How to make the best use of Live Sessions

- Please login on time
- Please do a check on your network connection and audio before the class to have a smooth session
- All participants will be on mute, by default. You will be unmuted when requested or as needed
- Please use the “Questions” panel on your webinar tool to interact with the instructor at any point during the class
- Ask and answer questions to make your learning interactive
- Please have the support phone number (US : 1855 818 0063 (toll free), India : +91 90191 17772) and raise tickets from LMS in case of any issues with the tool
- Most often logging off or rejoining will help solve the tool related issues

edureka!

Extracting, Cleaning & Pre-processing Text

Objectives

After completing this module, you should be able to:

- Clean and preprocess text data, using sentence tokenization
- Clean and preprocess text data, using word tokenization
- Demonstrate the use of Bigrams, Trigrams and ngrams
- Work on text data with Stemming, Lemmatization and Stop-Word removal
- Describe your text data with POS tags and Named Entitites

What Is Tokenization?

edureka!

Tokenization

A process of breaking strings into tokens, which in turn are small structures or units that can be used for ***Tokenization***

The other boy runs.

Tokenization

The	other	boy	runs	.

Use Of Tokenization

01 Break a complex sentence into words

02 Understand the importance of each of the words with respect to the sentence

03 Produces a structural description on an input sentence

Tokenization – Example

Let's consider a document of type string and understand the significance of its tokens:

```
gold = """Gold is a chemical element with symbol Au (from Latin: aurum) and
atomic number 79, making it one of the higher atomic number elements that
occur naturally. In its purest form, it is a bright, slightly reddish
yellow, dense, soft, malleable, and ductile metal. Chemically,a…. """
```

```
type(gold)
```

```
Out[2]: str
```

Import the necessary libraries:

```
import nltk
from nltk.tokenize import word_tokenize
```

edureka!

Tokenization – Example

Now, we will run the word_tokenize function over the paragraph (‘word’) and assign it a name:

```
gold_word_tokenize = word_tokenize(gold)
gold_word_tokenize
```

```
Out[6]:  ['Gold',
          'is',
          'a',
          'chemical
          'element'
          'with',
          'symbol',
          'Au',
          '(',
          'from',
          'Latin',
          ':',
          'aurum',
          ')',
          'and',
          'atomic',
          'number',
          '79',
```

A list of words and special characters as separate items of the list

Exploring The Tokens

Let's start by checking the number of tokens:

```
len(gold_word_tokenize)
```

```
Out[7]:  479
```

How many of these 479 items are unique??

NLTK's FreqDist ()

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

```
for word in gold_word_tokenize:
    fdist[word.lower()]+=1
fdist
```

```
Out[11]: FreqDist({'%': 3,
                   "'s": 2,
                   '(': 5,
                   ')': 5,
                   ',': 39,
                   '.': 18,
                   '10': 1,
                   '11': 1,
                   '186,700': 1,
                   '1930s': 1,
                   '1971': 1,
                   '2015': 1,
```

Tokens here converted to lower case, so as to avoid the probability of considering a word with upper case and lower case as different

edureka!

NLTK's FreqDist ()

Let's check the frequency of a particular word, say 'gold':

```
fdist['gold']
```

```
22
```

Let's check the length of token set again:

```
len(fdist)
```

```
Out[14]: 224
```

We can see now, the length of number of unique tokens have reduced from 479 to 224

edureka!

NLTK's FreqDist ()

Suppose, if you were to select the top 10 tokens with highest frequency:

```
fdist_top10=fdist.most_common(10)
fdist_top10
```

```
Out[16]: [(',', 39),
          ('gold', 22),
          ('in', 22),
          ('.', 18),
          ('and', 17),
          ('a', 16),
          ('of', 12),
          ('is', 11),
          ('the', 10),
          ('as', 8)]
```

A list of 10 tuples with tokens and their frequencies

edureka!

Few Other Types Of Tokenizers

Regex Tokenizer

Let’s use the regular expression tokenizer over the same string:

```
from nltk.tokenize import regexp_tokenize
regexp_tokenize(gold,pattern='\d+')
```

```
Out[19]: ['79',
          '11',
          '5',
          '6',
          '1930',
          '1971',
          '186',
          '700',
          '2015',
          '7',
          '50',
          '40',
          '10',
          '8',
          '2016',
          '450']
```

Regular Expression pattern

List of all the tokens, that matches your regular expression

edureka!

Blank Line Tokenizer

Let's use the blank line tokenizer over the same string to tokenize the paragraph with respect to blank string

```
from nltk.tokenize import blankline_tokenize
gold_bl_tokenize=blankline_tokenize(gold)
len(gold_bl_tokenize)
```

```
Out[21]:  4
```

4 paragraphs separated by a blank line

```
gold_bl_tokenize[0]
```

```
'Gold is a chemical element with symbol Au (from Latin: aurum) and atomic number 79, making it one of the higher atomic number
elements that occur naturally. In its purest form, it is a bright, slightly reddish yellow, dense, soft, malleable, and ductile
metal. Chemically, gold is a transition metal and a group 11 element. It is one of the least reactive chemical elements and is
solid under standard conditions. Gold often occurs in free elemental (native) form, as nuggets or grains, in rocks, in veins, a
nd in alluvial deposits. It occurs in a solid solution series with the native element silver (as electrum) and also naturally a
lloyed with copper and palladium. Less commonly, it occurs in minerals as gold compounds, often with tellurium (gold telluride
s).'
```

edureka!

Sentence Tokenizer

NLTK also has a straight forward function of creating tokens of sentences:

```
from nltk.tokenize import sent_tokenize
gold_sent_tokenize=sent_tokenize(gold)
gold_sent_tokenize
```

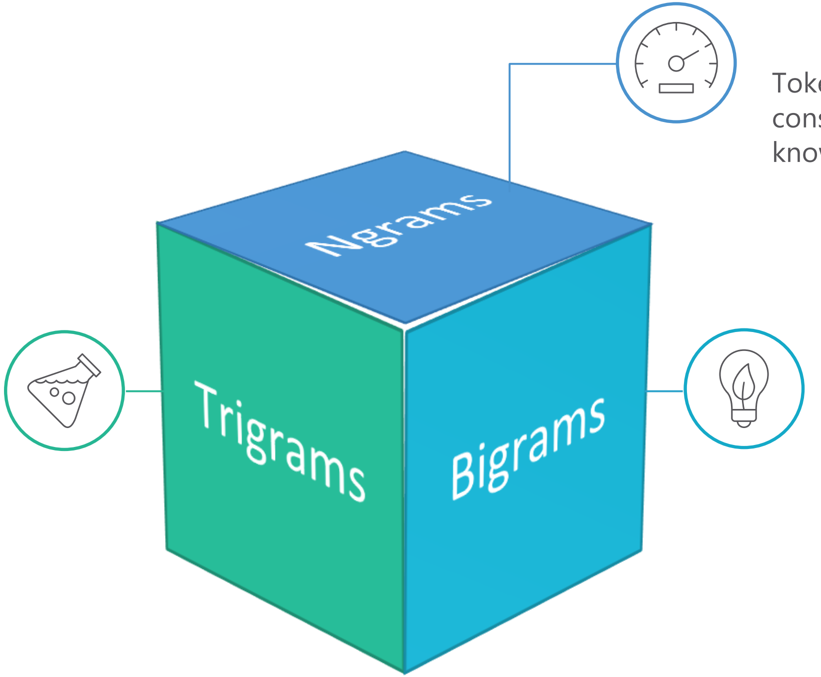
```
['Gold is a chemical element with symbol Au (from Latin: aurum) and atomic number 79, making it one of the higher atomic number
elements that occur naturally.',
 'In its purest form, it is a bright, slightly reddish yellow, dense, soft, malleable, and ductile metal.',
 'Chemically, gold is a transition metal and a group 11 element.',
```

edureka!

NLTK also allows you to tokenize phrases, containing more than one word

edureka!

Bigrams, Trigrams And Ngrams



Tokens of any number of consecutive written words known as Ngram

Tokens of three consecutive written words known as Trigram

Tokens of two consecutive written words known as Bigram

edureka!

Unigrams, Bigrams & Trigrams – Example

N = 1 : This | is | a | sentence — unigrams: this, is, a, sentence

N = 2 : This | is | a | sentence — bigrams: this, is, a, sentence

N = 2 : This | is | a | sentence — trigrams: this, is, a, sentence

Creating Bigrams Using NLTK

```
from nltk.util import bigrams, trigrams, ngrams
#let us consider the below string for the example
string = "The Mona Lisa is a half length portrait painting by the Italian
Renaissance artist Leonardo da Vinci"
```

Let us first create tokens of the above string using the word_tokenize():

```
mona_lisa_tokens=nltk.word_tokenize(string)
mona_lisa_tokens
```

```
Out[31]: ['The',
          'Mona',
          'Lisa',
          'is',
          'a',
          'half',
          'length',
          'portrait',
```

edureka!

Creating Bigrams Using NLTK

Let us now create bigrams of the list containing tokens:

```
mona_lisa_bigrams=list(nltk.bigrams(mona_lisa_tokens))
mona_lisa_bigrams
```

```
[('The', 'Mona'),
 ('Mona', 'Lisa'),
 ('Lisa', 'is'),
 ('is', 'a'),
 ('a', 'half'),
 ('half', 'length'),
 ('length', 'portrait'),
 ('portrait', 'painting'),
 ('painting', 'by'),
 ('by', 'the'),
```

Since, it is a generator, you have to convert it to a list and assign it a name

edureka!

Creating Trigrams Using NLTK

Let us now create trigrams of the list containing tokens:

```
mona_lisa_trigrams=list(nltk.trigrams(mona_lisa_tokens))
mona_lisa_trigrams
```

```
[('The', 'Mona', 'Lisa'),
 ('Mona', 'Lisa', 'is'),
 ('Lisa', 'is', 'a'),
 ('is', 'a', 'half'),
 ('a', 'half', 'length'),
 ('half', 'length', 'portrait'),
 ('length', 'portrait', 'painting'),
 ('portrait', 'painting', 'by'),
 ('painting', 'by', 'the'),
 ('by', 'the', 'Italian'),
 ('the', 'Italian', 'Renaissance'),
```

Creating Ngrams Using NLTK

Let us now create Ngrams of the list containing tokens:

```
mona_lisa_ngrams=list(nltk.ngrams(mona_lisa_tokens, 4))
mona_lisa_ngrams
```

```
[('The', 'Mona', 'Lisa', 'is'),
 ('Mona', 'Lisa', 'is', 'a'),
 ('Lisa', 'is', 'a', 'half'),
 ('is', 'a', 'half', 'length'),
 ('a', 'half', 'length', 'portrait'),
 ('half', 'length', 'portrait', 'painting'),
 ('length', 'portrait', 'painting', 'by'),
 ('portrait', 'painting', 'by', 'the'),
 ('painting', 'by', 'the', 'Italian'),
 ('by', 'the', 'Italian', 'Renaissance'),
```

Number of tokens, you need in your Ngram

Stemming

Stemming

01 Normalize words into its base form or root form

02 For example, the words *fish*, *fishes* and *fishing* all stem into **fish**

03 Result may not be the ‘root’ word

04 Common types of Stemming algorithms: Porter, Lancaster and Snowball

Stemming

- Normalize words into its base form or root form
- Result may not be the ‘root’ word
- For example, the words *fish*, *fishes* and *fishing* all stem into **fish**

Few words like study, studies and studying stems into ‘studi’, which is not an English word

Porter Stemmer With NLTK

Let's import Porter stemmer from NLTK:

```
from nltk.stem import PorterStemmer
pst=PorterStemmer()
```

Let's use the stemmer for the word 'having':

```
pst.stem("having")
```

```
Out[45]:  'have'
```

Using Porter Stemmer to stem a list of words:

```
words_to_stem=["give","giving","given","gave"]
for words in words_to_stem:
    print(words+ ":" +pst.stem(words))
```

```
give:give
giving:give
given:given
gave:gave
```

You can see, the stemmer removed only 'ing' and replaced it with 'e'

edureka!

Lancaster Stemmer

Let’s try to stem the same using Lancaster Stemmer:

```
from nltk.stem import LancasterStemmer
lst=LancasterStemmer()
for words in words_to_stem:
    print(words+ ":" +lst.stem(words))
```

```
give:giv
giving:giv
given:giv
gave:gav
```

You can see, the stemmer stemmed all the words. As a result of it, you can conclude that *Lancaster Stemmer is more aggressive than Porter Stemmer*

The use of each of the stemmers depends on the type of task, you want to perform. For eg: If you want to check, how many times the word ‘giv’ is used above: You can use **Lancaster Stemmer**

Snowball Stemmer

Let's try to stem, using the Snowball Stemmer:

```
from nltk.stem import SnowballStemmer
sbst=SnowballStemmer('english')
```

With Snowball Stemmer, you need to mention the language on which, you want the stemming algorithm to work

Let's check the list of languages Snowball Stemmer supports:

```
sbst.languages
```

```
Out[57]: ('arabic',
          'danish',
          'dutch',
          'english',
          'finnish',
          'french',
          'german',
          'hungarian',
```

```
'italian',
'norwegian',
'porter',
'portuguese',
'romanian',
'russian',
'spanish',
'swedish')
```

edureka!

Snowball Stemmer

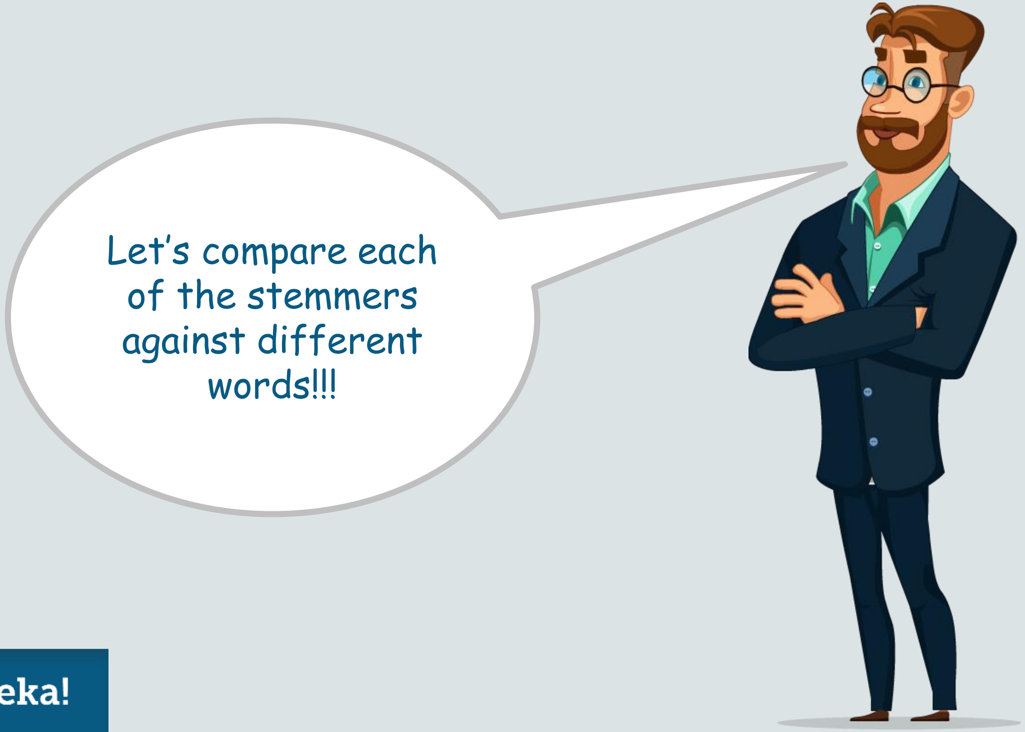
Now, stem the same using Snowball Stemmer:

```
sbst.stem('having')
```

```
'have'
```

```
for words in words_to_stem:
    print(words+ ":" +sbst.stem(words))
```

```
give:give
giving:give
given:given
gave:gave
```



edureka!

Compare: Porter, Lancaster And Snowball Stemmers

Create a function with Porter, Lancaster and Snowball Stemmers and use the function across different words:

```
def stemms(word):
    print("Porter:"+pst.stem(word))
    print("Lancaster:"+lst.stem(word))
    print("Snowball:"+sbst.stem(word))
    return
```

Check the stemms function on a word without ‘ing’:

```
stemms('data')
```

```
Porter:data
Lancaster:dat
Snowball:data
```

You can see, while porter and snowball kept the word ‘data’ as same, *Lancaster Stemmer removed the last ‘a’*

Check the stemms function on the word ‘curricula’:

```
stemms(‘curricula')
```

```
Porter:curricula
Lancaster:curricul
Snowball:curricula
```

You can see, while porter and snowball kept the word ‘curricula’ as same, *Lancaster Stemmer removed the last ‘a’*

edureka!


Let’s now look at another term in Text Processing, which is Lemmatization

edureka!

Lemmatization

L

Lemmatization

- Groups together different inflected forms of a word, called Lemma
- Somehow similar to stemming, as it maps several words into one common root
- Output of lemmatisation is a proper word
- For example, a lemmatiser should map *gone*, *going* and *went* into *go*

edureka!

Lemmatization Using NLTK

Let's import lemmatizer from NLTK:

```
from nltk.stem import wordnet
from nltk.stem import WordNetLemmatizer
word_lem=WordNetLemmatizer()
```

Let's check the lemmatizer on the word 'corpora':

```
word_lem.lemmatize('corpora')
```

```
Out[70]:  'corpus'
```

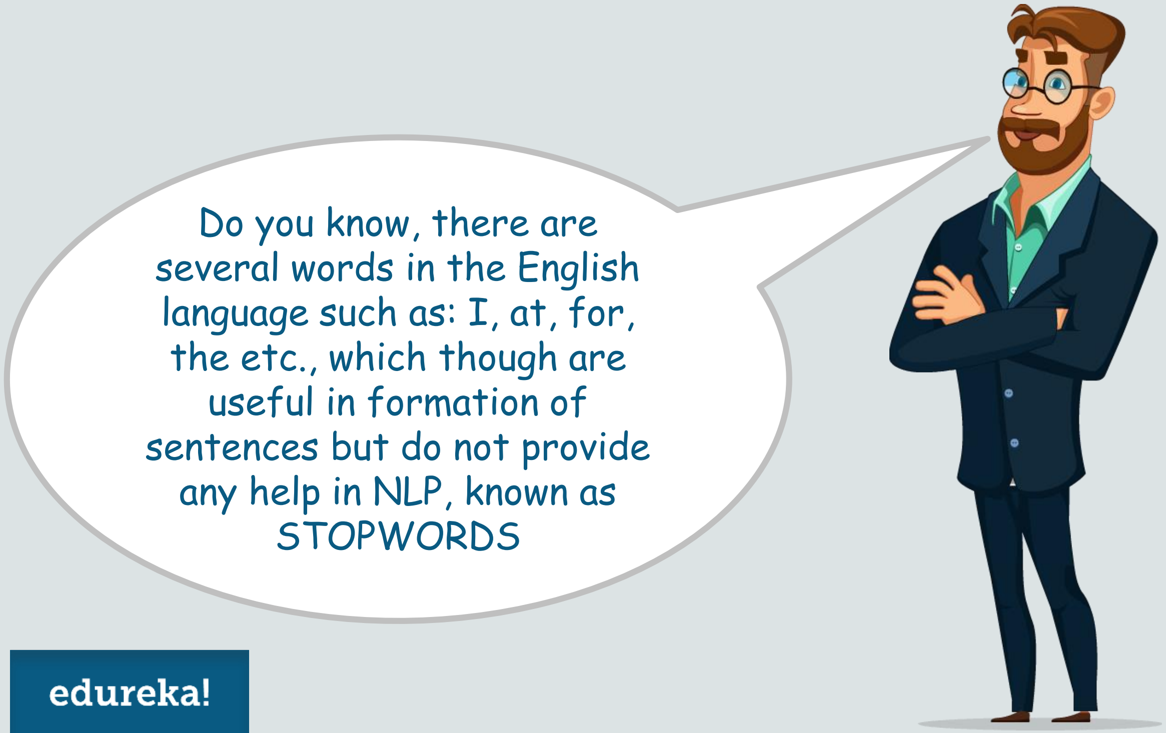
Let's check the lemmatizer on the list of words used earlier:

```
for words in words_to_stem:
    print(words+ ":" +word_lem.lemmatize(words))
```

```
give:give
giving:giving
given:given
gave:gave
```

You can see the lemmatizer has kept the words as it is, this is because you haven't assigned any POS tags and hence, it has assumed all the words as nouns

edureka!


Do you know, there are several words in the English language such as: I, at, for, the etc., which though are useful in formation of sentences but do not provide any help in NLP, known as STOPWORDS

edureka!

StopWords

NLTK has its own list of Stopwords, you can use the same by importing it from nltk.corpus:

```
from nltk.corpus import stopwords
```

Let's check the list of Stopwords in NLTK:

```
stopwords.words('english')
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
```

```
len(stopwords.words('english'))
```

```
179
```

edureka!

StopWord Removal

Now, remember the list of top 10 highest occurring words:

```
fdist_top10
```

```
[(',', 39),
 ('gold', 22),
 ('in', 22),
 ('.', 18),
 ('and', 17),
 ('a', 16),
 ('of', 12),
 ('is', 11),
 ('the', 10),
 ('as', 8)]
```

You can see that except 'gold' most of the words are either punctuation or Stopwords and hence, can be removed

You can remove the above list of stopwords using Regular Expressions

StopWord Removal Using The Re Module

```
import re
```

Now, we will use the compile() from the re module to create a string that matches any digit or special character

```
punctuation=re.compile(r'[-.?!,:;()|0-9]')
```

Now, we'll create an empty list and append the words without any punctuation into the list:

```
post_punctuation=[]
for words in gold_word_tokenize:
    word=punctuation.sub("",words)
    if len(word)>0:
        post_punctuation.append(word)
```

You can create another list and append the words in post_punctuation [ ] within the list removing stopwords:

```
post_stop_words=[]
for words in post_punctuation:
    words=words.lower()
    if words not in stp_words:
        post_stop_words.append(words)
```

edureka!

Check The Effect Of StopWord Removal

Now, let's compare the original list obtained as a result of Tokenization with the lists post_punctuation [ ] and post_stop_words [ ]:

```
len(gold_word_tokenize)
```

```
Out[88]:  479
```

```
len(post_punctuation) #list after removing punctuation
```

```
Out[114]:  397
```

```
len(post_stop_words)
```

```
Out[102]:  243
```

You can now instantiate the frequency distribution function and check the list of unique words post stop word removal:

```
fdist2=FreqDist()
```

edureka!

Check The Effect Of Stopword Removal: Unique Tokens

You can now check for the list of unique tokens in 'gold_word_tokenize':

```
for word in post_stop_words:
    fdist2[word]+=1
len(fdist2)
```

```
Out[105]:  170
```

Checking the top 10 most frequent tokens with highest frequency:

```
fdist2.most_common(10)
```

```
[('gold', 22),
 ('used', 5),
 ('chemical', 4),
 ('element', 4),
 ('acid', 4),
 ('[', 4),
 (']', 4),
 ('metal', 3),
 ('standard', 3),
 ('often', 3)]
```

More meaningful tokens(Unique Tokens) as compared to the previous list of top 10 most frequent tokens

```
[(',', 39),
 ('gold', 22),
 ('in', 22),
 ('.', 18),
 ('and', 17),
 ('a', 16),
 ('of', 12),
 ('is', 11),
 ('the', 10),
 ('as', 8)]
```

edureka!

Parts Of Speech (POS) Tagging

edureka!

Parts Of Speech (POS)

01 Generally speaking, the "grammatical type" of word:
Verb, Noun, Adjective, Adverb, Article, etc.,

02 Indicates, how a word functions in meaning as well as grammatically within the sentence

03 A word can have more than one part of speech based on the context in which it is used. For e.g.: "*Google Something on the internet*". Here *google* is used as a *verb* although it's a *proper noun*

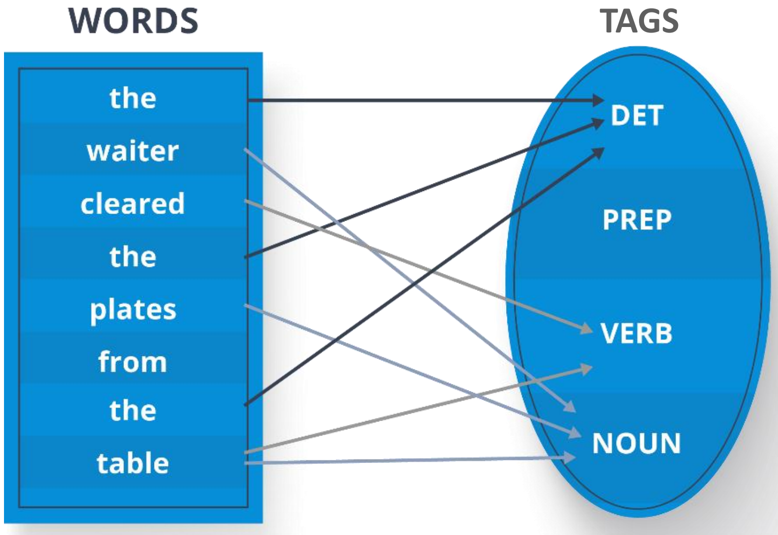
Some Common Tags & Their Descriptions

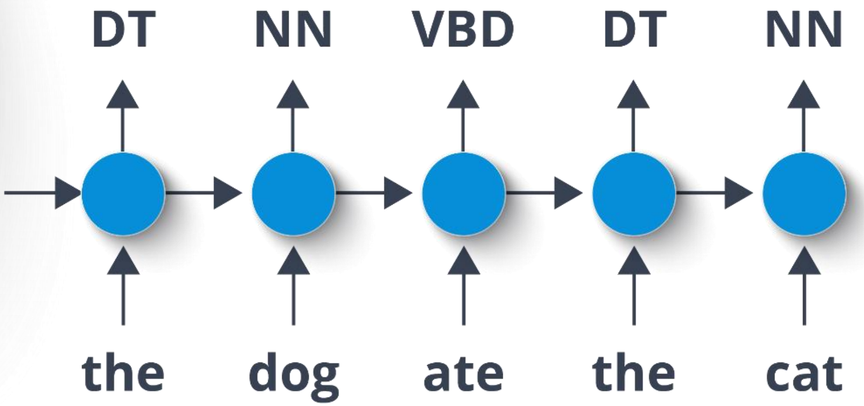
Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP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

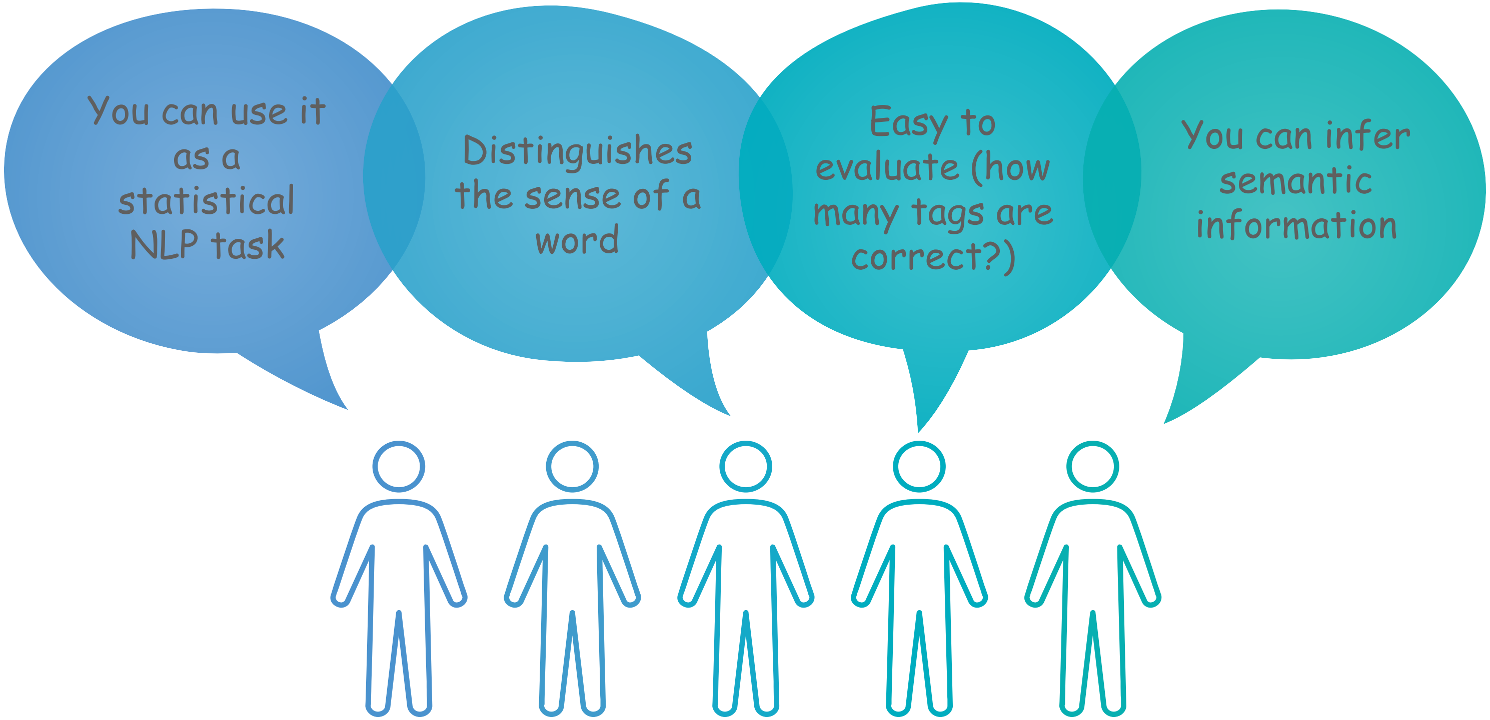
edureka!

POS Tags – Example





Need Of POS Tags



POS Tagging Using NLTK – Example 1

Let's consider a string and check how NLTK performs POS tagging on it:

```
sent = "Mary is driving a big car."
sent_tokens = word_tokenize(sent)
```

Use the pos_tag( ) function from the NLTK library to tag the tokens

```
for token in sent_tokens:
    print(nltk.pos_tag([token]))
```

```
[('Mary', 'NNP')]
[('is', 'VBZ')]
[('driving', 'VBG')]
[('a', 'DT')]
[('big', 'JJ')]
[('car', 'NN')]
[('.', '.')]
```

Words and their tags as tuples

The word should be inside [ ], else tagger will take it as a string

edureka!

POS Tagging Using NLTK – Example 2

```
sent2 = "John is eating a delicious cake"
sent2_tokens = word_tokenize(sent2)
for token in sent2_tokens:
    print(nltk.pos_tag([token]))
```

```
[('John', 'NNP')]
[('is', 'VBZ')]
[('eating', 'VBG')]
[('a', 'DT')]
[('delicious', 'JJ')]
[('cake', 'NN')]
```

You can see here, tagger has tagged both 'is' and 'eating' as verb because it has considered 'is eating' as a single term. This is one of the **shortcomings of POS tagger**


One way of dealing with this issue is to tokenize the sentence using Regular Expressions. Let's see how in the next example

edureka!

POS Tagging Using NLTK – Example 3

```
sent3= "Jim eats a banana"
sent3_tokens = word_tokenize(sent3)
for tokens in sent3_tokens:
    print(nltk.pos_tag([tokens]))
```

```
[('Jim', 'NNP')]
[('eats', 'NNS')]
[('a', 'DT')]
[('banana', 'NN')]
```

Here also, you can see the tagger has considered Jim and eats as a single term and hence, tagged them as Noun

Now, let's tokenize the same using Regular Expression tokenizer:

```
from nltk.tokenize import RegexpTokenizer
reg_tokenizer = RegexpTokenizer('(?u)\W+|\$[\d\.]+|\S+')
regex_tokenize = reg_tokenizer.tokenize(sent3)
```

edureka!

POS Tagging Using NLTK – Example 3

Now, let's tokenize the same using Regular Expression tokenizer:

```
regex_tokenize
```

```
Out[75]: ['Jim', ' ', 'eats', ' ', 'a', ' ', 'banana']
```

We now get a list of tokens, where even a space is a token

Now, we will tag all the tokens and get a list of tag for all the tokens:

```
regex_tag = nltk.pos_tag(regex_tokenize)
regex_tag
```

```
[('Jim', 'NNP'),
 (' ', 'NNP'),
 ('eats', 'VBZ'),
 (' ', 'VBP'),
 ('a', 'DT'),
 (' ', 'NN'),
 ('banana', 'NN')]
```

We can see here 'eats' now, is tagged as a verb

edureka!

Named Entity Recognition (NER)

edureka!

What Is NER?

01 Mix of Tagging as well as Chunking

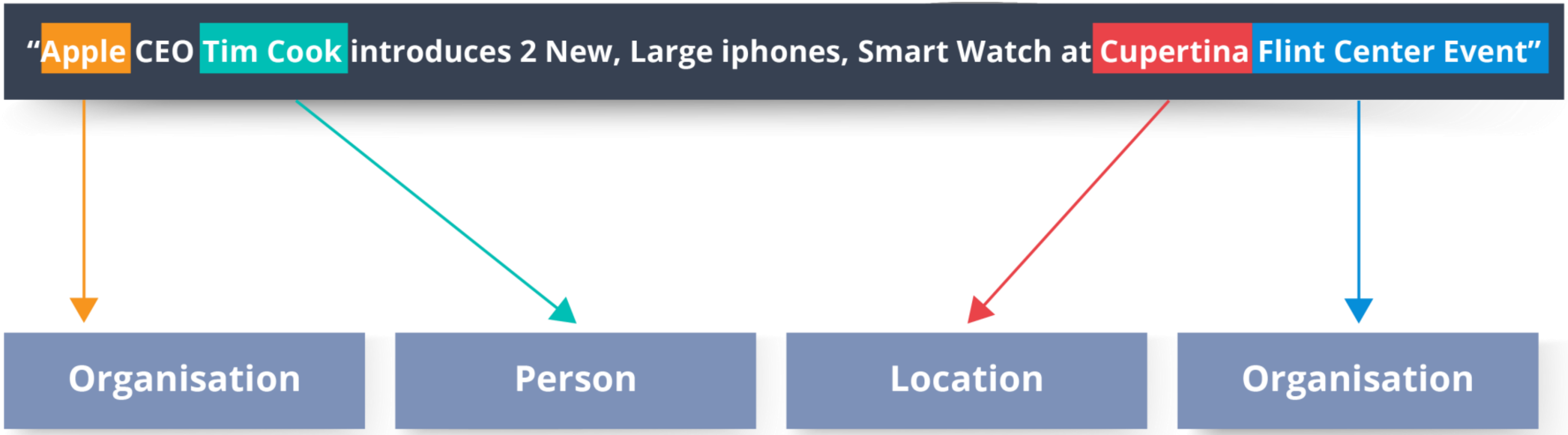
02 A part of IE (Information Exchange)

03 Automatic Identification and counting of occurrences of named entities in a collection of information

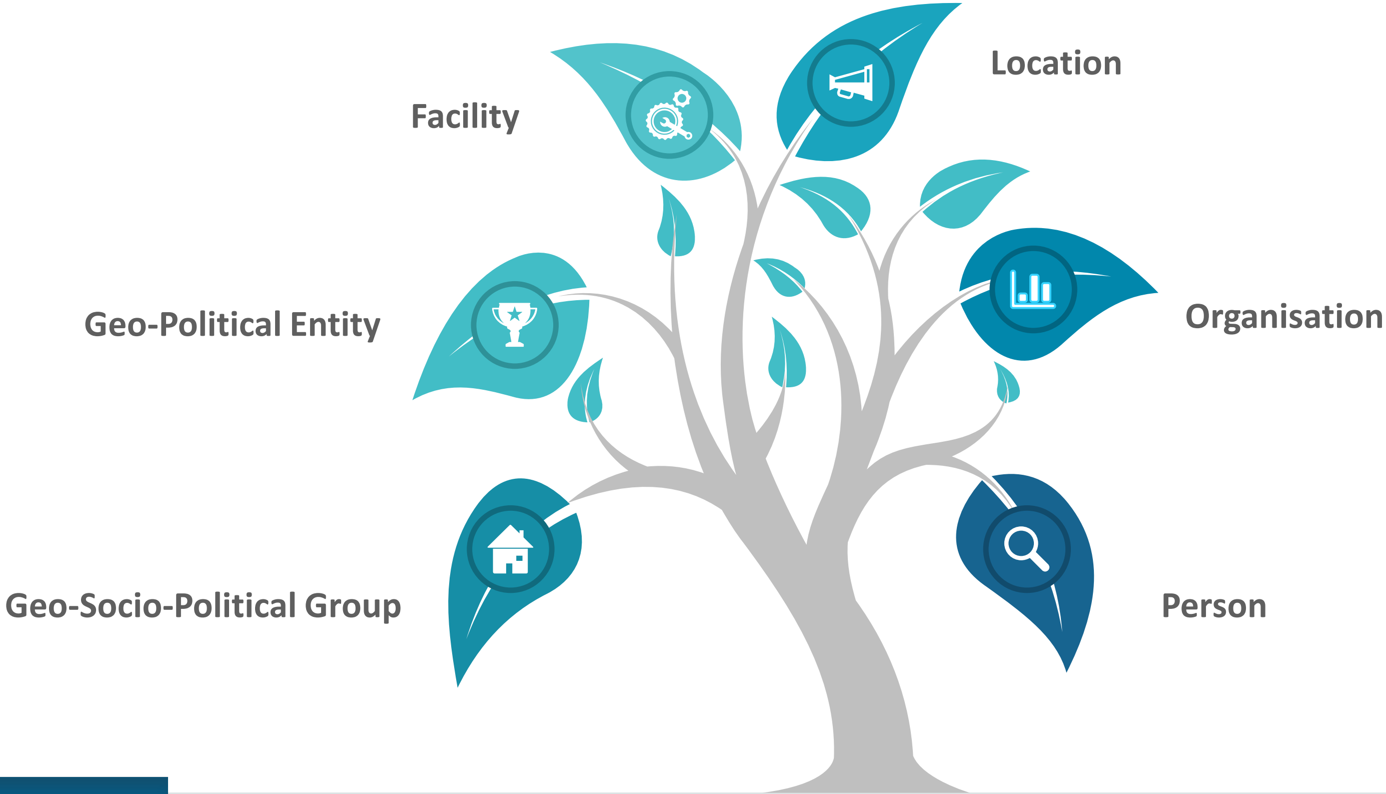
04 Associating the named entities to their appropriate types

edureka!

NER – An Example



List Of NER Entities



NER In Python

edureka!

NER In Python – Example 1

For using NER in Python, you will have to import the “ne_chunk” from the NLTK module in Python:

```
from nltk import ne_chunk
```

Consider a text data:

```
NE_sent = "The US President stays in the White House"
```

Now, let’s tokenize the sentence and also add part of speech tags to the same:

```
NE_tokens = word_tokenize(NE_sent)
NE_tags = nltk.pos_tag(NE_tokens)
```

NER In Python – Example 1

Now, we will use the ne_chunks ( ) and pass the list of tuples containing POS tags to it:

```
NE_NER = ne_chunk(NE_tags)
print(NE_NER)
```

```
(S
  The/DT
  (ORGANIZATION US/NNP)
  President/NNP
  stays/VBZ
  in/IN
  the/DT
  (FACILITY White/NNP House/NNP))
```

US is recognised as an organisation

White House is clubbed together as a single entity and is recognised as a facility

NER like POS tagger is also not 100% accurate and sometimes returns wrong entities

edureka!

NER In Python – Example 2

Let's consider another text:

```
NE_sent2 = "Apple is a fruit and Apple is a Company's name"
print(ne_chunk(nltk.pos_tag(word_tokenize(NE_sent2))))
```

```
(S
  (GPE Apple/NNP)
  is/VBZ
  a/DT
  fruit/NN
  and/CC
  (PERSON Apple/NNP)
  is/VBZ
  a/DT
  (ORGANIZATION Company/NN)
  's/POS
  name/NN)
```

Apple is recognised as a Geo Political Entity

Apple again is recognised as a person

While the first apple is recognised as a GPE, the second apple is recognised as a person, both of which are incorrect. Hence, NPE should be done with caution and checked for accuracy

edureka!

Summary

- Tokenization
- Bigrams, Trigrams & Ngrams
- Stemming
- Lemmatization
- Stopword Removal
- POS Tagging
- Named Entity Recognition (NER)

edureka!

Questions


Comments
Ratings
Suggestions
FEEDBACK
Survey
Likes
Ideas

edureka!

For more information please visit our website
www.edureka.co