



# Text Normalization



SLP3 Ch 2; INLP Ch 4.3.1

# Text Normalization

---

- ▶ Every NLP task requires text normalization:
  - ▶ Tokenizing (segmenting) words
  - ▶ Normalizing word formats
  - ▶ Segmenting sentences





# Word Tokenization



# Space-based tokenization

---

- ▶ A very simple way to tokenize
  - ▶ For language that use space characters between words
    - ▶ Arabic, Cyrillic, Greek, Latin, etc., based writing systems
  - ▶ Segment off a token between instances of spaces (and punctuations)
- ▶ Example
  - ▶ Sentence:  
`I love natural language processing, too.`
  - ▶ Tokenized:  
`I_love_natural_language_processing_too`
    - ▶ Represent the delimiter string as “\_”



# Issues in tokenization

---

- ▶ Can't just blindly remove punctuation:
  - ▶ [m.p.h.](#), [Ph.D.](#), [AT&T](#), [cap'n](#)
  - ▶ Prices ([\\$45.55](#))
  - ▶ Dates ([01/02/06](#))
  - ▶ URLs (<https://www.shanghaitech.edu.cn>)
  - ▶ Hashtags ([#nlproc](#))
  - ▶ Email addresses ([someone@shanghaitech.edu.cn](mailto:someone@shanghaitech.edu.cn))
- ▶ Clitic: a word that doesn't stand on its own
  - ▶ “[are](#)” in [we're](#), French “[je](#)” in [j'ai](#), “[le](#)” in [l'honneur](#)
- ▶ When should multiword expressions (MWE) be words?
  - ▶ [New York](#), [rock 'n' roll](#)



# Tokenization in Natural Language Toolkit (NLTK)

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

---

## ▶ Sentence:

That U.S.A. poster-print costs \$12.40...

## ▶ Expected output:

That\_U.S.A.\_poster-print\_costs\_\$12.40\_...

### ▶ Words with optional internal hyphens:

That, poster-print, costs

### ▶ Abbreviations: U.S.A.

### ▶ Currency and percentages: \$12.40

### ▶ Ellipsis: ...

### ▶ Separate tokens: ] [ . , ; ” ’ ? ( ) : - \_ `



# Tokenization in Natural Language Toolkit (NLTK)

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

## ► Idea:

- Write a pattern matching all possible tokens but matching none of non-tokens.
- Output all non-overlapping matches.

## ► Tool: regular expression (RE)

### ► Words with optional internal hyphens:

- Pattern in RE: `\w+(-\w+)*`
- Strings accepted:
  - `That`, `poster-print`, `costs`
- Strings rejected:
  - `U.S.A.`, `$12.40`, `...`
  - `non-`, `-ly`, `AT&T`

`\w` = any word character  
= `[a-zA-Z0-9_]`

`+` = match 1 or more times

`*` = match 0 or more times

`(...)` is a group (followed by `+` or `*`)

# Tokenization in Natural Language Toolkit (NLTK)

Bird, Loper and Klein (2009), *Natural Language Processing with Python*. O'Reilly

## ▶ Abbreviations:

- ▶ Pattern: `([A-Z]\.)+`
- ▶ Strings accepted: `U.S.A.`
- ▶ String rejected: `UU.S.A`, `I`

## ▶ Currency and percentages:

- ▶ Pattern: `\$?\d+(\.\d+)?%`
- ▶ String accepted: `$12.40`
- ▶ String rejected: `$.4`, `1.4.0`, `1%%`

## ▶ Ellipsis:

- ▶ Pattern: `\.\.\.`
- ▶ String accepted: `...`

## ▶ Separate tokens:

- ▶ Pattern: `[.,;"'()?:_`]`

`[A-Z]` = uppercase char

`\d` = digit = `[0-9]`

*In RE, some characters (e.g., `^$.?+*()[]`) have special meaning. They should be escaped to match them.*

`\.` = `"."`

`\$` = `"$"`

`+` = match 1 or more times

`?` = match 0 or 1 time



# Tokenization in languages without spaces

---

- ▶ Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!
- ▶ Same for compound nouns in some western languages
  - ▶ Example: German:  
**Freundschaftsbezeugungen** (demonstration of friendship)
  - ▶ Example: hashtags in social media:  
**#TrueLoveInFourWords**



# How to do word tokenization in Chinese?

---

▶ Sentence: 姚明进入总决赛

▶ 姚明/进入/总决赛

▶ 姚/明/进入/总/决赛

▶ 姚/明/进/入/总/决/赛

} Typically cast as **sequence labeling + supervised learning**  
(to be discussed later)

— single-character segmentation



# Subword tokenization

---

- ▶ Tokens can be parts of words as well as whole words
  - ▶ Use the data to tell us how to tokenize
- ▶ Three common algorithms:
  - ▶ **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
  - ▶ **Unigram language modeling tokenization** (Kudo, 2018)
  - ▶ **WordPiece** (Schuster and Nakajima, 2012)
- ▶ All have 2 parts:
  - ▶ A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
  - ▶ A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary



# Byte Pair Encoding (BPE) token learner

---

- ▶ Let vocabulary be the set of all individual characters  
= {A, B, C, D,..., a, b, c, d....}
- ▶ Repeat:
  - ▶ Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
  - ▶ Add a new merged symbol 'AB' to the vocabulary
  - ▶ Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- ▶ Until  $k$  merges have been done.



# BPE token learner

---

- ▶ Original corpus:

low low low low low lowest lowest newer newer newer newer newer  
newer wider wider wider new new

- ▶ Usually run inside space-separated tokens.

- ▶ First add end-of-word tokens “\_”

- ▶ So we can differentiate **est** in estate and smallest

- ▶ Resulting vocabulary:

**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



# BPE token learner

---

## 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

### ► Merge e r to er

## 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, er



# BPE token learner

---

## 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, e r

► Merge **er \_** to **er\_**

## 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, e r, e r\_

---



# BPE token learner

---

## 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, er, er\_

► Merge **n e** to **ne**

## corpus

5 l o w \_  
2 l o w e s t \_  
6 ne w er\_  
3 w i d er\_  
2 ne w \_

## vocabulary

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





# BPE token learner

---

## ► The next merges are:

| Merge      | Current Vocabulary   |
|------------|--|
| (ne, w)    | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new                        |
| (l, o)     | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo                    |
| (lo, w)    | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low               |
| (new, er_) | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_       |
| (low, _)   | _, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_ |



# BPE token segmenter algorithm

---

- ▶ The learned vocabulary:

\_\_, d, e, i, l, n, o, r, s, t, w, er, er\_\_, ne, new, lo, low, newer\_\_, low\_\_

- ▶ On the test data, run each merge learned from the training data:

- ▶ Greedily, in the order we learned them
- ▶ (test frequencies don't play a role)

- ▶ So: merge every **e r** to **er**, then merge **er \_** to **er\_**, etc.

- ▶ Result:

- ▶ Test set "**n e w e r \_**" would be tokenized as a full word
- ▶ Test set "**l o w e r \_**" would be two tokens: "**low er\_**"



# Properties of BPE tokens

---

- ▶ Usually include frequent words and frequent subwords
  - ▶ Which are often morphemes like **-est** or **-er**
- ▶ A **morpheme** is the smallest meaning-bearing unit of a language
  - ▶ **unbreakable** has 3 morphemes **un-**, **-break-**, and **-able**





# Word Normalization



# Word normalization

---

- ▶ Putting words/tokens in a standard format
  - ▶ U.S.A. or USA
  - ▶ uhhuh or uh-huh
  - ▶ Fed or fed
  - ▶ am, is, be, are



# Case folding

---

- ▶ Reduce all letters to lower case
- ▶ Good in some scenarios
  - ▶ Information retrieval: users tend to use lower case
  - ▶ Example:



- ▶ Not good in others
    - ▶ Example: General Motors, Fed, US
- 



# Lemmatization

---

- ▶ Represent all words as their lemma, their shared root  
= dictionary headword form
  - ▶ am, are, is → be
  - ▶ car, cars, car's, cars' → car
  - ▶ Spanish **quiero** ('I want'), **quieres** ('you want')  
→ **querer** 'want'
  - ▶ He is reading detective **stories**  
→ He **be read** detective **story**



# Lemmatization done by Morphological Parsing

---

- ▶ Morphemes:
  - ▶ The small meaningful units that make up words
  - ▶ **Stems**: The core meaning-bearing units
  - ▶ **Affixes**: Parts that adhere to stems, often with grammatical functions
- ▶ Morphological Parsers:
  - ▶ Parse *cats* into two morphemes *cat* and *s*
  - ▶ Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.
- ▶ We will talk about parsing later.





# Stemming

---

- ▶ Reduce terms to stems, **chopping off affixes crudely**
  - ▶ A simple and crude alternative to lemmatization
- ▶ Porter Stemmer
  - ▶ Based on a series of rewrite rules run in series
    - ▶ A cascade, in which output of each pass fed to next pass
  - ▶ Some sample rules

ATIONAL → ATE (e.g., relational → relate)

ING → ε if stem contains vowel (e.g., motoring → motor)

SSSES → SS (e.g., grasses → grass)



# Stemming

---

- ▶ Reduce terms to stems, **chopping off affixes crudely**
  - ▶ A simple and crude alternative to lemmatization
- ▶ Example
  - ▶ Sentence:
    - ▶ This was not the map we found in Billy Bones's chest.
  - ▶ Stemming:
    - ▶ Thi wa not the map we found in Billi Bone s chest.
  - ▶ Lemmatizing:
    - ▶ This be not the map we find in Billy Bones 's chest.





# Sentence Segmentation



# Sentence segmentation

---

- ▶ **!, ?** mostly unambiguous but period “.” is very ambiguous
  - ▶ Sentence boundary
  - ▶ Abbreviations like **Inc.** or **Dr.**
  - ▶ Numbers like **.02%** or **4.3**
- ▶ Common algorithm
  - ▶ Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.
    - ▶ An abbreviation dictionary can help
  - ▶ Sentence segmentation can then often be done by rules based on this tokenization.





# Summary



# Text Normalization

---

- ▶ Word tokenization
- ▶ Word normalization
- ▶ Sentence segmentation
- ▶ Methodology: Rules + ML

