UTILIZING SOCIAL MEDIA FOR LEAD GENERATION

CONFIGURATION MANUAL

Ajitesh Prakash, 15001113
M.Sc. Data Analytics, 2015-16
School of Computing

Table of Contents

Tools Utilized ..... **Error! Bookmark not defined.**1
Packages ..... **Error! Bookmark not defined.**2
Step by Step Guide ..... **Error! Bookmark not defined.**3

Tools Utilized:

1. R

R is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls, surveys of data miners, and studies of scholarly literature databases show that R's popularity has increased substantially in recent years. R is a GNU package. The source code for the R software environment is written primarily in C, Fortran, and R. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available. *(Source: Wikipedia)*

2. SQL

SQL Structured Query Language, is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and Data Control Language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

Packages

Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation.

1. SnowballC

Description: This function extracts the stems of each of the given words in the vector.

Usage: wordStem(words, language = "porter")

Arguments:

> **Words:** a character vector of words whose stems are to be extracted.
>
> **Language**: the name of a recognized language, as returned by get StemLanguages, or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes).

2. RODBC

Description: Package RODBC implements ODBC database connectivity

Usage: odbcTables(channel, catalog = NULL, schema = NULL, tableName = NULL, tableType = NULL, literal = FALSE)

Argument:

> **Channel**: connection handle as returned by odbcConnect, of class "RODBC".
>
> **Catalog, Schema:** tableName, tableType NULL or character: whether these do anything depends on the ODBC driver. The first three can be length-one character vectors, and tableType can specify zero or more types.

3. TM

TM or Topic Modelling package provides methods for data import, corpus handling, preprocessing, metadata management, and creation of term-document matrices.

Step by Step Guide of the Methodology

```
#Clearing the memory of R
rm(list=ls(all=TRUE))

#Set the working directory. This is ideally should be where data folder is.
setwd("C:/Users/Ajitesh/Desktop/Documents/RIC/Dissertation/App2")

#Load the Required Packages
require(RODBC)        # For establishing the channel with the SQL Server database.
require(tm)           # Topic Modelling Library
require(SnowballC)   #  Stemming Library

#Establish the channel with SQL Server database
#uid is the id to connect to the SQL Server and pwd the password for the same.
Channel= odbcConnect("Database_Name", uid= "userid", pwd= "**********")


#SQL query to extract the relevant profiles from the database.
#LUI is the Table with LinkedIn user data and LKC the table with LinkedIn information of companies.
query = pasteo("SELECT LUI.UserID, LUI.Full_Name, LUI.Headline, LUI.Industry,
LUI.Current_Employer, LKC.Specialties, LKC.Industry As 'CompInd',
FROM Linkedin_UserInfo LUI, Linkedin_Companies LKC, Twitter_UserInfo T
 WHERE(LUI.Headline LIKE '%Marketing%' OR LUI.Headline LIKE '%Public Relation%')
AND   LUI.LinkedinCompanies_ID    #  LKC.ID    # Joining the two table
AND   LKC.ID IS NOT NULL")

#Store the data extracted in a data table
input = sqlQuery(Channel, query)

#Reading Prospects Information
#This is the file with details of the seed profile.  Usually filtered from the feedback
#file given by the client
prospect = read.csv("DataProspects.csv")

#Appending Prospects information with the filtered profiles table.
data = rbind(input, prospect)

#Creating a collection of profiles referred to as a Corpus in the R environment.
#The Corpus() function can read from various sources.
#Selecting the attributes that are being considered in the business context

data$Corpus = paste(data$Headline, data$Current_Employer, data$Specialties,
data$CompInd, data$Description)
```

```
##########################################
##################
# Individual Similarity Scores
##################
##########################################
```

#Start preprocessing
#The *tm* package offers a number of transformations that aid in cleaning data.
```
library(tm)
```

#Stating Topic Modelling
 # Preliminary clean-up steps we need to do before we use the transformation necessary to proceed with processing the data

 # build a corpus, and specify the source to be character vectors
```
myCorpus <- Corpus(VectorSource(data$Corpus))
```

#Converting the corpus to lower case and removing all numbers. Since R is case sensitive, #"Hat" is not equal to "hat" – and hence the rationale for converting to a standard #case. This is done with the help of our new #friend, *content_transformer*.

 # convert to lower case
```
myCorpus <- tm_map(myCorpus, content_transformer(tolower))
```

#remove potentially problematic symbols
```
toSpace <- content_transformer(function(x, pattern) { return (gsub(pattern, " ", x))})
myCorpus <- tm_map(myCorpus, toSpace, "-")
```

 # Punctuation can be removed using our custom content #transformer, toSpace.

#Removing other characters
```
myCorpus <- tm_map(myCorpus, toSpace, '"')
myCorpus <- tm_map(myCorpus, toSpace, '"')
```

 # remove URLs
```
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeURL))
```

#Inspect random sections of corpus to check that the result is what you intend. If it all looks good, we can now apply the removePunctuation transformation. This is done as follows:

 # Remove anything other than English letters or space (punctuation and numbers)
```
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeNumPunct))
```

#The next step is to remove common words from the text. These include words such #as articles (*a, an, the*), conjunctions (*and, or but* etc.), common verbs (*is*), qualifiers #(*yet, however* etc) . The *tm* package includes a standard list of such stop **words as #they are referred to. We remove stop words using the #standard *removeWords* transformation like so: remove stopwords**

```
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))
```

#define and eliminate all custom stopwords

```
myStopwords <- c( "RT", "http", "via","can", "get", "new", "look", "one", "amp")
myCorpus <- tm_map(myCorpus, removeWords, myStopwords)
```

#remove whitespace

```
myCorpus <- tm_map(myCorpus, stripWhitespace)
myCorpusNEW <- myCorpus
```

 # Large corpus have many terms that have common root – for #example: play, player #and playing. Stemming is the process of reducing such related words to their #common root, which in this case would be the word 'play'.

#Stem Corpus

```
myCorpus <- tm_map(myCorpus,stemDocument)

stemCompletion2 <- function(x, dictionary)
{
  x <- unlist(strsplit(as.character(x), " "))
  # Unexpectedly, stemCompletion completes an empty string to
  # a word in dictionary. Remove empty string to avoid above issue.
  x <- x[x != ""]
  x <- stemCompletion(x, dictionary=dictionary)
  x <- paste(x, sep="", collapse=" ")
  PlainTextDocument(stripWhitespace(x))
}

myCorpus <- lapply(myCorpus, stemCompletion2, dictionary=myCorpusNEW)
myCorpus <- Corpus(VectorSource(myCorpus))
```

#Creation of the document term matrix (DTM)– DTM lists all occurrences of words in the corpus, by profile. In the DTM, the users are represented by rows and the terms from their corpus by columns. If a word occurs in a particular profile, then the entry in DTM for the row and column of the user is 1 else, it is 0 (if a term occurs more than once, say n, it is recorded as "n" in the relevant matrix entry).

#Create document-term matrix
#This will create a matrix with word corpus as rows we want them to be in columns

```
dtm <- DocumentTermMatrix(myCorpus, control = list(wordLengths = c(1, Inf)))

#Convert dtm to matrix
m <- as.matrix(dtm)


#Write as csv file
write.csv(m, file= "dtmTweet.csv")

#Collapse matrix by summing over columns
freq <- colSums(m)

#Length should be total number of terms
length(freq)

#Create sort order (descending)
ord <- order(freq,decreasing=TRUE)

#List all terms in decreasing order of freq and write to disk
freq[ord]
write.csv(freq[ord],"word_freq.csv")

# Checking for Similarity
cosineSim <- function(x){

  as.dist(x%*%t(x)/(sqrt(rowSums(x^2) %*% t(rowSums(x^2)))))

}

cs <- cosineSim(m)
write.csv(as.matrix(cs),file= "cosimTweet.csv")


#Adjacency matrix: set entries below a certain threshold to 0. This is an arbitrary choice
cs[cs < 0.15] <- 0
cs <- round(cs,3)
write.csv(as.matrix(cs),file= "AdjacencyMatrix.csv")


# Lead Extraction
cs_mat = as.matrix(cs)
dimension = dim(cs_mat)
dim = dimension[1]

rm(pfs)
```

```
rm(profsim)
profsim = data.frame()
i = 651

for (j in 1:dim){

  if ((i != j) && (cs_mat[i,j] != 0) )

  {
    Source       = inputEntity[i,3]
    Destination  = inputEntity[j,3]
    Score        = cs_mat[i,j]
    Headline     = inputEntity[j,4]
    Employer     = inputEntity[j,6]
    Industry     = inputEntity[j,8]
    pfs          = data.frame(Source, Destination, Score, Headline, Employer, Industry)
    profsim      = rbind(profsim, pfs)
  }
}

write.csv(profsim, "LeadList_AllEntity.csv")

##########################################
########                        ####
########## Sensitivity Analysis ############
########                        ####
##########################################

setwd("C:/Users/Ajitesh/Desktop/Documents/RIC/Dissertation/App2/4Sample")

# Sample of 4 profiles
sample = prospect[sample(nrow(prospect), 4), ]
sample = read.csv("sample.csv")
sample$X = NULL

# Binding the sample with the extracted profile list
dataS4 = rbind(input, sample)

# Creating Corpus of Attributes
dataS4$Corpus = paste(dataS4$Headline, dataS4$Current_Employer, dataS4$Specialties,
dataS4$CompInd)

#start preprocessing
library(tm)

#Stating Topic Modelling
# build a corpus, and specify the source to be character vectors
```

```
myCorpus <- Corpus(VectorSource(dataS4$Corpus))

# convert to lower case
myCorpus <- tm_map(myCorpus, content_transformer(tolower))

#remove potentially problematic symbols
toSpace <- content_transformer(function(x, pattern) { return (gsub(pattern, " ", x))})
myCorpus <- tm_map(myCorpus, toSpace, "-")

#Removing other charaters
myCorpus <- tm_map(myCorpus, toSpace, "")
myCorpus <- tm_map(myCorpus, toSpace, "")

# remove URLs
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeURL))


# Remove anything other than English letters or space (puntuation and numbers)
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeNumPunct))

#Remove stopwords
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))

#Define and eliminate all custom stopwords
myStopwords <- c( "RT", "http", "via","can", "get", "new", "look", "one", "amp")
myCorpus <- tm_map(myCorpus, removeWords, myStopwords)

#Remove whitespace
myCorpus <- tm_map(myCorpus, stripWhitespace)
myCorpusNEW <- myCorpus

#Stem document
myCorpus <- tm_map(myCorpus,stemDocument)

stemCompletion2 <- function(x, dictionary)
{
  x <- unlist(strsplit(as.character(x), " "))
  # Unexpectedly, stemCompletion completes an empty string to
  # a word in dictionary. Remove empty string to avoid above issue.
  x <- x[x != ""]
  x <- stemCompletion(x, dictionary=dictionary)
  x <- paste(x, sep="", collapse=" ")
  PlainTextDocument(stripWhitespace(x))
}
```

```
myCorpus <- lapply(myCorpus, stemCompletion2, dictionary=myCorpusNEW)
myCorpus <- Corpus(VectorSource(myCorpus))

#Create document-term matrix
#This will create a matrix with word corpus as rows we want them to be in columns

dtm <- DocumentTermMatrix(myCorpus, control = list(wordLengths = c(1, Inf)))

#convert dtm to matrix
m <- as.matrix(dtm)

#write as csv file
write.csv(m, file= "dtmTweet.csv")

#collapse matrix by summing over columns
freq <- colSums(m)

#length should be total number of terms
length(freq)

#create sort order (descending)
ord <- order(freq,decreasing=TRUE)

#List all terms in decreasing order of freq and write to disk
freq[ord]
write.csv(freq[ord],"word_freq.csv")

# Checking for Similarity
cosineSim <- function(x)
{
  as.dist(x%*%t(x)/(sqrt(rowSums(x^2) %*% t(rowSums(x^2)))))
}

cs <- cosineSim(m)
write.csv(as.matrix(cs),file= "cosimTweet.csv")


#Adjacency matrix: set entries below a certain threshold to 0.
#I choose half the magnitude of the largest element of the matrix
#As the cutoff. This is an arbitrary choice
cs[cs < 0.15] <- 0
cs <- round(cs,3)
write.csv(as.matrix(cs),file= "AdjacencyMatrix.csv")


# Lead Extraction
cs_mat = as.matrix(cs)
```

```
dimension = dim(cs_mat)
dim = dimension[1]


rm(pfs)
rm(profsim)
profsim = data.frame()
```

#Set i to be equal to the line items in the data file with the all entity row at the bottom of the file

```
i = 651

for (j in 1:dim){

 if ((i != j) && (cs_mat[i,j] != 0) )

 {
  Source              = dataS4[i,2]
  Destination     = dataS4[j,2]
  Score               = cs_mat[i,j]
  Headline          = dataS4[j,3]
  Employer       = dataS4[j,5]
  Industry         = dataS4[j,7]
  pfs                  = data.frame(Source, Destination, Score, Headline, Employer, Industry)
  profsim      = rbind(profsim, pfs)
 }
}

write.csv(profsim, "LeadList_Entity4.csv")

##########################################################
###
# 3 Profile
###
##########################################################


setwd("C:/Users/Ajitesh/Desktop/Documents/RIC/Dissertation/App2/3Sample")

# Sample of 3 profiles
sample = read.csv("sample.csv")

# Binding the sample with the extracted profile list
dataS3 = rbind(input, sample)
```

```
# Creating Corpus of Attributes
dataS3$Corpus = paste(dataS3$Headline, dataS3$Current_Employer, dataS3$Specialties, dataS3$CompInd)

#start preprocessing
library(tm)

#Stating Topic Modelling
# build a corpus, and specify the source to be character vectors

myCorpus <- Corpus(VectorSource(dataS3$Corpus))

# convert to lower case
myCorpus <- tm_map(myCorpus, content_transformer(tolower))

#remove potentially problematic symbols
toSpace <- content_transformer(function(x, pattern) { return (gsub(pattern, " ", x))})
myCorpus <- tm_map(myCorpus, toSpace, "-")

#Removing other charaters
myCorpus <- tm_map(myCorpus, toSpace, "")
myCorpus <- tm_map(myCorpus, toSpace, "")
myCorpus <- tm_map(myCorpus, toSpace, "-")

# remove URLs
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeURL))


# remove anything other than English letters or space (puntuation and numbers)
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeNumPunct))

#remove stopwords
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))

#define and eliminate all custom stopwords
myStopwords <- c( "RT", "http", "via","can", "get", "new", "look", "one", "amp")
myCorpus <- tm_map(myCorpus, removeWords, myStopwords)

#remove whitespace
myCorpus <- tm_map(myCorpus, stripWhitespace)
myCorpusNEW <- myCorpus

#Stem document
myCorpus <- tm_map(myCorpus,stemDocument)
```

```
stemCompletion2 <- function(x, dictionary)
{
  x <- unlist(strsplit(as.character(x), " "))
  # Unexpectedly, stemCompletion completes an empty string to
  # a word in dictionary. Remove empty string to avoid above issue.
  x <- x[x != ""]
  x <- stemCompletion(x, dictionary=dictionary)
  x <- paste(x, sep="", collapse=" ")
  PlainTextDocument(stripWhitespace(x))
}

myCorpus <- lapply(myCorpus, stemCompletion2, dictionary=myCorpusNEW)
myCorpus <- Corpus(VectorSource(myCorpus))

#Create document-term matrix
#This will create a matrix with word corpus as rows we want them to be in columns
dtm <- DocumentTermMatrix(myCorpus, control = list(wordLengths = c(1, Inf)))

#convert dtm to matrix
m <- as.matrix(dtm)

#write as csv file
write.csv(m, file= "dtmTweet.csv")

#collapse matrix by summing over columns
freq <- colSums(m)

#length should be total number of terms
length(freq)

#create sort order (descending)
ord <- order(freq,decreasing=TRUE)

#List all terms in decreasing order of freq and write to disk
freq[ord]
write.csv(freq[ord],"word_freq.csv")

# Checking for Similarity
cosineSim <- function(x){

  as.dist(x%*%t(x)/(sqrt(rowSums(x^2) %*% t(rowSums(x^2)))))

}

cs <- cosineSim(m)
write.csv(as.matrix(cs),file= "cosimTweet.csv")
```

```
#Adjacency matrix: set entries below a certain threshold to 0.
#Cut off level is arbitrary choice depending on business context

cs[cs < 0.15] <- 0
cs <- round(cs,3)
write.csv(as.matrix(cs),file= "AdjacencyMatrix.csv")


# Lead Extraction
cs_mat = as.matrix(cs)
dimension = dim(cs_mat)
dim = dimension[1]


rm(pfs)
rm(profsim)
profsim = data.frame()

i = 651

for (j in 1:dim){

 if ((i != j) && (cs_mat[i,j] != 0) )

 {
  Source      = dataS4[i,2]
  Destination  = dataS4[j,2]
  Score       = cs_mat[i,j]
  Headline    = dataS4[j,3]
  Employer    = dataS4[j,5]
  Industry    = dataS4[j,7]
  pfs         = data.frame(Source, Destination, Score, Headline, Employer, Industry)
  profsim     = rbind(profsim, pfs)
 }
}

write.csv(profsim, "LeadList_Entity3.csv")

########################################
############
# 2 Profile
###########
########################################

setwd("C:/Users/Ajitesh/Desktop/Documents/RIC/Dissertation/App2/2Sample")
```

```
# Sample of 2 profiles
sample = read.csv("sample.csv")

# Binding the sample with the extracted profile list
dataS2 = rbind(input, sample)

# Creating Corpus of Attributes
dataS2$Corpus = paste(dataS2$Headline, dataS2$Current_Employer, dataS2$Specialties, dataS2$CompInd)

#start preprocessing
library(tm)

#Stating Topic Modelling
# build a corpus, and specify the source to be character vectors

myCorpus <- Corpus(VectorSource(dataS2$Corpus))

# convert to lower case
myCorpus <- tm_map(myCorpus, content_transformer(tolower))

#remove potentially problematic symbols
toSpace <- content_transformer(function(x, pattern) { return (gsub(pattern, " ", x))})
myCorpus <- tm_map(myCorpus, toSpace, "-")

#Removing other charaters
myCorpus <- tm_map(myCorpus, toSpace, '"')
myCorpus <- tm_map(myCorpus, toSpace, '"')
myCorpus <- tm_map(myCorpus, toSpace, "-")

# remove URLs
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeURL))


# remove anything other than English letters or space (puntuation and numbers)
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeNumPunct))

#remove stopwords
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))

#define and eliminate all custom stopwords
myStopwords <- c( "RT", "http", "via","can", "get", "new", "look", "one", "amp")
myCorpus <- tm_map(myCorpus, removeWords, myStopwords)
```

```
#remove whitespace
myCorpus <- tm_map(myCorpus, stripWhitespace)
myCorpusNEW <- myCorpus

#Stem document
myCorpus <- tm_map(myCorpus,stemDocument)

stemCompletion2 <- function(x, dictionary)
{
 x <- unlist(strsplit(as.character(x), " "))
 # Unexpectedly, stemCompletion completes an empty string to
 # a word in dictionary. Remove empty string to avoid above issue.
 x <- x[x != ""]
 x <- stemCompletion(x, dictionary=dictionary)
 x <- paste(x, sep="", collapse=" ")
 PlainTextDocument(stripWhitespace(x))
}

myCorpus <- lapply(myCorpus, stemCompletion2, dictionary=myCorpusNEW)
myCorpus <- Corpus(VectorSource(myCorpus))

#Create document-term matrix
#This will create a matrix with word corpus as rows we want them to be in columns

dtm <- DocumentTermMatrix(myCorpus, control = list(wordLengths = c(1, Inf)))

#convert dtm to matrix
m <- as.matrix(dtm)

#write as csv file
write.csv(m, file= "dtmTweet.csv")

#collapse matrix by summing over columns
freq <- colSums(m)

#length should be total number of terms
length(freq)

#create sort order (descending)
ord <- order(freq,decreasing=TRUE)

#List all terms in decreasing order of freq and write to disk
freq[ord]
write.csv(freq[ord],"word_freq.csv")

# Checking for Similarity
```

```
cosineSim <- function(x){
   as.dist(x%*%t(x)/(sqrt(rowSums(x^2) %*% t(rowSums(x^2)))))
                                    }

cs <- cosineSim(m)
write.csv(as.matrix(cs),file= "cosimTweet.csv")
```

#Adjacency matrix: set entries below a certain threshold to 0. Level is an arbitrary choice

```
cs[cs < 0.15] <- 0
cs <- round(cs,3)
write.csv(as.matrix(cs),file= "AdjacencyMatrix.csv")
```

 # Lead Extraction

```
cs_mat = as.matrix(cs)
dimension = dim(cs_mat)
dim = dimension[1]
```

 # Creating two blank data frame in R for storing the output of loop below.

```
rm(pfs)
rm(profsim)
profsim = data.frame()
```

 # Loop to extract most relevant prospects

```
for (j in 1:dim){
  if ((i != j) && (cs_mat[i,j] != 0) )
  {
    Source           = dataS4[i,2]
    Destination  = dataS4[j,2]
    Score             = cs_mat[i,j]
    Headline       = dataS4[j,3]
    Employer       = dataS4[j,5]
    Industry        = dataS4[j,7]
    pfs          = data.frame(Source, Destination, Score, Headline, Employer, Industry)
    profsim      = rbind(profsim, pfs)
  }
}
```

#Store the output leads generated in a csv file for sharing.

```
write.csv(profsim, "LeadList_Entity2.csv")

# This files has the leads with their scores.
```