Twitter Sentiment Analysis

Aankur Bhatia

Friday, June 26, 2015

## Introduction

Twitter is an important social media tool for expressing and understanding perception, behavior, and sentiments. Twitter data is very valuable to data scientist in analyzing behavior trends and predicting events. More often than not, social media is first to report on an event, incident or attack. We shall limit this analysis to cyber security tweets only, however the model can be run on any topic, people or place.

Here is an attempt to analyze market sentiment using R programming language. In this post I will explain the method to search Twitter for feeds matching a specific search string and analyze that information comparing it to standard set of positive and negative words. The data set for the positive and negative opinion words (sentiment words) comes from [Hu and Liu, KDD-2004](http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon).

The main packages used in this analysis are twitteR, dplyr, stringr, ggplot2, tm, SnowballC, qdap, and wordcloud.It is important to install and load these packages using install.packages() and library() commands.

## Step 1 : Load Twitter API

The first step is to register in the [twitter application developer's portal](https://dev.twitter.com/) and get the authorization. You will need

api\_key= "your api key"  
api\_secret= "your api\_secret password"  
access\_token= "your access token"  
access\_token\_secret= "your access token password"

After obtaining credentials, you will need to setup authorizations to access twitter API.

setup\_twitter\_oauth(api\_key,api\_secret,access\_token,access\_token\_secret)

## Step 2 : Load word dictionaries

Next Step is to load the set of positive and negative sentiments words (dictionary) into your R working directory. The words are then accessed assigned to variables positive and negative as shown below.

positive=scan('positive-words.txt',what='character',comment.char=';')  
negative=scan('negative-words.txt',what='character',comment.char=';')  
positive[20:30]

## [1] "accurately" "achievable" "achievement" "achievements"  
## [5] "achievible" "acumen" "adaptable" "adaptive"   
## [9] "adequate" "adjustable" "admirable"

negative[500:510]

## [1] "byzantine" "cackle" "calamities" "calamitous"   
## [5] "calamitously" "calamity" "callous" "calumniate"   
## [9] "calumniation" "calumnies" "calumnious"

There are a total of 2006 positive and 4783 negative words. The above section also shows some examples of words from the two dictionaries

Additional words can be added or removed from the dictionaries. In the next command, I'm adding the word cloud to the positive word dictionary while removing from negative word dictionary.

positive=c(positive,"cloud")  
negative=negative[negative!="cloud"]

## Step 3 : Search twitter feeds

The next step is to define a twitter search string and assign to a variable. It is also good to pre-decide on the number of tweets you will be analyzing. Often, slow internet connections and/or complex search fields may result in extraordinary delay periods

findfd= "DataBreach"  
number= 3000

In the above code, we are using DataBreach string to retrieve 3000 tweets. The code for searching twitter for the feeds is

tweet=searchTwitter(findfd,number)

## Warning in doRppAPICall("search/tweets", n, params = params,  
## retryOnRateLimit = retryOnRateLimit, : 3000 tweets were requested but the  
## API can only return 2380

## Time difference of 45.79762 secs

It took 45.8 seconds to retrieve the tweets.

## Step 4 : Getting text from feeds

The Twitter feeds have a lot of superfluous information embedded which is not necessary useful in all analysis. We'll use the gettext() function to extract the text fields and assign the list to a variable tweetT. The function is applied to all 3000 tweets. The code below also shows text of the extracted first five feeds.

tweetT=lapply(tweet,function(t)t$getText())  
head(tweetT,5)

## [[1]]  
## [1] "TN: Manchester Hotel Hospitality first notifies customers of breach IHG knew about months ago http://t.co/hdybWAsdxg #DataBreach #ALERT"  
##   
## [[2]]  
## [1] "RT @BrianEFinch: Is China behind the U.S. #OPM #databreach, and if so, how should that impact #cybersecurity legislation? http://t.co/xWEug"  
##   
## [[3]]  
## [1] "TN: Manchester Hotel Hospitality first notifies customers of #databreach IHG knew about months ago: http://t.co/orCT0ryQiH"  
##   
## [[4]]  
## [1] "MD: Meritus contacting patients after privacy incident http://t.co/t0RNuO8tkf #DataBreach #ALERT"  
##   
## [[5]]  
## [1] "Court records link Scientology to convicted email hacker http://t.co/lq5ShH0Tpe #DataBreach #ALERT"

## Step 5 : Defining text cleaning functions

We'll be defining a function which executes a series of commands to clean the text, removes punctuations, special charaters, embedded http links, extra spaces, and digits. This function also changes upper case to lower case using tolower() function. However, this often leads to errors which stops the execution of the r code. To avoid this, we'll be first defining an error catching function tryTolower

tryTolower = function(x)  
{  
 # create missing value  
 # this is where the returned value will be  
 y = NA  
 # tryCatch error  
 try\_error = tryCatch(tolower(x), error = function(e) e)  
 # if not an error  
 if (!inherits(try\_error, "error"))  
 y = tolower(x)  
 return(y)  
}

The clean() function defined below will clean the twitter feeds and split the strings into a vector of words

clean=function(t){  
 t=gsub('[[:punct:]]','',t)  
 t=gsub('[[:cntrl:]]','',t)   
 t=gsub('\\d+','',t)  
 t=gsub('[[:digit:]]','',t)  
 t=gsub('@\\w+','',t)  
 t=gsub('http\\w+','',t)  
 t=gsub("^\\s+|\\s+$", "", t)  
 t=sapply(t,function(x) tryTolower(x))  
 t=str\_split(t," ")  
 t=unlist(t)  
 return(t)  
}

## Step 6 : Cleaning and splitting twitter feeds

In this step, we'll apply the clean() function defined above to clean the feeds. The resultant feeds are stored in the large list tweetclean. The code below also shows the first five feeds cleaned and split by this function

tweetclean=lapply(tweetT,function(x) clean(x))  
head(tweetclean,5)

## [[1]]  
## [1] "tn" "manchester" "hotel" "hospitality" "first"   
## [6] "notifies" "customers" "of" "breach" "ihg"   
## [11] "knew" "about" "months" "ago" ""   
## [16] "databreach" "alert"   
##   
## [[2]]  
## [1] "rt" "brianefinch" "is" "china"   
## [5] "behind" "the" "us" "opm"   
## [9] "databreach" "and" "if" "so"   
## [13] "how" "should" "that" "impact"   
## [17] "cybersecurity" "legislation" ""   
##   
## [[3]]  
## [1] "tn" "manchester" "hotel" "hospitality" "first"   
## [6] "notifies" "customers" "of" "databreach" "ihg"   
## [11] "knew" "about" "months" "ago"   
##   
## [[4]]  
## [1] "md" "meritus" "contacting" "patients" "after"   
## [6] "privacy" "incident" "" "databreach" "alert"   
##   
## [[5]]  
## [1] "court" "records" "link" "scientology" "to"   
## [6] "convicted" "email" "hacker" "" "databreach"   
## [11] "alert"

## Step 7 : Analyzing twitter feeds

Now we get into the actual meat of analyzing feeds by comparing the text strings with the word dictionaires and making hypothesis on sentiment. We will first define a function to count the number of positive words (in tweets) that are matching our database, we call the function returnpscore

returnpscore=function(tweet) {  
 pos.match=match(tweet,positive)  
 pos.match=!is.na(pos.match)  
 pos.score=sum(pos.match)  
 return(pos.score)  
}

Next we apply this function to the tweetclean list

positive.score=lapply(tweetclean,function(x) returnpscore(x))

We need a loop count the total number of positive words present in the tweets

pcount=0  
for (i in 1:length(positive.score)) {  
 pcount=pcount+positive.score[[i]]  
}  
pcount

## [1] 509

As observed, there are 509 positive words in the extracted tweets. We use similar methods to find negative score in the feeds.

We also need a function to find the positive words that have matched with the database. This data will be used subsequetnly to analyze strong or weak sentiment.

poswords=function(tweets){  
 pmatch=match(t,positive)  
 posw=positive[pmatch]  
 posw=posw[!is.na(posw)]  
 return(posw)  
 }

This function is applied to our tweetclean list and a loop is called to add words to a dataframe named pdatamart. The code below also shows first 10 matches of positive words

words=NULL  
pdatamart=data.frame(words)  
  
for (t in tweetclean) {  
 pdatamart=c(poswords(t),pdatamart)  
}  
head(pdatamart,10)

## [[1]]  
## [1] "successfully"  
##   
## [[2]]  
## [1] "trust"  
##   
## [[3]]  
## [1] "thank"  
##   
## [[4]]  
## [1] "blessing"  
##   
## [[5]]  
## [1] "protection"  
##   
## [[6]]  
## [1] "blessing"  
##   
## [[7]]  
## [1] "blessing"  
##   
## [[8]]  
## [1] "protection"  
##   
## [[9]]  
## [1] "safe"  
##   
## [[10]]  
## [1] "safe"

Similarly, a series of functions and loops are defined for finding negative words and sentiments in the tweets and are assigned to another dataframe ndatamart. Here are the first ten negative words as matched with the tweets.

head(ndatamart,10)

## [[1]]  
## [1] "hack"  
##   
## [[2]]  
## [1] "breach"  
##   
## [[3]]  
## [1] "breach"  
##   
## [[4]]  
## [1] "dark"  
##   
## [[5]]  
## [1] "bug"  
##   
## [[6]]  
## [1] "breach"  
##   
## [[7]]  
## [1] "breach"  
##   
## [[8]]  
## [1] "stolen"  
##   
## [[9]]  
## [1] "issue"  
##   
## [[10]]  
## [1] "odd"

## Step 8 : Plotting high frequency negative and positive words

In this step, we'll put some charts together to show the distribution of high frequency positive and negative words. Before that, we have to convert the word lists to vectors using unlist() function. The vector variables pwords and nwords are then converted to dataframe objects.

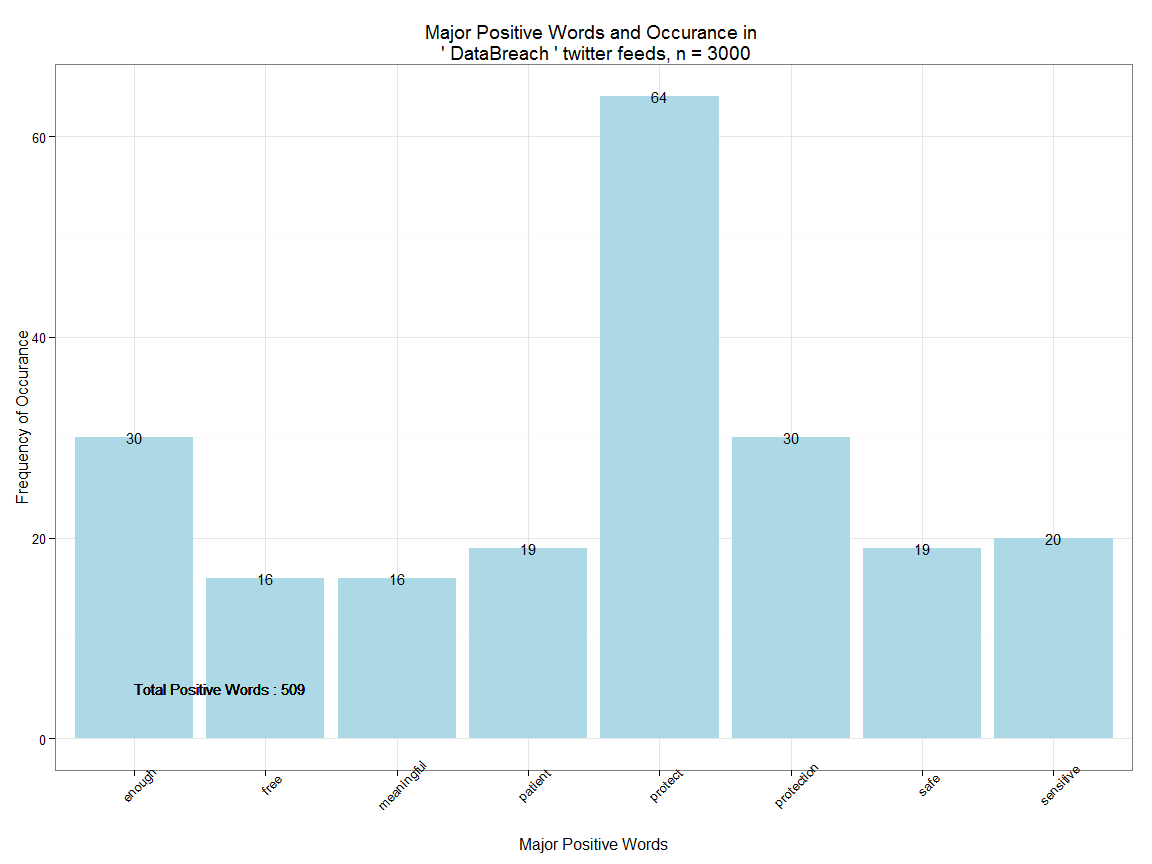
dpwords=data.frame(table(pwords))  
dnwords=data.frame(table(nwords))

Using dplyr package, we first mutate the words as character variables and then filter for frequency >15 repititions for both positive and negative words. The code below shows example only for positive words

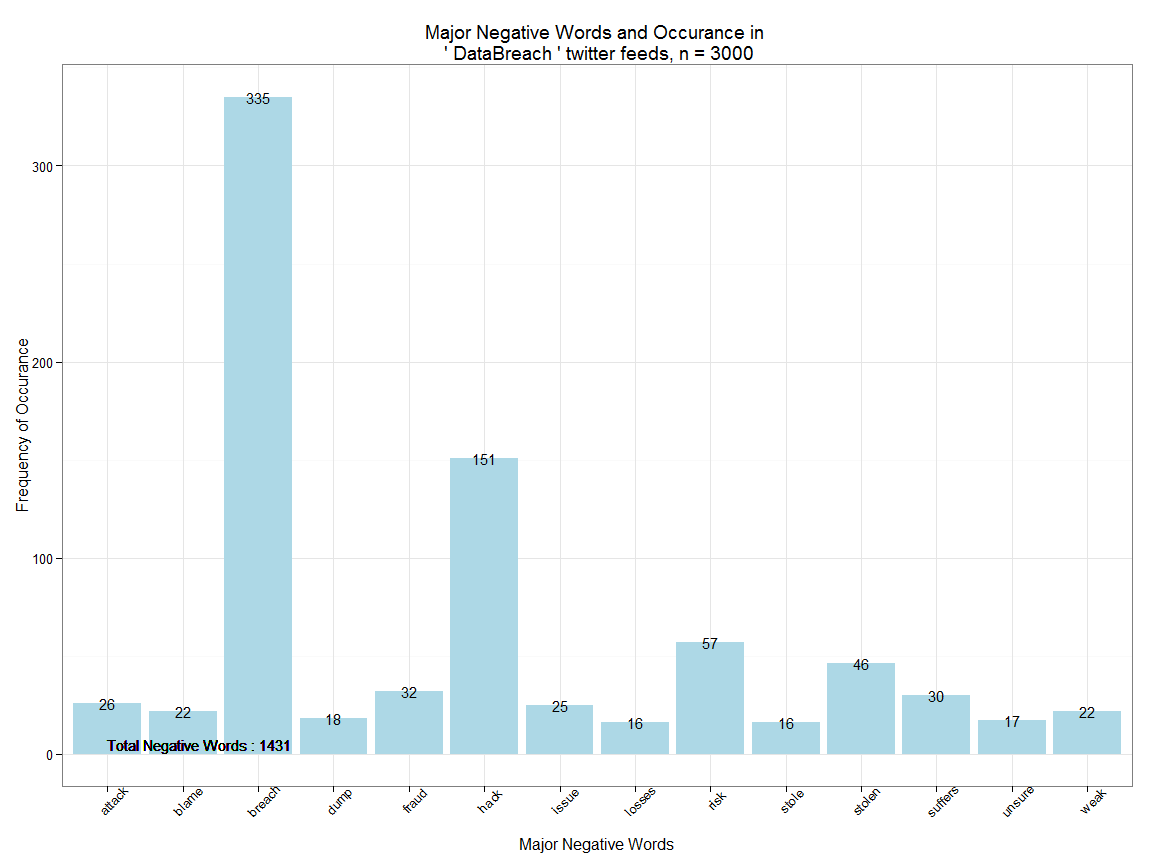
dpwords=dpwords%>%  
 mutate(pwords=as.character(pwords))%>%  
 filter(Freq>15)

We will now plot the major positive words and their frequency using ggplot2 package. As you notice, there are a total of 509 positive words in the twitter feeds. The frequency distribution also gives an indication of

ggplot(dpwords,aes(pwords,Freq))+geom\_bar(stat="identity",fill="lightblue")+theme\_bw()+  
 geom\_text(aes(pwords,Freq,label=Freq),size=4)+  
 labs(x="Major Positive Words", y="Frequency of Occurance",title=paste("Major Positive Words and Occurance in \n '",findfd,"' twitter feeds, n =",number))+  
 geom\_text(aes(1,5,label=paste("Total Positive Words :",pcount)),size=4,hjust=0)+theme(axis.text.x=element\_text(angle=45))



Likewise, we plot negative words and their frequency. As you observe, there are a total of 1431 negative words in 3000 twitter feeds on DataBreach search string



## Step 9: Removing common words and creating wordcloud

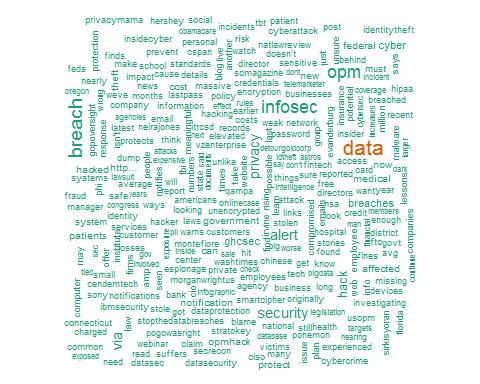
We will now convert the tweetclean into a corpus using the function VectorSource. A word corpus will enable us to eliminate superflous common words using the text mining package tm. Removing common words also called stop words will let us establish context and sentiment. The below code also provides few examples of 'Stop Words'

tweetscorpus=Corpus(VectorSource(tweetclean))  
tweetscorpus=tm\_map(tweetscorpus,removeWords,stopwords("english"))  
stopwords("english")[30:50]

## [1] "what" "which" "who" "whom" "this" "that" "these"   
## [8] "those" "am" "is" "are" "was" "were" "be"   
## [15] "been" "being" "have" "has" "had" "having" "do"

We will now create a Word Cloud of tweets using the wordcloud package. We are limiting the maximum words to 300

wordcloud(tweetscorpus,scale=c(5,0.5),random.order = TRUE,rot.per = 0.20,use.r.layout = FALSE,colors = brewer.pal(6,"Dark2"),max.words = 300)



## Step 10: Analyzing and plotting high frequency words

In this final step, firstly we'll convert the word corpus into a document matrix using the function DocumentTermMatrix. The Document matrix can be analyzed to examine most frequently occuring uncommon words. Next we'll remove the sparse terms (one off's) from the corpus. The below code displays the most frequent terms (with frequency of 50 or above)

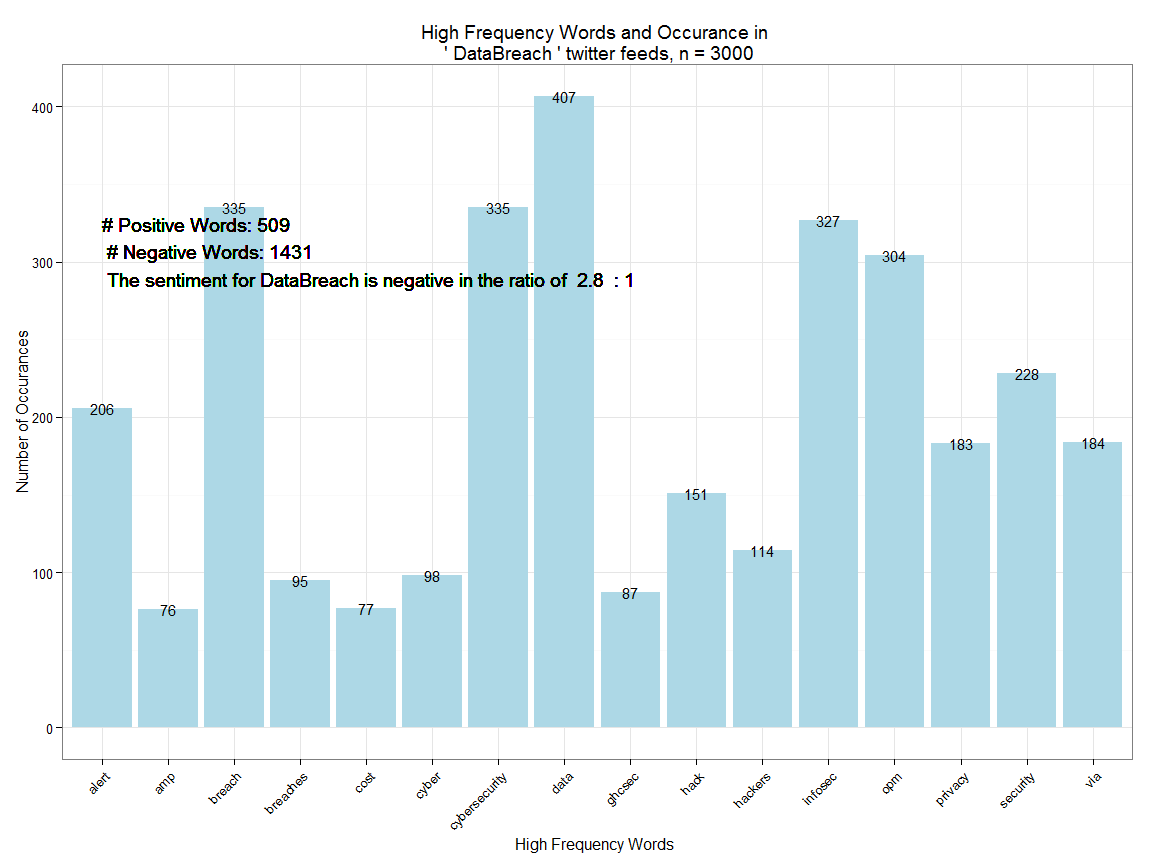
dtm=DocumentTermMatrix(tweetscorpus)  
# #removing sparse terms  
dtms=removeSparseTerms(dtm,.99)  
freq=sort(colSums(as.matrix(dtm)),decreasing=TRUE)  
#get some more frequent terms  
findFreqTerms(dtm,lowfreq=50)

## [1] "affected" "alert" "amp" "breach"   
## [5] "breaches" "can" "cost" "cyber"   
## [9] "cybersecurity" "data" "databreach" "employee"   
## [13] "federal" "ghcsec" "government" "hacked"   
## [17] "hack" "hackers" "http" "info"   
## [21] "infosec" "law" "medical" "million"   
## [25] "new" "now" "opm" "patients"   
## [29] "privacy" "protect" "risk" "security"   
## [33] "theft" "via"

finally, we'll convert the matrix to a dataframe, filter for Minimum frequency > 75and plot using ggplot2

wf=data.frame(word=names(freq),freq=freq)  
wfh=wf%>%  
 filter(freq>=75,!word==tolower(findfd))

ggplot(wfh,aes(word,freq))+geom\_bar(stat="identity",fill='lightblue')+theme\_bw()+  
 theme(axis.text.x=element\_text(angle=45,hjust=1))+  
 geom\_text(aes(word,freq,label=freq),size=4)+labs(x="High Frequency Words ",y="Number of Occurances", title=paste("High Frequency Words and Occurance in \n '",findfd,"' twitter feeds, n =",number))+  
 geom\_text(aes(1,max(freq)-100,label=paste("# Positive Words:",pcount,"\n","# Negative Words:",ncount,"\n",result(ncount,pcount))),size=5, hjust=0)



As seen from the observed data, The sentiment for DataBreach is negative in the ratio of 2.8 : 1. The result makes more sense if it's analyzed over a time period