Twitter Analysis with R

Archit Gupta

September 15, 2016

Contents

Social media mining using sentiment analysis

People are highly opinionated. We hold opinions about everything from international politics to pizza delivery. Sentiment analysis, synonymously referred to as opinion mining, is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions through written language. Practically speaking, this field allows us to measure, and thus harness, opinions.

The explosion of sentiment-laden content on the Internet, the increase in computing power, and advances in data mining techniques have turned social data mining into a thriving academic field and crucial commercial domain. Professor Richard Hamming famously pushes researchers to ask themselves, "What are the important problems in my field?" Researchers in the broad area of natural language processing (NLP) cannot help but list sentiment analysis as one such pressing problem. Sentiment analysis is not only a prominent and challenging research area, but also a powerful tool currently being employed in almost every business and social domain. This prominence is due, at least in part, to the centrality of opinions as both measures and causes of human behavior.

In today's connected world, many of us are members of at least one, if not more, social networking service. The influence and reach of social media enterprises such as Facebook is staggering. Facebook has 1.11 billion monthly active users and 751 million monthly active users of their mobile products (Facebook key facts). Twitter has more than 200 million (Twitter blog) active users. As communication tools, they offer a global reach to huge multinational audiences, delivering messages almost instantaneously.

Twitter was launched in March 21, 2006, and it took 3 years, 2 months, and 1 day to reach 1 billion tweets. Twitter users now send 1 billion tweets every 2.5 days.the size and scope of big social data, the fact that some of it is comprised of honest signals, and the fact that some of it can be validated with other data, lends it validity. In another sense, the "proof is in the pudding". Businesses, governments, and organizations are already using social media mining to good effect; thus, the data being mined must be at least moderately useful.

Quantitative approaches

In this research, we aim to mine and summarize online opinions in reviews, tweets, blogs, forum discussions, and so on. Our approach is highly quantitative (that is, mathematical and/or statistical) as opposed to qualitative (that is, involving close study of a few instances). In social sciences, these two approaches are sometimes at odds, or at least their practitioners are. In this section, we will lay out the rationale for a quantitative approach to understanding online opinions. Our use of quantitative approaches is entirely pragmatic rather than dogmatic. We do, however, find the famous Bill James' words relating to the quantitative and qualitative tension to resonate with our pragmatic voice.

"The alternative to good statistics is not"no statistics", it's bad statistics. People who argue against statistical reasoning often end up backing up their arguments with whatever numbers they have at their command, over- or under-adjusting in their eagerness to avoid anything systematic."

One traditional rationale for using qualitative approaches to sentiment analysis, such as focus groups, is lack of available data. Looking closely at what a handful of consumers think about a product is a viable way to generate opinion data if none, or very little, exists. However, in the era of big social data, analysts are awash in opinionladen text and online actions. In fact, the use of statistical approaches is often necessary to handle the sheer volume of data generated by the social web. Furthermore, the explosion of data is obviating traditional hypothesis-testing concerns about sampling, as samples converge in size towards the population of interest.

Perhaps the strongest reason to choose quantitative methods over qualitative ones is the ability of quantitative methods, when coupled with large and valid data-sets, to generate accurate measures in the face of analyst biases.

Mining Twitter with R

Twitter is one of several social media networks, and there is little reason to suspect that data from Twitter is fundamentally different from other socially generated data. Another answer is that Twitter is different in subtle but important ways. One distinction is Twitter's ability to foster second-order connections, or what Granovetter (1973-1983) calls weak ties. These weak ties are important as they bring information to individuals from those with whom they share less, thus dramatically increasing information exposure. Second, Twitter, perhaps more than some other social networks, allows users to self-organize. A third answer is that Twitter users actively use Twitter to gather insight, make recommendations, and lodge public complaints. The extent to which users find this information valuable gives credence to the notion of its validity.

Obtaining Twitter data

To begin ingesting social media data from Twitter, you will need a developer account on Twitter. You can start one (free of cost) at https://dev.twitter.com/ apps. Once you have a Twitter account, return to that page and enter your username and password. Now, simply click on the Create New Application button and enter the requested information. Note that these inputs are neither important nor binding. You simply need to provide a name, description, and website (even just a personal blog) in the required fields.

Once finished, you should see a page with a lot of information about your application. Included here is a section called OAuth settings. These are crucial in helping you authenticate your application with Twitter, thus allowing you to mine tweets. More specifically, these bits of information will authenticate you with the Twitter application programming interface (API). You'll want to copy the consumer key, consumer secret, request token URL, authorize URL, and access token URL to a file and keep them handy.

Now that we have set up an application with Twitter, we need to download the R package that allows us to pull tweets into our local R session. Though there are several packages that do this, we prefer the twitteR package for its ease of use and flexibility.

```
#install.packages("twitteR")
library(twitteR) # for twitter analysis
library(ROAuth) # for OAuthFactory function
library("tm") #for text analysis
```

Now, we are just a few lines of R code away from pulling in Twitter data. If you are using a Windows machine, there is an additional prestep of downloading a cecert. pem file, which forms a portion of certain types of certification schemes for Internet transfers, as shown in the following code snippet:

```
download.file(url="http://curl.haxx.se/ca/cacert.pem"
              ,destfile=file.path(path,"Documents/cacert.pem"))
```

With that done, pass this information to a function called OAuthFactory. The requestURL, accessURL, and authURL in the following code snippet are demonstrative, but you should verify this information with that provided by Twitter as a part of authorizing your application:

```
cred <- OAuthFactory$new(consumerKey=my.key
                         , consumerSecret=my.secret
                         , requestURL='https://api.twitter.com/oauth/request_token'
                         , accessURL='https://api.twitter.com/oauth/access_token'
                         , authURL='https://api.twitter.com/oauth/authorize')
```

Finally, input the cred handshake call that follows this paragraph, including the full path to where you saved your cacert.pem file. This will bring up a URL in the R console that you will have to copy and paste into a browser. Doing so will take you to a Twitter page that will supply you with a numeric code that you can copy and paste into your instance of R after the cred handshake call.

```
cred$handshake(cainfo=file.path(path, "Documents/cacert.pem"))
```

Finally, save your authentication settings as follows:

```
save(cred, file="twitter authentication.Rdata")
registerTwitterOAuth(cred)
```

setup_twitter_oauth: This function wraps the OAuth authentication handshake functions from the httr package for a twitteR session

```
setup_twitter_oauth(my.key,my.secret,access_token,access_token_secret)
```

```
## [1] "Using direct authentication"
```

let's pull in some tweets with the #bigdata hashtag and save them to an object called bigdata as follows:

```
bigdata <- searchTwitter("#bigdata", n=1500)
```

We can find out what class or type of object bigdata is by using the class function as follows:

```
class(bigdata)
```

```
## [1] "list"
```

We easily discover that bigdata is a list or a collection of objects. There is no guarantee that searchTwitter pulled in the number of tweets requested. We may have specified a small date range or an uncommon search term. Either way, we can check the length of the bigdata list-type object with the length() function as follows:

```
length(bigdata)
```

```
## [1] 1500
```

It should be noted that the Twitter REST API (v1.1) limits the number of searches that can be performed in any given time period. The limits vary based on the type of search, the type of application making the search, as well as other criteria. Generally speaking, however, when using searchTwitter, you will be limited to 15 searches every 15 minutes, so make them count! More specific information on Twitter's rate limits can be found at *https://dev.twitter.com/docs/rate-limiting/1.1/limits*

The main tip to avoid the rate limit becoming a hindrance is to search judiciously for particular users, themes, or hashtags. Another option is to more frequently search for users and/or themes that are more active and reserve less active users or themes to intermittent search windows. It is best practice to keep track of your searches and rate limit ceilings by querying in R, or by adding rate limit queries directly to your code. If you plan to create applications rather than merely analyze data in R, other options such as caching may prove useful. The following two lines of code return the current number of each type of search that remains in a user's allotment, as well as when each search limit will reset:

```
rate.limit <- getCurRateLimitInfo(c("lists"))
rate.limit
```

```
##                   resource limit remaining               reset
## 1              /lists/list    15        15 2016-09-16 06:24:10
## 2       /lists/memberships    15        15 2016-09-16 06:24:10
## 3  /lists/subscribers/show    15        15 2016-09-16 06:24:10
## 4           /lists/members   180       180 2016-09-16 06:24:10
## 5     /lists/subscriptions    15        15 2016-09-16 06:24:10
## 6              /lists/show    15        15 2016-09-16 06:24:10
## 7        /lists/ownerships    15        15 2016-09-16 06:24:10
## 8       /lists/subscribers   180       180 2016-09-16 06:24:10
## 9      /lists/members/show    15        15 2016-09-16 06:24:10
## 10         /lists/statuses   180       180 2016-09-16 06:24:10
```

To limit the number of searches we have to undertake, it can be useful to convert our search results to a data frame and then save them for later analysis. Only two lines of code are used, one to convert the bigdata list to a data frame and another to save that data frame as a comma-separated value file:

```
# conversion from list to data frame
bigdata.df <- do.call(rbind, lapply(bigdata, as.data.frame))

# write to csv
write.csv(bigdata.df, file.path(path,"Desktop/Study content/Twittr analysis/bigdata.csv"))
```

Preliminary analyses

Text data, such as tweets, comes with little structure compared to spreadsheets and other typical types of data. One very useful way to impose some structure on text data is to turn it into a document-term matrix. This is a matrix where each row represents a document and each term is represented as a column. Each element in the matrix represents the number of times a particular term (column) appears in a particular document (row). Put differently, the i, jth element counts the number of times the term j appears in the document i. Document-term matrices get their length from the number of input documents and their width from the number of unique words used in the collection of documents, which is often called a corpus. Throughout this book, we utilize the tm package to create document-term matrices and for other utilities. The following lines of code install the tm package, preprocess our list object, bigdata, and turn it into a document-term matrix:

```
#library("tm")
bigdata_list <- sapply(bigdata, function(x) x$getText())

######################cleaning data
# remove retweet entities
bigdata_list = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", "", bigdata_list)
# remove Atpeople
bigdata_list = gsub("@\\w+", "", bigdata_list)
# remove punctuation symbols
bigdata_list = gsub("[[:punct:]]", "", bigdata_list)
# remove numbers
bigdata_list = gsub("[[:digit:]]", "", bigdata_list)
# remove links
bigdata_list = gsub("http\\w+", "", bigdata_list)
```

```
#creating a corpus
bigdata_corpus <- Corpus(VectorSource(bigdata_list))
```

A corpus is an abstraction in R to represent a collection of documents.

```
# convert to lower case
bigdata_corpus <- tm_map(bigdata_corpus, tolower)
#removing punctuation
bigdata_corpus <- tm_map(bigdata_corpus, removePunctuation)
#remving stopwords
bigdata_corpus <- tm_map(bigdata_corpus, function(x)removeWords(x,stopwords()))
# remove extra white-spaces
bigdata_corpus = tm_map(bigdata_corpus, stripWhitespace)
#adding this to remove the error in TermDocumentMatrix (Error: inherits(doc, "TextDocument") is not TRU
bigdata_corpus = tm_map(bigdata_corpus, PlainTextDocument)
```

It convert all of the words to lowercase, remove all punctuation, and drop stopwords, respectively. stopwords are very common words that do not carry much meaning, such as "a," "are," "that," and "the." A more complete list of stop words can be found at http://www.ranks.nl/resources/stopwords.html.

Creating a wordcloud with out data:

```
#install.packages("wordcloud")
library("wordcloud")
```

```
## Loading required package: RColorBrewer
```

```
wordcloud(bigdata_corpus, min.freq = 5, max.words = 50, res = 100)
```

Now, let's go on to build the document-term matrix (or in this case, term-document matrix).

```
# term-document matrix
bigdata.tdm = TermDocumentMatrix(bigdata_corpus)
```

```
bigdata.tdm
```

```
## <<TermDocumentMatrix (terms: 2218, documents: 1500)>>
## Non-/sparse entries: 12261/3314739
## Sparsity           : 100%
## Maximal term length: 34
## Weighting          : term frequency (tf)
```

identify terms used at least 10 times

```
findFreqTerms(bigdata.tdm, lowfreq=10)
```

```
##   [1] "abdsc"                "accelerator"          "accepted"             "accessible"
##  [17] "big"                  "bigdata"              "blockchain"           "boundary"
##  [33] "converged"            "correlation"          "courses"              "customer"
##  [49] "deep"                 "deeplearning"         "demand"               "depends"
##  [65] "findings"             "fintech"              "free"                 "future"
##  [81] "health"               "healthcare"           "heartbeat"            "help"
##  [97] "initiatives"          "internet"             "iot"                  "java"
## [113] "lives"                "location"             "look"                 "machine"
## [129] "mining"               "mit"                  "model"                "monitoring"
## [145] "par"                  "performance"          "personalization"      "personas"
## [161] "qué"                  "quick"                "regression"           "robotics"
## [177] "smart"                "smartcities"          "smartcity"            "spark"
## [193] "toaster"              "tools"                "top"                  "training"
```

We can also explore the data in an associational sense by looking at collocation, or those terms that frequently co-occur. From the previous list, "thanks" seems to be unexpected; so, we can explore the association of "thanks" and other terms in the corpus as follows:

```
findAssocs(bigdata.tdm, 'thanks', 0.50)
```

```
## $thanks
## newwave     ret  retail   daily
##    1.00    1.00    0.71    0.63
```

A way to get a visual sense of a set of documents is to cluster them. Clustering, in general, is a way of finding associations between items (for example, documents). This necessitates a measure of how far each observation is from every other one. Nearby observations are binned together in groups, whereas the ones further apart are put into separate groups. There are many ways to implement clustering; here, we use a variant called hierarchical agglomerative clustering.

We implement our chosen method by first removing the sparsest terms from our term-document matrix. Sparse terms are those which only occur in a small proportion of documents. By removing sparse terms, we reduce the length of the term-document matrix dramatically without losing relations inherent in the matrix. The sparse argument in the following line of code details the proportion of zeroes a term must have before being considered sparse:

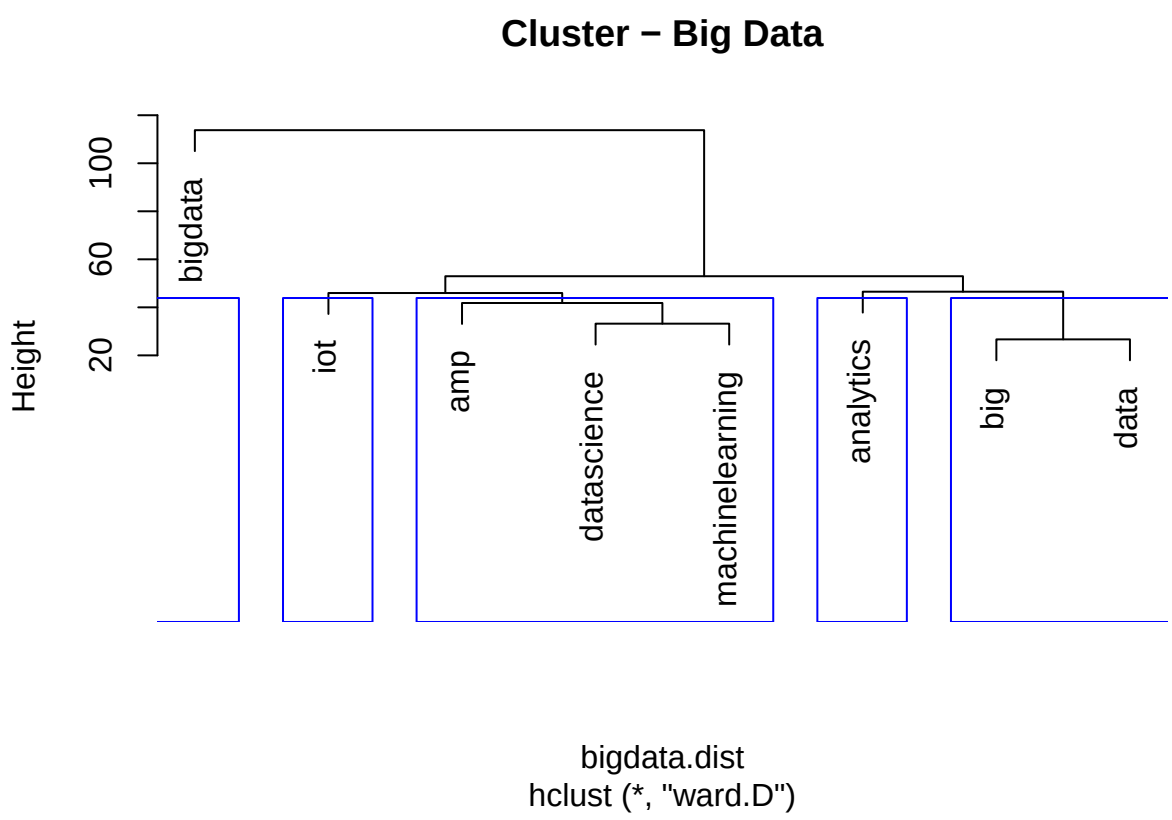
```
# Remove sparse terms from the term-document matrix
bigdata2.tdm <-removeSparseTerms(bigdata.tdm, sparse=0.92)
# Convert the term-document matrix to a data frame
bigdata2.df <- as.data.frame(as.matrix(bigdata2.tdm))
```

Next, we scale the term-document matrix because clustering is sensitive to the scale of the data used. Specifically, the scale function subtracts every element in a vector from the vector's mean and divides each element by the vector's standard deviation. Once scaled, we use the term-document matrix to compute a distance matrix, where each row is a document and each column is the same set of documents. Each cell represents the distance between each pair of documents.

```r
# scale the data
bigdata2.df.scale <- scale(bigdata2.df)

# Create the distance matrix
bigdata.dist <- dist(bigdata2.df.scale, method = "euclidean")

# Cluster the data
bigdata.fit <- hclust(bigdata.dist, method="ward")
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```r
# Visualize the result
plot(bigdata.fit, main="Cluster - Big Data")

# An example with five (k=5) clusters
groups <- cutree(bigdata.fit, k=5)

# Dendogram with blue clusters (k=5).
rect.hclust(bigdata.fit, k=5, border="blue")
```



The preceding dendrogram shows our clusters in a tree diagram. The bottom row shows individual observations with groupings increasing in size as we traverse up the tree. Our sample was small and so our dendrogram is small, and our clusters may reflect groupings that do not generalize outside of our sample.