Extracting Social & Semantic Networks from Twitter

Wouter van Atteveldt

June 29, 2016

Installing packages

This handout uses the following packages, which you will need to install if you haven't already:

```
install.packages("devtools")
install.packages("RTextTools")
devtools::install_github("kasperwelbers/corpus-tools")
devtools::install_github("kasperwelbers/semnet")
```

Downloading Tweets

Note that you need to setup OAuth first, see the handout on "Using API's from R"

Let's download tweets from UK MP's before and after the brexit referendum:

```
library(twitteR)
mptweets_after = searchTwitter("list:Tweetminster/UKMPs", n=2000, since="2016-06-23", until="2016-06-25
mptweets_before = searchTwitter("list:Tweetminster/UKMPs", n=2000, until="2016-06-22", since="2016-06-15
mptweets_after = plyr::ldply(mptweets_after, as.data.frame)
mptweets_before = plyr::ldply(mptweets_before, as.data.frame)
```

Constructing DTM

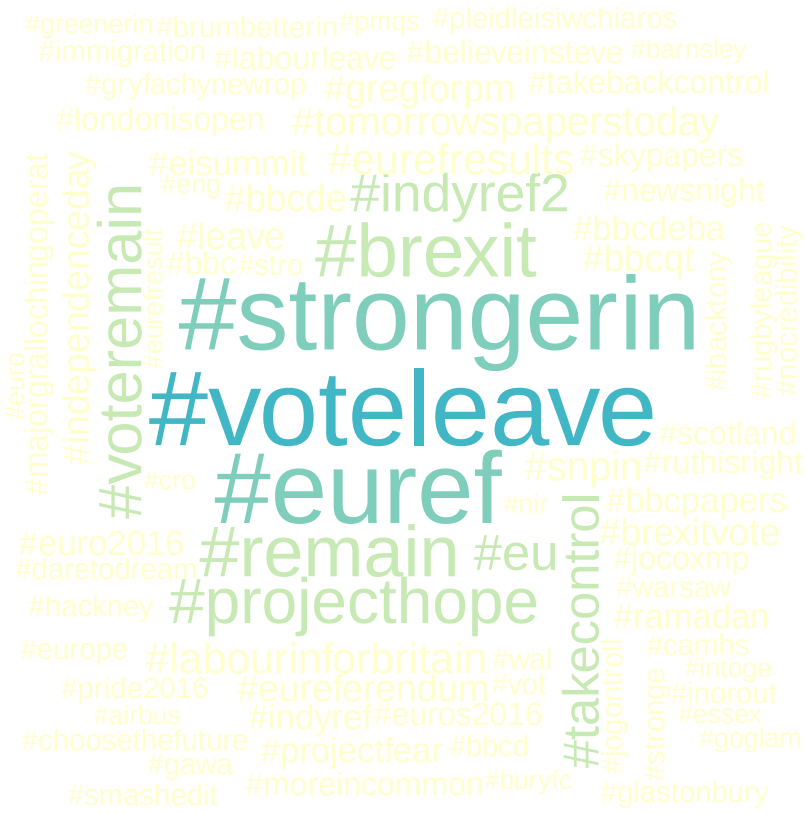
The first step in frequency based analysis is to create a document-term matrix, containing the frequency of each word in each document (tweet).

We use the `create_matrix` function from the `RTextTools` package. This can strip punctuation etc. automatically, but since we want to preserve #hashtags and @mentions but strip hyperlinks, we need to do some custom processing:

```
library(RTextTools)
tweets = rbind(mptweets_after, mptweets_before)
text = gsub("https://.*?( |$)", " ", tweets$text)
text = gsub("[^A-Za-z0-9#@_]+", " ", text)
dtm = create_matrix(text, removePunctuation = F)
rownames(dtm) = tweets$id
```

From this DTM we can create subsets using the regular matrix/data frame subsetting syntax (`dtm[rows, columns]`). For example, we can create a DTM with only the hash tags and visualize that as a tag cloud:

```
library(corpustools)
dtm.hash = dtm[, grepl("#", colnames(dtm))]
dtm.wordcloud(dtm.hash, freq.fun = sqrt)
```



We can use the `term.statistics` command to get frequency information from a corpus, for example to get an overview of the most frequent hash tags:

```
stats = term.statistics(dtm.hash)
stats = arrange(stats, -termfreq)
head(stats)
```

term	characters	number	nonalpha	termfreq	docfreq	reldocfreq	tfidf
#bbcdebate	10	FALSE	TRUE	624	621	0.4373239	0.8839396
#voteleave	10	FALSE	TRUE	170	170	0.1197183	1.4732418
#strongerin	11	FALSE	TRUE	154	154	0.1084507	1.9522668
#euref	6	FALSE	TRUE	147	147	0.1035211	2.1079351
#brexit	7	FALSE	TRUE	69	69	0.0485915	3.3220470
#remain	7	FALSE	TRUE	66	66	0.0464789	2.5418535

Comparing Corpora

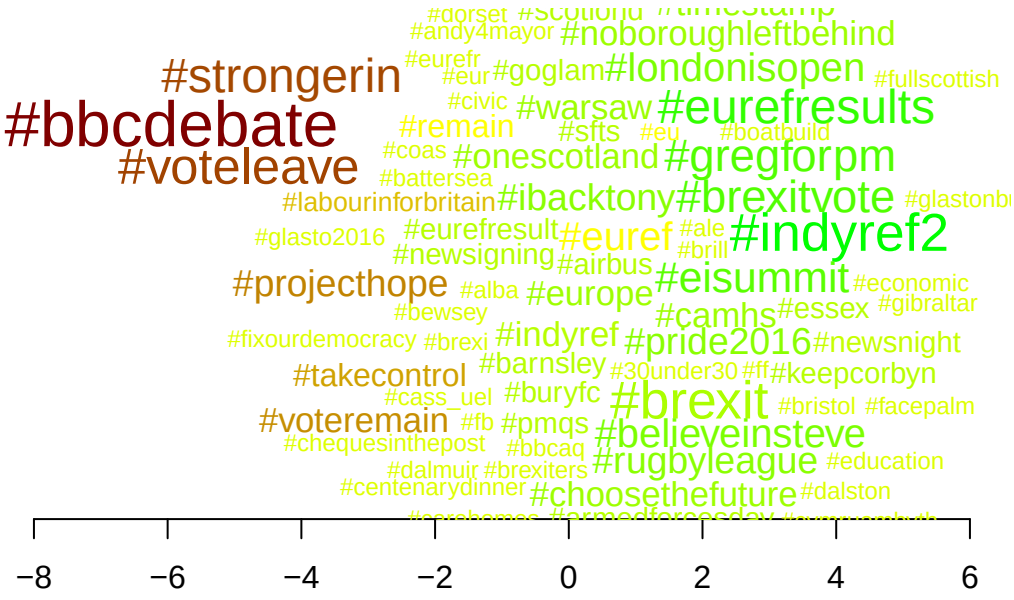
A useful way to get more useful information from a corpus is by comparing different subsets of it, for example from different sources, periods, or split on a specific keyword. In this example, we can compare the 'before' and 'after' tweets to see how hash tag use changed. To show the most typical 'before' tags, we take the 'after' tweets as selected corpus and sort on ascending overrepresentation (so the most underrepresented words are displayed first).

```
cmp = corpora.compare(dtm.hash, select.rows = as.character(mptweets_after$id))
cmp = arrange(cmp, over)
head(cmp)
```

term	termfreq.x	termfreq.y	termfreq	relfreq.x	relfreq.y	over	chi
#bbcdebate	0	624	624	0	0.3802559	0.0026229	258.604522
#voteleave	0	170	170	0	0.1035954	0.0095607	54.058623
#strongerin	0	154	154	0	0.0938452	0.0105435	48.572414
#projecthope	0	48	48	0	0.0292505	0.0330574	14.365319
#voteremain	0	39	39	0	0.0237660	0.0403779	11.621367
#takecontrol	0	26	26	0	0.0158440	0.0593683	7.699502

We can also display this information in a contrast plot, showing the most typical 'before' tags to the left, and the 'after' tags to the right:

```
with(arrange(cmp, -chi)[1:100, ],
     plotWords(x=log(over), words = term, wordfreq = chi, random.y = T))
```



Constructing a social network

To create the 'who mentions who' network, we select only the @-words from the DTM, and then extract the tweet-mention pairs. By adding (and normalizing) the screen name of the tweet author as sender and stripping the at-sign from the addressee, we have data frame of sender-addressee pairs:

```
dtm.ats = dtm[, grepl("@", colnames(dtm))]
mentions = dtm.to.df(dtm.ats)
mentions$sender = tolower(tweets$screenName[match(mentions$doc, tweets$id)])
mentions$addressee = gsub("@", "", mentions$term)
head(mentions)
```

doc	term	freq	sender	addressee
746492820370956292	@mehdirhasan	1	nazshahbfd	mehdirhasan
746491708196683776	@nfmusic	1	jreedmp	nfmusic
746490824377143296	@iainr0bertson	1	andyburnhammp	iainr0bertson
746490802453549056	@chukaumunna	1	jreedmp	chukaumunna
746490419916210176	@henriettasandwi	1	jreedmp	henriettasandwi
746490221328498688	@carolinejmolloy	1	jreedmp	carolinejmolloy

We can aggregate this to see who mentions whom, and remove all self-references and references to non-MPs:

```
edges = aggregate(cbind(weight=mentions$freq), mentions[c("sender", "addressee")], sum)
edges = edges[edges$addressee %in% edges$sender, ]
edges = edges[edges$addressee != edges$sender, ]
head(edges)
```

	sender	addressee	weight
42	matthancockmp	agriffithsmp	1
60	acunninghammp	albertowenmp	1
72	rogmull	alexsalmond	1
88	andrewselous	alistairburtmp	1
89	anna_soubry	alistairburtmp	1
90	ben4ipswich	alistairburtmp	1

Now, we can create an `igraph` object from the edge list:

```
g = igraph::graph.data.frame(edges)
vcount(g)
```

```
## [1] 210
```

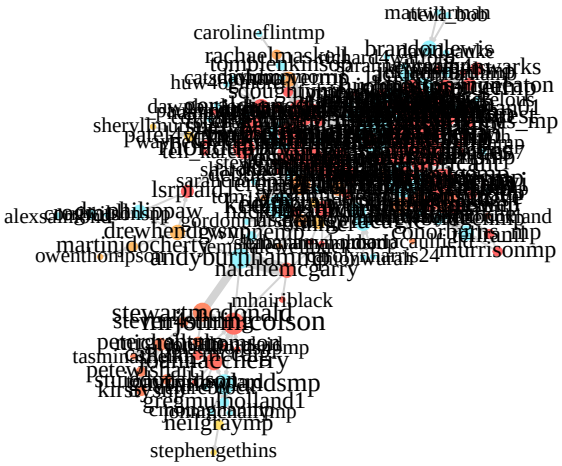
So we now have a social network with 210 vertices (MPs), and we can use that to e.g. select the most central MP (using betweenness centrality):

```
head(sort(betweenness(g), decreasing = T))
```

keeleymp	521.6667
jessphillips	447.6667
jreynoldsmp	439.5000
andyburnhammp	402.6667
mrjohnnicolson	351.5000
chukaumunna	318.0000

To visualize this network, we first select only the largest component, apply clustering, and then use the built-in plot function:

```
library(semnet)
g = decompose(g, max.comps=1, min.vertices=10)[[1]]
V(g)$cluster = edge.betweenness.community(g)$membership
g = setNetworkAttributes(g, cluster_attribute=V(g)$cluster)
plot(g)
```



Constructing a semantic network

To construct a semantic network of all co-occuring words, we can use the `coOccurenceNetwork` function from the `semnet` package:

```
dtm.words = dtm[,!grepl("[#@]", colnames(dtm))]
g = coOccurenceNetwork(dtm.words)
```

```
## Note: method with signature 'CsparseMatrix#Matrix#missing#replValue' chosen for function '[<-',
##  target signature 'dgCMatrix#nsCMatrix#missing#numeric'.
##  "Matrix#nsparseMatrix#missing#replValue" would also be valid
```

```
vcount(g)
```

```
## [1] 5869
```

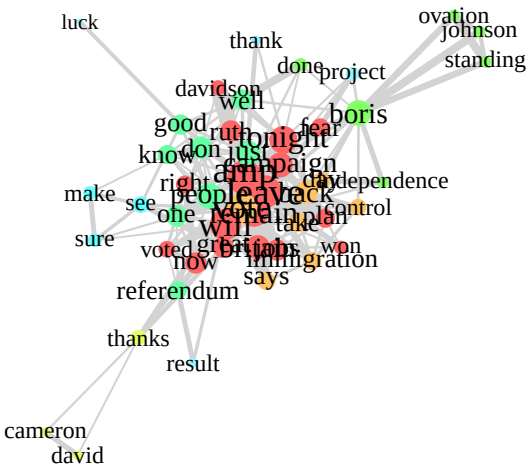
This network is too large to display directly, but we can extract the backbone of the 50 most important words and then select the largest component:

```
g = getBackboneNetwork(g, max.vertices = 50)
g = decompose(g, max.comps=1, min.vertices=10)[[1]]
vcount(g)
```

```
## [1] 48
```

Now, we can cluster the graph and plot it:

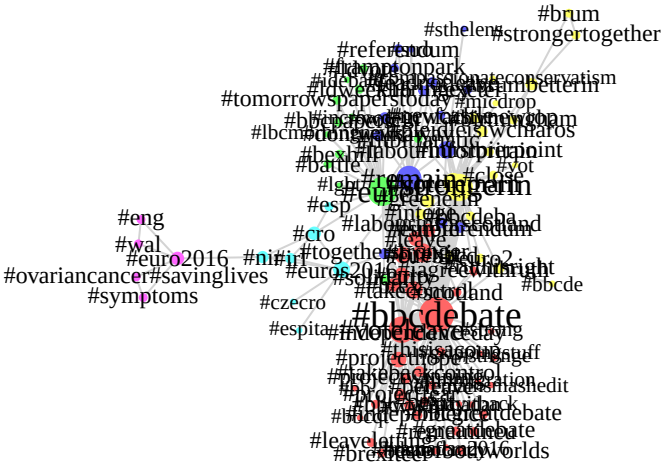
```
V(g)$cluster = edge.betweenness.community(g)$membership
g = setNetworkAttributes(g, cluster_attribute=V(g)$cluster)
plot(g)
```



As you can see, you would probably want to apply more preprocessing, e.g. removing stopwords, lemmatizing/stemming, selecting only certain parts-of-speech, and/or applying dictionaries or thesauri.

With tweets, one thing we can also do is limit ourselves to the semantic network of hash tags:

```
before = mptweets_before$id
dtm.tags = dtm[rownames(dtm) %in% before, grepl("#", colnames(dtm))]
g = coOccurenceNetwork(dtm.tags)
g = decompose(g, max.comps=1, min.vertices=10)[[1]]
V(g)$cluster = edge.betweenness.community(g)$membership
g = setNetworkAttributes(g, cluster_attribute=V(g)$cluster)
plot(g)
```



Combining the social and semantic networks

As a final example, we can create a network that combines the social network (who mentions whom) with the hash tag network (who uses which tags).

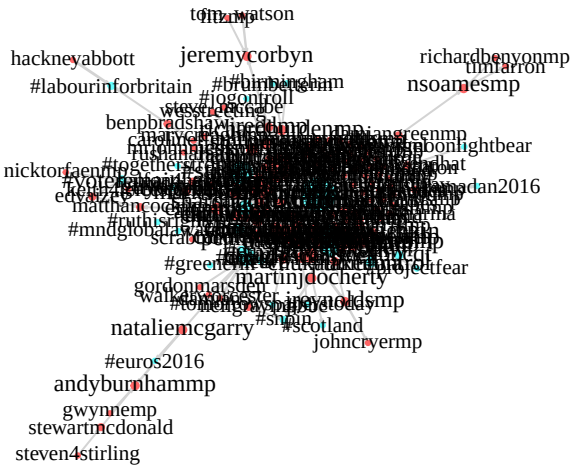
To do this, we first create the edges for the sender -> tag networks:

```
dtm.tags = dtm.tags[row_sums(dtm.tags) > 0, col_sums(dtm.tags) > 0]
tags = dtm.to.df(dtm.tags)
tags$sender = tolower(tweets$screenName[match(tags$doc, tweets$id)])
tags$addressee = gsub("@", "", tags$term)
tagedges = aggregate(cbind(weight=tags$freq), tags[c("sender", "addressee")], sum)
head(tagedges)
```

sender	addressee	weight
tomtugendhat	#barnabyfestival	1
ben4bath	#bath	1
huwmerriman	#battle	1
rebeccaharrismp	#bb	1
bernardjenkin	#bbc	1
chrisgreenmp	#bbc	1

Now, we can combine these edges with the social network edges created above, and create a graph for all edges with weight >= 2:

```
g = igraph::graph.data.frame(subset(rbind(edges, tagedges), weight>=2))
g = decompose(g, max.comps=1, min.vertices=10)[[1]]
g = setNetworkAttributes(g, cluster_attribute=grepl("#", V(g)$name))
plot(g)
```



Exporting graphs

You can export graphs to a number of formats using the `write.graph` function:

```
write.graph(g, file="graph.net", format="pajek")
```

This function does not support the gephi format, however. To export to gephi we use the rgexf package:

```
library(rgexf)
gefx = igraph.to.gexf(g)
print(gefx, file="graph.gefx")
```