In [1]:

```
#stopword

'''Removing stop words with NLTK in Python.
The process of converting data to something a computer can understand is referred to as pre
One of the major forms of pre-processing is to filter out useless data.
In natural language processing, useless words (data), are referred to as stop words.'''

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\91814\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:

```
#CMU wordlist  : CMUdict is a versioned python wrapper package for The CMU Pronouncing Dict
#The main purpose is to expose the data with little or no assumption on how it is to be use
import nltk
nltk.download('cmudict')
import nltk
entries=nltk.corpus.cmudict.entries()
len(entries)
```

```
[nltk_data] Downloading package cmudict to
[nltk_data]     C:\Users\91814\AppData\Roaming\nltk_data...
[nltk_data]   Package cmudict is already up-to-date!
```

Out[2]:

```
133737
```

In [3]:

```
from nltk.corpus import wordnet as wn

'''Look up a word using synsets();
this function has an optional pos argument which lets you constrain the part of speech of t

wn.synsets('motorcar')
```

Out[3]:

```
[Synset('car.n.01')]
```

In [4]:

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

Out[4]:

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

In [5]:

```
#TASK CLASSIFIER
def gender_features(word):
    return {'last_letter':word[-1]}  #extracting the last letter of the word. Which can be
```

In [6]:

```
gender_features('obama')
```

Out[6]:

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

In [7]:

```
import nltk
nltk.download('names')
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')])
```

```
[nltk_data] Downloading package names to
[nltk_data]     C:\Users\91814\AppData\Roaming\nltk_data...
[nltk_data]   Package names is already up-to-date!
```

In [8]:

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

In [9]:

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

In [10]:

```
train_set,test_test=featuresets[500:],featuresets[:500] #seprating test and train data.
```

In [11]:

```
import nltk
classifier=nltk.NaiveBayesClassifier.train(train_set) #using NaiveBayes classifier and trai
```

In [12]:

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

Out[12]:

```
'male'
```

In [13]:

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

Out[13]:

```
'female'
```

In [14]:

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

Out[14]:

```
'female'
```

In [15]:

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

Out[15]:

```
'female'
```

In [16]:

```
print(nltk.classify.accuracy(classifier,test_test)) # printing the classifier accuracy.
```

```
0.762
```

In [17]:

```
#Task 3 Vectoriser and cosine similarity   vectoriser used for word to number
from sklearn.feature_extraction.text import CountVectorizer
#from sklearn.feature_extraction.text import TfidVectorizer
```

In [18]:

```
vect=CountVectorizer(binary=True)
corpus = ["Tessaract is good optical character recognition engine  ", "optical character re
vect.fit(corpus)
```

Out[18]:

```
CountVectorizer(analyzer='word', binary=True, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='conten
t',
                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 [19]:

```
vocab=vect.vocabulary_
```

In [20]:

```
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 [21]:

```
"this is a good optical illusion"]).toarray()) #converting into boolean single dimension vec
```

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

In [22]:

```
print(vect.transform(corpus).toarray())
```

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

In [23]:

```
from sklearn.metrics.pairwise import cosine_similarity
```

In [24]:

```
#simalrity between two sentence from given corpus #using cosine similarity.
similarity = cosine_similarity(vect.transform(["Google Cloud Vision is a character recognit
```

In [25]:

```
print(similarity) #this show how similar two input sentences are.
```

```
[[0.89442719]]
```

In [ ]: