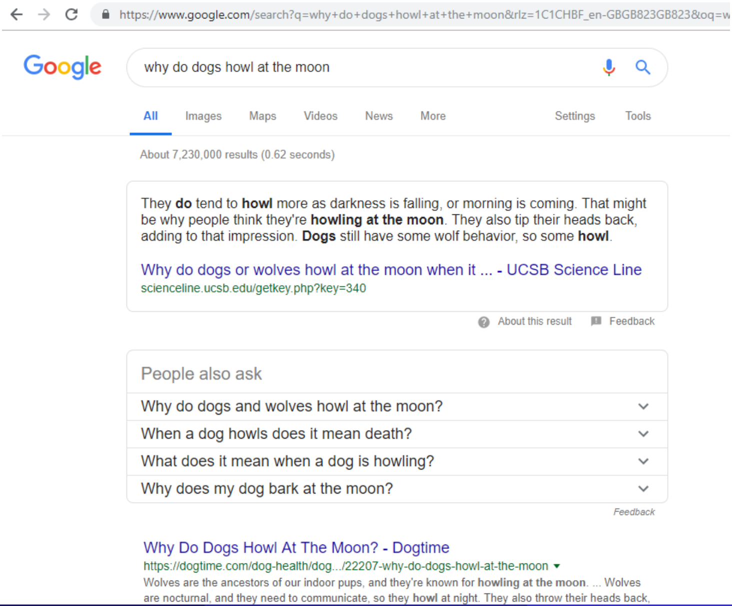
Text Mining

Department of Computer Science
University of Liverpool

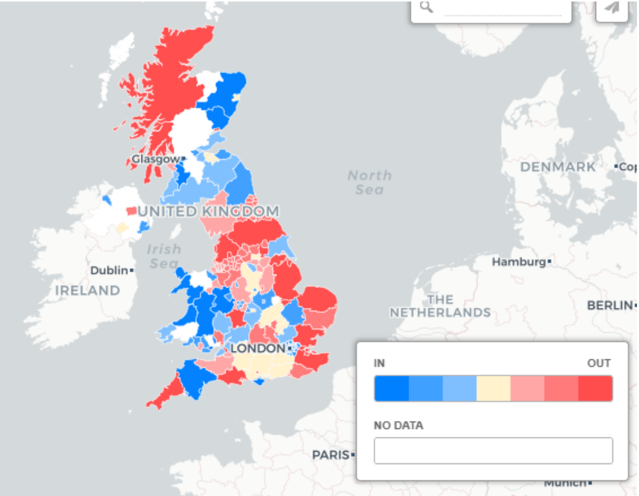
February 27, 2019

Overview

Simple Question: Why do dogs howl at the moon?

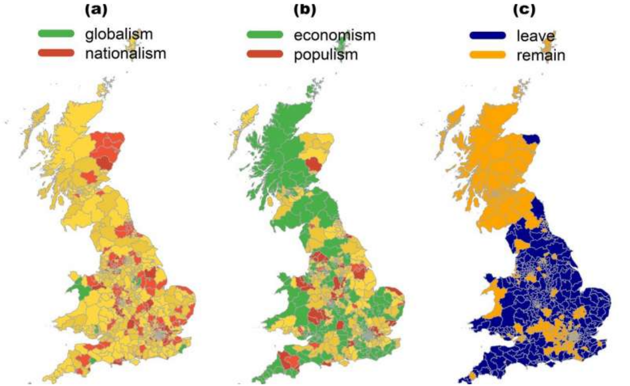


Text Mining Around Us - Sentiment Analysis



source: https://www.jellyfish.co.uk/news-and-views/update-eu-referendum-campaigns-seem-to-be-causing-little-impact

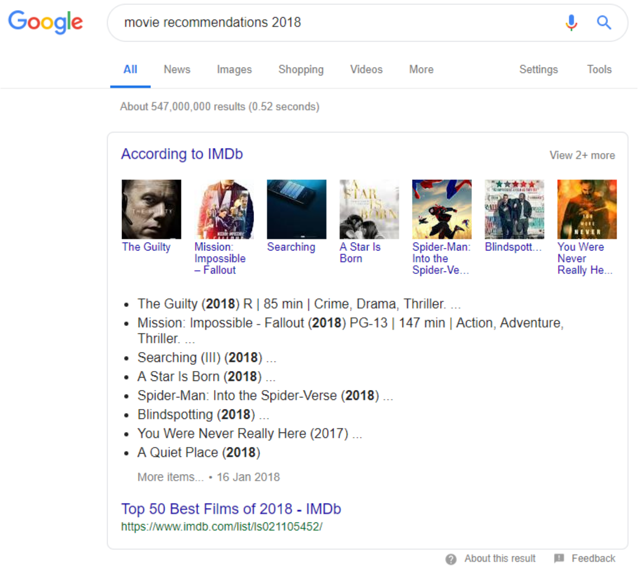
Text Mining Around Us - Opinion Mining



Color-coded heat map of UK parliamentary constituencies (see legend), In graphics (a) and (b), green is used for constituencies showing majority economic and globalist sentiment, and red is used for constituencies showing majority populist and nationalist sentiment. Yellow is the result of adding green to red, with these constituencies somewhere in the middle of the scales. Graphic (c) shows voting patterns in the referendum. Credit: Dr. Marco Bastos and Dr. Dan Mercea

source: https://phys.org/news/2018-04-brexit-debate-twitter-driven-economic.html

Text Mining Around Us - Movie Recommendation Systems



Text Mining Around Us - Document Summarization

All News Images Books Videos More Settings Tools

About 465,000,000 results (0.41 seconds)

Text mining, also referred to as **text** data **mining**, roughly equivalent to **text analytics**, is the process of deriving high-quality information from **text**. High-quality information is typically derived through the devising of patterns and trends through means such as statistical pattern learning.

Text mining - Wikipedia McAfee SECURE

https://en.wikipedia.org/wiki/Text_mining

About this result Feedback

People also ask

- What is text mining and how does it work?
- What is NLP in text mining?
- What are text mining techniques?
- What is text analytics How does it differ from text mining?

Feedback

Text mining

Text mining, also referred to as text data mining, roughly equivalent to text analytics, is the process of deriving high-quality information from text. High-quality information is typically derived through the devising of patterns and trends through means such as statistical pattern learning. Wikipedia

Feedback

Text Mining - Definition and Challenges

- Text mining
  - process of extracting interesting and non-trivial patterns or knowledge from unstructured text documents [Tan et al., 1999].
  - *a.k.a* text data mining [Hearst, 1997],
  - knowledge discovery from textual databases [Feldman and Dagan, 1995]
  - text analytics - application to solve business problems

Text Mining - Challenges

- Unorganized form of data
  - semi-structured or unstructured
- Deriving semantics from content
  - ambiguities at different levels - lexical, syntactic, semantic and pragmatic
  - Text has multiple interpretations
    Teacher Strikes Idle Kids
    Violinist linked to JAL crash blossoms
  - Word sense ambiguity
    Red Tape Holds Up New Bridges
- Non-standard English
  - language in Tweets
  - SOO PROUD of what U accomp.

Text Mining - Challenges

- New Words
  - 850 new words added dictionary at Merriam-Webster.com in 2018
  - Cryptocurrency
  - Chiweenie - a cross between a Chihuahua and a dachshund
  - Dumpster fire - a disastrous event
- Idioms
  - dark horse; get cold feet
- Combining information from multi-lingual texts
- Integrate domain knowledge

Steps in Text Mining

source: http://openminted.eu/text-mining-101/

Text Mining - Preprocessing Steps

- Tokenisation
- Stemming
- Stopword Removal
- Sentence Segmentation

Tokenisation

- Process of splitting text into words
- What is a word?

  string of contiguous alphanumeric characters with space on either side; may include hyphens and apostrophes, but no other punctuation marks [Kučera and Francis, 1967].

- Useful clue - space or tab (English)

Tokenisation - Problems

- Periods
  - usually helps if we remove them
  - but useful to retain in certain cases such as $22.50; Ed.,
- hypenation
  - useful to retain in some cases e.g., state-of-the-art
  - better to remove in other cases e.g., gold-import ban, 50-year-old
- Single apostrophes
  - useful to remove them e.g., *is'nt*, *didn't*
- space may not be a useful clue all the time
- sometimes we want to use words separated by space as 'single' word
- For example:
  - San Francisco
  - University of Liverpool
  - Danushka Bollegala

Regular Expressions for Tokenisation

- Regular Expressions Cheatsheet

REGEX	NOTE	EXAMPLE	EXPLANATION
\s	white space	\d\s\d	digit space digit
\S	not white space	\d\S\d	digit non-whitespace digit
\d	digit	\d\d\d-\d\d-\d\d\d\d	SSN
\D	not digit	\D\D\D	three non-digits
\w	word character (letter, number, or _)	\w\w\w	three word chars
\W	not a word character	\W\W\W	three non-word chars
[...]	any included character	[a-z0-9#]	any char that is a thru z, 0 thru 9, or #
[^...]	no included character	[^xyx]	any char but x, y, or z
*	zero or more	\w*	zero or more words chars
+	one or more	\d+	integer
?	zero or one	\d\d\d-?\d\d-?\d\d\d\d	SSN with dashes being optional
|	or	\w|\d	word or digit character

Regular Expressions for Tokenisation

```
raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
        though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
        well without--Maybe it's always pepper that makes people hot-tempered,'..."""

import re
print re.split(r' ', raw)
["'When", "I'M", 'a', "Duchess,'", 'she', 'said', 'to', 'herself,', '(not', 'in', 'a',
 'very', 'hopeful', 'tone\n\t\t', '', 'though),', "'I", "won't", 'have', 'any',
 'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL.', 'Soup', 'does', 'very\n\t\t', '',
 'well', 'without--Maybe', "it's", 'always', 'pepper', 'that', 'makes', 'people',
 "hot-tempered,'..."]

print re.split(r'[ \t\n]+', raw)
["'When", "I'M", 'a', "Duchess,'", 'she', 'said', 'to', 'herself,', '(not', 'in', 'a',
 'very', 'hopeful', 'tone', 'though),', "'I", "won't", 'have', 'any', 'pepper', 'in',
 'my', 'kitchen', 'AT', 'ALL.', 'Soup', 'does', 'very', 'well', 'without--Maybe',
 "it's", 'always', 'pepper', 'that', 'makes', 'people', "hot-tempered,'..."]

print re.findall(r"\w+ (?:[-']\w+)*|'|[-.(]+|\S\w*", raw)
["'", 'When ', 'I', "'", 'M ', 'a ', 'Duchess', ',', "'", 'she ', 'said ', 'to ',
'herself', ',', '(', 'not ', 'in ', 'a ', 'very ', 'hopeful ', 'tone', 'though',
')', ',', "'", 'I ', 'won', "'", 't ', 'have ', 'any ', 'pepper ', 'in ', 'my ',
'kitchen ', 'AT ', 'ALL', '.', 'Soup ', 'does ', 'very', 'well ', 'without', '--',
'Maybe ', 'it', "'", 's ', 'always ', 'pepper ', 'that ', 'makes ', 'people ',
'hot', '-', 'tempered', ',', "'", '...']
```

Stanford Parser for Tokenisation

```
raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
            though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
            well without--Maybe it's always pepper that makes people hot-tempered,'..."""

path_to_parser_jar = 'lib/stanford-parser.jar'
path_to_models_jar = 'lib/stanford-parser-3.5.1-models.jar'

# POS Tagger
from nltk.tokenize.stanford import StanfordTokenizer
tokenizer = StanfordTokenizer(path_to_parser_jar)

tokenized_text = tokenizer.tokenize(raw)
print tokenized_text

[u'`', u'When', u'I', u"'M", u'a', u'Duchess', u',', u"'", u'she', u'said', u'to',
u'herself', u',', u'-LRB-', u'not', u'in', u'a', u'very', u'hopeful', u'tone', u'though',
u'-RRB-', u',', u"'", u'I', u'wo', u"n't", u'have', u'any', u'pepper', u'in', u'my',
u'kitchen', u'AT', u'ALL', u'.', u'Soup', u'does', u'very', u'well', u'without', u'--',
u'Maybe', u'it', u"'s", u'always', u'pepper', u'that', u'makes', u'people', u'hot-tempered',
u',', u"'", u'...']
```

Tokenisation

- Tokenisation turns out to be more difficult than one expects
- No single solution works well
- Decide what counts as a token depending on the application domain

spaCy (https://spacy.io/)

- spaCy - a relatively new package for “Industrial strength NLP in Python”.
- Developed by Matt Honnibal at Explosion AI
- Designed with applied data scientist in mind
- spaCy supports:
  - Tokenisation
  - Lemmatisation
  - Part-of-speech tagging
  - Entity recognition
  - Dependency parsing
  - Sentence recognition
  - Word-to-vector transformations

spaCy - Feature Comparison

	SPACY	SYNTAXNET	NLTK	CORENLP
Programming language	Python	C++	Python	Java
Neural network models	✓	✓	✗	✓
Integrated word vectors	✓	✗	✗	✗
Multi-language support	✓	✓	✓	✓
Tokenization	✓	✓	✓	✓
Part-of-speech tagging	✓	✓	✓	✓
Sentence segmentation	✓	✓	✓	✓
Dependency parsing	✓	✓	✗	✓
Entity recognition	✓	✗	✓	✓
Coreference resolution	✗	✗	✗	✓

source: https://spacy.io/usage/facts-figures

spaCy - Benchmarks

SYSTEM	YEAR	LANGUAGE	ACCURACY	SPEED (WPS)
spaCy v2.x	2017	Python / Cython	**92.6**	*n/a* ?
spaCy v1.x	2015	Python / Cython	91.8	13,963
ClearNLP	2015	Java	91.7	10,271
CoreNLP	2015	Java	89.6	8,602
MATE	2015	Java	92.5	550
Turbo	2015	C++	92.4	349

source: https://spacy.io/usage/facts-figures

spaCy - Detailed Speed Comparison

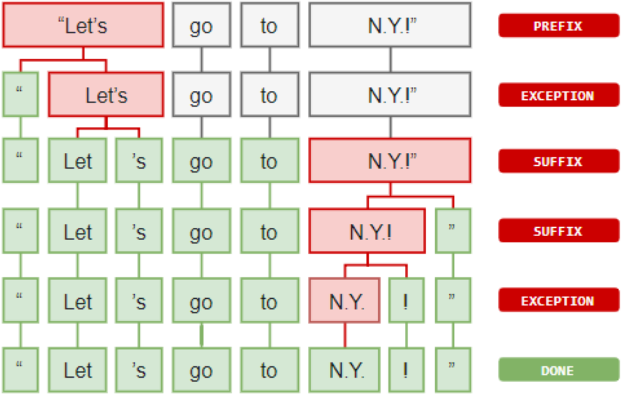
	ABSOLUTE (MS PER DOC)			RELATIVE (TO SPACY)		
SYSTEM	TOKENIZE	TAG	PARSE	TOKENIZE	TAG	PARSE
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	*n/a*	20x	443x	*n/a*

source: https://spacy.io/usage/facts-figures

Tokenization in SPACY

- Tokenizes text into words, puntuations and so on.
- Applies rules specific to each language
- Step 1: Split raw text based on whitespace characters (text.split(' '))
- Step 2: Processes each substring from left to right and performs two checks:
  - Does the substring match a tokenizer exception rule
  - e.g., "don't" ==> no whitespace ==> but split into two tokens "do" and "nt
  - "U.K." ==> remain as one token

Tokenization in spaCy



source: https://spacy.io/usage/spacy-101

Tokenization in spaCy

Editable code example (experimental)

v2.0.18 · Python 3 · via Binder

```python
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Apple is looking at buying U.K. startup for $1 billion')

for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
          token.shape_, token.is_alpha, token.is_stop)
```

RUN

source: https://spacy.io/usage/spacy-101

Tokenization in spaCy

TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	VERB	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False
startup	startup	NOUN	NN	dobj	xxxx	True	False
for	for	ADP	IN	prep	xxx	True	True
$	$	SYM	$	quantmod	$	False	False
1	1	NUM	CD	compound	d	False	False
billion	billion	NUM	CD	pobj	xxxx	True	False

source: https://spacy.io/usage/spacy-101

Stemming

- Removal of inflectional ending from words (strip off any affixes)
  - connections, connecting, connect, connected → connect
- Problems
  - Can conflate semantically different words
    - *Gallery* and *gall* may both be stemmed to *gall*
- Lemmatization: a further step to ensure that the resulting form is a word present in a dictionary

Regular Expressions for Stemming

```
import re
print re.findall(r'^(.*)(ing|ly|ed|ions|ies|ive|es|s|ment)$', 'processing')
[('process', 'ing')]

import re
print re.findall(r'^(.*)(ing|ly|ed|ions|ies|ive|es|s|ment)$', 'processes')
[('processe', 's')]

```

- note that the star operator is "greedy"
- the **.*** part of expression tries to consume as much as the input as possible
- for non-greedy version of the star operator $=$ ***?**

```
import re
print re.findall(r'^(.*?)(ing|ly|ed|ions|ies|ive|es|s|ment)$', 'processes')
[('process', 'es')]
```

Regular Expressions for Stemming

```python
import nltk, re

def stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ions|ies|ive|es|s|ment)?$'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

raw = """DENNIS: Listen, strange women lying in ponds distributing swords
        is no basis for a system of government. Supreme executive power derives from
        a mandate from the masses, not from some farcical aquatic ceremeony."""

tokens = nltk.word_tokenize(raw)
print [stem(t) for t in tokens]

['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'ly', 'in', 'pond', 'distribut', 'sword',
 'i', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern', '.', 'Supreme', 'execut', 'power',
 'deriv', 'from', 'a', 'mandate', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
 'farcical', 'aquatic', 'ceremeony', '.']

```

Regular Expressions for Stemming

```python
import nltk, re

def stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ions|ies|ive|es|s|ment)?$'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

raw = """DENNIS: Listen, strange women lying in ponds distributing swords
        is no basis for a system of government. Supreme executive power derives from
        a mandate from the masses, not from some farcical aquatic ceremeony."""

tokens = nltk.word_tokenize(raw)
print [stem(t) for t in tokens]

['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'ly', 'in', 'pond', 'distribut', 'sword',
 'i', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern', '.', 'Supreme', 'execut', 'power',
 'deriv', 'from', 'a', 'mandate', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
 'farcical', 'aquatic', 'ceremeony', '.']
```

- Problems
  - RE removes ‘s’ from ‘ponds’, but also from ‘is’ and ‘basis’
  - produces some non-words like ‘distribut’, ‘deriv’

NLTK Stemmers

- NLTK provides several off-the-shelf stemmers
- Porter and Lancaster stemmers have their own rules for stripping affixes

```
import nltk, re

raw = """DENNIS: Listen, strange women lying in ponds distributing swords
        is no basis for a system of government. Supreme executive power derives from
        a mandate from the masses, not from some farcical aquatic ceremeony."""

porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()
tokens = nltk.word_tokenize(raw)

print [porter.stem(t) for t in tokens]
[u'denni', ':', 'listen', ',', u'strang', 'women', u'lie', 'in', u'pond', u'distribut',
 u'sword', 'is', 'no', u'basi', 'for', 'a', 'system', 'of', u'govern', '.', u'suprem',
 u'execut', 'power', u'deriv', 'from', 'a', u'mandat', 'from', 'the', u'mass', ',', 'not',
 'from', 'some', u'farcic', u'aquat', u'ceremeoni', '.']

print [lancaster.stem(t) for t in tokens]
['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond', 'distribut', 'sword',
'is', 'no', 'bas', 'for', 'a', 'system', 'of', 'govern', '.', 'suprem', 'execut', 'pow',
'der', 'from', 'a', 'mand', 'from', 'the', 'mass', ',', 'not', 'from', 'som', 'farc',
'aqu', 'ceremeony', '.']
```

Is stemming useful?

- Provides some improvement for IR performance (especially for smaller documents).
- Very useful for some queries, but on an average does not help much.
- Since improvement is very minimal, often IR engines does not use stemming.

Stopword Removal

- Removal of high frequency words
- Most common words such as articles, prepositions, and pronouns etc. does not help in identifying meaning

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

Methods for stopword removal

- Classic Method
  - removing stop-words using pre-compiled lists
- Zipf's law (Z-methods)
  - frequency of a word is inversely proportional to its rank in the frequency table
  - remove most frequent words
- Mutual Information Method
  - supervised method that computes mutual information between a given term and a document class
  - low mutual information suggests low discrimination power of the term and hence should be removed

Sentence Segmentation

- Divide text into sentences
- Involves identifying **sentence boundaries** between words in different sentences
- *a.k.a* sentence boundary detection, sentence boundary disambiguation, sentence boundary recognition
- Useful and necessary for various NLP tasks such as
  - sentiment analysis
  - relation extraction
  - question answering systems
  - knowledge extraction

Sentence boundary detection algorithms

- Heuristic methods
- Statistical classification trees [Riley, 1989]
  - probability of a word occurring before or after a boundary, case and length of words
- Neural Networks [Palmer and Hearst, 1997]
  - POS distribution of preceding and following words
- Maximum entropy model [Mikheev 1998]

Sentence Segmentation - Using spaCy

Editable code example (experimental)

v2.0.18 · Python 3 · via Binder

```python
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u"This is a sentence. This is another sentence.")
for sent in doc.sents:
    print(sent.text)
```

RUN

```
This is a sentence.
This is another sentence.
```

Sentence Segmentation - Using spaCy

Editable code example (experimental)

v2.0.18 · Python 3 · via Binder

```python
import spacy

text = u"this is a sentence...hello...and another sentence."

nlp = spacy.load('en_core_web_sm')
doc = nlp(text)
print('Before:', [sent.text for sent in doc.sents])

def set_custom_boundaries(doc):
    for token in doc[:-1]:
        if token.text == '...':
            doc[token.i+1].is_sent_start = True
    return doc

nlp.add_pipe(set_custom_boundaries, before='parser')
doc = nlp(text)
print('After:', [sent.text for sent in doc.sents])
```

RUN

```
Before: ['this is a sentence...', 'hello...and another sentence.']
After: ['this is a sentence...', 'hello...', 'and another sentence.']
```

Part-of-Speech Tagging (POS)

- Task of tagging POS tags (Nouns, Verbs, Adjectives, Adverbs, ...) for words
- POS tags provide lot of information about a word
  - knowing whether a word is **noun** or **verb** gives information about neighbouring words
  - nouns are preceded by determiners; adjectives and verbs by nouns
  - useful for Named entity recognition; Machine Translation; Parsing; Word sense disambiguation

- Given a word, we assume it can belong to only of the POS tags.

- POS Tagging problem
  - Given a sentence $S = w_1 w_2 .... w_n$ consisting of n words, determine the corresponding tag sequence $P = P_1 P_2 .... P_n$

POS Tagging - Challenges

- Words often have more than one POS: e.g., back
  - *The back door* = adjective (JJ)
  - *On my back* = noun (NN)
  - *Win the voters back* = adverb (RB)
  - *Promised to back the bill* = verb (VB)

POS Tagging - Tagset

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	$
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Figure: Penn Treebank POS Tags

POS Tagging - Brown Corpus

- **Brown Corpus** - standard corpus used for POS tagging task
- first text corpus of American English
- published in 1963-1964 by Francis and Kucera
- consists of 1 million words (500 samples of 2000+ words each)
- Brown corpus is PoS tagged with Penn TreeBank tagset.
- $\approx$ 11% of the word types are ambiguous with regard to POS
- $\approx$ 40% of the word tokens are ambiguous
- ambiguity for common words. e.g. **that**
  - I know **that** he is honest = preposition (IN)
  - Yes, **that** play was nice = determiner (DT)
  - You can't to **that** far = adverb (RB)

Automatic POS Tagging

- Symbolic
  - Rule-based
  - Transformation-based
- Probabilistic
  - Hidden Markov Models
  - Maximum Entropy Markov Models
  - Conditional Random Fields

Automatic POS Tagging - Brill Tagger

- An example of Transformation-Based Learning
  - Basic idea: do a quick job first (using frequency), then revise it using contextual rules.
  - Painting metaphor from the readings
- Very popular (freely available, works fairly well)
- A supervised method: requires a tagged corpus

Automatic POS Tagging - Brill Tagger

- Start with simple (less accurate) rules…learn better ones from tagged corpus
  - Tag each word initially with most likely POS
  - Examine set of transformations to see which improves tagging decisions compared to tagged corpus
  - Re-tag corpus using best transformation
  - Repeat until, e.g., performance doesn't improve
  - Result: tagging procedure (ordered list of transformations) which can be applied to new, untagged text

Automatic POS Tagging: Brill Tagger - Example

- Examples:
  - They are expected to race tomorrow.
  - The race for outer space.
- Tagging algorithm:
  1. Tag all uses of "race" as NN (most likely tag in the Brown corpus)
     - They are expected to race/NN tomorrow
     - the race/NN for outer space
  2. Use a transformation rule to replace the tag NN with VB for all uses of "race" preceded by the tag TO:
     - They are expected to race/VB tomorrow
     - the race/NN for outer space

Automatic POS Tagging: Brill Tagger - Sample Final Rules

```
Rules:
NN -> NNP if the tag of words i+1...i+2 is 'NNP'
NN -> VB if the tag of the preceding word is 'TO'
NN -> VBD if the tag of the following word is 'DT'
NN -> VBD if the tag of the preceding word is 'NNS'
NN -> JJ if the tag of the preceding word is 'DT', and the tag of the followi
ng word is 'NN'
NN -> NNP if the tag of the preceding word is 'NN', and the tag of the follow
ing word is ','
NN -> NNP if the tag of words i+1...i+2 is 'NNP'
NN -> IN if the tag of the preceding word is '.'
NNP -> NN if the tag of words i-3...i-1 is 'JJ'
NN -> JJ if the tag of the following word is 'JJ'
NN -> VBP if the tag of the preceding word is 'PRP'
WDT -> IN if the tag of the following word is 'DT'
NN -> JJ if the tag of the preceding word is 'IN', and the tag of the followi
ng word is 'NN'
NN -> VBN if the tag of the preceding word is 'VBP'
VBD -> VB if the tag of the preceding word is 'MD'
NN -> JJ if the tag of the preceding word is 'CC', and the tag of the followi
ng word is 'NN'
```

Next Week

- Probabilistic Models for POS Tagging
- Relation Extraction
- Question and Answering