PARAGAG PUJARI 19MAI0017 LAB 3------12/05/2020

```
In [1]: #1. Mere Wordlist --- STOPWORDS
        #2. Word List with Information (Pronunciation in this case) ---CMU WORD
        LIST
        #3. Word List with Semantic Orientation --- WORDNET
```

```
In [2]: #STOPWORDS
        from nltk.corpus import stopwords
        stopwords.words('english')
```

```
Out[2]: ['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',
         'herself',
         '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',
         'if',
         'or',
         'because',
         'as',
         'until',
         'while',
         'of',
         'at',
         'by',
         'for',
         'with',
         'about',
         'against',
         'between',
         'into',
         'through',
         'during',
         'before',
         'after',
         'above',
         'below',
         'to',
         'from',
         'up',
         'down',
         'in',
         'out',
         'on',
         'off',
         'over',
         'under',
         'again',
         'further',
         'then',
         'once',
         'here',
         'there',
         'when',
         'where',
         'why',
         'how',
         'all',
         'any',
         'both',
         'each',
         'few',
         'more',
         'most',
         'other',
         'some',
         'such',
         'no',
         'nor',
         'not',
         'only',
         'own',
         'same',
         'so',
         'than',
         'too',
         'very',
         's',
         't',
         'can',
         'will',
         'just',
         'don',
         "don't",
         'should',
         "should've",
         'now',
         'd',
         'll',
         'm',
         'o',
         're',
         've',
         'y',
         'ain',
         'aren',
         "aren't",
         'couldn',
         "couldn'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 [3]:
```
#CMU WORDLIST
import nltk
ntries=nltk.corpus.cmudict.entries()
len(ntries)
```

Out[3]:
```
133737
```

In [4]:
```
ntries[:100]
```

Out[4]:
```
[('a', ['AH0']),
 ('a.', ['EY1']),
 ('a', ['EY1']),
 ('a42128',
  ['EY1',
   'F',
   'AO1',
   'R',
   'T',
   'UW1',
   'W',
   'AH1',
   'N',
   'T',
   'UW1',
   'EY1',
   'T']),
 ('aaa', ['T', 'R', 'IH2', 'P', 'AH0', 'L', 'EY1']),
 ('aaberg', ['AA1', 'B', 'ER0', 'G']),
 ('aachen', ['AA1', 'K', 'AH0', 'N']),
 ('aachener', ['AA1', 'K', 'AH0', 'N', 'ER0']),
 ('aaker', ['AA1', 'K', 'ER0']),
 ('aalseth', ['AA1', 'L', 'S', 'EH0', 'TH']),
 ('aamodt', ['AA1', 'M', 'AH0', 'T']),
 ('aancor', ['AA1', 'N', 'K', 'AO2', 'R']),
 ('aardema', ['AA0', 'R', 'D', 'EH1', 'M', 'AH0']),
 ('aardvark', ['AA1', 'R', 'D', 'V', 'AA2', 'R', 'K']),
 ('aaron', ['EH1', 'R', 'AH0', 'N']),
 ("aaron's", ['EH1', 'R', 'AH0', 'N', 'Z']),
 ('aarons', ['EH1', 'R', 'AH0', 'N', 'Z']),
 ('aaronson', ['EH1', 'R', 'AH0', 'N', 'S', 'AH0', 'N']),
 ('aaronson', ['AA1', 'R', 'AH0', 'N', 'S', 'AH0', 'N']),
 ("aaronson's", ['EH1', 'R', 'AH0', 'N', 'S', 'AH0', 'N', 'Z']),
 ("aaronson's", ['AA1', 'R', 'AH0', 'N', 'S', 'AH0', 'N', 'Z']),
 ('aarti', ['AA1', 'R', 'T', 'IY2']),
 ('aase', ['AA1', 'S']),
 ('aasen', ['AA1', 'S', 'AH0', 'N']),
 ('ab', ['AE1', 'B']),
 ('ab', ['EY1', 'B', 'IY1']),
 ('ababa', ['AH0', 'B', 'AA1', 'B', 'AH0']),
 ('ababa', ['AA1', 'B', 'AH0', 'B', 'AH0']),
 ('abacha', ['AE1', 'B', 'AH0', 'K', 'AH0']),
 ('aback', ['AH0', 'B', 'AE1', 'K']),
 ('abaco', ['AE1', 'B', 'AH0', 'K', 'OW2']),
 ('abacus', ['AE1', 'B', 'AH0', 'K', 'AH0', 'S']),
 ('abad', ['AH0', 'B', 'AA1', 'D']),
 ('abadaka', ['AH0', 'B', 'AE1', 'D', 'AH0', 'K', 'AH0']),
 ('abadi', ['AH0', 'B', 'AE1', 'D', 'IY0']),
 ('abadie', ['AH0', 'B', 'AE1', 'D', 'IY0']),
 ('abair', ['AH0', 'B', 'EH1', 'R']),
 ('abalkin', ['AH0', 'B', 'AA1', 'L', 'K', 'IH0', 'N']),
 ('abalone', ['AE2', 'B', 'AH0', 'L', 'OW1', 'N', 'IY0']),
 ('abalos', ['AA0', 'B', 'AA1', 'L', 'OW0', 'Z']),
 ('abandon', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N']),
 ('abandoned', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'D']),
 ('abandoning', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'IH0', 'N
G']),
 ('abandonment',
  ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'M', 'AH0', 'N', 'T']),
 ('abandonments',
  ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'M', 'AH0', 'N', 'T',
'S']),
 ('abandons', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'Z']),
 ('abanto', ['AH0', 'B', 'AE1', 'N', 'T', 'OW0']),
 ('abarca', ['AH0', 'B', 'AA1', 'R', 'K', 'AH0']),
 ('abare', ['AA0', 'B', 'AA1', 'R', 'IY0']),
 ('abascal', ['AE1', 'B', 'AH0', 'S', 'K', 'AH0', 'L']),
 ('abash', ['AH0', 'B', 'AE1', 'SH']),
 ('abashed', ['AH0', 'B', 'AE1', 'SH', 'T']),
 ('abate', ['AH0', 'B', 'EY1', 'T']),
 ('abated', ['AH0', 'B', 'EY1', 'T', 'IH0', 'D']),
 ('abatement', ['AH0', 'B', 'EY1', 'T', 'M', 'AH0', 'N', 'T']),
 ('abatements', ['AH0', 'B', 'EY1', 'T', 'M', 'AH0', 'N', 'T', 'S']),
 ('abates', ['AH0', 'B', 'EY1', 'T', 'S']),
 ('abating', ['AH0', 'B', 'EY1', 'T', 'IH0', 'NG']),
 ('abba', ['AE1', 'B', 'AH0']),
 ('abbado', ['AH0', 'B', 'AA1', 'D', 'OW0']),
 ('abbas', ['AH0', 'B', 'AA1', 'S']),
 ('abbasi', ['AA0', 'B', 'AA1', 'S', 'IY0']),
 ('abbate', ['AA1', 'B', 'EY0', 'T']),
 ('abbatiello', ['AA0', 'B', 'AA0', 'T', 'IY0', 'EH1', 'L', 'OW0']),
 ('abbe', ['AE1', 'B', 'IY0']),
 ('abbe', ['AE0', 'B', 'EY1']),
 ('abbenhaus', ['AE1', 'B', 'AH0', 'N', 'HH', 'AW2', 'S']),
 ('abbett', ['AH0', 'B', 'EH1', 'T']),
 ('abbeville', ['AE1', 'B', 'V', 'IH0', 'L']),
 ('abbey', ['AE1', 'B', 'IY0']),
 ("abbey's", ['AE1', 'B', 'IY0', 'Z']),
 ('abbie', ['AE1', 'B', 'IY0']),
 ('abbitt', ['AE1', 'B', 'IH0', 'T']),
 ('abbot', ['AE1', 'B', 'AH0', 'T']),
 ('abbotstown', ['AE1', 'B', 'AH0', 'T', 'S', 'T', 'AW1', 'N']),
 ('abbott', ['AE1', 'B', 'AH0', 'T']),
 ("abbott's", ['AE1', 'B', 'AH0', 'T', 'S']),
 ('abbottstown', ['AE1', 'B', 'AH0', 'T', 'S', 'T', 'AW1', 'N']),
 ('abboud', ['AH0', 'B', 'UW1', 'D']),
 ('abboud', ['AH0', 'B', 'AW1', 'D']),
 ('abbreviate', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T']),
 ('abbreviated', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'AH
0', 'D']),
 ('abbreviated', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'IH
0', 'D']),
 ('abbreviates', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T',
'S']),
 ('abbreviating',
  ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'IH0', 'NG']),
 ('abbreviation',
  ['AH0', 'B', 'R', 'IY2', 'V', 'IY0', 'EY1', 'SH', 'AH0', 'N']),
 ('abbreviations',
  ['AH0', 'B', 'R', 'IY2', 'V', 'IY0', 'EY1', 'SH', 'AH0', 'N', 'Z']),
 ('abbruzzese', ['AA0', 'B', 'R', 'UW0', 'T', 'S', 'EY1', 'Z', 'IY0']),
 ('abbs', ['AE1', 'B', 'Z']),
 ('abby', ['AE1', 'B', 'IY0']),
 ('abco', ['AE1', 'B', 'K', 'OW0']),
 ('abcotek', ['AE1', 'B', 'K', 'OW0', 'T', 'EH2', 'K']),
 ('abdalla', ['AE2', 'B', 'D', 'AE1', 'L', 'AH0']),
 ('abdallah', ['AE2', 'B', 'D', 'AE1', 'L', 'AH0']),
 ('abdel', ['AE1', 'B', 'D', 'EH2', 'L']),
 ('abdella', ['AE2', 'B', 'D', 'EH1', 'L', 'AH0']),
 ('abdicate', ['AE1', 'B', 'D', 'AH0', 'K', 'EY2', 'T']),
 ('abdicated', ['AE1', 'B', 'D', 'AH0', 'K', 'EY2', 'T', 'AH0', 'D']),
 ('abdicates', ['AE1', 'B', 'D', 'AH0', 'K', 'EY2', 'T', 'S']),
 ('abdicating', ['AE1', 'B', 'D', 'IH0', 'K', 'EY2', 'T', 'IH0', 'N
G'])]
```

In [5]:

```
#3.wordnet
from nltk.corpus import wordnet as wn
wn.synsets('machine')
```

Out[5]:

```
[Synset('machine.n.01'),
 Synset('machine.n.02'),
 Synset('machine.n.03'),
 Synset('machine.n.04'),
 Synset('machine.n.05'),
 Synset('car.n.01'),
 Synset('machine.v.01'),
 Synset('machine.v.02')]
```

In [6]:

```
wn.synset('car.n.01').lemma_names()
```

Out[6]:

```
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

In [7]:

```
#task 2-sample text classifier
```

```
def gender_features(word):
    return{'last_letter':word[-1]}
```

In [8]:

```
gender_features('Obama')
```

Out[8]:

```
{'last_letter': 'a'}
```

In [9]:

```
from nltk.corpus import names
labeled_names = ([(name, 'male') for name in names.words('male.txt')]+
[(name, 'female') for name in names.words('female.txt')])
```

In [10]:

```
import random
random.shuffle(labeled_names)
```

In [11]:

```
featuresets=[(gender_features(n),gender) for (n,gender) in labeled_name
s]
```

In [12]:

```
train_set,test_set=featuresets[500:],featuresets[:500]
```

In [13]:

```
import nltk
classifier=nltk.NaiveBayesClassifier.train(train_set)
```

In [14]:

```
classifier.classify(gender_features('Parag'))
```

Out[14]:

```
'male'
```

In [15]:

```
classifier.classify(gender_features('Obama'))
```

Out[15]:

```
'female'
```

In [16]:

```
classifier.classify(gender_features('Michelle'))
```

Out[16]:

```
'female'
```

In [17]:

```
classifier.classify(gender_features('David'))
```

```
Out[17]: 'male'
```

```
In [18]: print(nltk.classify.accuracy(classifier,test_set))
```

```
0.742
```

```
In [19]: #vectoriser and cosine siilarity
         from sklearn.feature_extraction.text import CountVectorizer
```

```
In [20]: vect=CountVectorizer(binary=True)
         corpus = ["Tessaract is good optical character recognition engine  ",
         "optical character recognition is significant "]
         vect.fit(corpus)
```

```
Out[20]: CountVectorizer(analyzer='word', binary=True, decode_error='strict',
                         dtype=<class 'numpy.int64'>, encoding='utf-8', input='c
         ontent',
                         lowercase=True, max_df=1.0, max_features=None, min_df=
         1,
                         ngram_range=(1, 1), preprocessor=None, stop_words=None,
                         strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                         tokenizer=None, vocabulary=None)
```

```
In [21]: vocab=vect.vocabulary_
```

```
In [22]: for key in sorted(vocab.keys()):
             print("{}:{}".format(key, vocab[key]))
```

```
character:0
engine:1
good:2
is:3
optical:4
recognition:5
significant:6
tessaract:7
```

In [23]:
```
print(vect.transform(["This is a good optical illusion"]).toarray())
```

```
[[0 0 1 1 1 0 0 0]]
```

In [24]:
```
print(vect.transform(corpus).toarray())
```

```
[[1 1 1 1 1 1 0 1]
 [1 0 0 1 1 1 1 0]]
```

In [25]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [26]:
```
similarity = cosine_similarity(vect.transform(["Google Cloud Vision is a character recognition engine"]).toarray(), vect.transform(["OCR is an optical character recognition engine"]).toarray())
```

In [27]:
```
similarity
```

Out[27]:
```
array([[0.89442719]])
```

In [ ]:
```

```