Stat243: Problem Set 2

Qingan Zhao
SID: 3033030808

17 Sep. 2017

Problem 1

(a)

In the first 2 files, we are saving out characters as ASCII so we should have about 1 byte per character (actually it could in principle be saved in 7 bits rather than 8 bits, but it seems that in this case it actually uses 8 bits = 1 byte)

```
#save letters in text format
chars <- sample(letters, 1e6, replace=TRUE)
write.table(chars, file='tmp1.csv', row.names=FALSE, quote=FALSE,
            col.names=FALSE)
system('ls -l tmp1.csv', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  2000000 Nov 14 21:14 tmp1.csv"

chars <- paste(chars, collapse = '')
write.table(chars, file='tmp2.csv', row.names=FALSE, quote=FALSE,
            col.names=FALSE)
system('ls -l tmp2.csv', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  1000001 Nov 14 21:14 tmp2.csv"
```

We see that in the second case above, to save 1000000 characters it takes 1000000 bytes (1 MB), which is exactly 1 byte per character. The first case takes 2 MB because each character is on its own line, which means we need to save a newline character for each character, so we have 2000000 characters.

Now consider saving numbers in binary or text format

```
#save in binary format
nums <- rnorm(1e6)
save(nums, file='tmp3.Rda')
system('ls -l tmp3.Rda', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  7678320 Nov 14 21:14 tmp3.Rda"

#save in text format
write.table(nums, file='tmp4.csv', row.names=FALSE, quote=FALSE,
            col.names=FALSE, sep=', ')
system('ls -l tmp4.csv', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  18161670 Nov 14 21:14 tmp4.csv"
```

```
write.table(round(nums, 2), file='tmp5.csv', row.names=FALSE,
            quote=FALSE, col.names=FALSE, sep=', ')
system('ls -l tmp5.csv', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  5379126 Nov 14 21:14 tmp5.csv"
```

(b)

We can try saving numbers and characters using *save*() but without compression.

```
chars <- sample(letters, 1e6, replace=TRUE)
chars <- paste(chars, collapse='')
save(chars, file='tmp6-unc.Rda', compress=FALSE)
system('ls -l tmp6.Rda', intern=TRUE)
chars <- rep('a', 1e6)
chars <- paste(chars, collapse='')
save(chars, file='tmp7.Rda', intern=TRUE)
system('ls -l tmp7.Rda', intern=TRUE)
nums <- rnorm(1e6)
save(nums, file='tmp8-unc.Rda',compress=FALSE)
system('ls -l tmp8.Rda', intern=TRUE)
```

Compression works by taking advantages of patterns and representing repeated information in a more efficient way. So it makes sense that when we have the same character repeated many times, the compressed file can be very small, but when we have random characters, little compression occurs. Hence the limited compression when writing a file with random characters (600 KB compared to 1 MB) and the extreme compression when writing a file with all the same character (1 KB compared to 1MB) as seen here and in the original problem:

```
chars <- sample(letters, 1e6, replace=TRUE)
chars <- paste(chars, collapse='')
save(chars, file='tmp6.Rda')
system('ls -l tmp6.Rda', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  635301 Nov 14 21:14 tmp6.Rda"

chars <- rep('a', 1e6)
chars <- paste(chars, collapse='')
save(chars, file='tmp7.Rda')
system('ls -l tmp7.Rda', intern=TRUE)

## [1] "-rw-r--r--  1 franklin  staff  1056 Nov 14 21:14 tmp7.Rda"
```

Finally, note that the compressed file containing the 1 million random numbers is abit smaller (7.6 MB compared to 8 MB) than the uncompressed version - this augments our answer to Problem 1a.

Problem 2

(a)

Webscraping with Google Scholar. Note that Google Scholar does not have an API, so we are forced to desconstruct the queries that are produced when we first point and click on the website.
Here is our function for getting the citations HTML and the scholar ID. We construct the initial query based

on the name and then find the URL corresponding to the research profile, noting for the user of we find something other than a single profile.

```
getCitations <- function(firstName=NULL, lastName=NULL,
                         extended=FALSE){
  if(is.null(firstName) && is.null(lastName))
    stop("Please provide at least one name.")
  url <- paste0('https://scholar.google.com/scholar?q="',
                firstName, '+',
                lastName,
                '"&btnG=&hl=en&as_sdt=0%2C5')
  researcherInfo <- htmlParse(readLines(url))
  ANodes <- getNodeSet(researcherInfo, "//a[@href]")
  num <- which(sapply(ANodes, function(x)
    str_detect(xmlValue(x), fixed('User profiles for'))))
  if(!length(num))
    stop("No profile found for that researcher")
  if(str_detect(xmlGetAttr(ANodes[[num+2]], 'href'),
                fixed('citations?user=')))
    warning('Found more than one profile; using the first.')
  num <- num+1 ##next link should be for the researcher
  profileUrl <- xmlGetAttr(ANodes[[num]], 'href')
  str <- str_extract(profileUrl, "user=[A-Za-z0-9]+")
  scholarID <- str_split(str, "=")[[1]][2]
  url <- paste0('http://scholar.google.com/', profileUrl)
  if(extended) ##this solves the extra credit
    url <- paste0(url, 'cstart=0&pagesize=100')
  citations <- htmlParse(readLines(url))
  return(list(id=scholarID, citationsHTML=citations))
}
result <- getCitations("Geoffrey", "Hinton")
getCitations("Mickey", "Mouse")
```

(b)

In this function we process the HTML using the XML functions we saw in unit 3. The trickiest part is the author-journal info as this is embedded in a "div" element that soecifies they are in a particular color and both author and journal are in the same format.

```
processCitations <- function(input){
  if(!is.list(input) || names(input) != c('id', 'citationsHTML'))
    stop("'input' should be produced by getCitations")
  authorJournalHTML <- getNodeSet(input$citationsHTML,
                                  "//div[@class='gs_gray']")
  authorJournal <- matrix(sapply(authorJournalHTML, xmlValue),
                          ncol=2, byrow=TRUE)
  titleHTML <- getNodeSet(input$citationsHTML,
                          "//a[@class='gsc_a_at']")
  title <- sapply(titleHTML, xmlValue)
  yearHTML <- getNodeSet(input$citationsHTML,
                         "//td[@class='gsc_a_y']")
  year <- as.numeric(sapply(yearHTML, xmlValue))
  numCitationsHTML <- getNodeSet(input$citationsHTML,
```

```
                                    "//td[@class='gsc_a_c']")
  numCitations <- sapply(numCitationsHTML, xmlValue)
  numCitations <- as.numeric(str_replace(numCitations,
                                         "\\*", ""))
  citations <- data.frame(author=authorJournal[, 1],
                          journal=authorJournal[, 2],
                          title=title,
                          year=year,
                          numCitations=numCitations,
                          stringsAsFactors=FALSE)
  return(citations)
}
citations <- processCitations(result)
```

(c)

```
test_that('test Geoffrey Hinton works',
          expect_true(is.list(getCitations('Geoffery', 'Hinton'))))
test_that('test Bugs Bunny fails',
          expect_error(getCitations('Bugs', 'Bunny')))
test_that('test Jordan warns',
          expect_warning(getCitations('Jordan'),
                         'Found more than one profile'))
zhaoCites <- processCitations(getCitations("Franklin", "Zhao"))
test_that('test Zhao data frame',
          expect_true(all(zhaoCites$year > 1970) &
                        sum(str_detect(zhaoCites$author, "Zhao")) > 15))
```

(d)

If we use browser develop tools when clicking on "Show More", we see that the URL has a form like this: *http://scholar.google.com/citations?user=yxUduqMAAAJ&hl=en&oi=ao&cstart=0&pagesize=100*. So when we make our query of the scholar profile, we need to add on the *start* and *pagesize* arguments to the query. This is seen in *getCitations*() with the *if*(*extended*) clause. However, experimentation reveals that the website will not provide more than 100 citations at a time. So one would need to write a loop that ask for 100 at a time and stops when the number returned is fewer than 100, indicating er have gotten all the citations.