

# Text Preprocessing

COMP90042

Natural Language Processing

Lecture 2

Semester 1 Week 1  
Jey Han Lau



THE UNIVERSITY OF  
MELBOURNE

# Definitions

- Words
  - Sequence of characters with a meaning and/or function
- Sentence
  - “The student is enrolled at the University of Melbourne.”
- Document: one or more sentences.
- Corpus: a collection of documents.
- Word **token**: each *instance* of a word.
  - E.g. 9 word tokens in the example sentence.
- Word **type**: *distinct* words.
  - Lexicon (“dictionary”): a group of word types.
  - E.g. 8 word types in the example sentence.

# How many words (types) are there in English?

- < 10K
- < 50K
- < 100K
- < 500K
- ???

[PollEv.com/jeyhanlau569](https://PollEv.com/jeyhanlau569)



# How Many Unique Words?

	#Tokens (N)	#Type (IVI)
Switchboard phone conversation	2.4 million	20 thousand
Shakespeare	800 thousand	31 thousand
Google N-gram	1 trillion	13 million

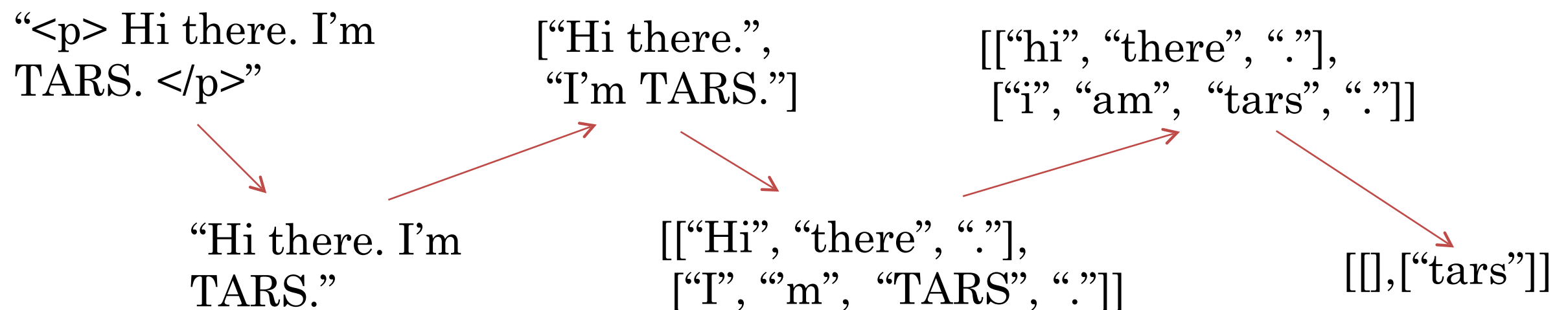
Church and Gale (1990):  $IVI > O(N^{1/2})$

# Why Preprocess?

- Most NLP applications have documents as inputs:
  - “This movie is so great!!! U should definitely watch it in the theater! Best sci-fi eva!” → 😊
  - “Eu estive em Melbourne no ano passado.” → “I was in Melbourne last year.”
- **Key point:** language is **compositional**. As humans, we can break these documents into individual components. To understand language, a computer should do the same.
- **Preprocessing** is the first step.

# Preprocessing Steps

1. Remove unwanted formatting (e.g. HTML)
2. **Sentence segmentation**: break documents into sentences
3. **Word tokenisation**: break sentences into words
4. **Word normalisation**: transform words into canonical forms
5. **Stopword removal**: delete unwanted words



# Sentence Segmentation

“Hi there. I’m  
TARS.”  [“Hi there.”,  
“I’m TARS.”]

# Sentence Segmentation

- Naïve approach: break on sentence punctuation ([.?!])
  - But periods are used for abbreviations!  
(U.S. dollar, ..., Yahoo! as a word)
- Second try: use regex to require capital ([.?!] [A-Z])
  - But abbreviations often followed by names (Mr. Brown)
- Better yet: have lexicons
  - But difficult to enumerate all names and abbreviations
- State-of-the-art uses machine learning, not rules



# Binary Classifier

- Looks at every “.” and decides whether it is the end of a sentence.
  - Decision trees, logistic regression
- Features
  - Look at the words before and after “.”
  - Word shapes:
    - Uppercase, lowercase, ALL\_CAPS, number
    - Character length
  - Part-of-speech tags:
    - Determiners tend to start a sentence

# Word Tokenisation

["Hi there.",  
"I'm TARs."] → [[["Hi", "there", "."],  
["I", "m", "TARs", "."]]]

# Word Tokenisation: English

- Naïve approach: separate out alphabetic strings (`\w+`)
- Abbreviations (*U.S.A.*)
- Hyphens (*merry-go-round* vs. *well-respected* vs. *yes-but*)
- Numbers (*1,000,00.01*)
- Dates (*3/1/2016*)
- Clitics (*n't* in *can't*)
- Internet language (*http://www.google.com*, *#metoo*, *:-*)
- Multiword units (*New Zealand*)

# Word Tokenisation: Chinese

- Some Asian languages are written without spaces between words
- In Chinese, words often correspond to more than one character

墨大      的      学生      与众不同

Unimelb      's      students (are)      special

# Word Tokenisation: Chinese

- Standard approach assumes an existing vocabulary
- MaxMatch algorithm
  - Greedily match longest word in the vocabulary

$V = \{\text{墨,大,的,学,生,与,众,不,同, 墨大,学生,不同,与众不同}\}$

墨大的学生与众不同

match 墨大, match 的, match 学生, match 与众不同,

move to 的    move to 学    move to 与    done

# Word Tokenisation: Chinese

- But how do we know what the vocabulary is
- And doesn't always work

去 买 新西兰 花

go buy New Zealand flowers

去 买 新 西兰花

go buy new broccoli

# Word Tokenisation: German

- *Lebensversicherungsgesellschaftsangestellter*
- = *life insurance company employee*
- Requires compound splitter

# Subword Tokenisation

- *Colourless green ideas sleep furiously* →  
*[colour] [less] [green] [idea] [s] [sleep] [furious] [ly]*
- One popular algorithm: byte-pair encoding (BPE)
- Core idea: iteratively merge frequent pairs of characters
- Advantage:
  - Data-informed tokenisation
  - Works for different languages
  - Deals better with unknown words



# Byte-Pair Encoding

- Corpus
  - [5] l o w \_
  - [2] l o w e s t \_
  - [6] n e w e r \_
  - [3] w i d e r \_
  - [2] n e w \_
- Vocabulary
  - \_, d, e, i, l, n, o, r, s, t, w

# Byte-Pair Encoding

- Corpus
  - [5] l o w \_
  - [2] l o w e s t \_
  - [6] n e w e r \_
  - [3] w i d e r \_
  - [2] n e w \_
- Vocabulary
  - \_, d, e, i, l, n, o, r, s, t, w, r \_

# Byte-Pair Encoding

- Corpus
  - [5] l o w \_
  - [2] l o w e s t \_
  - [6] n e w er\_
  - [3] w i d er\_
  - [2] n e w \_
- Vocabulary
  - \_, d, e, i, l, n, o, r, s, t, w, r\_, er\_

# Byte-Pair Encoding

- Corpus

- [5] l o w \_
- [2] l o w e s t \_
- [6] n ew er \_
- [3] w i d er \_
- [2] n ew \_

- Vocabulary

- \_, d, e, i, l, n, o, r, s, t, w, r\_, er\_, ew

# Byte-Pair Encoding

- Corpus
  - [5] l o w \_
  - [2] l o w e s t \_
  - [6] new er\_
  - [3] w i d er\_
  - [2] new \_
- Vocabulary
  - \_, d, e, i, l, n, o, r, s, t, w, r\_, er\_, ew, new

# Byte-Pair Encoding

- Vocabulary

- \_\_, d, e, i, l, n, o, r, s, t, w, r\_\_, er\_\_, ew, new
- \_\_, d, e, i, l, n, o, r, s, t, w, r\_\_, er\_\_, ew, new, **ow**
- \_\_, d, e, i, l, n, o, r, s, t, w, r\_\_, er\_\_, ew, new, ow, **low**
- \_\_, d, e, i, l, n, o, r, s, t, w, r\_\_, er\_\_, ew, new, ow, low, **newer\_\_**
- \_\_, d, e, i, l, n, o, r, s, t, w, r\_\_, er\_\_, ew, new, ow, low, newer\_\_, **low\_\_**

# Byte-Pair Encoding

- In practice BPE will run with thousands of merges, creating a large vocabulary
- Most frequent words will be represented as full words
- Rarer words will be broken into subwords
- In the worst case, unknown words in test data will be broken into individual letter

# Word Normalisation

[[“Hi”, “there”, “.”],  
[“I”, “m”, “TARS”, “.”]]  [[“hi”, “there”, “.”],  
[“i”, “am”, “tars”, “.”]]



# Word Normalisation

- Lower casing (Australia → australia)
- Removing morphology (cooking → cook)
- Correcting spelling (definitely → definitely)
- Expanding abbreviations (U.S.A → USA)
- Goal:
  - Reduce vocabulary
  - Maps words into the same type

# Inflectional Morphology

- Inflectional morphology creates grammatical variants
- English inflects nouns, verbs, and adjectives
  - Nouns: *number* of the noun (-s)
  - Verbs: *number* of the subject (-s), the *aspect* (-ing) of the action and the *tense* (-ed) of the action
  - Adjectives: *comparatives* (-er) and *superlatives* (-est)
- Many languages have much richer inflectional morphology than English
  - E.g. French inflects nouns for gender (*un chat, une chatte*)

# Lemmatisation

- Lemmatisation means removing any inflection to reach the uninflected form, the *lemma*
  - speaking → speak
- In English, there are irregularities that prevent a trivial solution:
  - poked → poke (not pok)
  - stopping → stop (not stopp)
  - watches → watch (not watche)
  - was → be (not wa)
- A lexicon of lemmas needed for accurate lemmatisation

# Derivational Morphology

- Derivational morphology creates distinct words
- English derivational *suffixes* often change the lexical category, e.g.
  - -ly (personal → personally)
  - -ise (final → finalise)
  - -er (write → writer)
- English derivational *prefixes* often change the meaning without changing the lexical category
  - write → rewrite
  - healthy → unhealthy

# Stemming

- Stemming strips off all suffixes, leaving a *stem*
  - E.g. automate, automatic, automation → automat
  - Often not an actual lexical item
- Even less lexical sparsity than lemmatisation
- Popular in information retrieval
- Stem not always interpretable

# The Porter Stemmer

- Most popular stemmer for English
- Applies rewrite rules in stages
  - First strip inflectional suffixes,
    - E.g. *-ies* → *-i*
  - Then derivational suffixes
    - E.g. *-isation* → *-ise* → *-i*

# The Porter Stemmer

- c (lowercase) = consonant; e.g. 'b', 'c', 'd'
- v (lowercase) = vowel; e.g. 'a', 'e', 'i', 'o', 'u'
- C = a sequence of consonants
  - s, ss, tr, bl
- V = a sequence of vowels
  - o, oo, ee, io

# The Porter Stemmer

- A word has one of the four forms:
  - CVCV ... C
  - CVCV ... V
  - VCVC ... C
  - VCVC ... V
- Which can be represented as:
  - [C]VCVC ... [V]
  - [C] (VC)<sup>m</sup> [V]
  - m = **measure**



$[C] (VC)^m [V]$ 

- TREE

- = CV
- = C(VC)<sup>0</sup>V
- [m=0]

- TREES

- = CVC
- = C(VC)<sup>1</sup>
- [m=1]


- TROUBLES

- = CVVC
- = C(VC)<sup>2</sup>
- [m=2]

# Examples

- $m=0$ : TR, EE, TREE, Y, BY
- $m=1$ : TROUBLE, OATS, TREES, IVY
- $m=2$ : TROUBLES, PRIVATE, OATEN, ORRERY





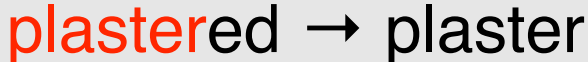

# The Porter Stemmer

- Rules format: (condition) S1 → S2
- e.g. (m > 1) EMENT → null
  - ▶  **CVCVC** = C(VC)<sup>2</sup> = [m=2]
  - ▶ **REPLACEMENT**
  - ▶ → REPLAC
- Always use the longest matching S1
  - ▶ CARE**SSES** → CARE**SS**
  - ▶ CARE**SS** → CARE**SS**
  - ▶ CARE**S** → CARE

Rules:  
SSES → SS  
IES → I  
SS → SS  
S → null

# The Porter Stemmer

- Step 1: plurals and inflectional morphology

	Rule	Positive Example	Negative Example
a	SSES → SS	caresses → caress	
	IES → I	ponies → poni	
	SS → SS	caress → caress	
	S → null	cats → cat 	
b	(m>0) EED → EE	 agreed → agree	 feed → feed
	(*v*) ED → null *v* = stem has vowel	 plastered → plaster	 bled → bled
	(*v*) ING →	motoring → motor	sing → sing
b+	AT → ATE	conflat(ed) → conflate	
c	(*v*) Y → I	happy → happi	

# The Porter Stemmer

- Step 2, 3, 4: derivational inflections

	Rule	Positive Example
2	(m>0) ATIONAL → ATE	relational → relate
	(m>0) TIONAL → TION	conditional → condition
	(m>0) ENCI → ENCE	valenci → valence
	(m>0) ANCI → ANCE	hesitanci → hesitance
3	(m>0) ICATE → IC	triplicate → triplic
	(m>0) ATIVE → null	formative → form
	(m>0) ALIZE → AL	formalize → formal
4	(m>1) AL → null	revival → reviv
	(m>1) ER → null	airliner → airlin
	(m>1) ATE → null	activate → activ

# The Porter Stemmer

- Step 5: tidying up

	Rule	Positive Example
5	$(m > 1) E \rightarrow \text{null}$	probate $\rightarrow$ probat
	$(m > 1 \text{ and } *d \text{ and } *L)$ null $\rightarrow$ single letter *d = stem ends with double consonant (e.g. -TT) *L = stem ends with 'l'	controll $\rightarrow$ control

# The Porter Stemmer

- computational → comput
  - step 2: ATIONAL → ATE: computate
  - step 4: ATE → null: comput
- computer → comput
  - step 4: ER → null: comput

# Stopword Removal

[[“hi”, “there”, “.”],  
[“i”, “am”, “tars”, “.”]]  [[], [“tars”]]



# Stop Words

- Definition: a list of words to be removed from the document
  - Typical in bag-of-word (BOW) representations
  - Not appropriate when sequence is important
- How to choose them?
  - All *closed-class* or *function* words
    - E.g. *the, a, of, for, he, ...*
  - Any high frequency words
  - NLTK, spaCy NLP toolkits

# A Final Word

- Preprocessing unavoidable in text analysis
- Can have a major effect on downstream applications
- Exact steps may vary depending on corpus, task
- Simple rule-based systems work well, but rarely perfectly
- Language-dependent

# Further Reading

- J&M3 Ch 2.4
- Details on the Porter Stemmer algorithm <http://snowball.tartarus.org/algorithms/porter/stemmer.html>