Natural Language Processing Tokenization

```
import nltk #Library
nltk.download() #download all the libraries which require
Paragraph = """ I have three vision for India. In 3000 years of our history, people from all over the world have come invaded us, Captured our lands, Conquered our minds."""
```

inbuilt function

```
Sentences = nltk.sent_tokenize(paragraph) #para -> Sentence in the form of List
words = nltk.word_tokenize(paragraph) #para -> words it counts how many words are there.
```

Tokenization

- Tracing the paragraph or corpus and then will be converting those paragraph into sentences

Stemming And Lemmatization

Stemming

Process of Reducing Infected words to their Word STEM

history / histori — histori
historical — histori
" " " — — — —
finally / final — fina
finalized
going / goes — go

Lemmatization

history / history
historical — history
finally / finally — final
finalized — final

Tokenization = Is the process of breaking text into pieces called tokens.

Date
Page

- Stemming
  - Use case like Sentiment Analysis, Restaurant Review Spam Clipper. Some of the word representation will not have the meaning. It does not take of time to find out the base word.
- Limmatization
  - Basically it creates more "meaning for words that can humans understand
    - eg:- history → history
      historical
  - It takes time for process
  - Use case :- chatbot, QA application.

* Stemming (Stopwords)

Stopwords: we apply stopwords to remove these kind of words like that don't plays much meaningful ⇒ is, of, this, I'll, ... etc (this kind of words repeated again and again) — helps to remove these type of words

```
import nltk
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
stopwords (is basically present in nltk.corpus)
Paragraph = """ ..... """
Sentences = nltk.sent_tokenize(Paragraph)
stemmer = PorterStemmer()
# for stemming purpose
```

#Stemming

```
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [stemmer.stem(word) for word in words
             if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)
```

(if of words)

(can be any language)

set helps us to take only the unique stopwords in the English language

Problem

→ Produced intermediate representation of the word may not have any meaning

eg:- Intelligent → Intelligen

finall → fina, etc

* Lemmatization

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

paragraph = """ """
sentences = nltk.sent_tokenize(paragraph)
lemmatizer = WordNetLemmatizer()  # this is responsible for doing lemmatization
for i in range(len(sentence)):
    word = nltk.word_tokenize(sentences[i])
    word = [lemmatizer.lemmatize(word) for word in
            words if word not in set(stopwords.words('english'))]
sentences[i] = ' '.join(words)
```

* Bag of Words

Sentence 1 → He is good boy

Sentence 2 → She is good girl

Sentence 3 → Boy & girl are good

Stop words: is, He, are, and, She

Lowering Sentence

sent 1 → good, boy

sent 2 → good, girl

sent 3 → Boy, girl, good

Creating a histogram basically what is the frequency, what is the total count

Words	frequency
good	3
boy	2
girl	2

vector

Bow → to → vector

	good	boy	girl	o/p
sent 1	present 1	present 1	not present 0	
sent 2	1	0	1	
sent 3	1	1	1	

f1 f2 f3

count (i) good = 3

(ii) Boy = 2

(iii) girl = 2

~~Binary Bow~~

Disadvantages :- My values are either '1' or '0', but one important thing to note over here i have this Good value = 1, Boy value = 1, both are having Equal representation the Semantics of this particular words are also some, we are not able to derive which word is much more important.

→ In Sentiment Analysis the major thing

is that we ~~should~~ should know that which word is more important like in this particular case Good should be more important
eg:- He is Intelligent
major word

Note:- To avoid this disadvantage of Bow we use TF-IDF

- Bow can be used in Sentiment Analysis
- But, If you have a Huge Data Set Don't Go with Bow, use word2vec to solve.

* ~~Bag of words~~

Bag of words ("Code")

> import nltk Bag: Bag of words → Document matrix, all together

> Paragraph = """ ------- """

> # cleaning the texts

import re import regular Expression for Lowering

from nltk.corpus import the text

from nltk.stem.porter import PorterStemmer

from nltk.stem import WordNetLemmatizer

> ps = PorterStemmer()

> wordnet = WordNetLemmatizer()

> sentences = nltk.sent_tokenize(paragraph)

> corpus = []

for i in range(len(sentences)):

review = re.sub('[^a-zA-Z]', ' ', sentences[i])

removing all the (comma, punctuation, space, question mark not required)

review = review.lower()

review = review.split() # bag of words

review = [ps.stem(word) for word in review if word not in set(stopwords.words('english'))]

stemming

corpus.append(review)

wordnet.lemmatize(word)

lemmatize

> # Creating the Bag of words model

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(max_features = 1500)

X = cv.fit_transform(corpus).toarray()

creating a matrix

* TF - IDF

Sent 1 → He is good boy
Sent 2 → She is good girl
Sent 3 → Boy & Girl are Good

after applying (Stemming, Lemmatization, Stopwords)

Sent 1 → good boy
Sent 2 → good girl
Sent 3 → Boy girl good

Sent 1 → good boy
Sent 2 → good girl
Sent 3 → boy girl good

Formula of

- Term Frequency (TF) = $\frac{\text{No. of repetation of words in sentence}}{\text{No. of words in sentence}}$
- Inverse Document frequency (IDF)

$$IDF = \log\left(\frac{\text{No. of sentences}}{\text{No. of sentences containing words}}\right)$$

Finally

$$TF * IDF$$

words	frequency
good	3
boy	2
girl	2

Tf-Idf → vectors

	Tf		
	Sent 1	Sent 2	Sent 3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

Idf	
words	Idf
good	log(3/3) = 0
boy	log(3/2)
girl	log(3/2)

	f1	f2	f3	O/P
	good	boy	girl	
Sent 1	1/2 * log(3/3) = 0	1/2 * log(3/2)	0 * log(3/2) = 0	
Sent 2	1/2 * log(3/2)	0 * log(3/2) = 0	1/2 * log(3/2)	
Sent 3	1/3 * log(3/3) = 0	1/3 * log(3/2)	1/3 * log(3/2)	

Note:
- In BOW we have values in 1 and 0
- In TF-IDF after computing features (independent values) we will not get values like 1 and 0, we will get values in Decimal like 1.32, 4.7

* TF - IDF ("code")

```
>> import nltk

> paragraph = """ _ _ _ _ _ _ _ _ _ _ """

> # cleaning the text
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer

> ps = PorterStemmer() # Stemming
> wordnet = WordNetLemmatizer() # Lemmatization
> sentences = nltk.sent_tokenize(paragraph)
> corpus = []

> for i in range(len(sentences)):
    review = re.sub('[^a-zA-Z]', ' ', sentences[i])
    review = review.lower
    review = review.split
Stemming
  → review = [ps.stem(word) for word in review
             if not word in set(stopwords.words('english'))]
Lemmatizer
  → review = [wordnet.lemmatize(word) for word in
             review if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

Creating the Bag of words model

```
from sklearn.feature_extraction.text import TfidfVectorizer
cv = TfidfVectorizer()
x = cv.fit_transform(corpus).toarray()
```

N-Grams

⇒ N-Grams are way to help machines understand a word in the context to get a better understanding of words. N-gram is a neighboring sequence of n-items from a given sample of text. N-item means we can have two items, three items and so on. So, it is a contiguous sequence of some items. It helped to predicting the next words in a sentence. Things will be characters, words, sentences.

→ When n is 2 then we can call it as bigrams

→ When n is 3 then we can call it as trigrams.

Based on sentence we can change the value of "n".

- **unigram model**

uni = 1

unigram = 1 word in a sentence

Ex:- "he tried to find a position as a teacher but had no luck at first"

unigram:

he

tried

to

find

a

position

:

unigram model ⇒ P(wt)

- **Bigram Model**

Bi=2

Bigram = 2 Consecutive words in a sentence

Ex :- "he tried to find a position as a teacher but had no luck at first"

Bigram :

he tried

tried to

to find

find a

a position

⋮

Bigram model :- $P(w_t \mid w_{t-1})$

- **Trigram Model**

Tri = 3

Trigram = 3 Consecutive words in a sentence

Ex :- "he tried to find a position as a teacher but had no luck at first"

Bigrams:

he tried to

tried to find

to find a

find a position

a position as

⋮

Trigram model = $P(w_t \mid w_{t-1}, w_{t-2})$

* Word 2 Vec

- **Bag of words and TF-IDF Problems**

→ Both BoW and TF-IDF approach semantic information is not stored. TF-IDF gives importance to uncommon words.

→ There is definitely chance of overfitting.

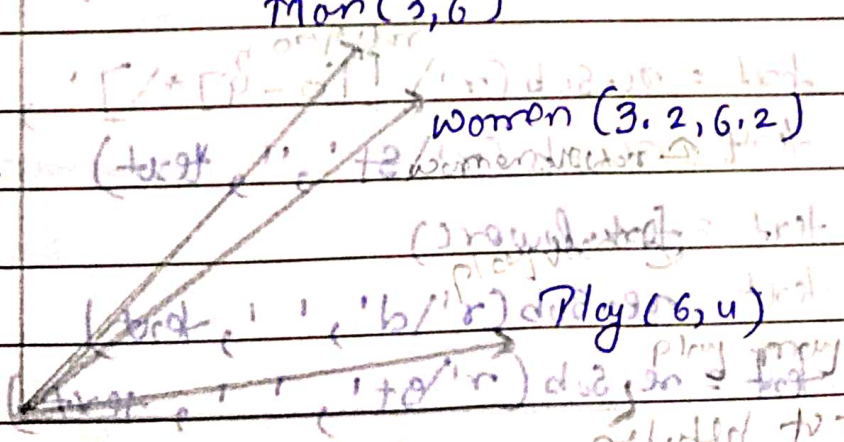
Solution → Word 2 vec

→ In this specific models each word is basically represented as a vector of 32 or more dimension instead of a single number

→ Here the semantic information and relation between different words is also presened.

- **Visual Representation - Word 2 vec**

(2D) (x, y)



King - Man + woman = Queen

- Steps to create word2vec
  - → Tokenization of the sentences
  - → Create Histograms
  - → Take most frequent words
  - → Create a matrix with all the unique words. It also represent the occurrence relation between the words.
- Gensim library —(for using)→ Word2Vec model.

```
>import nltk
```

Each and every word will represent different dimension

```
>from gensim.models import Word2Vec
>from nltk.corpus import stopwords
>import re

>Paragraph = """ ------ """

># Preprocessing the data
To remove space, special character, some words
text = re.sub(r'\[[0-9]*\]', ' ', Paragraph)
text = re.sub(r'\s+', ' ', text)
text = text.lower()
text = re.sub(r'\d', ' ', text)
text = re.sub(r'\s+', ' ', text)
```

Date :
Page :

Preparing the dataset

```
sentences = nltk.sent_tokenize(text)
sentences = [nltk.word_tokenize(sentence) for sentence
             in sentences]  # converting all the sentences into words
                            # for removing all the stopwords
for i in range(len(sentences)):
    sentences[i] = [word for word in sentences[i] if word
                    not in stopwords.words('english')]
```

Training the word2vec model

```
model = Word2Vec(sentences, min_count=1)
```

(on the 2, 3, 4, 5 ... etc)

if the word is present less than 1, i am going to just skip that particular word

vocabulary

```
words = model.wv.vocab
```

for each and every word we will get vectors, dimensions, sparse, it will represent word to vectors

Finding word vectors

extracting any word from paragraph

```
vector = model.wv["word"]
```

Most similar words

```
similar = model.wv.most_similar("word")
```

from "freedom"