TASK 1 : LEXICON

In [92]:

```
# Lexicon -> collection of word/phrases + Information (POS; tense Definition..)
#Lexicon has lexical entries -> each entry is word/Phrase -> has a headword(Lemma)

#1.Stopwords
from nltk.corpus import stopwords
stopwords.words("english")
```

Out[92]:

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
```

In [93]:

```
#2. CMU WordList
import nltk
entries = nltk.corpus.cmudict.entries()
len(entries)
```

Out[93]:

```
133737
```

In [94]:

```
entries[:100]
```

Out[94]:

```
[('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']),
```

In [95]:

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

Out[95]:

```
[Synset('abandon.n.01'),
 Synset('wildness.n.01'),
 Synset('abandon.v.01'),
 Synset('abandon.v.02'),
 Synset('vacate.v.02'),
 Synset('abandon.v.04'),
 Synset('abandon.v.05')]
```

In [96]:

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

Out[96]:

```
['abandon', 'wantonness', 'unconstraint']
```

In [97]:

```
wn.synset('abandon.v.01').lemma_names()
```

Out[97]:

```
['abandon']
```

In [98]:

```
wn.synset('abandon.v.02').lemma_names()
```

Out[98]:

```
['abandon', 'give_up']
```

TASK 2 : SIMPLE TEXT CLASSIFIER

In [99]:

```
# TASK 2 - SIMPLE TEXT CLASSIFIER

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

In [100]:

```
gender_features('Trumph')
```

Out[100]:

```
{'Last_letter': 'h'}
```

In [101]:

```
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 [102]:

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

In [103]:

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

In [104]:

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

In [105]:

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

In [106]:

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

Out[106]:

```
'female'
```

In [107]:

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

Out[107]:

```
'female'
```

In [108]:

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

Out[108]:

```
'female'
```

In [109]:

```
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.756
```

TASK 3 : VECTORISERS & COSINE SIMILARITY

In [110]:

```
#TASK 3 - VECTORISERS & COSINE SIMILARITY

from sklearn.feature_extraction.text import CountVectorizer
#from sklearn.feature_extraction.text import TfidfVectorizer
```

In [111]:

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

Out[111]:

```
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 [112]:

```
vocab = vect.vocabulary_
```

In [113]:

```
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 [114]:

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

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

In [115]:

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

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

In [116]:

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

In [117]:

```
similarity = cosine_similarity(vect.transform(["Google Cloud Vision is a character recognit
print(similarity)
```

```
[[0.89442719]]
```

Task 4 : Document Classification

In [118]:

```
# Import movie Review Corpus
from nltk.corpus import movie_reviews

documents = [(list(movie_reviews.words(fileid)), category)
                    for category in movie_reviews.categories()
                            for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
```

In [119]:

```
# Frequency Distribution on Movie reviews corpus
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

# list of the 3000 most frequent words in the overall corpus
word_features = list(all_words)[:3000]

# Define a feature extractor that simply checks whether
# each of words from word_features is present in a given document
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

In [120]:

```
document_features(movie_reviews.words('pos/cv957_8737.txt'))
```

```
 'contains(drive)': False,
 'contains(.)': True,
 'contains(they)': True,
 'contains(get)': True,
 'contains(into)': True,
 'contains(an)': True,
 'contains(accident)': False,
 'contains(one)': True,
 'contains(of)': True,
 'contains(the)': True,
 'contains(guys)': False,
 'contains(dies)': False,
 'contains(but)': True,
 'contains(his)': True,
 'contains(girlfriend)': True,
 'contains(continues)': False,
 'contains(see)': False,
 'contains(him)': True,
 'contains(in)': True,
 'contains(her)': False,
```

In [121]:

```
featuresets = [(document_features(d), c) for (d,c) in documents]

# Split featuresets into training and testing data
train_set, test_set = featuresets[1500:], featuresets[:1500]

# Train the model on training dataset
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [122]:

```
# Compute the accuracy on the test set
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.7753333333333333
```

In [123]:

```
# Find out which features the classifier found to be most informative
# Here the ratio of Negative:Positive or Positive:Negative is given for all words
# "ridiculous" is about 10 times more likely to be negative
# "adult" is about 9.6 times more likely to be positive

classifier.show_most_informative_features(10)
```

```
Most Informative Features
         contains(sorry) = True              neg : pos    =     12.6 : 1.0
    contains(ridiculous) = True              neg : pos    =     10.0 : 1.0
         contains(adult) = True              pos : neg    =      9.6 : 1.0
         contains(awful) = True              neg : pos    =      9.5 : 1.0
        contains(finger) = True              neg : pos    =      7.7 : 1.0
         contains(waste) = True              neg : pos    =      7.5 : 1.0
    contains(whatsoever) = True              neg : pos    =      6.9 : 1.0
           contains(gag) = True              neg : pos    =      6.9 : 1.0
   contains(distracting) = True              pos : neg    =      6.8 : 1.0
         contains(stops) = True              pos : neg    =      6.8 : 1.0
```

In [ ]: