Text Mining for the Social Sciences
Lecture 2: Acquiring and Processing Text Data

Stephen Hansen

The Ideal Dataset

Ideally, we'd start with a database that looked like the following:

Speaker	Date	Location	Text
A	Jan 14	Madrid	...
B	Jun 14	Valencia	...
C	Jun 14	Barcelona	...
B	Aug 14	Valencia	...
...	...	...	...

The text is unstructured data, but it lives a structured, rectangular database with associated metadata.

...In the Real World

Organizing text and metadata into a database can be hugely tedious and time consuming.

Cliché but true: Data science is 80% cleaning, 20% analysis.

We will begin today by discussing issues in database construction.

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.

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.

Key problems:

1. Separate metadata from text
2. Break texts up at appropriate points (paragraphs, bullet points, etc)
3. Remove graphs and charts

In many cases, even within the same organization, authors write documents in highly specific ways, which makes full automation of data collection very difficult.

PDF Files

The first issue with PDF files is they aren't immediately editable (or readable).

The typical solution is to use Optical Character Recognition (OCR) software to batch convert PDF files into some readable format like RTF or simple text.

There can be substantial variation across software in terms of performance.

Not realistic to expect 100% accuracy, but nor is this typically necessary.

Editable Files

Editable files vary in their structure.

Simple text files are merely a stream of encoded bytes:

1. Regex is basically only way to re-structure
2. When there are charts, figures, etc., may be better to do manually. Cheap and reliable services exist for this.

More complex formats contain useful information besides encoded bytes. For example, docx files are XML underneath, can exploit tag structure. See *python-docx* module.

Webpages

The web provides a rich source for text data.

See example notebook of scraping static HTML webpage.

Also useful for bulk downloads of PDF documents.

APIs

Many web applications like Twitter and Facebook offer access to structured text data via their APIs.

Example of this in Topics in Big Data Analytics.

Lesson: structuring digital data can greatly facilitate text mining!

Text Processing

Suppose we have a clean database in front of us.

Before we perform any mathematical operations on text, we have to process the raw strings to form document-term matrix.

Basic steps:

1. Break statements into tokens.
2. Decide which tokens you want to keep.
3. Decide whether and how you want to transform tokens.

Tokenizing

Tokenizing a string is breaking it up into its linguistic elements: words, numbers, punctuation, etc.

The simplest way is to use regex and split on all non-alphanumeric characters.

```
pattern = re.compile('(\W)',re.UNICODE)
tok_string = re.split(pattern, string)
tok_string = filter(lambda x: x not in set([u'',u' ']), tok_string)
```

Example

```
In [86]: string = u" there are some models in which that is the case. it is not obvious how we rule them out. the
second thing is\u2014i do not know how you evaluate this\u2014you must be thinking whether this means that in every
moderate-sized recession henceforth we will view the federal reserve's best policy as extending\u2014 "

In [87]: tok_string = re.split(pattern, string)

In [88]: tok_string
Out[88]: [u'', u' ', u'there', u' ', u'are', u' ', u'some', u' ', u'models', u' ', u'in', u' ', u'which', u' ',
u'that', u' ', u'is', u' ', u'the', u' ', u'case', u'.', u'', u' ', u'it', u' ', u'is', u' ', u'not', u' ',
u'obvious', u' ', u'how', u' ', u'we', u' ', u'rule', u' ', u'them', u' ', u'out', u'.', u'', u' ', u'the', u' ',
u'second', u' ', u'thing', u' ', u'is', u'\u2014', u'i', u' ', u'do', u' ', u'not', u' ', u'know', u' ', u'how', u'
', u'you', u' ', u'evaluate', u' ', u'this', u'\u2014', u'you', u' ', u'must', u' ', u'be', u' ', u'thinking', u' ',
u'whether', u' ', u'this', u' ', u'means', u' ', u'that', u' ', u'in', u' ', u'every', u' ', u'moderate', u'-',
u'sized', u' ', u'recession', u' ', u'henceforth', u' ', u'we', u' ', u'will', u' ', u'view', u' ', u'the', u' ',
u'federal', u' ', u'reserve', u"'", u's', u' ', u'best', u' ', u'policy', u' ', u'as', u' ', u'extending',
u'\u2014', u'', u' ', u'']
```

Example

```
In [108]: string = u'Έκτακτη συνεδρίαση της ομάδας εργασίας του Eurogroup,  για να εξεταστεί το ελληνικό αίτημα για παράταση της
ευρωπαϊκής βοήθειας, θα γίνει αύριο, Πέμπτη, στις Βρυξέλλες. Το αίτημα της Αθήνας αναμένεται να αποσταλεί αύριο.'

In [109]: tok_string = re.split(pattern, string)

In [110]: tok_string = filter(lambda x: x not in set([u'',u' ']), tok_string)

In [111]: for p in tok_string: print p
Έκτακτη
συνεδρίαση
της
ομάδας
εργασίας
του
Eurogroup
,
για
να
εξεταστεί
το
ελληνικό
αίτημα
για
παράταση
της
ευρωπαϊκής
βοήθειας
,
θα
γίνει
αύριο
,
Πέμπτη
,
```

Importance of unicode flag

```
In [112]: pattern = re.compile('(\W)')

In [113]: string = u'Έκτακτη συνεδρίαση της ομάδας εργασίας του Eurogroup,  για να εξεταστεί το ελληνικό αίτημα για παράταση της ευρωπαϊκής
στις Βρυξέλλες. Το αίτημα της Αθήνας αναμένεται να αποσταλεί αύριο.'

In [114]: tok_string = re.split(pattern, string)

In [115]: tok_string = filter(lambda x: x not in set([u'',u' ']), tok_string)

In [116]: for p in tok_string: print p
Έ
κ
τ
α
κ
τ
η
σ
υ
ν
ε
δ
ρ
ί
α
σ
η
τ
η
ς
ο
μ
ά
δ
α
ς
```

Issues

Sometimes punctuation joins words together, or there are natural phrases.

We probably don't want to represent:

- 'aren't' as 'aren' "'" 't'
- 'risk-weighted assets' as 'risk' '-' 'weighted' 'assets'

Issues

Sometimes punctuation joins words together, or there are natural phrases.

We probably don't want to represent:

- 'aren't' as 'aren' "'" 't'
- 'risk-weighted assets' as 'risk' '-' 'weighted' 'assets'

Options:

1. Remove all punctuation that has an alphabetic character on either side before tokenizing
2. Make custom list of transformations
3. Ignore the problem

Case Folding

A common step after tokenizing is to convert all tokens to lowercase:

'I' 'think' ',' 'therefore' 'I' 'am' '.' $\longrightarrow$ 'i' 'think' ',' 'therefore' 'i' 'am' '.'

Main issue is that capitalization sometimes affects meaning, for example 'US' $\neq$ 'us' nor 'CAT' $\neq$ 'cat'.

But even Google can't fully overcome this problem! (Try a search for 'C.A.T.').

Stopword Removal

A large percentage of words in any text are articles and prepositions like 'a', 'the', 'to', etc.

Stopwords are common words that we remove from text as part of pre-processing. Lists available online, e.g. `http://snowball.tartarus.org/algorithms/english/stop.txt`.

Also common to drop words that appear in few and many documents.

Fancier option in next lecture based on weights.

Also common to remove punctuation, but these can also be informative depending on context.

Linguistic Roots

In text, the same word has many forms, like ‘prefer’, ‘prefers’, ‘preferences’, etc.

This variation is not useful in many information retrieval contexts.

Two ways of dealing with this: *lemmatization* and *stemming*.

Lemmatization

First-best.

Step 1: Part of speech tagging to associate each word to its part of speech.

Step 2: 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.

Stemming

In much applied work, a shortcut is taken.

Stemming is the process of following a deterministic algorithm for removing the ends of words.

Porter stemmer is popular.

Output need not be an English word. For example, stem of 'inflation' is 'inflat'.

Sometimes equivalence between tokens is misleading: 'university' and 'universe' stemmed to same form.

Multi-Word Expressions

The bag of words model destroys information on word phrases like "national football league".

Machine learning algorithms exists to identify such multi-word expressions.

An easier strategy is to tabulate the frequency of all bigrams (and trigrams) in the data, and convert the most common into single token.

In FOMC data:

1. Most common bigrams include 'interest rate', 'labor market', 'basi point'
2. Most common trigrams include 'feder fund rate', 'real interest rate', 'real gdp growth', 'unit labor cost'

Effect of Processing on Dimensionality

The following table represents the effect of pre-processing steps on the dimensionality of the FOMC transcript data.

	All terms	Alpha terms	No stopwords	Stems	MWE
# terms	6249776	5519606	2505261	2505261	2438480
Unique terms	26030	24801	24611	13734	13823

Substantial reductions, but still very high-dimensional space.

Conclusion

After acquiring, organizing, and pre-processing our data, we are ready to begin analysis.

Rest of course assumes we have constructed a document-term matrix, and covers relevant statistical methodologies.