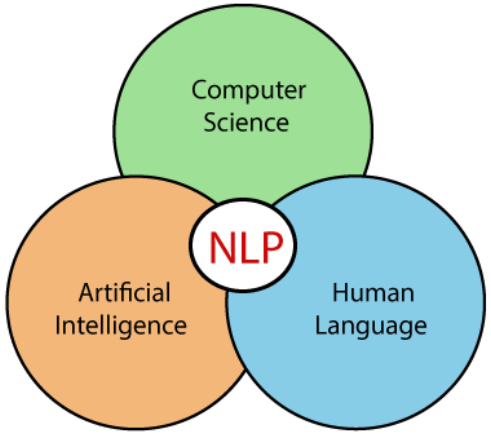
Natural Language Processing
Course code: CSE3015
Module 1
Introduction to NLP

Syllabus

- Overview: Origins and challenges of NLP-Need of NLP, python and NLTK for NLP, Text Wrangling and cleansing- Text cleansing, sentence splitter, tokenization, stemming, lemmatization, stop word removal, rare word removal, spell correction

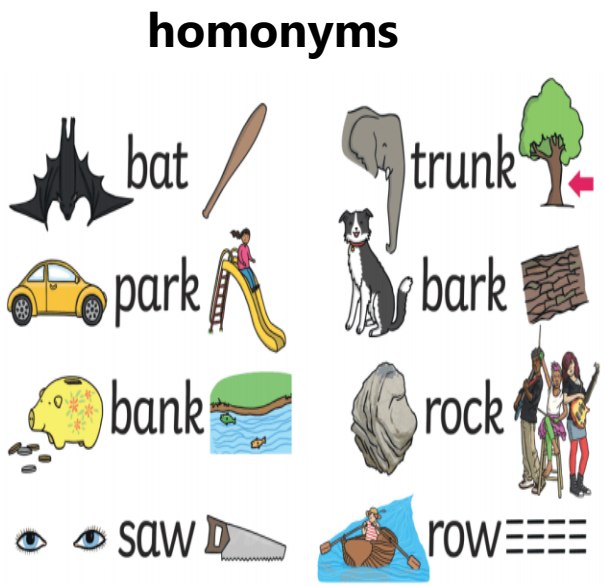
Introduction

- NLP stands for **Natural Language Processing**, which is a part of **Computer Science, Human language,** and **Artificial Intelligence**.
- It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages.
- It helps developers to organize knowledge for performing tasks such as **translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction,** and **topic segmentation**.



Challenges in NLP

- NLP is a powerful tool with huge benefits, but there are still a number of Natural Language Processing limitations and problems:
  - Contextual words and phrases and homonyms
    - I **ran** to the store because we **ran** out of milk.
    - The house is looking really **run** down
  - Synonyms
    - small, little, tiny, minute
  - Irony and sarcasm
    - Saying the opposite of what you mean for the purpose of humour or criticism



Challenges in NLP

- Ambiguity
  - sentences and phrases that potentially have two or more possible interpretations.
  - Lexical
    - The ambiguity of a single word is called lexical ambiguity. A word that could be used as a verb, noun, or adjective
  - Semantic
    - This kind of ambiguity occurs when the meaning of the words themselves can be misinterpreted
    - “The car hit the pole while **it** was moving”
  - Syntactic
    - when a sentence is parsed in different ways
    - “**The man saw the girl with the telescope**”
  - Anaphoric
    - the use of anaphora entities in discourse
    - “the horse ran up the hill. **It** was very steep. **It** soon got tired”
  - Pragmatic
    - situation where the context of a phrase gives it multiple interpretations
    - arises when the statement is not specific
    - “I like you too”

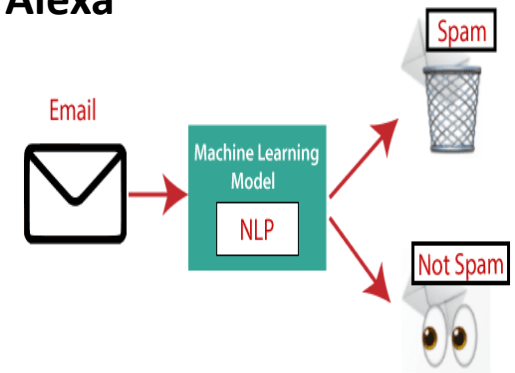
Challenges in NLP

- Errors in text or speech
  - Misspelled or misused words can create problems for text analysis.
- Colloquialisms and slang
  - use informal words and expressions.
  - Informal phrases, expressions, idioms, and culture-specific lingo present a number of problems for NLP.
  - And cultural slang is constantly morphing and expanding, so new words pop up every day
- Domain-specific language
  - Different businesses and industries often use very different language.
- Low-resource languages
  - many languages, especially those spoken by people with less access to technology often go overlooked and under processed
- Lack of research and development
  - The more data NLP models are trained on, the smarter they become.

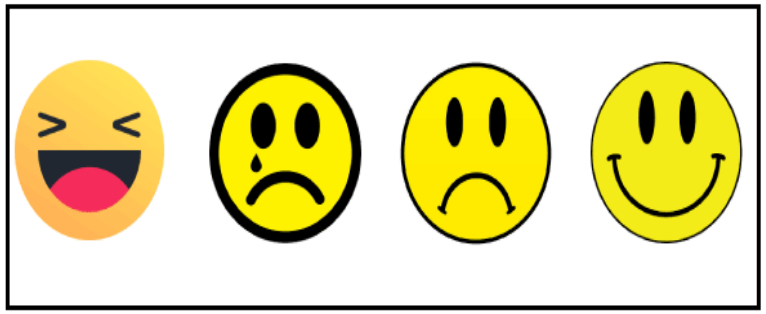
Application of NLP

Google Home , Alexa

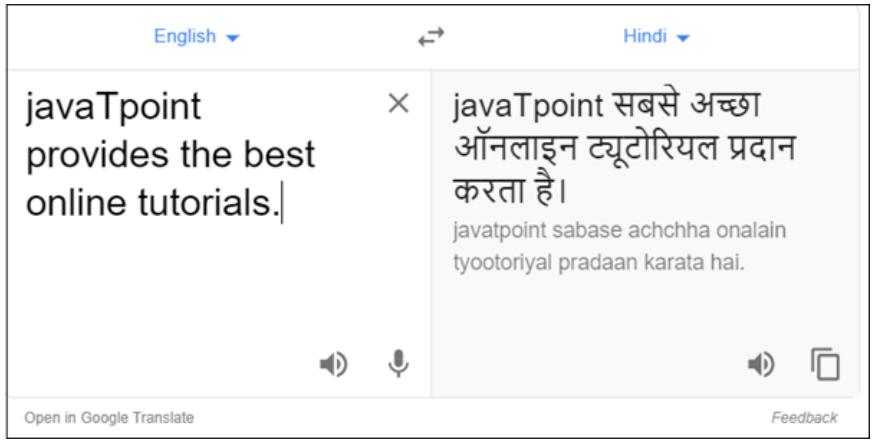
Question Answering



Spam Detection

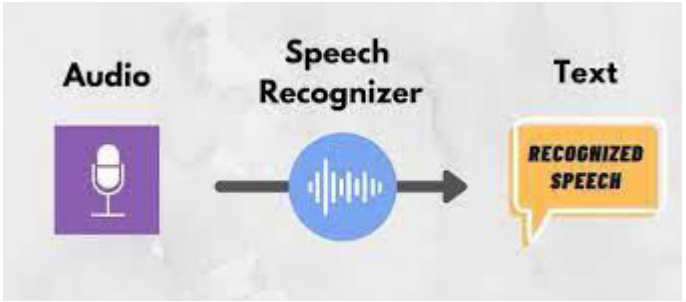


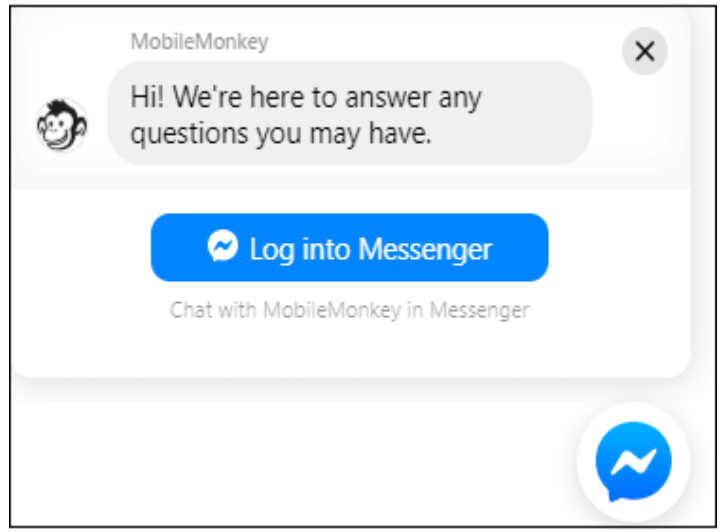
Sentiment Analysis



Machine Translation

Spelling correction





Chatbot

Introduction to NLTK

- NLTK is a toolkit build for working with NLP in Python.
- It provides us various text processing libraries with a lot of test datasets.
- To install
  - pip install nltk
- A variety of tasks can be performed using NLTK are
  - Tokenization
  - Lower case conversion
  - Stop Words removal
  - Stemming
  - Lemmatization
  - Parse tree or Syntax Tree generation
  - POS Tagging

Some important packages of nltk library

- nltk.classify
- nltk.cluster
- nltk.corpus
- nltk.metrics
- Nltk.parse
- Nltk.stem
- Nltk.tokenize
- Nltk.twitter

Text cleaning and Wrangling

- Gathering, sorting, and preparing data is the most important step in the data analysis process – bad data can have cumulative negative effects downstream if it is not corrected.
- Data preparation, aka data wrangling, meaning the manipulation of data so that it is most suitable for machine interpretation is therefore critical to accurate analysis.
- The goal of data preparation is to produce *'clean text'* that machines can analyze error free.

Text cleaning techniques

- Sentence Tokenization
  - Spit the text into sentences.
- Word Tokenization
  - split a sentence into words.
- Stemming
  - lowers inflection in words to their root forms, hence aiding in the preprocessing of text, words, and documents for text normalization.
- Lemmatization
  - lowers inflection in words to their root forms with preserving the meaning.
- stop word removal
  - Removal of most common words
- rare word removal
  - Removal of less important words.(low distribution)
- spell correction
  - Correcting of spelling in given word

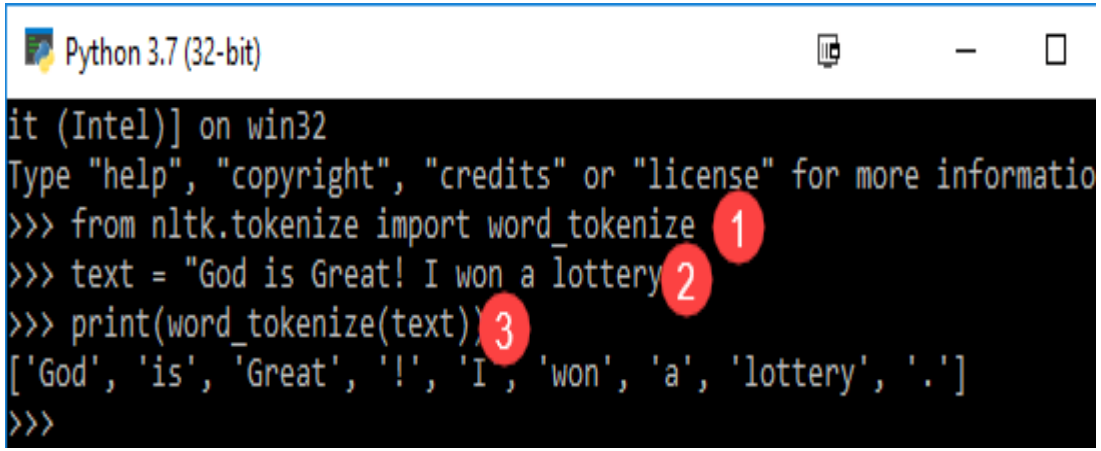
Tokenization

- Tokenization is breaking the raw text into small chunks.
- Tokenization breaks the raw text into words, sentences called tokens.
- These tokens help in understanding the context or developing the model for the NLP.
- Tokenization can be done to either separate words or sentences.
- If the text is split into words using some separation technique it is called word tokenization and same separation done for sentences is called sentence tokenization.
  - from nltk.tokenize import sent_tokenize
  - from nltk.tokenize import word_tokenize

Tokenization Example

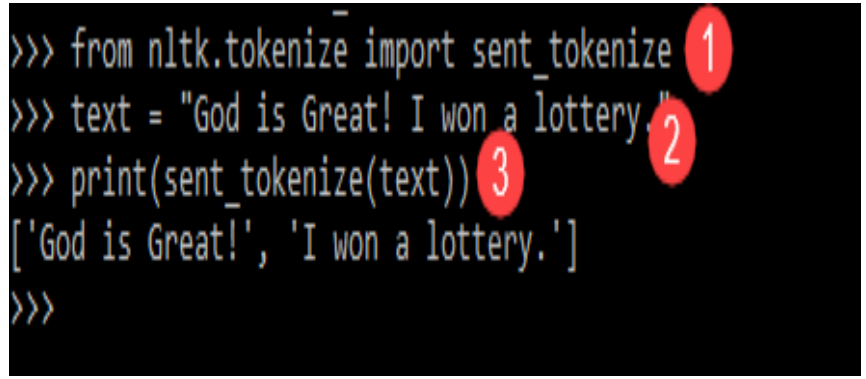
```
from nltk.tokenize import word_tokenize
text = "God is Great! I won a lottery."
print(word_tokenize(text))

Output: ['God', 'is', 'Great', '!', 'I', 'won', 'a', 'lottery', '.']
```




```
from nltk.tokenize import sent_tokenize
text = "God is Great! I won a lottery."
print(sent_tokenize(text))

Output: ['God is Great!', 'I won a lottery ']
```



Stop word removal

- The words which are generally filtered out before processing a natural language are called **stop words**.
- These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text.
- Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.

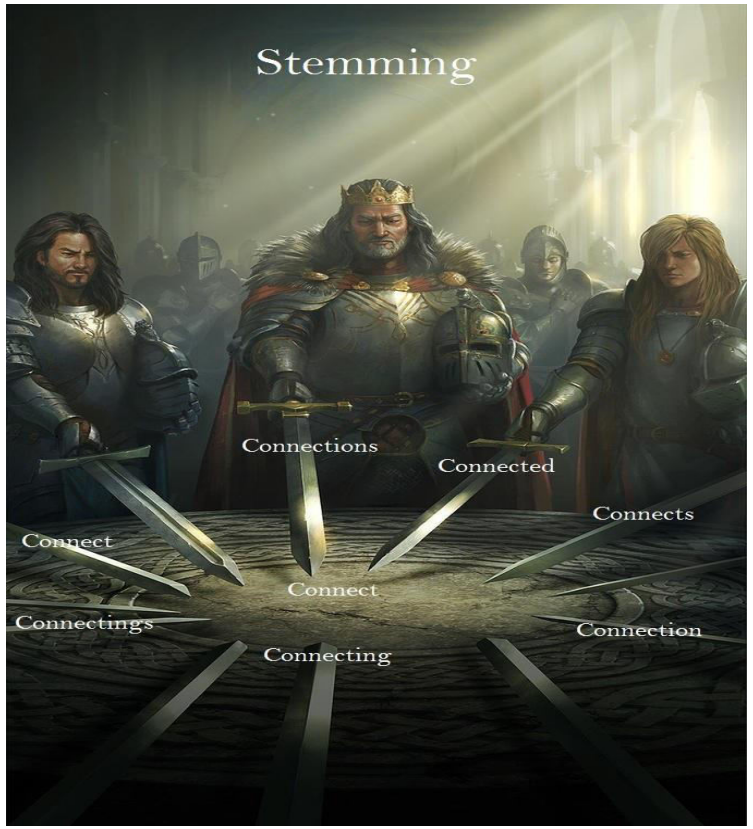
Example of stop word removal

```
Import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
text = "Nick likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]
print(tokens_without_sw)
```

['Nick', 'likes', 'play', 'football', ',', 'however', 'fond', 'tennis', '.']

Stemmer and its types in nltk

- Stemming is a technique used to extract the base form of the words by removing affixes from them.
- It is just like cutting down the branches of a tree to its stems.
- For example, the stem of the words ***eating, eats, eaten*** is ***eat***.
- Types
  - Porter Stemmer
    - from nltk.stem import PorterStemmer
  - Snowball Stemmer
    - from nltk.stem.snowball import SnowballStemmer
  - Lancaster Stemmer
    - from nltk.stem.lancaster import LancasterStemme
  - Regexp Stemmer
    - from nltk.stem import RegexpStemmer



Porter Stemmer Example

```
from nltk.stem import  PorterStemmer
porter = PorterStemmer()
words = ['generous','generate','generously'
,'generation']
for word in words:
    print(word,"--->",porter.stem(word))
```

```
generous ---> gener
generate ---> gener
generously ---> gener
generation ---> gener
```

Example of SnowballStemmer

- In the example below, we first construct an instance of SnowballStemmer() to use the Snowball algorithm to stem the list of words.

```
from nltk.stem import SnowballStemmer
snowball = SnowballStemmer(language='english')
words = ['generous','generate','generously','generation']
for word in words:
    print(word,"--->",snowball.stem(word))
```

```
generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat
```

Example of LancasterStemmer()

- In the example below, we construct an instance of LancasterStemmer() and then use the Lancaster algorithm to stem the list of words.

```
from nltk.stem import LancasterStemmer
lancaster = LancasterStemmer()
words = ['eating','eats','eaten','puts','putting']
for word in words:
    print(word,"--->",lancaster.stem(word))
```

```
eating ---> eat
eats ---> eat
eaten ---> eat
puts ---> put
putting ---> put
```

Example of RegexpStemmer

- In this example, we first construct an object of RegexpStemmer() and then use the Regex stemming method to stem the list of words.
- can create custom stemmer by programmer

```
from nltk.stem import RegexpStemmer
regexp = RegexpStemmer('ing$|s$|e$|able$', min=4)
words = ['mass','was','bee','computer','advisable']
for word in words:
    print(word,"--->",regexp.stem(word))
```

```
mass ---> mas
was ---> was
bee ---> bee
computer ---> computer
advisable ---> advis
```

Comparison of 4 types

```
from nltk.stem import PorterStemmer, SnowballStemmer, LancasterStemmer, RegexpStemmer
porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer(language='english')
regexp = RegexpStemmer('ing$|s$|e$|able$', min=4)
word_list = ["friend", "friendship", "friends", "friendships"]

print("{0:20}{1:20}{2:20}{3:30}{4:40}".format("Word","Porter Stemmer","Snowball Stemmer","Lancaster Stemmer",'Regexp Stemmer'))
for word in word_list:
        print("{0:20}{1:20}{2:20}{3:30}{4:40}".format(word,porter.stem(word),snowball.stem(word),lancaster.stem(word),regexp.stem(word)))
```

Word	**Porter**	**Snowball**	**Lancaster**	**Regexp**
friend	friend	friend	friend	friend
friendship	friendship	friendship	friend	friendship
friends	friend	friend	friend	friend
friendships	friendship	friendship	friend	friendship

lemmatization

- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.
- Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings to one word.
- It is another way to extract the base form of words, normally aiming to remove inflectional endings by using vocabulary and morphological analysis.
- After lemmatization, the base form of any word is called lemma.

Problems in stemming

Lemmatization Example

#Step 1:

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
```

#Step 2:

```
paragraph = """I have three visions for India. In 3000 years
of our history, people from all over
        the world have come and invaded us, captured our
lands, conquered our minds.
        From Alexander onwards, the Greeks, the Turks,
the Moguls, the Portuguese, the British,
        the French, the Dutch, all of them came and looted
us, took over what was ours.
        Yet we have not done this to any other nation. We
have not conquered anyone.
        We have not grabbed their land, their culture,
        their history and tried to enforce our way of life on
them.
        """
```

#Step 3:

```
sentences = nltk.sent_tokenize(paragraph)
print(sentences)
```

#Step 4:

```
lemmatizer = WordNetLemmatizer()
# Lemmatization
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [lemmatizer.lemmatize(word) for word
in words if word not in
set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)
```

#Step 5:

```
print(sentences)
```

Output

Before Lemmatization

['I have three visions for India.', 'In 3000 years of our history, people from all over \n the world have come and invaded us, captured our lands, conquered our minds.', 'From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,\n the French, the Dutch, all of them came and looted us, took over what was ours.', 'Yet we have not done this to any other nation.', 'We have not conquered anyone.', 'We have not grabbed their land, their culture, \n their history and tried to enforce our way of life on them.']

After Lemmatization

['I three vision India .', 'In 3000 year history , people world come invaded u , captured land , conquered mind .', 'From Alexander onwards , Greeks , Turks , Moguls , Portuguese , British , French , Dutch , came looted u , took .', 'Yet done nation .', 'We conquered anyone .', 'We grabbed land , culture , history tried enforce way life .']

Lemmatization with POS tag

```
from nltk import word_tokenize
from nltk import pos_tag
from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
s_words=stopwords.words('english')
input_text="I have three visions for India. In 3000 years of our history, people from all over the world have come and invaded us, captured our lands, conquered our minds. From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,the French, the Dutch, all of them came and looted us, took over what was ours. Yet we have not done this to any other nation. We have not conquered anyone. We have not grabbed their land, their culture, their history and tried to enforce our way of life on them"
```

```
def get_pos_lemma(word):
   t=pos_tag([word])[0][1][0].upper()

maping={'N':wordnet.NOUN,'V':wordnet.VERB,'J':wordnet.ADJ,'R':wordnet.ADV}
   return maping.get(t,wordnet.NOUN)
lemmatizer=WordNetLemmatizer()
f_tokens=[w for w in word_tokenize(input_text) if w not in s_words]
f_tokens2=[]
for i in f_tokens:
            f_tokens2.append(lemmatizer.lemmatize(i,get_pos_lemma(i)))
print("before lemmatization")
print(f_tokens)
print("after lemmatization")
print(f_tokens2)
#print(lemmatizer.lemmatize("grabbed",wordnet.VERB))
```

output

before lemmatization

['I', 'three', 'visions', 'India', '.', 'In', '3000', 'years', 'history', ',', 'people', 'world', 'come', 'invaded', 'us', ',', 'captured', 'lands', ',', 'conquered', 'minds', '.', 'From', 'Alexander', 'onwards', ',', 'Greeks', ',', 'Turks', ',', 'Moguls', ',', 'Portuguese', ',', 'British', ',', 'French', ',', 'Dutch', ',', 'came', 'looted', 'us', ',', 'took', '.', 'Yet', 'done', 'nation', '.', 'We', 'conquered', 'anyone', '.', 'We', 'grabbed', 'land', ',', 'culture', ',', 'history', 'tried', 'enforce', 'way', 'life’]

after lemmatization

['I', 'three', 'vision', 'India', '.', 'In', '3000', 'year', 'history', ',', 'people', 'world', 'come', 'invade', 'u', ',', 'capture', 'land', ',', 'conquer', 'mind', '.', 'From', 'Alexander', 'onwards', ',', 'Greeks', ',', 'Turks', ',', 'Moguls', ',', 'Portuguese', ',', 'British', ',', 'French', ',', 'Dutch', ',', 'come', 'loot', 'u', ',', 'take', '.', 'Yet', 'do', 'nation', '.', 'We', 'conquer', 'anyone', '.', 'We', 'grabbed', 'land', ',', 'culture', ',', 'history', 'try', 'enforce', 'way', 'life']

Wordnet POS tags list

Description	Penn POS Tag	Equivalent SentiMI POS Tag
Verb, base form	VB	V
Verb, past tense	VBD	V
Verb, gerund or present participle	VBG	V
Verb, past participle	VBN	V
Verb, non-3rd person singular present	VBP	V
Verb, 3rd person singular present	VBZ	V
Noun, singular or mass	NN	N
Noun, plural	NNS	N
Proper noun, singular	NNP	N
Proper noun, plural	NNPS	N
Adjective	JJ	A
Adjective, comparative	JJR	A
Adjective, superlative	JJS	A
Adverb	RB	R
Adverb, comparative	RBR	R
Adverb, superlative	RBS	R

Rare word Removal

- Some times we need to remove the words that are very unique in nature like names, brands, product names, and some of the noise characters, such as html leftouts.
- This is considered as “rare word removal”.
- Those words are appeared less frequently in the text.
- Method
  - FreqDist() is used to get the distribution of the terms in the corpus.
  - Can get the less distributed word as rare words.

Example

```
from nltk import FreqDist
tokens=['hi','i','am','am','whatever','this','is','just','a','test','test','java','python','java']
freq_dist = FreqDist(tokens)
sorted_tokens=dict(sorted(freq_dist.items(), key=lambda x:x[1]))
final_tokens=[]
for x,y in sorted_tokens.items():
    if y>1:
        final_tokens.append(x)
print(final_tokens)

Output:
['am', 'test', 'java']
```

Correcting Words using NLTK in Python

- Spelling corrections are important phase of text cleaning process, since misspelled words will leads a wrong prediction during machine learning process.
- Two methods are available in nltk library to correct the spelling of the incorrect words.
  - Jaccard distance method
  - Edit distance method

Jaccard Distance

- the opposite of the Jaccard coefficient, is used to measure the dissimilarity between two sample sets.
- We get Jaccard distance by subtracting the Jaccard coefficient from 1.
- We can also get it by dividing the difference between the sizes of the union and the intersection of two sets by the size of the union.
- We work with Q-grams (these are equivalent to N-grams) which are referred to as characters instead of tokens.
- Jaccard Distance is given by the following formula

$$JaccardIndex = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

$$JaccardDistance = 1 - JaccardIndex$$

Continue.....

Step 3: define the list of incorrect_words for which we need the correct spellings. Then run a loop for each word in the incorrect words list in which we calculate the Edit distance of the incorrect word with each correct spelling word having the same initial letter. finally sort them in ascending order so the shortest distance is on top and extract the word corresponding to it and print it.

```
# list of incorrect spellings
# that need to be corrected
incorrect_words=['happpy', 'azmaing', 'intelliengt']

# loop for finding correct spellings
# based on edit distance and
# printing the correct words
for word in incorrect_words:
        temp = [(edit_distance(word, w),w) for w in
correct_words if w[0]==word[0]]
        print(sorted(temp, key = lambda val:val[0])[0][1])
```

Output:

```
[nltk_data] Downloading package words to /home/rahul/nltk_data...
[nltk_data]   Package words is already up-to-date!

happy
aiming
intelligent
```

Example using Jaccard distance

Step 1: First, we install and import the nltk suite and Jaccard distance metric that we discussed before. ‘ngrams’ are used to get a set of co-occurring words in a given window and are imported from nltk.utils package.

```
# importing the nltk suite
import nltk

# importing jaccard distance
# and ngrams from nltk.util
from nltk.metrics.distance import jaccard_distance
from nltk.util import ngrams
```

Step 2: Now, we download the ‘words’ resource (which contains the list of correct spellings of words) from the nltk downloader and import it through nltk.corpus and assign it to correct_words.

```
# Downloading and importing
# package 'words' from nltk corpus
nltk.download('words')
from nltk.corpus import words



correct_words = words.words()
```

Continue……

Step 3: define the list of incorrect_words for which we need the correct spellings. run a loop for each word in the incorrect words list in which we calculate the Jaccard distance of the incorrect word with each correct spelling word having the same initial letter in the form of bigrams of characters. Then sort them in ascending order so the shortest distance is on top and extract the word corresponding to it and print it.

```
# list of incorrect spellings
# that need to be corrected
incorrect_words=['happpy', 'azmaing', 'intelliengt']

# loop for finding correct spellings
# based on jaccard distance
# and printing the correct word
for word in incorrect_words:
        temp = [(jaccard_distance(set(ngrams(word, 2)),set(ngrams(w, 2))),w) for w in correct_words if w[0]==word[0]]
        print(sorted(temp, key = lambda val:val[0])[0][1])
```

Output:

```
[nltk_data] Downloading package words to /home/rahul/nltk_data...
[nltk_data]   Package words is already up-to-date!

happy
amazing
intelligent
```

Edit distance method

Edit Distance measures dissimilarity between two strings by finding the minimum number of operations needed to transform one string into the other.

The transformations that can be performed are:

- inserting a new character:
  - bat -> bats (insertion of 's')
- Deleting an existing character.
  - care -> car (deletion of 'e')
- Substituting an existing character.
  - bin -> bit (substitution of n with t)
- Transposition of two existing consecutive characters.
  - sing -> sign (transposition of ng to gn)

Example for Edit distance

Step 1: First of all, we install and import the nltk suite.

```
# importing the nltk suite
import nltk

# importing edit distance
from nltk.metrics.distance import edit_distance
```

Step 2: Now, we download the ‘words’ resource (which contains correct spellings of words) from the nltk downloader and import it through nltk.corpus and assign it to correct_words.

```
# Downloading and importing package 'words'
nltk.download('words')
from nltk.corpus import words
correct_words = words.words()
```