Problem Set 2

Jeffrey Kwarsick

September 15, 2017

1 Problem 1

1.1 Part (a)

In tmp1.csv, there are $2 \cdot 10^6$ bytes. The number of bytes accounts for the $1 \cdot 10^6$ letters from the vector created in R plus the end-of-line (or comma delimiter) and end-of-file bytes. In tmp2.csv, there are 1,000,001 bytes in the file. This accounts for each of the one million letters written into the file from the 'chars' vector plus one additional byte associated with the end-of-file.

For tmp3.Rda, the raw R data saved in binary format, it is considerably smaller than a csv file of the of the same list of numbers. Since the numbers are either 17 or 18 bytes in length, depending which if the number is positive or negative and accounting for end-of-line and end-of-file bytes, the file should be what tmp4.csv should by, over 18 MB in size. Since tmp3.Rda is a binary file, I would expect the numbers to be stored in less bytes or accounts for repetitions in the list. For tmp5.csv, with the 'nums' list rounded to only have two decimal points, the length of each number is reduced to four or five bytes. Given the length of each number, the file size appears to be correct according to the filesize reported in the problem set.

1.2 Part (b)

When saving a file in R with save(), it will compress the file as much as it can based on repetitions. In the case of tmp6.Rda, it is a list of one million characters. There are repetitions that likely occur, making it easier to compress the file in comparison to a list of one million unique elements. This is better illustrated with tmp7.Rda, where it is a list of one million 'a' characters. The long continuous list of the a single character can be compressed to a great extent in comparison to the list of 26 repeated over a list length of a million as in tmp6.Rda.

2 Problem 2

2.1 Using Developer Tools

Below is work used to find where the relevant information required for the webscrapping user profile information for part (a) of problem 2. Using the Developr Tools in Google Chrome, I was able to discern the proper information to use to properly call up the user profile page on Google Scholar for Geoffrey Hinton. I proceeded stepwise until I completed extracted the user-ID as well as the html text of the top 20 citations on the profile page.

```
#Loading in relevant packages to query and download information from scholar.google.com
library('bitops')
library('XML')
library('curl')
library('RCurl')
library('stringr')
#url for searching for Geoffrey Hinton
#I wish to download the html file to extract the user-ID associated with searched user
```

```
#In this case, it is Geoffrey Hinton
url<-"https://scholar.google.com/scholar?hl=en&q=Geoffrey+Hinton"

#Reads lines from the url given.

#html<-readLines(url)
doc<-htmlParse(readLines(url))

## Warning in readLines(url):  incomplete final line found on 'https://scholar.google.com/scholar?hl=en&

#pulls out all relevant links in the html file
#includes the link containing user-ID from relevant search
#but there are many different user-IDs, must pick correct one

#links<-gethtmlLinks(html)
listofANodes<-getNodeSet(doc, "//a[@href]")

#Create empty vector
#Take list of a nodes and convert from 'externalptr' to 'character'
linklist<-c()
for (item in listofANodes) {
  temp<-as(item, "character")
  linklist<-c(linklist, temp)
}
#Searches string vector of links for user name of interest
#Geoffrey Hinton in this case
grep("Geoffrey Hinton", linklist)

## [1] 40 41

#returns the positions in the vector where the username is found

#take on of the instances where the username is found
#use gsub to strip everything in front and after the link of interest
tmp2<-gsub('(<a href=\\")|(".*)', "", tmp1) #end game url

## Error in gsub("(<a href=\\")|(".*)", "", tmp1):  object 'tmp1' not found

base_url<-"https:\\scholar.google.com"
full_url<-paste0(base_url,tmp2)

## Error in paste0(base_url, tmp2):  object 'tmp2' not found
```

2.2 Part (a)

After utilizing the developer tools, I crafted the following function below. The function takes the user input as a first and last name separated by a single space and parses it into a url. It searches the page and finds the user profile ID, required for the opening and scraping from the Google Scholar user profile page. It returns a list of length two; the first item in the list is the Google Scholar User Profile ID and the second item in the list is the html text of the researcher. The html text will be passed to the the second function for Part (b).

```
researcherhtml<-function()
{
```

```
  #Relevant libraries for the operation of the function
  library('bitops')
  library('XML')
  library('curl')
  library('RCurl')
  library('stringr')

  #Parts of the url needed for the researcher query
  base_url<-"https://scholar.google.com"
  url1<-"/scholar?q="

  #User input to obtain name of researcher of interest
  usr_input<-readline(prompt="Enter researcher name (first and last): ")
  spltname<-strsplit(usr_input, " ") #Splits the name based on prescence of whitespace
  #completed url needed for query
  complete_url<-paste0(base_url,url1,spltname[[1]][1],"+",spltname[[1]][2])

  #Reads lines from the url given. Stores locally for processing
  doc<-htmlParse(readLines(complete_url))

  Sys.sleep(2) #time for the system to sleep so rapid querying does not occur
  #Collects list of nodes from downloaded html
  #pattern is "//a[@href]"
  listofANodes<-getNodeSet(doc, "//a[@href]")

  #Empty vector linklist needed for type conversion
  #for-loop handles conversion from 'externalptr' to 'character'
  linklist<-c()
  for (item in listofANodes) {
    temp<-as(item, "character")
    linklist<-c(linklist, temp)
  }
  ##search1<-paste0(spltname[[1]][1]," ",spltname[[1]][2])
  ##search2<-paste0(spltname[[1]][2]," ",spltname[[1]][1])
  #locates the position of the user name of interest in the link
  occurence_position<-grep(usr_input, linklist)
  #now it takes the first occurence and extracts the url needed for researcher user profile citation pa
  location<-as.numeric(occurence_position[[2]])
  loc<-linklist[[location]]
  url2<-gsub('(<a href=\\")|(".*)',"",loc)
  #constructs the full url of researcher user profile
  full_url<-paste0(base_url,url2)
  #grabs the user ID from the string version of the extracted link
  user_ID<-gsub('(.*user=)|(&.*)',"",loc)

  doc2<-htmlParse(readLines(full_url))
  Sys.sleep(2)
  returnlist<-c(user_ID, doc2)
  return(returnlist)
}
#Command used to extract information for Geoffrey Hinton from Google Scholar
#GeoffreyHinton<-researcherhtml()
```

2.3 Part (b)

The function, gs_citation_dataframe(), reads in the html text from the output of the previous function and utilizes the getNodeSet() function to extract the journal article information requested in the problem; article title, authors, journal information, number of citations, and year published. The lists are then converted from the type 'externalptr' to 'character.' The gsub() function is then used to strip away the non-relevant text, before and after, the title, authors, journal information, number of citations, and the year published. The information is saved into vectors and then the data frame is constructed and returned.

```
##Function for Problem 2 Part (b) of Problem Set II
##Write a function to extract the Article Title, Authors, Journal Information, number of citations
##and year published to a dataframe.
gs_citation_dataframe<-function(html_citation_page)
{
  #Relevant libraries for the operation of the function
  library('bitops')
  library('XML')
  library('curl')
  library('RCurl')
  library('stringr')

  #function for converting contents of node lists to character class, from 'externalptr'
  character_conv<-function(NODELIST)
  {
    linklist<-c()
    for (item in NODELIST) {
      temp<-as(item, "character")
      linklist<-c(linklist, temp)
    }
    return(linklist)
  }

  #Take the html text from the output from the previous function
  #Use getNodeSet() in order to extract article information, citation numbers, and year published
  articleinfoNode<-getNodeSet(html_citation_page,'//tr[@class="gsc_a_tr"]')
  citationNode<-getNodeSet(html_citation_page,'//td[@class="gsc_a_c"]')
  yearNode<-getNodeSet(html_citation_page,'//td[@class="gsc_a_y"]')

  #Convert the lists of relevant information to characters for string processing
  articleinfoList<-character_conv(articleinfoNode)
  citationList<-character_conv(citationNode)
  yearList<-character_conv(yearNode)

  #Now perform string processing to extract the following--
  #title, author, journal info, number of citations, year published
  #Extracts the desired information by stripping away text before and after important text

  #Number of Citations
  cL<-gsub('(</a>.*)',"",citationList)
  num_cite<-gsub('(.*">)',"",cL)
  #Year Published
  year_1<-gsub('.*gsc_a_h">',"",yearList)
  year_pub<-gsub('</span>.*',"",year_1)
  #article title
```

```
  title_1<-gsub('(.*gsc_a_at">)',"",articleinfoList)
  titles<-gsub('(</a>.*)',"",title_1)
  #authors
  author_1<-gsub('(</div>\\n.*)',"",articleinfoList)
  authors<-gsub('(.*gs_gray">)',"",author_1)
  #journal information
  journal_info_1<-gsub('(.*gs_gray">)',"",articleinfoList)
  journal_info<-gsub('(<span.*)',"",journal_info_1)

  #Now create the dataframe of citations for user profile on Google Scholar
  user_profile_info<-data.frame(titles, authors, journal_info, year_pub, num_cite)
  return(user_profile_info)
}
#Line used to create data fram for Geoffrey Hinton
#GHinton<-gs_citation_dataframe(GeoffreyHinton[[2]])

#Now to pick another researcher, let's say Terrence Sejnowski
#Commands used are below
#Terrence<-researcherhtml()
#TSejnowski<-gs_citation_dataframe(Terrence[[2]])
```

2.4 Part (c)

This part must be able to look for an error from Google Scholar if there is no result from the user query on Google Scholar. If there is an improper input, it must recognize it and allow the user of the function to input another name.

2.5 Part (d)

When you click 'Show More' on the on the Google Scholar Citation Page, it changes the value given under spangscann.