**Lab 4**

**POS Tagging, Chunking, Named Entity Recognition (NER)**

**POS tagging**

* POS Tagging is the process of tagging words in a sentence with corresponding parts of speech like noun, pronoun, verb, adverb, preposition, etc.
* Tagging the words of a text with parts of speech helps to understand how the word functions grammatically in the context of the sentence.
* A word can assume different parts of speech depending on the context of the sentence.
* POS Tagging is useful in sentence parsing, information retrieval, sentiment analysis, etc.
* it is a prerequisite for the process of Chunking and Named Entity Recognition in NLP.
* POS Tagging in NLTK library is done using pos\_tag() function which takes the tokens of a sentence as input and it returns the POS tag for each word.
* NLTK actually provides many categories and sub-categories of tags
* Below is the complete list of NLTK POS tag

|  | **Pos\_tag** | **tag\_name** | **example** |
| --- | --- | --- | --- |
| **0** | $ | dollar | [, −, −, —, A, A, C, HK, HK, M, NZ, NZ, S, U.S., U.S, … |
| **1** | ” | closing quotation mark | [‘, ”] |
| **2** | ( | opening parenthesis | [(, [, {] |
| **3** | ) | closing parenthesis | [), ], }] |
| **4** | , | comma | [,] |
| **5** | — | dash | [–] |
| **6** | . | sentence terminator | [., !, ?] |
| **7** | : | colon or ellipsis | [:, ;, …] |
| **8** | CC | conjunction, coordinating | [&, ‘n, and, both, but, either, et, for, less,… |
| **9** | CD | numeral, cardinal | [mid-1890, nine-thirty, forty-two, one-tenth, … |
| **10** | DT | determiner | [all, an, another, any, both, del, each, eithe… |
| **11** | EX | existential there | [there] |
| **12** | FW | foreign word | [gemeinschaft, hund, ich, jeux, habeas, Haemen… |
| **13** | IN | preposition or conjunction, subordinating | [astride, among, uppon, whether, out, inside, … |
| **14** | JJ | adjective or numeral, ordinal | [third, ill-mannered, pre-war, regrettable, oi… |
| **15** | JJR | adjective, comparative | [bleaker, braver, breezier, briefer, brighter,… |
| **16** | JJS | adjective, superlative | [calmest, cheapest, choicest, classiest, clean… |
| **17** | LS | list item marker | [A, A., B, B., C, C., D, E, F, First, G, H, I,… |
| **18** | MD | modal auxiliary | [can, cannot, could, couldn’t, dare, may, migh… |
| **19** | NN | noun, common, singular or mass | [common-carrier, cabbage, knuckle-duster, Casi… |
| **20** | NNP | noun, proper, singular | [Motown, Venneboerger, Czestochwa, Ranzer, Con… |
| **21** | NNPS | noun, proper, plural | [Americans, Americas, Amharas, Amityvilles, Am… |
| **22** | NNS | noun, common, plural | [undergraduates, scotches, bric-a-brac, produc… |
| **23** | PDT | pre-determiner | [all, both, half, many, quite, such, sure, this] |
| **24** | POS | genitive marker | [‘, ‘s] |
| **25** | PRP | pronoun, personal | [hers, herself, him, himself, hisself, it, its… |
| **26** | PRP$ | pronoun, possessive | [her, his, mine, my, our, ours, their, thy, your] |
| **27** | RB | adverb | [occasionally, unabatingly, maddeningly, adven… |
| **28** | RBR | adverb, comparative | [further, gloomier, grander, graver, greater, … |
| **29** | RBS | adverb, superlative | [best, biggest, bluntest, earliest, farthest, … |
| **30** | RP | particle | [aboard, about, across, along, apart, around, … |
| **31** | SYM | symbol | [%, &, ‘, ”, ”., ), )., \*, +, ,., <, =, >, @… |
| **32** | TO | “to” as preposition or infinitive marker | [to] |
| **33** | UH | interjection | [Goodbye, Goody, Gosh, Wow, Jeepers, Jee-sus, … |
| **34** | VB | verb, base form | [ask, assemble, assess, assign, assume, atone,… |
| **35** | VBD | verb, past tense | [dipped, pleaded, swiped, regummed, soaked, ti… |
| **36** | VBG | verb, present participle or gerund | [telegraphing, stirring, focusing, angering, j… |
| **37** | VBN | verb, past participle | [multihulled, dilapidated, aerosolized, chaire… |
| **38** | VBP | verb, present tense, not 3rd person singular | [predominate, wrap, resort, sue, twist, spill,… |
| **39** | VBZ | verb, present tense, 3rd person singular | [bases, reconstructs, marks, mixes, displeases… |
| **40** | WDT | WH-determiner | [that, what, whatever, which, whichever] |
| **41** | WP | WH-pronoun | [that, what, whatever, whatsoever, which, who,… |
| **42** | WP$ | WH-pronoun, possessive | [whose] |
| **43** | WRB | Wh-adverb | [how, however, whence, whenever, where, whereby… |
| **44** | “ | opening quotation mark | [`, “] |

**POS Tagging in NLTK**

In [1]:

from nltk import pos\_tag

from nltk import word\_tokenize

text = "The way to get started is to quit talking and begin doing."

tokenizer = word\_tokenize(text)

pos\_tag(tokenizer)

[Out] :

[('The', 'DT'),

('way', 'NN'),

('to', 'TO'),

('get', 'VB'),

('started', 'VBN'),

('is', 'VBZ'),

('to', 'TO'),

('quit', 'VB'),

('talking', 'VBG'),

('and', 'CC'),

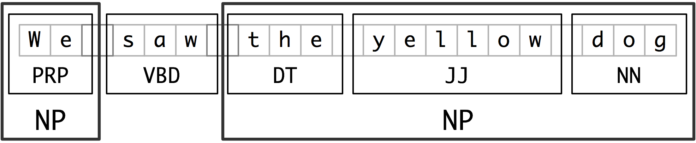
('begin', 'VB'),

('doing', 'VBG'),

('.', '.')]

**Chunking in NLP**

* We can break down a sentence into tokens of words and then do POS tagging for identifying parts of speech for those words
* In NLP, chunking is the process of breaking down a text into phrases such as Noun Phrases, Verb Phrases, Adjective Phrases, Adverb phrases, and Preposition Phrases.
* Chunking is essential for understanding the semantics of the text and helps in information retrieval.



The process of chunking in NLTK is a multi-step process as explained below –

**Step1 :**

Tokenize the sentence and perform POS Tagging.

**Step 2:**

Define the grammar to perform chunking. This is a very important step because grammar lays the rule of chunking.

**Step 3:**

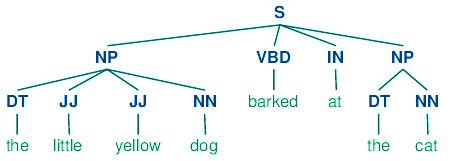
Using this grammar, we create a chunk parser with the help of RegexpParser and apply it to our sentence.

**Step 4:**

The above step produces the result which can either be printed as it is or we can draw a graph for better visualization.

**Example of Chunking in NLTK**

* We first tokenize the sample sentence and perform POS Tagging on it.
* Then we define the grammar for Noun Phrase as  *NP: {<DT>?<JJ>\*<NN>} which means -*
* a chunk will be constructed when an optional Determiner (DT) is followed by any number of Adjective (JJ) or Noun (NN).
* Then initialize an instance of nltk.RegexpParser() with this grammar and use it to parse the tokenized sample sentence.
* import nltk
* sentence = "the little yellow dog barked at the cat"
* tokens = nltk.word\_tokenize(sentence)
* print(tokens)
* tag = nltk.pos\_tag(tokens)
* print(tag)
* grammar = "NP: {<DT>?<JJ>\*<NN>}"
* cp = nltk.RegexpParser(grammar)
* result = cp.parse(tag)
* print(result)
* result.draw()
* [Out] :
* ['the', 'little', 'yellow', 'dog', 'barked', 'at', 'the', 'cat']
* [('the', 'DT'), ('little', 'JJ'), ('yellow', 'JJ'), ('dog', 'NN'), ('barked', 'VBD'), ('at', 'IN'), ('the', 'DT'), ('cat', 'NN')]
* (S
* (NP the/DT little/JJ yellow/JJ dog/NN)
* barked/VBD
* at/IN
* (NP the/DT cat/NN))



**Named Entity Recognition**

**Named Entity**

* Named entities are proper nouns that refer to specific entities that can be a person, organization, location, date, etc.
* Consider this example – *“Mount Everest is the tallest mountain”*.
* Here *Mount Everest* is a named entity of type location as it refers to a specific entity.

**Named Entity Recognition**

In information retrieval and natural language processing, Named Entity Recognition (NER) is the process of extracting Named Entities from the text.

**Uses of Named Entity Recognition**

* The field of academics by easy and faster extraction of information for the students and researchers from the searching data.
* In Question Answer system to provide answers from the data by the machine and hence minimizing human efforts.
* In content [classification](https://machinelearningknowledge.ai/glossary/classification/) by identifying the theme and subject of the contents and makes the process faster and easy, suggesting the best content of interest.
* **Example -1**

In the below example of named entity recognition in NLTK, we have taken a text from times of India and have applied tokenization and POS tagging to the text.

NLTK provides a function nltk.ne\_chunk() that is already a pre-trained classifier to recognize named entity using POS tag as input.

In the output, we can see that the classifier has added category labels such as PERSON, ORGANIZATION, and GPE (geographical physical location) where ever it founded named entity.

In [1]:

import nltk

from nltk import word\_tokenize,pos\_tag

text = "NASA awarded Elon Musk’s SpaceX a $2.9 billion contract to build the lunar lander."

tokens = word\_tokenize(text)

tag=pos\_tag(tokens)

print(tag)

ne\_tree = nltk.ne\_chunk(tag)

print(ne\_tree)

In the output, we can see that the classifier has added category labels such as PERSON, ORGANIZATION, and GPE (geographical physical location) where ever it founded named entity.

[Out] :

[('NASA', 'NNP'), ('awarded', 'VBD'), ('Elon', 'NNP'), ('Musk', 'NNP'), ('’', 'NNP'), ('s', 'VBD'), ('SpaceX', 'NNP'), ('a', 'DT'), ('$', '$'), ('2.9', 'CD'), ('billion', 'CD'), ('contract', 'NN'), ('to', 'TO'), ('build', 'VB'), ('the', 'DT'), ('lunar', 'NN'), ('lander', 'NN'), ('.', '.')]

(S

(ORGANIZATION NASA/NNP)

awarded/VBD

(PERSON Elon/NNP Musk/NNP)

’/NNP

s/VBD

(ORGANIZATION SpaceX/NNP)

a/DT

$/$

2.9/CD

billion/CD

contract/NN

to/TO

build/VB

the/DT

lunar/NN

lander/NN

./.)

1. **Example -2**

Let us see one more example where we have used already present tagged sentences provided by the NLTK library.

In [2]:

>>> sent = nltk.corpus.treebank.tagged\_sents()

>>> print(nltk.ne\_chunk(sent[0]))

[Out] :

(S

(PERSON Pierre/NNP)

(ORGANIZATION Vinken/NNP)

,/,

61/CD

years/NNS

old/JJ

,/,

will/MD

join/VB

the/DT

board/NN

as/IN

a/DT

nonexecutive/JJ

director/NN

Nov./NNP

29/CD

./.)