Text Visualization

Creating a Bag of Words model

→ Break the sentence into words
→ Get Freq distribution of words
→ Sort the words based on freq
→ Pick Top few words
→ Create a bag of words Table

```
import nltk
import re
dataset = nltk.sent_tokenize(paragraph)
for i in range(len(dataset)):
    dataset[i] = dataset[i].lower()
    dataset[i] = re.sub(r'\W', ' ', dataset[i])  //remove special characters
    dataset[i] = re.sub(r'\s+', ' ', dataset[i])
                    // to replace multiple spaces with single space
```

```
word2count = { }
for data in dataset:
    words = nltk.word_tokenize(data)
    for word in words:
        if word not in word2count.keys():
            word2count[word] = 1
        else:
            word2count[word] += 1
```

→ when we are building a BOW on large corpus of data (50k movie reviews). We will end up in a large set of words. So, we will select most frequent "n" words.

```
import heapq
freq_words = heapq.nlargest(100, word2count, key = word2count.get)
                                  ↓ Dictionary

X = [ ]   // this will be our bag of words model.

for data in dataset:
    Vector = [ ]  // going to contain complete vector of sentence/doc
    for word in freq_words:
        If word in nltk.word_tokenize(data):
            Vector.append(1)
        Else:
            Vector.append(0)
    X.append(Vector)   // list of lists

import numpy as np
X = np.asarray(X)
```

→ word frequency can also build using a function as below

```
wordlist = nltk.FreqDist(wordlist)
features = wordlist.keys()
           .values()

Return type = ( [ ] )
                ↓
               dict.
```

Bag of words - Problems

→ All words have the same importance
→ no semantic information preserved
↳ which word is important & which word is not

To overcome this we have the below solution

TF-IDF (Term Frequency - Inverse document Frequency)

* Some semantic information is preserved as uncommon words are given more importance than common words.

Ex:- 'she is beautiful'. Here beautiful will have more importance than 'she' & 'is'

when we change the 'beautiful' to 'ugly' then the meaning of the sentence changes

→ Converting the sentence to lower
→ Converting sentence to a words

TF = Term Frequency

= Term Frequency of word in that document
= TF of the word for each document is not same.

IDF = Inverse document Frequency

= ~~Inverse~~ IDF is not calculated for the document, but for the whole corpus
= IDF for each word will be the same. Through of the document

TF = Formula :- $\frac{\text{(number of occurrences of a word in a document)}}{\text{(number of words in that document)}}$

"to be or not to be"

TF of 'to' = $\frac{1+1}{6}$ = 0.33

Tf of 'be' = $\frac{1+1}{6}$ = 0.33

tf of 'or' = 1/6 = 0.16

IF matrix

words/Docum	Document 1	Docume 2	Document 3
going	0.16	0.16	0.12
to	0.16	0	0.12
today	0.16	0.16	0
i	0	0.16	0.12
am	0	0.16	0.12
it	0.16	0	0
is	0.16	0	0
rain.	0.16	0	0

IDF= Formula

$$\log_e\left(\frac{\text{(number of documents)}}{\text{(Number of document containing word)}}\right)$$

"to be or not to be"

"i have to be"

"you got to be"

to : $\log_e(3/3) = \log(1) = 0$

be : $\log_e(3/3) = 0$

have : $\log_e(3/1)$ =

words	IDF	
going	log(3/3)	0
to	log(3/2)	0.41
today	log(3/2)	0.41
I	log(3/2)	0.41
am	log(3/2)	0.41
it	log(3/1)	1.09
is	log(3/1)	1.09
rain	log(3/1)	1.09

word/ Document	going	to	today	?	am	it	is	rain
	0 ≠ 0.16 = 0	0.09	0.07	0	0	0.17	0.12	0.12
Doc 1	↓	↓	↓	↓	↓	↓	↓	↓
Doc 2	0	0	0.07	0.07	0.07	0	0	
Doc 3	0	0.05	0	0.05	0.05	0	0	0

TFIDF(word) = TF(Document, word) * IDF(words)

→ This method is extensively used in Text classifications, opinion mining etc.,

Why log in IDF

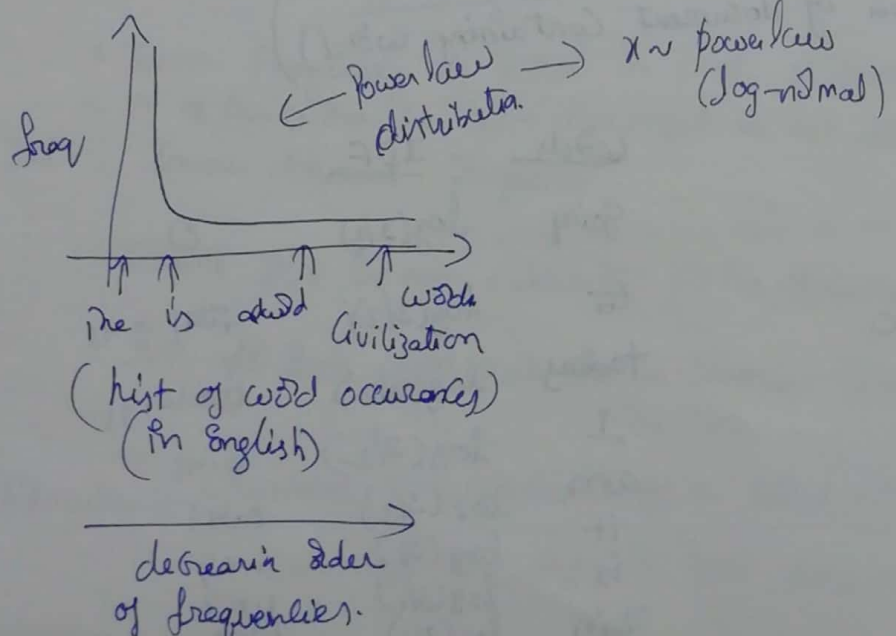
$$IDF(w_i, D_c) = \log\left(\frac{N}{n_i}\right)$$

$\frac{N}{n_i}$ = Total # of Documents / # of docs which contain w_i

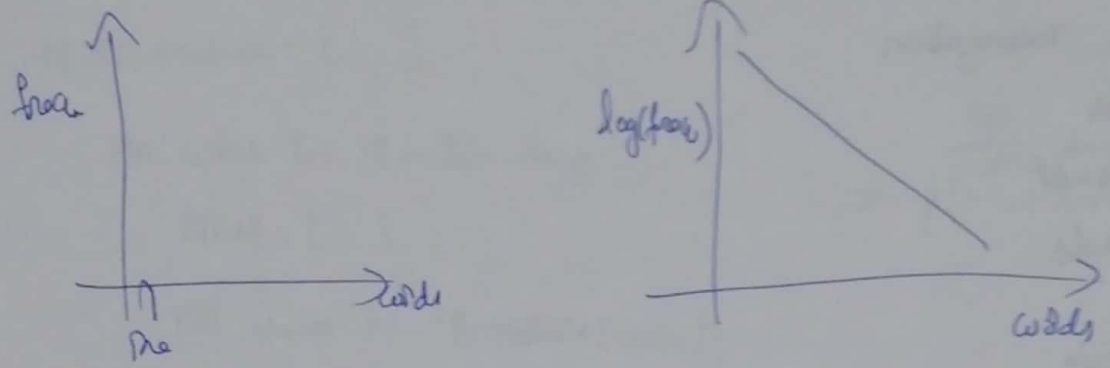
→ 1972 research paper where IDF is introduced.

→ not very strongly based on theory

→ Refering to Zipf's law



According to box-cox Transformation taking log of the Variable.



Practical Reason

	$\frac{N}{n_i}$	$\log_e\left(\frac{N}{n_i}\right)$
the →	1	0
is	1	0
⋮		
Civilization.	$\frac{1000}{1}$	≈ 6.9

(occurin 1 in Every thousand document)

TF*IDF

↳ If word does not occur, the IDF dominates

TF-IDF Model

→ Sentence Tokenization
→ Preprocess
→ word2 count
→ Freq_words

```
# IDF_matrix
dataset = nltk.sent_tokenize(Paragraph)
words_idfs = { }

for word in Freq_words
    doc_count = 0
    For data in dataset:
        If word in nltk.word_tokenize(data):
            doc_count += 1
    word_idfs[word] = np.log(len(dataset)/(doc_count+1))
                                                  ↑
                                                 bias
```

```
# TF-Matrix
tf_matrix { }
for word in freq_words:
    doc_Tf = [ ]
    for data in Sentence:
        frequency = 0
        for w in nltk.word_tokenize(data):
            If w == word:
                frequency += 1
        tf_word = frequ/len(nltk.word_tokenize(data))
        doc_tf.append(tf_word)
    tf_matrix[word] = doc_tf.
```

o/p

```
{'after': [0.02, 0.0, 0.0, 0.02]}
        doc1  doc2 doc3 doc4
    :
}
```

tfIDF Matrix (calculation)

98 word
4 sentence

```
tfidf_matrix = [ ]

For word in tf_matrix.keys();
    tfidf = [ ]
    For value in tf_matrix[word]
        score = value * word_idf[word]
        tfidf.append(score)
    tf_idf_matrix.append(tfidf)
```

o/p

word → [[0.2, 0.3, 0.4, 0.5]
         [ ]
         [ ]]

(doc1, doc2, doc3, doc4)

Converting tfIDF to a ~~Matrix~~ nd array

```
X = np.asarray(X)
```

4 rows * 98 columns