Document Representation: Vector Space Models

Ricardo Navares

Abstract

This is a step by step guide for document processing via Vector Space Models. It will be explained in detail the different steps needed to convert documents into this representation and the comparison of a local and global document weight functions.

Introduction

A corpus is a collection of documents out of which, the representation is created. We are going to use a collection of HTML documents from CNBC articles which are detailed in the attachment *corpus.txt*. This text document consists on a list of urls which will be scrapped in order to retrieve their contents. The suggested articles can be change as desired but keeping in mind they urls must have HTML contents. The articles contained are the following:

```
http://www.cnbc.com/2016/01/20/us-housing-starts-permits-fall-25-in-december.html
http://www.cnbc.com/2016/01/22/5-a-head-the-great-canadian-cauliflower-crisis.html
http://www.cnbc.com/2015/03/06/worlds-most-expensive-airport-car-parks-want-cheap-parking-dont-go-here.html
http://www.cnbc.com/2015/03/06/europe-is-far-less-hassle-than-us-start-up.html
http://www.cnbc.com/2016/01/21/us-crude-prices-stabilize-after-jumping-from-2003-lows.html
http://www.cnbc.com/2016/01/23/goldman-morgan-stanley-ceos-see-2015-pay-cuts.html
http://www.cnbc.com/2016/01/20/spotify-rival-deezer-just-secured-a-109-million-funding-round.html
http://www.cnbc.com/2016/01/28/is-a-market-storm-coming-ask-an-escort-commentary.html
http://www.cnbc.com/2015/12/24/saudis-kingdom-holding-among-group-investing-2477m-in-lyft.html
http://www.cnbc.com/2016/01/28/zika-epidemic-on-us-doorstep-spurs-vaccine-hunt.html
http://www.cnbc.com/2016/01/29/russia-holds-rates-at-11-as-economy-lags.html
http://www.cnbc.com/2016/01/14/harvesting-solar-6km-in-the-air.html
http://www.cnbc.com/2016/01/28/how-big-business-can-tackle-those-hot-tech-start-ups.html
http://www.cnbc.com/2016/01/27/ads-are-about-to-get-way-more-personal-thanks-to-artificial-intelligence.html
```

From the contents several steps will be performed to select and reduce the features so the weighting functions can be applied. The implementation was done in R[1] programming language, which is also included in this repository. This document also serves as a code guide along with the theoretical background.

[1] https://www.r-project.org/about.html

Requirements

```
R version 3.2.3 (2015-12-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.3 LTS

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] wordcloud_2.5      RColorBrewer_1.1-2 SnowballC_0.5.1    XML_3.98-1.1       RCurl_1.95-4.7
[6] bitops_1.0-6       tm_0.6-2           NLP_0.1-8
```

The *main* function only needs the corpus file *corpus.txt* with the urls and the R script *htmlToText.R* provided by [1]. As an output it generates a file with the vocabulary of the corpus, a file with the global weights, one file per document with the local weights and the inverted file with all results.

Process

These steps are applied iteratively for each document in the corpus. Thus, we can easily identify each document and distinguish between local and global metrics.

Scrap HTML documents

Through a *cUrl*[2] we obtain from the url the contents of each article which is stored in a variable,

```r
html = getURL(as.character(corpus_url[i,1]))
# convert HTML to text
html2txt = lapply(html, htmlToText)
# clean out non-ASCII characters
html2txtclean = sapply(html2txt,
                       function(x)
                       iconv(x, "latin1", "ASCII", sub=""))

# make corpus for text mining
corpus = Corpus(VectorSource(html2txtclean),list(language="en"))
```

As a first step, all non ASCII characters needs to be removed in order to convert the content to a vector of characters through *VectorSource* [2]

[2] https://en.wikipedia.org/wiki/CURL

Lexical analysis and Stop-words removal

In any document there are several characters which contain no relevant information and generate noise if we want to classify text. Those characters have little use or none for statistical analysis, among them punctuation signs such as commas, semicolons, ... are frequent but provide no additional information. For the same reasoning all numerical characters are also removed. Hence compounded words such as *post-graduate* turn into *postgraduate*.

Same applies to additional blanks and capital letters as from the point of view of representation, *word* will be considered the same as *Word*, *WORD*... so we can increment its count for every appearance in the text.

```
    skipWords = function(x) removeWords(x, stopwords("english"))
    funcs = list(tolower, removePunctuation, removeNumbers,
                 stripWhitespace, skipWords)
    corpus_lexical = tm_map(corpus, FUN = tm_reduce, tmFuns = funcs)
    corpus_text = tm_map(corpus_lexical, PlainTextDocument)
```

A *stop-word* is a word which has no semantic relevance in the meaining but they are needed articulate the speech. The proposed set of *stop-words* in english is defined in Tartarus[3]:

```
i me my myself we our ours ourselves you your yours yourself yourselves he him his himself she her hers herself it its
itself they them their theirs themselves what which who whom this that these those am is are was were be been being have
has had having do does did doing would should could ought i'm you're he's she's it's we're they're i've you've we've
they've i'd you'd he'd she'd we'd they'd i'll you'll he'll she'll we'll they'll isn't aren't wasn't weren't hasn't
haven't hadn't doesn't don't didn't won't wouldn't shan't shouldn't can't cannot couldn't mustn't let's that's who's
what's here's there's when's where's why's how's a an the and but if or because as until while of at by for with about
against between into through during before after above below to from up down in out on off over under again further
then once here there when where why how all any both each few more most other some such no nor not only own same so
than too very
```

As an example, the following text shows an extract of the first article as it is represented in the variable *corpus*. Highlighted in blue it is shown the words which require some lexical change and in red those tagged as *stop-words*

\t U.S. housing starts and permits fell in December after hefty gains the prior month, adding to a raft of weak data that have raised concerns over the health of the economy. \n\t\t Groundbreaking dropped 2.5 percent to a seasonally adjusted annual pace of 1.15 million units, the Commerce Department said on Wednesday.

[3]http://svn.tartarus.org/snowball/trunk/website/algorithms/english/stop.txt?revision=431&view=markup

After the lexical analysis and *stop-words* removal, the text resutls as follows,

> *us housing starts permits fell december hefty gains prior month adding raft weak data raised concerns health economy groundbreaking dropped percent seasonally adjusted annual pace million units commerce department said wednesday*

Stemming

Automatic truncation consists on removing suffixes in order to analyze the information. It logic behind is based on words with same root usually have similar meanings so they can be represented by the same term which is the root. *Porter* algorithm was used to provide this representation and it is implemented by the R function *stemDocument* which, uses as a default the *Snowball*[4] stemmer for English.

The following text shows another extract from an article obtained from the proposed corpus before and after applying the lexical analysis, *stop-words* removal.

> *unilever discusses impact artificial intelligence will consumers advertisers use artificial intelligence ai advertising set grow top marketing chief told cnbc brands*

After stemming the result is as follows,

> *unilev discuss impact artifici intellig will consum advertis use artifici intellig ai advertis set grow top market chief told cnbc brand*

Document term matrices

After performing the previously detailed steps, the analysis continues through the generation of the frequencies matrices (or document term matrices DTM) on which the terms are represented on each row and the columns are represented by columns,

$$\left[\begin{array}{c|cccc} w_1 & f_{1,1} & f_{1,2} & \cdots & f_{1,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n & f_{n,1} & f_{n,2} & \cdots & f_{n,k} \end{array}\right]$$

[4]`http://snowball.tartarus.org/algorithms/english/`

where w_i is the word and $f_{i,k}$ is the frequency of the word i in document k. As we iteratively read each article in the corpus, every time a new word appears, a new row is added. Assumming a new word w_{n+1} appears in the third document, the matrix becomes:

$$\left[\begin{array}{c|ccc} w_1 & f_{1,1} & f_{1,2} & f_{1,3} \\ \vdots & \vdots & \vdots & \vdots \\ w_n & f_{n,1} & f_{n,2} & f_{n,3} \\ w_{n+1} & 0 & 0 & f_{n,3} \end{array}\right]$$

An so on until all corpus is completed. Over this matrix the different weighting functions will be applied.

```
dtm = DocumentTermMatrix(stemmed)
m = as.matrix(dtm)
# get word counts in decreasing order
word_freqs = sort(colSums(m), decreasing=TRUE)
# create a data frame with words and their frequencies
dm = data.frame(word=names(word_freqs), freq=word_freqs)
#generate the matrix
#first document
if (i == 1) {
  corpus_matrix = dm
  corpus_matrix[,1] = as.character(corpus_matrix[,1])
  corpus_matrix[,2] = as.numeric(corpus_matrix[,2])

} else {
  #create a column
  corpus_matrix[,(i+1)] = rep(0,length(corpus_matrix[,1]))
  for(j in 1:length(dm[,1])){
    #if word doesnt exist add row
    if(length(which(as.character(dm$word[j]) == corpus_matrix[,1]))==0){
      row = c(as.character(dm$word[j]),
              rep(0,(length(corpus_matrix)-2)),
              as.numeric(dm$freq[j]))
      corpus_matrix = rbind(corpus_matrix,row)
    }else{
      corpus_matrix[which(as.character(dm$word[j]) ==
      as.character(corpus_matrix$word)),(i+1)] = as.numeric(dm$freq[j])
    }
  }

}
```

Weighting functions

From the DTM the weighting functions can be directly calculated by aggregating the frequencies per row or per columns. Notice that aggregating per column we obtain the frequencies per document so the metric will be local, while per row we obtain the metric for each word across documents generating a global metric.

$$\left[\begin{array}{c|cccc|c} w_1 & f_{1,1} & f_{1,2} & \cdots & f_{1,k} & \sum_{i=1}^{k} f_{1,i} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ w_n & f_{n,1} & f_{n,2} & \cdots & f_{n,k} & \sum_{i=1}^{k} f_{n,i} \\ \hline & \sum_{i=1}^{n} f_{i,1} & \sum_{i=1}^{n} f_{i,2} & \cdots & \sum_{i=1}^{n} f_{i,k} & \sum_{i=1}^{n}\sum_{j=1}^{k} f_{i,j} \end{array}\right]$$

As a local weighting function we will study the *weighted term frequency* (WTF). This metric uses the frequency of each term and weights by the sum of the frequencies of each document. Using the same nomenclature as before, the function is defined by,

$$F : WTF(\vec{t_i}, \vec{d_j}) = \frac{f_{ij}}{\sum_{t_p \in d_j} f_{pj}} \quad (1)$$

```
#local weights
local_total_freq = colSums(corpus_matrix[2:(length(corpus_matrix[1,])-3)])
for (i in 1: N) {
  weighted_term_freq = corpus_matrix[,(i+1)]/
                        rep(local_total_freq[i],length(corpus_matrix$word))
  #take words which belong to the document
  w = corpus_matrix$word[which(corpus_matrix[,(i+1)]!=0)]
  f = weighted_term_freq[which(weighted_term_freq!=0)]
  write(paste(format(round(as.numeric(f),5),nsmall=5),w,sep=""),
        file="fichero_invertido.txt",ncolumns=length(w),append=TRUE)
}
```

For instance the term *percent* appears 8 time in the first document, which has a total of 478 words. Then applying (1) the local weight for document 1 becomes $WTF(\textit{percent,doc1}) = \frac{4}{478} = 0.00836$.

As a global function *term frequecy - inverse document frequency* (TF-IDF) was selected. It multiplies the global frequency of the term by the logarithm of the total number of documents (N) divided by the number of documents where the term appears ($df(\vec{t_i})$),

$$F : TF - IDF(\vec{t_i}, \vec{d_j}) = f_{ij} \times \log(\frac{N}{df(\vec{t_i})}) \quad (2)$$

```
#number of docs
N = length(corpus_url[,1])
#get number of docs where each word appear
corpus_matrix$n_docs = apply(corpus_matrix[2:(length(corpus_matrix[1,])-1)],
                      1,function(x) length(which((x!=0)==TRUE)))
#get global weights TF-IDF
corpus_matrix$TF_IDF = corpus_matrix$total *
                      log(rep(N,length(corpus_matrix$n_docs))
                      /corpus_matrix$n_docs)
```

Using the same term we see on the DTM that the corresponding frequencies correponding to the 14 documents are:

$$f(percent) = \{f_1, f_2, ..., f_{14}\} = \{4, 2, 0, 0, 5, 10, 0, 0, 3, 1, 13, 0, 1, 1\}$$

with a total of 40 appearances in 9 documents and applying (2) we obtain $40 \times \log(\frac{14}{9}) = 7.675$

Results

The script generates 17 files, one with the vocabulary, one with the inverted file, one with the global weights and 14 with the local weights correponding to each *url* with the following format

```
#file Vocabulario.txt
cnbc video europ hous news start asia financ portfolio...

#file Pesos_Globales.txt
0.000cnbc  0.000video  0.000europ 17.513hous...

#file Pesos_Locales_doc1.txt
0.03347cnbc 0.02720video 0.01674europ 0.01464hous...
...
#file Pesos_Locales_doc14.txt
0.03009cnbc 0.01862video 0.01146europ 0.01146news...

#file Fichero_Invertido.txt
cnbc -> doc1:16 doc2:17 doc3:16 doc4:17 doc5:16...
video -> doc1:13 doc2:13 doc3:13 doc4:13 doc5:13...
...
```

Conclusions and comments

We have used several representations of a corpus which consist on several HTML documents. As a first step the text was processed by performing the

(a) Frecuencias Totales (b) TF-IDF

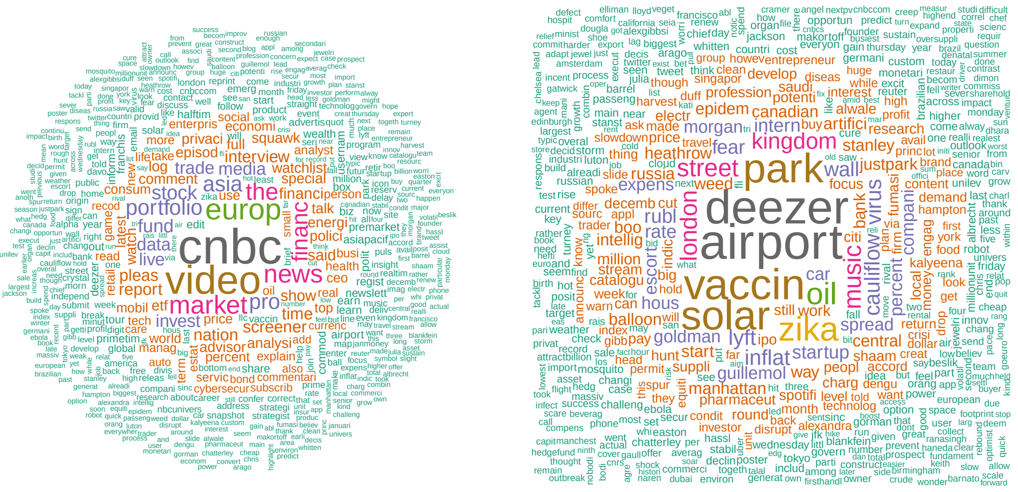

Figure 1: Wordcloud using global frequencies (a) and TF-IDF (b). TF-IDF also show those weights with value 0, thus its squaered shape.

lexical analysis, stop-words removal and stemming. Then several weight functions were used for information retrieval.

It can be checked after the analysis of the 14 CNBC articles that several terms such as *cnbc* and *hour* are extremely frequent given their representation in the local weight function. As a first conclusion, this information might not be relevant as we already know the procedence of the articles so it might not be interenting for the user. In order to solve this situation the list of stop-words can be extended to provide a more exhaustive analysis by, for instance, including the term *cnbc* as a stop-word. Notice that this kind of decisions are problem-specific and this is a step by step general guide to perform this analysis.

Different weight functions result in different conclusion as can be seen in Figure 1 which show the wordcloud using different metrics. It can be seen that the term *cnbc* obtains high importance using global weights while its weight is zero using TF-IDF. Again, the weight function to be used is also problem-specific.

References

[1] Tony Breyal. Blog-Reference-Functions. GitHub. `https://raw.github.com/tonybreyal/Blog-Reference-Functions/master/R/htmlToText/htmlToText.R`.

[2] Ingo Feinerer, Kurt Hornik, Artifex Software, Inc. *Package tm.* `https://cran.r-project.org/web/packages/tm/tm.pdf`.

[3] M.F. Porter. Program, 14 130–137 *An algorithm for suffix stripping.* `http://tartarus.org/martin/PorterStemmer/def.txt`. 1980.