Introduction to Text Mining

Jen Helsby and Eric Potash

Computation for Public Policy

Lecture 12: February 10, 2016

computationforpolicy.github.io

Kinds of Structure

- Unstructured
  - Free text
- Weakly Structured
  - Articles with headlines, dates, and author names
- Semi-structured
  - E-mails with subject line, from/to/cc/bcc names and addresses, timestamps
- Structured
  - Data tables

Text

- “Most of the world’s data is unstructured”
- A lot of important information is contained in free text:
  - Doctor’s notes
  - Journals
  - Legislation
  - News stories
- Would like to try to include it in our quantitative analyses.

Text Mining

- Extracting structured data from unstructured data.

Terminology

A *corpus* is a collection of *documents*, each of which is a collection of *terms*

Term Frequency (TF)

The *term frequency* is defined as the number of times a given term appears in a given document:

$$TF(d, t) = \#\{\text{occurrences of term } t \text{ in document } d\}$$

Term Frequency (TF)

Term frequency implicitly assumes:

- The more frequent a word in a document, the more important it is.
- Moreover, that a term that is twice as frequent is twice as important.

Is that a good assumption?

Document Frequency (DF)

The document frequency of a term is the number of documents that a term appears in and denoted DF(t).

$$DF(t) = \#\{\text{documents containing term } t\}$$

Document Frequency (DF)

Document frequency assumes:

- Common terms are less descriptive than uncommon terms
    - i.e. the smaller *DF(t)* the more important *t* is.

Is that a good assumption?

Inverse Document Frequency (IDF)

We want a metric which is more important when it is *larger* not smaller. Could use

#{documents} / #{documents containing *t* }

But that “blows up” too quickly so instead we define *Inverse Document Frequency* as

IDF(t) = log (#{documents} / #{documents containing *t* })

TF-IDF

Multiplying the two metrics gives us a new metric

$$TF\text{-}IDF(d, t) = TF(d, t) * IDF(t)$$

TF-IDF is a good measure of “how important a word is to a document in a collection or corpus.”

Text Mining Problems

- Homographs
  - Terms with the same spelling but different meaning
- Synonyms
  - Distinct terms with the same or similar meaning
- Variations
  - Plural and possessive noun endings, verb conjugations, etc.
- Typos
  - Erroneous terms

Normalization

One straightforward way to reduce the number of terms is to *normalize* them:

- Make all characters upper-case
  - python’s upper()
- Could have unintended consequences with names
  - e.g. ‘Steve Jobs’, ‘jobs’
- Fix typos using a dictionary
  - Error-prone, especially with proper nouns.

Stop Words

- Some words are almost universally unimportant and can be excluded
  - e.g. the, an, a
- Particular documents might have particular stopwords
  - e.g. legal documents
- These are called stop words. Using stop words makes text mining easier by reducing the number of words to keep track of, i.e. the “dimensionality.

Aside: Regular Expressions

- Rule-based pattern-matching of strings
- Available in every programming language (python, SQL, java, C, javascript, etc.)
- Easy and flexible but also can be messy.

Stemming

- Consider these words:
  - ‘drives’, ‘drive’, ‘driven’, ‘driving’, ‘drove’, etc.
- If we naively bag words then each one of these variations is counted separately.
- *Stemming* transforms each variation to its root.
  - e.g. ‘driv’

bigrams

- A pair of consecutive terms is called a *bigram*.
- Sometimes the meaning of the bigram is different than the individual meanings
    - e.g. ‘vice president’
- We can replace terms (unigrams) with bigrams and repeat all of the above analysis.
    - e.g. we can count the number of occurrences of each bigram in each document.
- More generally, an *n-gram* is a sequence of *n* terms.
    - e.g. ‘central intelligence agency’

Part of Speech (POS) tagging

- “the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context”

```
>>> text = """The board's action shows what free enterprise
...     is up against in our complex maze of regulatory laws ."""
>>> tokens = text.split()
>>> tagger.tag(tokens)
[('The', 'AT'), ("board's", 'NN$'), ('action', 'NN'), ('shows', 'NNS'),
('what', 'WDT'), ('free', 'JJ'), ('enterprise', 'NN'), ('is', 'BEZ'),
('up', 'RP'), ('against', 'IN'), ('in', 'IN'), ('our', 'PP$'), ('complex', 'JJ'),
('maze', 'NN'), ('of', 'IN'), ('regulatory', 'NN'), ('laws', 'NNS'), ('.', '.')]
```