Information Retrieval

Amin Nazari

Spring 2025

Chapter 2

Text preprocessing

Recall

- Documents unit and document parsing is a challenge.
- Tokenization is a challenge.
- A large corpus is a challenge.
- Phrase retrieval
- Deterministic vs Probabilistic(Ranking)

Outline

- Documents
- Tokenize
  - Words
  - Sub-word
- Normalization
  - Whitespace
  - Accents
  - Stop words
  - Lemmatization
  - Stemming
  - Emoji
  - Spelling Correction

Documents unit

- What is the document unit for indexing?
  - A book?
  - A file?
  - An email?
  - An email with 5 attachments?
  - A group of files (ppt or latex in HTML)?

Document parsing

- What format is it in? pdf, word, excel, html, etc.
- What language is it in?
- What character set is in use?
- Example: Multilingual search engines (yahoo vs google)

Tokenize

- Definitions
  - Word – A delimited string of characters as it appears in the text.
  - Term – A “normalized” word (case, morphology, spelling, etc); an equivalence class of words.
  - Token – An instance of a word or term occurring in a document.
  - Type – The same as a term in most cases: an equivalence class of tokens.
- Word Tokenization
- Sub-word Tokenization

Word Tokenization

Input: "Tokenization is an important NLP task."
Output: ["Tokenization", "is", "an", "important", "NLP", "task", "."]

Problems?

- State-of-the-art
- co-education
- San Francisco
- Los Angeles

Sub-word Tokenization(Byte Pair Encoding)

1. Initialize the vocabulary with all the bytes or characters in the text corpus
2. Calculate the frequency of each byte or character in the text corpus.
3. Repeat the following steps until the desired vocabulary size is reached:
    1. Find the most frequent pair of consecutive bytes or characters in the text corpus
    2. Merge the pair to create a new sub-word unit.
    3. Update the frequency counts of all the bytes or characters that contain the merged pair.
    4. Add the new sub-word unit to the vocabulary.
4. Represent the text corpus using the sub-word units in the vocabulary.

Byte Pair Encoding(Example)

Suppose we have four words: “ab”, “bc”, “bcd”, and “cde”.

Step 1: Vocabulary = {"a", "b", "c", "d", "e"}

Step 2: Frequency = {"a": 1, "b": 3, "c": 3, "d": 2, "e": 1}

Step 3a: "bc": 2.

- **Step 3b**: Merge "bc" to create a new sub-word unit "bc".
- **Step 3c**: Frequency = {"a": 1, "b": 2, "c": 3, "d": 2, "e": 1, "bc": 2}
- **Step 3d**: Vocabulary = {"a", "b", "c", "d", "e", "bc"}

Repeat steps 3a-3d until the desired vocabulary size is reached.

Step 4: Represent the text corpus using sub-word units

{"a", "b", "c", "d", "e", "bc", "cd", "de","ab","bcd","cde"}.

Byte Pair Encoding(Quiz)

- “low lower lowest”

Normalization

- We need to “normalize” terms in indexed text and query terms in the same form.
- The same equivalence class
  - USA → U.S.A., USA
  - window → window, windows
  - windows → Windows, windows
- Why don’t you want to put window, Windows, windows, and Windows in the same equivalence class?

Normalization

- Normalization and language detection interact.
  - PETER WILL NICHT MIT. → MIT = mit
  - He got his PhD from MIT. → MIT ≠ mit
- Numbers
  - 3/20/91
  - 20/3/91
  - Mar 20, 1991
  - B-52
  - 100.2.86.144
  - (800) 234-2333
  - 800.234.2333

Normalization

- Whitespace
- Accents
  - naïve vs naive
- Stop words
  - stop words = extremely common words that would appear to be of little value in helping select documents matching a user‘s need.
  - Examples: *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*
  - Stop word elimination used to be standard in older IR systems.
  - But you need “stop words” for phrase queries, e.g. “King of Denmark”
  - Most web search engines index stop words.

Normalization

- Lemmatization (Reduce inflectional/variant forms to base form): Headword
  - Example: *am, are, is* → *be*
  - Example: *car, cars, car's, cars'* → *car*
  - Inflectional morphology (*cutting* → *cut)* vs. derivational morphology (*destruction* → *destroy)*
- Stemming
  - The crude heuristic process truncates words in an attempt to achieve principled lemmatization.
  - Example for derivational: automate, automatic, automation all reduce to "automat"

Stemming vs lemmatization

- Stemming and lemmatization are both NLP techniques used to reduce words to their base forms. Stemming removes suffixes to derive roots, which may not be actual words (pseudo-stems), while lemmatization converts words to their linguistically valid base forms (lemmas). Lemmatization generally relies on a dictionary and linguistic analysis, making it more precise but slower compared to stemming.

Word	Stemming	Lemmatization
information	inform	information
informative	inform	informative
computers	comput	computer
feet	feet	foot

Porter stemmer algorithm

Step 1a

1. SSES → SS
2. IES → I
3. SS → SS
4. S →

Step 1b

1. (m>0) EED → EE
2. (*v*) ED →
3. (*v*) ING →

If the second or third of the rules in Step 1b is successful, the following is performed.

1. AT → ATE
2. BL → BLE
3. IZ → IZE
4. (*d and not (*L or *S or *Z)) → single letter
5. (m=1 and *o) → E

Step 1c

1. (*v*) Y → I

Step 2

1. (m>0) ATIONAL → ATE
2. (m>0) TIONAL → TION
3. (m>0) ENCI → ENCE
4. (m>0) ANCI → ANCE
5. (m>0) IZER → IZE
6. (m>0) ABLI → ABLE
7. (m>0) ALLI → AL
8. (m>0) ENTLI → ENT
9. (m>0) ELI → E
10. (m>0) OUSLI → OUS
11. (m>0) IZATION → IZE
12. (m>0) ATION → ATE
13. (m>0) ATOR → ATE
14. (m>0) ALISM → AL
15. (m>0) IVENESS → IVE
16. (m>0) FULNESS → FUL
17. (m>0) OUSNESS → OUS
18. (m>0) ALITI → AL
19. (m>0) IVITI → IVE
20. (m>0) BILITI → BLE

Porter stemmer algorithm

Step 3

1. (m>0) ICATE → IC
2. (m>0) ATIVE →
3. (m>0) ALIZE → AL
4. (m>0) ICITI → IC
5. (m>0) ICAL → IC
6. (m>0) FUL →
7. (m>0) NESS →

Step 4

1. (m>1) AL →
2. (m>1) ANCE →
3. (m>1) ENCE →
4. (m>1) ER →
5. (m>1) IC →
6. (m>1) ABLE →
7. (m>1) IBLE →
8. (m>1) ANT →
9. (m>1) EMENT →
10. (m>1) MENT →
11. (m>1) ENT →
12. (m>1 and (*S or *T)) ION →
13. (m>1) OU →
14. (m>1) ISM →
15. (m>1) ATE →
16. (m>1) ITI →
17. (m>1) OUS →
18. (m>1) IVE →
19. (m>1) IZE →

Step 5a

1. (m>1) E →
2. (m=1 and not *o) E →

Step 5b

1. (m > 1 and *d and *L) → single letter

Does stemming improve effectiveness?

Emoji replacement

Emotion	Emoji	Description	Score
Negativity	⚖️	Balance scale	1.03
	😡	Pouting face	1.01
	🤬	Face with symbols on mouth	0.99
Anger	⚖️	Balance scale	0.68
	🤬	Face with symbols on mouth	0.56
	😡	Pouting face	0.55
Disgust	🤮	Face vomiting	0.49
	🤢	Nauseated face	0.48
	👿	Angry face with horns	0.45
Fear	⚖️	Balance scale	0.68
	✝️	Latin cross	0.63
	⚠️	Warning	0.57
Sadness	⚖️	Balance scale	0.55
	⚠️	Warning	0.49
	😡	Pouting face	0.48
Positivity	🎂	Birthday cake	2.80
	🎈	Balloon	2.15
	🎁	Wrapped gift	2.10
Anticipation	🎂	Birthday cake	2.11
	🎈	Balloon	1.59
	🎁	Wrapped gift	1.51
Joy	🎂	Birthday cake	2.39
	🎈	Balloon	1.81
	🎁	Wrapped gift	1.59
Surprise	🎂	Birthday cake	1.12
	🎁	Wrapped gift	0.90
	🍀	Four leaf clover	0.86
Trust	✝️	Latin cross	1.55
	⚖️	Balance scale	1.39
	🎂	Birthday cake	1.36

Regular Expressions

1. Basics:

- `.` → Matches any ch
- `^` → Matches the sta
- `$` → Matches the en
- `*` → Matches 0 or m
- `+` → Matches 1 or m
- `?` → Matches 0 or 1
- `{n}` → Matches exac
- `{n,}` → Matches `n`
- `{n,m}` → Matches be

2. Character Classes:

- `[abc]` → Matches `a`, `b`, or `c`
- `[^abc]` → Matches any character except `a`, `b`, or `c`
- `[a-z]` → Matches any lowercase letter
- `[A-Z]` → Matches any uppercase letter
- `[0-9]` → Matches any digit
- `\d` → Matches any digit (equivalent to `[0-9]`)
- `\D` → Matches any non-digit character
- `\w` → Matches any word character (alphanumeric + underscore)
- `\W` → Matches any non-word character
- `\s` → Matches any whitespace character
- `\S` → Matches any non-whitespace character

Regular Expressions

3. Anchors:

- `^` → Matches the start of a string
- `$` → Matches the end of a string
- `\b` → Matches a word boundary
- `\B` → Matches a non-word boundary

4. Groups and Alternation:

- `(abc)` → Matches the exact sequence `abc`
- `|` → Acts as an OR operator (e.g., `a|b` matches `a` or `b`)
- `(?:...)` → Non-capturing group

5. Escape Sequences:

- `\.` → Matches a literal `.`
- `\*` → Matches a literal `*`
- `\n` → Matches a newline
- `\t` → Matches a tab

Regular Expressions

- Match an Email Address
  - ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
- Match a URL
  - ^https?:\/\/[^\s]+$
- Match a Date (YYYY-MM-DD)
  - ^\d{4}-\d{2}-\d{2}$

Regular Expressions(Quiz)

- Match a Hex Color Code
- Match a Time (24-Hour Format)
- Match HTML Tags

Python

- Text Cleaning (remove white spaces, specific characters, numbers, URL, …)
- Tokenization (Words)
- Stop Words Removal
- Stemming and Lemmatization
- Handling emoji's and Emoticons
- Spell Checking

Dataset

- SemEval
- Kaggle
- UCI

Summary

- One of the foundational steps in NLP is text preprocessing, which involves cleaning and preparing raw text data for further analysis or model training.
- Proper text preprocessing can significantly impact the performance and accuracy of NLP models.
- This chapter delve into the essential steps involved in text preprocessing for NLP tasks.