing the data

Extract summary features of the data

Perform PCA

Perform regression analyses

Create tables and figures used in final write up