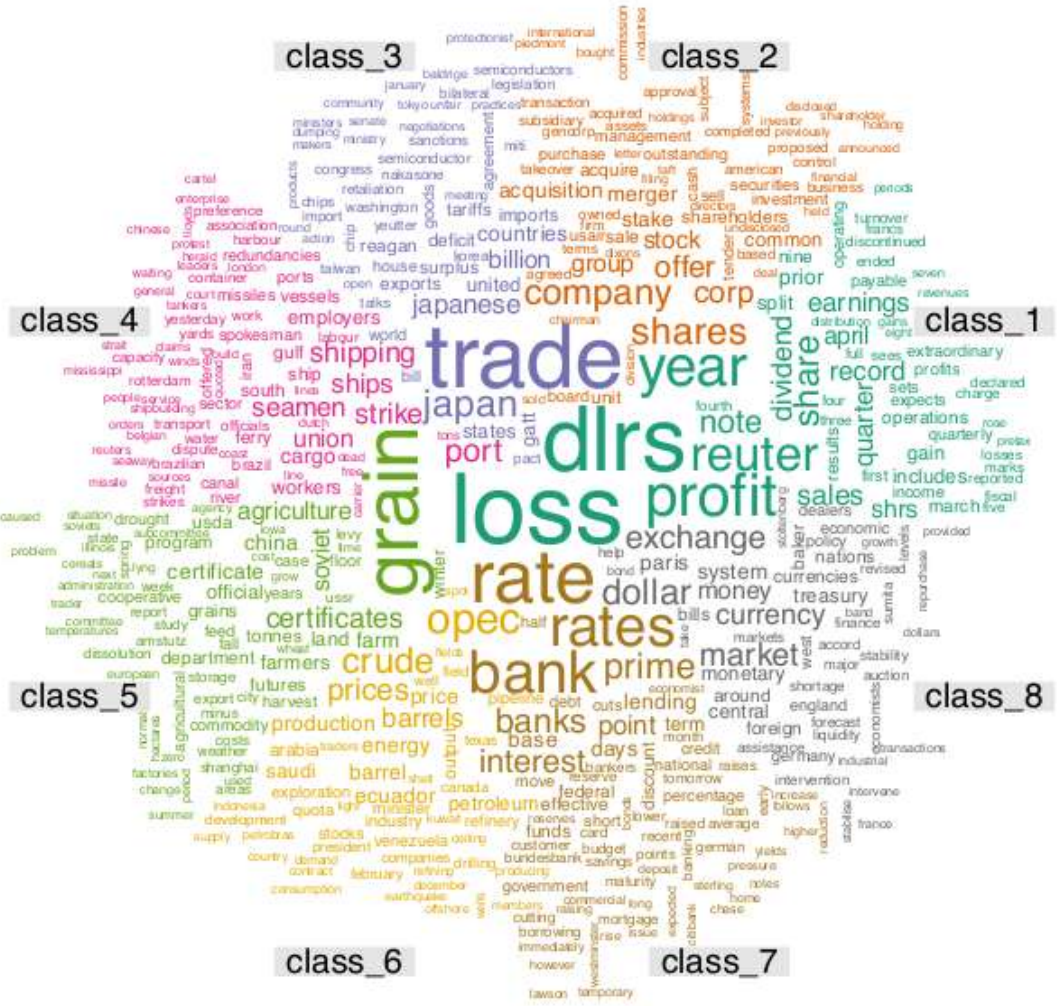


Twitter Data Analysis with R

Text Mining and Social Network Analysis

Siddharth Shukla | 15JE001462 | B. Tech 4th semester. | April , 2017

Acknowledgement

I take this opportunity to express my deep sense of gratitude and respect towards my project guide, **Dr. Rajendra Pamula**, Assistant Professor, Department of Computer Science and Engineering, Indian Institute of Technology (I.S.M), Dhanbad. I am very much indebted to him for the generosity, expertise and guidance that I have received from him while working on this project.

I wish to express profound gratitude to **Prof. Chiranjeev Kumar**, Head of Department, Computer Science and Engineering, Indian Institute of Technology (I.S.M), Dhanbad for his inspiration and guidance.

Date: -

Siddharth Shukla

15JE001462

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY (I.S.M)

DHANBAD – 826004

CERTIFICATE

This is to certify that the report entitled "Twitter Data Analysis with R" submitted by **Siddharth Shukla**, Department of Computer Science and Engineering, Indian Institute of Technology, Dhanbad has successfully completed a project in 4th Semester B. Tech of Academic year 2016-2017.

Prof. Chiranjeev Kumar
HOD
Department of CSE
IIT (ISM), Dhanbad

Dr. Rajendra Pamula
Assistant Professor (Project Guide)
Department of CSE
IIT (ISM), Dhanbad

INTRODUCTION

Text mining is a burgeoning new field that attempts to glean meaningful information from natural language text. It may be loosely characterized as the process of analyzing text to extract information that is useful for particular purposes. Compared with the kind of data stored in databases, text is unstructured, amorphous, and difficult to deal with algorithmically. Nevertheless, in modern culture, text is the most common vehicle for the formal exchange of information. The field of text mining usually deals with texts whose function is the communication of factual information or opinions, and the motivation for trying to extract information from such text automatically is compelling—even if success is only partial.

Text Mining (or Text Analytics) applies analytic tools to learn from collections of text data, like social media, books, newspapers, emails, etc. The goal can be considered to be similar to humans learning by reading such material. However, using automated algorithms we can learn from massive amounts of text, very much more than a human can. The material could consist of millions of newspaper articles to perhaps summarize the main themes and to identify those that are of most interest to particular people. Or we might be monitoring twitter feeds to identify emerging topics that we might need to act upon, as it emerges.

Text mining and data mining

Just as data mining can be loosely described as looking for patterns in data, text mining is about looking for patterns in text. Data mining can be more fully characterized as the extraction of implicit, previously unknown, and potentially useful information from data. The information is implicit in the input data: it is hidden, unknown, and could hardly be extracted without recourse to automatic techniques of data mining. With text mining, however, the information to be extracted is clearly and explicitly stated in the text. It's not hidden at all—most authors go to great pains to make sure that they express themselves clearly and unambiguously—and, from a human point of view, the only sense in which it is "previously unknown" is that human resource restrictions make it infeasible for people to read the text themselves. The problem, of course, is that the information is not couched in a manner that is amenable to automatic processing. Text mining strives to bring it out of the text in a form that is suitable for consumption by computers directly, with no need for a human intermediary. Another requirement that is common to both data and text mining is that the information extracted should be "potentially useful." In one sense,

this means *actionable*—capable of providing a basis for actions to be taken automatically.

Steps in Text Mining

- **Extracting information for human consumption**

We begin with situations in which information mined from text is expressed in a form that is intended for consumption by people rather than computers. The result is not "actionable" in the sense discussed above, and therefore lies on the boundary of what is normally meant by "text mining."

- **Text summarization**

A text summarizer strives to produce a condensed representation of its input, intended for human consumption. It may condense individual documents or groups of documents. Text compression, a related area, also condenses documents, but summarization differs in that its output is intended to be human-readable. The output of text compression algorithms is certainly not human-readable, but neither is it actionable.

- **Document retrieval**

Given a corpus of documents and a user's information need expressed as some sort of query, document retrieval is the task of identifying and returning the most relevant documents. Traditional libraries provide catalogues (whether physical card catalogues or computerized information systems) that allow users to identify documents based on surrogates consisting of *metadata*—salient features of the document such as author, title, subject classification, subject headings, and keywords.

- **Information retrieval**

Information retrieval might be regarded as an extension to document retrieval where the documents that are returned are processed to condense or extract the particular information sought by the user. Thus document retrieval could be followed by a text summarization stage that focuses on the query posed by the user, or an information extraction stage.

- **Assessing document similarity**

Many text mining problems involve assessing the similarity between different documents; for example, assigning documents to pre-defined categories and grouping documents into natural clusters.

- **Text categorization**

Text categorization (or text classification) is the assignment of natural language documents to predefined categories according to their content. The set of categories is often called a "controlled vocabulary." Automatic text categorization has many practical applications, including indexing for document retrieval, automatically extracting metadata, word sense disambiguation by detecting the topics a document covers, and organizing and maintaining large catalogues of Web resources. As in other areas of text mining, until the 1990s text categorization was dominated by *ad hoc* techniques of "knowledge engineering" that sought to elicit categorization rules from human experts and code them into a system that could apply them automatically to new documents. Since then—and particularly in the research community—the dominant approach has been to use techniques of machine learning to infer categories automatically from a training set of pre-classified documents.

- **Document clustering**

Text categorization is a kind of "supervised" learning where the categories are known beforehand and determined in advance for each training document. In contrast, document clustering is "unsupervised" learning in which there is no predefined category or "class," but groups of documents that belong together are sought. For example, document clustering assists in retrieval by creating links between similar documents, which in turn allows related documents to be retrieved once one of the documents has been deemed relevant to a query.

- **Extracting structured information**

An important form of text mining takes the form of a search for structured data inside documents. Ordinary documents are full of structured information: phone numbers, fax numbers, street addresses, email addresses, email signatures, abstracts, tables of contents, lists of references, tables, figures, captions, Web addresses, and more. Machine learning has been applied to the information extraction task by seeking pattern-match rules that extract fillers for slots in the template.

- **Learning rules from text**

Taking information extraction a step further, the extracted information can be used in a subsequent step to learn rules—not rules about how to extract information, but rules that characterize the content of the text itself.

Twitter Data Analysis

- An online social networking service that enables users to send and read short 140-character messages called “tweets".
- Over 300 million monthly active users (as of 2015).
- Creating over 500 million tweets per day.

Techniques

- Text mining
- Topic modelling
- Sentiment analysis
- Social network analysis

Tools

R Studio and its packages:

- twitteR
- tm
- topicmodels
- sentiment140
- igraph

Process

- Extract tweets and followers from the Twitter website with **R** and the **twitteR package.**
- With the **tm package**, clean text by removing punctuations, numbers, hyperlinks and stop words, followed by stemming and stem completion.
- Build a term-document matrix.
- Analyze topics with the **topicmodels package**.
- Analyze sentiment with the **sentiment140 package**.
- Analyze following/followed and retweeting relationships with the **igraph package.**

Basics

The primary package for text mining, **tm**, provides a framework within which we perform our text mining. A collection of other standard R packages add value to the data processing and visualizations for text mining. The basic concept is that of a **corpus**. This is a collection of texts, usually stored electronically, and from which we perform our analysis. A corpus might be a collection of news articles from or the published works of Shakespeare. Within each corpus we will have separate documents, which might be articles, stories, or book volumes. Each document is treated as a separate entity or record. Documents which we wish to analyze come in different formats. Quite a few formats are supported by tm package. The supported formats include text, PDF, Microsoft Word, and XML.

We will be performing:

- Extracting Tweets
- Text Cleaning
- Frequent Words and Word Cloud
- Word Associations
- Sentiment Analysis
- Social Network Analysis
- Retweeting Analysis

1. Retrieve Tweets

Console:

```
> library(SnowballC)
> library(twitteR)
> getwd()
[1] "C:/Users/Siddharth Shukla/Documents"
> setwd("~/work/datasciencecoursera/RDM/")
> twitterF <- "rdmTweets-201306.RData"
> if(!file.exists(twitterF)){
+   url <- "http://www.rdatamining.com/data/rdmTweets.RData"
+   download.file(url, destfile =twitterF)
+   url <- "http://www.rdatamining.com/data/rdmTweets-201306.RData"
+   download.file(url, destfile =twitterF)
+ }
>
> load(file = twitterF)
> (n.tweet <- length(tweets))
[1] 320
> tweets[1:5]
[[1]]
[1] "RDataMining: Examples on calling Java code from R \nhttp://t.co/Yg
1AivsO1R"

[[2]]
[1] "RDataMining: Simulating Map-Reduce in R for Big Data Analysis Usin
g Flights Data http://t.co/uIAh6PgvQv via @rbloggers"

[[3]]
[1] "RDataMining: Job opportunity: Senior Analyst - Big Data at Wesfarm
ers Industrial &amp;amp; Safety - Sydney Area, Australia #jobs http://t.co/
gXogcvR4XT"

[[4]]
[1] "RDataMining: CLAVIN: an open source software package for document
geotagging and geoparsing http://t.co/gTGbTanKCI"

[[5]]
[1] "RDataMining: An online book on Natural Language Processing (with P
ython) http://t.co/5j31FhtrA6"
```

The tweets are then converted to a data frame using the function **twListToDF()** which summarizes information about a list of tweets in a data frame and then to a **corpus** using the function **Corpus(), which is a collection of text documents.** After that, the corpus can be processed with functions provided in package tm.

```
> # convert tweets to a data frame
> tweets.df <- twListToDF(tweets)
> dim(tweets.df)
[1] 320  14
```

2. Text Cleaning:

Corpus is a collection of texts, usually stored electronically, and from which we perf orm our analysis. A corpus might be a collection of news articles from or the publis hed works of Shakespeare. Within each corpus we will have separate documents, wh ich might be articles, stories, or book volumes. Each document is treated as a separa te entity or record.

Console:

```
> library(tm)
>
> # build a corpus, and specify the source to be character vectors
> myCorpus <- Corpus(VectorSource(tweets.df$text))
> # convert to lower case # myCorpus <- tm_map(myCorpus, tolower)
> # tm v0.6
> myCorpus <- tm_map(myCorpus, content_transformer(tolower))
> # remove punctuation
> myCorpus <- tm_map(myCorpus, removePunctuation)
> # remove numbers
> myCorpus <- tm_map(myCorpus, removeNumbers)
> # remove URLs
> removeURL <- function(x) gsub("http[[:alnum:]]*", "", x)
> ### myCorpus <- tm_map(myCorpus, removeURL, lazy=TRUE)
> myCorpus <- tm_map(myCorpus, content_transformer(removeURL))  #??
> # add two extra stop words: 'available' and 'via'
> myStopwords <- c(stopwords("english"), "available", "via")
> # remove 'r' and 'big' from stopwords
> myStopwords <- setdiff(myStopwords, c("r", "big"))
> # remove stopwords from corpus
> myCorpus <- tm_map(myCorpus, removeWords, myStopwords)
> #
> ## keep a copy of corpus to use later as a dictionary for stem
> # completion
> myCorpusCopy <- myCorpus
```

The corpus needs a couple of transformations, including changing letters to lowercase, and removing punctuations, numbers and stop words. Hyperlinks are also removed. This is done using the function **tm_map()** which is an interface to apply transformation functions (also denoted as mappings) to corpus. Changing case, removing punctuations, removing numbers, removing words and stopwords etc. can be done by passing corresponding parameters in the above mentioned function.

3. Stemming and Stem Completion:

Stemming is the process of removing suffixes from words to get the common origin. In statistical analysis, it greatly helps when comparing texts to be able to identify words with a common meaning and form as being identical. For example, we would like to count the words "stopped" and "stopping" as being the same and derived from "stop". Stemming identifies these common forms. **Word stemming can be done with the snowball stemmer, which requires packages Snowball.**

Console:

```
# stem words
> myCorpus <- tm_map(myCorpus, stemDocument)
>
> # inspect the first 5 documents (tweets) inspect(myCorpus[1:5])
> # The code below is used for to make text fit for paper width
> for (i in 1:5) {
+   cat(paste("[[", i, "]] ", sep = ""))
+   #writeLines(myCorpus[[i]])
+   writeLines(as.character(myCorpus[[i]]))
+ }
[[1]] exampl call java code r
[[2]] simul mapreduc r big data analysi use flight data rblogger
[[3]] job opportun senior analyst big data wesfarm industri amp safeti
sydney area australia job
[[4]] clavin open sourc softwar packag document geotag geopars
[[5]] onlin book natur languag process python
> tdm <- TermDocumentMatrix(myCorpus, control = list(wordLengths = c(1,
Inf)))
> tdm
<<TermDocumentMatrix (terms: 865, documents: 320)>>
Non-/sparse entries: 2506/274294
Sparsity           : 99%
Maximal term length: 25
Weighting          : term frequency (tf)
> idx <- which(dimnames(tdm)$Terms == "r")
> inspect(tdm[idx + (0:5), 101:110])
<<TermDocumentMatrix (terms: 6, documents: 10)>>
Non-/sparse entries: 8/52
Sparsity           : 87%
Maximal term length: 8
Weighting          : term frequency (tf)
Sample             :
          Docs
Terms      101 102 103 104 105 106 107 108 109 110
  analysi    0   0   0   0   0   0   0   0   0   0
  big        0   1   0   0   0   0   0   0   0   0
  data       0   1   0   0   1   0   1   0   0   0
  flight     0   0   0   0   0   0   0   0   0   0
```

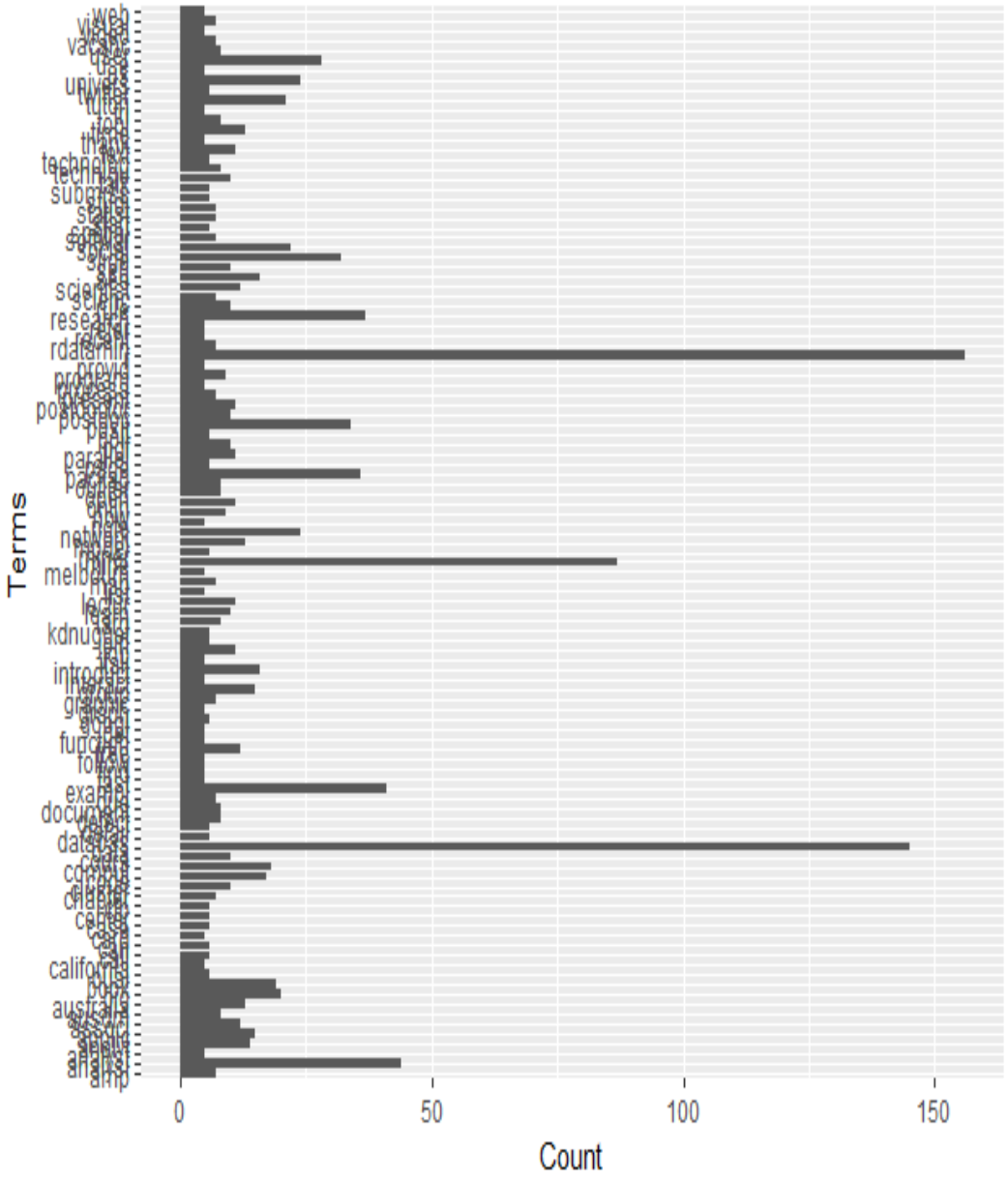
```
mapreduc   0   0   0   0   0   0   0   0   0   0
r          0   1   1   0   0   0   0   0   1   1
```

In above code, stemming is done again by passing required parameters to the interfa ce function **tm_map(). cat()** function is used to concatenate [[i]] to ith tweet. **write Lines()** is used to print tweets from the corpus containing stemmed words. Then a T erm Document Matrix is built using **TermDocumentMatrix()**.

4. Inspecting Frequent Words:

Console:

```
#inspect frequent words
> (freq.terms <- findFreqTerms(tdm, lowfreq=15))
 [1] "code"      "exampl"    "r"         "analysi"   "big"       "data"
"use"
 [8] "packag"    "book"      "introduct" "mine"      "network"   "slide
"     "social"
[15] "see"       "comput"    "group"     "applic"    "research"  "posit
"     "tutori"
[22] "univers"
>
> term.freq <- rowSums(as.matrix(tdm))
> term.freq <- subset(term.freq, term.freq >=5)
> df <- data.frame(term = names(term.freq), freq = term.freq)
> library(ggplot2)
> ggplot(df, aes(x=term, y=freq)) + geom_bar(stat = "identity") + xlab(
"Terms") + ylab("Count") +coord_flip()
> findAssocs(tdm, "rdatamin", 0.2)
$rdatamin
              group              produc            linkedin          co
mprehens
               0.41                0.37                0.37
0.37
grouprdataminingcom           subdomain                swap
two
               0.37                0.37                0.37
0.37
             follow               share             extract
blog
               0.27                0.26                0.26
0.26
            project                 map               tweet
creat
               0.21                0.21                0.21
0.21
              build               visit
               0.21                0.21
```



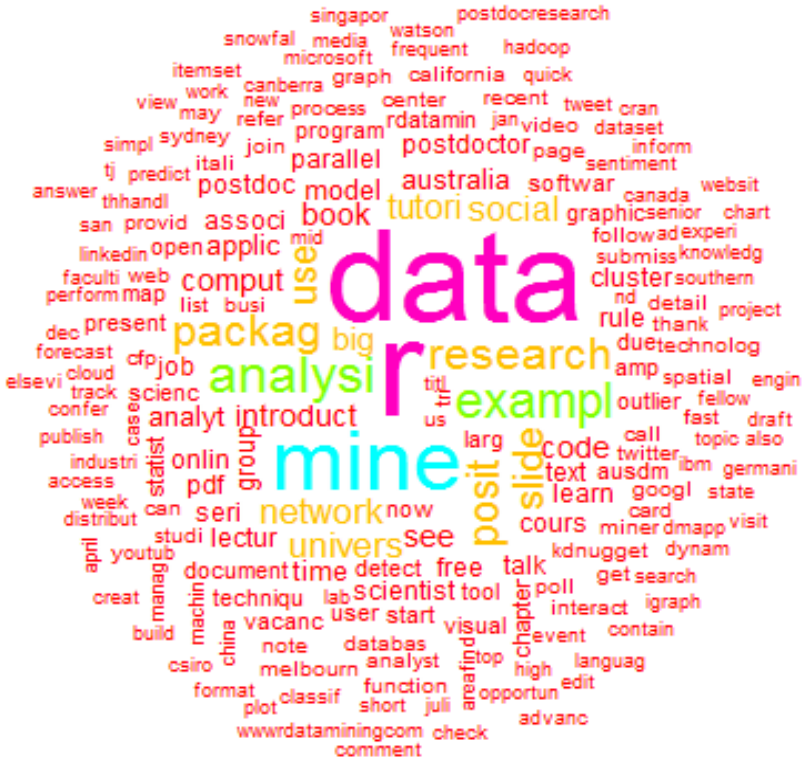
Firstly, frequent words in a document-term matrix are found using **findFreqTerms()**. **findAssocs()** is used to find associations/correlations in a document-term matrix. The value 0.2 as parameter in the **findAssocs()** function gives the lower limit of the correlation coefficient. On precise and enlarged view of the plot it is observed that the word "r" has the maximum frequency, represented by the longest horizontal bar in the plot. It is followed by "data", which in turn is followed by "mining", as is found by observation.

5. Wordcloud:

We can generate a word cloud as an effective alternative to providing a quick visual overview of the frequency of words in a corpus. We can show the importance of words with a word cloud (also known as a tag cloud), which can be easily produced with package wordcloud. The more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud. With **wordcloud( ),** the first two parameters give a list of words and their frequencies. Words with frequency below three are not plotted, as specified by **min.freq=3**. By setting **random.order=F**, frequent words are plotted first, which makes them appear in the center of cloud. We also set the colors to gray levels based on frequency. Firstly, the words are sorted in decreasing order using **sort()** so that words with maximum frequency are placed first.

Console:

```
> library(wordcloud)
> m <- as.matrix(tdm)
> # calculate the frequency of words and sort it by frequency
> word.freq <- sort(rowSums(m), decreasing = T)
> wordcloud(words = names(word.freq), freq = word.freq, min.freq = 3,
+           random.order = F,colors = rainbow(6))
```



The above word cloud clearly shows again that “r", “data" and “mine" are the top three words, which validates that the @RDataMining tweets present information on R and data mining.

6. Clustering Words:

We then try to and clusters of words with hierarchical clustering. Point in the same cluster are similar to each other while those in different clusters are dissimilar. Sparse terms are removed using **removeSparseTerms()**, so that the plot of clustering will not be crowded with words. Then the distances between terms are calculated with **dist()** after scaling. After that, the terms are clustered with **hclust()** and the dendrogram is cut into 6 clusters. The agglomeration method is set to ward, which denotes the increase in variance when two clusters are merged. Each cluster has a well-defined centroid which represents each cluster.

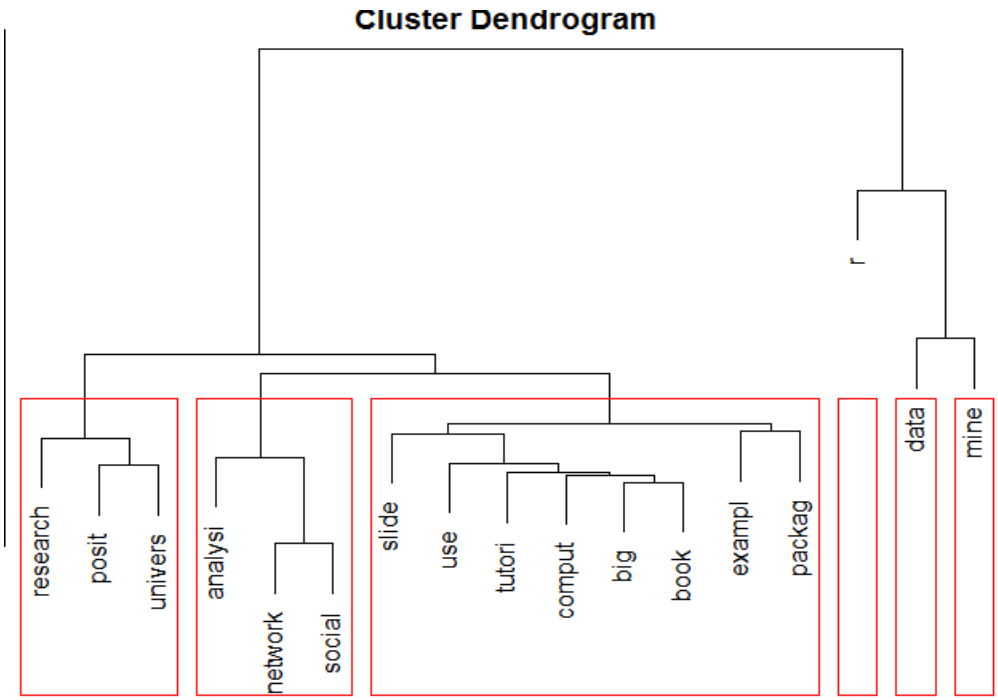
- **Clustering by k-means algorithm**

Console:

```
> #clustering using k-means
>
> # remove sparse terms
> tdm2 <- removeSparseTerms(tdm, sparse = 0.95)
> m2 <- as.matrix(tdm2)
> # cluster terms
> distMatrix <- dist(scale(m2))
> fit <- hclust(distMatrix, method = "ward.D")
>
> plot(fit)
> rect.hclust(fit, k = 6) # cut tree into 6 clusters
>
> m3 <- t(m2) # transpose the matrix to cluster documents (tweets)
> set.seed(122) # set a fixed random seed
> k <- 6 # number of clusters
> kmeansResult <- kmeans(m3, k)
> round(kmeansResult$centers, digits = 3) # cluster centers
  exampl     r analysi   big  data   use packag  book  mine network sli
de social
1  0.044 0.191   0.132 0.147 1.044 0.015  0.015 0.029 0.382   0.000 0.0
88  0.000
2  0.216 1.000   0.027 0.135 1.459 0.243  0.216 0.243 1.054   0.000 0.0
54  0.000
3  0.095 0.286   0.857 0.000 0.048 0.095  0.095 0.000 0.095   0.952 0.0
95  0.810
4  0.065 0.000   0.078 0.013 0.000 0.052  0.117 0.052 0.104   0.013 0.1
04  0.013
5  0.274 1.190   0.083 0.000 0.024 0.143  0.190 0.048 0.107   0.036 0.1
67  0.000
6  0.000 0.000   0.091 0.121 0.515 0.000  0.000 0.000 0.091   0.000 0.0
00  0.121
  comput research posit tutori univers
```

```
1  0.059    0.029 0.074  0.074   0.029
2  0.027    0.027 0.000  0.000   0.000
3  0.000    0.048 0.190  0.190   0.048
4  0.052    0.013 0.039  0.052   0.091
5  0.107    0.000 0.000  0.095   0.000
6  0.000    0.970 0.667  0.000   0.424

> for (i in 1:k) {
+   cat(paste("cluster ", i, ": ", sep=""))
+    s <- sort(kmeansResult$centers[i,], decreasing=T)
+    cat(names(s)[1:3], "\n") }
cluster 1: data mine r
cluster 2: data mine r
cluster 3: network analysi social
cluster 4: packag mine slide
cluster 5: r exampl packag
cluster 6: research posit data
```



In the above dendrogram, we can see the topics in the tweets. Words “analysis", “network" and “social" are clustered into one group, because there are a couple of tweets on social network analysis. The second cluster from left comprises “positions", “postdoctoral" and “research", and they are clustered into one group because of tweets on vacancies of research and postdoctoral positions.

- **Clustering by k-medoids algorithm**

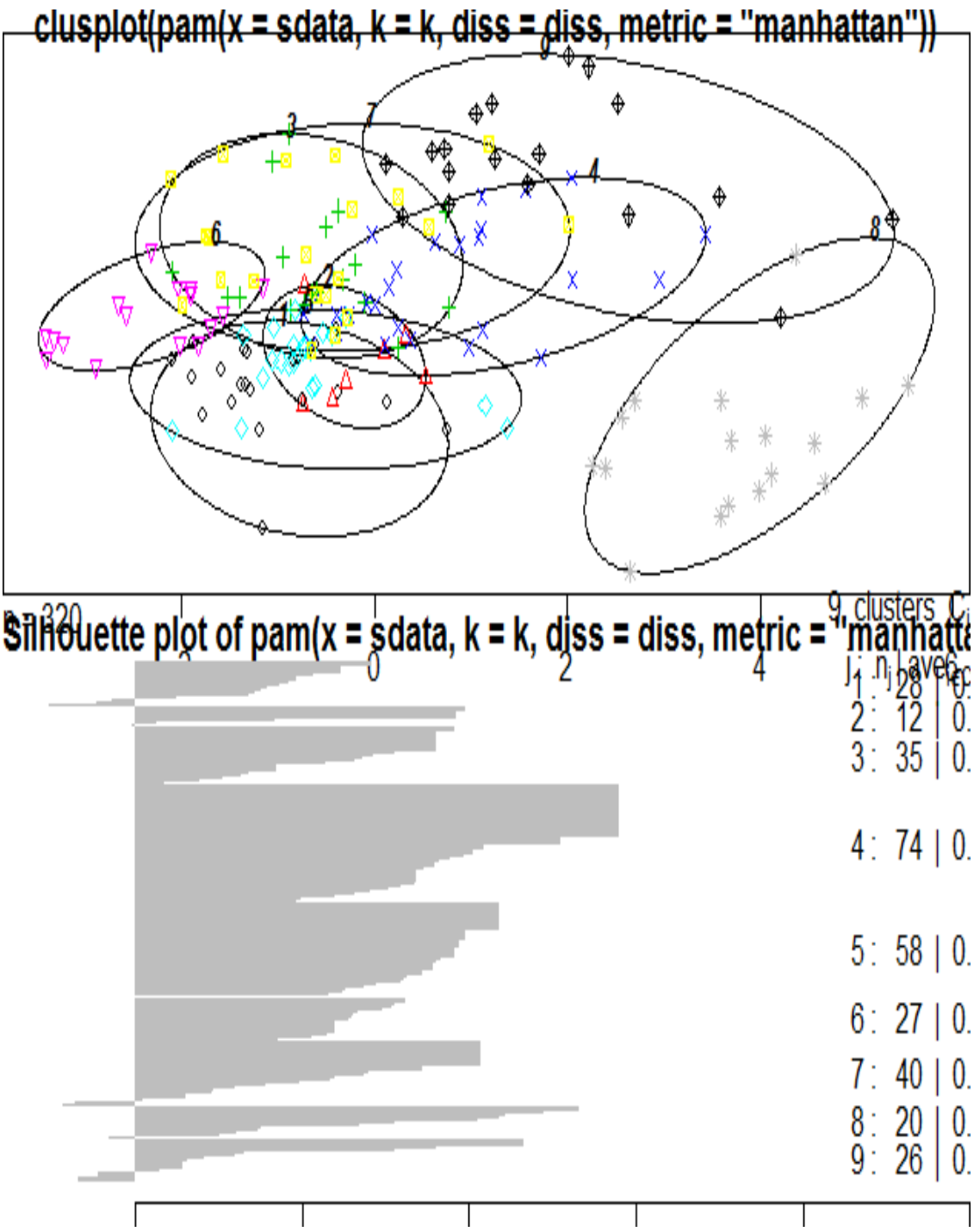
We then try k-medoids clustering with the Partitioning Around Medoids (PAM) algorithm, which uses medoids (representative objects) instead of means to represent

clusters. It is more robust to noise and outliers than k-means clustering, and provides a display of the silhouette plot to show the quality of clustering. We use function **pamk()** from **package fpc**, which calls the function **pam()** with the number of clusters estimated by optimum average silhouette.

Console:

```
> library(fpc)
> pamResult <- pamk(m3, metric = "manhattan")
> k <- pamResult$nc
> pamResult <- pamResult$pamobject
>  # print cluster medoids
>  for (i in 1:k) {
+  cat(paste("cluster", i, ": "))
+  cat(colnames(pamResult$medoids)[which(pamResult$medoids[i,]==1)], "\
n")}
cluster 1 : exampl r
cluster 2 : r analysi data
cluster 3 : data
cluster 4 :
cluster 5 : r
cluster 6 : r data mine
cluster 7 : data mine
cluster 8 : analysi network social
cluster 9 : data research posit
> # plot clustering result
>  layout(matrix(c(1,2),2,1)) # set to two graphs per page
> plot(pamResult, color=F, labels=4, lines=0, cex=.8, col.clus=1,
+        col.p=pamResult$clustering)
> layout(matrix(1)) # change back to one graph per page
```

The first chart is a 2D “clusplot" (clustering plot) of the k clusters, and the second one shows their silhouettes. With the silhouette, a large si (almost 1) suggests that the corresponding observations are very well clustered, a small si (around 0) means that the observation lies between two clusters, and observations with a negative si are probably placed in the wrong cluster. The average silhouette width is 0.29, which suggests that the clusters are not well separated from one another. The above results and Figure 10.4 show that there are nine clusters of tweets. Clusters 1, 2, 3, 5 and 9 are well separated groups, with each of them focusing on a specific topic. Cluster 7 is composed of tweets not fitted well into other clusters, and it overlaps all other clusters. There is also a big overlap between cluster 6 and 8, which is understandable from their medoids. Some observations in cluster 8 are of negative silhouette width, which means that they may fit better in other clusters than cluster 8.


clusplot(pam(x = sdata, k = k, diss = diss, metric = "manhattan"))
9
3
7
4
6
8
2
1
-320
0
2
4
9 clusters C
Silhouette plot of pam(x = sdata, k = k, diss = diss, metric = "manhatta
j : n_j | ave
1 : 28 | 0.
2 : 12 | 0.
3 : 35 | 0.
4 : 74 | 0.
5 : 58 | 0.
6 : 27 | 0.
7 : 40 | 0.
8 : 20 | 0.
9 : 26 | 0.

Social Network Analysis

The data analyzed is Twitter text data used earlier. Putting it in a general scenario of social networks, the terms can be taken as people and the tweets as groups on LinkedIn1, and the term-document matrix can then be taken as the group membership of people. **I will first build a network of terms based on their co-occurrence in the same tweets, and then build a network of tweets based on the terms shared by them. We will build a network of terms based on their co-occurrence in tweets. After that, it is transformed into a term-term adjacency matrix using %*% operator, based on which a graph is built. Then we plot the graph to show the relationship between frequent terms, and also make the graph more readable by setting colors, font sizes and transparency of vertices and edges.**

Console:

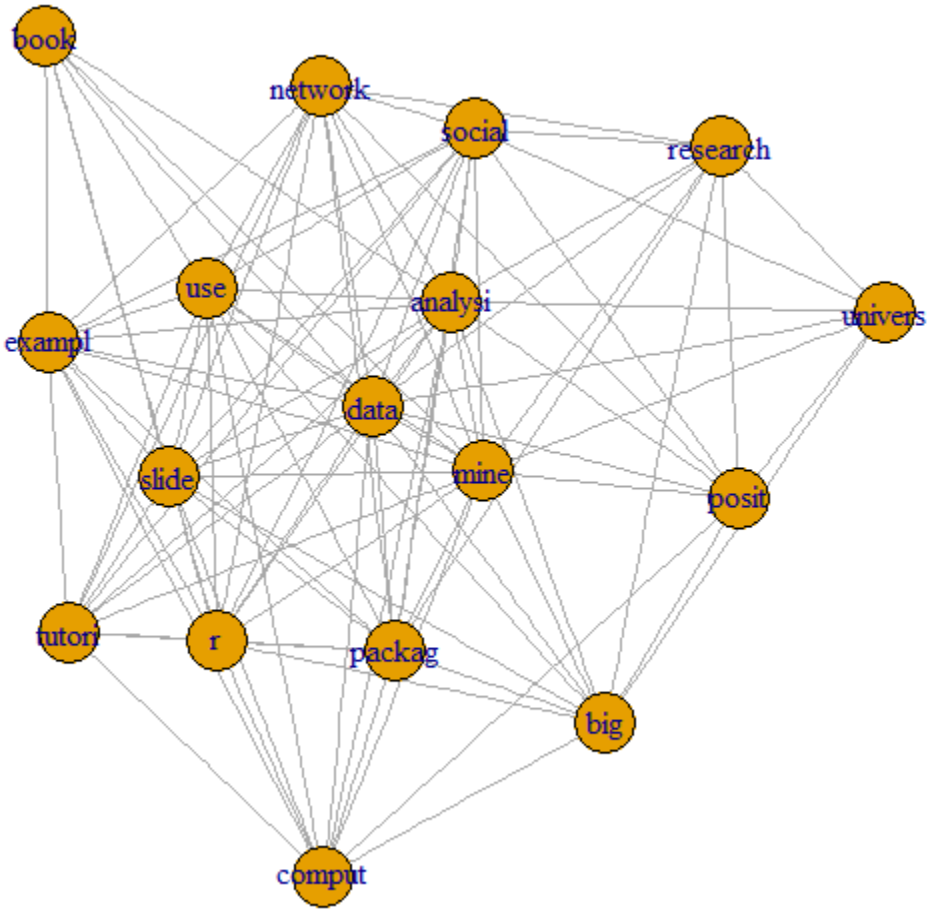
```
> # SOCIAL NETWORK ANALYSIS
> termDocMatrix<-m2
> termDocMatrix[5:10,1:20]
         Docs
Terms     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
  data    0 2 1 0 0 0 1 0 0  0  1  1  0  1  0  3  0  0  0  0
  use     0 1 0 0 0 0 0 0 1  0  0  0  0  0  0  1  0  0  0  1
  packag  0 0 0 1 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
  book    0 0 0 0 1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
  mine    0 0 0 0 0 0 1 0 0  0  0  0  0  0  0  1  0  0  0  0
  network 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  1  0  0
> # change it to a Boolean matrix
> termDocMatrix[termDocMatrix>=1] <- 1
> # transform into a term-term adjacency matrix
> termMatrix <- termDocMatrix %*% t(termDocMatrix)
> termMatrix[5:10,5:10]
         Terms
Terms     data use packag book mine network
  data     124  10      9   10   60       1
  use       10  27      8    1    8       2
  packag     9   8     34    0    6       2
  book      10   1      0   18   11       0
  mine      60   8      6   11   81       1
  network    1   2      2    0    1      24
```

Now I have built a term-term adjacency matrix, where the rows and columns represent terms, and every entry is the number of concurrences of two terms. Next I can build a graph with **graph.adjacency()** from **package igraph.**

Console:

```
> library(igraph)
> # build a graph from the above matrix
> g <- graph.adjacency(termMatrix, weighted=T, mode = "undirected")
> # remove loops
```

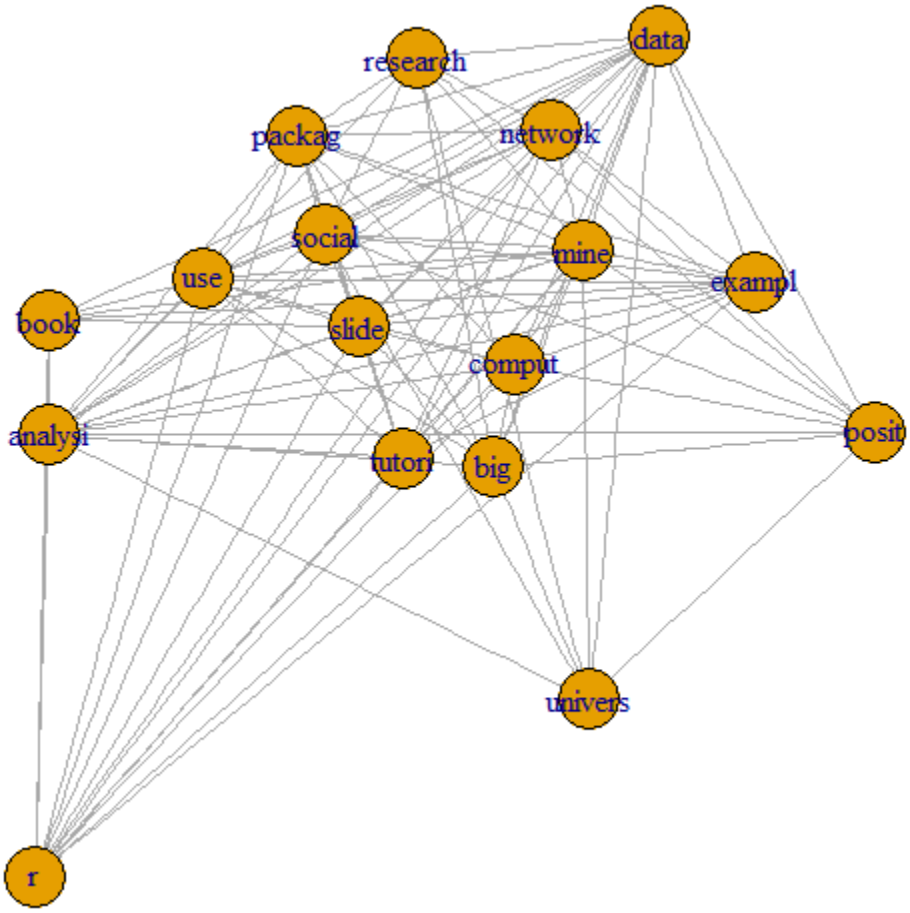
```
> g <- simplify(g)
> # set labels and degrees of vertices
> V(g)$label <- V(g)$name
> V(g)$degree <- degree(g)
> # set seed to make the layout reproducible
> set.seed(3952)
> layout1 <- layout.fruchterman.reingold(g)
> plot(g, layout=layout1)
```



A different layout can be generated with the first line of code below. The second line produces an interactive plot, which allows me to manually rearrange the layout.

Console:

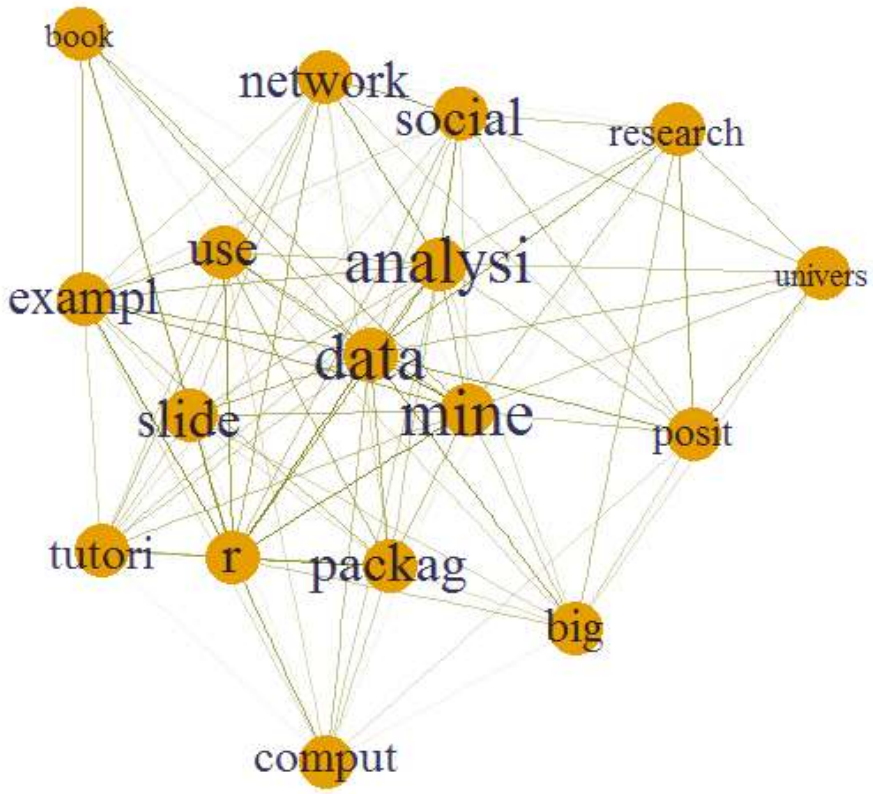
```
> plot(g, layout=layout.kamada.kawai)
> tkplot(g, layout=layout.kamada.kawai)
[1] 1
```



Next, I will set the label size of vertices based on their degrees, to make important terms stand out. Similarly, I will also set the width and transparency of edges based on their weights. This is useful in applications where graphs are crowded with many vertices and edges. In the code below, **the vertices and edges are accessed with V() and E(). Function rgb(red, green, blue, alpha) defines a color, with an alpha transparency.**

Console:

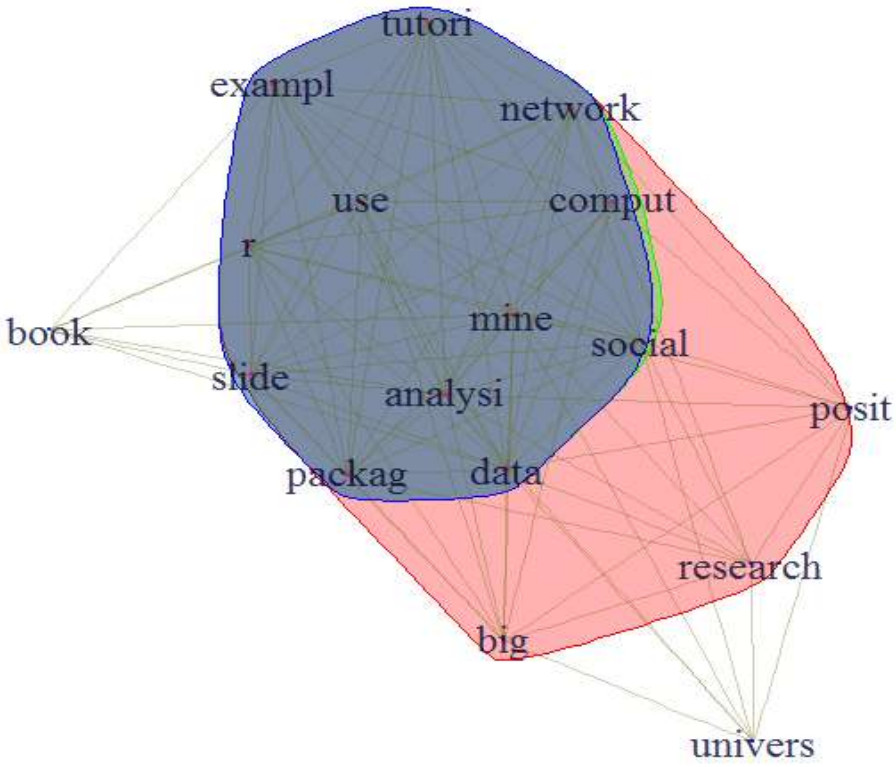
```
> V(g)$label.cex <- 2.2 * V(g)$degree / max(V(g)$degree)+ .2
> V(g)$label.color <- rgb(0, 0, .2, .8)
> V(g)$frame.color <- NA
> egam <- (log(E(g)$weight)+.4) / max(log(E(g)$weight)+.4)
> E(g)$color <- rgb(.5, .5, 0, egam)
> E(g)$width <- egam
> # plot the graph in layout1
> plot(g, layout=layout1)
```



Next, we try to detect communities from the graph by plotting cohesive blocks using **cohesive.blocks()** function. Cohesive blocking is a method of determining hierarchical subsets of graph vertices based on their vertex connectivity.

Console:

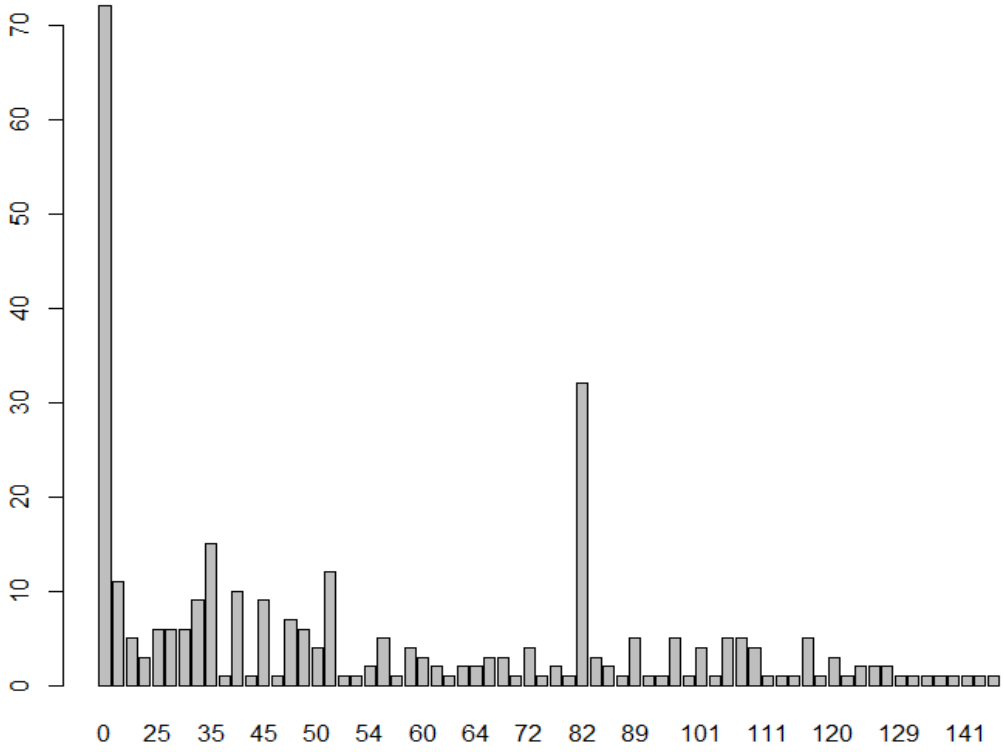
```
> blocks <- cohesive.blocks(g)
> blocks
Cohesive block structure:
B-1            c  7, n 17
'- B-2         c  8, n 15   ooooooo.oo oooooo.
   '- B-3      c  9, n 12   ooo.ooo.oo ooo..o.
      '- B-4   c 10, n 11   ooo.ooo.oo oo...o.
> plot(blocks, g, vertex.size=.3, vertex.label.cex=1.5, edge.color=rgb(
.4,.4,0,.3))
```



We can also build a graph of tweets base on the number of terms that they have in common. **Because most tweets contain one or more words from "r", "data" and "mining", most tweets are connected with others and the graph of tweets is very crowded. To simplify the graph and find relationship between tweets beyond the above three keywords, we remove the three words before building a graph.**

Console:

```
> # remove "r", "data" and "mining"
> idx<-which(dimnames(termDocMatrix)$Terms %in% c("r", "data", "mining"
))
> M <- termDocMatrix[-idx,]
> # build a tweet-tweet adjacency matrix
> tweetMatrix <- t(M) %*% M
> library(igraph)
> g <- graph.adjacency(tweetMatrix, weighted=T, mode = "undirected")
> V(g)$degree <- degree(g)
> g <- simplify(g)
> # set labels of vertices to tweet IDs
> V(g)$label <- V(g)$name
> V(g)$label.cex <- 1
> V(g)$label.color <- rgb(.4, 0, 0, .7)
> V(g)$size <- 2
> V(g)$frame.color <- NA
> barplot(table(V(g)$degree))
```



It can be seen that there are around 40 isolated vertices (with a degree of zero). Note that most of them are caused by the removal of the three keywords, “r", “data" and “mining".

Now I set vertex colors based on degree, and set labels of isolated vertices to tweet IDs and the first 20 characters of every tweet. The labels of other vertices are set to tweet IDs only, so that the graph will not be overcrowded with labels. I will also set the color and width of edges based on their weights.

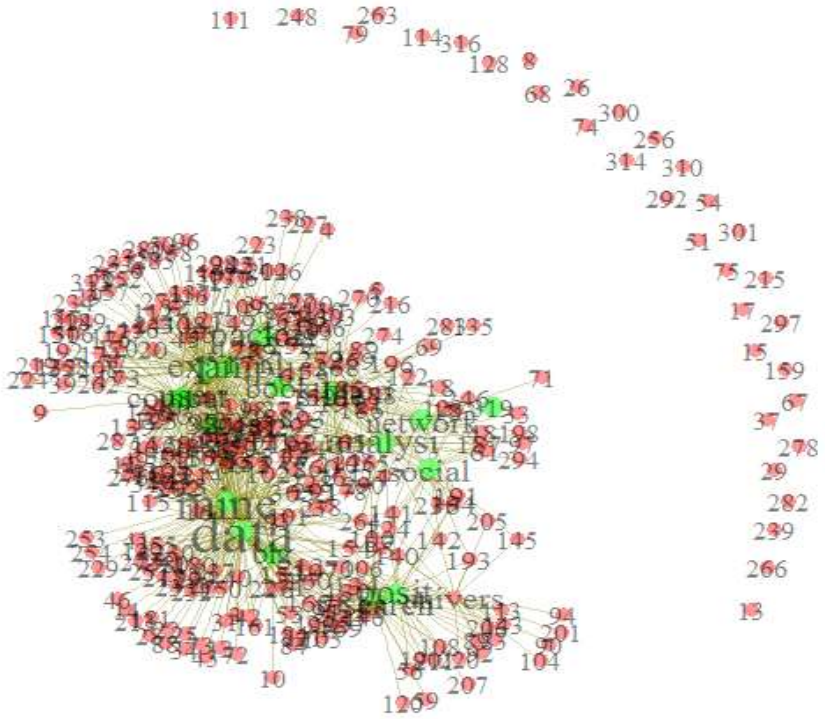
Console:

```
> idx <- V(g)$degree == 0
> V(g)$label.color[idx] <- rgb(0, 0, .3, .7)
> # set labels to the IDs and the first 10 characters of tweets
> V(g)$label[idx] <- paste(V(g)$name[idx], substr(df$text[idx], 1, 20),
sep=": ")
> egam <- (log(E(g)$weight)+.2) / max(log(E(g)$weight)+.2)
> E(g)$color <- rgb(.5, .5, 0, egam)
> E(g)$width <- egam
> set.seed(3152)
> layout2 <- layout.fruchterman.reingold(g)
> plot(g, layout=layout2)
```

I remove the isolated vertices from graph with function **delete.vertices()** and re-plot the graph. Similarly, I can also remove edges with low degrees using **delete.edges()** to simplify the graph. Below with function **delete.edges(),** I remove edges which have

weight of one. After removing edges, some vertices become isolated and are also removed.

```
> g3 <- delete.edges(g, E(g)[E(g)$weight <= 1])
> g3 <- delete.vertices(g3, V(g3)[degree(g3) == 0])
> plot(g3, layout=layout.fruchterman.reingold)
> # create a graph
> g <- graph.incidence(termDocMatrix, mode=c("all"))
> # get index for term vertices and tweet vertices
> nTerms <- nrow(M)
> nDocs <- ncol(M)
> idx.terms <- 1:nTerms
> idx.docs <- (nTerms+1):(nTerms+nDocs)
> # set colors and sizes for vertices
> V(g)$degree <- degree(g)
> V(g)$color[idx.terms] <- rgb(0, 1, 0, .5)
> V(g)$color[idx.docs] <- rgb(1, 0, 0, .4)
> V(g)$size[idx.docs] <- 4
> V(g)$frame.color <- NA
> # set vertex labels and their colors and sizes
> V(g)$label <- V(g)$name
> V(g)$label.color <- rgb(0, 0, 0, 0.5)
> V(g)$label.cex <- 1.4*V(g)$degree/max(V(g)$degree) + 1
> E(g)$width <- .3
> E(g)$color <- rgb(.5, .5, 0, .3)
> set.seed(958)#5365, 227
> plot(g, layout=layout.fruchterman.reingold)
```



Sentiment Analysis:

Sentiment analysis deals with identifying and classifying opinions or sentiments expressed in source text. Social media is generating a vast amount of sentiment rich data in the form of tweets, status updates, blog posts etc. **Sentiment analysis of this user generated data is very useful in knowing the opinion of the crowd.** Twitter sentiment analysis is difficult compared to general sentiment analysis due to the presence of slang words and misspellings. The maximum limit of characters that are allowed in Twitter is 140. Knowledge base approach and Machine learning approach are the two strategies used for analyzing sentiments from the text. Firstly, **sentiment140 and okugami79 packages** are loaded into current session. **Sentiment text analysis tool in R uses sentiment140 web service engine through internet, provides instant negative, positive, neutral text expression recognition, tune to Twitter text analysis.**

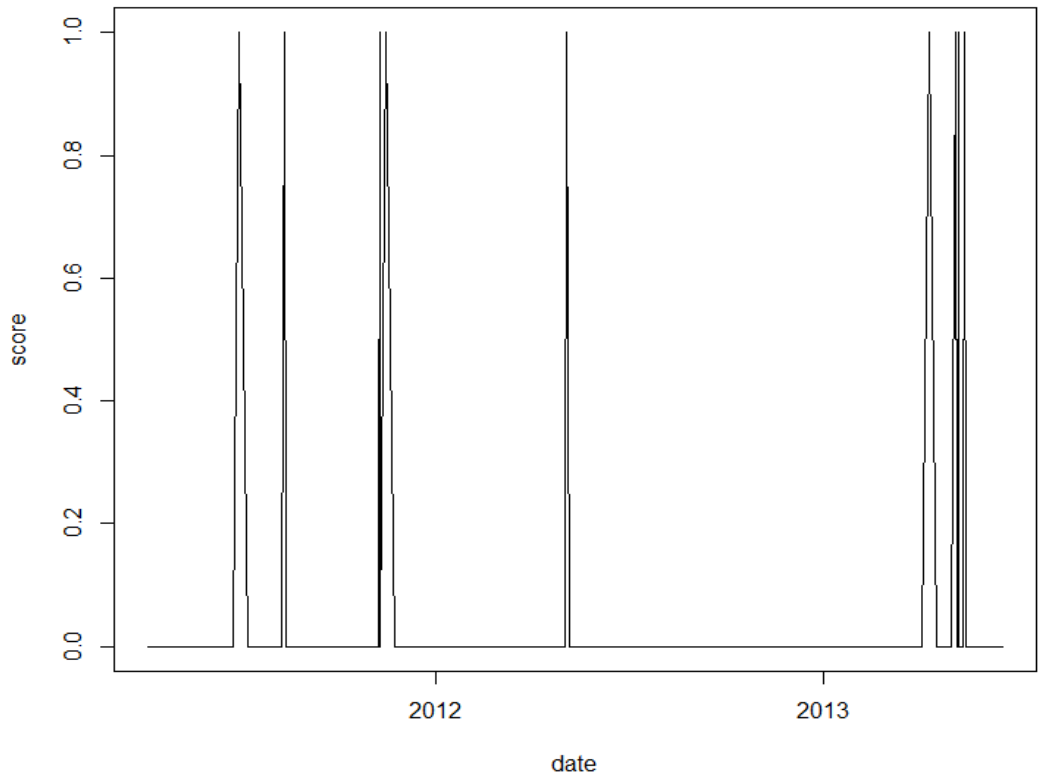
Console:

```
> #sentiment analysis

> require(devtools)

> install_github("sentiment140", "okugami79")

> library(sentiment)
> sentiments <- sentiment(tweets.df$text)
> table(sentiments$polarity)
 neutral positive
     311        9
> library(data.table)
> sentiments$score <- 0
> sentiments$score[sentiments$polarity == "positive"] <- 1
> sentiments$score[sentiments$polarity == "negative"] <- -1
> sentiments$date <- as.IDate(tweets.df$created)
> result <- aggregate(score ~ date, data = sentiments, sum)
> plot(result, type = "l")
```



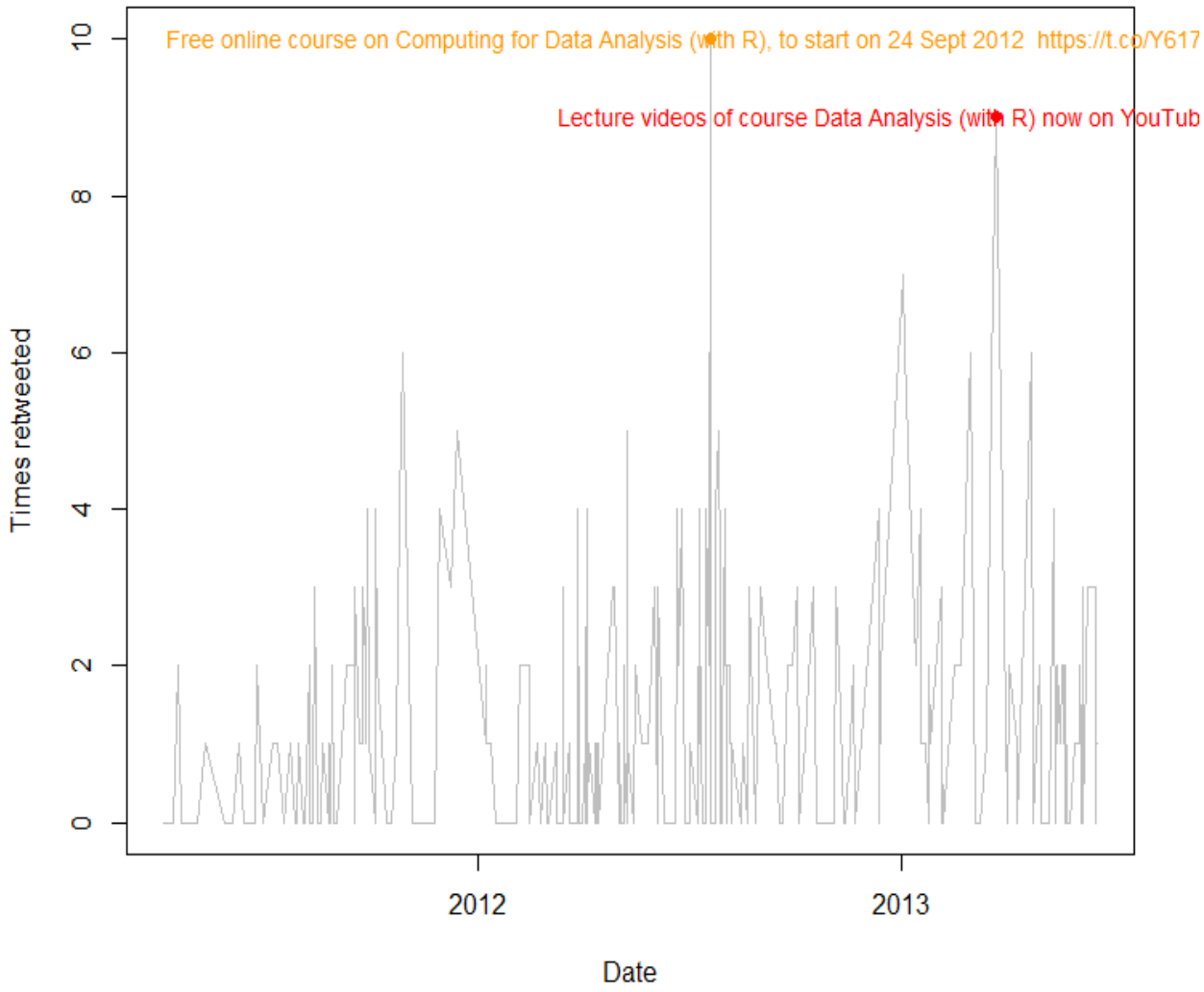
Retweets:

The retweets are counted using **retweetCount attribute** that ever tweet has. The function **strptime()** is used to convert between character representations that represent calendar dates and times.

Console:

```
> #top retweeted tweets
> table(tweets.df$retweetCount)
  0   1   2   3   4   5   6   7   9   10
163  64  42  22  18   3   4   2   1   1
> selected <- which(tweets.df$retweetCount >= 9)
> # plotting them
> dates <- strptime(tweets.df$created, format="%Y-%m-%d")
> plot(x=dates, y=tweets.df$retweetCount, type="l", col="grey",
+      xlab="Date", ylab="Times retweeted")
> colors <- rainbow(10)[1:length(selected)]
> points(dates[selected], tweets.df$retweetCount[selected],
+        pch=19, col=colors)
> text(dates[selected], tweets.df$retweetCount[selected],
```

```
+      tweets.df$text[selected], col=colors, cex=.9)
```



We observe from the retweet count table that only 2 tweets have number of retweets greater than 9, which are denoted by point at the peak with the tweet in the plot as depicted above.

Synopsis

In today's world, the amount of stored information has been enormously increasing day by day which is generally in the unstructured form and cannot be used for any processing to extract useful information, so several techniques such as summarization, classification, clustering, information extraction and visualization are available for the same which comes under the category of text mining. Text Mining can be defined as a technique which is used to extract interesting information or knowledge from the text documents. Text mining, also known as text data mining or knowledge discovery from textual databases, refers to the process of extracting interesting and non-trivial patterns or knowledge from text documents. Regarded by many as the next wave of knowledge discovery, text mining has very high commercial values.

As compared to data mining, there is more convolutions in text mining because of its unstructured and obscure content. Text mining contains features of data mining, but the distinctive point between these processes is that data mining tools are designed to cope with structured data from databases, while text mining is able to handle unstructured or semi-structured data sets which include full-text documents, emails, and HTML files etc.

Nowadays, textual content is the most present in comparison to image, audio and video. Recent studies indicate that 80% of companies' information are text documents. The production of digital data is increasing in volume because the accessibility of this media has become something.

References

- R Reference Card, by Tom Short
  http://cran.r-project.org/doc/contrib/Short-refcard.pdf

- The R Manuals, including an Introduction to R, R Language Definition, R Data Import/Export, and other R manuals
  http://cran.r-project.org/manuals.html

- R You Ready?
  http://pj.freefaculty.org/R/RUReady.pdf

- Introduction to the tm Package Text Mining in R
  http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

- R for networks: a short tutorial
  http://sites.stat.psu.edu/~dhunter/Rnetworks/