Text Mining

February 2018

Outline

- Text Mining - Introduction
- Text Mining Pipeline (Preprocessing)
  - Tokenisation
  - Stemming
  - Stopword Removal
  - Sentence Segmentation
- Part-Of-Speech (POS) tagging

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 - 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
- Challenges
  - deriving semantics from the content
  - combine information from multi-lingual texts
  - integrate domain knowledge

Text Mining - Framework

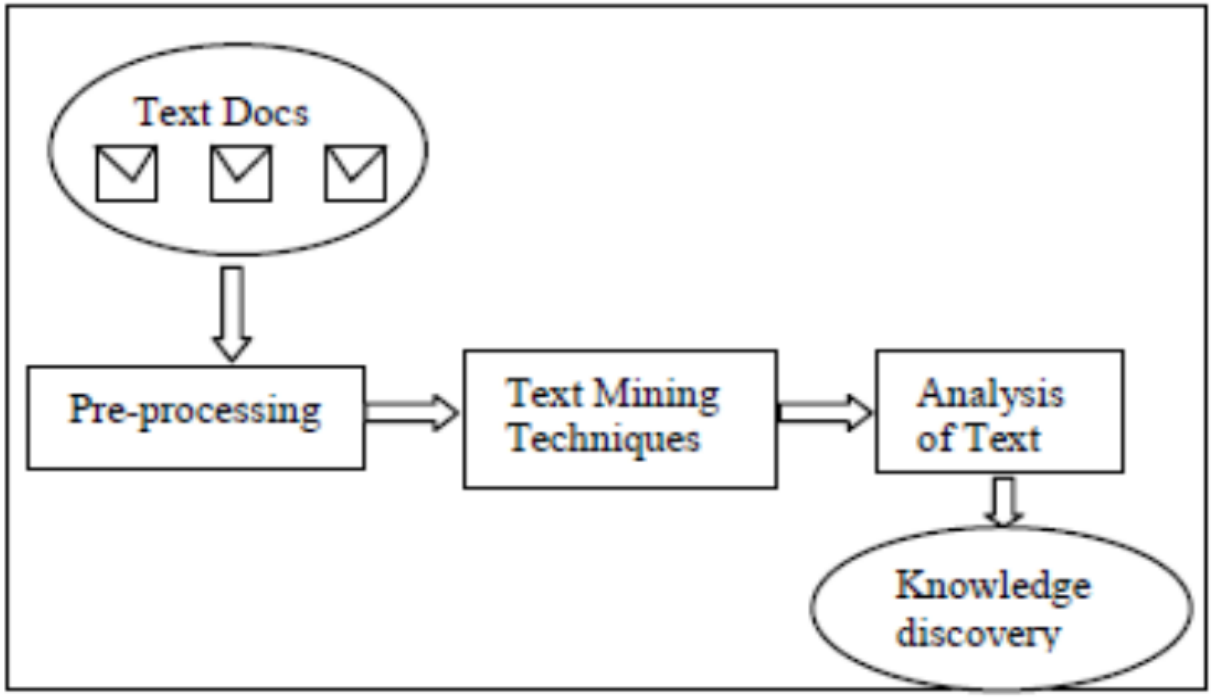


Figure1: Framework of Text Mining

Text Mining - Preprocessing

- 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.,
- Hyphenation
  - 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

Tokenisation - Problems

- Periods
  - usually helps if we remove them
  - but useful to retain in certain cases such as $22.50; Ed.,
- Hyphenation
  - 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'...']
```

- Stanford CoreNLP - https://stanfordnlp.github.io/CoreNLP/

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

Stemming

- Removal of inflectional ending from words (strip off any affixes)
  - connections, connecting, connect, connected $\rightarrow$ 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

```
1  import re
2  print re.findall(r'^(.*)(ing|ly|ed|ions|ies|ive|es|s|ment)$', 'processing')
3  [('process', 'ing')]
4
5  import re
6  print re.findall(r'^(.*)(ing|ly|ed|ions|ies|ive|es|s|ment)$', 'processes')
7  [('processe', 's')]
8
```

- 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 = ***?**

```
9  import re
10 print re.findall(r'^(.*?)(ing|ly|ed|ions|ies|ive|es|s|ment)$', 'processes')
11 [('process', 'es')]
12
13
```

Regular Expressions for Stemming

```
78  import nltk, re
79
80  def stem(word):
81      regexp = r'^(.*?)(ing|ly|ed|ions|ies|ive|es|s|ment)?$'
82      stem, suffix = re.findall(regexp, word)[0]
83      return stem
84
85  raw = """DENNIS: Listen, strange women lying in ponds distributing swords
86            is no basis for a system of government. Supreme executive power derives from
87            a mandate from the masses, not from some farcical aquatic ceremeony."""
88
89  tokens = nltk.word_tokenize(raw)
90  print [stem(t) for t in tokens]
91
92  ['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'ly', 'in', 'pond', 'distribut', 'sword',
93   'i', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern', '.', 'Supreme', 'execut', 'power',
94   'deriv', 'from', 'a', 'mandate', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
95   'farcical', 'aquatic', 'ceremeony', '.']
96
97
```

Regular Expressions for Stemming

```
78  import nltk, re
79
80  def stem(word):
81      regexp = r'^(.*?)(ing|ly|ed|ions|ies|ive|es|s|ment)?$'
82      stem, suffix = re.findall(regexp, word)[0]
83      return stem
84
85  raw = """DENNIS: Listen, strange women lying in ponds distributing swords
86            is no basis for a system of government. Supreme executive power derives from
87            a mandate from the masses, not from some farcical aquatic ceremeony."""
88
89  tokens = nltk.word_tokenize(raw)
90  print [stem(t) for t in tokens]
91
92  ['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'ly', 'in', 'pond', 'distribut', 'sword',
93   'i', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern', '.', 'Supreme', 'execut', 'power',
94   'deriv', 'from', 'a', 'mandate', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
95   'farcical', 'aquatic', 'ceremeony', '.']
96
97
```

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

NLTK Stemmers

- NLTK (https://www.nltk.org/) provides off-the-shelf stemmers
- Porter Stemmer (*www.nltk.org/modules/nltk/stem/porter.html*)
  Lancaster Stemmer (*www.nltk.org/modules/nltk/stem/lancaster.html*)
- Stemmers have their own rules for stripping affixes

```
1  import nltk, re
2
3  raw = """DENNIS: Listen, strange women lying in ponds distributing swords
4            is no basis for a system of government. Supreme executive power derives from
5            a mandate from the masses, not from some farcical aquatic ceremeony."""
6
7  porter = nltk.PorterStemmer()
8  lancaster = nltk.LancasterStemmer()
9  tokens = nltk.word_tokenize(raw)
10
11 print [porter.stem(t) for t in tokens]
12 [u'denni', ':', 'listen', ',', u'strang', 'women', u'lie', 'in', u'pond', u'distribut',
13  u'sword', 'is', 'no', u'basi', 'for', 'a', 'system', 'of', u'govern', '.', u'suprem',
14  u'execut', 'power', u'deriv', 'from', 'a', u'mandat', 'from', 'the', u'mass', ',', 'not',
15  'from', 'some', u'farcic', u'aquat', u'ceremeoni', '.']
16
17 print [lancaster.stem(t) for t in tokens]
18 ['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond', 'distribut', 'sword',
19 'is', 'no', 'bas', 'for', 'a', 'system', 'of', 'govern', '.', 'suprem', 'execut', 'pow',
20 'der', 'from', 'a', 'mand', 'from', 'the', 'mass', ',', 'not', 'from', 'som', 'farc',
21 '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.

Stemming vs. Lemmatization

- Stemming
  - crude heuristic process that chops off the ends of words
  - often includes the removal of derivational affixes
  - Example: 'Gallery' , 'gall' $\rightarrow$ 'gall'
- Lemmatization
  - doing things more properly, using a vocabulary and morphological analysis of words
  - aims to remove inflectional endings only
  - return the base or dictionary form of a word, known as lemma
  - Example: 'better' $\rightarrow$ 'good'

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: Stop list of 25 semantically non-selective words common in Reuters-RCV1

- Uses
  - significantly reduce feature space
  - speed up learning

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]

NLTK Tools for Sentence Segmentation: Punkt Sentence Segmenter

```
import nltk, re

sent_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents = sent_tokenizer.tokenize(text)
print sents[100:110]

[u'inquired Gregory sarcastically.',
 u'"I tell you," went on Syme with passion, "that every time a \
    train\ncomes in I feel that it has broken past batteries of besiegers, \
    and\nthat man has won a battle against chaos.',
 u'You say contemptuously\nthat when one has left Sloane Square one must come \
   to Victoria.', u'I\nsay that one might do a thousand things instead, and that \
   whenever\nI really come there I have the sense of hairbreadth escape.',
 u"And\nwhen I hear the guard shout out the word 'Victoria,' it is not \
   an\nunmeaning word.",
 u'It is to me the cry of a herald announcing\nconquest.',
 u'It is to me indeed \'Victoria\'; it is the victory of\nAdam."',
 u'Gregory wagged his heavy, red head with a slow and sad smile.',]
```

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
  - provide useful features for **named entity recognition**

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
  - provide useful features for **named entity recognition**
- 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 $P = P_1 P_2 .... P_n$

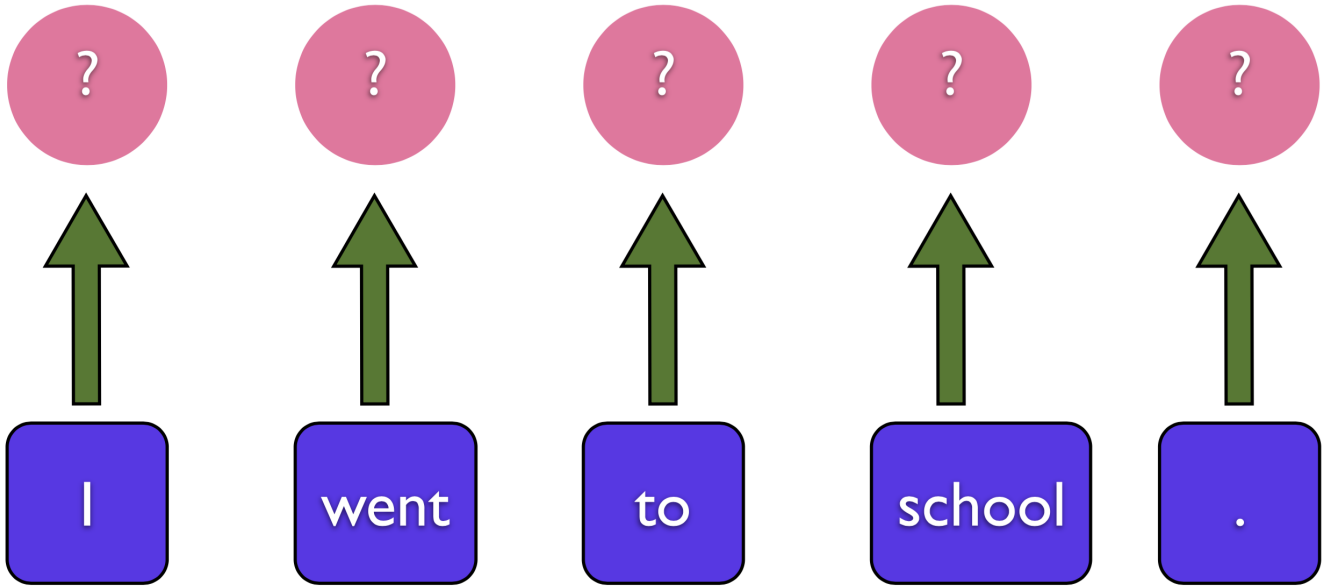
Part-of-Speech Tagging (POS)

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*			

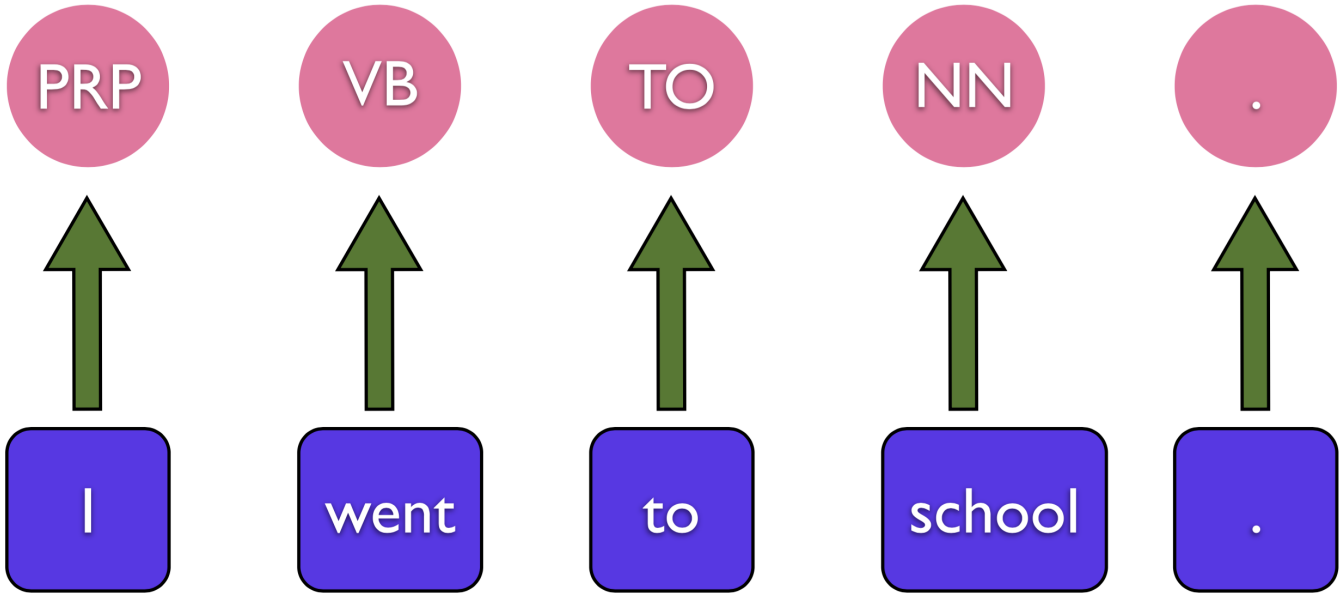
Figure: Penn Treebank POS Tags

Hidden Markov Model (HMM)

Given a sequence of words (observed states) determine a sequence of state transitions (unobserved states)

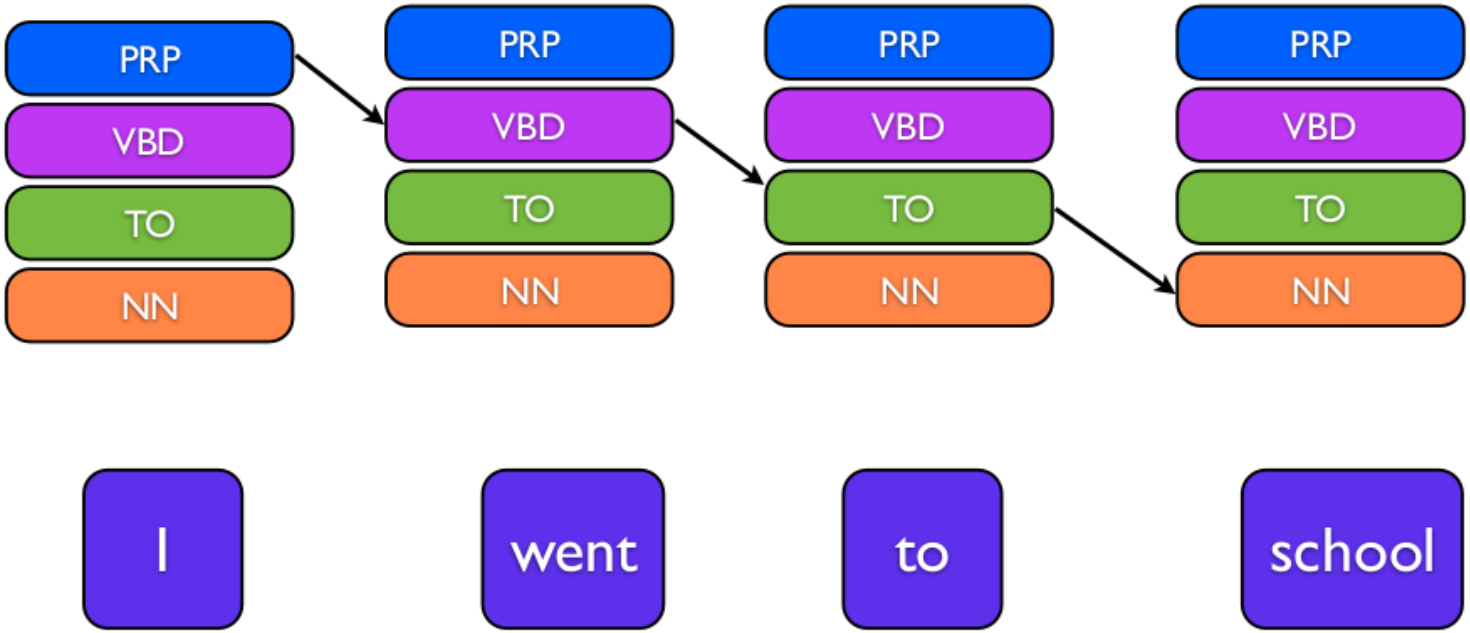


Hidden Markov Model (HMM)



Hidden Markov Model (HMM)

Why is it difficult?



Even if there were only four POS tags, then this is just one of 4x4x4x4 = 256 possible state sequences!

Hidden Markov Model (HMM)

Quiz 1

- If there are T number of POS tags, and you have a sentence with N number of words, how many different possible POS tag sequences can you get?

- **Next Class**
  - Look at POS Tagging task in detail
  - Hidden Markov Model
  - Three Problems of HMMs
    - Decoding; Computing likelihood; Learning
  - Decoding - Viterbi Algorithm

Cavnar, W. B., Trenkle, J. M., et al. (1994).
N-gram-based text categorization.
Ann Arbor MI, 48113(2):161–175.

Feldman, R. and Dagan, I. (1995).
Knowledge discovery in textual databases (kdt).
In *KDD*, volume 95, pages 112–117.

Hearst, M. A. (1997).
Texttiling: Segmenting text into multi-paragraph subtopic passages.
Computational linguistics, 23(1):33–64.

Kučera, H. and Francis, W. N. (1967).
Computational analysis of present-day American English.
Dartmouth Publishing Group.

Padró, M. and Padró, L. (2004).
Comparing methods for language identification.
Procesamiento del lenguaje natural, 33:155–162.

Cavnar, W. B., Trenkle, J. M., et al. (1994).
N-gram-based text categorization.
Ann Arbor MI, 48113(2):161–175.

Feldman, R. and Dagan, I. (1995).
Knowledge discovery in textual databases (kdt).
In *KDD*, volume 95, pages 112–117.

Hearst, M. A. (1997).
Texttiling: Segmenting text into multi-paragraph subtopic passages.
Computational linguistics, 23(1):33–64.

Kučera, H. and Francis, W. N. (1967).
Computational analysis of present-day American English.
Dartmouth Publishing Group.

Padró, M. and Padró, L. (2004).
Comparing methods for language identification.
Procesamiento del lenguaje natural, 33:155–162.