Type, Token, and Normalization

27 November 2024 17:49

Stop words are function words.

↓

words of little lexical meaning
BUT
are impt. to sentence structure

* ~ func^n word = content word.

Type vs token

Type → unique word

token → all words

lexical richness = avg. count of how many times each unique word appears in text.

↓

1/LR = type token ratio → measure of vocabulary variation

↓

TTR = type count / token count.

Tokenization → "process — segmenting (sentence segmentn) — string (char) → words"

→ starting + ending pt?
Binary classifier — simple decision tree.

→ Also handle: a) handling hyphens
b) named entities
c) word segmentation (greedy algo) — match words w/ lexicon.

But why normalization for tokeniz^n?

Same text, many ways → USA, US, U.S.A, United Stat... America

Indexed text + query items MUST have same form
→ If not: failed retrieval.

Equivalence classes

Case folding → all in lower case

↓

Normalization strategies:

A. Lemmatization
- variant → base form
- dictionary head word req.

} use morphology.

B. Stemming
- rule based — not dict. dependent
- remove suffixes (crudely)
- language dependent.

} set of 'if-then-else'

- doesn't always produce words.

↓

Algorithmic Stemmers

Sa:

Algorithmic Stemmers

+ fast + lean
+ more efficient to not use dict.
+ ignore irregular forms rather than complicating algo.

Eg:

I/P	Strip 'ed'	Repair
Hoped	hop	hope (add e if short)
Hopped	hope	hop (delete one if doubled)

What if we've spelling errors?

Minimum edit distance – Levenshtein. Nuh dog.

→ Valid changes?
- insert char
- delete char
- change char.
- transpose.

Usual normalization flow:

1. Convert to lowercase
2. Remove numb. + whitespace (regex)
3. Tokenization
4. Remove stopwords.
5. Stemming / lemmatization
6. POS tagging
7. Chunking / shallow parsing
8. Named entity recog.
9. Coreference resolution.