In [ ]:

```
'''
The following code uses movie_reviews data from nltk.corpus. The data contains files with movie reviews and
their corresponding labels(positive or negative)
It does text preprocessing such as removing stop words and tokenization.
After preprocessing different classifiers like Naive Bayesian classifier,SVC_classifier,BernoulliNB_classifier,
SGDClassifier_classifier,LogisticRegression_classifier are implemented.
Finally a aggregator classifer which outputs the most voted label and its confidence is implemented.


__author__='Soniya Rode'
__citation__="pythonprogramming"
'''
```

In [78]:

```
import nltk
import random
from nltk.corpus import movie_reviews
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
import pickle
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.naive_bayes import MultinomialNB,BernoulliNB
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.svm import SVC
from nltk.classify import ClassifierI
from statistics import mode
```

In [31]:

```
#Initalise the stop words set and tokenizer to remove punctuations.
stopwords = set(stopwords.words('english'))

tokenizer = RegexpTokenizer(r'\w+')
```

In [32]:

```
#Get all words in the movie reviews
words = []

for w in movie_reviews.words():
    #If the word is not a punctuation mark and not in stopwords add it to the list
    if tokenizer.tokenize(w) and w not in stopwords:
        words.append(w.lower())
        #print(words)


#Get frequencies of all words
words = nltk.FreqDist(words)
```

Get top most 10 common words

In [33]:
```
words.most_common(10)
```

Out[33]:
```
[('film', 9517),
 ('one', 5852),
 ('movie', 5771),
 ('like', 3690),
 ('even', 2565),
 ('good', 2411),
 ('time', 2411),
 ('story', 2169),
 ('would', 2109),
 ('much', 2049)]
```

In [34]:
```
#Get the top 4000 common words
commonWords=list(words.keys())[:4000]
```

In [35]:
```
#There are two categories pos,neg
#for every category, get the files associated with it.
# Add the review words, the category as a tuple to the reviews
reviews = [(list(movie_reviews.words(fileid)), category)
            for category in movie_reviews.categories()
            for fileid in movie_reviews.fileids(category)]

#Shuffle the data since arranged categorywise
#reviews variable contains all reviews with their labels
random.shuffle(reviews)
```

In [36]:
```
#Function to return boolean value for words from the review which a
re amongst the most common words.
#Return True if the word in the review is among the top 4000 words.
def check_commonWords(words):
    words = set(words)
    listOfCommonWords = {}
    for w in commonWords:
        listOfCommonWords[w] = (w in words)

    return listOfCommonWords
```

In [28]:
```
#Now
featuresets = [(check_commonWords(rev), category) for (rev, categor
y) in reviews]
```

In [37]:
```
print(len(featuresets))
```

```
2000
```

In [38]:
```
#split the data into training and testing
training_set = featuresets[:1800]

# set that we'll test against.
testing_set = featuresets[1800:]
```

In [42]:
```
#Use NLTK's Naive Bayes classifier
classifier = nltk.NaiveBayesClassifier.train(training_set)
```

In [43]:
```
print("accuracy:",(nltk.classify.accuracy(classifier, testing_set))
*100)
```

```
accuracy: 80.5
```

In [44]:
```
classifier.show_most_informative_features(15)
```

```
Most Informative Features
                bothered = True              neg : pos    =      8
.9 : 1.0
                 miscast = True              neg : pos    =      8
.1 : 1.0
                 frances = True              pos : neg    =      7
.7 : 1.0
                sundance = True              pos : neg    =      7
.7 : 1.0
              schumacher = True              neg : pos    =      7
.4 : 1.0
           unimaginative = True              neg : pos    =      7
.0 : 1.0
                  shoddy = True              neg : pos    =      7
.0 : 1.0
               atrocious = True              neg : pos    =      7
.0 : 1.0
                jeopardy = True              neg : pos    =      6
.3 : 1.0
                  suvari = True              neg : pos    =      6
.3 : 1.0
                    mena = True              neg : pos    =      6
.3 : 1.0
                  wasted = True              neg : pos    =      5
.7 : 1.0
                 singers = True              pos : neg    =      5
.7 : 1.0
                 bronson = True              neg : pos    =      5
.6 : 1.0
               underwood = True              neg : pos    =      5
.6 : 1.0
```

In [48]:
```
#Store the classifier
Pickle_classifier = open("naivebayes.pickle","wb")
pickle.dump(classifier, Pickle_classifier)
Pickle_classifier.close()
```

Using sklearn classifiers

In [73]:
```
#Logistic Regression Classifier
LogisticRegression_classifier = SklearnClassifier(LogisticRegressio
n(solver='lbfgs'))
LogisticRegression_classifier.train(training_set)
```

Out[73]:
```
<SklearnClassifier(LogisticRegression(C=1.0, class_weight=None, du
al=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lb
fgs',
          tol=0.0001, verbose=0, warm_start=False))>
```

In [108]:
```
#stochastic gradient descent (SGD) classifier
SGDClassifier_classifier = SklearnClassifier(SGDClassifier( max_ite
r=1000))
SGDClassifier_classifier.train(training_set)
```

Out[108]:
```
<SklearnClassifier(SGDClassifier(alpha=0.0001, average=False, clas
s_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=
True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_i
ter=1000,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False))>
```

In [83]:
```
#Support Vector machine Classifier
SVC_classifier = SklearnClassifier(SVC(gamma='auto'))
SVC_classifier.train(training_set)
```

Out[83]:
```
<SklearnClassifier(SVC(C=1.0, cache_size=200, class_weight=None, c
oef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='r
bf',
  max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
  tol=0.001, verbose=False))>
```

In [102]:

```
BernoulliNB_classifier = SklearnClassifier(BernoulliNB())
BernoulliNB_classifier.train(training_set)
print("BernoulliNB_classifier accuracy percent:", (nltk.classify.accuracy(BernoulliNB_classifier, testing_set))*100)
```

```
BernoulliNB_classifier accuracy percent: 81.0
```

In [84]:

```
#Classifier Accuracies:
print("Classifier accuracies :")
print("Naive Bayes Classifier accuracy:",(nltk.classify.accuracy(classifier, testing_set))*100)
print("LogisticRegression_classifier accuracy:", (nltk.classify.accuracy(LogisticRegression_classifier, testing_set))*100)

print("SGDClassifier_classifier accuracy:", (nltk.classify.accuracy(SGDClassifier_classifier, testing_set))*100)
print("SVC_classifier accuracy:", (nltk.classify.accuracy(SVC_classifier, testing_set))*100)
```

```
Classifier accuracies :
Naive Bayes Classifier accuracy: 80.5
LogisticRegression_classifier accuracy: 78.5
SGDClassifier_classifier accuracy: 80.0
SVC_classifier accuracy: 80.0
```

In [99]:

```
'''
Aggregated Classifier takes different classifiers as input. To classify it takes vote from each classifier, and
returns the class label with most number of votes(mode)
The get_confidence method returns the confidence on the vote(class label)

The get_confidence uses mode method from statistics which raisies statistics.StatisticsError
statistics.StatisticsError: no unique mode; found 2 equally common values
Since number of classifiers used in uneven, this error is avoided.
'''
class AggregatedClassifier(ClassifierI):
  def __init__(self, *classifiers):
    self._classifiers = classifiers
  def classify(self, features):
      votes = []
      for classifier in self._classifiers:
        votes.append(classifier.classify(features))
      return mode(votes)
  def get_confidence(self, features):
    votes = []
    for classifier in self._classifiers:
      votes.append(classifier.classify(features))


    confidence = votes.count(mode(votes)) / len(votes)
    return confidence
```

In [104]:

```
agg_classifier = AggregatedClassifier(classifier,SVC_classifier,BernoulliNB_classifier,SGDClassifier_classifier,LogisticRegression_classifier)

#print("voted_classifier accuracy percent:", (nltk.classify.accuracy(voted_classifier, testing_set))*100)
for i in range(20):
    print("Predicted Class label:", agg_classifier.classify(testing_set[i][0]), "Confidence %:",agg_classifier.get_confidence(testing_set[i][0])*100)
```

```
Predicted Class label: pos Confidence %: 100.0
Predicted Class label: neg Confidence %: 80.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: pos Confidence %: 80.0
Predicted Class label: neg Confidence %: 80.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: neg Confidence %: 60.0
Predicted Class label: neg Confidence %: 80.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: pos Confidence %: 100.0
Predicted Class label: pos Confidence %: 100.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: pos Confidence %: 100.0
Predicted Class label: neg Confidence %: 60.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: neg Confidence %: 100.0
Predicted Class label: neg Confidence %: 60.0
Predicted Class label: neg Confidence %: 80.0
```