M.EIC

Natural Language Processing

Henrique Lopes Cardoso

FEUP / LIACC

hlc@fe.up.pt

Basic Text Processing

regular expressions, words, corpora, sentence segmentation, tokenization, normalization, lemmatization

Regular Expressions

- A regular expression is a sequence of characters that define a search pattern
  - Makes use of meta-characters, such as `{} [] () ^$.|*+?\`
    - `[A-Z]` uppercase letter
    - `[a-z]` lowercase letter
    - `[0-9]` digit
    - `^` negation
    - `|` disjunction
    - `?` optional
    - `*` zero or more
    - `+` one or more
    - `.` Any
    - …

Example: find all instances of the word "**the**" in a text

- `the` misses capitalized letters
- `[tT]he` returns "other" or "theology"
- `[^a-zA-Z][tT]he[^a-zA-Z]`

Regular Expressions: False Positives and False Negatives

- False negatives: not matching strings that we should
  - `the` misses capitalized letters
- False positives: matching strings that we should not
  - `[tT]he` returns "other" or "theology"

- Reducing the error rate for an application (in NLP or otherwise) often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives)

The role of Regular Expressions

- Sophisticated sequences of regular expressions are often a first model for many text processing tasks
  - E.g. detecting named entities based on Capitalization

- For harder tasks or increased performance, we use machine learning classifiers
  - But regular expressions are still used for pre-processing, or as features in the classifiers
  - Can be very useful in capturing generalizations

Words

- What counts as a word?

He stepped out into the hall, was delighted to encounter a water brother.

  - 13 words if we don't count punctuation, 15 if we do
  - Treating '.' or ',' as words depends on the task!

- Transcribing spoken language:

I do uh main- mainly business data processing

  - Whether disfluencies are considered depends on the application

Lemmas, Wordforms, Types, and Tokens

- Dictionary-word variations
  - Capitalization: “They” vs “they”
  - Inflected forms: “cat” vs “cats”, “buy vs “bought”, “ir” vs “vou”

- Lemma: same stem, part-of-speech and word sense – “cat”, “buy”, “ir”
- Wordform: full inflected form of the word – “cat” and “cats”, “buy” and “bought”, “ir” and “vou”

- Types: distinct words in the vocabulary used in a corpus (set of documents)
- Token: an instance of a type in running text

Types and Tokens

They picnicked by the pool, then lay back on the grass and looked at the stars.

- 16 tokens,14 types (ignoring punctuation)

- Types and tokens in corpora
  - Herdan's or Heaps' Law: $|V| = kN^{\beta},\ k > 0,\ 0 < \beta < 1$

Corpus	**Tokens = N**	**Types = $\|V\|$**
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google n-grams	1 trillion	13 million

Shakespeare has a very broad vocabulary!

only types appearing 40 or more times

Corpora

- Text production: specific speakers or writers, specific dialect-language, specific time/place, for a specific function
- Languages: there are ~7097 languages in the world!
- Variants: European Portuguese, Brazilian Portuguese, African dialects
- Code switching: *Chuva outra vez? What the hell... Já devia ser summertime!*
- Genre: newswire, fiction, scientific articles, Wikipedia, religious texts, user-generated content, transcripts of spoken language, …
- Author demographics: age, gender, race, socioeconomic class, …
- Time: language changes over time, historical texts

Sentence Segmentation

- Splitting a text into sentences
  - Typically based on punctuation marks
  - But the period '.' is particularly ambiguous (e.g. Mr., Ph.D., Inc., Sr., …)
  - Decide (learn) whether a period is part of the word or is a sentence-boundary marker
    - Abbreviation dictionary can help determine whether the period is part of a commonly used abbreviation

```
from nltk.tokenize import sent_tokenize
text = "Hello. Are you Mr. Smith? Just to let you know that I have finished my M.Sc. and Ph.D. on AI. I loved it!"
print(sent_tokenize(text))
```

```
['Hello.', 'Are you Mr. Smith?', 'Just to let you know that I have finished my M.Sc.', 'and Ph.D. on AI.', 'I loved it!']
```

Text Normalization

- Converting text to a more convenient, standard form
- Tokenization: segmenting words in a text
- Word normalization
  - Case folding
  - Lemmatization
  - Stemming

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

LIACC

Word Tokenization

- Initial approach: look for spaces, punctuation and other special characters
- What about:
  - Ph.D., AT&T, can't, we're, state-of-the-art, guarda-chuva
  - $45.55, 123,456.78, 123.456,78
  - 07/04/2020, April 4, 2020
  - http://www.fe.up.pt, hlc@fe.up.pt, #iart
  - New York, Vila Nova de Gaia
- Certain languages do not have space splitting!
  - Chinese, Japanese, some words in German, …

```python
import nltk
from nltk import word_tokenize

text = 'That U.S.A. poster-print costs $12.40...'
tokens = word_tokenize(text)
print(len(tokens))
print(tokens)
```

```
7
['That', 'U.S.A.', 'poster-print', 'costs', '$', '12.40', '...']
```

```python
word_tokenize("I don't think we're flying today.")
```

```
['I', 'do', "n't", 'think', 'we', "'re", 'flying', 'today', '.']
```

Sub-word Tokens

- What if we tokenize by word pieces?
- Advantages:
  - Dealing with unknown words (particularly relevant for Machine Learning)
    - E.g. training corpus containing "low" and "lowest", but not "lower", which appears in the test corpus
  - Robustness to misspellings
  - Dealing with multi-lingual data
- Byte-Pair Encoding (BPE) [Sennrich et al., 2016] (used, for instance, in RoBERTa and GPT)
- Unigram language model [Kudo, 2018]
- WordPiece [Schuster and Nakajima, 2012] (used, for instance, in BERT)
  - Given the token "intention" and the dictionary ["in", "tent", "intent", "##tent", "##tention", "##tion", "#ion"], obtains the tokens ["intent", "##ion"]

Byte-Pair Encoding (BPE)

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

```
V ← all unique characters in C            # initial set of tokens is characters
for i = 1 to k do                          # merge tokens til k times
    t_L, t_R ← Most frequent pair of adjacent tokens in C
    t_NEW ← t_L + t_R                      # make new token by concatenating
    V ← V + t_NEW                          # update the vocabulary
    Replace each occurrence of t_L, t_R in C with t_NEW    # and update the corpus
return V
```

- BPE is usually run with many thousands of merges on a very large input corpus
  - most words will be represented as full symbols
  - very rare words (and unknown words) will be represented by their parts

Byte-Pair Encoding (BPE)

- A very small corpus:

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

- Most frequent adjacent “tokens”: e and r (or r _)

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

- er and _

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

- n and e

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2	l o w e s t _	
6	ne w er_	
3	w i d er_	
2	ne w _	

- …and so on:

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_

Multi-word Expressions (MWE)

- New York, Futebol Clube do Porto
- Simple approach: MWE dictionary

```
word_tokenize("Good muffins cost $3.88\nin New York.")
```

```
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.']
```

```
from nltk.tokenize import MWETokenizer
from nltk import sent_tokenize, word_tokenize

s = "Good muffins cost $3.88\nin New York."
mwe = MWETokenizer([('New', 'York'), ('Hong', 'Kong')], separator=' ')

[mwe.tokenize(word_tokenize(sent)) for sent in sent_tokenize(s)]
```

```
[['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New York', '.']]
```

- Statistical approach: detect frequently used n-grams and stick them together

- Tokenization of MWE is tied up with named entity recognition

Word Normalization

- Putting words/tokens in a standard format
  - Reduces the vocabulary size
  - Helps Machine Learning models to generalize
- Case folding
  - Putting every word in lower case
  - Not always helpful, and thus not always performed
    - Sentiment analysis: uppercase might denote anger, …
    - Part-of-speech or named-entity tagging: US/us, Mike Pence/mike pence, …

Lemmatization

- Determining the root of the word: many words have the same root!
  - “am”, “are”, “is” → “be”
  - “He is reading detective stories” → “He be read detective story”

- Morphological parsing: words are built from morphemes
  - Stem: the central morpheme of a word, supplying the main meaning
  - Affix: adding additional meaning
  - *cats* = *cat* (stem) + *s* (affix)
  - *iremos* = *ir* (stem) + 1st plural + future tense (morphological features)

Stemming

- Lemmatization algorithms can be complex
- Stemming: a simpler and cruder method that simply cuts off word final affixes
    - Subject to over- and under-generalization
- The Porter stemmer [Porter, 1980]
    - Set of rules run in series
        - ATIONAL → ATE (e.g., relational → relate)
        - ING → ε if stem contains vowel (e.g., motoring → motor)
        - SSES → SS (e.g., grasses → grass)

The European Commission has funded a numerical study to analyze the purchase of a pipe organ with no noise for Europe's organization. Numerous donations have followed the analysis after a noisy debate.

⇩

the european commiss ha fund a numer studi to analyz the purchas of a pipe organ with no nois for europ 's organ . numer donat have follow the analysi after a noisi debat .

The Python Notebook

jupyter

preprocessing_.ipynb

- Tokenization: regular expressions, NLTK
- Dealing with multi-word expressions
- Stemming
- Lemmatization
- spaCy language processing pipelines

NLTK
https://www.nltk.org/

spaCy
https://spacy.io/