# NLP basics using NLTK

# Contents

# Tokenization

Split an input text  into sentence and words. You can also then use to determine bigrams and trigrams

```
from nltk.tokenize import sent_tokenize, word_tokenize
sentence = sent_tokenize(user_input)
words = word_tokenizer(user_input)

bigrams = list(nltk.bigrams(words))
trigrams = list(nltk.trigrams(words))
```

# Stop Words Removal

Stop words are commonly used words in a language that are often filtered out or removed in natural language processing (NLP) tasks because they are deemed to be of little value in text analysis

Here are some examples of stop words in English:

- Articles: "a", "an", "the"
- Prepositions: "in", "on", "at", "by", "with"
- Conjunctions: "and", "or", "but", "so", "for"
- Pronouns: "he", "she", "it", "they", "we"

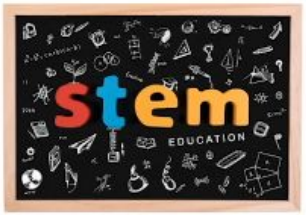
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

def remove_stopwords(text):
    # Tokenize the input text
    tokens = word_tokenize(text)

    # Load the stopwords list
    stop_words = set(stopwords.words('english'))

    # Remove stopwords from the tokenized text
    filtered_tokens = [word for word in tokens if
word.lower() not in stop_words]

    # Join the filtered tokens back into a string
    filtered_text = ' '.join(filtered_tokens)

    return filtered_text
```

# Stemming


Normalize words into its base form or root form

Affectation | Affects | Affections | Affected | Affection | Affecting

# Stemming




Normalize words into its base form or root form

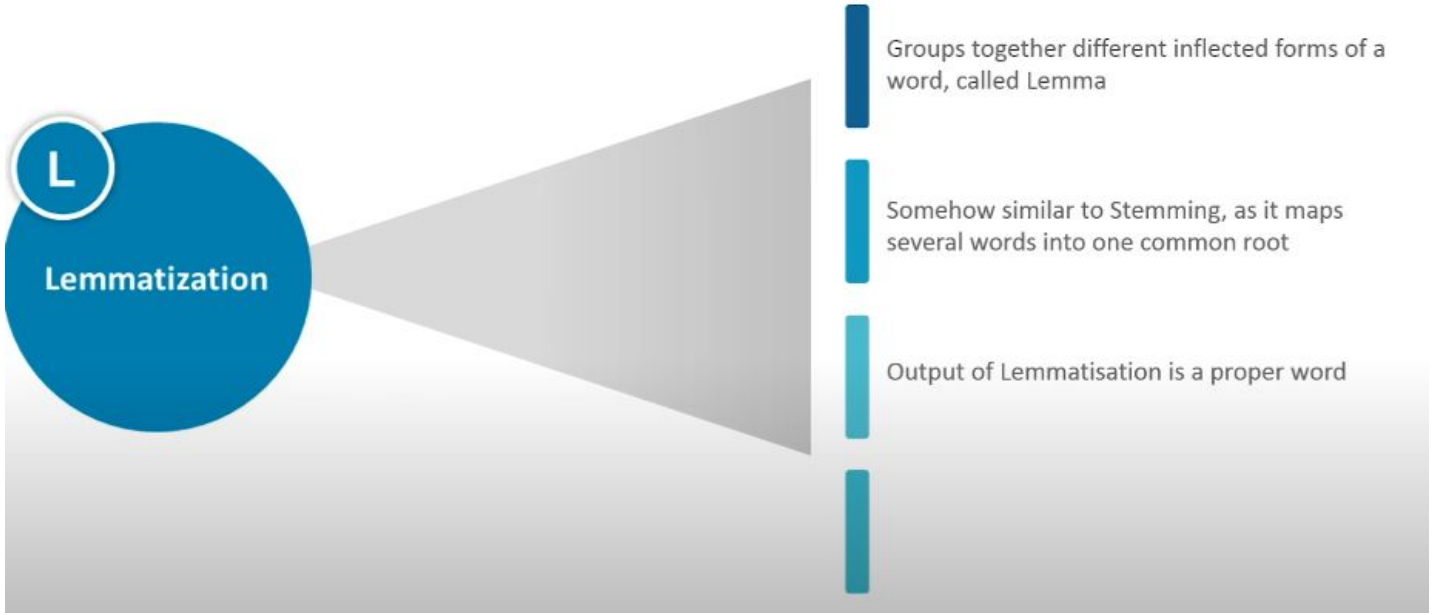
Affect

# Stemming

NLTK provides various kinds of stemming algo PorterStemmer, LancasterStemmer, and SnowballStemmer

```
from nltk.stem import PorterStemmer
pts = PorterStemmer()
words_to_stem = [“give”,”given”,”gave”]
for words in words_to_stem: print(word,’:’, pts.stem(word))

##Output
give: give
giving: give
gave:gave
```

# Lemmatization



# Lemmatization

```
from nltk.stem import wordnet, WordNetLemmatizer

pts = WordNetLemmatizer()
words_to_stem = [“give”,”given”,”gave”]
for words in words_to_stem: print(word,’:’, pts.lemmatize(word))

##Output
give: give
giving: giving
gave:gave
```

Why no change? – By default Lemmatizer assumes POS of input word as Noun whereas all are verbs

# POS: Part of Speech

# POS: Tags and Description

| 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 |

# POS tagging

sent = “John is eating a delicious cake”

sent_tokens = word_tokenize(sent)

for token in sent_tokens: print(nltk.pos_tag(token))

##Output

[(John, NBP)], [(is,VBZ), (eating, VBZ), (a, DT), (delicious, JJ), (cake, NN)]

# NER (Named Entity Recognition)

MOVIE

MONETARY VALUE

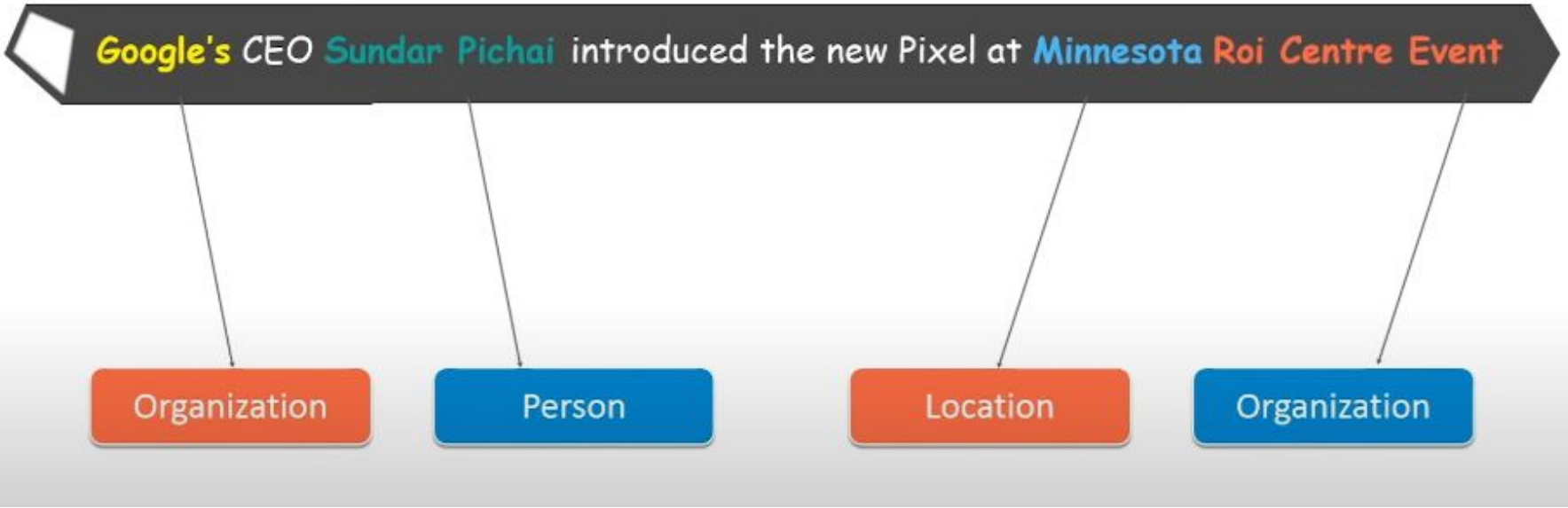
ORGANIZATION

LOCATION

QUANTITIES

PERSON

# NER


Google's CEO Sundar Pichai introduced the new Pixel at Minnesota Roi Centre Event
Organization
Person
Location
Organization

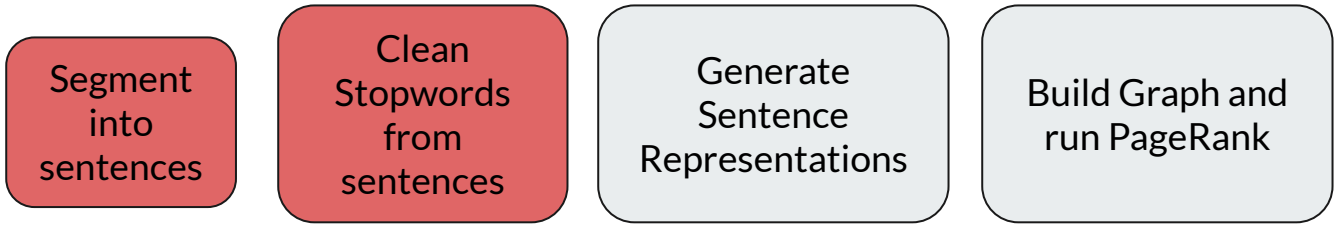
# NER

```
from nltk import ne_chunk
NE_sent =“The US President stays in the White House”
NE_tokens = word_tokenize(NE_sent)
NE_tags = nltk.pos_tag(NE_tokens)
NE_NER = ne_chunk(NE_tags)

#Output
The/DT (ORG US/NNP) President/NNP stays/VBZ in/IN the/DT (Facility White/NNP House/NNP)
```

# Practical Use-Case: Extractive Text Summarization

# Extractive Text Summarization

Given an input document, identify top-5 sentences that  captures the gist of the document. We will follow the following pipeline to generate the summary,



# Generate Sentence Representation

For each sentence, we encode it to a vector using tf-idf vector representation.

- Term Frequency:tf(t,s) In sentence s, the frequency of a given word t
- Document Frequency: df(t) = occurrence of t in all sentences
- Inverse Document Frequency: idf(t) = |S|/df(t)

```
tf-idf(t, d) = tf(t, d) * idf(t)
```

**However you do not have to write this from scratch, we will use scikit-learn**

# TF-IDF using scikit-learn

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign sentences
d0 = 'Geeks for geeks'
d1 = 'Geeks'
d2 = 'r2j'

# merge sentences into a single corpus
string = [d0, d1, d2]

# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)
```

```
Word indexes:
{'geeks': 1, 'for': 0, 'r2j': 2}
```

```
tf-idf values in matrix form:
[[0.54935123 0.83559154 0.        ]
 [0.         1.         0.        ]
 [0.         0.         1.        ]]
```

# Pagerank-based summarization

Create a fully-connected graph using sentences as nodes and tf-idf similarity as edge weights

Run Pagerank to identify important nodes (Sentences)

Select top-5 nodes (sentences) as the summary.