Vidyavardhini's College of Engineering and Technology, Vasai

Department of Computer Science & Engineering (Data Science)

AY: 2025-26

Class:	**BE-CSE(DS)**	**Semester:**	**VII**
Course Code:	**CSDOL7011**	**Course Name:**	**NLP Lab**

Name of Student:	**Sahil Salunke**
Roll No. :	**45**
Experiment No.:	**5**
Title of the Experiment:	**Performing Part-of-Speech Tagging and Syntactic Analysis using NLTK**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty :

Signature :

Date :

Aim: To perform Part-of-Speech tagging on sentences using NLTK and understand syntactic categories of words.

Objective: • To apply Part-of-Speech tagging for syntactic analysis of sentences using NLTK.

Tools Required:

1. Python (Jupyter Notebook or Google Colab)
2. nltk

Procedure:

1. Install and import libraries:
    a. import nltk
    b. Run nltk.download('punkt') and nltk.download('averaged_perceptron_tagger')

2. Input or define a sample sentence.

3. Tokenize the sentence into words:
    a. Use nltk.word_tokenize(sentence)

4. Apply POS tagging:
    a. Use nltk.pos_tag(tokens) to assign part-of-speech tags to each token.

5. Display the results:
    a. Print each word along with its corresponding POS tag.

6. Optional: Visualize the tagged structure using nltk.tree.Tree or nltk.ne_chunk().

Description of the Experiment:

This experiment introduces POS tagging, where each word in a sentence is labeled with its grammatical category. It helps in syntactic understanding of the sentence structure and prepares students for further syntactic and semantic parsing tasks.

Detailed Description of the NLP Technique:

Part-of-Speech (POS) Tagging:

POS tagging is the process of assigning a grammatical category (like noun, verb, adjective, etc.) to each word in a sentence.

POS Tags Examples (Penn Treebank Tagset):

NN: Noun

VB: Verb (base form)

JJ: Adjective

RB: Adverb IN:

Preposition

PRP: Pronoun

DT: Determiner

Why POS Tagging is Important:

- Enables syntactic parsing.
- Helps in understanding sentence structure.

- Aids downstream tasks like Named Entity Recognition (NER), chunking, parsing, and machine translation.

Techniques Used in POS Tagging:

- Rule-based taggers: Apply hand-written rules to assign tags.
- Statistical taggers: Use models like Hidden Markov Models (HMMs).
- Machine learning-based taggers: Train classifiers (e.g., Maximum Entropy, CRF).

NLTK Tagger:

- The nltk.pos_tag() function uses a pre-trained Averaged Perceptron tagger.
- It uses the context of the word and its features to assign the most probable POS tag.

CODE AND OUTPUT:

1. Install and import libraries

```
import nltk

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

2. Input or define a sample sentence

```
[9]  sentence = "The quick brown fox jumps over the lazy dog."
```

3. Tokenize the sentence into words

```
[10] tokens = nltk.word_tokenize(sentence)
```

4. Apply POS tagging

```
pos_tags = nltk.pos_tag(tokens)
```

5. Display the results

```
[12] for word, tag in pos_tags:
         print(f"{word} → {tag}")
```

```
The → DT
quick → JJ
brown → NN
fox → NN
jumps → VBZ
over → IN
the → DT
lazy → JJ
dog → NN
. → .
```

6. Visualize the tagged structure

```
[15] nltk.download('maxent_ne_chunker')
     nltk.download('maxent_ne_chunker_tab')
     nltk.download('words')
```

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package maxent_ne_chunker_tab to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker_tab.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
True
```

```
[17] from nltk import ne_chunk
     from nltk.tree import Tree
```

```
[18] tree = ne_chunk(pos_tags)
     print(tree)
```

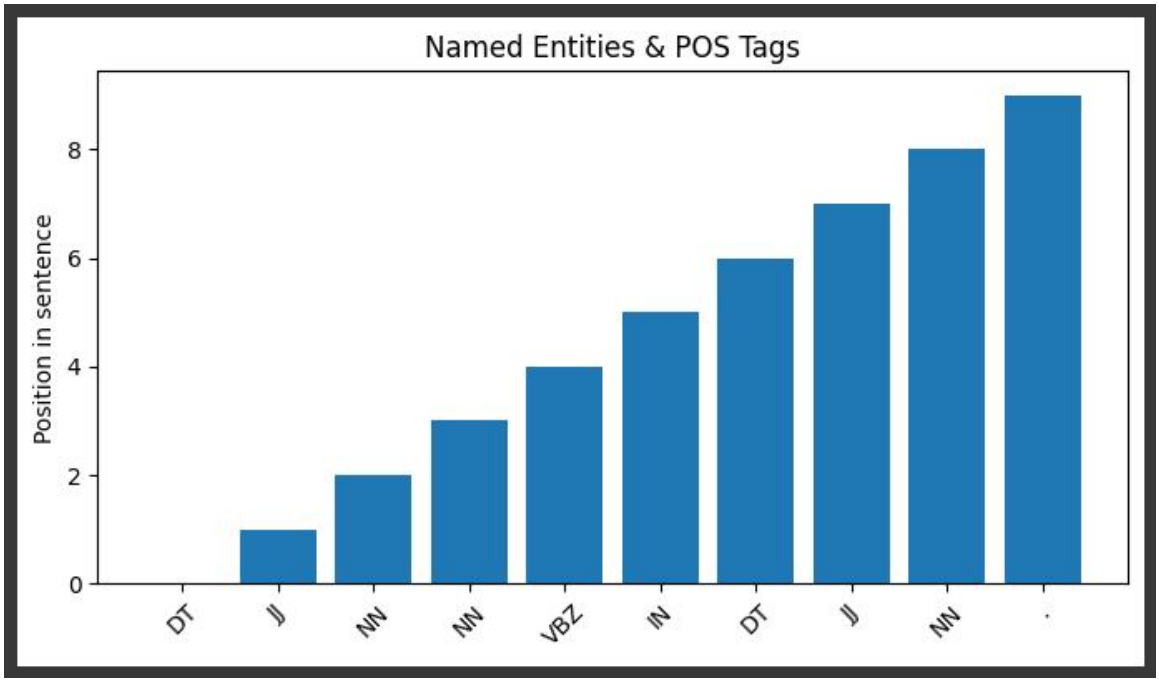
```
(S
  The/DT
  quick/JJ
  brown/NN
  fox/NN
  jumps/VBZ
  over/IN
  the/DT
  lazy/JJ
  dog/NN
  ./.)
```

```
[21] entities = []
     for subtree in tree:
         if isinstance(subtree, Tree):
             entity_text = " ".join([token for token, pos in subtree.leaves()])
             entities.append((entity_text, subtree.label()))
         else:
             token, tag = subtree
             entities.append((token, tag))
```

```
[20] import matplotlib.pyplot as plt
```

```
[22] words, labels = zip(*entities)

[23] plt.figure(figsize=(8, 4))
     plt.bar(words, range(len(words)), tick_label=labels)
     plt.xticks(rotation=45)
     plt.ylabel("Position in sentence")
     plt.title("Named Entities & POS Tags")
     plt.show()
```



Conclusion:

Correct POS Tagging – Common parts of speech like nouns (NN), verbs (VB), adjectives (JJ), and determiners (DT) are accurately assigned, showing that nltk.pos_tag() correctly identified the grammatical role of each token.

Entity Detection – Named entities such as people, locations, or organizations are grouped and labeled (PERSON, ORGANIZATION, GPE), demonstrating the capability of nltk.ne_chunk() to extract meaningful real-world references from plain text.

Non-Entity Tokens – Words that are not named entities are preserved with their original POS tags, so the context of the sentence is not lost.

Structured Representation – The hierarchical tree (or extracted entity list) makes the relationship between words and entities clear, which is useful for downstream NLP tasks like question answering or information extraction.

Limitations – Since NER is rule-based in NLTK, it may miss ambiguous or rare entities, and performance could vary depending on sentence complexity.