Twitter Sentiment Analysis Tutorial

Contents

Analysis performed on 2015-06-27 at 2015-06-27 22:38:16

Introduction

Social media including Twitter, Facebook, LinkedIn are the most popular free public platforms for expressing opinion on a diverse range of subjects. With millions of tweets (feeds) daily, there is a wealth of information out there. Twitter, in particular is used extensively by individuals and companies for status updates and product/ services marketing. Twitter is also a great text mining source for data scientists. From analyzing behavior, incidents, sentiments to predicting stock markets, events and trends, it provides a ton of data for mining and contextual analytic s.

In this post, I will show how to do a simple sentiment analysis. We will download twitter feeds on a subject and compare it to a database of positive, negative words. The ratio of the matched positive and negative words is the `sentiment ratio`. We will also define functions to find most frequently occurring words. These words can provide useful contextual information on public opinion and sentiments. The data set for the positive and negative opinion words (sentiment words) comes from Hu and Liu, KDD-2004.

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.

Load Twitter API

The first step is to register in the twitter application developer's portal and get the authorization. You 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, we setup authorizations to access twitter API.

```
setup_twitter_oauth(api_key,api_secret,access_token,access_token_secret)
```

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 and 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 code below, we add the word `cloud` to the `positive` word dictionary and remove it from the `negative` word dictionary.

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

Search twitter feeds

The next step is defining a twitter search string and assigning to a variable, `findfd`. Number of tweets to be extracted is assigned to another variable, `number`. The time to perform the twitter search and extraction is affected by this number. A slow internet connections and/or complex search fields may result in additional delays.

```
findfd= "CyberSecurity"
number= 5000
```

In the above code, we use CyberSecurity string to retrieve 5000 tweets. The code for searching twitter for the feeds is

```
tweet=searchTwitter(findfd,number)
```

```
## Time difference of 1.423481 mins
```

Getting text from feeds

Twitter feeds have tons of additional fields and embedded superfluous information. We use the `gettext()` function to extract the text fields and assign the list to a variable `tweetT`. The function is applied to all 5000 tweets. The code below also shows results of extraction for the first 5 feeds.

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

```
## [[1]]
## [1] "US IT Cyber Security Center Di... - #Tampa , FL (http://t.co/JBxoLPrMQK) Get Cyber Security Jobs
##
## [[2]]
## [1] "Cybersecurity experts try to predict hackers' next moves - Tribune-Review http://t.co/3aXbLzl5X
##
## [[3]]
## [1] "Tom Still: Revisiting recent topics: Startups, innovation, trade ... - http://t.co/zGAihLoep9 h
##
## [[4]]
## [1] "RT @moixsec: Stolen logins for US government agencies found all over the web #cybersecurity http
##
## [[5]]
## [1] "RT @thehill: Senior ATF official reportedly under investigation for possible data breach: http:/
```

Defining text cleaning functions

In this step, we write a function which executes a series of commands to clean text, removes punctuation, special characters, embedded HTTP links, extra spaces, and digits. This function also changes upper case characters to lower case string using `tolower()` function. Many times, the tolower() function stops unexpectedly as it encounters special characters stopping execution of the r code. To avoid this, we write an error catching function, `tryTolower`, and embed it in the code of the text cleaning function.

```
tryTolower = function(x)
{
  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 cleans the twitter feeds and splits 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)
}
```

Cleaning and splitting twitter feeds

In this step, we apply the `clean()` function defined above to clean the list of 5000 feeds. The resultant feeds are stored in a list object, `tweetclean`. The following code also shows the first 5 tweets cleaned and split using this function

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

```
## [[1]]
##  [1] "us"            "it"            "cyber"         "security"
##  [5] "center"        "di"            ""              "tampa"
##  [9] ""              "fl"            ""              "get"
## [13] "cyber"         "security"      "jobs"          "cybersecurity"
## [17] "jobs"          "job"           "getalljobs"
##
## [[2]]
##  [1] "cybersecurity" "experts"       "try"           "to"
##  [5] "predict"       "hackers"       "next"          "moves"
##  [9] ""              "tribunereview" ""              "cybersecurity"
##
## [[3]]
##  [1] "tom"           "still"         "revisiting"    "recent"
##  [5] "topics"        "startups"      "innovation"    "trade"
##  [9] ""              ""              ""              ""
## [13] "cybersecurity"
##
## [[4]]
##  [1] "rt"            "moixsec"       "stolen"        "logins"
##  [5] "for"           "us"            "government"    "agencies"
##  [9] "found"         "all"           "over"          "the"
## [13] "web"           "cybersecurity"
##
## [[5]]
##  [1] "rt"            "thehill"       "senior"        "atf"
##  [5] "official"      "reportedly"    "under"         "investigation"
##  [9] "for"           "possible"      "data"          "breach"
```

Analyzing twitter feeds

Here we get into the actual task of analyzing feeds. We compare the twitter text feeds with the word dictionaries and retrieve out the matching words. To do this, we first define a function to count the number of positive and negative words that are matching our database. Here is the code for the function, `returnpscore` for counting positive matching words.

```
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))
```

Next we define a loop to 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] 1543
```

As seen above, there are 1543 positive words in the extracted tweets. Similar method is used to find negative score in the feeds.

The following code retrieves the positive and negative matching words.

```
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 data frame, `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] "worth"
##
## [[2]]
## [1] "top"
##
## [[3]]
## [1] "innovation"
##
## [[4]]
## [1] "secure"
##
## [[5]]
## [1] "like"
##
## [[6]]
## [1] "pros"
##
## [[7]]
## [1] "top"
##
## [[8]]
## [1] "well"
##
## [[9]]
## [1] "advantage"
##
## [[10]]
## [1] "secure"
```

Similarly, we write a series of functions and loops for finding negative words and sentiments. This is assigned to another data frame, `ndatamart`. Here are the first 10 negative words from the tweets.

```
head(ndatamart,10)
```

```
## [[1]]
## [1] "breach"
##
## [[2]]
## [1] "cheats"
##
## [[3]]
## [1] "stole"
##
## [[4]]
## [1] "stolen"
##
## [[5]]
## [1] "confusion"
##
## [[6]]
## [1] "breach"
##
```

```
## [[7]]
## [1] "blunder"
##
## [[8]]
## [1] "negligence"
##
## [[9]]
## [1] "blunder"
##
## [[10]]
## [1] "negligence"
```

Plotting high frequency negative and positive words

In this step, we create some charts to show the distribution of high frequency positive and negative words.We use the `unlist()` function to convert the list to vectors. The vector variables `pwords` and `nwords` are converted to `dataframe` objects.

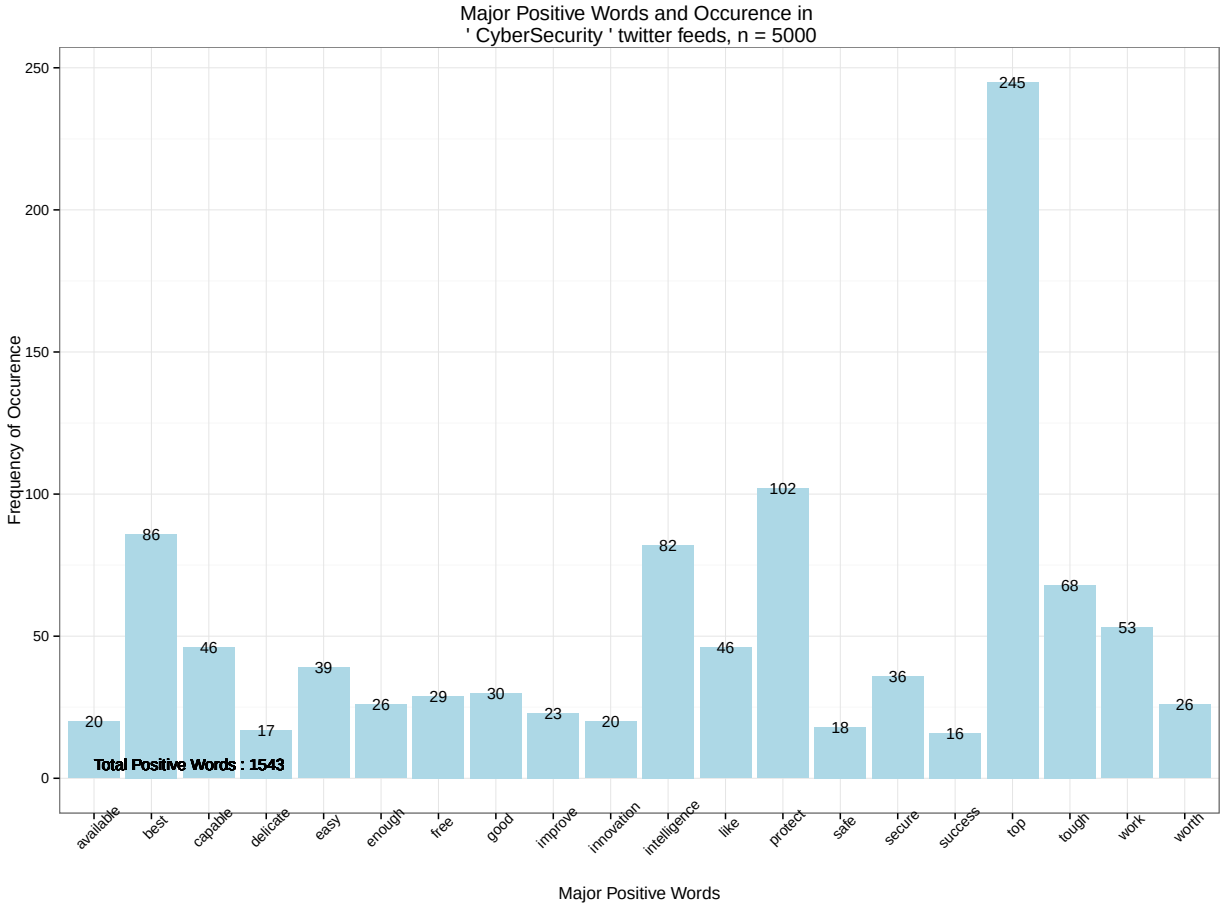
```
dpwords=data.frame(table(pwords))
dnwords=data.frame(table(nwords))
```

Using `dplyr` package, we first `mutate` the words as `character` variables and then filter for `frequency >15` repetitions for positive and negative words.

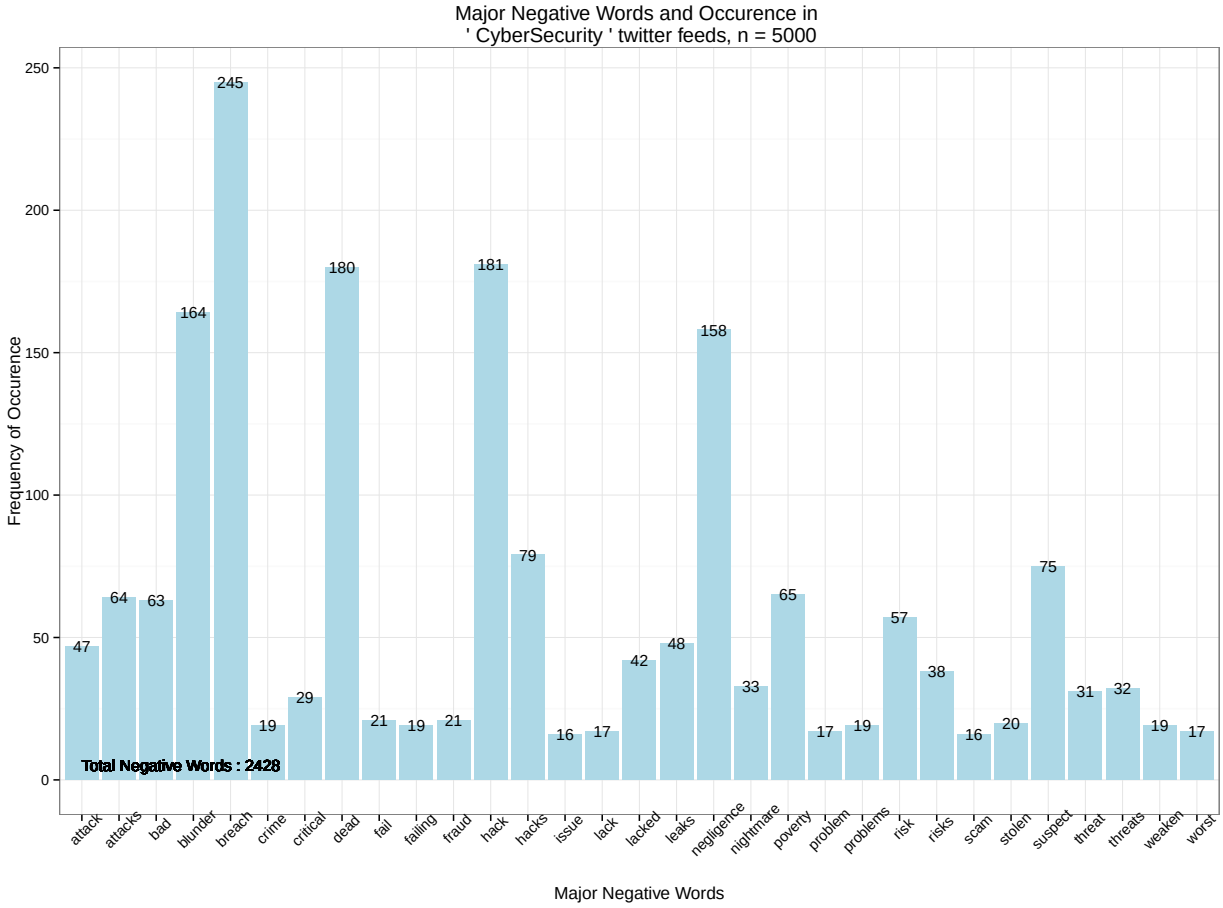
```
dpwords=dpwords%>%
  mutate(pwords=as.character(pwords))%>%
  filter(Freq>15)
```

We plot the major `positive` words and their `frequency` using `ggplot2` package. As seen, there are a total of 1543 positive words. The frequency distribution gives an indication of the degree of positive sentiment.

```
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 Occurence",title=paste("Major Positive Words and Occur
  geom_text(aes(1,5,label=paste("Total Positive Words :",pcount)),size=4,hjust=0)+theme(axis.text.x=ele
```



Likewise, we plot `negative` words and their `frequency`. There are a total of 2428 negative words in 5000 twitter feeds on CyberSecurity search string



Removing common words and creating wordcloud

Here we convert the `tweetclean` into a word corpus using the function `VectorSource`. A word corpus enables us to eliminate superfluous common words using the text mining package `tm`. Removing common words, also called `stop words`, lets us focus on the important words and help establishing context. The following code below 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"
```

Next, we create a Word Cloud of tweets using the `wordcloud` package. Notice that 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 = 
```

Analyzing and plotting high frequency words

In this final step, we convert the word corpus into a document matrix using the function `DocumentTermMatrix`. The Document matrix can be analyzed to examine most frequently occurring uncommon words. Next, we remove the sparse terms (extremely low frequency words) from the corpus. The 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=100)
```

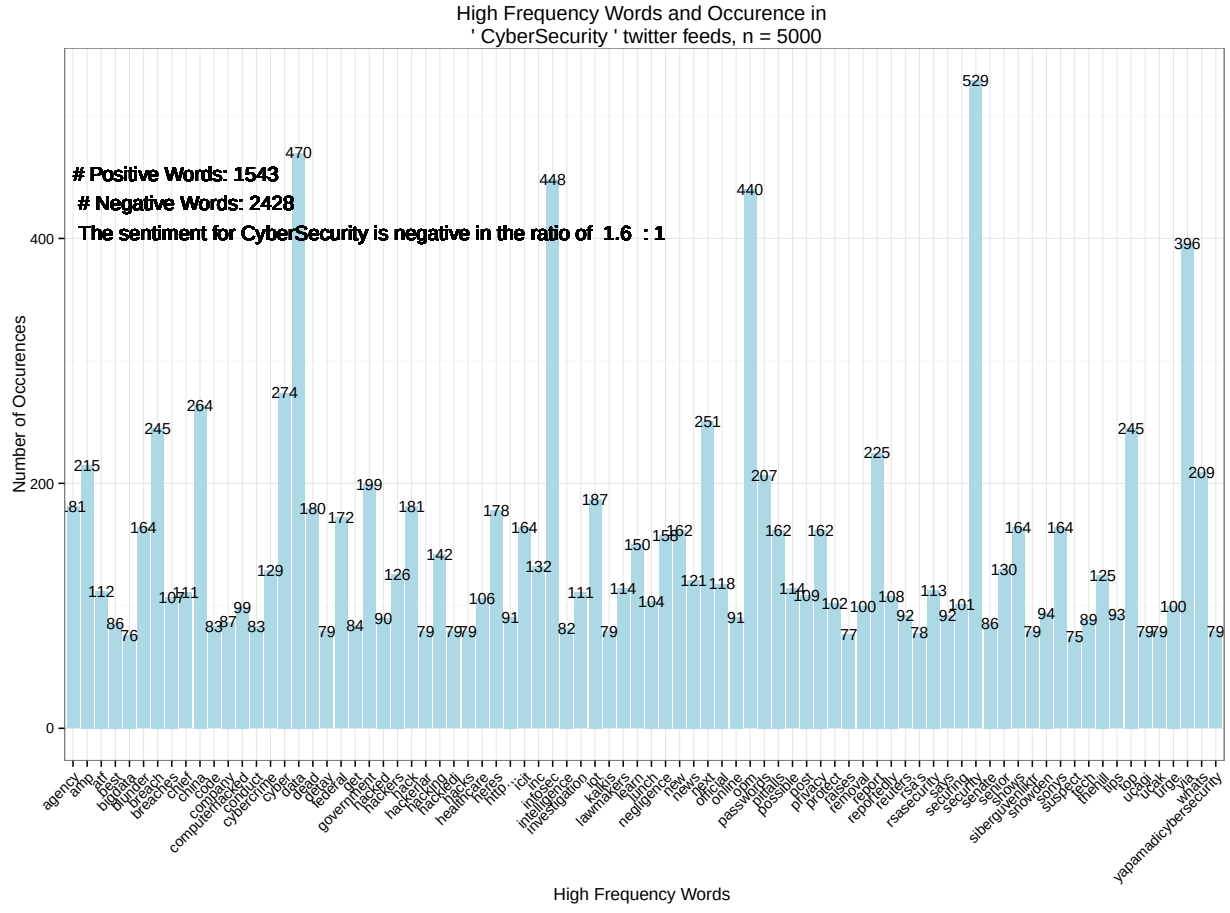
```
##  [1] "agency"        "amp"           "atf"           "blunder"
##  [5] "breach"        "breaches"      "chief"         "china"
##  [9] "cyber"         "cybercrime"    "cybersecurity" "data"
## [13] "dead"          "federal"       "government"    "hack"
## [17] "hackers"       "hacking"       "healthcare"    "heres"
## [21] "icit"          "inc"           "infosec"       "investigation"
## [25] "iot"           "lawmakers"     "learn"         "lunch"
## [29] "negligence"    "new"           "news"          "next"
## [33] "official"      "opm"           "passwords"     "pitfalls"
## [37] "possible"      "post"          "privacy"       "protect"
```

```
## [41] "removal"       "report"        "reportedly"    "rsasecurity"
## [45] "securing"      "security"      "senior"        "shows"
## [49] "sonys"         "thehill"       "top"           "urge"
## [53] "via"           "whats"
```

finally, we convert the matrix to a data frame, filter for `Minimum frequency > 75`and 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 Occurences",
  geom_text(aes(1,max(freq)-100,label=paste("# Positive Words:",pcount,"\n","# Negative Words:",ncount,
```



Conclusion

As observed, The sentiment for CyberSecurity is negative in the ratio of 1.6 : 1. The same analysis can be extended over multiple time periods for trend analysis. It can also be run iteratively over related subjects to compare and analyze relative sentiment rating.