STAT243: Problem Set 2

Zicheng Huang

09/15/2017

```
library(XML)
library(curl)
library(stringr)
library(testthat)
```

1.(a)

1. temp1.csv is saved in text format with a random sample of 1e6 letters. Since each of the letter take up 1 byte of space and followed by a delimiter, a newline, which also takes up 1 byte of space, the total size of temp1.csv is 2000000 byte in size as there are 1e6 letters and 1e6 delimiters.

2. temp2.csv is also saved in text format with the same set of letters as those in temp1.csv, but temp2.csv is saved by removing all the delimiters between the letters with the paste function and only the delimiter which is originally after the last letter is kept. As each letter takes up 1 byte and there are 1e6 of them, together with the only delimiter following the last letter, the size of temp2.csv is therefore 1000001 byte in size.

3. temp3.Rda is saved in binary format containing 1e6 numbers sampled from a standard normal distribution, each of the number takes up 8 bytes of space. However, the save function tends to compress the size, so the actually size of temp3.Rda is smaller than the expected 8000000 bytes.

4. temp4.csv is saved in text format containing 1e6 numbers sampled from a standard normal distribution, each decimal number has 15 digits after the decimal points and one digit before the decimal point. Together with the decimal point itself, the delimiter, and the minus sign if the number is negative, we see that each decimal number takes up either 18 or 19 bytes as each digit or character takes up 1 byte. So the size of the file is between 18000000 and 19000000.

5. temp5.csv is saved in text format containing 1e6 numbers sampled from standard normal distribution, with each decimal number rounded to the second decimal digits. Thus, there are 2 decimal digits after the decimal point, decimal digit before the decimal point, 1 decimal point, 1 delimiter, and 1 minus sign if the number is negative. Therefore, each number takes up either 5 or 6 bytes of space. So the size of the file is between 5000000 and 6000000.

1.(b)

1. temp6.Rda is saved in binary format containing 1e6 random letters with only one delimiter after the vary last letter. As the save function tends to compress the file, the size of temp6.Rda is smaller than the expected size of 8000001 bytes.

2. temp7.Rda is also saved in binary format and compressed by the save function as temp6.csv, but the letters in temp7.Rda are the same, which leads to the fact that temp7.csv is compressed to a much smaller size compared to that of temp6.csv.

2.(a)

This function takes a character string of scholar's name as input, and returns a list containging the html text corresponding to the scholar's citation page and the scholar's Google Scholar ID. This function cannot check if the input is valid or not, meaning that the function will not return any error message if the input scholar name is invalid or if the input scholar name does not match any search.

```
getCitationAndID <- function(name) {
  # replace the spaces in the name character string to plus sign
  fullname <- str_replace(name, " ", "+")
  # paste the fullname character string to url which leads to the search
  # page of the scholar
  URL <- paste("https://scholar.google.com/scholar?q=",
               fullname, "&btnG=&hl=en&as_sdt=0,5", sep = '')
  # get the html text corresponding to the search page
  html <- readLines(URL)
  # get the links contained in the html text
  links <- getHTMLLinks(html)
  # the 36th links always contains the information of the scholar's
  # Google Scholar ID, extract that information
  ID <- str_extract(links[36], "user=.*hl=en")
  # clean up what we extracted to keep only the string of Scholar ID
  userID <- str_replace(ID, "user=", "")
  UserID <- str_replace(userID, "&hl=en", "")
  # fill in the url with the resulting Scholar ID to direct to the
  # scholar's citation page
  userURL <- paste("https://scholar.google.com/citations?user=",
                   UserID, "&hl=en&oi=ao", sep = '')
  # get the html text of the corresponding scholar's citation page
  citation <- readLines(userURL)
  # make the html text into a more readable format
  citationPage <- htmlParse(citation)
  # return a list contains the html text and the Google Scholar ID
  result <- c(citationPage, UserID)
  return(result)
}

# This is the Google Scholar ID for Geoffrey Hinton
getCitationAndID("Geoffrey Hinton")[[2]]

## [1] "JicYPdAAAAAJ"

# This is the Google Scholar ID for Bin Yu
getCitationAndID("Bin Yu")[[2]]

## [1] "xT19JcOAAAAJ"
```

2.(b)

This function takes a character string of scholar's name as input, and returns an R data frame that contains the article title, authors, journal information, number of citations, and years of publicaiton. This function will internally call the getCitationAndID function defined in 2.(a) to get the html text corresponding to the scholar's citation page. This function cannot check if the input is valid or not, meaning that the function will not return any error message if the input scholar name is invalid or if the input scholar name does not match any search.

```
getInformation <- function(name) {
  # call the function getCitationAndID on the name input to get the
  # corresponding html text for the scholar's citation page
  doc <- getCitationAndID(name)[[1]]
  # extract all the article titles and save it as character strings
  # called Titles
  t <- getNodeSet(doc, "//a[@class = 'gsc_a_at']")
  Titles <- sapply(t, xmlValue)
  # extract the information containing all author and journal
  # information, seperate them into two seperate character strings
  # one called Authors that contains all the author information and
  # one called Journals that contains all the journal information
  a_j <- getNodeSet(doc, "//div[@class = 'gs_gray']")
  Author_Journals <- sapply(a_j, xmlValue)
  Authors <- Author_Journals[c(TRUE, FALSE)]
  Journals <- Author_Journals[c(FALSE, TRUE)]
  # extract all the years of publication information and save it as
  # character strings called Years
  y <- getNodeSet(doc, "//td[@class = 'gsc_a_y']")
  Years <- sapply(y, xmlValue)
  # extract all the number of citations information and save it as
  # character strings called Citations
  c <- getNodeSet(doc, "//td[@class = 'gsc_a_c']")
  Citations <- sapply(c, xmlValue)
  # create an R data frame that contains the article title, authors,
  # journal information, number of citations, and years of publicaiton.
  table <- data.frame(Titles, Authors, Journals, Citations, Years)
  return(table)
}

# get the information about the first article on the citation page
# for Geoffrey Hinton
getInformation("Geoffrey Hinton")[1,]

##                                                Titles
## 1 Learning representations by back-propagating errors
##                                 Authors                  Journals Citations
## 1 DE Rumelhart, GE Hinton, RJ Williams Nature 323, 533-536, 1986    34900*
##   Years
## 1  1986
```

2.(c)

Here we modify the function in 2.(a) so that it can check to see if the input is valid or not. Specifically, the input should be the fullname of the scholar with exactly one space in between the firstname, middlename, and lastname, and no spaces in front of firstname of after lastname. The input name should not contain any digits as well. If the input is invalid, the function willl return error message depending on the situation. After modifying the function in 2.(a), we will modify the function in 2.(b) to allow it to check for the input as well.

```
getCitationAndID_check <- function(name) {
  # if the input name does not follow the pattern of exactly one space
  # in between the firstname, middlename, and lastname, and no spaces
  # in front of firstname of after lastname, the function will return
  # relevant error message
  if (!str_detect(name, "^[a-zA-Z]+[[:space:]][a-zA-Z]+$")) {
    return(paste("Invalid Input: Please enter fullname with only one space",
                 "in between firstname, middlename, and lastname.", sep = ''))
  }
  # keep on searching for scholar if input is valid with similar steps in 2.(a)
  else {
    fullname <- str_replace(name, " ", "+")
    URL <- paste("https://scholar.google.com/scholar?q=",
                 fullname, "&btnG=&hl=en&as_sdt=0,5", sep = '')
    html <- readLines(URL)
    links <- getHTMLLinks(html)
    # if the input name does not match any existed scholar user profile, the
    # function will return relevant error message
    if (!str_detect(links[36], "user=")) {
      return("Invalid Input: Google Scholar search no result")
    }
    # apply the steps in 2.(a) if the name of input matches a existing scholar,
    # and return the corresponding html text of the citation page and the scholar's
    # Google Scholar ID
    else {
      ID <- str_extract(links[36], "user=.*hl=en")
      userID <- str_replace(ID, "user=", "")
      UserID <- str_replace(userID, "&hl=en", "")
      userURL <- paste("https://scholar.google.com/citations?user=",
                       UserID, "&hl=en&oi=ao", sep = '')
      citation <- readLines(userURL)
      citationPage <- htmlParse(citation)
      result <- c(citationPage, UserID)
      return(result)
    }
  }
}
```

After the modification, we can use the testthat package to check for the modified function.

```
expect_equal(getCitationAndID_check("Paul Johnson")[[2]], "A9vaVpEAAAAJ")
expect_equal(getCitationAndID_check("Bin Yu")[[2]], "xT19JcOAAAAJ")
expect_equal(getCitationAndID_check("abcd efgh"),
             "Invalid Input: Google Scholar search no result")
```

```
expect_equal(getCitationAndID_check("abcdefgh"),
             paste("Invalid Input: Please enter fullname with only one space",
                   "in between firstname, middlename, and lastname.", sep = ''))
expect_equal(getCitationAndID_check(" abcdefgh"),
             paste("Invalid Input: Please enter fullname with only one space",
                   "in between firstname, middlename, and lastname.", sep = ''))
expect_equal(getCitationAndID_check("abcdefgh "),
             paste("Invalid Input: Please enter fullname with only one space",
                   "in between firstname, middlename, and lastname.", sep = ''))
```

We can also modify the function in 2.(b) to allow for a check.

```
getInformation_check <- function(name) {
  # if the input is invalid or the search result does not match,
  # the getCitationAndID_Check functino will return the corresponding
  # error message, here we just re-return that message here as a check
  doc <- getCitationAndID_check(name)[[1]]
  if (is.character(doc)) {
    return(doc)
  }
  # if the input is valid and search has result, follow the steps in
  # 2.(b) to create an R data frame that contains the desired information
  else {
    t <- getNodeSet(doc, "//a[@class = 'gsc_a_at']")
    Titles <- sapply(t, xmlValue)
    a_j <- getNodeSet(doc, "//div[@class = 'gs_gray']")
    Author_Journals <- sapply(a_j, xmlValue)
    Authors <- Author_Journals[c(TRUE, FALSE)]
    Journals <- Author_Journals[c(FALSE, TRUE)]
    y <- getNodeSet(doc, "//td[@class = 'gsc_a_y']")
    Years <- sapply(y, xmlValue)
    c <- getNodeSet(doc, "//td[@class = 'gsc_a_c']")
    Citations <- sapply(c, xmlValue)
    table <- data.frame(Titles, Authors, Journals, Citations, Years)
    return(table)
  }
}
```

After the modification, we can use the testthat package to check for the modified function.

```
expect_equal(getInformation_check("abcd efgh"),
             "Invalid Input: Google Scholar search no result")
expect_equal(getInformation_check("abcdefgh"),
             paste("Invalid Input: Please enter fullname with only one space",
                   "in between firstname, middlename, and lastname.", sep = ''))
expect_equal(getInformation_check(" abcdefgh"),
             paste("Invalid Input: Please enter fullname with only one space",
                   "in between firstname, middlename, and lastname.", sep = ''))
expect_equal(getInformation_check("abcdefgh "),
             paste("Invalid Input: Please enter fullname with only one space",
                   "in between firstname, middlename, and lastname.", sep = ''))
```

2.(d) Here we modify the function in 2.(c) so that it can extract all the citation information of this scholar, instead of only the information on the first page.

```
getInformation_all_check <- function(name) {
  # create a empty data frame
  allInformation <- data.frame()
  # used the similar steps to check for validation of input
  result <- getCitationAndID_check(name)[[1]]
  if (is.character(result)) {
    return(result)
  }
  # if input is valid, run a while loop to get all the information
  else {
    # get the Google Scholar ID
    ID <- getCitationAndID_check(name)[[2]]
    # let n = 0 to start from the first page
    n = 0
    # the while loop will stop once there is no information on the page,
    # specifically, we set the page size to be 100 and each time of a new
    # loop add 100 to n to extract the next 100 information, stop the loop
    # when there is no information to extract on the page
    while (
      is.null(
        sapply(
          getNodeSet(
            htmlParse(
              readLines(
                paste("https://scholar.google.com/citations?user=", ID,
                      "&hl=en&oi=ao&cstart=", n, "&pagesize=100", sep = '')
              )
            ),
            "//a[@class = 'gsc_a_at']"
          ),
          xmlValue
        )[1][[1]]
      ) == FALSE
    ) {
      # these steps are the same as those in 2.(b) to extract information
      URL <- paste("https://scholar.google.com/citations?user=", ID,
                   "&hl=en&oi=ao&cstart=", n, "&pagesize=100", sep = '')
      doc <- htmlParse(readLines(URL))
      t <- getNodeSet(doc, "//a[@class = 'gsc_a_at']")
      Titles <- sapply(t, xmlValue)
      a_j <- getNodeSet(doc, "//div[@class = 'gs_gray']")
      Author_Journals <- sapply(a_j, xmlValue)
      Authors <- Author_Journals[c(TRUE, FALSE)]
      Journals <- Author_Journals[c(FALSE, TRUE)]
      y <- getNodeSet(doc, "//td[@class = 'gsc_a_y']")
      Years <- sapply(y, xmlValue)
      c <- getNodeSet(doc, "//td[@class = 'gsc_a_c']")
      Citations <- sapply(c, xmlValue)
      # create a temp data frame to capture the information generated
      # in each loop
      Information_tmp <- data.frame(Titles, Authors, Journals, Citations, Years)
```

```
      # row bind the temp data frame to the empty data frame created eariler
      # to get all the information once the loop is done
      allInformation <- rbind(allInformation, Information_tmp)
      n = n + 100
    }
    # return the data frame containing all the information
    return(allInformation)
  }
}

# check the function by returning the dimension of the resulting data frame
dim(getInformation_all_check("Bin Yu"))

## [1] 250   5
```