STA 242: Assignment 4

Scott Frost and Tom Shapland

March 18, 2011

**Preamble**

In this assignment we manipulated text data using regular expressions in order to import text data into R. Once the text data was in R, we explored features of the data using regular expressions and network plots. Our work focused on the Enron email data set.

**1. Description of problems**

First, we will describe the problems we encountered as we designed the code and our strategies for solving the problems.

We wanted to extract the email header information (sender, recipient, date, and subject) and the email body. While it was easy to identify the lines containing this information using regular expressions, it was difficult to extract the information associated with these lines. We were advised to use the read.dcf() function, which befuddled Tom. Scott figured out how to use the read.dcf() command, but Tom decided to extract the data in a different manner. We will describe how Scott used the read.dcf() command and the alternative method that Tom used.

Scott’s Method: getHddr():

getHddr(emaillist, "Date", "To", "From", "Subject")

The overall object of the getHddr function was to input an email, along with a list of header fields, and return a list of headers. The final version of the function getHddr (see Appendix A) uses two lapply functions to extract the relevant email header information. The initial lapply parses each email at the last line of the header, and returns a list of headers. This header list is then passed to the second lapply which uses read.dcf() to extract the requested information.

The initial version was created with a for loop (see Appendix A). A matrix is first created, then each email is looped over and line numbers 2 – 5 lines are pulled out and placed in the matrix. These lines corresponded to the Date, To, From, and Subject fields. Read.dcf is then used to clean up the fields. The approach of calling individual lines is not flexible, and would only apply to email fitting the given pattern. The for loop also is not an efficient approach to handling a large dataset.

The second version of getHddr was first modified to remove the for loop. Two lapply functions were used as described above (see Appendix A). This version was unable to complete the requested call. An error was produced that stated the lengths of the columns don’t match. The bug was found in the second lapply. The connection was not being closed on exit from the function. This caused problems when requesting to open a large number of connections.

The final version of getHddr consists of a main function and two sub functions, cutHere and getDataFromText (see Appendix A). The sub functions are called from the FUN = field of the lapply functions. Having lapply call to a sub function gives greater overall flexibility. The issue of opening and closing the connection was easily resolved, and the sub function can be applied to additional tasks.

Tom’s Method: Regular Expressions

In Tom's method, we used regular expressions to identify lines in the data starting with the string “From: ”, which contained the sender information (see Appendix B). In regular emails, the sender line is preceded by the date line, and followed by the recipient line and the subject line. Once we identified the sender lines, we indentified the date, recipient, and subject lines by their position relative to the sender line. The contents of the sender, date, recipient, and subject lines were extracted. Tom’s method ignored the email body.

In some cases the emails were malformed. In other words, the recipient line did not follow the sender line. If a line that follows a sender line does not start with “To: “, then the line was identified as coming from a malformed email. Malformed emails were culled. We may have lost important information in our culled emails, but hopefully we preserved enough material to indict Kenneth Lay before he suffers heart failure. Using Tom’s method, the total number of emails was 447,380 and the number of malformed emails was 144,050, i.e., nearly 1/3 of all the emails were malformed.

It was difficult to extract the header information and email bodies without running into memory problems. There were 27,390,842 lines in the Enron email data set. Initially Scott's method required reading in the whole data set in order to extract the header information and the email body. R was unable to read in 27 million lines. Scott got around this problem by looping over the 150 email recipients. All the emails from a single person, i.e., p-allen, were read in and processed. This introduced a new problem for the getHddr function. It was unable to deal with malformed email headers. The malformed headers exhausted the buffer and crashed the function. Initially it was assumed that the functions was not opening and closing the connection correctly. This lead to a dead-end, as it turned out opening and closing connections was not the problem. The malformed headers contained lines that needed to be cut in order to “fit” inside the buffer. Two possible next steps would be to implement a function similar to Duncan’s “fixHeaderLines or a “try catch” scenario to bypass the malformed lines.

Tom also ran into memory problems with his method. Tom resolved this problem by reading 10,000 line segments of the data set into R, extracting the header information, outputting the header information to a text file, then repeating the loop.

Only a small portion of our time was spent developing usable code. For most of our work in this assignment we did not use version control because we only had a few lines of quasi-functional code. Once we had a small amount of functional code, we used version control.

**2. Interesting features of the Enron dataset**

We will next describe interesting features of the Enron data set (see Appendix C). We created 2 network maps. The first network map describes the first 10,000 emails in the data set. The second network map describes the email communications involving Kenneth Lay.

For the first network map we originally had hoped to produce a network map for the entire data set, but our machines crashed during the computation of the very large network plot. We instead used a subset of the email data consisting of the first 10,000 lines. Because Tom was unable to install Rgraphviz on any machine he touched, we decided to use the igraph package to create the network map. The igraph package is available at the CRAN repository. The igraph package is subject to the requirements of packages on the CRAN repository, so it readily installs on most machines. The igraph package doesn’t seem to have functionality for weighting the edges, so we were unable to include this feature in our analysis.

We used a force-based algorithm, the Fruchterman*-*Reingold algorithm, for creating the layout of the network plot. Force-based algorithms make the nodes mutually repulsive, while the edges counteract the repulsive forces between the nodes. The plot has the aesthetically-pleasing characteristic of nearly equal spacing between the nodes in a network. If there are two groups of nodes on the same plot, i.e., there are two sets of mutually-exclusive networks, then the two independent networks separate.

In the plot of the network of the first 10,000 emails of the data, we see 2 groups containing many nodes. We also see many more small groups along the periphery of the plot. The large groups describe the email network of 2 individuals at Enron. The center of each large network is an Enron employee, who sent many emails to many different people. The small groups were created from email addresses that were included in the list of email recipients. If each email had only one sender and one recipient, then there would be no small peripheral groups.

The plot of Kenneth Lay’s network included 4,366 emails. We see similar features in Kenneth lay’s network as we saw in the first network map. The plot has a large group at the center representing Kenneth Lay’s network, and small satellite groups representing the relationship between email addresses for the case of more than one email recipient.

We also explored the email data using regular expressions. A search of the subject of Ken Lay’s emails for words “salary”, “stock”, “price”, “bank”, and “share” yielded 1113 returns. Some of the more interesting returns were…

"New Stock Option Plan Memo"

"Bankruptcy"

"A dissapointed stockholder"

"Demand Ken Lay Donate Proceeds from Enron Stock Sales"

"How can you sell Enron for $9.50 per share? We've already\tlost"

"Concerns With the Current Stock Price"

Kenneth Lay was indicted for misleading employee stockholders. We also searched the email subjects for references to the Bush family.

"BUSH-CHENEY RECOUNT FUND"

"Re: Conversation w/President & Mrs. Bush"

"PRESIDENT BUSH invite letter from Ken Lay"

Kenneth Lay had ties with the Bush family, which has played into the conspiracy theories that he faked his death before his sentencing.  
**3. Interesting features of the R-mailing list**Because we encountered so much difficulty in processing the Enron email data set, we did not have time to explore the R-mailing list.

**4. Figures**

C:\Documents and Settings\0shapland.tm\My Documents\EnronEmailNetwork.tiff

Figure 1. Network Map of first 10,000 emails in the Enron email data set. C:\Documents and Settings\0shapland.tm\My Documents\KenLayEmailNetwork.tiff

Figure 2. Kenneth Lay’s email network map

**Appendix A: Scott’s Method**

**#### First Version of getHddr ####**

**EMhead** = function(li, ...) {

mat = matrix(ncol = 4, nrow = length(li))

for(i in 1:length(li)){

tmp = c(unlist(li[i])[2], unlist(li[i])[3], unlist(li[i])[4], unlist(li[i])[5])

mat[i,] = read.dcf(textConnection(tmp))

}

rownames(mat) = 1:length(li)

colnames(mat) = c("Date", "From", "To", "Subject")

as.data.frame(mat)

return(mat)

}

**#### Second Version of getHddr ####**

**testFOO** = function(ME, ...) {

info = c(...)

tmpHD = lapply(ME, function(i){unlist(i)[1:min(which(unlist(i) == ""))-1]})

ans = lapply(tmpHD, function(i){read.dcf(textConnection(i), fields = info)})

ans = as.data.frame(do.call('rbind', ans))

return(ans)

}

**#### Final Version of getHddr ####**

**getHddr** = function(emailList, ...){

tmpHD = lapply(emailList, cutHere)

ans = lapply(tmpHD, getDataFromText, info = c(...))

ans = as.data.frame(do.call('rbind', ans))

return(ans)

}

**getDataFromText** = function(tx, info){

con = textConnection(tx)

tmp = read.dcf(con, fields = info)

on.exit(close(con))

return(tmp)

}

**cutHere** = function(x){

Here = min(which(unlist(x) == ""))-1

return(x[1:Here])

}

**Appendix B: Tom’s Method**

#Create a connection to the compressed data set

con = gzfile("enron\_mail\_030204.tar.gz", "r")

#close(con)

# Determine the number of rows (records) in the data set

nrec = length(count.fields(con, sep="\n") ) #ans: 27390842

# Prepare the looping structure for reading in segments of the data set

left = nrec

nLines = 10000

# The looping structure for reading in segments of the data set

while(left > 0)

{

# read in a segment of the dataset

interval = readLines(con, n = nLines)

# identify the rows which start with "From: "

index = grep("^From: (.\*)@[a-zA-Z\_]+?\\.[a-zA-Z]{3}", interval)

indexBody = grep("", interval)

# identify the rows which have the Date, To, and Subject by

# their position relative to the "From" rows

Date = interval[index - 1]

From = interval[index]

To = interval[index + 1]

Subject = interval[index + 2]

# output the Date, To, From, and Subject

cat(Date, sep = "\n", file = "Date.txt", append = TRUE)

cat(To, sep = "\n", file = "To.txt", append = TRUE)

cat(From, sep = "\n", file = "From.txt", append = TRUE)

cat(Subject, sep = "\n", file = "Subject.txt", append = TRUE)

left = left - nLines

}

close(con)

# read in the Date, To, From and Subject, creating one table

table = cbind(readLines("Date.txt"),

readLines("From.txt"), readLines("To.txt"), readLines("Subject.txt"))

# determine the number of emails in the table, including malformed emails

length(table[,1]) #ans: 447380

# cut out all emails for which the "Date" row doesn't start with "Date"

# this eliminates malformed email headers

table = subset(table, grepl("^Date:", table[,1]) & grepl("^From:", table[,2]) &

grepl("^To:", table[,3]) & grepl("^Subject:", table[,4]) )

# determine the number of emails in the table, discluding malformed emails

length(table[,1]) #ans: 303330

# assign column names

names(table) = c("Date", "From", "To", "Subject")

# remove the "From", "To", "Date", "Subject" string from the beginning of each column

table[,1] = gsub("^Date: ", "", table[,1])

table[,2] = gsub("^From: ", "", table[,2])

table[,3] = gsub("^To: ", "", table[,3])

table[,4] = gsub("^Subject: ", "", table[,4])

# save the table, so that the process-time-intensive steps don't need repeating

save(table, file = "emailheaders.rda")

**Appendix C: Code for exploring the Enron Data set**

# create network graph

load(file = "emailheaders.rda")

# create a small subset of the data for testing code

tablesample = table[1:10000,]

# load the igraph package

library(igraph)

# create the igraph object and plot the network graph

table = data.frame(from = tablesample[,2], to = tablesample[,3])

f = graph.data.frame(table, directed = TRUE)

plot(f, vertex.label = NA, layout = layout.fruchterman.reingold, vertex.size=4,

vertex.label.dist=.5, vertex.color="red", edge.arrow.size=0.5,

main = "Enron email network", margin = 0)

### subset Ken Lay's email data, so we can make a network plot of his email data

load(file = "emailheaders.rda")

# extract header information from Ken Lay's sent and recieved emails

kenFrom = table[grep("(kenneth.lay@enron.com|rosalee.fleming@enron.com|klay@enron.com)", table[,2]),]

kenTo = table[grep("(kenneth.lay@enron.com|rosalee.fleming@enron.com|klay@enron.com)", table[,3]),]

ken = rbind(kenFrom,kenTo)

# substitute Ken Lay's email aliases for "kenneth.lay@enron.com"

ken[,2] = gsub("(rosalee.fleming@enron.com|klay@enron.com)", "kenneth.lay@enron.com", ken[,2])

ken[,3] = gsub("(rosalee.fleming@enron.com|klay@enron.com)", "kenneth.lay@enron.com", ken[,3])

# create network plot

table = data.frame(from = ken[,2], to = ken[,3])

f = graph.data.frame(table, directed = TRUE)

plot(f, vertex.label = NA, layout = layout.fruchterman.reingold, vertex.size=2,

vertex.label.dist=.5, vertex.color="red", edge.arrow.size=0.2,

main = "Kenneth Lay's email network", margin = 0)

# check the subject line for interesting material

length(ken[,4]) #4366

ken[grep("(salary|stock|price|share|bank)",ignore.case = TRUE, ken[,4]),4] #return 1108

ken[grep("bush",ignore.case = TRUE, ken[,4]),4]

sdlkFigure