Info Retrieval Basics

Stephen Hansen
University of Oxford

Textual Databases

A single observation in a textual database is called a *document*.

The set of documents that make up the dataset is called a *corpus*.

We often have covariates associated with each document that are sometimes called *metadata*.

Example

In "Transparency and Deliberation" we use a corpus of verbatim FOMC transcripts from the era of Alan Greenspan:

- 149 meetings from August 1987 through January 2006.
- A document is a single statement by a speaker in a meeting (46,502).
- Associated metadata: speaker biographical information, macroeconomic conditions, etc.

Data Sources

There are many potential sources for text data, such as:

1. PDF files or other non-editable formats
2. Word documents or other editable formats
3. Web pages
4. Application Programming Interfaces (API) for web applications.

From Files to Databases

Turning raw text files into structured databases is often a challenge:

1. Separate metadata from text
2. Identify relevant portions of text (paragraphs, sections, etc)
3. Remove graphs and charts

First step for non-editable files is conversion to editable format, usually with optical character recognition software.

With raw text files, we can use regular expressions to identify relevant patterns.

HTML and XML pages provide structure through tagging.

If all else fails, relatively cheap and reliable services exist for manual extraction.

What is Text?

At an abstract level, text is simply a string of characters.

Some of these may be from the Latin alphabet—‘a’, ‘A’, ’p’ and so on—but there may also be:

1. Decorated Latin letters (e.g. ö)
2. Non-Latin alphabetic characters (e.g. Chinese and Arabic)
3. Punctuation (e.g. ‘!’)
4. White spaces, tabs, newlines
5. Numbers
6. Non-alphanumeric characters (e.g. ‘@’)

Key Question: How can we obtain an informative, quantitative representation of these character strings? This is the goal of text mining.

First step is to *pre-process* strings to obtain a cleaner representation.

Pre-Processing I: Tokenization

Tokenization is the splitting of a raw character string into individual elements of interest.

Often these elements are words, but we may also want to keep numbers or punctuation as well.

Simple rules work well, but not perfectly. For example, splitting on white space and punctuation will separate hyphenated phrases as in ‘risk-averse agent’ and contractions as in ‘aren’t’.

In practice, you should (probably) use a specialized library for tokenization.

Pre-Processing II: Stopword Removal

The frequency distribution of words in natural languages is highly skewed, with a few dozen words accounting for the bulk of text.

These *stopwords* are typically stripped out of the tokenized representation of text as they take up memory but do not help distinguish one document from another.

Examples from English are ‘a’, ‘the’, ‘to’, ‘for’ and so on.

No definitive list, but example on `http://snowball.tartarus.org/algorithms/english/stop.txt`.

Pre-Processing II: Stopword Removal

The frequency distribution of words in natural languages is highly skewed, with a few dozen words accounting for the bulk of text.

These *stopwords* are typically stripped out of the tokenized representation of text as they take up memory but do not help distinguish one document from another.

Examples from English are ‘a’, ‘the’, ‘to’, ‘for’ and so on.

No definitive list, but example on `http://snowball.tartarus.org/algorithms/english/stop.txt`.

Also common to drop rare words, for example those that appear is less than some fixed percentage of documents.

Pre-Processing III: Linguistic Roots

For many applications, the relevant information in tokens is their linguistic root, not their grammatical form. We may want to treat ‘prefer’, ‘prefers’, ‘preferences’ as equivalent tokens.

Two options:

- *Stemming*: Deterministic algorithm for removing suffixes. Porter stemmer is popular.
  Stem need not be an English word: Porter stemmer maps ‘inflation’ to ‘inflat’.
- *Lemmatizing*: Tag each token with its part of speech, then look up each (word, POS) pair in a dictionary to find linguistic root.
  E.g. ‘saw’ tagged as verb would be converted to ‘see’, ‘saw’ tagged as noun left unchanged.

A related transformation is *case-folding* each alphabetic token into lowercase. Not without ambiguity, e.g. ‘US’ and ‘us’ each mapped into same token.

Pre-Processing IV: Multi-Word Phrases

Sometimes groups of individual tokens like “Bank Indonesia” or “text mining” have a specific meaning.

One ad-hoc strategy is to tabulate the frequency of all unique two-token (bigram) or three-token (trigram) phrases in the data, and convert the most common into a single token.

In FOMC data, most common bigrams include ‘interest rate’, ‘labor market’, ‘basi point’; most common trigrams include ‘feder fund rate’, ‘real interest rate’, ‘real gdp growth’, ‘unit labor cost’.

More Systematic Approach

Some phrases have meaning because they stand in for specific names, like "Bank Indonesia". One can used named-entity recognition software applied to raw, tokenized text data to identify these.

Other phrases have meaning because they denote a recurring concept, like "housing bubble". To find these, one can apply part-of-speech tagging, then tabulate the frequency of the following tag patterns:

$$\text{AN/NN/AAN/ANN/NAN/NNN/NPN.}$$

See chapter on collocations in Manning and Schütze's *Foundations of Statistical Natural Language Processing* for more details.

Example from NYT Corpus

$C(w^1\ w^2)$	w^1	w^2	tag pattern
11487	New	York	A N
7261	United	States	A N
5412	Los	Angeles	N N
3301	last	year	A N
3191	Saudi	Arabia	N N
2699	last	week	A N
2514	vice	president	A N
2378	Persian	Gulf	A N
2161	San	Francisco	N N
2106	President	Bush	N N
2001	Middle	East	A N
1942	Saddam	Hussein	N N
1867	Soviet	Union	A N
1850	White	House	A N
1633	United	Nations	A N
1337	York	City	N N
1328	oil	prices	N N

Pre-Processing of FOMC Corpus

	All terms	Alpha terms	No stopwords	Stems
# total terms	6249776	5519606	2505261	2505261
# unique terms	26030	24801	24611	13734

Notation

The corpus is composed of D documents indexed by d.

After pre-processing, each document is a finite, length-N_d list of terms $\mathbf{w}_d = (w_{d,1}, \ldots, w_{d,N_d})$ with generic element $w_{d,n}$.

Let $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_D)$ be a list of all terms in the corpus, and let $N \equiv \sum_d N_d$ be the total number of terms in the corpus.

Suppose there are V **unique** terms in $\mathbf{w}$, where $1 \leq V \leq N$, each indexed by v.

We can then map each term in the corpus into this index, so that $w_{d,n} \in \{1, \ldots, V\}$.

Let $x_{d,v} \equiv \sum_n \mathbb{1}(w_{d,n} = v)$ be the count of term v in document d.

Example

Consider three documents:

1. ‘stephen is nice’
2. ‘john is also nice’
3. ‘george is mean’

We can consider the set of unique terms as $\{\text{stephen}, \text{is}, \text{nice}, \text{john}, \text{also}, \text{george}, \text{mean}\}$ so that $V = 7$.

Construct the following index:

stephen	is	nice	john	also	george	mean
1	2	3	4	5	6	7

We then have $\mathbf{w}_1 = (1, 2, 3)$; $\mathbf{w}_2 = (4, 2, 5, 3)$; $\mathbf{w}_3 = (6, 2, 7)$.

Moreover $x_{1,1} = 1$, $x_{2,1} = 0$, $x_{3,1} = 0$, etc.

Document-Term Matrix

A popular quantitative representation of text is the *document-term matrix* $\mathbf{X}$, which collects the counts $x_{d,v}$ into a $D \times V$ matrix.

In the previous example, we have

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

The key characteristics of the document-term matrix are its:

1. High dimensionality
2. Sparsity

Ngram Models

In the above example, we made an explicit choice to count individual terms, which destroys all information on word order.

In some contexts, this may be sufficient for our information needs, but in others we might lose valuable information.

We could alternatively have counted all adjacent two-term phrases, called bigrams or, more generally, all adjacent N-term phrases, called Ngrams.

This is perfectly consistent with the model described above, where v now indexes unique bigrams rather than unique unigrams:

stephen.is	is.nice	john.is	is.also	also.nice	george.is	is.mean
1	2	3	4	5	6	7

We then have $\mathbf{w}_1 = (1, 2)$; $\mathbf{w}_2 = (3, 4, 5)$; $\mathbf{w}_3 = (6, 7)$.

Dimensionality Reduction through Keywords

The first approach to handling **X** is to limit attention to a subset of columns of interest.

In the natural language context, this is equivalent to representing text using the distribution of keywords across documents.

One can either look at the incidence of keywords (Boolean search), or else their frequency (dictionary methods).

The researcher must decide in advance which are the keywords of interest.

Application

The recent work of Baker, Bloom, and Davis on measuring economic policy uncertainty (`http://www.policyuncertainty.com/`) is largely based on a media index constructed via Boolean searches of US and European newspapers.

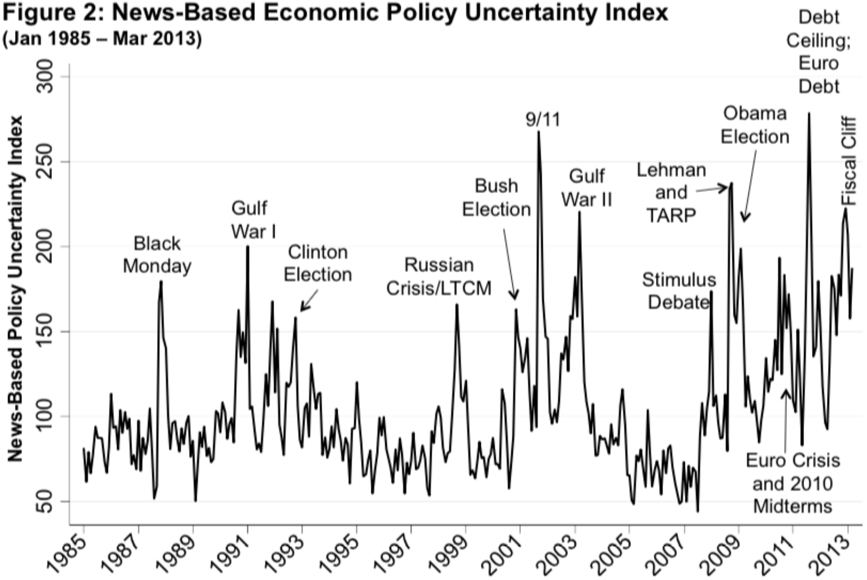
For each paper on each day since 1985, identify articles that contain:

1. "uncertain" OR "uncertainty", AND
2. "economic" OR "economy", AND
3. "congress" OR "deficit" OR "federal reserve" OR "legislation" OR "regulation" OR "white house"

Normalize resulting article counts by total newspaper articles that month.

Results

Figure 2: News-Based Economic Policy Uncertainty Index
(Jan 1985 – Mar 2013)



Why Text?

VIX is an asset-based measure of uncertainty: implied S&P 500 volatility at 30-day horizon using option prices.

So what does text add to this?

1. Focus on broader type of uncertainty besides equity prices.
2. Much richer historical time series.
3. Cross-country measures.

Term Weighting

Dictionary methods are based on raw counts of words.

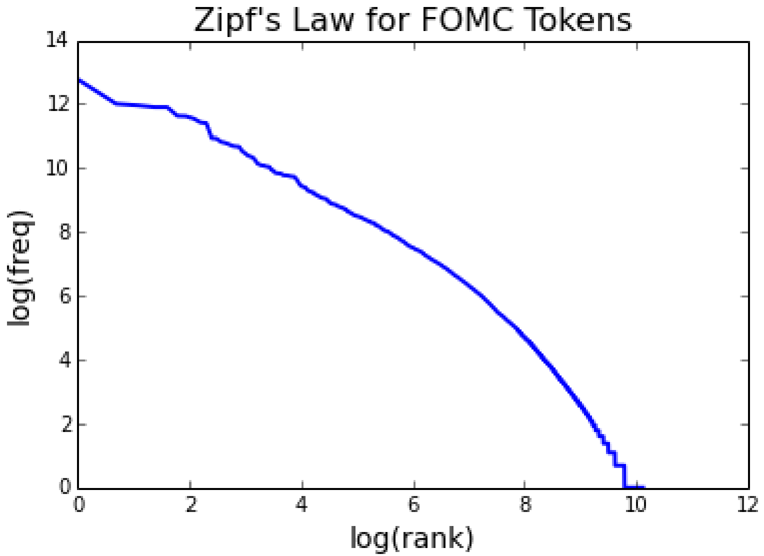
But the frequency of words in natural language can distort raw counts.

Zipf's Law is an empirical regularity for most natural languages that maintains that the frequency of a particular term is inversely proportional to its rank.

Means that a few terms will have very large counts, many terms have small counts.

Example of a *power law*.

Zipf's Law in FOMC Transcript Data



Rescaling Counts

Let $x_{d,v}$ be the count of the vth term in document d.

To dampen the power-law effect can express counts as

$$tf_{d,v} = \begin{cases} 0 & \text{if } x_{d,v} = 0 \\ 1 + \log(x_{d,v}) & \text{otherwise} \end{cases}$$

which is the *term frequency* of v in d.

Thought Experiment

Consider a two-term dictionary $\mathfrak{D} = \{v', v''\}$.

Suppose two documents d' and d'' are such that:

$$x_{d',v'} > x_{d'',v'} \text{ and } x_{d',v''} < x_{d'',v''}.$$

Now suppose that no other document uses term v' but every other document uses term v''.

Which document is "more about" the theme the dictionary captures?

Inverse Document Frequency

Let df_v be the number of documents that contain the term v.

The *inverse document frequency* is

$$\mathrm{idf}_v = \log\left(\frac{D}{df_v}\right),$$

where D is the number of documents.

Properties:

1. Higher weight for words in fewer documents.
2. Log dampens effect of weighting.

TF-IDF Weighting

Combining the two observations from above allows us to express the *term frequency - inverse document frequency* of term v in document d as

$$\text{tf-idf}_{d,v} = tf_{d,v} \times idf_v .$$

Gives prominence to words that occur many times in few documents.

Can now score each document as $s_d = \sum_{v \in \mathfrak{D}} \text{tf-idf}_{d,v}$ and then compare.

In practice, this can provide better results than simple counts.

Data-Driven Stopwords

Stopword lists are useful for generic language, but there are also context-specific frequently used words.

For example, in a corpus of court proceedings, words like ‘lawyer’, ‘law’, ‘justice’ will show up a lot.

Can also define term-frequency across entire corpus as

$$tf_v = 1 + \log\left(\sum_d x_{d,v}\right).$$

One can then rank each term in the corpus according to $tf_v \times idf_v$, and choose a threshold below which to drop terms.

This provides a means for data-driven stopword selection.

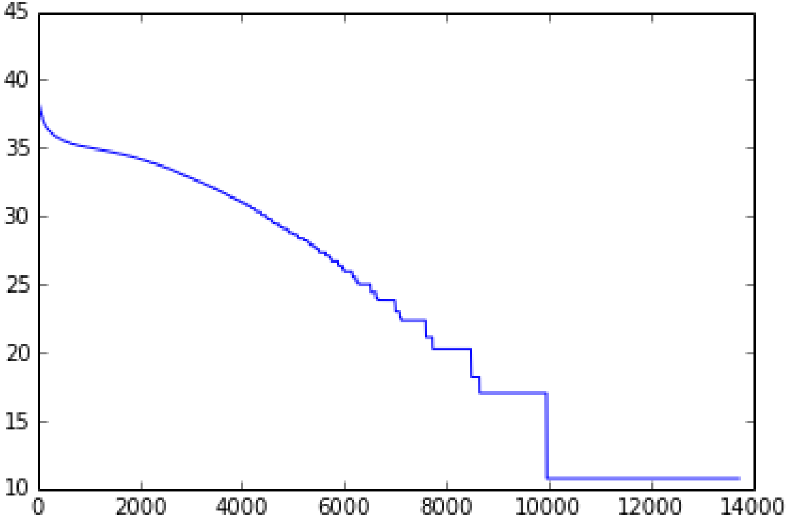
Stem Rankings in FOMC Transcript Data

R1 = collection frequency ranking

R2 = tf-idf-weighted ranking

Rank	1	2	3	4	5	6	7	8	9
R1	rate	think	year	will	market	growth	inflat	price	percent
R2	panel	katrina	graph	fedex	wal	mart	mbs	mfp	euro

Ranking of All FOMC Stems



45
40
35
30
25
20
15
10
0
2000
4000
6000
8000
10000
12000
14000