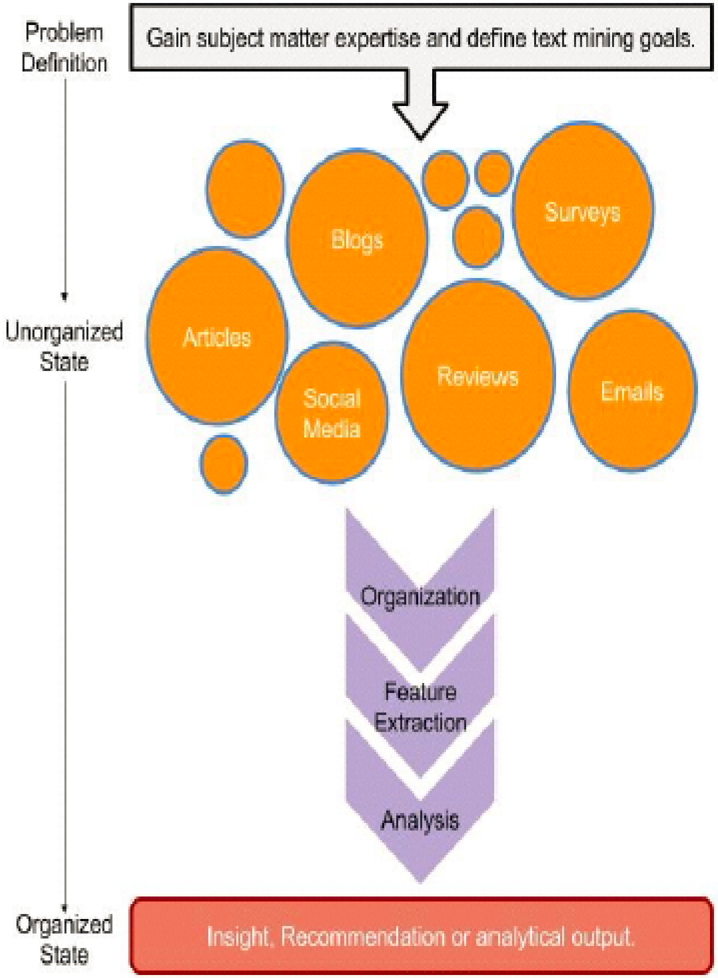
Text Data Analysis

The Process of distilling actionable insights from text

Text mining workflow



1 - Problem definition & specific goals

2 - Identify text to be collected

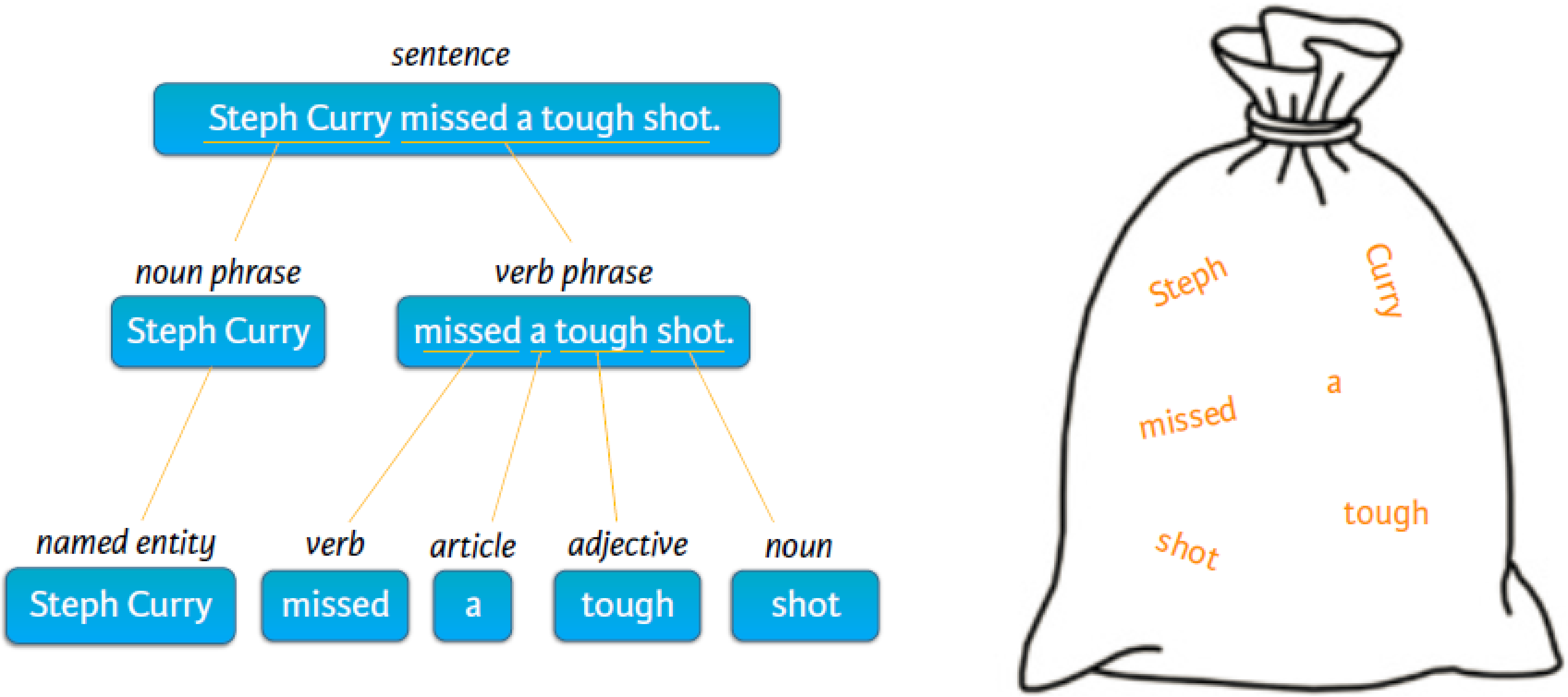
3 - Text organization

4 - Feature extraction

5 - Analysis

6 - Reach an insight, recommendation or output

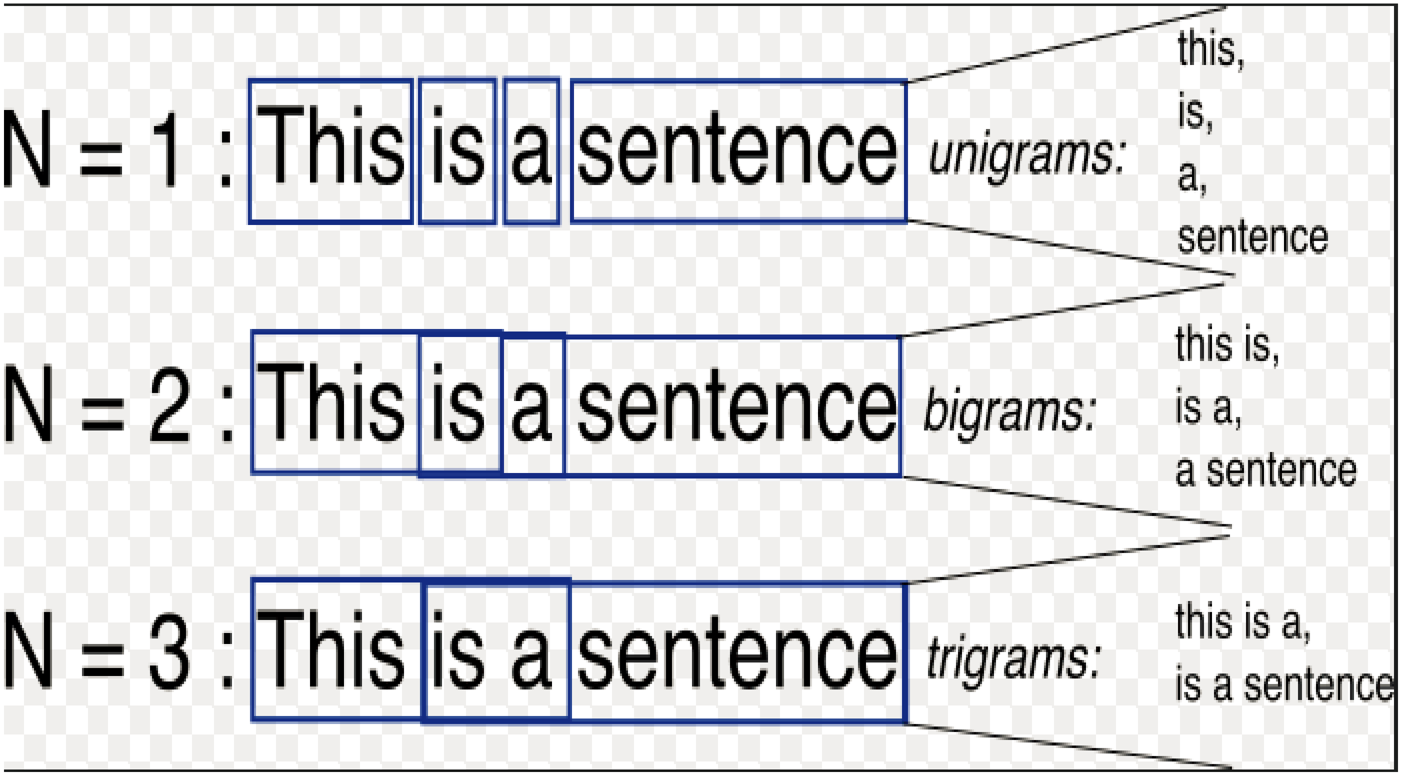
Semantic parsing vs. bag of words



Bag of words

- bag of words text mining represents a way to count terms, or *n-grams*, across a collection of documents. Consider the following sentences:
- text<- “data analytics is the new sensation of the modern times, often misunderstood as only graphs and dashboards. Its time for businesses to think beyond charts and start leveraging the science behind the data ”
- Manually counting words in the sentences above is a pain!

N-grams


N = 1 : This is a sentence
unigrams:
this,
is,
a,
sentence
N = 2 : This is a sentence
bigrams:
this is,
is a,
a sentence
N = 3 : This is a sentence
trigrams:
this is a,
is a sentence

Steps to follow

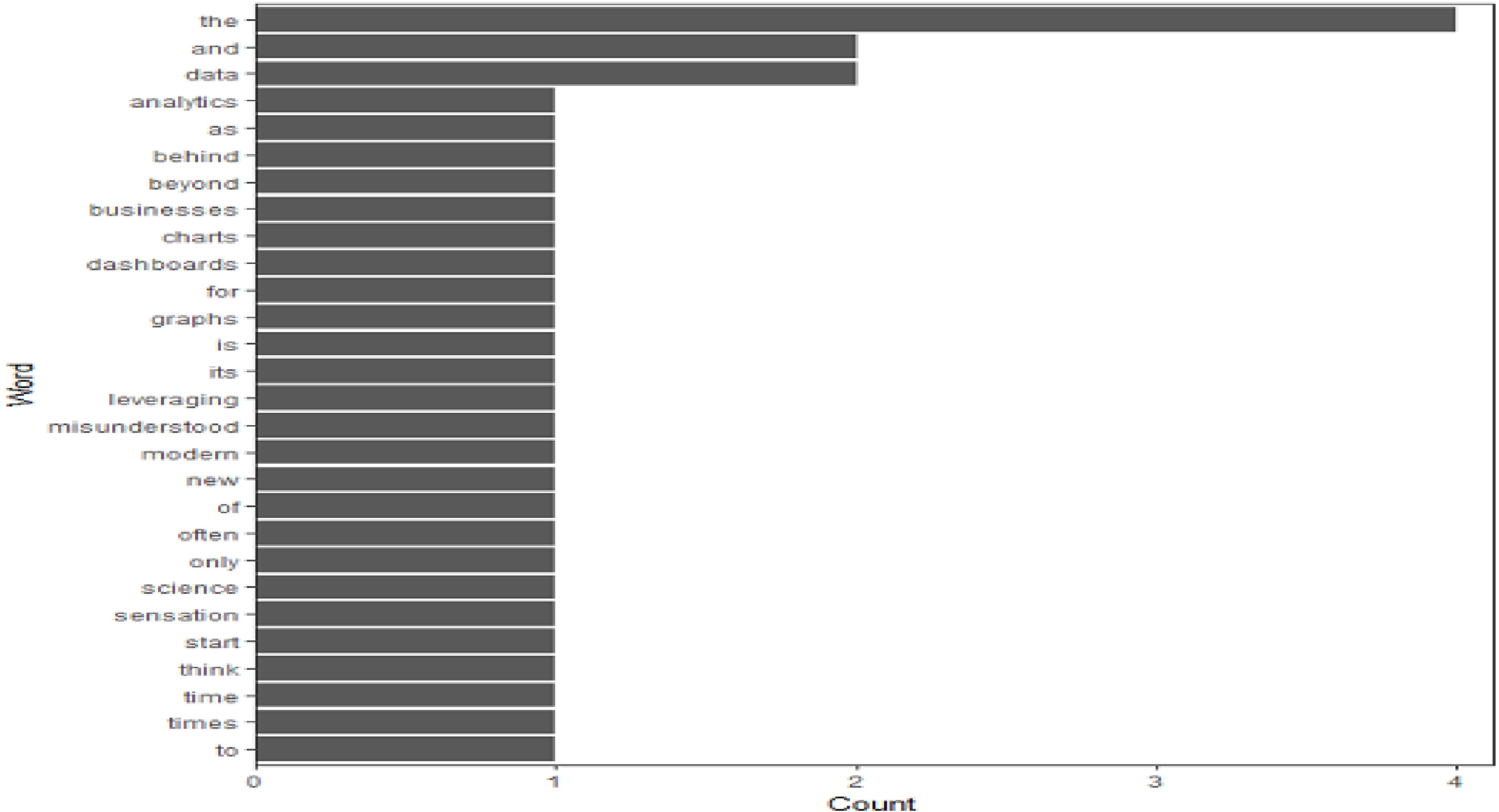
```
# Load qdap

# Print text to the console

# Find the 4 most frequent terms: term_count

# Plot term_count
```

```
> text <- "data analytics is the new sensation of the modern times, o
ften   misunderstood as only graphs and dashboards. Its time for busi
nesses  to think beyond charts and start leveraging the science behin
d the data"
> library(qdap)
> freq_terms<-freq_terms(text,4)
> plot(freq_terms)
> 
```



Load some text

- # Import text data
- tweets<-read.csv(" XiaomiIndiaTweets.csv",stringsAsFactors = FALSE)
- # View the structure of tweets
- str(tweets)
- # Print out the number of rows in tweets
- nrow(tweets)
- # Isolate text from tweets: xiaomi_tweets
- xiaomi_tweets <- tweets$text
- xiaomi_tweets

Make the vector a VCorpus object

- We have loaded your text data as a vector called xiaomi_tweets in the last exercise.
- Your next step is to convert this vector containing the text data to a corpus.
- Corpus is a collection of documents, but it's also important to know that in the tm domain, R recognizes it as a data type.
- There are two kinds of the corpus data type, the *permanent corpus*, PCorpus, and the *volatile corpus*, VCorpus. In essence, the difference between the two has to do with how the collection of documents is stored in your computer. In this course, we will use the volatile corpus, which is held in your computer's RAM rather than saved to disk, just to be more memory efficient.
- To make a volatile corpus, R needs to interpret each element in our vector of text, xiaomi_tweets, as a document.
- And the tm package provides what are called *Source* functions to do just that!
- In this exercise, we'll use a Source function called VectorSource() because our text data is contained in a vector.
- The output of this function is called a Source object.

Console ~/ML/bangaloresession1/

```
> library(tm)
> xiaomi_source<-VectorSource(xiaomi_tweets)
>
```

VCorpus

- As we've converted our vector to a Source object, we pass it to another tm function, VCorpus(), to create our volatile corpus.
- The VCorpus object is a nested list, or list of lists. At each index of the VCorpus object, there is a PlainTextDocument object, which is essentially a list that contains the actual text data (content), as well as some corresponding metadata (meta).
- It can help to visualize a VCorpus object to conceptualize the whole thing.
- For example, to examine the contents of the second tweet in xiaomi_corpus, you'd subset twice.
- Once to specify the second PlainTextDocument corresponding to the second tweet and again to extract the first (or content) element of that PlainTextDocument:

- xiaomi_corpus[[15]][1]

Steps

- ## xiaomi_source is already in your workspace
- # Make a volatile corpus: xiaomi_corpus
- xiaomi_corpus <- VCorpus(xiaomi_source)
- # Print out xiaomi_corpus
- xiaomi_corpus
- # Print data on the 15th tweet in xiaomi_corpus
- xiaomi_corpus[[15]]
- # Print the content of the 15th tweet in xiaomi_corpus
- xiaomi_corpus[[15]][1]

Console ~/ML/bangaloresession1/

```
> library(tm)
> xiaomi_source<-VectorSource(xiaomi_tweets)
> xiaomi_corpus <- VCorpus(xiaomi_source)
> xiaomi_corpus
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 1561
> xiaomi_corpus[[15]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 140
> xiaomi_corpus[[15]][1]
$content
[1] "Mi fans! We bring you super saving deals on your favourite Mi pr
oducts. The Republic Day sale is live. Check it now… https://t.co/PGA
DMTHyFh"
```

Make a VCorpus from a data frame

- Because another common text source is a data frame, there is a Source function called DataframeSource(). The DataframeSource() function must have a specific structure:
- Column **one** must be called doc_id and contain a unique string for each row.
- Column **two** must be called text with "UTF-8" encoding (pretty standard).
- Any other columns, **3+** are considered meta-data and will be retained as such.

Common preprocessing functions

TM Function	Description	Before	After
`tolower()`	Makes all text lowercase	Starbucks is from Seattle.	starbucks is from seattle.
`removePunctuation()`	Removes punctuation like periods and exclamation points	Watch out! That coffee is going to spill!	Watch out That coffee is going to spill
`removeNumbers()`	Removes numbers	I drank 4 cups of coffee 2 days	I drank cups of coffee days ago.
`stripWhiteSpace()`	Removes tabs and extra spaces	I like coffee.	I like coffee.
`removeWords()`	Removes specific words (e.g. "the", "of") defined by the data scientist	The coffee house and barista he visited were nice, she said hello.	The coffee house barista visited nice, said hello.

Preprocessing in practice

Common cleaning functions from tm

- Common preprocessing functions include:
- tolower(): Make all characters lowercase
- removePunctuation(): Remove all punctuation marks
- removeNumbers(): Remove numbers
- stripWhitespace(): Remove excess whitespace

Console ~/ML/bangaloresession1/

```
> text <- "data analytics is the new sensation of the modern times, o
ften   misunderstood as !!!! : only graphs and dashboards. Its time f
or businesses  to think beyond charts and start leveraging the scienc
e behind the data in 2018   "
> removeNumbers(text)
[1] "data analytics is the new sensation of the modern times, often
 misunderstood as !!!! : only graphs and dashboards. Its time for bus
inesses  to think beyond charts and start leveraging the science behi
nd the data in    "
> removePunctuation(text)
[1] "data analytics is the new sensation of the modern times often
misunderstood as   only graphs and dashboards Its time for businesses
  to think beyond charts and start leveraging the science behind the
data in 2018   "
> stripWhitespace(text)
[1] "data analytics is the new sensation of the modern times, often m
isunderstood as !!!! : only graphs and dashboards. Its time for busin
esses to think beyond charts and start leveraging the science behind
the data in 2018 "
>
```

Cleaning with qdap

- The qdap package offers other text cleaning functions. Each is useful in its own way and is particularly powerful when combined with the others.
- bracketX(): Remove all text within brackets (e.g. "It's (so) cool" becomes "It's cool")
- replace_number(): Replace numbers with their word equivalents (e.g. "2" becomes "two")
- replace_abbreviation(): Replace abbreviations with their full text equivalents (e.g. "Sr" becomes "Senior")
- replace_contraction(): Convert contractions back to their base words (e.g. "shouldn't" becomes "should not")
- replace_symbol() Replace common symbols with their word equivalents (e.g. "$" becomes "dollar")

Stop Words

- Often there are words that are frequent but provide little information. So you may want to remove these so-called *stop words*. Some common English stop words include "I", "she'll", "the", etc. In the tm package, there are 174 stop words on this common list.

Stop Words

- In fact, when you are doing an analysis you will likely need to add to this list. In our xiaomi tweet example, all tweets contain “xiaomi", so it's important to pull out that word in addition to the common stop words. Leaving it in doesn't add any insight and will cause it to be overemphasized in a frequency analysis.
- Using the c() function allows you to add new words (separated by commas) to the stop words list. For example, the following would add "word1" and "word2" to the default list of English stop words:
- all_stops <- c("word1", "word2", stopwords("en"))
- Once you have a list of stop words that makes sense, you will use the removeWords() function on your text.
- RemoveWords()takes two arguments: the text object to which it's being applied and the list of words to remove

Tweets example

- stopwords("en")
- removeWords(xiaomi_tweets, stopwords("en"))
- new_stops <- c("xiaomi", "redmi", stopwords("en"))
- data<-removeWords(xiaomi_tweets, new_stops)

Intro to word stemming and stem completion

- One useful preprocessing step involves *word stemming* and *stem completion*.
- The tm package provides the stemDocument() function to get to a word's root. This function either takes in a character vector and returns a character vector, or takes in a PlainTextDocument and returns a PlainTextDocument.

For example,

- stemDocument(c("computational", "computers", "computation"))

Console ~/ML/bangaloresession1/

```
> # Create complicate
> complicate <- c("complicated", "complication", "complicatedly")
>
> # Perform word stemming: stem_doc
> stem_doc <- stemDocument(complicate)
>
> # Create the completion dictionary: comp_dict
> comp_dict <- "complicate"
>
> # Perform stem completion: complete_text
> complete_text <- stemCompletion(stem_doc, comp_dict)
>
> # Print complete_text
> complete_text
      complic      complic      complic
"complicate" "complicate" "complicate"
```

Word stemming and stem completion on a sentence

text_data<- "In a complicated haste, Ram rushed to fix a new complication, too complicatedly.“

This sentence contains the same three forms of the word "complicate" that we saw in the previous exercise. The difference here is that even if you called stemDocument() on this sentence, it would return the sentence without stemming any words. Take a moment and try it out in the console. Be sure to include the punctuation marks.

Stemming on sentence

- This happens because stemDocument() treats the whole sentence as one word.
- In other words, our document is a character vector of length 1, instead of length n, where n is the number of words in the document.
- To solve this problem, we first remove the punctation marks with the removePunctuation()function, which you learned a few exercises back.
- We then strsplit()this character vector of length 1 to length n,
- unlist(), then proceed to stem and re-complete.

Console ~/ML/bangaloresession1/

```
> text_data<- "In a complicated haste, Tom rushed to fix a new compl
ication, too complicatedly."
> # Remove punctuation: rm_punc
> rm_punc <- removePunctuation(text_data)
>
> # Create character vector: n_char_vec
> n_char_vec <- unlist(strsplit(rm_punc, split = ' '))
> #
> # Perform word stemming: stem_doc
> stem_doc <- stemDocument(n_char_vec)
>
> # Print stem_doc
> stem_doc
 [1] "In"      "a"       "complic" "hast"    "Tom"     "rush"
 [7] "to"      "fix"     "a"       "new"     "complic" "too"
[13] "complic"
>
> # Re-complete stemmed document: complete_doc
> complete_doc <- stemCompletion(stem_doc,comp_dict)
> # Print complete_doc
> complete_doc
         In            a      complic         hast          Tom
         ""           "" "complicate"           ""           ""
       rush           to          fix            a          new
         ""           ""           ""           ""           ""

    complic          too      complic
"complicate"          "" "complicate"
```

Apply preprocessing steps to a corpus

```
clean_corpus <- function(corpus){
  corpus <- tm_map(corpus, stripWhitespace)
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, content_transformer(replace_abbreviation))
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, removeWords, c(stopwords("en"), "xiaomi","redmi"))
  return(corpus)
}
```

Console ~/ML/bangaloresession1/

```
> clean_corpus <- function(corpus){
+     corpus <- tm_map(corpus, stripWhitespace)
+     corpus <- tm_map(corpus, removePunctuation)
+     corpus <- tm_map(corpus, content_transformer(tolower))
+     corpus <- tm_map(corpus, content_transformer(replace_abbreviat
ion))
+     corpus <- tm_map(corpus, removeNumbers)
+     corpus <- tm_map(corpus, removeWords, c(stopwords("en"), "xiao
mi","redmi"))
+     return(corpus)
+ }
> # Apply your customized function to the tweet_corp: clean_corp
> clean_corp<-clean_corpus(xiaomi_corpus)
>
> # Print out a cleaned up tweet
> clean_corp[[227]][1]
$content
[1] "check     shotonmia entries   mi fans  one   think   best    yo
ure u… httpstcojizojhm"
> # Print out the same tweet in original form
> tweets$text[227]
[1] "Check out some of the  #ShotOnMiA1 entries by our Mi fans! Whic
h one do you think is the best in these?\nIf you're u… https://t.co/
ji087ZoJHM"
>
```

TDM vs. DTM

	Tweet 1	Tweet 2	Tweet 3	...	Tweet N
Term 1	0	0	0	0	0
Term 2	1	1	0	0	0
Term 3	1	0	0	0	0
...	0	0	3	1	1
Term M	0	0	0	1	0

Term Document Matrix (TDM)

	Term 1	Term 2	Term 3	...	Term M
Tweet 1	0	1	1	0	0
Tweet 2	0	1	0	0	0
Tweet 3	0	0	0	3	0
...	0	0	0	1	1
Tweet N	0	0	0	1	0

Document Term Matrix (DTM)

Console ~/ML/bangaloresession1/

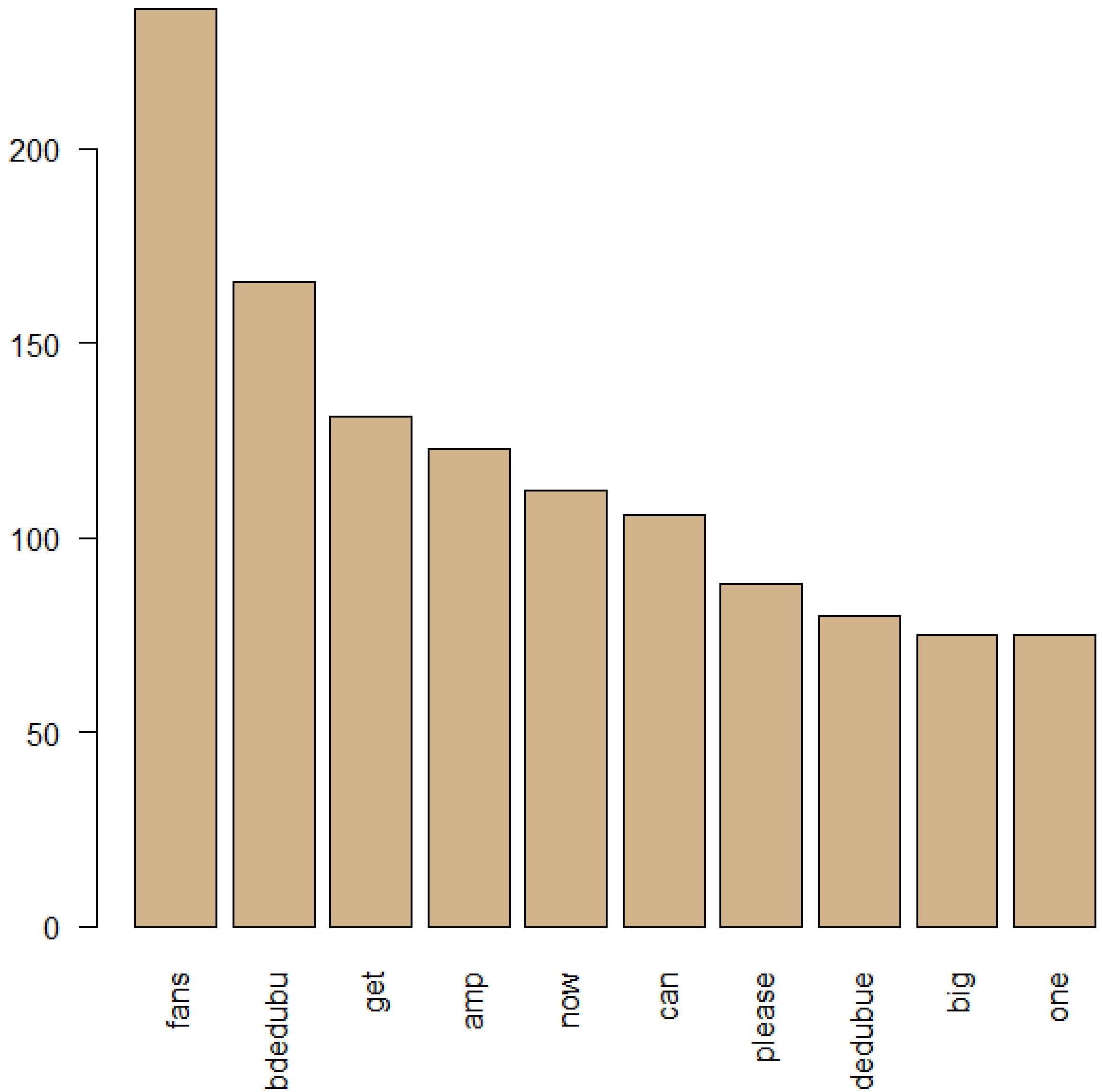
```
> xiaomi_tdm<-TermDocumentMatrix(clean_corp)
> xiaomi_dtm<-DocumentTermMatrix(clean_corp)
> xiaomi_m<- as.matrix(xiaomi_dtm)
> dim(xiaomi_m)
[1] 1561 4401
> xiaomi_m[100:103,2390:2393]
     Terms
Docs  hugo hundreds hungama hungamacom
  100    0        0       0          0
  101    0        0       0          0
  102    0        0       0          0
  103    0        0       0          0
> xiaomi_m[148:150,2587:2590]
     Terms
Docs  justoverspoken jyotigujarati kabiradvani kai
  148              0             0           0   0
  149              0             0           0   0
  150              0             0           0   0
>
```

Console ~/ML/bangaloresession1/

```
> xiaomi_t<-as.matrix(xiaomi_tdm)
> xiaomi_t[2587:2590,148:150]
                 Docs
Terms             148 149 150
  justoverspoken    0   0   0
  jyotigujarati     0   0   0
  kabiradvani       0   0   0
  kai               0   0   0
> xiaomi_t[2087:2090,95:100]
                    Docs
Terms                95 96 97 98 99 100
  httpstcornbwkfhou   0  0  0  0  0   0
  httpstcornplffwfk   0  0  0  0  0   0
  httpstcornsligl     0  0  0  0  0   0
  httpstcoroydv       0  0  0  0  0   0
>
```

Frequent terms with tm

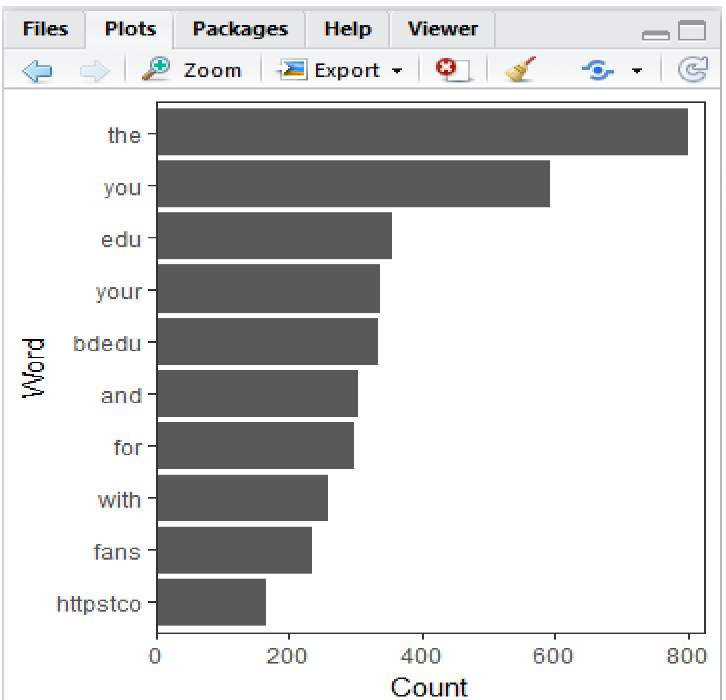
```
Console ~/ML/bangaloresession1/
> ## xiaomi_tdm is still loaded in your workspace
> # Create a matrix: xiaomi_m
> xiaomi_m <- as.matrix(xiaomi_tdm)
> # Calculate the rowSums: term_frequency
> term_frequency <- rowSums(xiaomi_m)
> # Sort term_frequency in descending order
> term_frequency <- sort(term_frequency, decreasing = TRUE)
> # View the top 10 most common words
> term_frequency[1:10]
       fans  eduaubdedubu           get           amp
        236           166           131           123
        now           can        please eduaubdedubue
        112           106            88            80
        big           one
         75            75
> # Plot a barchart of the 10 most common words
> barplot(term_frequency[1:10], col = "tan", las = 2)
```

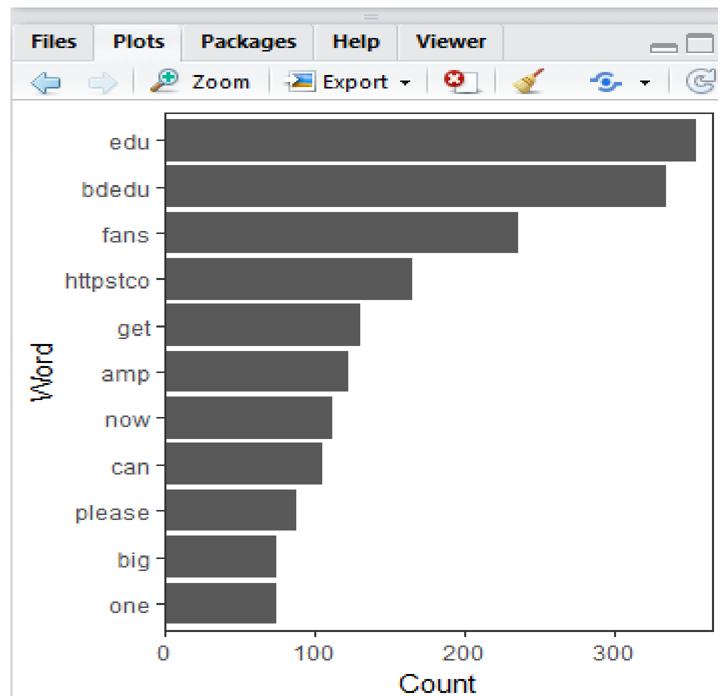

200
150
100
50
0
fans
get
amp
now
can
please
big
one

Frequent terms with qdap

Console ~/ML/bangaloresession1/

```
> library(qdap)
> frequency <- freq_terms(tweets$text, top = 10, at.least = 3,
+                         stopwords = "Top200Words")
> plot(frequency)
> frequency2 <- freq_terms(tweets$text, top = 10, at.least = 3,
+                          stopwords = tm::stopwords("english"))
> plot(frequency2)
```





Distance matrix and dendrogram

- A simple way to do word cluster analysis is with a dendrogram on your term-document matrix. Once you have a TDM, you can call dist() to compute the differences between each row of the matrix.
- Next, you call hclust() to perform cluster analysis on the dissimilarities of the distance matrix. Lastly, you can visualize the word frequency distances using a dendrogram and plot().

Make a distance matrix and dendrogram from a TDM

- We can apply them to text. But first, you have to limit the number of words in your TDM using removeSparseTerms() from tm.

- Why would you want to adjust the sparsity of the TDM/DTM?

- TDMs and DTMs are sparse, meaning they contain mostly zeros.

- Remember that 1000 tweets can become a TDM with over 3000 terms! You won't be able to easily interpret a dendrogram that is so cluttered, especially if you are working on more text.
- A good TDM has between 25 and 70 terms.
- The lower the sparse value, the more terms are kept.
- The closer it is to 1, the fewer are kept.
- This value is a percentage cutoff of zeros for each term in the TDM.

Console ~/ML/bangaloresession1/

```
>  dim(xiaomi_tdm)
[1] 4401 1561
> tdm1 <- removeSparseTerms(xiaomi_tdm, sparse = 0.95)
> tdm1
<<TermDocumentMatrix (terms: 8, documents: 1561)>>
Non-/sparse entries: 1006/11482
Sparsity           : 92%
Maximal term length: 13
Weighting          : term frequency (tf)
> tdm2 <- removeSparseTerms(xiaomi_tdm, sparse = 0.975)
> tdm2
<<TermDocumentMatrix (terms: 38, documents: 1561)>>
Non-/sparse entries: 2629/56689
Sparsity           : 96%
Maximal term length: 14
Weighting          : term frequency (tf)
> 
```

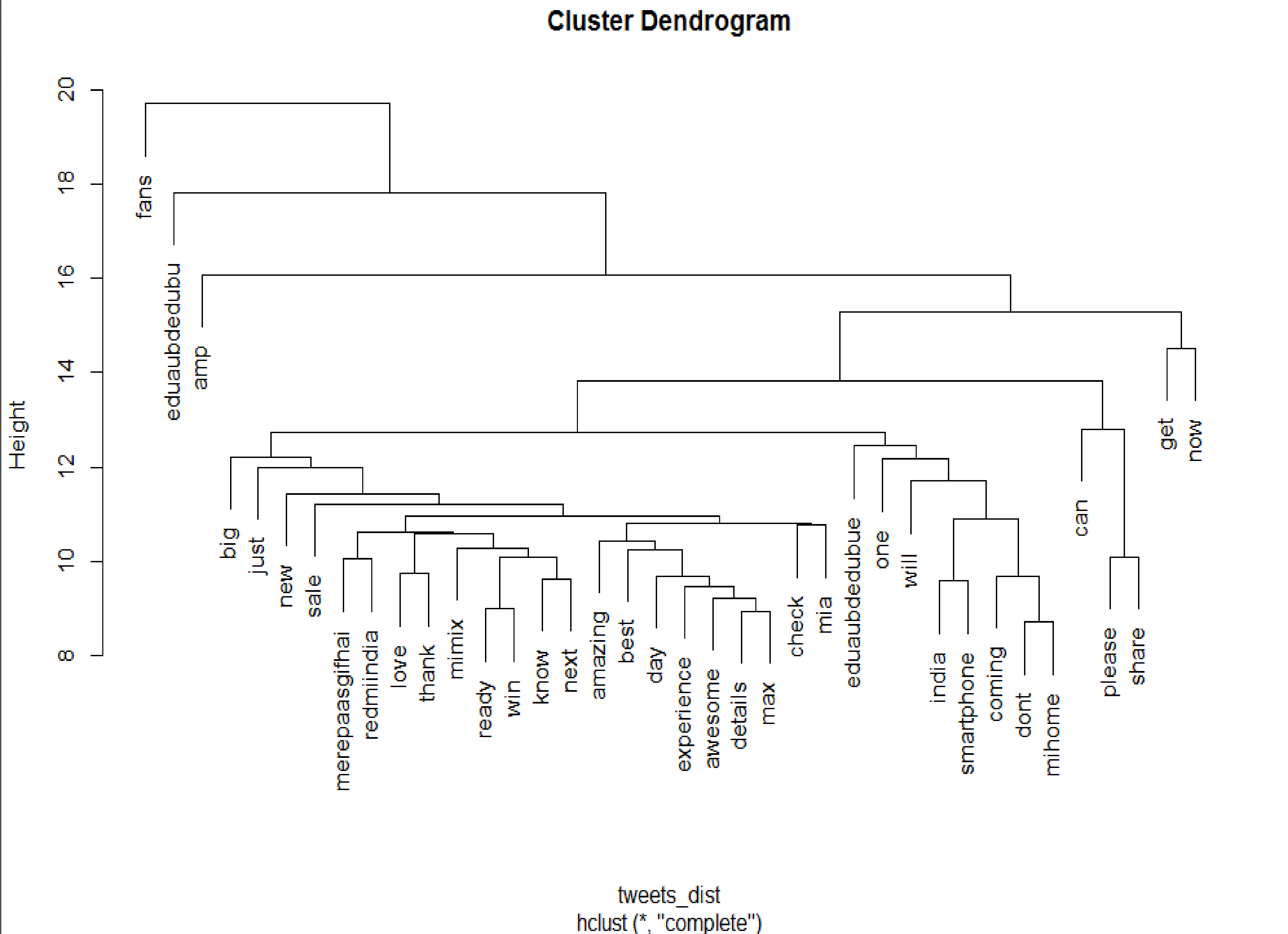
Put it all together: a text based dendrogram

- Let’s work to make your first text-based dendrogram. Remember, dendrograms reduce information to help you make sense of the data.
- This is much like how an average tells you something, but not everything, about a population. Both can be misleading.
- With text, there are often a lot of nonsensical clusters, but some valuable clusters may also appear.
- A peculiarity of TDM and DTM objects is that you have to convert them first to matrices (with as.matrix()), then to data frames (with as.data.frame()), before using them with the dist()function.

Source

Console ~/ML/bangaloresession1/

```
> tdm2 <- removeSparseTerms(xiaomi_tdm, sparse = 0.975)
> tdm_m<-as.matrix(tdm2)
> tdm_df<-as.data.frame(tdm_m)
> tweets_dist<-dist(tdm_df)
> hc<-hclust(tweets_dist)
> plot(hc)
>
```


Cluster Dendrogram
Height
20
18
16
14
12
10
8
fans
eduaubdedubu
amp
big
just
new
sale
merepaasgifhai
redmiindia
love
thank
mimix
ready
win
know
next
amazing
best
day
experience
awesome
details
max
check
mia
eduaubdedubue
one
will
india
smartphone
coming
dont
mihome
can
please
share
get
now
tweets_dist
hclust (*, "complete")

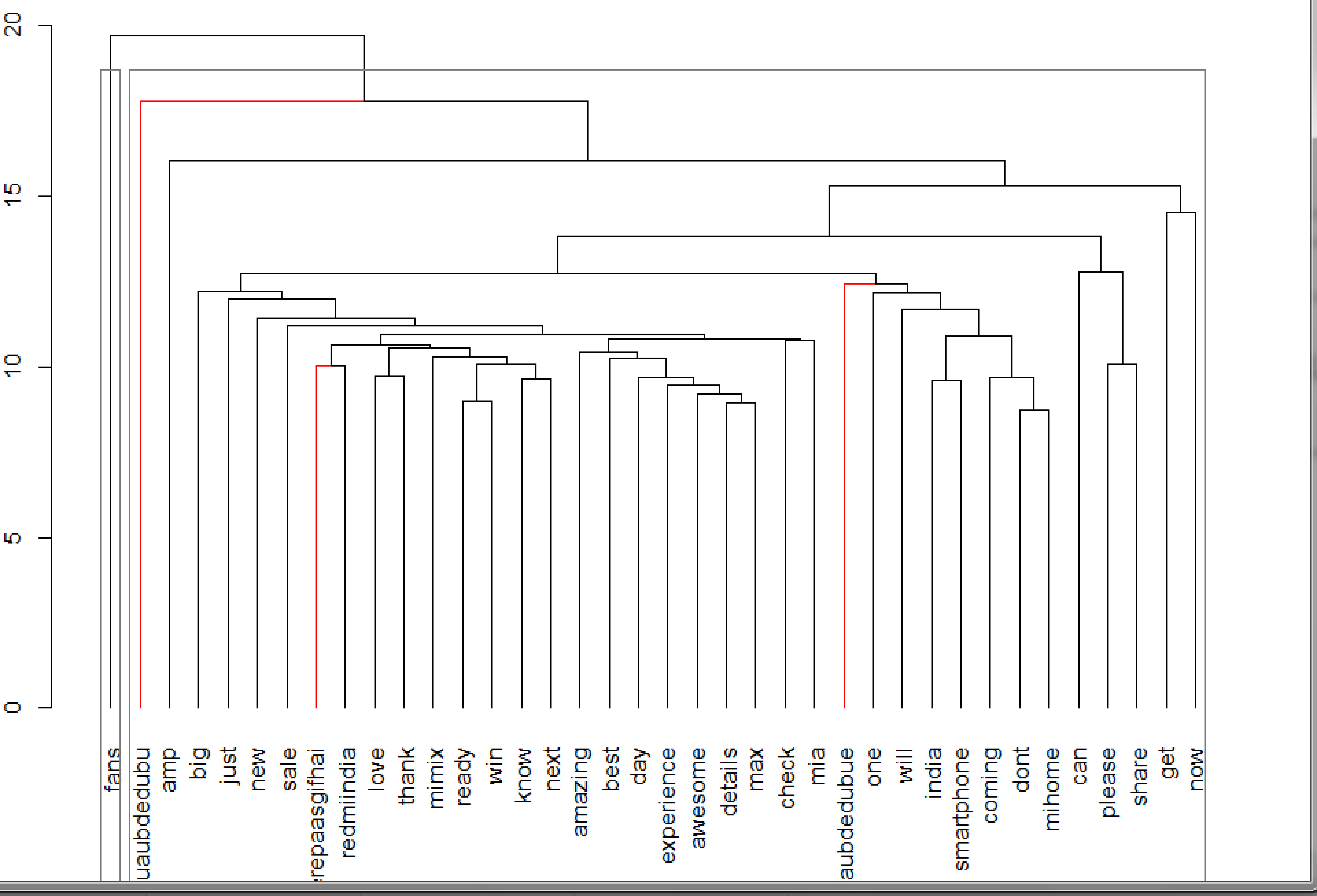
Dendrogram aesthetics

Source

Console ~/ML/bangaloresession1/

```
> library(dendextend)
> hcd<-as.dendrogram(hc)
> hcd <- branches_attr_by_labels(hcd, c("eduaubdedubu", "eduaubdedubue","merepaasgifhai"), "red")
> plot(hcd, main = "Better Dendrogram")
> rect.dendrogram(hcd, k = 2, border = "grey50")
>
```

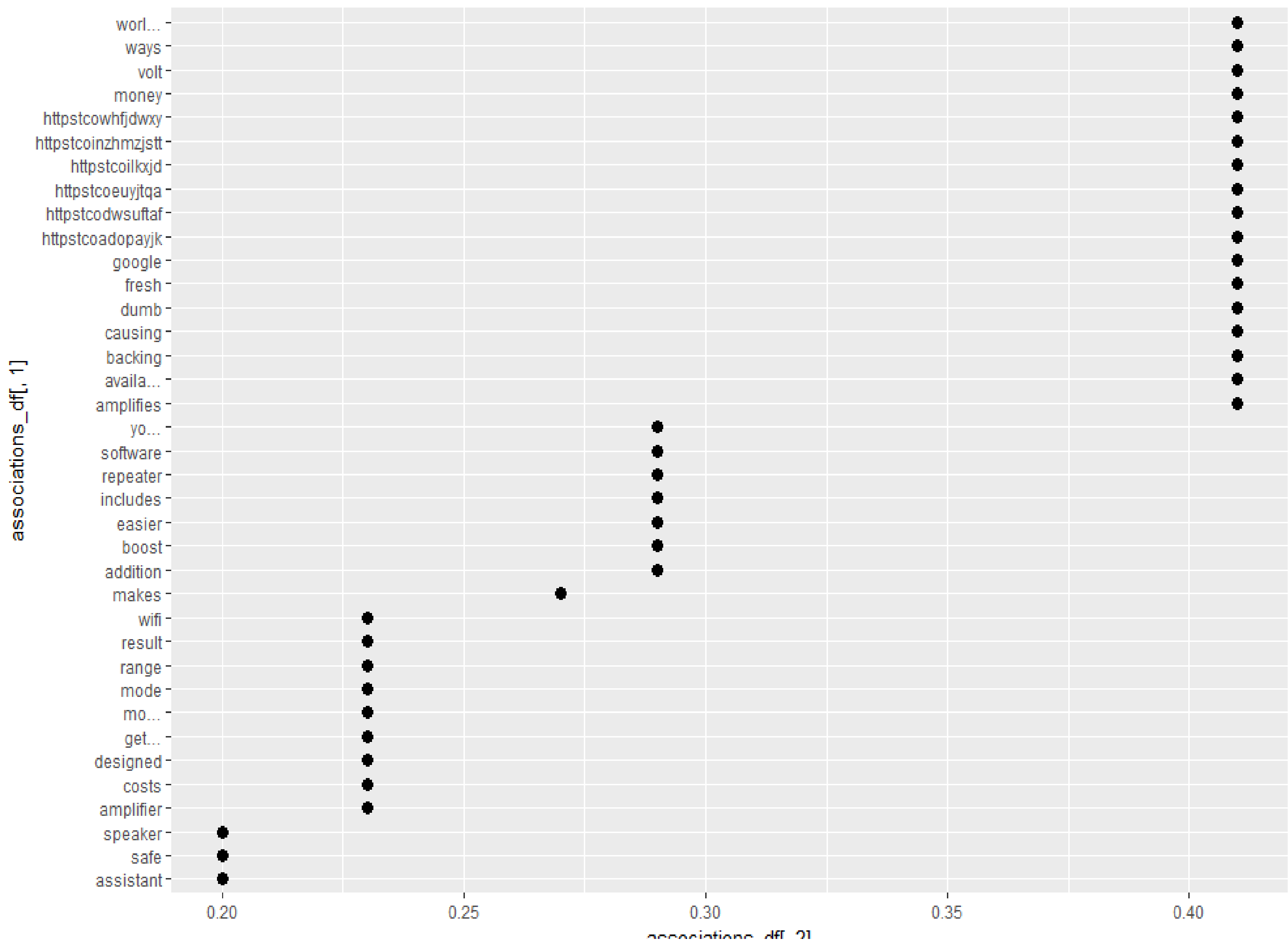
Better Dendrogram



Using word association

- Another way to think about word relationships is with the findAssocs() function in the tm package.
- For any given word, findAssocs() calculates its correlation with every other word in a TDM or DTM.
- Scores range from 0 to 1. A score of 1 means that two words always appear together, while a score of 0 means that they never appear together.
- To use findAssocs() pass in a TDM or DTM, the search term, and a minimum correlation.
- The function will return a list of all other terms that meet or exceed the minimum threshold.
- findAssocs(tdm, "word", 0.25) Minimum correlation values are often relatively low because of word diversity.
- Don't be surprised if 0.10 demonstrates a strong pairwise term association.

- # Create associations
- associations <- findAssocs(xiaomi_tdm, "smart", 0.2)
- # View the associations
- associations
- # Create associations_df
- associations_df <- list_vect2df(associations)[, 2:3]
- # Plot the associations_df values (don't change this)ggplot(associations_df, aes(y = associations_df[, 1])) +   geom_point(aes(x = associations_df[, 2]), data = associations_df, size = 3)



worl...
ways
volt
money
httpstcowhfjdwxy
httpstcoinzhmzjstt
httpstcoilkxjd
httpstcoeuyjtqa
httpstcodwsuftaf
httpstcoadopayjk
google
fresh
dumb
causing
backing
availa...
amplifies
yo...
software
repeater
includes
easier
boost
addition
makes
wifi
result
range
mode
mo...
get...
designed
costs
amplifier
speaker
safe
assistant
associations_df[, 1]
0.20
0.25
0.30
0.35
0.40