In [67]:

```
print("rocks",lemmatizer.lemmatize("rocks"))

print("corpora",lemmatizer.lemmatize("corpora"))

print("better",lemmatizer.lemmatize("better","a"))  # 2nd argument is pos (part of speech (noun,verb,adj
```

```
rocks rock
corpora corpus
better good
```

In [68]:

```
print("can't",lemmatizer.lemmatize("can't"))
print("what's",lemmatizer.lemmatize("what's"))
print("couldn't",lemmatizer.lemmatize("couldn't"))
print("wasn't",lemmatizer.lemmatize("wasn't"))
```

```
can't can't
what's what's
couldn't couldn't
wasn't wasn't
```

In [69]:

```
print("can't",ps.stem("can't"))
```

```
can't can't
```

In [ ]:

```

```

Stop words

Stop words are words which are filtered out before or after processing of the text.

When applying machine learning to text these words can add a lot of noise. hence we want to remove those irrelevant words.

Stop words are usually reffered to the most common words such as **"and","the","a"** in a language, but there is no single universal list of stop words available.

The list of stop words can change depending on your application you work on.

NLTK tool has a predefined list of stopwords that refers to the most common word.

If you use it in your code for the 1st time you need to download it using the command below.:

In [70]:
```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to /home/punit/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[70]:
```
True
```

In [71]:
```
from nltk.corpus import stopwords
print(stopwords.words("english"))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd",
'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'h
erself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which',
'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been',
'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'i
f', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'betwee
n', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'ou
t', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'wh
y', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'no
t', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 's
hould', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "could
n't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

In [ ]:
```

```

In [81]:
```
stop_word = set(stopwords.words("english"))
```

In [82]:
```
stop_word
```

```
'own',
're',
's',
'same',
'shan',
"shan't",
'she',
"she's",
'should',
"should've",
'shouldn',
"shouldn't",
'so',
'some',
'such',
't',
'than',
'that',
"that'll",
'the'
```

In [74]:
```
sentence = "Cricket is one of the most common games followed in India"
```

In [77]:
```
words = nltk.word_tokenize(sentence)
```

In [78]:
```
words
```

Out[78]:
```
['Cricket',
 'is',
 'one',
 'of',
 'the',
 'most',
 'common',
 'games',
 'followed',
 'in',
 'India']
```

In [83]:
```
cleaned_data = [item for item in words if not item in stop_word ]
```

In [84]:
```
cleaned_data
```

Out[84]:
```
['Cricket', 'one', 'common', 'games', 'followed', 'India']
```

In [ ]:
```

```

Bag of Words

Machine Learning algorithm cannot work with raw text directly, we need to convert the text into vector of numbers

Thus particular process is called as feature extraction.

The **bag of words** model is a popular and simple feature extraction technique used when we word with text.

It describes the occurnace of each word with in a document.

Steps to use :

1) Design the vocabulary of known words (called as tokens)

2) Choose a measure of the presence of known words.

Any Information about the order or structure of words is discarded. Thats why it is called as bag of words.

The model is trying to understand whether a known word occur in a document, but we don't know where is that word in the document.

In [85]:
```
"""
I am Punit
I am a Python developer
I like coding in python
"""
```

Out[85]:
```
'\nI am Punit\nI am a Python developer\nI like coding in python\n'
```

Type *Markdown* and LaTeX: α^2

In [86]:
```
with open('inp_data.txt','r') as data:
    raw_data = data.read().splitlines()

print(raw_data)
```

```
["I like this movie, it's funny", 'I hate this movie', 'This was awesome! I like it ', 'Nice one I love it
']
```

In [ ]:
```

```

Design the vocabulary

```
to get all the unique words from the four loaded sentence ignoring the case, punctuation and one
character token
```

In [87]:
```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
```

In [88]:
```
count_vectorize = CountVectorizer()
```

In [89]:

```
# to create a bag of vector model

bag_of_words = count_vectorize.fit_transform(raw_data)

# show bag of words model

feature_name = count_vectorize.get_feature_names()
pd.DataFrame(bag_of_words.toarray(),columns=feature_name)
```

Out[89]:

	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0	1	0	1	1	0	1	0	0	1	0
1	0	0	1	0	0	0	1	0	0	1	0
2	1	0	0	1	1	0	0	0	0	1	1
3	0	0	0	1	0	1	0	1	1	0	0

In [90]:

```
print(raw_data)
```

```
["I like this movie, it's funny", 'I hate this movie', 'This was awesome! I like it ', 'Nice one I love it
']
```

In [ ]:

```

```

In [ ]:

```
"i live in mumbai", "i am a software developer"


2 / 2
```

TF-IDF

```
One of the problem with scoring word frequency is that the most frequent word in the document start to have the highest score.

These frequent words may or may not contain much information to the model compared with some other domain related specific words.

```

```
One of the technique to fix the problem is to penalize words that are frequent across all the document.

This approach is called as TF-IDF

TF-IDF (also called as Term Frequency - Inverse Document Frequency) is a statistical measure, which is used to evaluate the importance of a word in a document

THE TF-IDF scoring value increases propotionaly to the number of time a word appear in a document.


**Formula**

Term Frequency (TF) = Number of time terms appear in a document / Total number of item in the document


Inverse Term Frequency (ITF) = log(Total number of document / Number of document with tem in it)

TFIDF = TF(term) * IDF(term)
```

In [ ]:

```python

```

In [92]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

tfidf_vectorizer = TfidfVectorizer()
```

In [93]:

```python
values = tfidf_vectorizer.fit_transform(raw_data)
```

In [94]:

```python
feature_name = tfidf_vectorizer.get_feature_names()
pd.DataFrame(values.toarray(),columns=feature_name)
```

Out[94]:

	awesome	funny	hate	it	like	love	movie	nice	one	this	was
0	0.000000	0.571848	0.000000	0.365003	0.450852	0.000000	0.450852	0.000000	0.000000	0.365003	0.000000
1	0.000000	0.000000	0.702035	0.000000	0.000000	0.000000	0.553492	0.000000	0.000000	0.448100	0.000000
2	0.539445	0.000000	0.000000	0.344321	0.425305	0.000000	0.000000	0.000000	0.000000	0.344321	0.539445
3	0.000000	0.000000	0.000000	0.345783	0.000000	0.541736	0.000000	0.541736	0.541736	0.000000	0.000000

In [95]:
```
raw_data
```

Out[95]:
```
["I like this movie, it's funny",
 'I hate this movie',
 'This was awesome! I like it ',
 'Nice one I love it ']
```

In [ ]:
```

```