Automated Text Analysis in Political Science

Martijn Schoonvelde

17 April 2018

Today's class

- Clean text in R: string operations and regular expressions
- Pre-process data: going from text to numbers
- Create a bag of words in **quanteda**

Principles of Automated Text Analysis (Grimmer & Stewart 2013)

1. All quantitative models of language are wrong – but some are useful
   - "the data generation process for any text is a mystery" (p. 3)
   - "models should be evaluated based on their ability to perform some useful social scientific task"
2. Quantitative methods for text amplify resources and augment humans
3. There is nog globally best methods for automated text analysis
   - Some methods are good, others not, but this not *because* they are quantitative
4. Validate, validate, validate
   - Validation may take several forms, depending on your approach (supervised or unsupervised)

Assumptions in Automated Text Analysis

- Text is an empirical implication of a (latent) characteristic of interest
- Texts can be represented through extracting relevant "features", which are analyzed using quantitative methods
- Oftentimes (and in most applications in this class): "bag of words"

Bag of words

- Also known as **document-term-matrix** (dtm) or in quanteda **document-feature-matrix** (dfm)
  - When transposed: **term-document matrix** (tdm)
- Matrix with each row a document and each column a word / feature
- Each cell denotes the number of times a particular n-gram appears in a particular document
- Order in which words occur is discarded
- Catch all term – specific structure may vary
  - 1-gram, 2-gram, 3-gram
  - Word weights (tf-idf)
  - Yes / No

Implications of bag of words

- Pros
  - Reduce complexity while retaining lots of information
  - “This sentence is about a computer” versus c(“sentence”, “about”, “computer”)
- Cons
  - Negations are discarded (“not good”) appears as c(“not”, “good”)
    - But taken into account when using bigrams
  - More generally: context is lost

Lost context

Quora Home Answer Notifications Search Quora

Clem Wang, Practicing Feature Engineer for over 15 years.
Answered Dec 18, 2016

Bag of words ignores the **context of words.** This can simplify the problem at some cost.

It can fail badly depending on specific case.

My specific example:

Toy Dog != **Dog Toy**

The first phrase is a type of breed (small), the second phrase is an object of play for a canine of any size.

Bag of Words treatment couldn't distinguish these two cases

2.7k Views · View Upvoters

Upvote · 2 Downvote

From text to numbers (Welbers et al 2017)

1. Importing text
2. String operations to clean text
3. Pre-processing: stemming, lemmatization, number removal, stop word removal
   - This also referred to as feature selection
4. Filtering and weighting of features
   - TF-IDF

Welbers et al. (2017)

Table 1. An overview of text analysis operations, with the R packages used in this Teacher's Corner.

	R packages	
Operation	example	alternatives
Data preparation		
importing text	*readtext*	*jsonlite, XML, antiword, readxl, pdftools*
string operations	*stringi*	*stringr*
preprocessing	*quanteda*	*stringi, tokenizers, snowballC, tm, etc.*
document-term matrix (DTM)	*quanteda*	*tm, tidytext, Matrix*
filtering and weighting	*quanteda*	*tm, tidytext, Matrix*
Analysis		
dictionary	*quanteda*	*tm, tidytext, koRpus, corpustools*
supervised machine learning	*quanteda*	*RTextTools, kerasR, austin*
unsupervised machine learning	*topicmodels*	*quanteda, stm, austin, text2vec*
text statistics	*quanteda*	*koRpus, corpustools, textreuse*
Advanced topics		
advanced NLP	*spacyr*	*coreNLP, cleanNLP, koRpus*
word positions and syntax	*corpustools*	*quanteda, tidytext, koRpus*

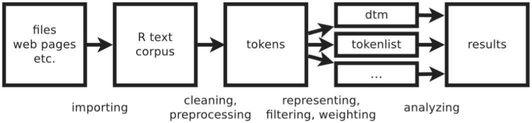

Figure 1. Order of text analysis operations for data preparation and analysis.

Importing text

- Exact steps depend on how texts is stored
  - When stored as txt or csv files you can work with **foreign** library in R: **read.csv(), read.txt()** etc.
  - Often text is stored in other formats (JSON, XML, HTML) – needs to be transformed into R objects
  - In R

    `readtext`
- Remember: often there exist multiple libraries and functions for doing the same thing
  - R open source – different groups of developers tackling similar issues
  - Use that to your advantage: google / stackexchange / other resources

Clean text, string operations

- This can be done in base R
  - That is, no packages needed
- But also specific libraries
  - `stringr, stringi`
- Text analysis packages can do a lot, but often necessary to dive into regular expressions in R
  - Can be quite painful
  - See Rstudio regex cheat sheet: `https://bit.ly/2GeeWV2`

Base R example

```
> corpus <- c(" This Is text_one <font size = '6'> ",
+             "And here is teXt Number 2!?",
+             "And %%$ number 3")

> #remove html tag
> corpus <- gsub("<.*?>", "", corpus)
> print(corpus)

[1] " This Is text_one  "
[2] "And here is teXt Number 2!?"
[3] "And %%$ number 3"
```

Stringr example

```
> #transform to lower case
> corpus <- stringr::tolower(corpus)
> print(corpus)

[1] " this is text_one  "
[2] "and here is text number 2!?"
[3] "and %%$ number 3"
```

Stringr example

```
> #remove anything but letters / alphabetic characters
> corpus <- stringr:str_replace_all(corpus, "[^[:alpha:]]", " ")
> print(corpus)

[1] " this is text one  "
[2] "and here is text number    "
[3] "and     number  "
```

Stringr example

```
> #strip surrounding white space
> corpus <- stringr::str_trim(corpus, side = "both")
> print(corpus)

[1] "this is text one"
[2] "and here is text number"
[3] "and     number"
```

Pre-processing data

- Tokenization
- Lowercasing
- Stemming
- Lemmatization
- Stopword removal

Tokenization: unigrams

```
> library(quanteda)
> sentence <- "One small step for man,
one giant leap for mankind."
> unigrams <- tokens(sentence)
> print(unigrams)
tokens from 1 document.
text1 :
 [1] "One"     "small"   "step"    "for"     "man"     ","
 [8] "giant"   "leap"    "for"     "mankind" "."
```

NB everything that is not a white space is tokenized

Tokenization: bigrams

```
> bigrams <- tokens(sentence, ngrams = 2)
> print(bigrams)
tokens from 1 document.
text1 :
 [1] "One_small"   "small_step"  "step_for"    "for_man"     "ma
 [6] ",_one"       "one_giant"   "giant_leap"  "leap_for"    "fo
[11] "mankind_."
```

Stemming

```
> sentence <- "The fish went fishing with the fishes"
> tokens <- tokens(sentence)
> stems <- tokens_wordstem(tokens)
> print(stems)
tokens from 1 document.
text1 :
[1] "The"  "fish" "went" "fish" "with" "the"  "fish"
```

- Stemming: algorithmic conversion of inflected forms of words into their root forms
- Fast but not perfect:
  - Unrelated words may be grouped together; related words may not be grouped together
  - Stems may not be words themselves – problematic if further analysis is based on dictionaries
- If you work in a different language this will require a different stemming algorithm
  - Success depends on whether language is inflected

Lemmatization

- Use of a dictionary to replace words with their root form
- More sophisticated than stemming
- Results are always words; neither does it group together unrelated words, nor does it miss to group together related words
- Less support in R but see for example:

  `koRpus`
- See also: `http://www.bernhardlearns.com/2017/04/cleaning-words-with-r-stemming.html`

Stopwords

```
> sentence <- "The fish went fishing with the fishes"
> tokens <- tokens(sentence)
> stopwords <- stopwords("english")
> head(stopwords)
[1] "i"        "me"       "my"       "myself" "we"       "our"
> stems <- tokens_remove(tokens, stopwords)
> print(stems)

tokens from 1 document.
text1 :
[1] "fish"    "went"    "fishing" "fishes"
```

dfm in quanteda

```
> text <- c("This is a sentence",
+           "This sentence is about a cat",
+           "This sentence is about a dog",
+           "This sentence is about a dog and a cat")
> dfm_text <- quanteda::dfm(text, tolower = TRUE,
+                           stem = TRUE,
+                           remove = stopwords("english"))
> print(dfm_text)
Document-feature matrix of: 4 documents, 3 features (33.3% spars
4 x 3 sparse Matrix of class "dfm"
       features
docs    sentenc cat dog
  text1       1   0   0
  text2       1   1   0
  text3       1   0   1
  text4       1   1   1
```

Filtering and weighting

- Not all terms are equally informative – feature selection
  - Very common words and very rare words
- A simple but effective method is to filter on document frequencies (the number of documents in which a term occurs), using a threshold for minimum and maximum number (or proportion) of documents
- Other possibility, assign weights, using, for example, tf-idf

tf-idf

- **term frequency-inverse document frequency**
- Number of times a word appears in a document offset by the frequency of the word in the corpus
- $\frac{\text{term frequency i}}{\text{total terms document i}} \times log(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}})$
- Adjust for the fact that some words appear more frequently in general

tf-idf in quanteda

```
> dfm_text <- dfm_tfidf(dfm_text, scheme_tf = "prop")
> print(dfm_text)
Document-feature matrix of: 4 documents, 3 features (33.3% spars
4 x 3 sparse Matrix of class "dfm"
       features
docs     sentenc    cat    dog
  text1  0          0      0
  text2  0          0.15   0
  text3  0          0      0.15
  text4  0          0.10   0.10
```

Calculate tf-idf

- tf("cat") equals:
  - text1 $= 0$
  - text2 $= 0.5$
  - text3 $= 0$
  - text4 $= 0.33$
- idf("cat") equals $log_{10}(\frac{4}{2}) = 0.301$
- tf-idf("cat") equals:
  - text1 $= 0$
  - text2 $= 0.15$
  - text3 $= 0$
  - text4 $= 0.10$

Example

Let's get started in R