NLP (Natural Language Processing)

Essentials of NLP

Basic Text Analysis

```
import nltk

sentence = 'Our nation faces a profound climate crisis that is impacting every American'
```

```
# Function to check if a word exists in the text
def find_word(word, text):
    return word in text


print(find_word('nation', sentence))                    # OUTPUT: True
```

```
# Function to get the index of a word in the text
def get_index(word, text):
    words = text.split()
    return text.index(word)


print(get_index('climate', sentence))                   # OUTPUT: 5
```

```
# Function to get the word at a given index
def get_word(index, text):
    words = text.split()
    return words[index]


print(get_word(4, sentence))                           # OUTPUT: profound
```

```
# Function to concatenate the first and last words in the sentence
def concat_words(text):
    words = text.split()

    first_word = words[0]
    last_word = words[len(words)-1]

    return f'{first_word} {last_word}'


print(concat_words(sentence))                           # OUTPUT: Qur American
```

```
# Function to get the words in even positions
def get_even_position_words(text):
    words = text.split()
    return [words[i] for i in range(len(words)) if i%2==0]


print(get_even_position_words(sentence))
# OUTPUT: ['Our', 'faces', 'profound', 'crisis', 'is', 'every']
```

```
# Function to get the last 'n' letters of the text
def get_last_n_letters(text, n):
    return text[-n:]


print(get_last_n_letters(text, 3))              # OUTPUT: can
```

```
# Function to reverse the entire text
def get_reverse(text):
    return text[::-1]

print(get_reverse(sentence))
# OUTPUT: naciremA yreve gnitcapmi si taht sisirc etamilc dnuoforp a secaf noitan ruO
```

```
# Function to reverse each word in the text
def get_word_reverse(text):
    words = text.split()
    return ' '.join([word[::-1] for word in words])
```

```
print(get_word_reverse(sentence))
# OUTPUT: ruO noitan secaf a dnuoforp etamilc sisirc taht si gnitcapmi yreve naciremA
```

```
# Function to reverse the order of words in the sentence
def get_reverse_sentence(text):
    words = text.split()
    return ' '.join(words[::-1])

print(get_reverse_sentence(sentence))
# OUTPUT: American every impacting is that crisis climate profound a faces nation Our
```

Tokenization

What is Tokenization?

- **Tokenization** is the process of breaking a sentence into smaller parts, like words or phases, called unigrams. We do this to make it easier to search for specific information in the sentence.

```
import nltk
from nltk import word_tokenize
nltk.download('punkt_tab')

sentence = 'Natural Langauge processing is one of the most important concepts in the industry.'


# Create a function to tokenize the sentence into words.
def get_tokens(text):
    return word_tokenize(text)


# Call the function and print the tokens
tokens = get_tokens(sentence)
print(tokens)

# OUTPUT:
# ['Natural', 'Langauge', 'processing', 'is', 'one', 'of', 'the', 'most', 'important', 'concepts',
#  'in', 'the', 'industry' '.']
```

POS Tagging

- **POS (Part of Speech) tagging** refers to the process of tagging words within sentences with their respective POS.
- You need to learn about POS of words because it helps us to understand the true meaning of word.

```
import nltk
from nltk import word_tokenize, pos_tag
nltk.download(['punkt', 'averaged_perceptron_tagger', 'tagsets'])

sentence = 'I am learning NLP fundamentals.'

def get_tokens(text):
    return word_tokenize(text)

tokens = get_tokens(sentence)
print(tokens)             # ['I', 'am', 'learning', 'NLP', 'fundamentals', '.']

def get_pos(tokens):
    return pos_tag(tokens)

# Get part of speech tags
pos_tags = get_pos(tokens)
print(pos_tags)

# OUTPUT:
# [('I', 'PRP'), ('am', 'VBP'), ('learning', 'VBG'), ('NLP', 'NNP'), ('fundamentals', 'NNS'),
# ('.', '.')]

# Print all the POS tags and their descriptions
print(nltk.help.upenn_tagset())
```

```
CC    Coordinating conjunction
CD    Cardinal number
DT    Determiner
EX    Existential there
FW    Foreign word
IN    Preposition or subordinating conjunction
JJ    Adjective
JJR   Adjective, comparative
JJS   Adjective, superlative
LS    List item marker
MD    Modal verb
NN    Noun, singular or mass
NNS   Noun, plural
NNP   Proper noun, singular
NNPS  Proper noun, plural
```

```
PDT   Predeterminer
POS   Possessive ending
PRP   Personal pronoun
PRP$  Possessive pronoun
RB    Adverb
RBR   Adverb, comparative
RBS   Adverb, superlative
RP    Particle
SYM   Symbol
TO    to
UH    Interjection
VB    Verb, base form
VBD   Verb, past tense
VBG   Verb, gerund or present participle
VBN   Verb, past participle
VBP   Verb, non-3rd person singular present
VBZ   Verb, 3rd person singular present
WDT   Wh-determiner
WP    Wh-pronoun
WP$   Possessive wh-pronoun
WRB   Wh-adverb
```

Punctuation Removal

- **Punctuation removal** is the process of eliminating punctuation marks (like periods, commas, exclamation points, etc.) from a text.
- This is often done in text processing to focus only on the words and simplify analysis.

```
import nltk
from nltk.tokenize import word_tokenize
from string import punctuation
nltk.download(['punkt_tab'])


sentence = '''Natural Language Processing (NLP) has revolutionized industries like healthcare,
            finance, and customer service, especially in AI-driven applications!'''


# Create a function to tokenize the sentence into words and remove punctuation.
def remove_punctuation(text):
    return ' '.join([word for word in word_tokenize(text) if word not in punctuation])

updated_sentence = remove_punctuation(sentence)
print(updated_sentence)

# OUTPUT:
# Natural Language Processing NLP has revolutionized industries like healthcare finance and
# customer service especially in AI-driven applications

print(punctuation)                # !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Stop Word Removal

- **Stop word removal** is the process of eliminating common words (like "the," "is," "and," etc.) from text, as they don't add significant meaning for analysis.
- This helps focus on more important words, improving the efficiency of text processing and analysis.

```
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
nltk.download(['punkt_tab', 'stopwords'])           # resouces

# List of stopwords
stop_words = stopwords.words('english')

sentence = 'I am learning Python. It is one of the most popular programming language'

# Function to remove stop words
def remove_stop_words(text, stopwords):
    return ' '.join([for word in word_tokenize(text) if word not in stopwords])


updated_sentence = remove_stop_words(sentence)
print(updated_sentence, stop_words)
# OUTPUT: I learning Python . It one popular programming language


# Extend stop_words list with custom words
stop_words.extend(['I', 'It', 'one'])


updated_sentence2 = remove_stop_words(sentence, stop_words)
print(updated_sentence2)
# OUTPUT: learning Python . popular programming language
```

Text Normalization

- **Text normalization** is the process of converting variations of words or text into a standard format. It helps to ensure that different forms of a word or phrase are treated as the same, making analysis more consistent and effective.

```
sentence = 'I visited the US from the UK on 24-10-18'

def normalize(text):
    return text.replace('US', 'United States').replace('UK', 'United Kingdom').replace('18', '2018')

normalized_sentence =  normalize(sentence)
print(normalized_sentence)
# OUTPUT: I visited the United States from the United Kingdom on 24-10-2018
```

Spelling Correction

- **Spelling correction** helps identify and fix misspelled words in a document or sentence. It uses a method called a "speller" to automatically suggest the correct spelling based on context and known word patterns.
- Speller compares a misspelled word to an online dictionary and finds the closest matching word. If there are multiple close matches, it suggests the most likely correction.

```
pip install autocorrect
```

```
import nltk
from nltk import word_tokenize
from autocorrect import Speller
nltk.download('punkt_tab')

spell = Speller(lang='en')

sentence = '''Ntural Luanguage Processin deals with the art of extracting insightes from Natural
           Languaes'''


# Function to correct spelling
def correct_spelling(text):
    return ' '.join([spell(word) for word in word_tokenize(text)])


corrected_sentence = correct_spelling(sentence)
print(corrected_sentence)
# OUTPUT: Natural Language Processing deals with the art of extracting insights from Natural Languages
```

Stemming

- **Stemming** is the process of reducing words to their root form by removing variations (e.g., "running" becomes "run").
- Types of Stemmer
  - Porter Stemmer : Reduces words to their base form using simple rules.
  - Snowball Stemmer : Improved version of Porter, more accurate for multiple languages.
  - Lancaster Stemmer : Aggressive stemmer that gives shorter roots.
  - Regexp Stemmer : Uses regular expressions to match and transform words into their root form.

```
import nltk
from nltk import word_tokenize
from nltk.stem import PorterStemmer
nltk.download('punkt_tab')

sentence = 'production products coming firing battling'

# Initialize the Porter Stemmer
stemmter = PorterStemmer()


# Function to get the stemmed version of the text
def get_stem(text, stemmter):
    return ' '.join([stemmter.stem(word) for word in word_tokenize(text)])


stemed_sentence = get_stem(sentence, stemmter)
print(stemed_sentence)
# OUTPUT: product product come fire battl
```

- Stemming reduces word variations to their root form, helping to simplify text. However, it can sometimes create misspellings by removing prefixes and suffixes.

Lemmatization

- **Lemmatization** fixes the inaccuracies of stemming by considering the word's meaning and context.
- It is slower than stemming but provides more accurate results.

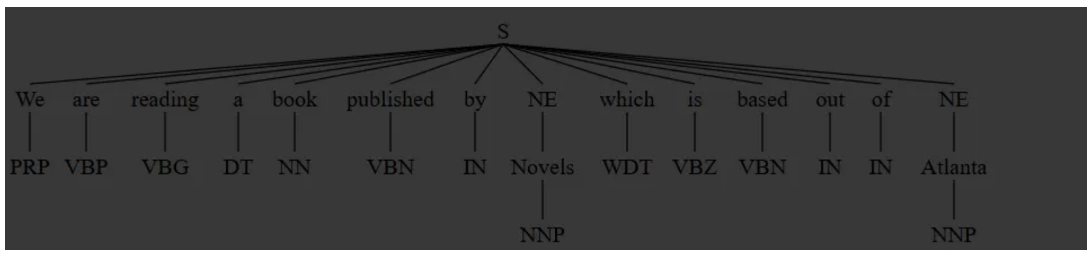
```
import nltk
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
```

```
nltk.download(['punkt_tab', 'wordnet'])

sentence = 'production products coming firing battling'

stemmter = PorterStemmer()
lemmatizer = WordNetLemmatizer()


# Function to get the stemmed version of the text
def get_stem(text, stemmter):
    return ' '.join([stemmter.stem(word) for word in word_tokenize(text)])


stemed_sentence = get_stem(sentence, stemmter)
print(stemed_sentence)
# OUTPUT: product product come fire battl


# Function to get the lemmatized version of the text
def get_lem(text):
    return ' '.join([lemmatizer.lemmatize(word) for word in word_tokenize(stemed_sentence)])


lemmatized_sentence = get_lem(sentence)
print(lemmatized_sentence)
# OUTPUT: product product come fire battle
```

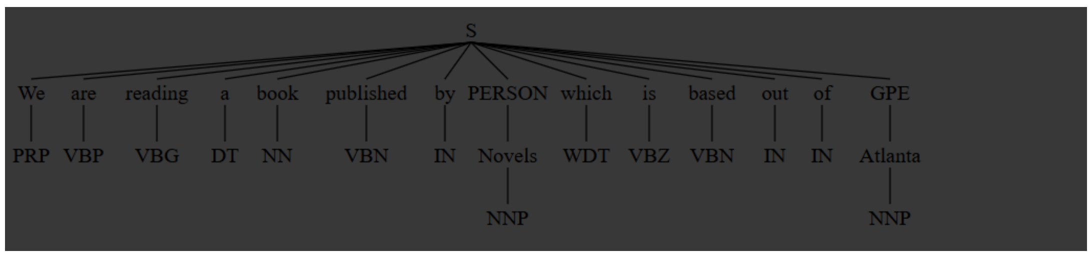
Named Entity Recognition (NER)

- **NER** is the process of extracting important entities such as person names, place names, organization names, etc.
- It helps us to understand the true meaning of a given words rather than understanding its grammatical meaning through POS tagging.

```
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk
nltk.download(['punkt_tab', 'maxent_ne_chunker_tab', 'words', 'averaged_perceptron_tagger_eng'])

# pip install svgling

sentence = 'We are reading a book published by Novels which is based out of Atlanta'

# Function to perform Named Entity Recognition (NER)
def get_NER(text):
    return ne_chunk(pos_tag(word_tokenize(text)), binary=True)

print(get_NER(sentence))
```

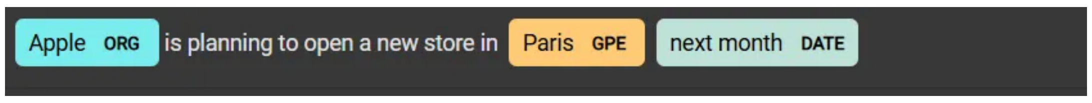


```
def get_NER2(text):
  return ne_chunk(pos_tag(word_tokenize(text)), binary=False)

print(get_NER2(sentence))

# binary=False shows more detailed entities
```



```
# Import Spacy and displacy
import spacy
from spacy import displacy

# Using SpaCy for NER visualization
nlp = spacy.load('en_core_web_sm')

# Text to be processed with SpaCy
para = nlp(u"""Apple is planning to open a new store in Paris next month""")
```

```
# Visualize NER in Jupyter notebook using displacy
displacy.render(para, style='ent', jupyter=True)
```



Word Sense Disambiguation

- **Word Sense Disambiguation** helps determine a word's meaning based on its context and surrounding words. The **Lesk algorithm** is used for this, relying on a large corpus and the online dictionary WordNet.

```
import nltk
from nltk import word_tokenize
from nltk.wsd import lesk
download(['punkt_tab', 'wordnet'])

# Sentences with the word 'bank' having different meanings
sentence1 = 'Keep your savings in the bank'
sentence2 = "It's so risky to drive over the banks of the road"


# Function to get the synset for a given word
def get_synset(text, word):
    return lesk(word_tokenize(text), word)

# Get the synset for 'bank' in sentence 1
print(get_synset(sentence1, 'bank'))

# Get the synset for 'bank' in sentence 2
print(get_synset(sentence2, 'bank'))

# OUTPUT:
Synset('savings_bank.n.02')  # Referring to a financial bank (e.g., "savings bank")
Synset('bank.n.06')  # Referring to the edge of a river or road (e.g., "the banks of the road")
```

Sentence Boundary Detection

- **Sentence Boundary Detection** is the method of identifying where a sentence ends. While punctuation may seem like the best way to detect sentence endings, it can be misleading for machines, as punctuation is also used in other contexts.

```
import nltk
from nltk.tokenize import sent_tokenize
nltk.download(['punkt_tab'])

sentence1 = """You've obsessed over it, dreamed about it, saved up for it. Well, guess what? Black
            Friday weekneed is NOW! There's a mind-bemding array of sales to take advantage of,
            and Amazon is, of course, at the center of the action."""
sentence2 = "Mr. Donald Trump was a president of the USA. Before joining politics,
             he was a businessman.""""

def get_sentences(text):
    return sent_tokenize(text)

print(get_sentences(sentence1))
print(get_sentences(sentence2))

# OUTPUT:
["You've obsessed over it, dreamed about it, saved up for it.",
 'Well, guess what?',
 'Black Friday weekneed is NOW!',
 "There's a mind-bemding array of sales to take advantage of, and Amazon is, of course, at the center
 of the action."]

 ['Mr. Donald Trump was a president of the USA.',
 'Before joining politics, he was a businessman.']
```