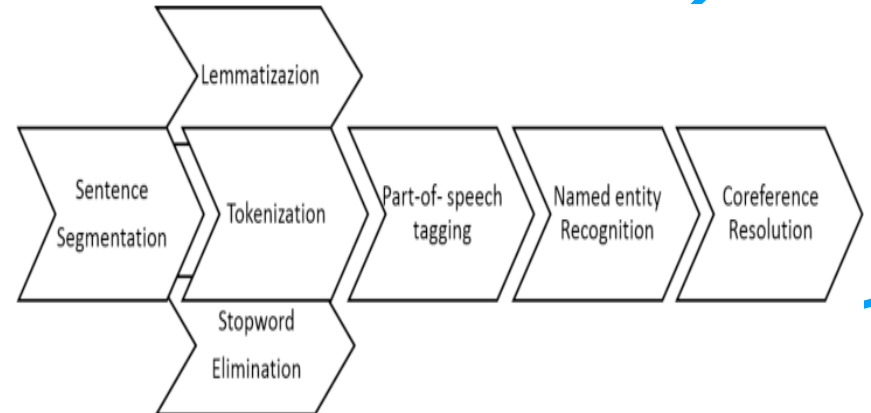




TEXT PREPROCESSING

THEORY & APPLICATIONS OF NLP

WOA7013

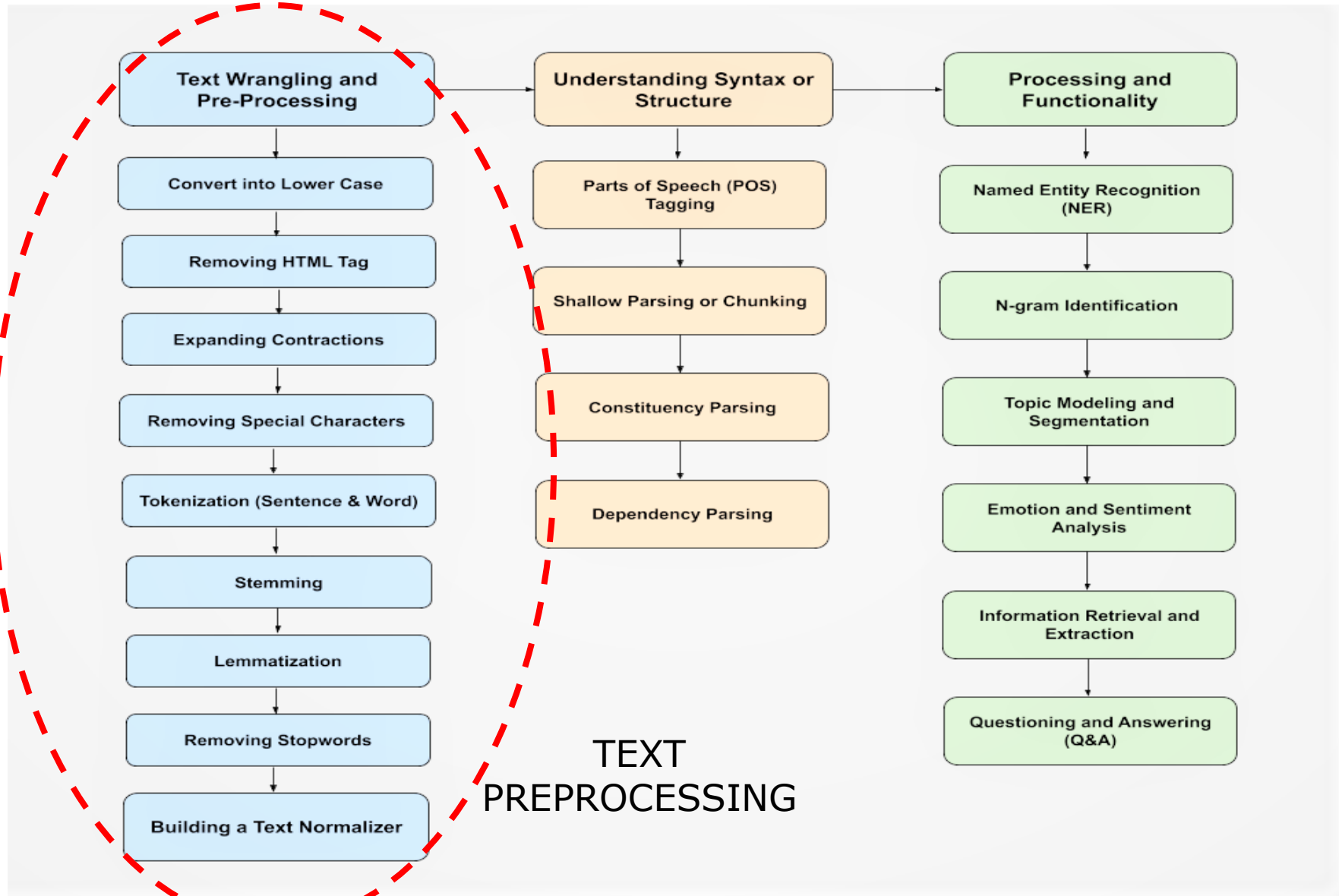


LANGUAGE AND TEXT

- Has been present since early days of human civilization
- 21st Century:
 - So Much Text!
 - Problem: Information overload!
 - Unstructured Text!
 - Problem: Noisy text!



NLP PIPELINE



NLP Tasks

Preprocessing Tasks

- Raw text is formatted to make subsequent NLP tasks easier
- Stemming

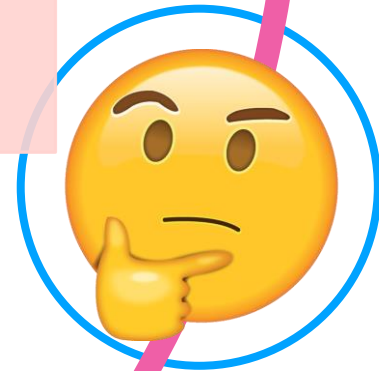
Low level Tasks

- Keyword extraction using simple rule-based methods / statistical methods
- POS Tagging

High Level Tasks

- Content analysis
- Sentiment Analysis

Discuss more examples of each NLP task.





INTRODUCTION

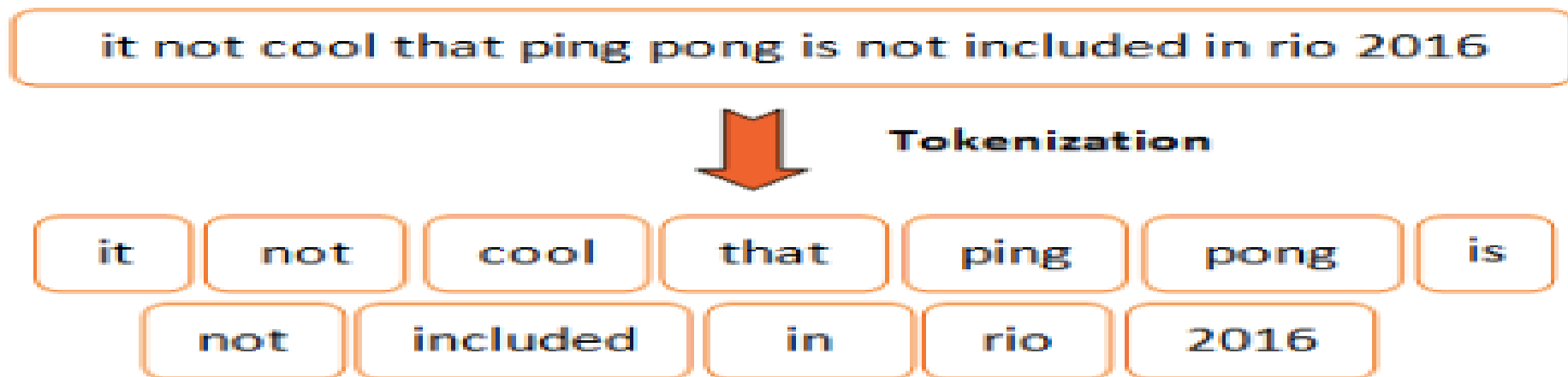
- Text preprocessing is traditionally an important step for NLP tasks, however it is a severely overlooked step in NLP applications.
- Text preprocessing if done correctly can help to increase the accuracy of the NLP tasks.
- Text preprocessing comprises a few steps/processes to transform the text into specific format based on different task or application.

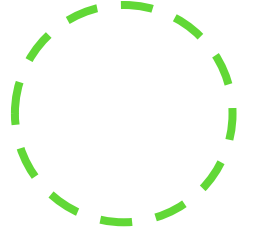


○



1. TOKENIZATION

- All natural language is based on tokens, and complex natural language is built up from more basic tokens.
- Tokenization is a process of breaking up a stream of text into words, phrases or symbols and other meaningful elements called tokens.
- For example, a sentence into words.





Tokenization is about splitting strings of text into smaller pieces, or “tokens”. Paragraphs can be tokenized into sentences and sentences can be tokenized into words.

Sentence Segmentation

- Text is segmented into sentences.
- In many languages, such as English, punctuation, especially the full stop/period, exclamation and question marks can be used to identify the end of the sentence.

Word Tokenization

- Sentence is break up into words.

Input:`import nltk`

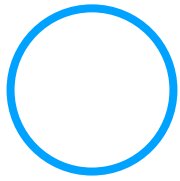
`from nltk.tokenize import sent_tokenize`

`from nltk.tokenize import word_tokenize`

`para = "Heavy rain over the last few days has caused some local flooding. Temperatures are expected to drop towards the end of the week causing the rain to turn to snow."`

`sentences = sent_tokenize(para)`

`print(sentences)`Output: ['Heavy rain over the last few days has caused some local flooding.', 'Temperatures are expected to drop towards the end of the week causing the rain to turn to snow.']





EXAMPLES

Word tokenize

```
Slide Type -
from nltk.tokenize import word_tokenize

words = word_tokenize('Nice book! Though it is lack of advanced topics.
                      it's still good for beginners.').lower()

print(words)

['nice', 'book', '!', 'though', 'it', 'is', 'lack', 'of', 'advanced', 'top
ics', '.', 'it's', 'still', 'good', 'for', 'beginners', '.']
```

```
Slide Type -
sum( sentiment_dictionary.get(word, 0) for word in words )
```

3



Sentence Segmentation

Input: `import nltk`

`from nltk.tokenize import sent_tokenize`

`from nltk.tokenize import word_tokenize`

`para = "Heavy rain over the last few days has caused some local flooding. Temperatures are expected to drop towards the end of the week causing the rain to turn to snow."`

`sentences = sent_tokenize(para)`

`print(sentences)`

Output: `['Heavy rain over the last few days has caused some local flooding.', 'Temperatures are expected to drop towards the end of the week causing the rain to turn to snow.']`

E.g. Given a sentence: *They lay back on the grass and looked at the stars*



How many words?

11

How many tokens?

11

How many types?

10

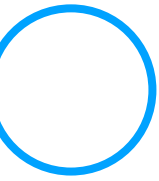
Word can be defined as:

Type: an element of the vocabulary

- *Lemma* : same stem, part of speech
- *cat and cats = same lemma*

Token: An instance of that type in running text

Cat and cats = different wordform





ISSUES IN TOKENIZATION

1. Punctuation marks:



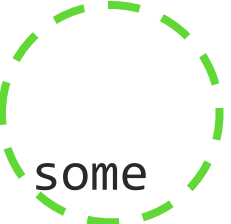
- period (.) can be used in abbreviations, such as Mr., PhD. where period does not signify the end of a sentence.
- *student's essay* → *student* | *students* | *student's* ?
- *we're, I'm* → *we are, I am*


2. Compound words or collocations:

- *Hewlett-Packard* → *Hewlett Packet*?
- *state of the art, car park, Kuala Lumpur* → 1 or 2 token?

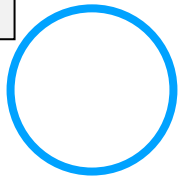
3. Abbreviations/Acronyms:

- AIDS, PIN → ???
- 
- 

- 
- 
- 
- Sent_tokenize is a bit more sophisticated than just using some punctuation to determine the end of a sentence.
 - The function uses an instance of **PunktSentenceTokenizer** which has been pre-trained to identify sentences within longer blocks of text using statistical model.
 - Notice that the function did not consider U.N. as the end and start of new sentences, instead, it returned two sentences in the result list as we would like.

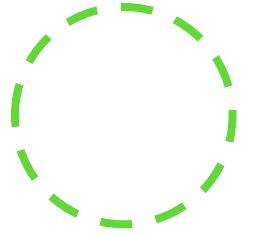


```
Input:new_para = "The U.N. has had a peace keeping force
along the border for many years. The recent signing of a
peace treaty could mean this force can go home soon."
print(sent_tokenize(new_para))
Output:['The U.N. has had a peace keeping force along the
border for many years.', 'The recent signing of a peace
treaty could mean this force can go home soon.']
```



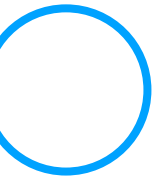


2. LOWERCASING



- One of the simplest and most effective form of text preprocessing.
- Reducing all letters to lower case
 - Lowercasing is very useful for search such as in Search Engine
 - Possible exception: uppercase in mid-sentence? e.g. SEAL vs seal
- It is applicable to most text mining and NLP applications and can help in cases where your dataset is not very large and significantly helps with consistency of expected output.

Raw	Lowercased
Canada CanadA CANADA	canada
TOMCAT Tomcat toMcat	tomcat

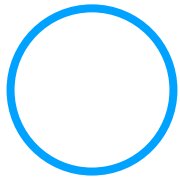





3. NOISE REMOVAL

Noise removal is one of the important steps in Text Mining and NLP especially when dealing with noisy texts such as social media comments, text messages and comments to blog post..

There are various ways to remove noise. This includes *punctuation removal*, *special character removal*, *numbers removal*, *html formatting removal*, *source code removal*, *header removal* and more.






PUNCTUATIONS AND HTML TAGS REMOVAL



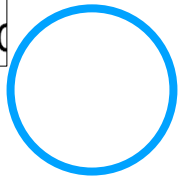
- Removing punctuations (‘.’, ‘,’, etc.) and HTML tags that can interfere with text analysis.
- For e.g., if the texts are web scraped, chances are they will contain some HTML tags. Since these tags are not useful for our NLP tasks, it is better to remove them.



the actors at play. It could very well have been the
the actors. I just don't know.

 But could
the chef in love with? He seemed more enamored
of himself and his youthful exploits, than of anybo
in love with the princess.

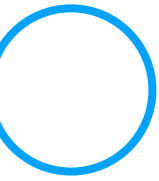
 I was disappo

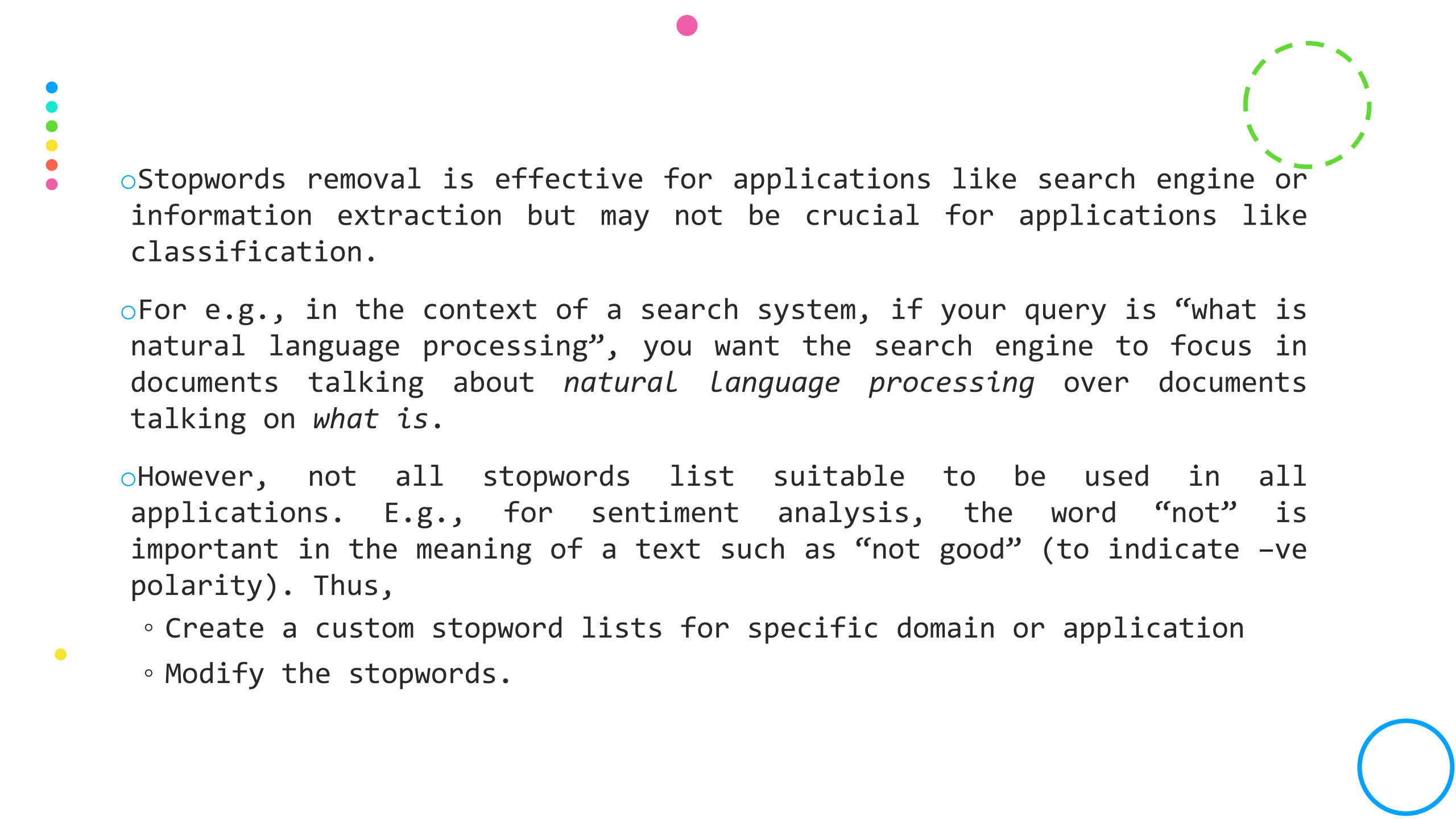




3. STOPWORDS REMOVAL

- Most common words contribute little to overall meaning.
- These words are treated as stopwords which are filtered out before further processing of text.
- The importance of stopwords removal:
 - by removing low information words from text, we can focus on the important words instead.
 - to save computing time and efforts in processing large volumes of text.
- Stopwords removal approach: Classic Method
 - Removing stopwords obtained from pre-compiled lists
 - <http://www.ranks.nl/stopwords>



- 
- Stopwords removal is effective for applications like search engine or information extraction but may not be crucial for applications like classification.
 - For e.g., in the context of a search system, if your query is “what is natural language processing”, you want the search engine to focus in documents talking about *natural language processing* over documents talking on *what is*.
 - However, not all stopwords list suitable to be used in all applications. E.g., for sentiment analysis, the word “not” is important in the meaning of a text such as “not good” (to indicate -ve polarity). Thus,
 - Create a custom stopwords lists for specific domain or application
 - Modify the stopwords.


```
Input:from nltk.corpus import stopwords
text = "Perhaps the biggest hurdle many emerging artists face is money or more
precisely a lack of it. In fact money problems are never far away for many
artists."
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(text)
filtered_text = [w for w in word_tokens if not w in stop_words]
print(word_tokens)
print(filtered_text)
```

```
Output:['Perhaps', 'the', 'biggest', 'hurdle', 'many', 'emerging', 'artists',
'face', 'is', 'money', 'or', 'more', 'precisely', 'a', 'lack', 'of', 'it', '.',
'In', 'fact', 'money', 'problems', 'are', 'never', 'far', 'away', 'for',
'many', 'artists', '.']
```

```
['Perhaps', 'biggest', 'hurdle', 'many', 'emerging', 'artists', 'face',
'money', 'precisely', 'lack', '.', 'In', 'fact', 'money', 'problems', 'never',
'far', 'away', 'many', 'artists', '.']
```

4. STEMMING & LEMMATIZATION

Stemming - a process of reducing inflection in words to their root form.

E.g. *connects* → *connect*

Stemming uses a crude heuristic process that chops off the ends of words in the hope of correctly transforming words into its root form.

In stemming, the 'stem' is obtained after applying a set of rules but without bothering about the part of speech (POS) or the context of the word occurrence. The resulting stem is not necessarily a word.

E.g. *troubles* → *troubl*

Lemmatization - very similar to stemming, where the goal is to remove inflections and map a word to its root form.

However, lemmatization does not just chop things off, it actually transforms words to the actual root.

In contrast, lemmatizing deals with obtaining the 'lemma' of a word which involves reducing the word forms to its root form after understanding the POS and the context of the word in the given sentence.

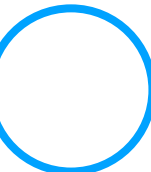


	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

	original_word	stemmed_word
0	trouble	troubl
1	troubled	troubl
2	troubles	troubl
3	troublesome	troublesom

	original_word	lemmatized_word
0	trouble	trouble
1	troubling	trouble
2	troubled	trouble
3	troubles	trouble

	original_word	lemmatized_word
0	goose	goose
1	geese	goose




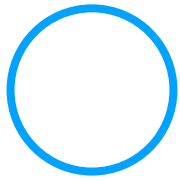


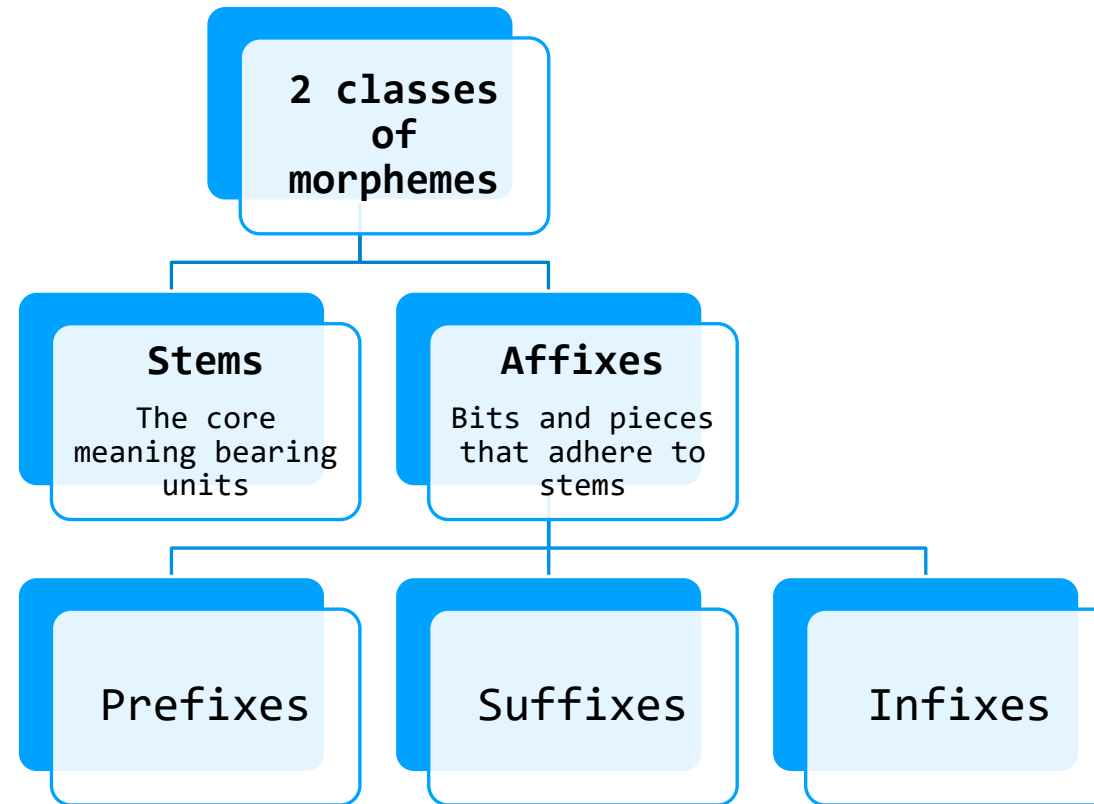
MORPHOLOGY

The study of word structure and word formation processes

The area of linguistics concerned with the internal structure of words.

Questions:

- What is a word? How are words constructed?
 - How is the meaning of a word derived from the components elements?
 - How are words related to each other?
- 
- 



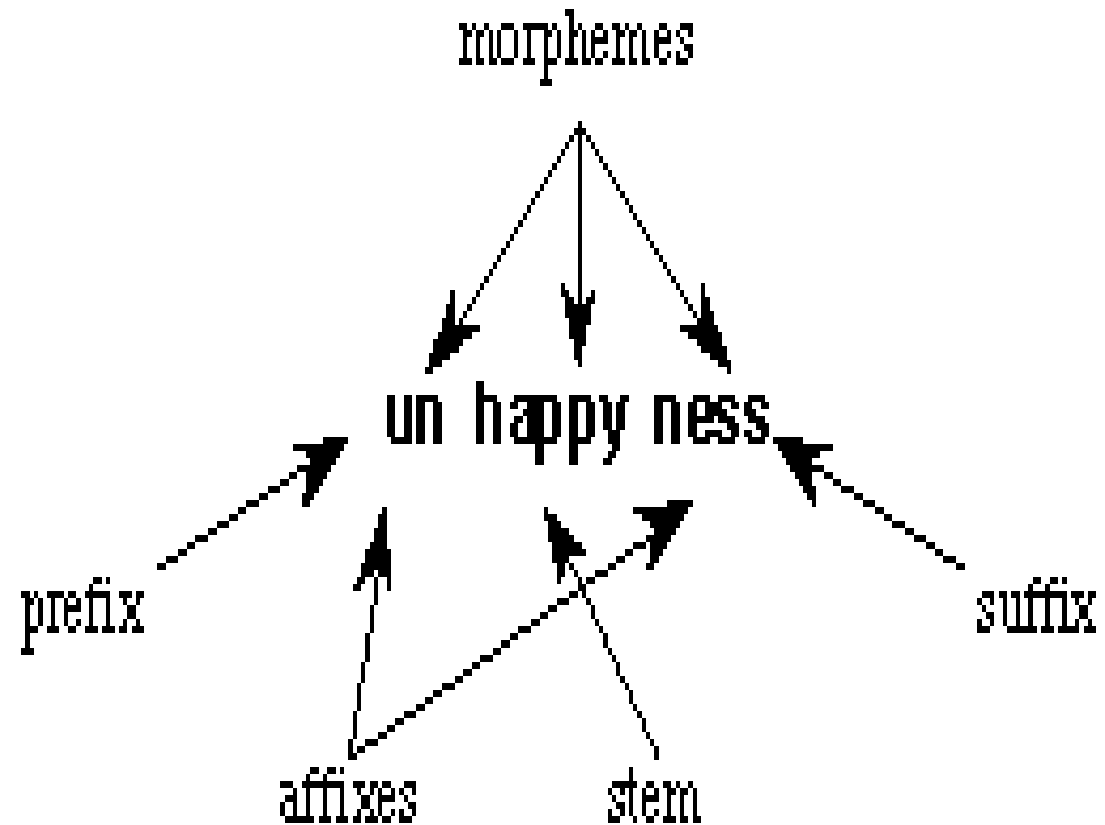
Morpheme


The small meaningful units that make up words.
It is defined as the "minimal unit of meaning".

Consider a word "*unhappiness*".

This word consists of three morphemes each carrying a certain amount of meaning:

- *un* means "not",
- *ness* means "being in a state or condition".
- *Happy* is a *free morpheme* because it can appear on its own (as a "word" in its own right).





Bound morphemes have to be attached to a free morpheme, and so cannot be words in their own right.

In the example given above of *unhappiness*, we saw two kinds of affix, a prefix and a suffix.

Just to show that languages do really vary greatly, there are also infixes.

E.g., the Malay word use an infix *el* to change the word *tapak* (*base*) into *telapak* (*palm of our hand*).



Inflectional Morphology

Derivational Morphology

Cliticization

Non-Concatenative Morphology

Types of Morphology



INFLECTIONAL MORPHOLOGY

Describes predictable changes a word undergoes as a result of syntax

E.g. Plural form for nouns and past tense form for verbs

These changes have no effect on a word's part-of-speech
– noun still remains a noun after pluralization.

Root word + Affix; Affix can be Suffix, Prefix or Infix





NOUN

Regular form:

- the plural form of a noun in English is usually formed from the singular form by adding an s:
- Car + s → cars
- Dog + s → dogs



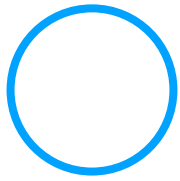
Irregular form:

- Wolf → wolves
- Knife → knives
- Foot → feet



VERB

Verbs are relatively simple:

- Jump – stem
 - Jumps – third person singular, present tense
 - Jumping – present participle
 - Jumped – past tense
- 



DERIVATIONAL MORPHOLOGY

May or may not affect a word's part-of-speech, and may or may not affect it's meaning.

For instance, adding the suffix *-ity* changes the pronunciation of the root of *active* so the stress is on the second syllable: *activity*.

The obvious use of derivational morphology in NLP systems is to reduce the number of forms of words to be stored.





CLITICIZATION



Clitic – a unit whose status lies between that of an affix and a word.

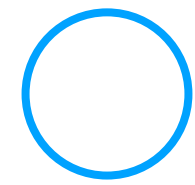
- Proclitics – clitics preceding a word
- Enclitics – clitics following a word

English clitics include these auxiliary verbal form. e.g:

Full form	Clitic	Full form	Clitic
am	'm	have	've
are	're	has	's
is	's	had	'd
will	'll	would	'd

Clitics in English are ambiguous: e.g.

- *She's* – *she is* or *she has*





NON-CONCATENATIVE MORPHOLOGY

Morphemes are combined in more complex ways or through non linear process by modifying internal structure of morphemes.

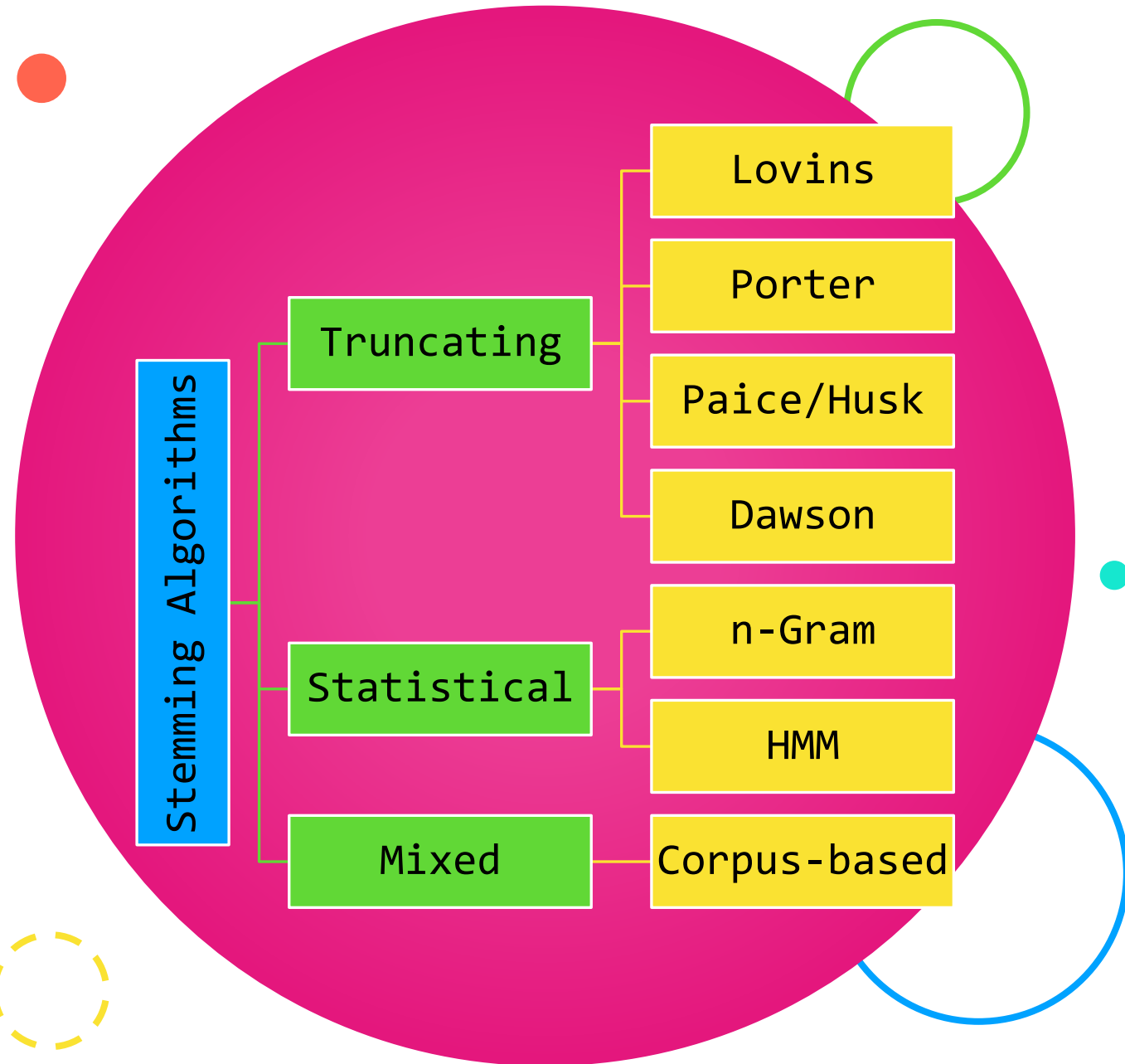
- Circumfixes and infixes

Occurred in Semitic languages such as Arabic and Hebrew.



STEMMING ALGORITHM

A computational procedure which reduces all words with the same root to a common form, by stripping each word of its derivational and inflectional suffixes (Lovins, 1968)





S STEMMER



A simple approach of stemming

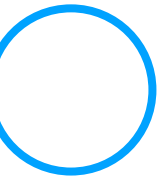
An algorithm conflating singular and plural forms of English nouns.

The algorithm has rules to remove suffixes in plurals so as to convert them to the singular forms

IF a word ends in "ies," but not "cics" or "aies"
THEN "ies" → "y"

IF a word ends in "es," but not "aes," "ees," or "oes"
THEN "es" → "e"

IF a word ends in "s," but not "us" or "ss"
THEN "s" → NULL





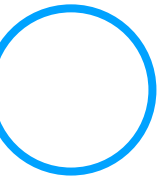
PORTER STEMMER

The most common algorithm, which is also known to be empirically effective for English.

Developed by Martin Porter at the University of Cambridge in 1980.


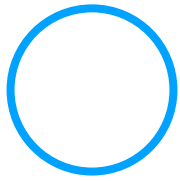
The stemmer is based on the idea that the suffixes in the English language (approximately 1200) are mostly made up of a combination of smaller and simpler suffixes.

- The algorithm - suffix stripping algorithm - consists of 5 steps applying rules within each step.



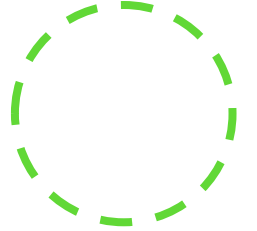





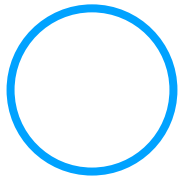
5. TEXT NORMALIZATION

- ❖ Text normalization aims to replace the non-standard tokens that carry significant meanings with the context-appropriate standard words (Liu, Weng, Wang, and Liu, 2011).
 - ❖ Text normalization is important for noisy texts such as social media comments, text messages and comments to blog posts where abbreviations, misspellings and use of out-of-vocabulary words (OOV) are prevalent.
- 
- 

Raw	Normalized
2moro 2mrrw 2morrow 2mrw tomrw	tomorrow
b4	before
otw	on the way
:) :-) ;-)	smile

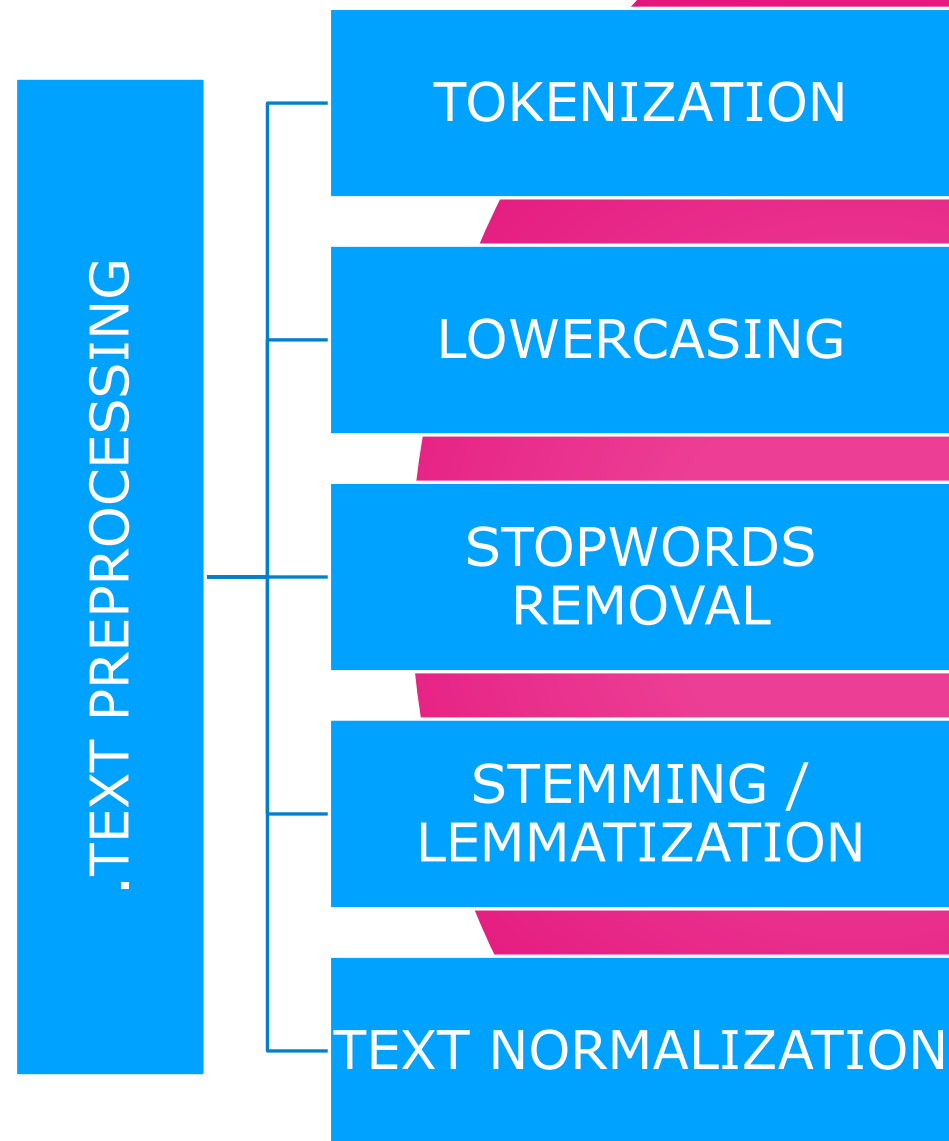
- 
- 
- 
- There isn't a standard way to normalize texts. It typically depends on the task.
 - For example, the way you would normalize clinical texts would arguably be different from how you normalize Tweets data.

Some common approaches to text normalization include:

- ❖ Dictionary-based / lexicon based method
 - ❖ Statistical machine translation (SMT)
- 
- 



SUMMARY





Not all NLP applications need the same level of preprocessing.

For some applications, only minimum preprocessing is required. For e.g. if you have a lot of well written texts to work with in a fairly general domain, then preprocessing is not extremely critical.

However, if you are working in a very narrow domain and data is sparse and noisy (e.g. Tweets on Restaurant review), more preprocessing layers would be helpful to achieve a better performance.

CONCLUSION