UNIT 4

INT426:GENERATIVE ARTIFICIAL INTELLIGENCE

- NLP stands for *Natural Language Processing*, which is a part of Computer Science, Human language, and Artificial Intelligence.
- It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages.
- It helps developers to organize knowledge for performing tasks such as translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction, and topic segmentation.

- NLP problem can be divided into two tasks:

  - ✓ **Processing written text**, using lexical, syntactic and semantic knowledge of the language as well as the required real world information.

  - ✓ **Processing spoken language**, using all the information needed above plus additional knowledge about phonology as well as enough added information to handle the further ambiguities that arise in speech.

- Problem : English sentences are incomplete descriptions of the information that they are intended to convey.

- dogs are outside is incomplete – it can mean
  - Some dogs are on the lawn.
  - Three dogs are on the lawn.
  - Moti, Hira & Buzo are on the lawn.

- The good side : Language allows speakers to be as vague or as precise as they like. It also allows speakers to leave out things that the hearers already know.

- **The Problem :** The same expression means different things in different context.

  - Where's the water? (In a lab, it must be pure)
  - Where's the water? (When you are thirsty, it must be potable or drinkable)
  - Where's the water? (Dealing with a leaky roof, it can be filthy)

- **The good side :** Language lets us communicate about an infinite world using a finite number of symbols.

- **The problem :** There are lots of ways to say the same thing :
  - Mary was born on October 11.
  - Mary's birthday is October 11.

- **The good side :** When you know a lot, facts imply each other. Language is intended to be used by agents who know a lot.

Components of NLP

There are the following two components of NLP -

1. Natural Language Understanding (NLU)

- Natural Language Understanding (NLU) helps the machine to understand and analyze human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.
- NLU mainly used in Business applications to understand the customer's problem in both spoken and written language.

NLU involves the following tasks -

- It is used to map the given input into useful representation.
- It is used to analyze different aspects of the language.

2. Natural Language Generation (NLG)

Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation.

It mainly involves –

Text planning − It includes retrieving the relevant content from knowledge base.

Sentence planning − It includes choosing required words, forming meaningful phrases, setting tone of the sentence.

Text Realization − It is mapping sentence plan into sentence structure.

The NLU is harder than NLG.

Difference between NLU and NLG

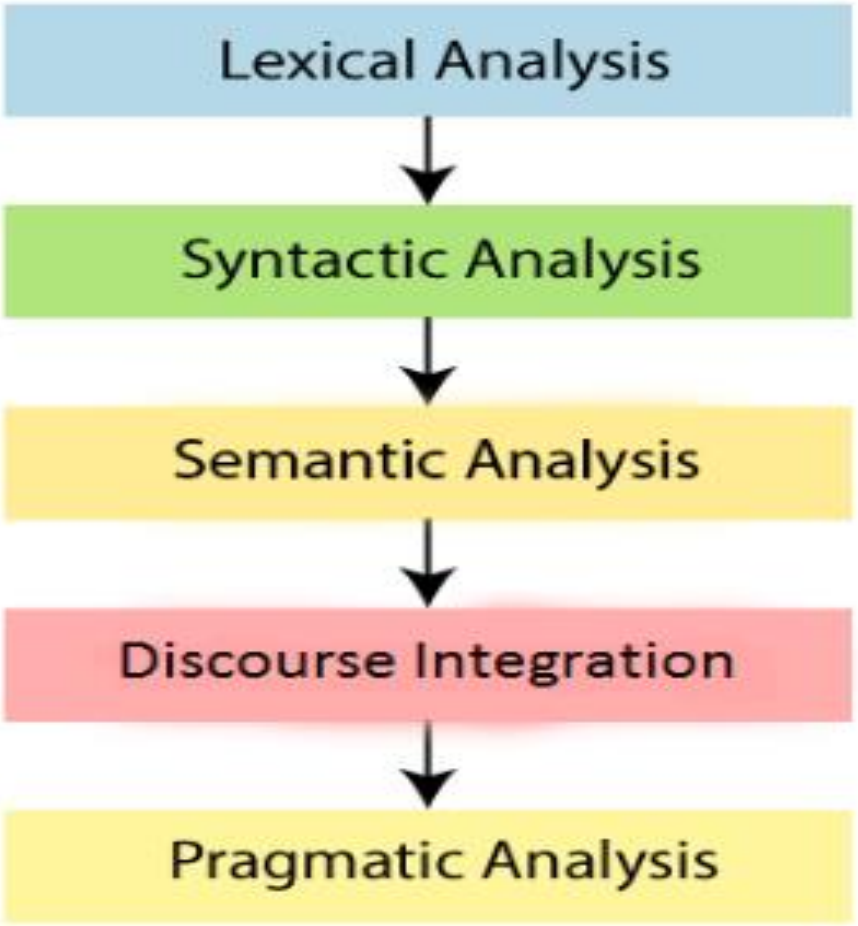
NLU	NLG
NLU is the process of reading and interpreting language.	NLG is the process of writing or generating language.
It produces non-linguistic outputs from natural language inputs.	It produces constructing natural language outputs from non-linguistic inputs.

Applications of NLP

1. **Question Answering**
2. **Spam Detection**
3. **Sentiment Analysis**
4. **Machine Translation**
5. **Spelling correction**
6. **Speech Recognition**
7. **Chatbot**
8. **Information extraction**

Phases of NLP

There are the following five phases of NLP:



1. Lexical Analysis and Morphological

- The first phase of NLP is the Lexical Analysis.
- This phase scans the source code as a stream of characters and converts it into meaningful lexemes. The validity of words are checked according to **lexicon.**
- **Lexicon** stands for dictionary, is a collection of all possible valid words of the language along with their meaning.
- **Lexicon** provides all necessary rules and data for carrying out the first state analysis.
- It divides the whole text into paragraphs, sentences, and words.
- The details of words, like their type (noun, verb & adverb, and other details of nouns & verb etc.) are checked.

2. Syntactic Analysis (Parsing)

- Syntax refers to the study of formal relationships between words of sentences. In this phase the validity of a sentence checked according to grammar rules.
- To perform the syntactic analysis, the knowledge of grammar and parsing technique is required.
- The grammar is formal specification of rules allowable in the language, and parsing is a method of analyzing a sentence to determines its structure according to grammar.
- Syntactic analysis is done using parsing. Two basic parsing techniques are used: **top-down parsing** and **bottom-up parsing.**

Example: *Agra goes to the Poonam*

In the real world, Agra goes to the Poonam, does not make any sense, so this sentence is rejected by the Syntactic analyzer.

3. Semantic Analysis

- Semantic analysis is concerned with the meaning representation. It mainly focuses on the literal meaning of words, phrases, and sentences.

4. Discourse Integration

- Discourse Integration depends upon the sentences that proceeds it and also invokes the meaning of the sentences that follow it.

5. Pragmatic Analysis

- Pragmatic is the fifth and last phase of NLP. It helps you to discover the intended effect by applying a set of rules that characterize cooperative dialogues.

For Example: "Open the door" is interpreted as a request instead of an order.

Why NLP is difficult?

NLP is difficult because Ambiguity and Uncertainty exist in the language.

Ambiguity

There are the following three ambiguity -

✓**Lexical Ambiguity**

Lexical Ambiguity exists in the presence of two or more possible meanings of the sentence within a single word.

Example:

- Manya is looking for a **match**.
- In the above example, the word match refers to that either Manya is looking for a partner or Manya is looking for a match. (Cricket or other match)

✓ **Syntactic Ambiguity**

- Syntactic Ambiguity exists in the presence of two or more possible meanings within the sentence.

Example:

- I saw the girl with the binocular.
- In the above example, did *I have the binoculars?* Or *did the girl have the binoculars?*

✓ **Referential Ambiguity**

- Referential Ambiguity exists when you are referring to something using the pronoun.

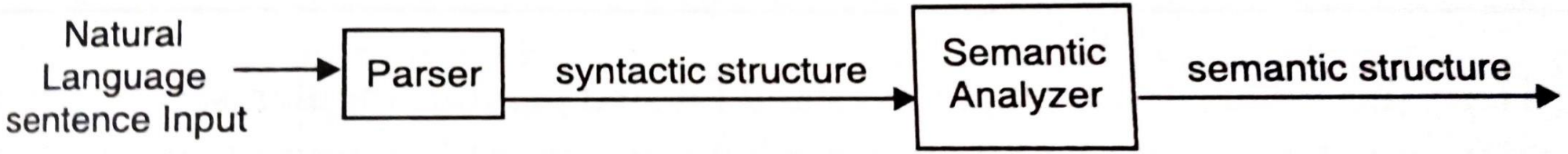
Example: *Kiran went to Sunita. She said, "I am hungry."*

- In the above sentence, you do not know that who is hungry, either Kiran or Sunita.

Parse Tree

The first task for any NLP-based system is to *read (or to parse)* the text

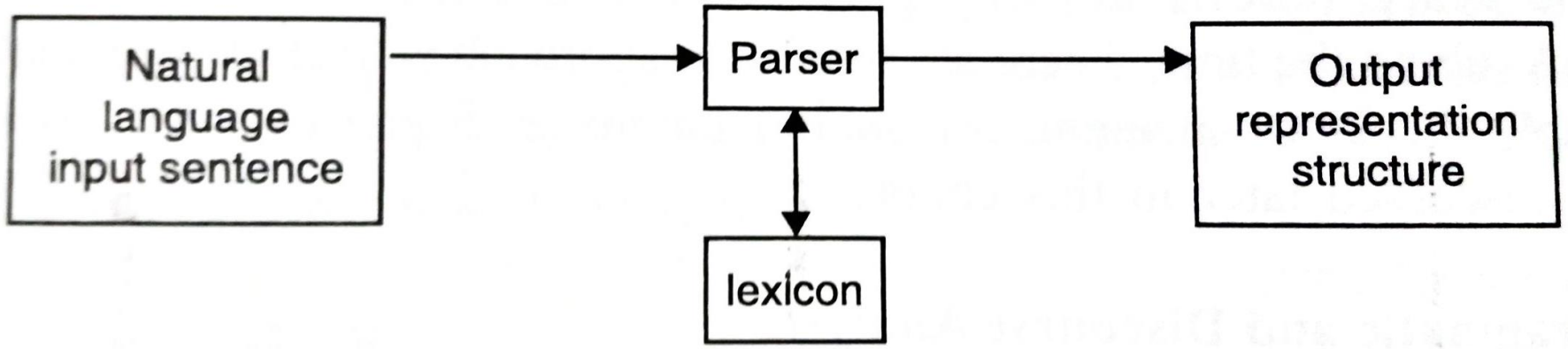
- During language analysis, the parser generates basic syntactic structure of the sentence.
- This stage uses grammatical information to perform some structural preprocessing on the input.
- It performs the task of applying grammar rules and computes syntactic representation of the meaning.
- During parsing, it checks the *validity of a sentence according to grammar rules*.
- It can not check *semantic validity of a sentence*. If a sentence is grammatically correct and semantically incorrect , will be considered as correct after first phase.



The Process of Natural language understanding

Parsing depends on three components of a language-

1. Lexicon
2. Categorization
3. Grammar Rules



Basic parsing technique

Lexicon

stench | breeze | glitter | nothing | wumpus | pit | pits | gold | east | ..
is | see | smell | shoot | feel | stinks | go | grab | carry | kill | turn | …
right | left | east | south | back | smelly | …
here | there | nearby | ahead | right | left | east | south | back | …
me | you | I | it | S=HE | Y'ALL …
John | Mary | Boston | UCB | PAJC | …
the | a | an | …
to | in | on | near | …
and | or | but | …
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Categorization

Noun > stench | breeze | glitter | nothing | wumpus | pit | pits | gold | east | ..
Verb > is | see | smell | shoot | feel | stinks | go | grab | carry | kill | turn | …
Adjective > right | left | east | south | back | smelly | …
Adverb > here | there | nearby | ahead | right | left | east | south | back | …
Pronoun > me | you | I | it | S=HE | Y'ALL …
Name > John | Mary | Boston | UCB | PAJC | …
Article > the | a | an | …
Preposition > to | in | on | near | …
Conjunction > and | or | but | …
Digit > 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Grammar Rules

- “The large cat”
- This phrase can be parsed by an NLP-system if it has a grammar like

Noun Phrase -> Determiner + Adjective + Noun

- If your system finds a phrase or sentence that has a pattern not mentioned in its set of Grammar Rules it won’t be able to parse them.

Therefore,

- Parsing is the process of using grammar rules to determine whether a sentence is legal,
  - and to obtain its Syntactic Tree

Modeling a Sentence using Phase Structure

Every sentence consists of an internal structure which could be modeled with the phrase structure.

Algorithm : Steps

- Apply rules on an proposition
- The base proposition would be :

  S (the root, ie the sentence).
- The first production rule would be :

  (NP = noun phrase, VP = verb phrase)

  S -> (NP, VP)
- Apply rules for the 'branches'

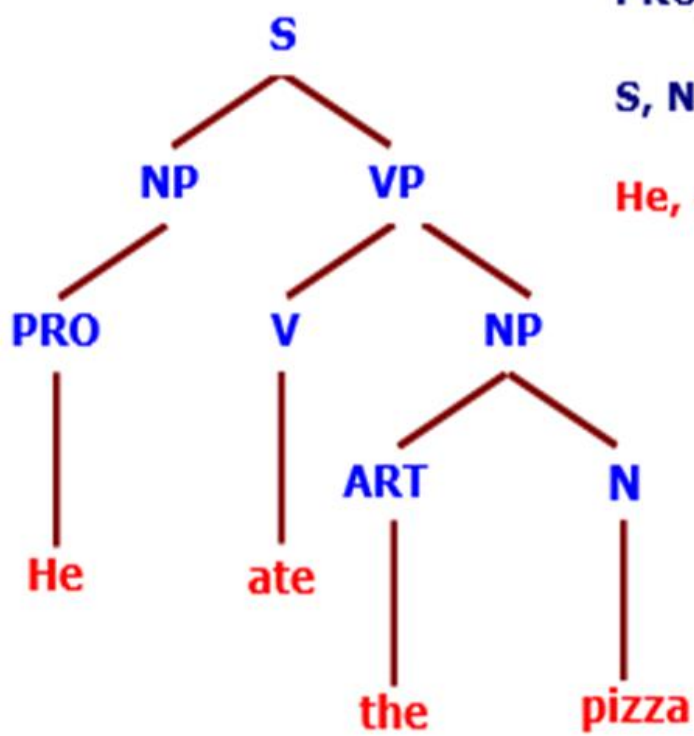
  NP -> noun VP -> verb, NP
- The verb and noun have terminal nodes which could be any word in the lexicon for the appropriate category.
- The end is a tree with the words as terminal nodes, which is referred as the sentence.

Example : Parse tree

- sentence "He ate the pizza",
- apply the grammar with rules

  S -> NP VP, NP -> PRO, NP -> ART N, VP -> V NP,
- the lexicon structure is

  ("ate" V) ("he" PRO) ("pizza" N) ("the" ART)
- The parse tree is



PRO, V, ART, N - lexical non-terminals

S, NP, VP - phrasal non-terminal

He, ate, the, pizza - words or terminal

Tokenization for Natural Language Processing

- The input in natural language processing is text.
- The data collection for this text happens from a lot of sources.
- This requires a lot of cleaning and processing before the data can be used for analysis.

These are some of the methods of processing the data in NLP:

- Tokenization
- Stop words removal
- Stemming
- Normalization
- Lemmatization
- Parts of speech tagging

Tokenizing text data

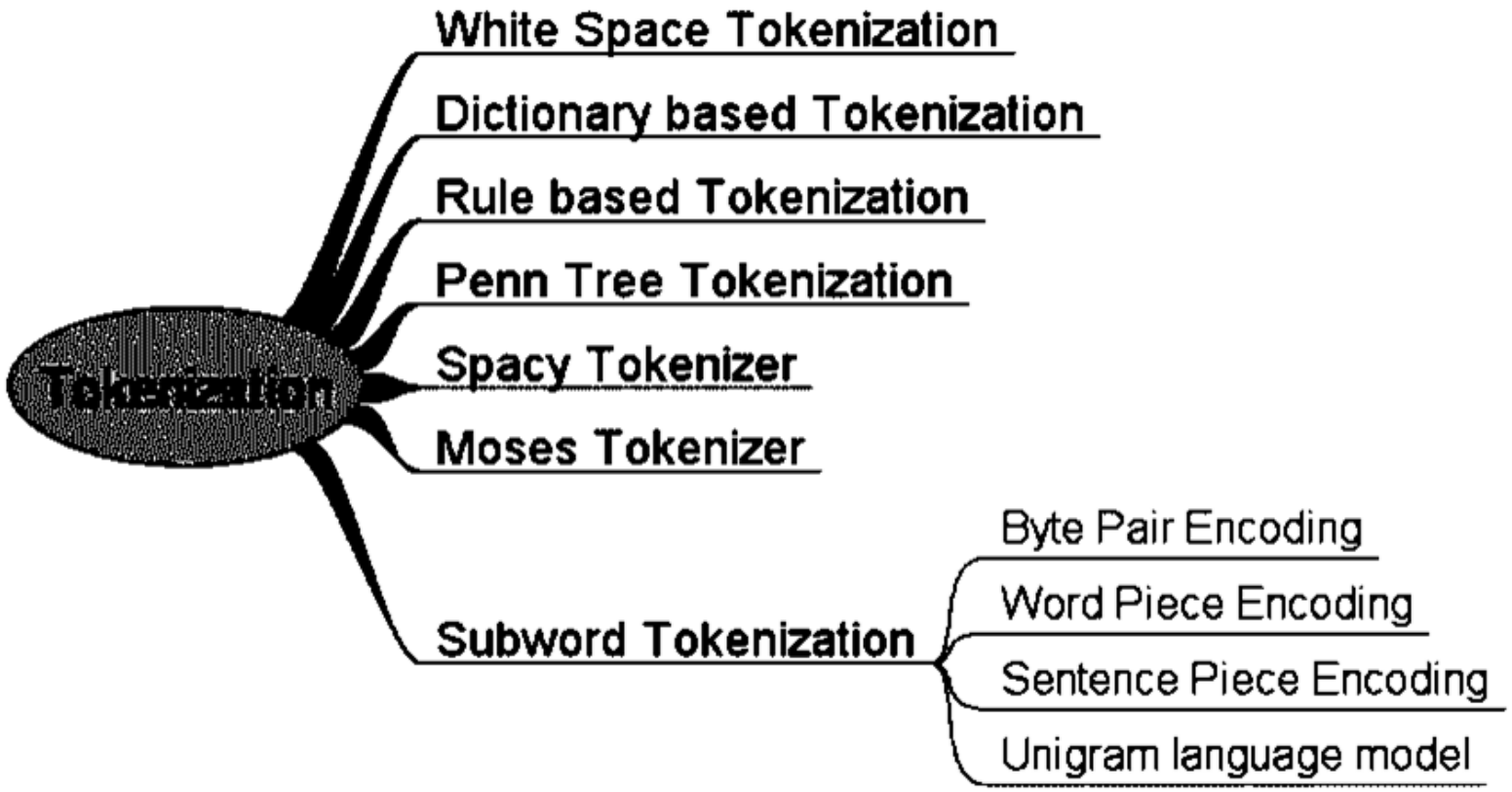
- When we deal with text, we need to break it down into smaller pieces for analysis.
- It is the process of dividing the input text into a set of pieces like words or sentences. These pieces are called tokens.
- These tokens help in understanding the context or developing the model for the NLP.
- The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.
- For example, the text **“It is raining” can be tokenized into ‘It’, ‘is’, ‘raining’**

- Tokenization can be done to either separate *words or sentences*.
- If the text is split into words using some separation technique *it is called word tokenization* and same separation done for sentences *is called sentence tokenization*.

- ***There are different methods and libraries available to perform tokenization in python such as NLTK, Gensim, Keras***

- ***Stop words*** are those words in the text which does not add any meaning to the sentence and their removal will not affect the processing of text for the defined purpose.
- They are removed from the vocabulary to *reduce noise and to reduce the dimension of the feature set.*

tokenization techniques



White Space Tokenization
This is the simplest tokenization technique. Given a sentence or paragraph it tokenizes into words by splitting the input whenever a white space in encountered. This is the fastest tokenization technique but will work for languages in which the white space breaks apart the sentence into meaningful words. Example: English.

Dictionary Based Tokenization
In this method the tokens are found based on the tokens already existing in the dictionary. If the token is not found, then special rules are used to tokenize it. It is an advanced technique compared to whitespace tokenizer.

Rule Based Tokenization
In this technique a set of rules are created for the specific problem. The tokenization is done based on the rules. For example creating rules bases on grammar for particular language.

Regular Expression Tokenizer
This technique uses regular expression to control the tokenization of text into tokens. Regular expression can be simple to complex and sometimes difficult to comprehend. This technique should be preferred when the above methods does not serve the required purpose. It is a rule based tokenizer.

Penn TreeBank Tokenization
Tree bank is a corpus created which gives the semantic and syntactical annotation of language. Penn Treebank is one of the largest treebanks which was published. This technique of tokenization separates the punctuation, clitics (words that occur along with other words like I’m, don’t) and hyphenated words together.

Spacy Tokenizer

This is a modern technique of tokenization which faster and easily customizable. It provides the flexibility to specify special tokens that need not be segmented or need to be segmented using special rules. Suppose you want to keep $ as a separate token, it takes precedence over other tokenization operations.

Moses Tokenizer

This is a tokenizer which is advanced and is available before Spacy was introduced. It is basically a collection of complex normalization and segmentation logic which works very well for structured language like English.

Subword Tokenization

- This tokenization is very useful for specific application where sub words make significance.
- In this technique the most frequently used words are given unique ids and less frequent words are split into sub words and they best represent the meaning independently.
- For example if the word **few** is appearing frequently in the text it will be assigned a unique id, where **fewer** and **fewest** which are rare words and are less frequent in the text will be **split into sub words like few, er, and est**.
- This helps the language model not to learn fewer and fewest as two separate words.
- This allows to identify the unknown words in the data set during training.

Tokenization with NLTK(Natural Language Toolkit)

- Create a new Python file and import the following packages:

  from nltk.tokenize import sent_tokenize, word_tokenize, WordPunctTokenizer

- Define some input text that will be used for tokenization:
  # Define input text
  input_text = "Do you know how tokenization works? It's actually quite interesting! Let's analyze a couple of sentences and figure it out."

- Divide the input text into sentence tokens:
  # Sentence tokenizer
  print("\nSentence tokenizer:")
  print(sent_tokenize(input_text))

- Divide the input text into word tokens:
  # Word tokenizer
  print("\nWord tokenizer:")
  print(word_tokenize(input_text))

- Divide the input text into word tokens using word punct tokenizer:
  # WordPunct tokenizer
  print("\nWord punct tokenizer:")
  print(WordPunctTokenizer().tokenize(input_text))

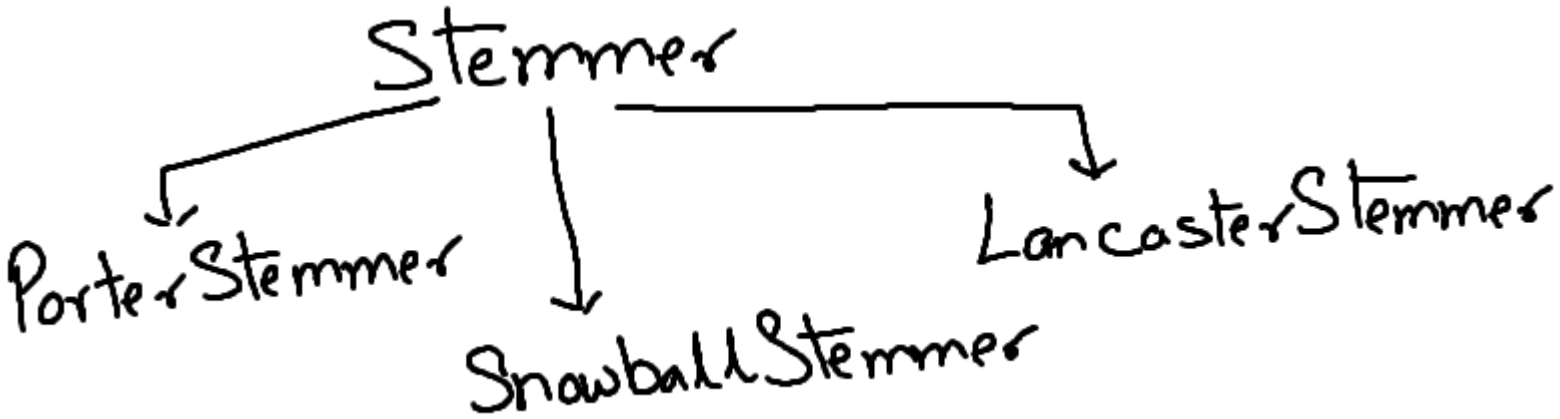
```
Sentence tokenizer:
['Do you know how tokenization works?', "It's actually quite interesting!", "Let's analyze a couple of se
ntences and figure it out."]

Word tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'s", 'actually', 'quite', 'interesting'
, '!', 'Let', "'s", 'analyze', 'a', 'couple', 'of', 'sentences', 'and', 'figure', 'it', 'out', '.']

Word punct tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'", 's', 'actually', 'quite', 'interest
ing', '!', 'Let', "'", 's', 'analyze', 'a', 'couple', 'of', 'sentences', 'and', 'figure', 'it', 'out', '.
']
```

Converting words to their base forms using stemming

Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems.
For example, the stem of the words ***eating, eats, eaten*** is ***eat***.



```
import nltk from nltk.stem import PorterStemmer
word_stemmer = PorterStemmer()
word_stemmer.stem('writing')
```

Output:
write

```
import nltk from nltk.stem import LancatserStemmer
Lanc_stemmer = LancasterStemmer()
Lanc_stemmer.stem('eats')
```
Output
eat

```
import nltk from nltk.stem import SnowballStemmer
French_stemmer = SnowballStemmer(‘french’)
French_stemmer.stem (‘Bonjoura’)
```
Output
bonjour

SnowballStemmer support 15 no-English languages

INPUT WORD	PORTER	LANCASTER	SNOWBALL
writing	write	writ	write
calves	calv	calv	calv
be	be	be	be
branded	brand	brand	brand
horse	hors	hors	hors
randomize	random	random	random
possibly	possibl	poss	possibl
provision	provis	provid	provis
hospital	hospit	hospit	hospit
kept	kept	kept	kept
scratchy	scratchi	scratchy	scratchi
code	code	cod	code

Converting words to their base forms using lemmatization

- **Lemmatization** is another way of reducing words to their base forms.
- To create the base forms that were obtained from those stemmers didn't make sense. For example, all the three stemmers said that the base form of *calves* is *calv*, which is not a real word

- Lemmatization takes a more structured approach to solve this problem.
- The lemmatization process uses a vocabulary and morphological analysis of words. It obtains the base forms by removing the inflectional word endings such as *ing* or *ed*.
- This base form of any word is **known as the lemma**.

```
import nltk from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatizer.lemmatize('books')
```

Output

```
'book'
```

Create a new python file and import the following packages:

```
from nltk.stem import WordNetLemmatizer
```

Define some input words. We will be using the same set of words that we used in the previous section so that we can compare the outputs.

```
input_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',
        'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']
```

Create a `lemmatizer` object:

```
# Create lemmatizer object
lemmatizer = WordNetLemmatizer()
```

Create a list of `lemmatizer` names for table display and format the text accordingly:

```
# Create a list of lemmatizer names for display
lemmatizer_names = ['NOUN LEMMATIZER', 'VERB LEMMATIZER']
formatted_text = '{:>24}' * (len(lemmatizer_names) + 1)
print('\n', formatted_text.format('INPUT WORD', *lemmatizer_names),
        '\n', '='*75)
```

Iterate through the words and lemmatize the words using Noun and Verb lemmatizers:

```
# Lemmatize each word and display the output
for word in input_words:
    output = [word, lemmatizer.lemmatize(word, pos='n'),
            lemmatizer.lemmatize(word, pos='v')]
    print(formatted_text.format(*output))
```

INPUT WORD	NOUN LEMMATIZER	VERB LEMMATIZER
writing	writing	write
calves	calf	calve
be	be	be
branded	branded	brand
horse	horse	horse
randomize	randomize	randomize
possibly	possibly	possibly
provision	provision	provision
hospital	hospital	hospital
kept	kept	keep
scratchy	scratchy	scratchy
code	code	code

Spell Checking

- A **Spell checker** is one of the basic tools required for language processing.
- Spell checking involves identifying words and non words and also suggesting the possible alternatives for its correction.

- used in
  - Word processing
  - Character or text recognition
  - Speech recognition and generation.
- Most available spell checkers focus on processing isolated words and do not take into account the context.

- “Henry **sar** on the box”
- “Henry **at** on the box”

Spelling Errors

- Three cause of error are:
- **Insertion:** Insertion of extra letter while typing. E.g. maximum typed as maxiimum.
- **Deletion**: A case of a letter missing or not typed in a word. E.g. netwrk instead of network.
- **Substitution:** Typing of a letter in place of the correct one. E.g. intellugence.

- Spelling errors may be classified into following types:
- Typographic errors:
  - Cause due to mistakes committed while typing.
  - E.g. netwrk in place of network.
- Orthographic errors:
  - Result due to a lack of comprehension of the concerned language on part of user.
  - E.g. arithmetic, wellcome, accomodation.
- Phonetic errors:
  - result due to poor cognition on part of listner.
  - E.g. the word rough could be spelt as ruff.
  - Listen as lisen, piece as peace or peas, reed as read.

Spell checking techniques

- Spell checking techniques can be broadly classified into three categories-
- (a) **Non-Word error detection**: This process involves the detection of misspelled words or non-words.
  - E.g. the word sopar is a non-word ; its correct form is super or sober.
  - The most commonly used techniques to detect such errors are the **N-gram analysis** and **Dictionary look-up.**

- An N-gram model is built by counting how often word sequences occur in corpus text and then estimating the probabilities.
- An N-gram model is one type of a Language Model (LM), which is about finding the probability distribution over word sequences
- N-gram is the sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”, and a 3-gram (or trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”
- Used in text (handwritten or printed) recognition.

- **Dictionary look-up** involves the use of an efficient dictionary lookup coupled with pattern-matching algorithm (such as hashing technique, Finite state automata), dictionary portioning schemes and morphological processing methods.

(b) Isolated-word error correction:

- Focus on the correction of an isolated non-words by finding its nearest and meaningful word and make an attempt to rectify the error.
- It thus transform the word “soper” into super.
- Isolated word correction may be looked upon as a combination of three sub-problems - Error detection, candidate (correct word) generation, and ranking of correct candidates.

Minimum Edit distance technique

- Wanger[1974] define the minimum edit distance between the misspelled word and the possible correct candidate as the minimum number of edit operations needed to transform the misspelled word to the correct candidate.
- Edit operation-insertion, deletion, and substitution of a single character.
- The minimum number of such operations required to affect the transformation is commonly known as *Levenshtein distance(measure the difference between two written words.)*.

• Input: str1 = "cat", str2 = "cut"

Output: 1

We can convert str1 into str2 by replacing 'a' with 'u'.

• Input: str1 = "sunday", str2 = "saturday"

Output: 3

Last three and first characters are same. We basically need to convert "un" to "atur". This can be done using below three operations. Replace 'n' with 'r', insert t, insert a

(c) Context dependent error detection and correction:

- In addition to detect errors, try to find whether the corrected word fits in to context of the sentence.
- More complex to implement.
- “Peace comes from within”, “Piece comes from within” ; first word in both sentence is a correct word.
- This involves correction of real-word errors or those that result in another valid error.

Soundex Algorithm

- **Soundex** is a phonetic algorithm and is based on how close two words are depending on their English pronunciation
- Algorithm:
  - Remove all punctuation marks and capitalize the letters in the word.
  - Retain the first letter of the word.
  - Remove occurrence of letters-A,E,I,O,U,H,W,Y apart from very first letter.
  - Replace the letter other than first as shown in table.

Letters	Digits
B, F, P, V	1
C, G, J, K, Q, S, X, Z	2
D, T	3
L	4
M, N	5
R	6

–If two or more adjacent letters , non separated by vowels, have the same numeric value retain only one of them.

–Return the 1st four character; pad with zeros if there are less than four.

Word	Soundex Code
Grate, Great	G630
Network, network	N362
Torn	T650
Worn	W650
Horn	H650
Henry, Henary	H560

Used to measure similarity of two words

Bag of Words Model

Bag of Words (BoW) **model** is a simple algorithm used in **Natural Language Processing** for text modelling.
In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.
In BoW **model** a sentence or a document is considered as a '**Bag**' containing **words**.
It will take into account the **words** and their frequency of occurrence in the sentence or the document disregarding semantic relationship in the sentences.

Why is the Bag-of-Words algorithm used

One of the biggest problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length **vector.**

The machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

Example of the Bag-of-Words Model

Step 1: Collect Data

- Below is a snippet of the first few lines of text from the book "A Tale of Two Cities" by Charles Dickens, taken from Project Gutenberg.
- *It was the best of times,*
  it was the worst of times,
  it was the age of wisdom,
  it was the age of foolishness,
- For this small example, let's treat each line as a separate "document" and the 4 lines as our entire corpus of documents.

Step 2: Design the Vocabulary

Now we can make a list of all of the words in our model vocabulary.

- The unique words here (ignoring case and punctuation) are:

“it”

“was”

“the”

“best”

“of”

“times”

“worst”

“age”

“wisdom”

“foolishness”

- That is a vocabulary of 10 words from a corpus containing 24 words.

Step 3: Create Document Vectors

- The next step is to score the words in each document.
- The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.
- Because we know the vocabulary has 10 words, we can use a fixed-length document representation of 10, with one position in the vector to score each word.
- The simplest scoring method is to mark the presence of words as a boolean value, 0 for absent, 1 for present.
- Using the arbitrary ordering of words listed above in our vocabulary, we can step through the first document ("*It was the best of times*") and convert it into a binary vector.

As a binarv vector. this would look as follows:

```
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
```

The other three documents would look as follows:

```
1 "it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
2 "it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
3 "it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
```

without preprocessing:

Sentence 1: “Welcome to Machine Learning, Now start learning”
Sentence 2: “Learning is a good practice”

Sentence 1	Sentence 2
Welcome	Learning
to	is
Machine	a
Learning	good
,	practice
Now	
start	
learning	

Step 1: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

- Welcome
- to
- Machine
- Learning
- ,
- Now
- start
- learning
- is
- a
- good
- practice

Note that the words ‘Learning’ and ‘ learning’ are not the same here because of the difference in their cases and hence are repeated. Also, note that a comma ‘ , ’ is also taken in the list.

The scoring of sentence 1 would look as follows:

Word	Frequency
Welcome	1
to	1
Machine	1
Learning	1
,	1
Now	1
start	1
learning	1
is	0
a	0
good	0
practice	0

The scoring of sentence 2 would look as follows:

Word	Frequency
Welcome	0
to	0
Machine	0
Learning	1
,	0
Now	0
start	0
learning	0
is	1
a	1
good	1
practice	1

Writing the above frequencies in the vector

Sentence 1 → **[1,1,1,1,1,1,1,1,0,0,0,0]**

Sentence 2 → **[0,0,0,1,0,0,0,0,1,1,1,1]**

With preprocessing: Same example

Sentence 1: “Welcome to Machine Learning, Now start learning”
Sentence 2: “Learning is a good practice”

Step 1: Convert the above sentences in lower case as the case of the word does not hold any information.
Step 2: Remove special characters and stopwords from the text. Stopwords are the words that do not contain much information about text like ‘is’, ‘a’,’the and many more’.
After applying the above steps, the sentences are changed to

Sentence 1: ”welcome machine learning now start learning”
Sentence 2: “learning good practice”
Although the above sentences do not make much sense the maximum information is contained in these words only.
Step 3: Go through all the words in the above text and make a list of all of the words in our model vocabulary.

•welcome
•machine
•learning
•now
•start
•good
•practice

For sentence 1, the count of words is as follow:

Word	Frequency
welcome	1
machine	1
learning	2
now	1
start	1
good	0
practice	0

Now for sentence 2, the scoring would be like

Word	Frequency
welcome	0
machine	0
learning	1
now	0
start	0
good	1
practice	1

Writing the above frequencies in the vector

Sentence 1 → **[1,1,2,1,1,0,0]**

Sentence 2 → **[0,0,1,0,0,1,1]**

Limitations of Bag-of-Words

1.The model ignores the location information of the word. The location information is a piece of very important information in the text.
For example "today is off" and "Is today off", have the exact same vector representation in the BoW model.

2. Bag of word models doesn't respect the semantics of the word.
For example, words 'soccer' and 'football' are often used in the same context. However, the vectors corresponding to these words are quite different in the bag of words model. The problem becomes more serious while modeling sentences. Ex: "Buy used cars" and "Purchase old automobiles" are represented by totally different vectors in the Bag-of-words model.

3. The range of vocabulary is a big issue faced by the Bag-of-Words model.
For example, if the model comes across a new word it has not seen yet, rather we say a rare, but informative word like Biblioklept(means one who steals books). The BoW model will probably end up ignoring this word as this word has not been seen by the model yet.