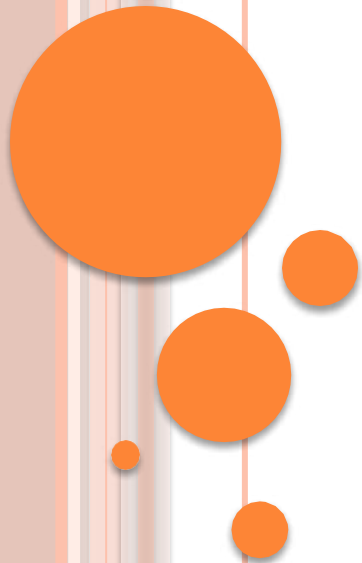


# **NATURAL LANGUAGE PROCESSING**

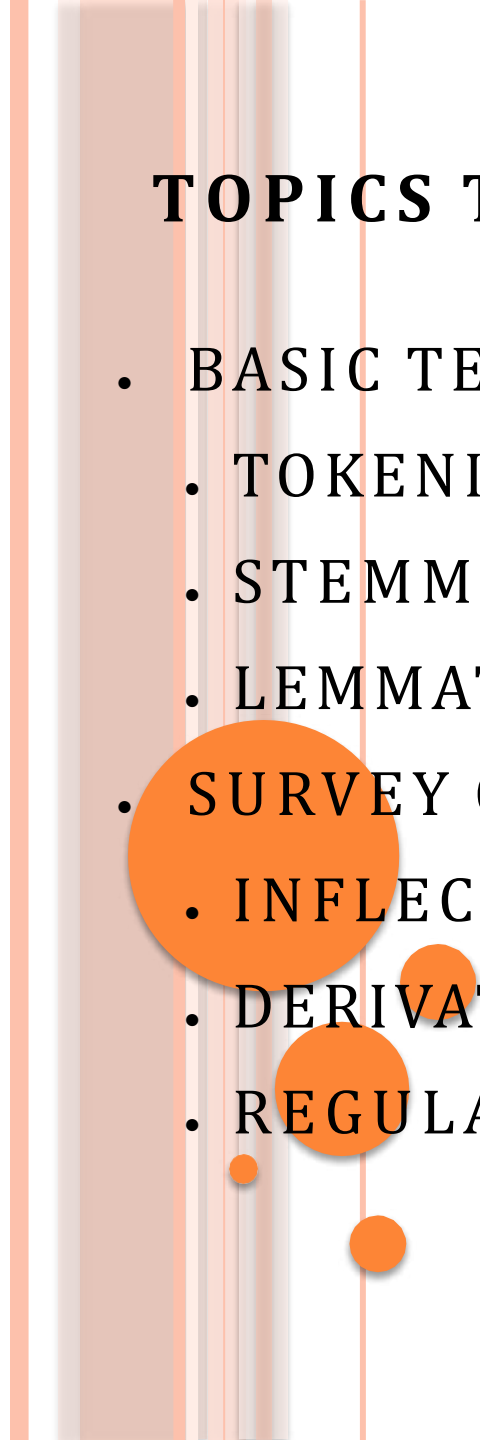
**Presented by:  
Prof Nirali Arora**

# **CHAPTER 2**

## **WORD LEVEL ANALYSIS**



# TOPICS TO BE COVERED

- BASIC TERMS:
    - TOKENIZATION
    - STEMMING
    - LEMMATIZATION
  - SURVEY OF ENGLISH MORPHOLOGY
    - INFLECTIONAL MORPHOLOGY
    - DERIVATIONAL MORPHOLOGY
    - REGULAR EXPRESSION WITH ITS TYPES
- 

# TOKENIZATION

- Tokenization is a common task in Natural Language Processing (NLP). It's a fundamental step in NLP.
- Tokenization is a way of separating a piece of text into smaller units called tokens.
- Here, tokens can be either sentence, words, or character.
- Hence, tokenization can be broadly classified into 3 types – sentence, words and character tokenization.
- If the text is split into words, then its called as 'Word Tokenization' and if it's split into sentences then its called as 'Sentence Tokenization'.
- Generally 'space' is used to perform the word tokenization and characters like 'periods, exclamation point and newline char are used for Sentence Tokenization.

## Input Text

Tokenization is one of the first step in any NLP pipeline. Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens.

## Word Tokenization

Tokenization	is	one	of
the	first	step	in
any	NLP	pipeline	Tokenization
is	nothing	but	splitting
the	raw	text	into
small	chunks	of	words
or	sentences	called	tokens

## Sentence Tokenization

Tokenization is one of the first step in any NLP pipeline

Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens

- Why Tokenization is Required?
- Every sentence gets its meaning by the words present in it.
- So by analyzing the words present in the text we can easily interpret the meaning of the text.
- Once we have a list of words we can also use statistical tools and methods to get more insights into the text.
- For example, we can use word count and word frequency to find out important of word in that sentence or document.

- **Sentence Tokenization**

- Sentence Tokenization is use for splitting the sentences in the paragraph.
- For Example- "Hello everyone. You are studying NLP Subject."
- Sentence Tokenizer generates the following result:  
[‘Hello everyone.’, ‘You are studying NLP Subject.’]
- ```
import nltk
```
- ```
from nltk.tokenize import sent_tokenize
```
- ```
text = "Hello everyone. You are studying NLP Subject."
```
- ```
print(sent_tokenize( text))
```
- Output- [‘Hello everyone.’, ‘You are studying NLP Subject.’]

- **Word Tokenization-**

- Word Tokenization is the most commonly used tokenization algorithm.
- It splits a piece of text into individual words based on a certain delimiter (characters like ‘,’ or ‘;’ or ““,””). Depending upon delimiters, different word-level tokens are formed.
- For Example- Never Give Up
- Word Tokenizer generates the following result:
  - ‘Never’, ‘Give’, ‘Up’
- ```
import nltk from nltk.tokenize import word_tokenize
```
- ```
text = "Hello everyone."
```
- ```
print(word_tokenize( text))
```
- Output- [‘Hello’, ‘everyone’, ‘.’]



- Drawbacks of Word Tokenization
- One of the major issues with word tokens is dealing with Out Of Vocabulary (OOV) words.
- OOV words refer to the new words which are encountered at testing.
- These new words do not exist in the vocabulary. Hence, these methods fail in handling OOV words.
- This is often solved by replacing unknown words with a simple token that communicates that a word is unknown.
- This is a rough solution, especially since 5 'unknown' word tokens could be 5 completely different unknown words or could all be the exact same word.

- **Character Tokenization-**

- Character Tokenization splits a piece of text into a set of characters. It overcomes the drawbacks we saw above about Word Tokenization.
- Character Tokenizers handle OOV words coherently by preserving the information of the word.
- It breaks down the OOV word into characters and represents the word in terms of these characters.

- **Drawbacks of Character Tokenization**

- Character tokens solve the OOV problem but the length of the input and output sentences increases rapidly as we are representing a sentence as a sequence of characters.
- As a result, it becomes challenging to learn the relationship between the characters to form meaningful words.

- **Sub word tokenization-**

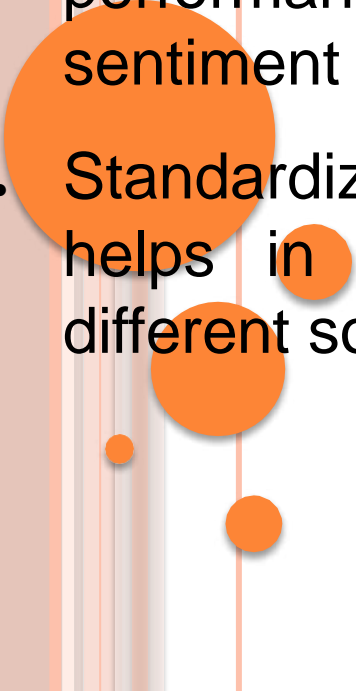
- It is similar to word tokenization, but it breaks individual words down a little bit further using specific linguistic rules.
- One of the main tools they utilize is breaking off affixes. Because prefixes, suffixes, and infixes change the inherent meaning of words, they can also help programs understand a word's function.
- The sub word model will search for these sub words and break down words that include them into distinct parts.
- For example, the query "What is the tallest building?" would be broken down into 'what' 'is' 'the' 'tall' 'est' 'build' 'ing'

## • **Tokenization Challenges in NLP**

- Languages like English or French define the boundary of the sentences by using white spaces, or punctuation marks. Unfortunately, this method couldn't be applicable for other languages like Chinese, Japanese, Korean Thai, Urdu, Tamil, and others.
- This problem creates the need to develop a common tokenization tool that combines all languages.
- Another challenge is symbols that change the meaning of the word significantly. We intuitively understand that a '\$' sign with a number attached to it (\$100) means something different than the number itself (100). This can cause an issue for machines trying to isolate their meaning as a part of a data string.
- Contractions such as 'you're' and 'I'm' also need to be properly broken down into their respective parts. Failing to properly tokenize every part of the sentence can lead to misunderstandings later in the NLP process.

# STEMMING

- Stemming is used to normalize words into its base form or root form.
- For example, celebrates, celebrated and celebrating, all these words are originated with a single root word "celebrate."
- The big problem with stemming is that sometimes it produces the root word which may not have any meaning.
- For Example, intelligence, intelligent, and intelligently, all these words are originated with a single root word "intelligen." In English, the word "intelligen" do not have any meaning.

- Reasons for using stemming:
  - Text simplification: Stemming helps simplify text data by reducing words to their base forms, making it easier for NLP models to process and analyze the text.
  - Improved model performance: By reducing word variations, stemming can lead to better model performance in tasks such as text classification, sentiment analysis, and information retrieval.
  - Standardization: Stemming standardizes words, which helps in comparing and matching text data across different sources and contexts.
- 

# Types of Stemmer in NLTK

## 1. Porter Stemmer – PorterStemmer()

- Here five steps of word reduction are used in the method, each with its own set of mapping rules.
- Porter Stemmer is the original stemmer and is renowned for its ease of use and rapidity.
- The resultant stem is a shorter word with the same root meaning.
- PorterStemmer() is a module in NLTK that implements the Porter Stemming technique.
- Example of PorterStemmer()
- In the example below, we construct an instance of PorterStemmer() and use the Porter algorithm to stem the list of words.

- `from nltk.stem import PorterStemmer`
- `Porter = PorterStemmer()`
- `words =`  
`['connects','connections','connected','connection','connecting']`
- `for word in words:`
- `print(word,"-->",porter.stem(word))`
- OUTPUT:
- `connects-->connect`
- `connecting-->connect`
- `connections-->connect`
- `connected-->connect`
- `connection-->connect`



## **2. Snowball Stemmer – SnowballStemmer()**

- Martin Porter also created Snowball Stemmer.
- The method utilized in this instance is more precise and is referred to as “English Stemmer” or “Porter2 Stemmer.”
- It is somewhat faster and more logical than the original Porter Stemmer.
- SnowballStemmer() is a module in NLTK that implements the Snowball stemming technique.
- Let us examine this form of stemming using an example.

```
from nltk.stem import SnowballStemmer
snowball = SnowballStemmer(language='english')
Words=['generous','generate','generously','generation']
for word in words:
    print(word,"--->",snowball.stem(word))
```

**[Out] :**

```
generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat
```

### 3. Lancaster Stemmer – LancasterStemmer()

- Lancaster Stemmer is straightforward, although it often produces results with excessive stemming.
- LancasterStemmer() is a module in NLTK that implements the Lancaster stemming technique.
- Lets illustrate this with an example.
- ```
from nltk.stem import LancasterStemmer  
lancaster = LancasterStemmer()  
words = ['eating','eats','eaten','puts','putting']  
for word in words:  
    print(word,"--->",lancaster.stem(word))
```

Output:

eating ---> eat

eats ---> eat

eaten ---> eat

puts ---> put

putting ---> put

## 4. Regex Stemmer – RegexStemmer()

- Regex stemmer identifies morphological affixes using regular expressions.
- Substrings matching the regular expressions will be discarded.
- RegexStemmer() is a module in NLTK that implements the Regex stemming technique.
- Let us try to understand this with an example.

```
from nltk.stem import RegexStemmer
regex=RegexStemmer('ing$|s$|e$|able$',
min=4)
words = ['mass','was','bee','computer','advisable']
for word in words:
print(word,"--->",regex.stem(word))
```

Output:

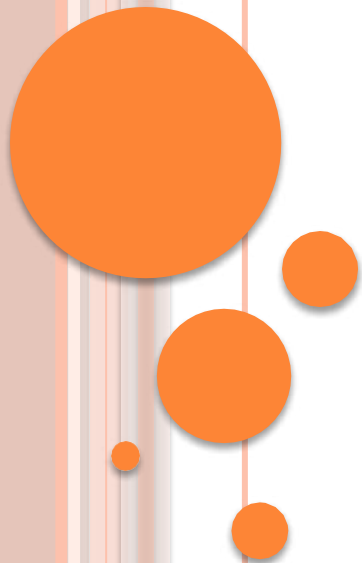
mass ---> mas

was ---> was

bee ---> bee

computer ---> computer

advisable ---> advis

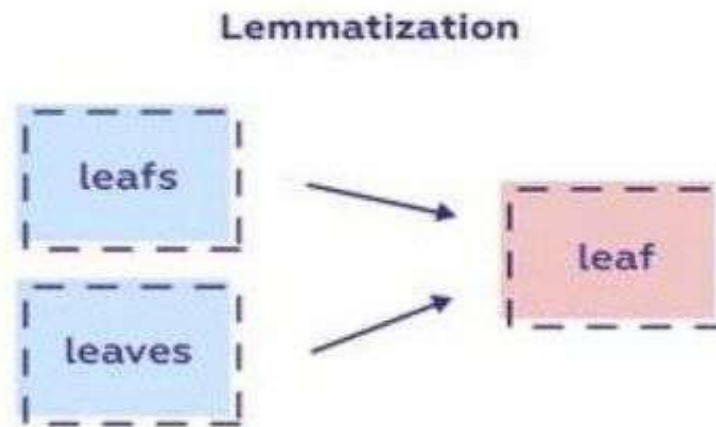


# LEMMATIZATION

Lemmatization is a text normalization technique used in Natural Language Processing (NLP), that switches any kind of a word to its base root mode.

Lemmatization is responsible for grouping different inflected forms of words into the root form, having the same meaning.

Lemmatization involves using a vocabulary and morphological analysis of words, removing inflectional endings, and returning the base form of a word (lemma).



# What is Lemmatization used for?

- Lemmatization is among the best ways to help chatbots understand your customers' queries to a better extent.
- Since this involves a morphological analysis of the words, the chatbot can understand the contextual form of the words in the text and can gain a better understanding of the overall meaning of the sentence that is being lemmatized.
- Lemmatization is also used to enable robots to speak and converse.
- This makes lemmatization a rather important part of natural language understanding (NLU) and natural language processing (NLP) in artificial intelligence.

# Stemming Vs. Lemmatization

Stemming	Lemmatization
In stemming, the end or beginning of a word is cut off, keeping common prefixes and suffixes that can be found in inflected words in mind.	Lemmatization uses dictionaries to conduct a morphological analysis of the word and link it to its lemma. Lemmatization always returns the dictionary meaning of the word while converting into root-form.
Stemming tends to be a faster process than lemmatization because it chops words without knowing the context of the word in the sentences which they are in.	Lemmatization, on the other hand, is a slower process than stemming, it knows the context of the word before proceeding.
stemming is a rule-based approach	lemmatization is a dictionary-based approach
The process of stemming also has a lower degree of accuracy	The process of stemming also has a higher degree of accuracy
Eg: Change, Changing, Changes, Changed, Changer → Chang	Eg: Change, Changing, Changes, Changed, Changer → Change