NLP Core using NLTK

Dr. Muhammad Nouman Durrani

Tokenization

- The process of breaking down a text paragraph into smaller chunks such as words or sentences is called Tokenization
- Token is a single entity that is building blocks for sentence or paragraph

Sentence Tokenization

- Sentence tokenizer breaks text paragraph into sentences

```
from nltk.tokenize import sent_tokenize
text="""Hello Mr. Smith, how are you doing today? The weather is great, and city is awesome.
The sky is pinkish-blue. You shouldn't eat cardboard"""
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Hello Mr. Smith, how are you doing today?', 'The weather is great, and city is awesome.', 'The sky is pinkish-blue.', "You shouldn't eat cardboard"]
```

Tokenization

Word Tokenization

- Word tokenizer breaks text paragraph into words

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',',
'and', 'city', 'is', 'awesome', '.', 'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat',
'cardboard']
```

Tokenization

- **Frequency Distribution**

```
from nltk.probability import FreqDist
fdist = FreqDist(tokenized_word)
print(fdist)
```

```
<FreqDist with 25 samples and 30 outcomes>
```

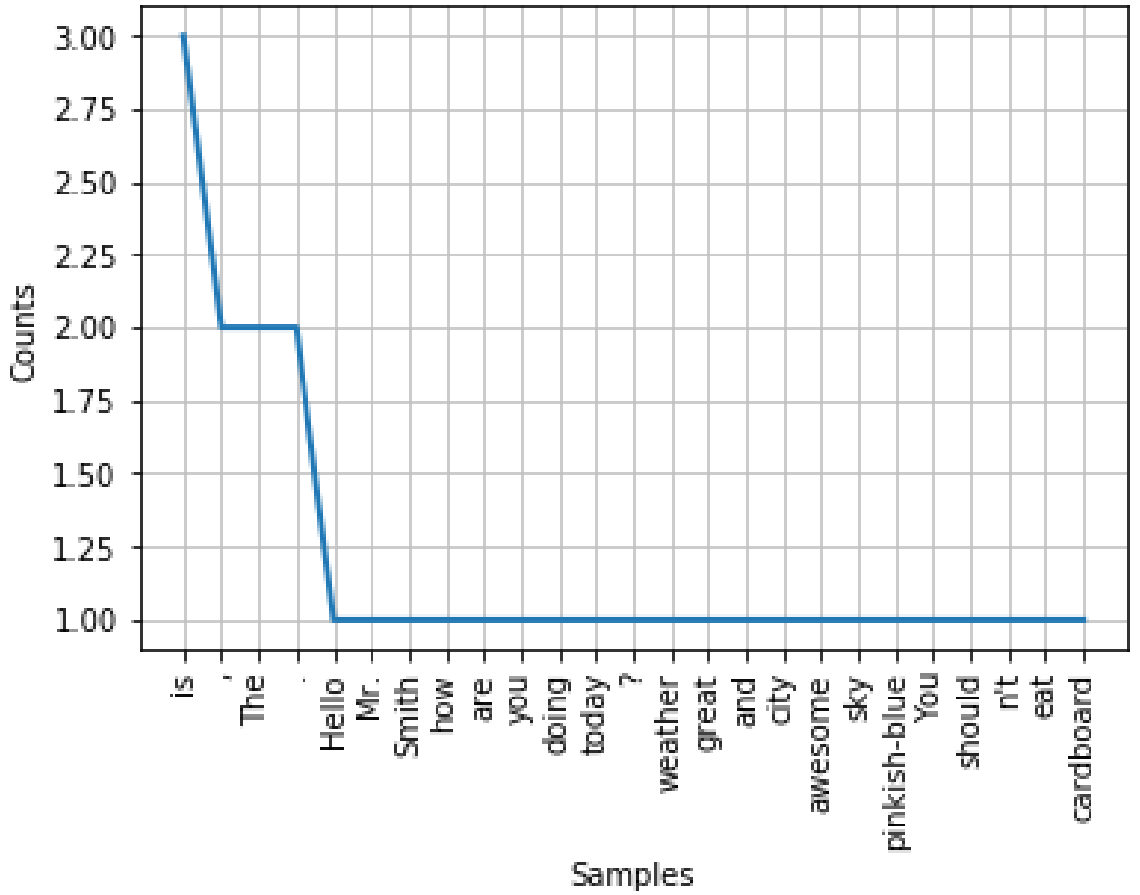
```
fdist.most_common(2)

[('is', 3), (',', 2)]
```

Tokenization

- Frequency Distribution Plot

```
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```



Tokenize Non-English Languages Text

- To tokenize other languages, you can specify the language like this:

```
from nltk.tokenize import sent_tokenize

mytext = "Bonjour M. Adam, comment allez-vous? J'espère que tout va bien. Aujourd'hui est un bon jour."
print(sent_tokenize(mytext"french"))
```

The result will be like this:

```
['Bonjour M. Adam, comment allez-vous?', "J'espère que tout va bien.", "Aujourd'hui est un bon jour."]
```

Stopwords

- Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc.

- In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'their', 'then', 'not', 'ma', 'here', 'other', 'won', 'up', 'weren', 'being', 'we', 'those', 'an', 'them', 'which', 'him', 'so', 'yourselves', 'what', 'own', 'has', 'should', 'above', 'in', 'myself', 'against', 'that', 'before', 't', 'just', 'into', 'about', 'most', 'd', 'where', 'our', 'or', 'such', 'ours', 'of', 'doesn', 'further', 'needn', 'now', 'some', 'too', 'hasn', 'more', 'the', 'yours', 'her', 'below', 'same', 'how', 'very', 'is', 'did', 'you', 'his', 'when', 'few', 'does', 'down', 'yourself', 'i', 'do', 'both', 'shan', 'have', 'itself', 'shouldn', 'through', 'themselves', 'o', 'didn', 've', 'm', 'off', 'out', 'but', 'and', 'doing', 'any', 'nor', 'over', 'had', 'because', 'himself', 'theirs', 'me', 'by', 'she', 'whom', 'hers', 're', 'hadn', 'who', 'he', 'my', 'if', 'will', 'are', 'why', 'from', 'am', 'with', 'been', 'its', 'ourselves', 'ain', 'couldn', 'a', 'aren', 'under', 'll', 'on', 'y', 'can', 'they', 'than', 'after', 'wouldn', 'each', 'once', 'mightn', 'for', 'this', 'these', 's', 'only', 'haven', 'having', 'all', 'don', 'it', 'there', 'until', 'again', 'to', 'while', 'be', 'no', 'during', 'herself', 'as', 'mustn', 'between', 'was', 'at', 'your', 'were', 'isn', 'wasn'}
```

Stopwords

```
filtered_sent=[]
for w in tokenized_sent:
    if w not in stop_words:
        filtered_sent.append(w)
print("Tokenized Sentence:",tokenized_sent)
print("Filterd Sentence:",filtered_sent)
```

```
Tokenized Sentence: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?']
Filterd Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?']
```

Get Synonyms From WordNet

- WordNet is a database built for natural language processing
- It includes groups of synonyms and a brief definition

```
from nltk.corpus import wordnet
syn = wordnet.synsets("pain")
print(syn[0].definition())
print(syn[0].examples())
```

```
a symptom of some physical hurt or disorder

['the patient developed severe pain and distension']
```

Get Synonyms From WordNet

- You can use WordNet to get synonymous words like this:

```
from nltk.corpus import wordnet
synonyms = []
for syn in wordnet.synsets('Computer'):
for lemma in syn.lemmas():
synonyms.append(lemma.name())
print(synonyms)
```

The output is:

['computer', 'computing_machine', 'computing_device', 'data_processor', 'electronic_computer', 'information_processing_system', 'calculator', 'reckoner', 'figurer', 'estimator', 'computer']

Get Antonyms From WordNet

- You can get the antonyms of words the same way
- Use the lemmas before adding them to the array
- it's an antonym or not

```
from nltk.corpus import wordnet
antonyms = []
for syn in wordnet.synsets("small"):
    for l in syn.lemmas():
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())
print(antonyms)
```

```
['large', 'big', 'big']
```

NLTK Word Stemming

- Word stemming means removing affixes from words and returning the root word. (The stem of the word working is work.)
- Search engines use this technique when indexing pages, so many people write different versions for the same word and all of them are stemmed to the root word
- NLTK has a class called PorterStemmer that uses this algorithm.

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
print(stemmer.stem('working'))
```

The result is: work.

Lemmatizing Words Using WordNet

- Word lemmatizing is similar to stemming, but the difference is the result of lemmatizing is a real word

When we stem some words, it will result as follows:

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
print(stemmer.stem('increases'))
```

The result is: increas.

When we lemmatize the same word using NLTK WordNet, the result is increase:

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

The result is increase.

Lemmatizing Words Using WordNet

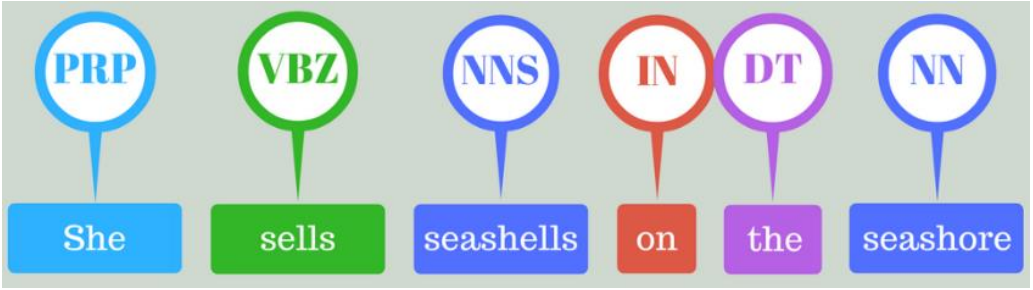
- If we try to lemmatize a word like “playing”, it will end up with the same word
  - This is because the default part of speech is nouns
  - To get verbs, adjective, or adverb, we should specify it (See Example)
  - Actually, this is a very good level of text compression.
  - We end up with about 50% to 60% compression

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('playing', pos="v"))
print(lemmatizer.lemmatize('playing', pos="n"))
print(lemmatizer.lemmatize('playing', pos="a"))
print(lemmatizer.lemmatize('playing', pos="r"))
```

The result is:

```
play
playing
playing
playing
```

Part of speech tagging (POS)



- Part-of-speech tagging is used to assign parts of speech to each word of a given text (such as nouns, verbs, pronouns, adverb, conjunction, adjectives, interjection) based on its definition and its context.

```
text = “vote to choose a particular man or a group (party) to represent them in parliament”
tex = word_tokenize(text)  #Tokenize the text
for token in tex:
print(nltk.pos_tag([token]))
```

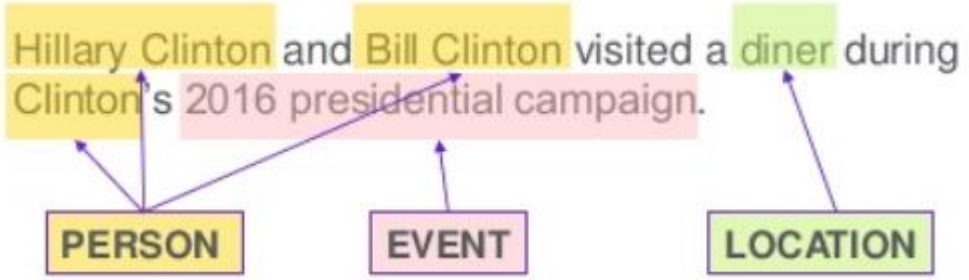
```
[('vote', 'NN')]
[('to', 'TO')]
[('choose', 'NN')]
[('a', 'DT')]
[('particular', 'JJ')]
[('man', 'NN')]
[('or', 'CC')]
[('a', 'DT')]
[('group', 'NN')]
[('(', '(')]
[('party', 'NN')]
[(')', ')')]
[('to', 'TO')]
[('represent', 'NN')]
[('them', 'PRP')]
[('in', 'IN')]
[('parliament', 'NN')]
```

POS : Tags and Descriptions

Tag	Description
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
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBR	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, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WP$	Possessive whpronoun
WRB	Whadverb

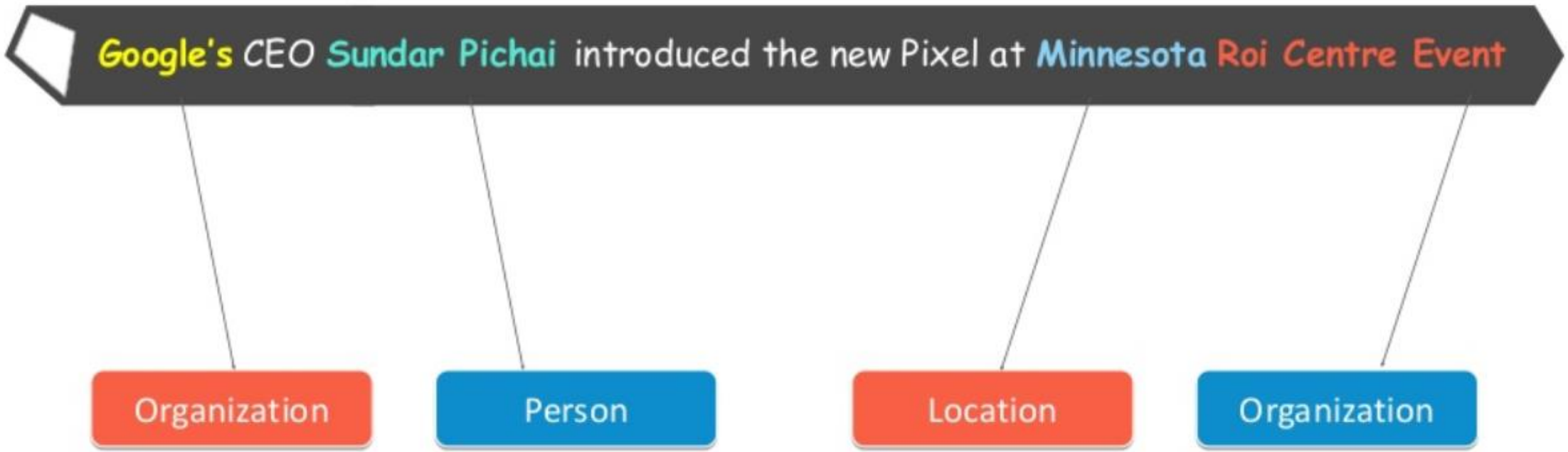
Named entity recognition



- It is the process of detecting the named entities such as the person name, the location name, the company name, the quantities and the monetary value.

```
text = "Google's CEO Sundar Pichai introduced the new Pixel at Minnesota Roi Centre Event"          #importing chunk library from nltk
from nltk import ne_chunk          # tokenize and POS Tagging before doing chunk
token = word_tokenize(text)
tags = nltk.pos_tag(token)
chunk = ne_chunk(tags)
chunk
```

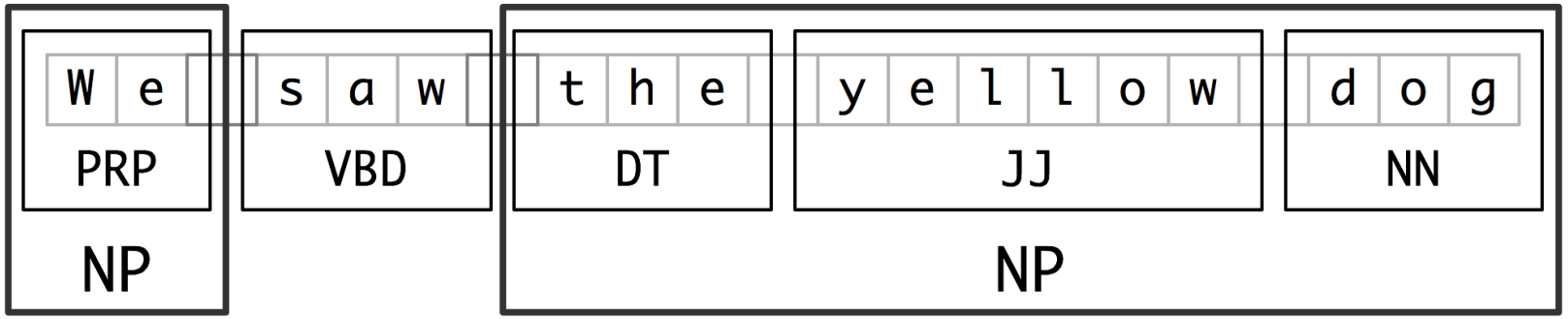
NER : Named Entity Recognition



POS Tagging Output:

Tree('S', [Tree('GPE', [('Google', 'NNP')]), ("'s", 'POS'), Tree('ORGANIZATION', [('CEO', 'NNP'), ('Sundar', 'NNP'), ('Pichai', 'NNP')]), ('introduced', 'VBD'), ('the', 'DT'), ('new', 'JJ'), ('Pixel', 'NNP'), ('at', 'IN'), Tree('ORGANIZATION', [('Minnesota', 'NNP'), ('Roi', 'NNP'), ('Centre', 'NNP')]), ('Event', 'NNP')])

Chunking



- Chunking means picking up individual pieces of information and grouping them into bigger pieces.
- In the context of NLP and text mining, chunking means grouping of words or tokens into chunks.

```
text = “We saw the yellow dog”
token = word_tokenize(text)
tags = nltk.pos_tag(token)reg = “NP: {<DT>?<JJ>*<NN>}”
a = nltk.RegexpParser(reg)
result = a.parse(tags)
print(result)
```

(S We/PRP saw/VBD (NP the/DT yellow/JJ dog/NN))