In [1]:
```
#Stopwords
from nltk.corpus import stopwords
stopwords.words('english')
```

Out[1]:
```
['a',
 'about',
 'an',
 'and',
 'are',
 'as',
 'at',
 'be',
 'by',
 'for',
 'from',
 'has',
 'he',
 'in',
 'is',
 'it',
 'its',
 'of',
 'on',
 'that',
 'the',
 'to',
 'was',
 'were',
 'will',
 'with',
 '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",
 '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',
 'an',
 '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 [4]:
```
# CMU wordlist
import nltk
entries = nltk.corpus.cmudict.entries()
len(entries)
entries[:100]
```

Out[4]:
```
[('a', ['AH0']),
 ('a', ['EY1']),
 ('aa', ['AA1']),
 ('aachen',
  ['AA2', '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']),
 ('aardsma', ['AA1', 'R', 'D', 'Z', 'M', 'AH0']),
 ('aaron', ['EH1', 'R', 'AH0', 'N']),
 ('aaronson', ['EH1', 'R', 'AH0', 'N', 'S', 'AH0', 'N']),
 ('aaronsons', ['EH1', 'R', 'AH0', 'N', 'S', 'AH0', 'N', 'Z']),
 ('aarsvold', ['AA1', 'R', 'S', 'V', 'OW2', 'L', 'D']),
 ('aarti', ['AA1', 'R', 'T', 'IY2']),
 ('aase', ['AA1', 'S']),
 ('aasen', ['AA1', 'S', 'AH0', 'N']),
 ('ab', ['AE1', 'B']),
 ('ab', ['EY1', 'B', 'IY0']),
 ('ababa', ['AA0', 'B', 'AA1', 'B', 'AH0']),
 ('abacha', ['AA0', 'B', 'AA1', 'CH', 'AH0']),
 ('abacha', ['AE1', 'B', 'AH0', 'K', 'AH0']),
 ('aback', ['AH0', 'B', 'AE1', 'K']),
 ('abacus', ['AE1', 'B', 'AH0', 'K', 'AH0', 'S']),
 ('abacus', ['AE1', 'B', 'AH0', 'S', 'AH0']),
 ('abad', ['AE1', 'B', 'AH0', 'D']),
 ('abadaka', ['AA0', 'B', 'AA1', 'D', 'AH0', 'K', 'AH0']),
 ('abadi', ['AH0', 'B', 'AA1', 'D', 'IY0']),
 ('abadie', ['AH0', 'B', 'AE1', 'D', 'IY0']),
 ('abair', ['AH0', 'B', 'EH1', 'R']),
 ('abaka', ['AH0', 'B', 'AA1', 'K', 'AH0']),
 ('abalone', ['AE2', 'B', 'AH0', 'L', 'OW1', 'N', 'IY0']),
 ('abalos', ['AH0', 'B', 'AA1', 'L', 'OW0', 'S']),
 ('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', 'NG']),
 ('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']),
 ('abbass', ['AH0', 'B', 'AA1', 'S', 'IY0']),
 ('abbate', ['AH0', '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']),
 ('abbie', ['AE1', 'B', 'IY0']),
 ('abbie', ['AE1', 'B', 'IY0', 'Z']),
 ('abbitt', ['AE1', 'B', 'IH0', 'T']),
 ('abbot', ['AE1', 'B', 'AH0', 'T']),
 ('abbotstown', ['AE1', 'B', 'AH0', 'T', 'S', 'T', 'AW1', 'N']),
 ('abbott', ['AE1', 'B', 'AH0', 'T']),
 ('abbotts', ['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', 'AH0', 'D']),
 ('abbreviates', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'S']),
 ('abbreviates', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'IH0', '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']),
 ('abcir', ['AE1', 'B', 'K', 'ER0']),
 ('abdalla', ['AE2', 'B', 'D', 'AA1', 'T', 'EH2', 'A']),
 ('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', 'NG'])]
```

In [8]:
```
#2. wordnet
from nltk.corpus import wordnet as wn
wn.synsets('good')
```

Out[8]:
```
[Synset('good.n.01'),
 Synset('good.n.02'),
 Synset('good.n.03'),
 Synset('commodity.n.01'),
 Synset('good.a.01'),
 Synset('full.s.06'),
 Synset('good.a.03'),
 Synset('estimable.s.02'),
 Synset('beneficial.s.01'),
 Synset('good.s.06'),
 Synset('good.s.07'),
 Synset('adept.s.01'),
 Synset('good.s.09'),
 Synset('dear.s.02'),
 Synset('dependable.s.04'),
 Synset('good.s.12'),
 Synset('good.s.13'),
 Synset('effective.s.04'),
 Synset('good.s.15'),
 Synset('good.s.16'),
 Synset('good.s.17'),
 Synset('good.s.18'),
 Synset('good.s.19'),
 Synset('good.s.20'),
 Synset('good.s.21'),
 Synset('well.r.01'),
 Synset('thoroughly.r.02')]
```

In [9]:
```
wn.synset('good.n.12').lemma_names()
```

Out[9]:
```
['good', 'right', 'ripe']
```

In [11]:
```
# Task 2 - Simple Text Classifier
def gender_features(word):
    return {'last_letter':word[-1]}
```

In [13]:
```
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 [15]:
```
import random
random.shuffle(labeled_names)
```

In [16]:
```
featuresets = [(gender_features(n),gender) for (n,gender) in labeled_names]
```

In [18]:
```
train_set, test_test = featuresets[500:],featuresets[:500]
```

In [19]:
```
import nltk
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

In [20]:
```
classifier.classify(gender_features('nivedita'))
```

Out[20]:
```
'female'
```

In [21]:
```
classifier.classify(gender_features('chaithanya'))
```

Out[21]:
```
'female'
```

In [22]:
```
print(nltk.classify.accuracy(classifier,test_test))
```

```
0.79
```

In [23]:
```
# Task 3 - VECTORIZERS & COSINE SIMILARITY
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [24]:
```
vect = CountVectorizer(binary = True)
corpus = ["Tesseract is good optical character recognition engine ", "optical character recognition is significant "]
vect.fit(corpus)
```

Out[24]:
```
CountVectorizer(analyzer='word', binary=True, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                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 [25]:
```
vocab = vect.vocabulary_
```

In [26]:
```
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
tesseract:7
```

In [29]:
```
print(vect.transform(["this is a good optical illusion"]).toarray())
```

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

In [30]:
```
print(vect.transform(corpus).toarray())
```

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

In [31]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [34]:
```
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 [35]:
```
similarity
```

Out[35]:
```
array([[0.89442719]])
```

In [ ]:
```

```