Text Processing

Natural Language Processing

Master in Business Analytics and Big Data

acastellanos@faculty.ie.edu

Basic Concepts & Ideas

- **Corpus**: Set of documents (our dataset)
- **Documents**: Basic Unit or object (each row in the dataset)
- **Words <> Terms**
  - **Words**: Components of a document (what you see in a .txt)
  - **Terms**: Words processed: the features of the documents (columns in our dataset)

Basic Concepts & Ideas

Figure from J&M 3rd ed. draft, sec 7.1

Basic Concepts & Ideas

```
> inspect(tdm[20:30, 1:30])
<<TermDocumentMatrix (terms: 11, documents: 30)>>
Non-/sparse entries: 1/329
Sparsity           : 100%
Maximal term length: 7
Weighting          : term frequency (tf)

         Docs
Terms     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
  abs     0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absb    0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absenc  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absolut 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  absorb  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  abu     0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  abus    0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  abut    0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  academi 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  acceler 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  accept  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
>
```

Basic Concepts & Ideas

- Binary Weighting

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0, 1\}^{|V|}$.

Basic Concepts & Ideas

- TF Weighting

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Bag-of-words

In [1]:
```
import nltk
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
```

In [2]:
```
shakespeare_df = pd.DataFrame(columns=["book", "words"])
for ii, book in enumerate(nltk.corpus.shakespeare.fileids()):
    shakespeare_df.loc[ii] = (book, " ".join(nltk.corpus.shakespeare.words(book)))
shakespeare_df
```

Out[2]:

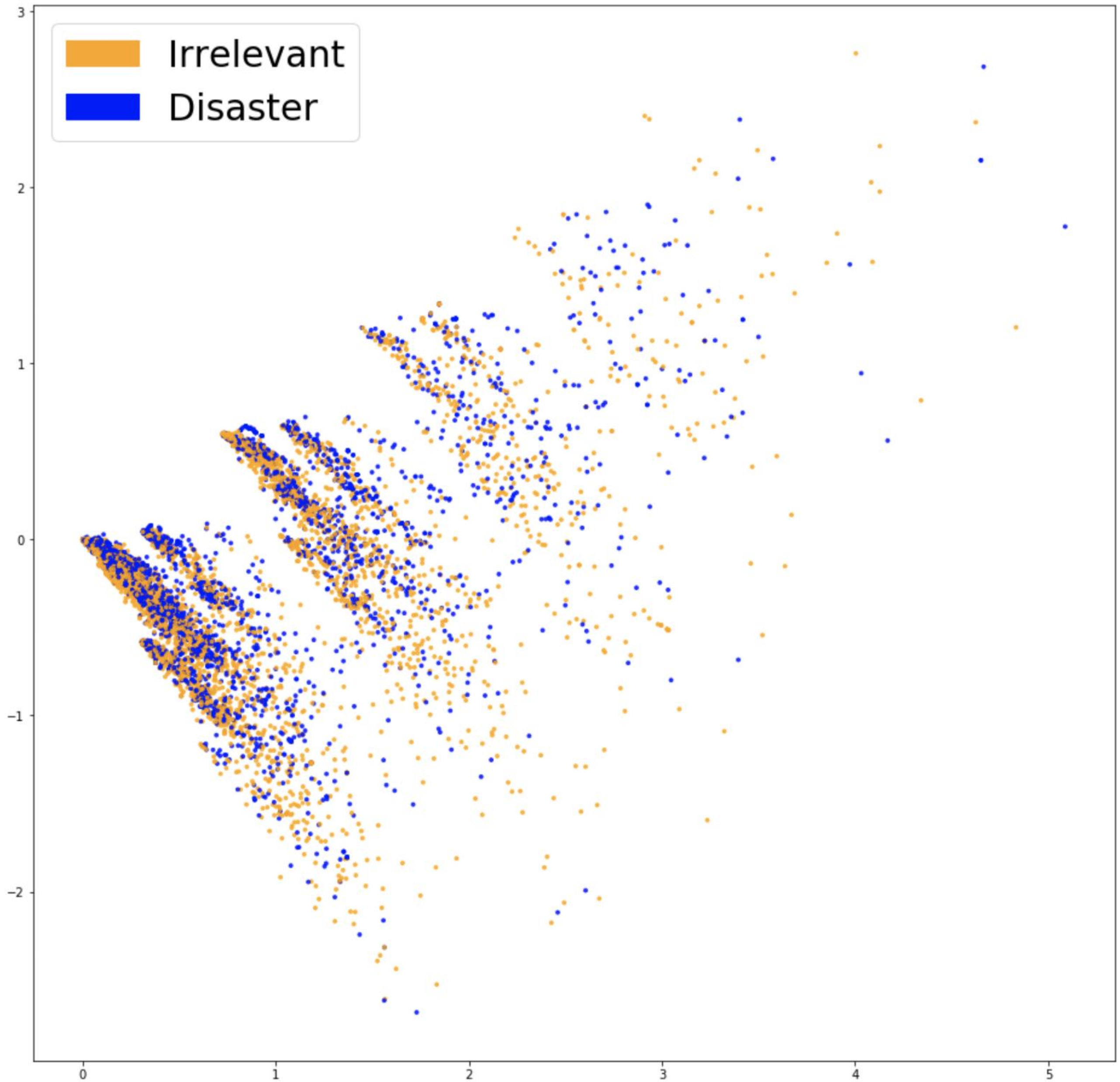
	book	words
0	a_and_c.xml	The Tragedy of Antony and Cleopatra Dramatis P...
1	dream.xml	A Midsummer Night ' s Dream Dramatis Personae ...
2	hamlet.xml	The Tragedy of Hamlet , Prince of Denmark Dram...
3	j_caesar.xml	The Tragedy of Julius Caesar Dramatis Personae...
4	macbeth.xml	The Tragedy of Macbeth Dramatis Personae DUNCA...
5	merchant.xml	The Merchant of Venice Dramatis Personae The D...
6	othello.xml	The Tragedy of Othello , the Moor of Venice Dr...
7	r_and_j.xml	The Tragedy of Romeo and Juliet Text placed in...

In [3]:
```
tf_weighting = CountVectorizer()
tf_shakespeare = tf_weighting.fit_transform(shakespeare_df.words)
pd.DataFrame(tf_shakespeare.A, columns=tf_weighting.get_feature_names())
```

Out[3]:

	1992	1996	1998	1999	abandon	abate	abatements	abbey	abhor	abhorred	...	your	yours	yourself	yourselves	youth	youthful	youths	zeal	zone
0	0	0	0	0	0	0	0	0	0	0	...	140	11	15	1	5	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	...	123	4	3	3	7	0	0	0	0
2	0	0	0	0	0	1	1	0	0	1	...	242	6	15	1	16	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	...	130	10	12	6	0	1	1	0	0
4	0	0	0	0	0	0	0	0	0	1	...	121	3	2	3	1	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	...	175	16	4	0	8	1	0	1	0
6	0	0	0	0	1	0	0	0	3	0	...	205	6	16	0	5	0	0	0	0
7	1	1	1	1	0	1	0	1	0	1	...	103	4	5	0	6	3	0	0	0

8 rows × 11316 columns



Irrelevant
Disaster
3
2
1
0
−1
−2
0
1
2
3
4
5

Basic Concepts & Ideas

- **TF-IDF Weighting**
  - **Based on the idea of Zipf-Law**
    - There are a few very frequent terms and very many very rare terms.

Zipf's law

The i^{th} most frequent term has frequency cf_i proportional to $1/i$:

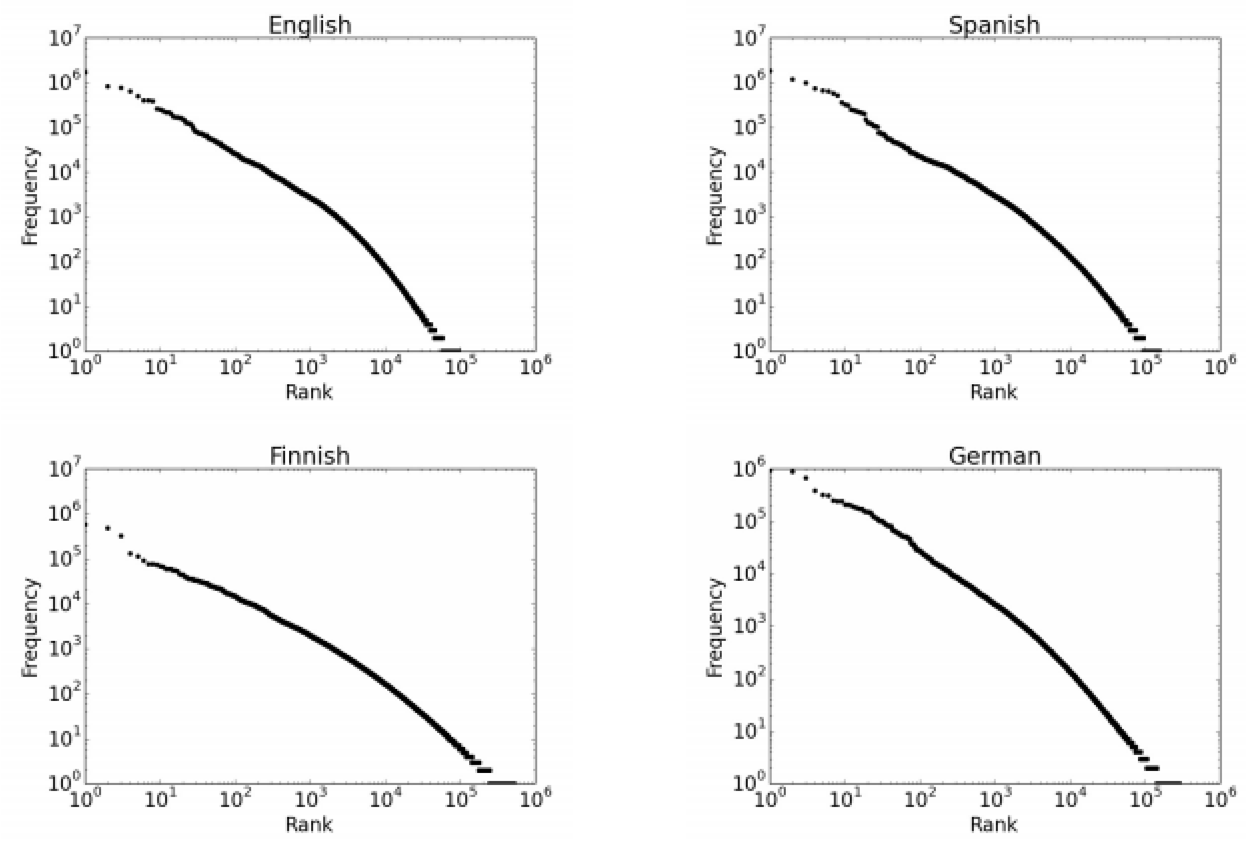
$$cf_i \propto \frac{1}{i}$$

    - *cfi* is collection frequency: the number of occurrences of the term *ti* in the collection

Basic Concepts & Ideas

Word	Count f	rank r	fr
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
two	104	100	10400
turned	51	200	10200
comes	16	500	8000
family	8	1000	8000
brushed	4	2000	8000
Could	2	4000	8000
Applausive	1	8000	8000

Basic Concepts & Ideas



Basic Concepts & Ideas

- TF-IDF Weighting
  - Rare terms are more informative than frequent terms (e.g., ***arachnocentric***).
  - We want **high weights for rare terms** like *arachnocentric*.
  - We want to take into account the **term frequency**

tf-idf weight

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

Basic Concepts & Ideas

tf-idf weight

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

$$w_{\boldsymbol{arachnocentric},d_x} = (1 + \log(10)) \cdot \log\left(\frac{10^5}{1}\right) = 2 \cdot 5$$

$$w_{\boldsymbol{he},d_x} = (1 + \log(877)) \cdot \log\left(\frac{10^5}{10^4}\right) = 4{,}94 \cdot 1$$

Basic Concepts & Ideas

- TF-IDF Weighting

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

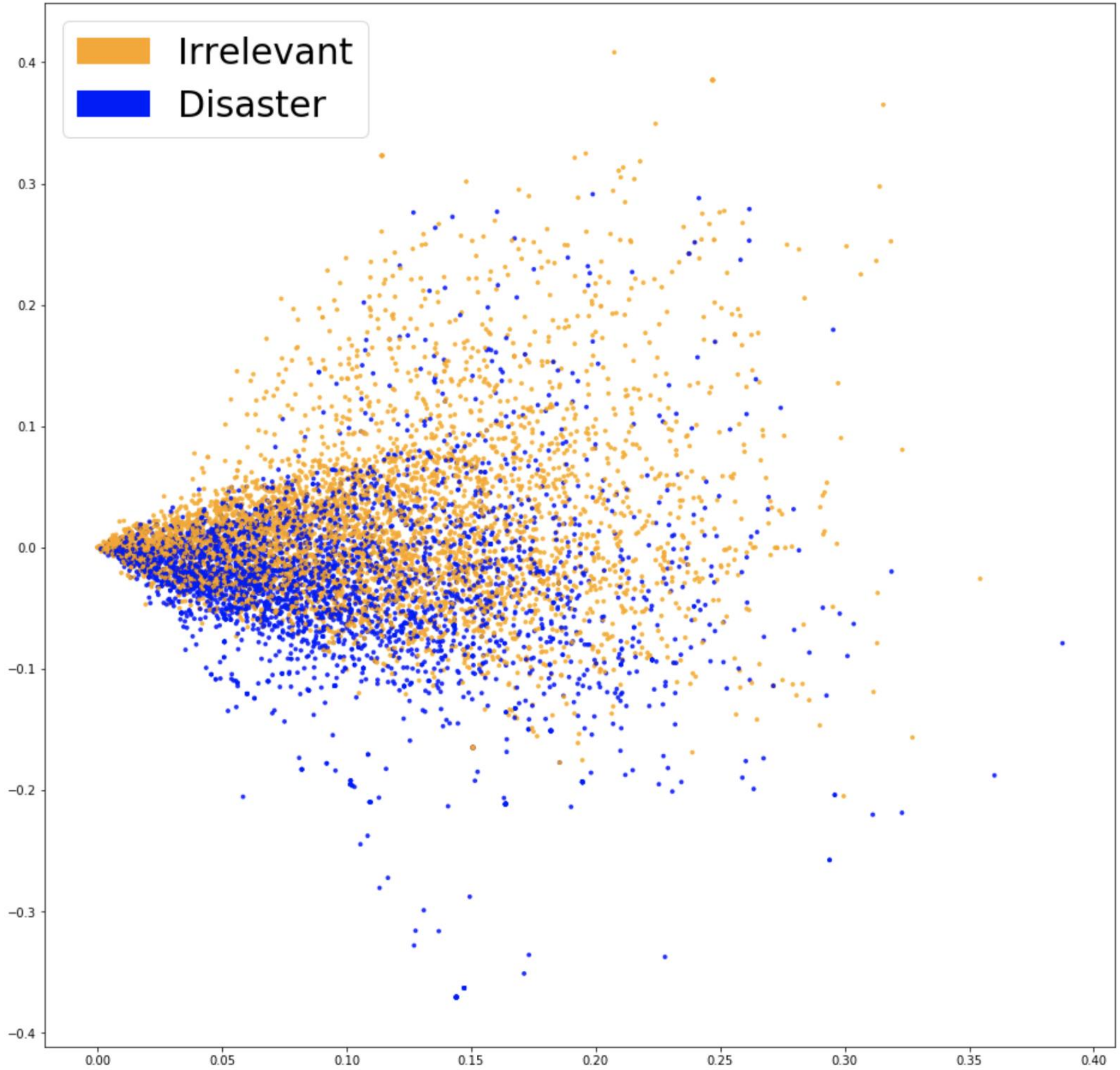
Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

In [5]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_weighting = TfidfVectorizer()
tf_idf_shakespeare = tf_idf_weighting.fit_transform(shakespeare_df.words)
pd.DataFrame(tf_idf_shakespeare.A, columns=tf_idf_weighting.get_feature_names())
```

Out[5]:

	1992	1996	1998	1999	abandon	abate	abatements	abbey	abhor	abhorred	...	your	yours	yourself	yourselves
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.062407	0.004903	0.006686	0.000627
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.001132	0.000000	0.000000	0.000000	0.000000	...	0.087673	0.002851	0.002138	0.003005
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000551	0.000869	0.000000	0.000000	0.000628	...	0.083986	0.002082	0.005206	0.000488
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.071664	0.005513	0.006615	0.004649
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001116	...	0.074558	0.001849	0.001232	0.002598
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000843	0.000000	0.000000	0.000000	0.000000	...	0.092911	0.008495	0.002124	0.000000
6	0.000000	0.000000	0.000000	0.000000	0.000973	0.000000	0.000000	0.000000	0.002918	0.000000	...	0.079621	0.002330	0.006214	0.000000
7	0.001163	0.001163	0.001163	0.001163	0.000000	0.000738	0.000000	0.001163	0.000000	0.000841	...	0.047856	0.001858	0.002323	0.000000

8 rows × 11316 columns


Irrelevant
Disaster
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40

Stopwords

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

Figure 2.5: A stop list of 25 semantically non-selective words which are common in Reuters-RCV1.

- Why?:
  - Extremely common
  - Uninformative: Little discriminative value
- How?:
  - Stoplists
  - Domain Specific stopwords (e.g., HTML tags)
  - **Why not using TF-IDF?**

StopWords

In [3]:
```python
from nltk.corpus import stopwords
stopwords.fileids()
```

Out[3]:
```
[u'arabic',
 u'azerbaijani',
 u'danish',
 u'dutch',
 u'english',
 u'finnish',
 u'french',
 u'german',
 u'greek',
 u'hungarian',
 u'indonesian',
 u'italian',
 u'kazakh',
 u'nepali',
 u'norwegian',
 u'portuguese',
 u'romanian',
 u'russian',
 u'spanish',
 u'swedish',
 u'turkish']
```

In [5]:
```python
stopwords.words("english")[:25]
```

Out[5]:
```
[u'i',
 u'me',
 u'my',
 u'myself',
 u'we',
 u'our',
 u'ours',
 u'ourselves',
 u'you',
 u"you're",
 u"you've",
 u"you'll",
 u"you'd",
 u'your',
 u'yours',
 u'yourself',
 u'yourselves',
 u'he',
 u'him',
 u'his',
 u'himself',
 u'she',
 u"she's",
 u'her',
 u'hers']
```

Regular Expressions

- A **formal language for specifying text strings**
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- **Letters** inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- **Ranges** `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Disjunctions

- **Negations** [^Ss]

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]*	Neither e nor ^	Look here
a^b	The pattern a^b	Look up a^b now

*Carat means negation only when first in []

*Carat means negation only between []

Regular Expressions: More Disjunction

- Woodchucks is another name for groundhog!
- The **pipe** | for disjunction

Pattern	Matches
groundhog|woodchuck	groundhog woodchunk
yours|mine	yours mine
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	groundhog Woodchunk woodchunk Groundhog

Regular Expressions: Wildcards ? * + .

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n	Any symbol	begin begun begun beg3n

Regular Expressions: Anchors ^ $

Pattern	Matches
`^[A-Z]`	`Palo Alto`
`^[^A-Za-z]`	`1` `“Hello”`
`\.$`	`The end.`
`.$`	`The end?` `The end!`

Regular Expressions: Example

- Find me all instances of the word “the” in a text.

  `text= The subject of the report is focused on theology as well as other religious aspects`

`re.findall(r'The', text)` Misses capitalized examples

`re.findall(r'[tT]he', text)` Incorrectly returns `other` or `theology`

`re.findall(r'[^a-zA-Z][tT]he[^a-zA-Z]', text)`

Regular Expressions: Example

- Regular expressions play a **surprisingly large role**
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
    - Named Entities, Dates, web-links…
- For harder tasks, we use **machine learning classifiers**
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

Validating Email Addresses

Authentication system requires users to log in before they can be allowed access to the system, usually using the e-mail as nickname. We can use regular expression to check if an email address supplied is in a valid format.

```
import re

email = "example@gmail.com"

if not re.match(re.compile(r'^.+@[^.].*\.[a-z]{2,10}$', flags=re.IGNORECASE), email):
    print("Enter a valid email address")
else:
    print("Email address is valid")
```

```
Email address is valid
```

```
email = "example@gmail"

if not re.match(re.compile(r'^.+@[^.].*\.[a-z]{2,10}$', flags=re.IGNORECASE), email):
    print("Enter a valid email address")
else:
    print("Email address is valid")
```

```
Enter a valid email address
```

Filtering Unwanted Content

Regular expressions can also be used to filter certain undesired words out of post comments, which is particularly useful in blog posts and social media.

```
curse_words = ["f---", "bar", "baz"]
comment = "This string contains f---."
curse_count = 0

for word in curse_words:
    if re.search(word, comment):
        curse_count += 1

print("Comment has " + str(curse_count) + " curse word(s). Watch your mouth!")
```

```
Comment has 1 curse word(s). Watch your mouth!
```

Word Tokenization

- Split text (document) into words and sentences

There was an earthquake near D.C. I've even felt it in Philadelphia, New York, etc.

```
re.findall(r'\w+|\S\w*', raw)
```

There + was + an + earthquake + near + D.C.

I + ve + even + felt + it + in + Philadelphia, + New + York, + etc.

How many words?

- Special Signs
  - @dbamman have you seen this :) http://popvssoda.com
- I do uh main- mainly business data processing
  - Fragments, filled pauses
- Seuss's cat in the hat is different from other cats!
  - **Lemma**: same stem, part of speech, rough word sense
    - cat and cats = same lemma
  - **Wordform**: the full inflected surface form
    - cat and cats = different wordforms

How many words?

N = number of tokens

```
tokens = nltk.wordpunct_tokenize(raw)
text = nltk.Text(tokens)
```

V = vocabulary = Types = set of tokens

```
words = [w.lower() for w in text]
vocab = sorted(set(words))
```

Church and Gale (1990): $|V| > O(N^{½})$

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Issues in Tokenization

```
Finland's capital      →  Finland Finlands Finland's ?
what're, I'm, isn't    →  What are, I am, is not
Hewlett-Packard        →  Hewlett Packard ?
state-of-the-art       →  state of the art ?
Lowercase            →  lower-case lowercase lower case?
San Francisco        →  one token or two?
m.p.h., PhD.                 →  ??
```

Language Issues

- French
  - ***L'ensemble*** $\rightarrow$ one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - Want ***l'ensemble*** to match with ***un ensemble***

- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German information retrieval needs **compound splitter**

Language Issues

- Chinese and Japanese no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Katakana Hiragana Kanji Romaji

Maximum Matching Word Segmentation Algorithm

- Given a wordlist of Chinese, and a string.
  1. Start a pointer at the beginning of the string
  2. Find the longest word in dictionary that matches the string starting at pointer
  3. Move the pointer over the word in string
  4. Go to 2

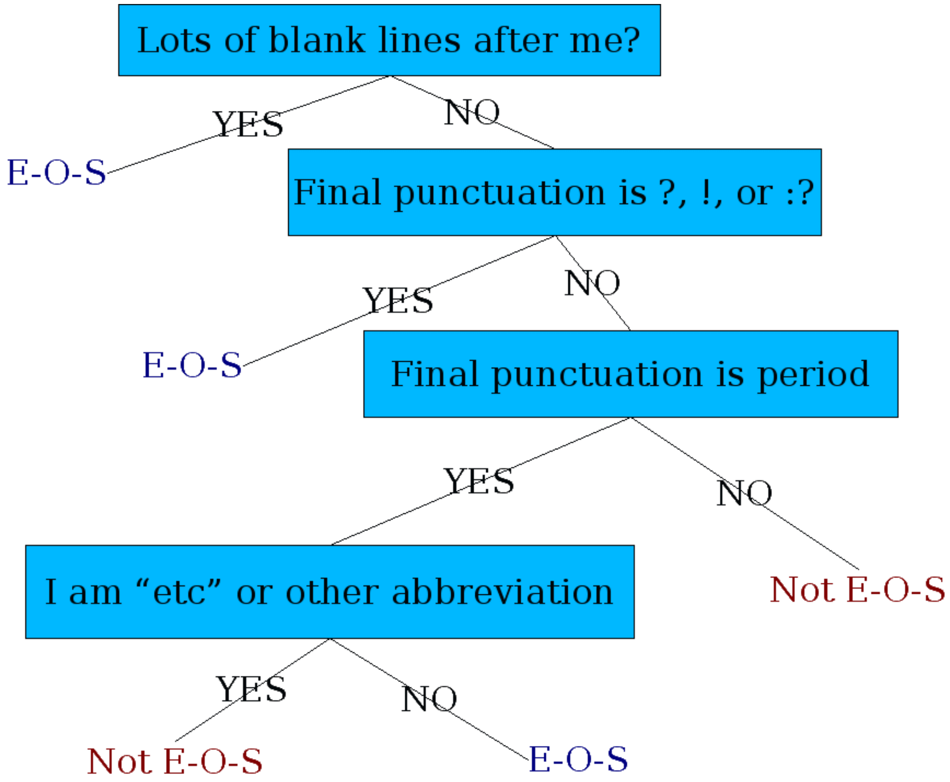
- Doesn't generally work in English!
- But works astonishingly well in Chinese

Sentence Segmentation

- !, ? are relatively unambiguous
- Period "." is quite ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Use a pre-compiled sentence **tokenizer**

```
sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents = sent_tokenizer.tokenize(text)
```

Determining if a word is end-of-sentence: a Decision Tree



More sophisticated decision tree features

- Case of word with ".": Upper, Lower, Cap, Number
- Case of word after ".": Upper, Lower, Cap, Number

- Numeric features
  - Length of word with "."
  - Probability(word with "." occurs at end-of-s)
  - Probability(word after "." occurs at beginning-of-s)

NLTK Tokenizers

- **http://www.nltk.org/api/nltk.tokenize.html**
  - nltk.tokenize.regexp.**RegexpTokenizer**
  - nltk.tokenize.regexp.**WhitespaceTokenizer**
  - nnltk.tokenize.regexp.**WordPunctTokenizer**

  - nltk.tokenize.simple.**LineTokenizer**
  - nltk.tokenize.mwe.**MWETokenizer**

  - nltk.tokenize.casual.**TweetTokenizer**
  - nltk.tokenize.stanford.**CoreNLPTokenize**

Normalization

- **Need to “normalize” terms**
  - Information Retrieval: indexed text & query terms must have same form.
    - We want to match ***U.S.A.*** and ***USA***
- We implicitly **define equivalence classes of terms**
  - e.g., deleting periods in a term
- Alternative: asymmetric expansion:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

Case Folding

- Applications like IR: reduce all letters to lower case
  - Since users tend to use lower case `w.lower()`
  - Possible exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***

- For sentiment analysis, MT, Information extraction
  - Case is helpful (***US*** versus ***us*** is important)

Stemming

Minimal units of meaning

- **Morpheme** = Minimal unit of meaning in a word
  - Walk
  - -ed
- **Simple Words cannot be broken down**
  - Base words or **stems**
- **Affixes are attached to modify meaning**
  - Prefixes, infixes, suffixes, circumfixes

Stemming

- ***Stemming*** **is crude chopping of affixes**
    - language dependent
    - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.
- **Porter's Algorithm:** Link

```
porter = nltk.PorterStemmer()
[porter.stem(t) for t in tokens]
```

for example compressed and compression are both accepted as equivalent to compress.

→

for exampl compress and compress ar both accept as equival to compress

Stemming

- Porter Stemmer

Step 1a

```
sses → ss   caresses → caress
ies  → i    ponies   → poni
ss   → ss   caress   → caress
s    → ø    cats     → cat
```

Step 1b

```
(*v*)ing → ø  walking   → walk
              sing      → sing
(*v*)ed  → ø  plastered → plaster
…
```

Step 2 (for long stems)

```
ational→ ate  relational→ relate
izer→ ize     digitizer → digitize
ator→ ate     operator  → operate
…
```

Step 3 (for longer stems)

```
al    → ø  revival    → reviv
able  → ø  adjustable → adjust
ate   → ø  activate   → activ
…
```

Lemmatization

- **Reduce inflections or variant forms to base form**
  - *am, are, is $\rightarrow$ be*
  - *car, cars, car's, cars' $\rightarrow$ car*
  - *the boy's cars are different colors $\rightarrow$ the boy car be different color*

```
wnl = nltk.WordNetLemmatizer()
[wnl.lemmatize(t) for t in tokens]
```

- Find **correct dictionary headword** form
- Machine translation
  - Spanish quiero (‘I want’), quieres (‘you want’) same lemma as querer ‘want’

Stemming vs Lemmatization

- **Lemmatization** handles **matching "car" to "cars" along with matching "am" to "be".**
  - **Must be** a valid word
- **Stemming** handles **matching "car" to "cars"**
  - **Might not be** a valid word
- **Lemmatization takes much more time than stemming!**
- How much your application depends **on getting the meaning of a word in context correct**:
  - **Machine Translation:** lemmatization to avoid mistranslating a word.
  - **Information Retrieval** over a billion documents with 99% of your queries ranging from 1-3 words, you can settle for stemming.

Is it useful?

Benefits of deep NLP-based lemmatization for information retrieval

stem	deriv	comp	year	MAP	MRR	ret/rel
no	no	no	2005	21.31	48.17	648/939
			2006	18.30	44.95	759/1308

Table 5: Baseline: without stemming

deriv	comp	year	MAP	MRR	P10	ret/rel
no	no	2005	0.3227	0.6491	0.3400	795/939
		2006	0.2797	0.6465	0.3860	987/1308
yes	no	2005	0.3361	0.6983	0.3500	805/939
		2006	0.2933	0.6307	0.4020	1042/1308
no	yes	2005	0.3746	0.7074	0.3660	870/939
		2006	0.3317	0.6827	0.4180	1099/1308
yes	yes	2005	0.3926	0.7698	0.3800	882/939
		2006	0.3482	0.6967	0.4300	1152/1308

Table 6: Results of different stemming methods *without* disambiguation

Try it by yourself!

Raw Text	Processed	Steps	Task	How Pipeline Suits Task
She sells seashells by the seashore.	['she','sell','seashell','seashore']	tokenization, lemmatization, stop word removal, punctuation removal	topic modeling	We only care about high level, thematic, and semantically heavy words
John is capable.	John/PROPN is/VERB capable/ADJ ./PUNCT	tokenization, part of speech tagging	named entity recognition	We care about every word, but want to indicate the role each word plays to build a list of NER candidates
Who won? I didn't check the scores.	[u'who', u'win'], [u'i',u'do',u'not',u'check',u'score']	tokenization, lemmatization, sentence segmentation, punctuation removal, string encoding	sentiment analysis	We need all words, including negations since they can negate positive statements, but don't care about tense or word form

Some useful links

- **Cleaning Tweets:**

  https://github.com/hb20007/hands-on-nltk-tutorial/blob/master/6-1-Twitter-Stream-and-Cleaning-Tweets.ipynb

  - **Notebook for cleaning Twitter data based on RE**

- **How to Write a Spelling Corrector**
  - Peter Norvig's Notebook

  https://norvig.com/spell-correct.html