Modern Text Analysis with Machine Learning: Week 2

Professor Anastasopoulos (ljanastas@uga.edu)

08/21/19

For today

1. Example of discussion leader presentation.
2. Building and extracting corpora with APIs.
3. Pre-processing text data: tokenization, stemming, removing stop words.
4. Document-feature matrix and TF-IDF.

Intro to natural language processing

- NLP is a term that is understood as a set of methods which map natural language units (words, sentences, paragraphs, etc) into a machine readible form.
- Once we can figure out how to represent language to a machine, we can then use statistical/mathematical tools to learn things about texts (without having to read them!).

NLP Applications (Real World)

- SPAM Filters.
- Speech recognition (Siri).
- Machine translation.
- Information retrieval (search engines).
- Artificial intelligence.

NLP Applications (Social Science)

- *Political sentiment and media bias* - Soroka, Stuart and Lori Young. 2012. "Affective News: The Automated Coding of Sentiment in Political Texts" Political Communication 29: 205-231.
- *Identification of politically relevant features of texts* - Monroe, Burt, Michael Colaresi, and Kevin Quinn. 2008. "Fightin' Words: Lexical Feature Selection and Evaluation for Identifying the Content of Political Conflict".Political Analysis 16(4)
- *Measuring expressed agendas in political texts* - Grimmer, J., 2010. A Bayesian hierarchical topic model for political texts: Measuring expressed agendas in Senate press releases. Political Analysis, pp.1-35.
- *And much much more...*

NLP Basic Terminology

$$\textit{word} \subset \textit{document} \subset \textit{corpus}$$

- **document** - A collection of words, usually the unit of observation.
- **corpus** - A collection of documents or a single document.
- The **corpus** can be thought of as our entire dataset.
- The **document** can be thought of as an observation

Example: Measuring political ideology from Tweets

Barberá, P., 2013. “Birds of the same feather tweet together, bayesian ideal point estimation using twitter data” *Political Analysis*

- Measuring the political ideology of Twitter users from tweets and patterns of following/followers.
- **document** - Each tweet is a document.
- **corpus** - All of the tweets that are analyzed are the corpus.

Acquiring text data

1. *Online corpora* - There are many built in packages in **R** and in **Python** that you can load which have thousands of texts that you can access.
2. *Build your own corpora*
    (a) Using an application program interface (API)
    (b) Webscraping

Online corpora

```
## Warning: package 'pacman' was built under R version 3.5.

## Warning: unable to access index for repository http://da
##   cannot open URL 'http://datacube.wu.ac.at/bin/macosx/e

## installing the source package 'tm.corpus.Reuters21578'
```

```r
library(tm)
install.packages("tm.corpus.Reuters21578",
                 repos = "http://datacube.wu.ac.at")
library(tm.corpus.Reuters21578)
data(Reuters21578)
```

- There are **thousands** of online corpora that are available.
- **R** is not really that good for accessing online corpora because they're not as easy to acquire.
- Here we are accessing the **Reuters** corpus which is a collection of 21,578 Reuters articles from the Reuters newswire in 1987.

Reuters Corpus

- Let's use the **tm** package to see what these articles look like.

```
inspect(Reuters21578[1:2])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed)
## Content:  documents: 2
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  16
## Content:  chars: 2860
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  16
## Content:  chars: 438
```

Reuters Corpus

- We can also read the articles to see how they're structured in **R**

```
Reuters21578[[1]]$content
```

```
## [1] "Showers continued throughout the week in\nthe Bahia
```

Building your own corpora using APIs

- In general, ready-to-go corpora are not that useful.
- They contain old documents and are unlikely to have the info you want.
- For the most part, when you're doing text analysis, you'll have to build your own corpora.

Building your own corpora using APIs

- The easiest way to do this is to extract data from one of the MILLIONS of APIs out there.
- APIs were originally designed to allow developers of apps to have constant streaming access to data.
- But they are a treasure trove of information of immesurable use to social scientists who know how to tap into them.

Politics APIs

- **GovTrack.us API** - Get any information about Congress (bills, legislators, voting) over several Congresses.
- **OpenStates API** - Tons of information about state legislators, bills, voting, etc.
- **Opensecrets.org** - Money in politics and campaign finance database.
- **Twitter API** - Get streaming tweets from Twitter with user information etc.

Extracting data from APIs using R

- Three packages are very useful for this purpose: **twitteR**, **httr**, and **jsonlite**.
- You first must establish *OAuth* credentials if you would like to access Tweets.
- You can find out how to do so here Getting OAuth Credentials

Extracting Tweets using TwitteR

```
library(twitteR)
setup_twitter_oauth(
  consumer_key="PeVki23tnu1dNzYoG1pJS9khO", # This is the
  consumer_secret="RksPKIMCK2s7UBpiFuIJzUzQwCkXKY6nVtChlDOx
  access_token="18249358-ctVZdV7IpFxOcorUUdNqjeahb9Tb23yVoc
  access_secret="46JNswwM5I3Q6JwgDsUtyYKCMhs7yDY9jsrxslH55u
```

```
## [1] "Using direct authentication"
```

Extracting Tweets using TwitteR

Tweets are saved as a list in *R*

```
UGATweets = searchTwitter("@universityofga")[1:2]
UGATweets[[1]]
```

```
## [1] "UGA_Innovation: .@uga_genetics @universityofga @UGA
```

Example 2: Tapping into APIs using “jsonlite”

Example 2: Tapping into APIs using "jsonlite"

```
# We can retrieve the title and other information about the
# I'm creating a data frame with the bill title, bill id, b
# id and the bill sponsors gender

billtitles<-bills$objects$title
billid<-bills$objects$id
sponsorid<-bills$objects$sponsor$id
sponsiridgender<-bills$objects$sponsor$gender

refugeebilldat<-
  data.frame(billtitles,sponsiridgender)

#head(refugeebilldat)
```

Pre-processing text data

- Now that we have the data, we need to prepare it for analysis.
- This involves a couple of steps which take **strings** and breaks them down into analyzeable units.

Pre-processing text data steps

1. **Tokenization** - splits the document into tokens which can be words or n-grams (phrases).
2. **Formatting** - punctuation, numbers, case, spacing.
3. **Stop word removal** - removal of “stop words”
4. **Stemming** - removal of certain types of suffixes.

Tokenization

- "**Bag of words**" model - most text analysis methods treat documents as a big bunch of words or terms.
- Order is generally not taken into account, just word and term frequencies.
- There are ways to parse documents into *ngrams* or *words* but we'll stick with words for now.

Tokenization example: Tweets mentioning "@realDonaldTrump"

```r
library(plyr)
library(quanteda) # This is a great text cleaning package
```

```
## Warning: package 'quanteda' was built under R version 3.

## Package version: 1.4.0

## Parallel computing: 2 of 4 threads used.

## See https://quanteda.io for tutorials and examples.

##
## Attaching package: 'quanteda'

## The following objects are masked from 'package:tm':
##
##     as.DocumentTermMatrix, stopwords

## The following object is masked from 'package:utils':
##
```

Tokenization example: Tweets mentioning “@realDonaldTrump”

```
library(quanteda)
potustweets.vector.tokens  = tokens(potustweets.vector, # T
                                    remove_numbers = TRUE,
                                    remove_punct = TRUE,
                                    remove_symbols = TRUE,
                                    ngrams = 1:3,# Return n-gr
                                    remove_twitter= TRUE, # Re
                                    remove_url = TRUE) # Remov
```

Tokenization example: Tweets mentioning "@realDonaldTrump"

```
potustweets.vector.tokens[1:2]
```

```
## tokens from 2 documents.
## text1 :
## [1] "RT"                              "JimKilbane"
## [3] "realDonaldTrump"                 "RT_JimKilbane"
## [5] "JimKilbane_realDonaldTrump"      "RT_JimKilbane_realI
##
## text2 :
##  [1] "RudyGiuliani"
##  [2] "realDonaldTrump"
##  [3] "They"
##  [4] "swore"
##  [5] "an"
##  [6] "oath"
##  [7] "to"
##  [8] "the"
```

Stop words

- Stop words are simply words that removed during text processing.
- They tend to be words that are very common “the”, “and”, “is” etc.
- These common words can cause problems for machine learning algorithms and search engines because they add noise.
- **BEWARE** Each package defines different lists of stop words and sometimes removal can decrease performance of supervised mechine learning classifiers.

Stemming

- In linguistics, stemming is the process of reducing words to their stems.
- "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu"
- This is especially useful for unsupervised machine learning algorithms but may introuce issues in supervised machine learning.
- For example "cats" and "catty" would both be reduced to the term "cat".

Building the document-feature matrix

```
potustweets.dfm <- dfm(potustweets.vector.tokens,  # Constr
                            remove = stopwords("english"),
                            stem = TRUE, remove_punct = TRU

potustweets.dfm
```

```
## Document-feature matrix of: 10 documents, 308 features (
```

Building the document-term matrix

```
potustweets.dfm[1:5, 1:5]
```

```
## Document-feature matrix of: 5 documents, 5 features (56.
## 5 x 5 sparse Matrix of class "dfm"
##          features
## docs     rt jimkilban realdonaldtrump rt_jimkilban
##    text1  1         1               1            1
##    text2  0         0               1            0
##    text3  1         0               1            0
##    text4  1         0               1            0
##    text5  0         0               1            0
##          features
## docs      jimkilban_realdonaldtrump
##    text1                          1
##    text2                          0
##    text3                          0
##    text4                          0
##    text5                          0
```

Additional document-feature matrix options.

- There are a few other options that are useful for pre-processing a DFM in quanteda.
- *Dictionaries*: you can specify your own dictionary.
- *Remove*: you can remove specific words or patterns.
- *Automatic Sparsity Reduction* - An automated method for reducing the sparsity of a DFM. May help performance of ML algorithms.
- *Minimum/Max Number of Characters*: useful for topic modeling.
- *TF/IDF Weighting*: Multipurpose, useful primarily for *supervised learning.*

Sparsity reduction

- Practically all DFMs are SPARSE (they are mostly 0s).
- Reducing the sparsity of a matrix can improve performance of machine learning algorithms.

Automatic sparsity reduction code

```
potustweets.dfm.reduced = dfm_trim(potustweets.dfm, sparsit
potustweets.dfm.reduced
```

```
## Document-feature matrix of: 10 documents, 308 features (
```

TF/IDF Weighting

Term frequency:

$$tf_d = t_{d,i}$$

Inverse document frequency measures how much “information” a word contains.

$$idf_d = log \frac{N}{n_i}$$