Text Mining Using Bag of Words

Contents

Text Mining Using Bag of Words

Refer to the Slides here

Preamble

The following code will set up a nice working environment:

(01) Jumping into Text Mining with bag of words

What is text Mining

General outline The idea of text mining is to reduce the amount of information, often there is too much information to work with. The goal is to reduce information and highlight what's important.

A Text mining work-flow looks like this:

There is a work-flow to follow:

1. Define the problem and specific goals
2. Identify the text to be collected
3. Organize the text
4. Extract Features from the text

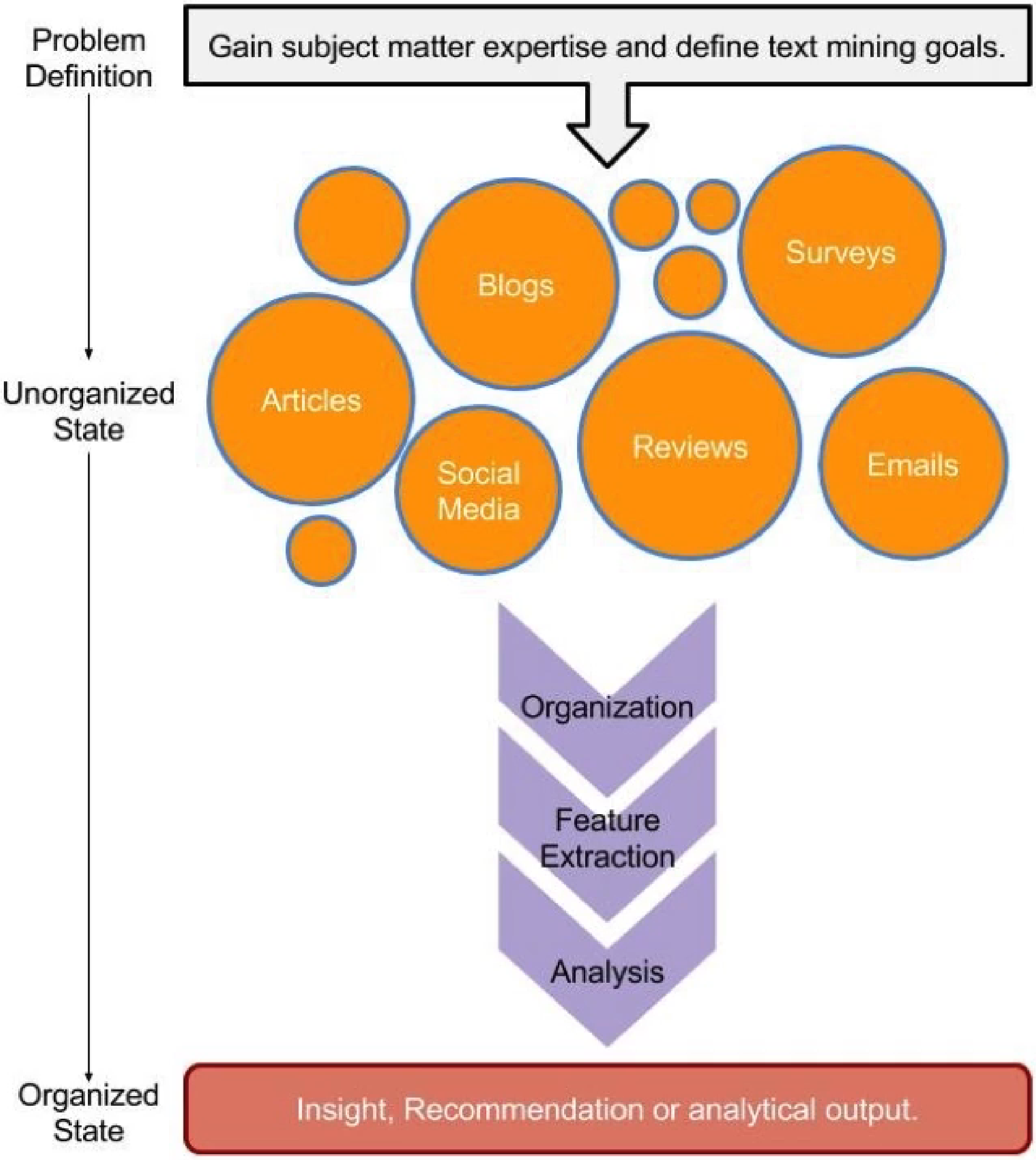


Figure 1: Image

5. Analyse the Text
6. Make a Conclusion

Types of Text Mining Strategies

Semantic parsing This is based on word syntax, you are interested in:

1. Word type
2. Word Order

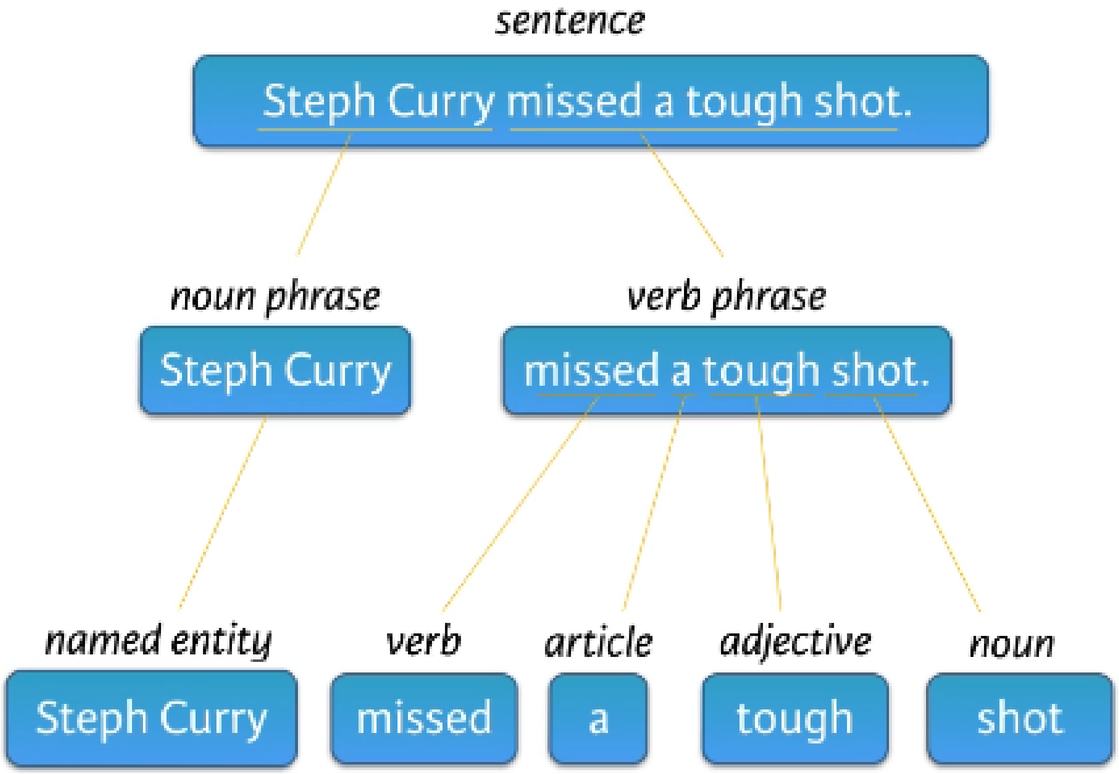


Figure 2: Image

Semantic parsing follows a tree structure to break up the text and apply tags to the text, such as adjective, part of a sentence, name etc.

So semantic parsing is feature rich because there are a lot of output variables.

Bag of Words The bag of words method does not care about word type or order and instead words are just attributes of the document.

This Course is only concerned with the bag of words method, later courses in the Text mining skill Track deal with semantic parsing.

Quick Taste of Text Mining

Some times it is sufficient to look at word frequency only, for example:

```
text <- "Text mining usually involves the process of structuring the input text. The overarching goal is

new_text <- "DataCamp is the first online learning platform that focuses on building the best learning e

# Load qdap
library(qdap)

# Print new_text to the console
new_text
```

```
## [1] "DataCamp is the first online learning platform that focuses on building the best learning experi
```

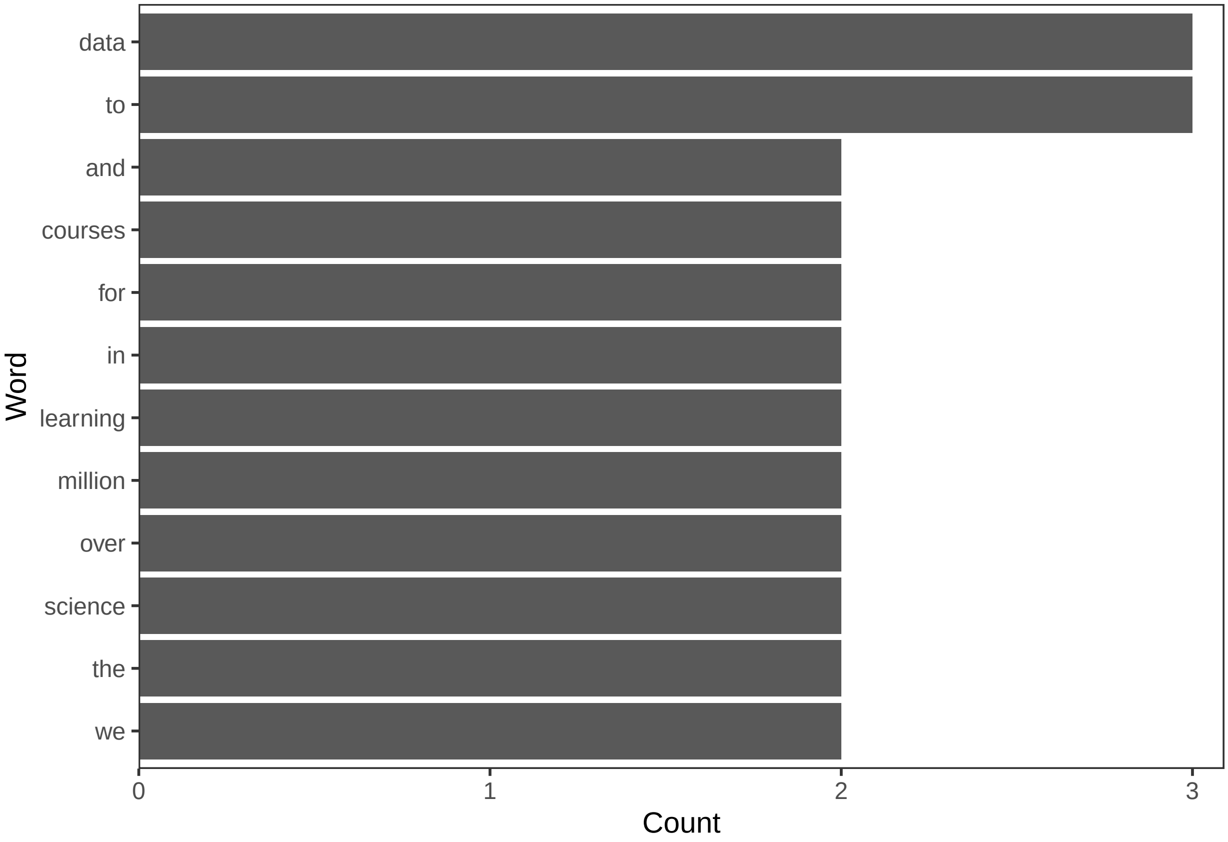
```
# Find the 10 most frequent terms: term_count
term_count <- freq_terms(new_text, 4)
summary(term_count)
```

```
##      WORD                FREQ
##  Length:12          Min.   :2.000
##  Class :character   1st Qu.:2.000
##  Mode  :character   Median :2.000
##                     Mean   :2.167
##                     3rd Qu.:2.000
##                     Max.   :3.000
```

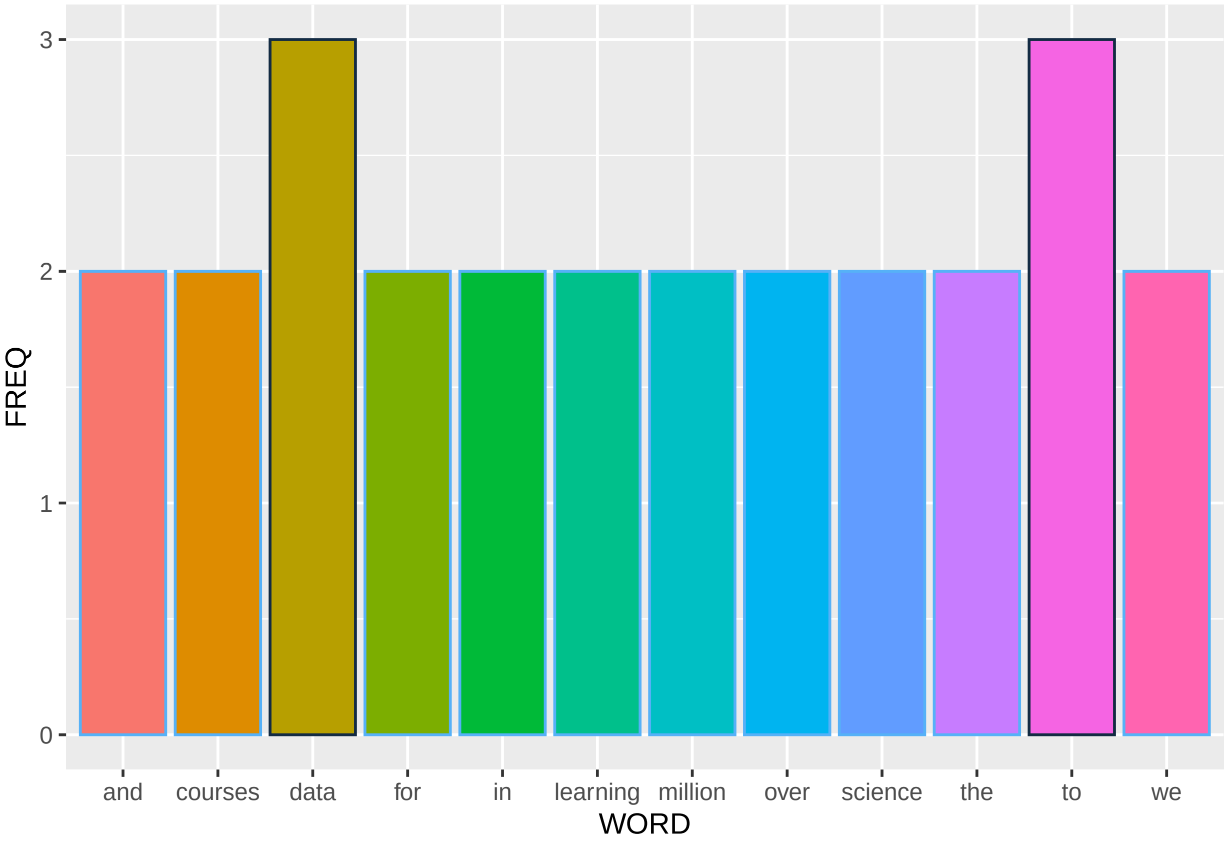
```
head(term_count)
```

```
##   WORD    FREQ
## 1 data       3
## 2 to         3
## 3 and        2
## 4 courses    2
## 5 for        2
## 6 in         2
```

```
# Plot term_count
plot(term_count)
```



```
# Plot Using GGPlot2
## Col (use this)
 ggplot(data = term_count, aes(x = WORD, y = FREQ, col = -FREQ, fill = WORD)) +
    geom_col() +
    guides(col = FALSE, fill = FALSE)
```



it's easy to tell that this is bag of words not semantic because the type of words isn't considered by the function that was used.

Getting Started

A corpus is a collection of documents, so first it is necessary to load in data and determine which columns correspond to features. When importing data it is important to specify `stringsAsFactors = FALSE` because otherwise the `read.csv()` function will import `character` strings as `factor` levels.

```r
tweets <- read.csv("~/Dropbox/Notes/DataSci/0DataSets/TextMining/coffee.csv",
    stringsAsFactors = FALSE)
str(tweets)
```

```
## 'data.frame':    1000 obs. of  15 variables:
##  $ num          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ text         : chr  "@ayyytylerb that is so true drink lots of coffee" "RT @bryzy_brib: Senior Mar
##  $ favorited    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ replyToSN    : chr  "ayyytylerb" NA NA NA ...
##  $ created      : chr  "8/9/2013 2:43" "8/9/2013 2:43" "8/9/2013 2:43" "8/9/2013 2:43" ...
##  $ truncated    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ replyToSID   : num  3.66e+17 NA NA NA NA ...
##  $ id           : num  3.66e+17 3.66e+17 3.66e+17 3.66e+17 3.66e+17 ...
##  $ replyToUID   : int  1637123977 NA NA NA NA NA NA 1316942208 NA NA ...
##  $ statusSource: chr  "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for i
##  $ screenName   : chr  "thejennagibson" "carolynicosia" "janeCkay" "AlexandriaOOTD" ...
##  $ retweetCount: int  0 1 0 0 2 0 0 0 1 2 ...
##  $ retweeted    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ longitude   : logi  NA NA NA NA NA NA ...
##  $ latitude    : logi  NA NA NA NA NA NA ...
```

```r
# isolate the text
coffee_tweets <- tweets$text
```

Now that the data is loaded in as a vector, it is necessary to convert the vector containing the text into a `corpus` data type. a `corpus` is a collection of documents but ***R*** recognizes it as a data type.

A corpus can be stored as a *volatile corpus* in memory as `VCorpus` as well as *Permanent corpus* which is saved to disk as `PCorpus`

to make a *volatile Corpus* ***R*** needs to interpret each element from the vector of text `coffee_tweets` as a as a document, the `tm` package provides `source` functions to do that:

```r
library(tm)
coffee_source <- tm::VectorSource(coffee_tweets)
coffee_source  %>% head()
```

```
## [1] "@ayyytylerb that is so true drink lots of coffee"
## [2] "RT @bryzy_brib: Senior March tmw morning at 7:25 A.M. in the SENIOR lot. Get up early, make yo
## [3] "If you believe in #gunsense tomorrow would be a very good day to have your coffee any place BUT
## [4] "My cute coffee mug. http://t.co/2udvMU6XIG"
## [5] "RT @slaredo21: I wish we had Starbucks here... Cause coffee dates in the morning sound perff!"
## [6] "Does anyone ever get a cup of coffee before a cocktail??"
```

Now that the vector is a source object it needs to be passed to `tm::VCorpus()` in order to create a *volatile corpus* object:

> The `VCorpus` object is a nested list, at each index there is a `PlainTextDocument` object which is a list containing text data known as `content` and some medadata known as `meta`, so in order to access the 15th document you would use `coffee_corpus[[15]]` and then the `content` or `meta` could be accessed with `[1]` or `[2]`.

```r
# Make the Corpus Object
coffee_corpus <- tm::VCorpus(coffee_source)

# Print the object
print(coffee_corpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1000
```

```r
# Print the 15th tweet
print(coffee_corpus[[15]])[1]
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 111

## $content
## [1] "@HeatherWhaley I was about 2 joke it takes 2 hands to hold hot coffee...then I read headline! #
```

```r
# or use content()
NLP::content(coffee_corpus[[15]])
```

```
## [1] "@HeatherWhaley I was about 2 joke it takes 2 hands to hold hot coffee...then I read headline! #
```

Make a VCorpus from a data Frame

If the text data is in a data frame you just use the `DataframeSource()` rather than extracting the vector. The data must have a specific structure:

- Column **One** must be called `doc_id` and each row must have a unique string.
- Column **two** must be called `text` with `UTF-*` encoding
- All following columns will be metadata.

```
# Create a dummy data frame
example_text <- data.frame(doc_id = 1:3, text = c("Text mining is fun", "Text analysis provides insights
example_text
```

```
##   doc_id                                text author       date
## 1      1                  Text mining is fun   John 1514953399
## 2      2     Text analysis provides insights    Jim 1514866998
## 3      3 qdap and tm are used in text mining   Bill 1514680598
```

```
# Create a text Source File
df_source <- tm::DataframeSource(example_text)
class(df_source)
```

```
## [1] "DataframeSource" "SimpleSource"    "Source"
```

```
head(df_source)
```

```
##   doc_id                                text author       date
## 1      1                  Text mining is fun   John 1514953399
## 2      2     Text analysis provides insights    Jim 1514866998
## 3      3 qdap and tm are used in text mining   Bill 1514680598
```

```
# Create a Volatile Corpus
df_corpus <- tm::VCorpus(df_source)
print(df_corpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 2
## Content:  documents: 3
```

```
# Examine the Meda Data
NLP::meta(df_corpus)
```

```
##   author       date
## 1   John 1514953399
## 2    Jim 1514866998
## 3   Bill 1514680598
```

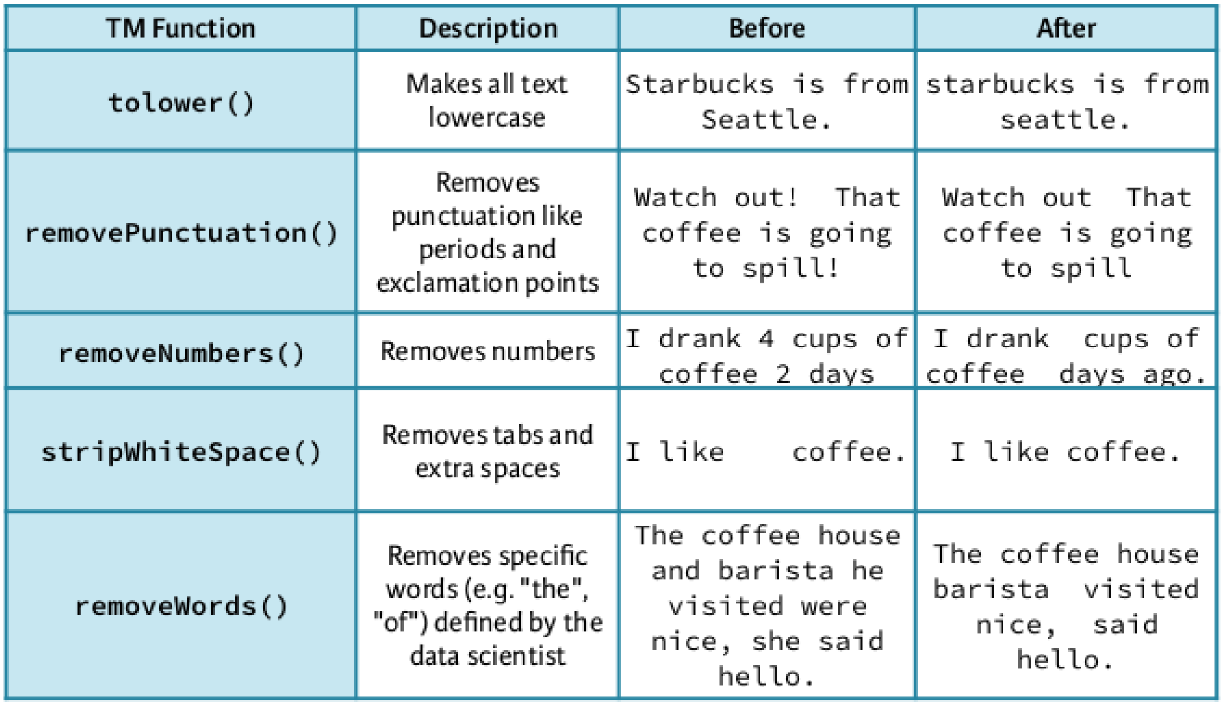
Cleaning and Preprocessing Text

Now that we have a corpus we need to start cleaning up the raw data

The *bag of words* text mining approach is such that cleaning helps to aggregate terms, for example it makes sense for the words `mine` , `miner` and `mining` to all simply be considered as the word `mine`.

Common preprocessing functions inlcude:

- `base::tolower()`
- `tm::removePunctuation()`
- `tm::removeNumbers()`



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.

Figure 3: Image

- tm::stripWhitespace()

```
text <- "<b>She</b> woke up at       6 A.M. It\'s so early!  She was only 10% awake and began drinking 

# Make lowercase
tolower(text)
```

```
## [1] "<b>she</b> woke up at       6 a.m. it's so early!  she was only 10% awake and began drinking co
```

```
# Remove punctuation
tm::removePunctuation(text)
```

```
## [1] "bSheb woke up at       6 AM Its so early  She was only 10 awake and began drinking coffee in fr
```

```
# Remove numbers
tm::removeNumbers
```

```
## function (x, ...)
## UseMethod("removeNumbers")
## <bytecode: 0x55dc23adcf00>
## <environment: namespace:tm>
```

```
# Remove whitespace
tm::stripWhitespace(text)
```

```
## [1] "<b>She</b> woke up at 6 A.M. It's so early! She was only 10% awake and began drinking coffee in
```

```
# Convert it from HTML to LaTeX
if (grep(pattern = '<b>', x = text)) {
    textTeX <- text
    textTeX <- gsub(pattern = '<br>' , replacement = "\\\\"  , textTeX)
    textTeX <- gsub(pattern = '<b>'  , replacement = "\\\textbf{", textTeX)
    textTeX <-  gsub(pattern = '</b>', replacement = "}"     , textTeX)
    # Huh % is hard
    textTeX <-  gsub(pattern = '%', replacement = "\\\\%"     , textTeX)
    print(textTeX)
}
```

```
## [1] "\textbf{She} woke up at       6 A.M. It's so early!  She was only 10\\% awake and began drinkin
```

Cleaning with qdap

the qdap package offers other text cleaning functions:

- bracketX()
  - will remove text within brackets
- replace_number()
  - turn numbers into words
- replace_abbreviation()
  - Create Full text from abbreviations.
- replace_contraction()
- replace_symbols()
  - turn words into symbols

```
text <- "Text mining usually involves the process of structuring the input text. The overarching goal i

## text is still loaded in your workspace

# Remove text within brackets
qdap::bracketX(text.var = text)
```

```
## [1] "Text mining usually involves the process of structuring the input text. The overarching goal is
```

```
# Replace numbers with words
qdap::replace_number(text)
```

```
## [1] "Text mining usually involves the process of structuring the input text. The overarching goal is
```

```
# Replace abbreviations
qdap::replace_abbreviation(text)
```

```
## [1] "Text mining usually involves the process of structuring the input text. The overarching goal is
```

```
# Replace contractions
qdap::replace_contraction(text)
```

```
## [1] "Text mining usually involves the process of structuring the input text. The overarching goal is
```

```
# Replace symbols with words
qdap::replace_symbol(text)
```

```
## [1] "Text mining usually involves the process of structuring the input text. The overarching goal is
```

All about Stop Words

Some words are frequent but more or less meaningless with respect to the *bag of words* method, these words include things like I, the, will et cetera.

Another example is a word for which meaning has already been accounted, in this case all the tweets are to do with `coffee` , so it will occur at a high rate of frequency and will demean any other possible insights that might be drawn from this analysis.

new words can be removed by using the vector `all_stops`:

```
text <- "Text mining usually involves the process of structuring the input text. The overarching goal i

# List standard English stop words
tm::stopwords("en")
```

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

```
# Print text without standard stop words
tm::removeWords(text, stopwords("en"))
```

```
## [1] "Text mining usually involves  process  structuring  input text. The overarching goal , essential
```

```
# Add "coffee" and "bean" to the list: new_stops
new_stops <- c("coffee", "bean", stopwords("en"))

# Remove stop words from text
tm::removeWords(text, new_stops)
```

```
## [1] "Text mining usually involves  process  structuring  input text. The overarching goal , essential
```

Intro to word stemming and stem completion

the `tm` package provides the `stemDocument()` function to get to a word's root. This function either:

- `character` vector –> `character` vector`character` vector
- `PlainTextDocument` –> `PlainTextDocument`

Then `stemCompletion()` can be used to reconstruct the words back into a known term. `stemCompletion()` accepts a character vector and a completion dictionary.

```
library(SnowballC)
# Create complicate
complicate <- c("complicated", "complication", "complicatedly")

# Perform word stemming: stem_doc
stem_doc <- tm::stemDocument(complicate)

# Create the completion dictionary: comp_dict
comp_dict <- "complicate"

# Perform stem completion: complete_text
complete_text <- tm::stemCompletion(stem_doc, comp_dict)


# Print complete_text
complete_text
```

```
##      complic      complic      complic
## "complicate" "complicate" "complicate"
```

Word stemming and stem completion on a sentence

Sentences are considered by ***R*** as a single character vector, before word stemming can be used it is necessary to split the sentence up into multiple words:

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

# Remove punctuation: rm_punc
rm_punc <- tm::removePunctuation(text_data)

# Create character vector: n_char_vec
n_char_vec <- unlist(strsplit(rm_punc, split = " "))

# Perform word stemming: stem_doc
stem_doc <- tm::stemDocument(n_char_vec)

# Create the completion Dictionary
comp_dict  <- c("In", "a", "complicate", "haste", "Tom", "rush", "to", "fix", "new", "too")

# Print stem_doc
stem_doc
```

```
##  [1] "In"      "a"       "complic" "hast"    "Tom"     "rush"    "to"      "fix"
##  [9] "a"       "new"     "complic" "too"     "complic"
```

```
# Re-complete stemmed document: complete_doc
complete_doc <- tm::stemCompletion(stem_doc, comp_dict)

# Print complete_doc
complete_doc
```

```
##           In            a      complic         hast          Tom         rush 
##         "In"          "a" "complicate"      "haste"        "Tom"       "rush" 
##           to          fix            a          new      complic          too 
##         "to"        "fix"          "a"        "new" "complicate"        "too" 
##      complic 
## "complicate"
```

Apply preprocessing steps to a corpus

the `tm` package provides the `tm_map()` function to apply cleaning functions to an entire corpus, making the cleaning steps easier. It takes two argumens:

1. A `corpus` object
2. A cleaning function (like `removeNumbers()`
   - If cleaning functions come from base ***R*** or `qdap` rather than `tm` it is necessary to wrap them in `content_transformer()`.

It is often more appropriate to write this is a function and then call the function.

```
clean_corpus <- function (corpus) {
    # Remove Punctuation
    corpus <- tm_map(corpus, removePunctuation)

    # Transform to Lower Case
    corpus <- tm_map(corpus, content_transformer(tolower))

    # Add more Stopwords
    corpus <- tm_map(corpus, removeWords, words = c(stopwords("en"), "coffee", "mug"))

    # Strip Whitespace
    corpus <- tm_map(corpus, stripWhitespace)

    return(corpus)
}

clean_corp <- clean_corpus(df_corpus)
print(clean_corp[[2]])
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 31
```

The TDM and DTM

With the text in a clean form it is necessary to give it structure, as far as *bag of text* text mining is concenerned, either the:

- Term Document Matrix (TDM), OR
- Document Term Matrix (DTM)

  - These are useful when you want to preserve time series for chronological data.

is used.

Think **Row-Col**-Matrix, where a term is extracted word and the column is the document (i.e. the tweet).

A TDM looks like this:

Tweet 1	**Tweet 2**	**Tweet 3**	**Tweet 4**
Term 1	0	0	0
Term 2	1	1	0
Term 3	1	1	0
Term 4	0	0	3
Term 5	0	0	1

A DTM looks like this:

Term 1	**Term 1**	**Term 2**	**Term 3**	**Term 4**	**Term 5**
Tweet 1	2	1	1	0	0
Tweet 2	0	1	1	0	0
Tweet 3	0	1	0	0	0
Tweet 4	0	0	0	3	1

They can be created by using either the `TermDocumentMatrix` or `DocumentTermMatrix` functions.

The `qdap` package relies on the word frequency matrix, but this is less popular and so wont be used. it's basically just a matrix with only one column.

Application to Coffee Dataset So now we'll apply this to the coffee dataset by first importing the data and arranging the columns

```r
# Load the Data Set
coffee <- read.csv(file = "../../0DataSets/TextMining/coffee.csv",
          stringsAsFactors = FALSE)

# Investigate the Data
str(coffee)
```

```
## 'data.frame':    1000 obs. of  15 variables:
##  $ num          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ text         : chr  "@ayyytylerb that is so true drink lots of coffee" "RT @bryzy_brib: Senior Mar
##  $ favorited    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ replyToSN    : chr  "ayyytylerb" NA NA NA ...
##  $ created      : chr  "8/9/2013 2:43" "8/9/2013 2:43" "8/9/2013 2:43" "8/9/2013 2:43" ...
##  $ truncated    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ replyToSID   : num  3.66e+17 NA NA NA NA ...
##  $ id           : num  3.66e+17 3.66e+17 3.66e+17 3.66e+17 3.66e+17 ...
##  $ replyToUID   : int  1637123977 NA NA NA NA NA NA 1316942208 NA NA ...
##  $ statusSource: chr  "<a href=\"http://twitter.com/download/iphone\" rel=\"nofollow\">Twitter for i
##  $ screenName   : chr  "thejennagibson" "carolynicosia" "janeCkay" "AlexandriaOOTD" ...
##  $ retweetCount: int  0 1 0 0 2 0 0 0 1 2 ...
##  $ retweeted    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ longitude    : logi  NA NA NA NA NA NA ...
##  $ latitude     : logi  NA NA NA NA NA NA ...
```

```r
names(coffee)
```

```
##  [1] "num"           "text"          "favorited"     "replyToSN"     "created"      
##  [6] "truncated"     "replyToSID"    "id"            "replyToUID"    "statusSource" 
## [11] "screenName"    "retweetCount" "retweeted"     "longitude"     "latitude"
```

```r
# Move the text column
text_num <- grep("text", names(coffee))
coffee <- coffee[,c(text_num, (1:ncol(coffee)[-text_num]))]
names(coffee)
```

```
##  [1] "text"          "num"           "text.1"        "favorited"     "replyToSN"    
##  [6] "created"       "truncated"     "replyToSID"    "id"            "replyToUID"   
## [11] "statusSource" "screenName"    "retweetCount" "retweeted"     "longitude"    
## [16] "latitude"
```

```r
# Move the id column
column_id_num <- grep("id", names(coffee))
coffee <- coffee[,c(column_id_num, (1:ncol(coffee))[-column_id_num])]
names(coffee) <- c("doc_id", names(coffee)[-1])
names(coffee)
```

```
##  [1] "doc_id"        "text"          "num"           "text.1"        "favorited"    
##  [6] "replyToSN"     "created"       "truncated"     "replyToSID"    "replyToUID"   
## [11] "statusSource" "screenName"    "retweetCount" "retweeted"     "longitude"    
## [16] "latitude"
```

Now the data frame can be made a source and then a VCorpus:

```r
coffee_source <- tm::DataframeSource(coffee)
coffee_corpus <- tm::VCorpus(coffee_source)
print(coffee_corpus)
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 14
## Content:  documents: 1000
```

Now we will reclean it by using the previous function (remember * jumps to a previous documnent):

```r
clean_corp <- clean_corpus(coffee_corpus)
```

Now we can create the document term matrix.

```r
# Create the document-term matrix from the corpus
coffee_dtm <- tm::DocumentTermMatrix(clean_corp)
coffee_tdm <- tm::TermDocumentMatrix(clean_corp)

# Print out coffee_dtm data
coffee_dtm
```

```
## <<DocumentTermMatrix (documents: 1000, terms: 3075)>>
## Non-/sparse entries: 7384/3067616
## Sparsity           : 100%
## Maximal term length: 27
## Weighting          : term frequency (tf)
```

```r
# Convert coffee_dtm to a matrix
coffee_m <- as.matrix(coffee_dtm)
```

```r
# Print the dimensions of coffee_m
dim(coffee_m)
```

```
## [1] 1000 3075
```

```r
# Review a portion of the matrix to get some Starbucks
 # so documents 25 to 35, columns matching star or bucks.
coffee_m[25:35, c("star", "starbucks")]
```

```
##              Terms
## Docs           star starbucks
##   3.65664e+17    0         0
##   3.65664e+17    0         1
##   3.65664e+17    0         1
##   3.65664e+17    0         0
##   3.65664e+17    0         0
##   3.65664e+17    0         0
##   3.65664e+17    0         0
##   3.65664e+17    0         0
##   3.65664e+17    0         0
##   3.65664e+17    0         1
##   3.65664e+17    0         0
```

(2) Word Clouds and More Interesting Visuals

Common Text Mining Visuals

(3) Adding to Text-Mining Skills

(4) Battle of the Tech Giants