COM6115: Text Processing

Background: Basic Technologies

Mark Hepple

Department of Computer Science
University of Sheffield

Tokenisation

- Text is commonly provided as a *string*
    - ◇ but, difficult to process in this format

- Usually more convenient to work with a list of *tokens*

  i.e. sequence of units we might find in a dictionary

  e.g. "*The bird sang*" ⇝ <*"The","bird","sang"*>

- *Tokenisation*: task of converting text string to list of tokens

- Simplest approach: split string on whitespace chars
    - ◇ but, can produce anomalous results

  e.g. *"Soon, he leaves."* ⇝ <*"Soon,", "he", "leaves."*>

  Here, the item "Soon," mixes word and punctuation

Tokenisation (contd)

- Instead, might split on whitespace and all punctuation chars
  - ◇ again, produces anomalous results

    e.g. *"He won't go."* $\rightsquigarrow$ <*"He", "won", "t" "go"*>

    Here, item "*won't*" is effectively lost
  - ◇ But, may want to preserve punctuation as separate tokens
- System might combine several methods, for example:
  - ◇ knowledge of tokens that include punctuation (e.g. *I'm*)
  - ◇ components recognising special cases, e.g. date 12/4/99
  - ◇ assumption that remaining punctuation chars are separate tokens

    e.g. *"Soon, I'm leaving."*

    $\rightsquigarrow$ <*"Soon", ",", "I'm", "leaving", "."*>

Stemming

- For some tasks, want to treat morphological variants as if they were the same single term
  - ◇ called **term conflation**

- For example, may want to find documents about *inventing*
  - ◇ might be signalled by presence of any of:

    invent, invents, invented, inventing, invention, inventions, . . .

- Term conflation can be done using a *lemmatizer*
  - ◇ reduces word to its *stem*, using linguistic knowledge
  - ◇ lexical look-up for irregulars

    e.g. *geese* → *goose, sought* → *seek*
  - ◇ otherwise, morphological rules applied
  - ◇ computational expense of lexical look-up

Stemming: the Porter stemmer

- Alternatively, may use a *simple stemmer*
  - ◇ encodes little/no linguistic knoweldge
  - ◇ no lexical look-up — much faster
  - ◇ ‘ad hoc’ morphology, without access to the lexicon
- Best-known simple stemmer: *the Porter stemmer*
  - ◇ provides set of rules to strip-off / substitute suffixes
  - ◇ first applicable rule used, rules applied iteratively
- E.g. for *plurals*, have rules such as:

IF word ends in:

–ies	(but not *–eies, –aies*)	::	*–ies* ⇒ *–y*	
–es	(but not *–aes, –ees, –oes*)	::	*–es* ⇒ *–e*	
–s	(but not *–us, –ss*)	::	*–s* ⇒ *–∅*	(null)

Stemming: problems

- Problems of stemming
  - ◇ over conflation, e.g. (porter)

    police/policy, university/universe, execute/executive
  - ◇ under conflation, e.g. (porter)

    Europe/European, machine/machinary, matrix/matrices
  - ◇ may produce obscure non-word stems – hard to interpret

    e.g. (porter) *iteration* → *iter*, *general* → *gener*
- Evidence that stemming brings (limited) benefits for some tasks

  e.g. *information retrieval*

Stemming: example (Porter)

- Initial text:

  In this department we focus on both the scientific and engineering aspects of computer science in teaching and research. This is reflected in our emphasis on the theoretical principles of computer science and their application to a wide range of systems.

- Porter output:

  in thi depart we focu on both the scientif and engin aspect of comput scienc in teach and research. thi is reflect in our emphasi on the theoret principl of comput scienc and their applic to a wide rang of system.

- Code free at: `www.tartarus.org/~martin/PorterStemmer`

Part of Speech Tagging

- Task of assigning *part of speech tags* to words in text
  - ◇ *tags*: atomic labels denoting parts of speech

    e.g. for input: `The dog ran home`

    return: `The/Det dog/N ran/V home/N`

- There are ∼10 core parts of speech. *BUT* *tagsets* used for practical corpus work usually *much bigger*, e.g
  - ◇ Penn Treebank tagset: 45 tags
  - ◇ Lancaster C7 tagset: 145 tags

- Why?
  - ◇ Incorporate additional, more fine-grained, distinctions concerning morphological form and syntactic function

POS Tagging: PTB tagset

- Penn Treebank (PTB) *Noun* tags:

NN	noun, singular or mass	*boy*
NNS	noun, plural	*boys*
NNP	proper noun	*Bill*
NNPS	plural proper noun	*Carolinas*

- PTB *Adjective* tags:

JJ	adjective	*big*
JJR	adjective, comparative	*bigger*
JJS	adjective, superlative	*biggest*

POS Tagging: PTB tagset (contd)

- Penn Treebank (PTB) *Verb* tags:

`VB`	verb, base form	*take*
`VBZ`	verb, present, 3rd sing.	*takes*
`VBP`	verb, present, other	*take*
`VBD`	verb, past	*took*
`VBG`	verb, present participle	*taking*
`VBN`	verb, past participle	*taken*
`MD`	model verb	*can, would*

POS Tagging: why is it difficult?

- POS tagging is non-trivial due to *ambiguity*
  - ◇ many words have >1 POS

    — must choose correct tag
  - ◇ encounter *unknown* words

    — must guess correct tag
  - ◇ exploit constraints from lexicon and local context, and morphological knowledge

The	green	trains	run	down	that	track
Det	Adj/NN	NNS/VBZ	NN/VB	Prep/Adv/Adj	SC/Pron	NN/VB
Det	Adj	NNS	VB	Prep	Pron	NN

POS Tagging: motivation

- POS tagging useful for further syntactic analysis
  - ◇ For 'full' parsing – greatly reduces search space
  - ◇ Provides basis for 'shallow' ('chunk') parsing methods
  - ◇ Handling of unknown words (robustness)
- Also useful for other text/NLP applications:
  - ◇ Text-to-Speech generation – pronunciation

    e.g. prosody (*stress* in pronunciation): **ex**port (noun) vs. ex**port** (verb)

    e.g. correct pronunciation of *wound* in:

    He <u>wound</u> the clock :: *wound*/`VBD` ⇝ "wow-nd"

    His <u>wound</u> is deep :: *wound*/`NN` ⇝ "woo-nd"
  - ◇ Spelling correction – filtering correction candidates

    e.g. *I will <u>wund</u> the clock* :: may correct to *wind* (`VB`), but **not** *wand* (`NN`)
  - ◇ Information Retrieval – sense disambiguation

POS Tagging: approaches

- Automatic POS taggers use a lexicon listing possible POS tags for each known word
  - ◇ may include information of relative likelihood of each tag
- Hand-written systems
  - ◇ use hand-written set of rules to eliminate incorrect candidate tags for ambiguous words
  - ◇ large overhead for creating such systems
- Data-driven approaches
  - ◇ large corpus of text with *correct* tag assigned to words
  - ◇ use to *train* a system to make disambiguation decisions
  - ◇ various *machine learning* approaches used, such as:
    - — statistical, e.g. *hidden markov models*
    - — rule-based, e.g. *transformation-based learning* (Brill)