

Tokenization, term, stem

UNIT-1

Initial stages of text processing

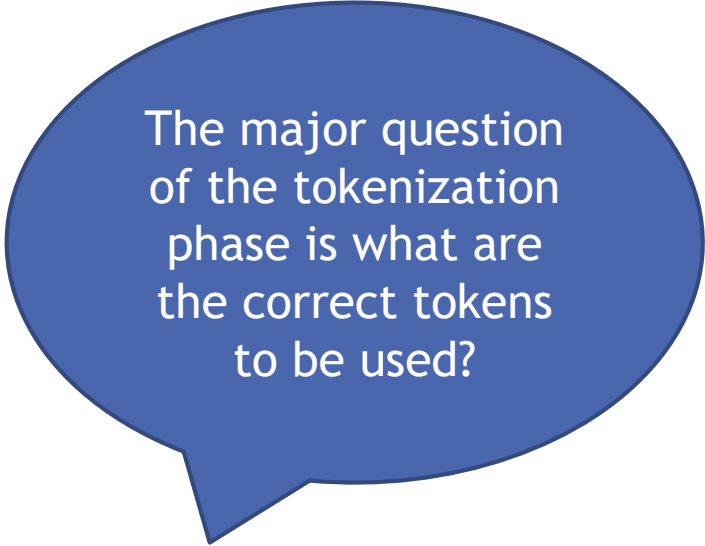
- ▶ Tokenization
 - ▶ Cut character sequence into word tokens
 - ▶ Deal with *“John’s”, a state-of-the-art solution*
- ▶ Normalization
 - ▶ Map text and query term to same form
 - ▶ You want *U.S.A.* and *USA* to match
- ▶ Stemming
 - ▶ We may wish different forms of a root to match
 - ▶ *authorize, authorization*
- ▶ Stop words
 - ▶ We may omit very common words (or not)
 - ▶ *the, a, to, of*

Tokenization

- ▶ Tokenization is the task of chopping up text into pieces, called tokens and same time ignoring certain characters.
 - ▶ Document: Are you able to login to system
 - ▶ Token: are, you, able, to, login, to, system (Total:07 Tokens)
 - ▶ Type: are, you, able, to, login, system (Total: 06 Types)
 - ▶ Terms: are, you, able, login, system (Total: 05 Terms)
- ▶ These tokens are often loosely referred to as terms or words, but it is sometimes important to make a type/token distinction.

Issues with Tokenization

- ▶ It looks fairly trivial: you chop on whitespace and throw away punctuation characters.



The major question of the tokenization phase is what are the correct tokens to be used?

Example: apostrophe

- Document: Mr. O' Neill thinks that the boy's stories about Chile's capital aren't amusing.

Neill	
ONeill	
O'	Neill
O	Neill

Date Formates:
2/6/2024
June/2/2024

White Spaces
1. Los angeles
2. San francisco

Co-education
Long-term
Lowercase?

Tokenization Challenges

- ▶ **Ambiguity.** Language is inherently ambiguous. Consider the sentence
 - ▶ "Flying planes can be dangerous." Depending on how it's tokenized and interpreted, it could mean that the act of piloting planes is risky or that planes in flight pose a danger.
 - ▶ Such ambiguities can lead to vastly different interpretations.
- ▶ **Languages without clear boundaries.**
 - ▶ Some languages, like Chinese or Japanese, don't have clear spaces between words, making tokenization a more complex task.
 - ▶ Determining where one-word ends and another begins can be a significant challenge in such languages.
- ▶ **Handling special characters.**
 - ▶ Texts often contain more than just words.
 - ▶ Email addresses, URLs, or special symbols can be tricky to tokenize. For instance, should "john.doe@email.com" be treated as a single token or split at the period or the "@" symbol?

Implementation of Tokenization

```
# download all the essential files and packages
import nltk
nltk.download('all')
```

```
# Sentence Tokenization : text into sentence tokenization
```

```
from nltk.tokenize import sent_tokenize
```

```
text = ""Everything we're doing is about going forward," Phoebe Philo told Vogue in 2009, shortly before showing her debut Resort collection for Céline. Although the label had garnered headlines when it was revived by Michael Kors in the late '90s, it was Philo who truly brought the till then somewhat somnambulant luxury house to the forefront. Critics credited her with pushing fashion in a new direction, toward a more spare, stripped-down kind of sophistication. What Céline now offered women was, as the magazine put it, "a grown-up and hip way to put themselves together.""
```

```
sent_tokenize(text)
```

```
# words tokenization : text into words
from nltk.tokenize import word_tokenize
word_tokens = word_tokenize(text)
print(word_tokens)
```

Implementing tokenization from scratch using python list operations

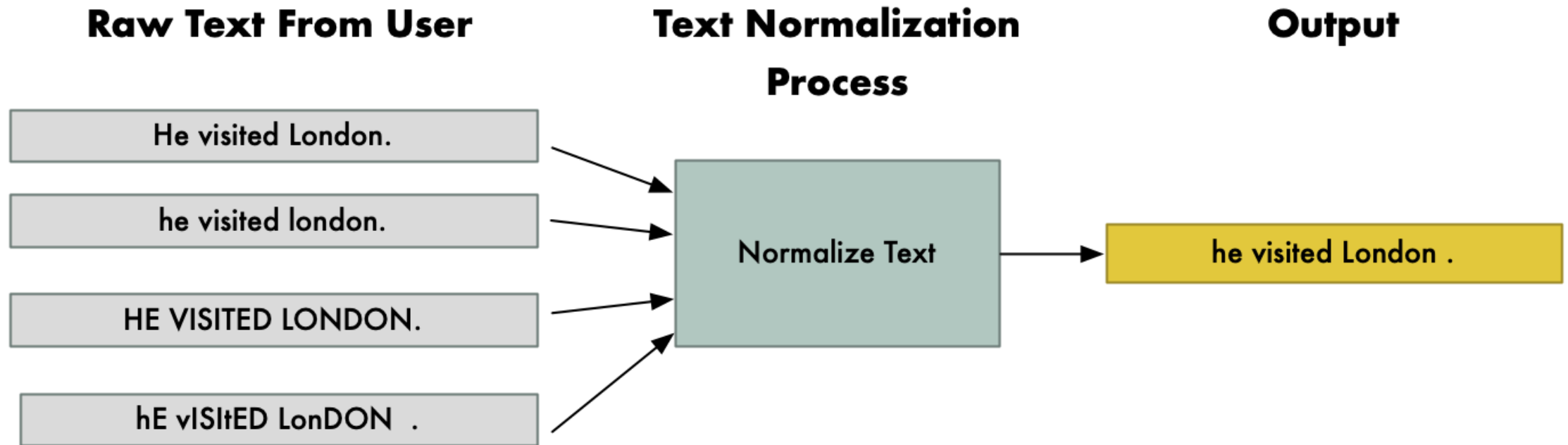
```
sentence_tokens = text.split('.')  
sentence_tokens
```

```
word_tokens = text.split(' ')  
print(word_tokens)
```


Text Normalization

- ▶ Text normalization is the process of transforming text into a single canonical (standard) form that it might not have had before.
 - ▶ For example, the word "gooooood" and "gud" can be transformed to "good", its canonical form
- ▶ Text normalization reduces variations in word forms to a common form when the variations mean the same thing.
 - ▶ For example, US and U.S.A become USA
 - ▶ Product, product and products become product
 - ▶ \$400 becomes 400 dollars
- ▶ Normalization is helpful in reducing the number of unique tokens present in the text, removing the variations in a text. and also cleaning the text by removing redundant information.

Text Normalization



Text Normalization

- ▶ Some of the common techniques for text normalization are :
 1. stop word removal
 2. white space removal
 3. morphological analysis
 4. word stemming (Porter Algorithm)
 5. case folding
 6. lemmatization
 7. Word statistics (Zipf's law, Heaps' Law)

Stop word removal

- ▶ Stop words are basically a set of commonly used words in any language, not just English.
- ▶ This increases both performance (fewer terms in your dictionary) and more relevant search results.
- ▶ Example: **how** **to** develop information retrieval applications”?

Implement of stop words removal - using NLTK

```
# fetching all the stop word of english language using NLTK library  
from nltk.corpus import stopwords
```

```
stop_words = set(stopwords.words('english'))  
len(stop_words)
```

```
list(stop_words)[:10]
```

Now, filtering stop words from word tokens.

```
filtered_tokens = [x for x in word_tokens if x not in stop_words]  
  
Print(filtered_tokens)
```

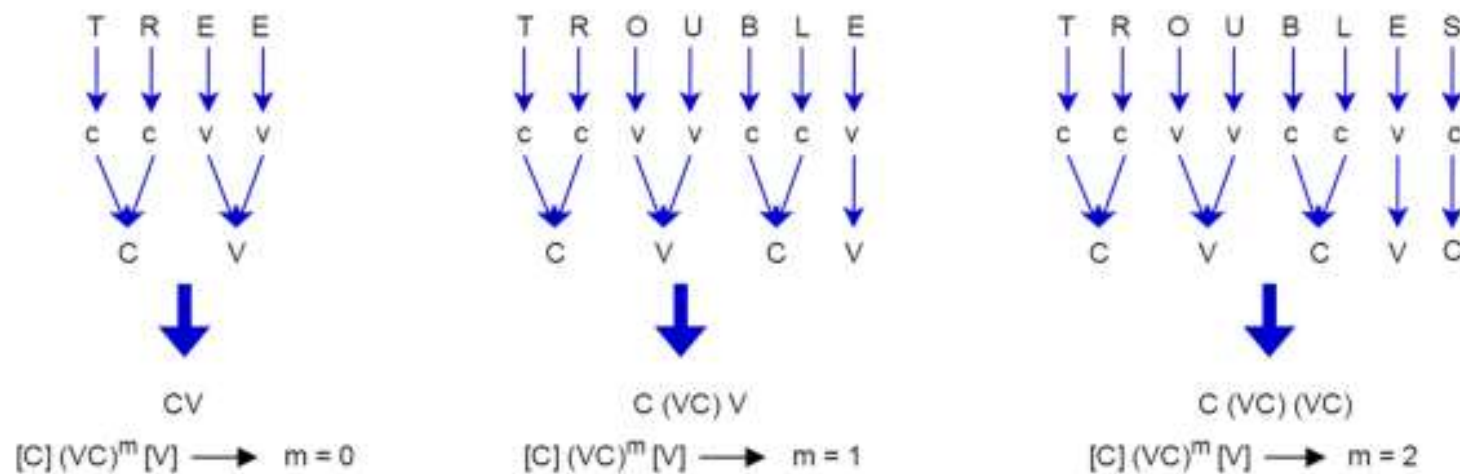
Now, let's see how many stop words are there in the word tokens.

```
len(word_tokens) - len(filtered_tokens)
```

Word Stemming

- ▶ Stemming is the process of reducing inflected words to their word stem, base or root form –generally a written word form.
- ▶ A stemming algorithm reduces the words “history”, “histories” to the word stem, “histori” and “retrived”, “retrive”, “retriving” reduce to the stem “retriv”.
- ▶ Stemming is desirable as it may reduce redundancy as most of the time the word stem and their inflected/derived words mean the same.
- ▶ Errors in stemming :
 - ▶ Over-stemming : false - positive
 - ▶ when two words are stemmed to same root that are of different stems.
 - ▶ Under-stemming : false - negative
 - ▶ when two words are stemmed from the same root that are not of different stems.

Porter stemming Algorithm



Porter Stemming Algorithm

SS	→	SS	(m>0) ATIONAL	→	ATE
IES	→	I	(m>0) TIONAL	→	TION
SS	→	SS	(m>0) ENCI	→	ENCE
S	→		(m>0) ANCI	→	ANCE

Porter stemming Algorithm

- ▶ Python Implementation
- ▶ `from nltk.stem import WordNetLemmatizer`
- ▶ `from nltk.stem import PorterStemmer`
- ▶ `porter = PorterStemmer()`
- ▶ `lem = WordNetLemmatizer()`
- ▶ `porter.stem("dancing")`
- ▶ `lem.lemmatize("dancing")`
- ▶ `lem.lemmatize("dancing", pos='a')` *# parameter pos stands for part of speech and the value a denotes adjective*
- ▶ `lem.lemmatize("dancing", pos='v')` *# v denotes verb*

Zipf's Law

- ▶ The law was originally proposed by American linguist George Kingsley Zipf (1902-50) for the frequency of usage of different words in the English language.
- ▶ Zipf's Law states that the relative frequency of a word is inversely proportional to its rank in the frequency table.

Zipf Law

▶ Mathematically

- ▶ $P_x \approx c / r(x)$
- ▶ Alternatively , rank of word times its probability (relative frequency) is approximately a constant.
- ▶ $P_x * r(x) \approx c$
- ▶ let , r_1 be the rank of a word x_1 , similarly, r_2 be the rank of x_2 , r_3 be the rank of x_3 ... and so on.
- ▶ Let, $p(x)$ denotes the relative frequency of a word x . Then according to zipf's law.
- ▶ $p(x_1) = c / r_1 = c / 1 = c$
- ▶ $p(x_2) = c / r_2 = c / 2 = c/2$
- ▶ $p(x_3) = c / r_3 = c/3 = c/3$
- ▶ where, c is a constant.
- ▶ Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.

Zipf's LAW

	Number of occurrences	% of Total Occurrences
the/The	8,543,794	6.8
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1.0
that	1,066,503	0.8
said	1,027,713	0.8

