Part 3: Text Mining (30 points)

Tired of Rotten Tomatoes (http://www.rottentomatoes.com/)? This is your chance to make your own movie review aggregator, slightly-smelly-tomatoes. In this assignment, you will be revisiting some of the NLTK commands from the Text Mining lecture and applying them to a text classification task. First, you'll classify movie reviews in a slightly-more-complicated version of the problem we saw in class. Next, you'll produce document similarity scores using the Reuters Corpus of 10,000 news articles. This will involve loading text data, pre-processing the text (tokenization, stemming, punctuation and stopword removal), performing simple feature generation for text data, and using the generated features to compute document similarity.

In [16]:
```
## Preliminaries

#Show plots in the notebook
%matplotlib inline

from sklearn import datasets, preprocessing, cross_validation, feature_
from sklearn import linear_model, svm, metrics, ensemble, neighbors
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib2
import nltk
import random
from nltk.corpus import movie_reviews

from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import string
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn import svm
from nltk.classify.scikitlearn import SklearnClassifier
```

Document Classification

Here's a cleaner version of the Movie Review Sentiment Classification example from class. Make sure you understand what each of these components does.

In [17]:

```
## Movie Review Sentiment Classification Task

def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words
    return features

def load_movie_review_data():
    # Generate lists of positive and negative reviews
    negids = movie_reviews.fileids('neg')
    posids = movie_reviews.fileids('pos')
    return [negids, posids]


def compute_preprocessing_features():
    # Compute word frequencies in corpus and select the top 2500 words
    all_words_list = movie_reviews.words()

    # ------ REMOVE STOP WORDS
    print len(all_words_list)
    stop = stopwords.words('english')
    all_words_list = [word for word in all_words_list if word not in st
    print "After stop words removal"
    print len(all_words_list)

    # ------ REMOVE PUNCTUATION
    all_words_list = [''.join(c for c in s if c not in string.punctuati
    all_words_list = filter(None, all_words_list)
    print "After removing punctuation"
    print len(all_words_list)

#     # ------ LANCASTER STEMMER
#     lancaster = nltk.LancasterStemmer()
#     all_words_list = [lancaster.stem(t) for t in all_words_list]

    # ------ PORTER STEMMER
    porter = nltk.PorterStemmer()
    all_words_list = [porter.stem(t) for t in all_words_list]

    all_words = nltk.FreqDist(w.lower() for w in all_words_list)
    word_features = [fpair[0] for fpair in list(all_words.most_common(2
    return word_features

def generate_review_features(negids, posids):
    # Generate features for positive and negative reviews
    negfeats = [(document_features(movie_reviews.words(fileids=[f]), wo
    posfeats = [(document_features(movie_reviews.words(fileids=[f]), wo
    return [negfeats, posfeats]


def generate_train_test_split(negfeats, posfeats, train, test):
    # Generate a train-test split
```

```
    combined_feats = negfeats + posfeats;
    trainfeats = [ combined_feats[i] for i in train ]
    testfeats = [ combined_feats[i] for i in test ]
    #print 'train on %d instances, test on %d instances' % (len(trainfe
    return [trainfeats, testfeats]

def train_classifier(trainfeats):
    # Train a classifier
    classifier = nltk.NaiveBayesClassifier.train(trainfeats)
    return classifier

def evaluate_performance(classifier, testfeats):
    # Evaluate classifier
    return nltk.classify.util.accuracy(classifier, testfeats)
    #classifier.show_most_informative_features()

[neg_fileids, pos_fileids] = load_movie_review_data()
word_features = compute_preprocessing_features()
[neg_features, pos_features] = generate_review_features(neg_fileids, po

print "Completed preprocessing"

foldnum=0
review_results = pd.DataFrame()
for train, test in cross_validation.KFold(len(neg_features)+len(pos_fea
                                          shuffle=True, n_folds=5):
    foldnum+=1
    [review_train, review_test] = generate_train_test_split(neg_feature
    clfr = train_classifier(review_train)
    review_results.loc[foldnum,'accuracy'] = evaluate_performance(clfr,

print review_results.mean()
```

```
1583820
After stop words removal
964269
After removing punctuation
717292
Completed preprocessing
accuracy    0.7865
dtype: float64
```

Question 1: Movie Rating Sentiment Classification (15 points):

The movie review example above has a number of issues. The words are not stemmed, stopwo haven't been removed, punctuation is still included, and the classifier only uses absence and presence information for each word. Perform each of these steps below to improve the classifie and report the change in classifier performance.

1. Remove stopwords
2. Remove punctuation
3. Use the Lancaster stemmer to stem the words
4. Instead, use the Porter stemmer to stem the words
5. Using an SVM classifier with word counts instead of boolean flags (using CountVectorizer (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) from scikit-learn)
6. Using the SVM again, but instead of using counts, perform the TF-IDF transformation to get the features (using TfidfTransformer (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html) from scikit-learn)

(Hint: I suggest using functions similar to the ones above, and specifying some boolean flags th control each step. You may need to implement new methods for train_classifier and evaluate_performance, and possibly return different features from compute_preprocessing_features and document_features based on the flags)

Answers 1

1. Remove stopwords Accuracy: 80.85%

2. Remove punctuation Accuracy: 80.08%

3. Lancaster Accuracy: 75.85%

4. Porter stopwords Accuracy: 78.65%

5. SVM with CountVectorizer: 66.6%

6. SVM with TF-IDF: 66.6%

In [12]:

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import string
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.svm import SVR
from sklearn import svm
from nltk.classify.scikitlearn import SklearnClassifier

# ------- BOOLEAN FEATURES
# def document_features(document, word_features):
#     document_words = set(document)
#     features = {}
#     for word in word_features:
#         features['contains({})'.format(word)] = (word in document_wor
#     return features


# # -------- COUNT VECTORIZER
# def document_features(document, word_features):
#     count_vect = CountVectorizer()
#     document_words = count_vect.fit_transform(document)
#     features = {}
#     for word in word_features:
#         count = count_vect.vocabulary_.get(word)
#         if count == None:
#             count = 0
#         features['contains({})'.format(word)] = count
#     return features

# -------TFIDF VECTORIZER
def document_features(document, word_features):
    tfidf_vectorizer = TfidfVectorizer()
    document_words = tfidf_vectorizer.fit_transform(document)
    features = {}
    for word in word_features:
        count = tfidf_vectorizer.vocabulary_.get(word)
        if count == None:
            count = 0
        features['contains({})'.format(word)] = count
    return features

def load_movie_review_data():
    # Generate lists of positive and negative reviews
    negids = movie_reviews.fileids('neg')
    posids = movie_reviews.fileids('pos')
    return [negids, posids]

def compute_preprocessing_features():
    # Compute word frequencies in corpus and select the top 2500 words
    all_words_list = movie_reviews.words()

    # ------ REMOVE STOP WORDS
```

```
    print len(all_words_list)
    stop = stopwords.words('english')
    all_words_list = [word for word in all_words_list if word not in st
    print "After stop words removal"
    print len(all_words_list)

    # ------ REMOVE PUNCTUATION
    all_words_list = [''.join(c for c in s if c not in string.punctuati
    all_words_list = filter(None, all_words_list)
    print "After removing punctuation"
    print len(all_words_list)

#     # ------ LANCASTER STEMMER
#     lancaster = nltk.LancasterStemmer()
#     all_words_list = [lancaster.stem(t) for t in all_words_list]

    # ------ PORTER STEMMER
    porter = nltk.PorterStemmer()
    all_words_list = [porter.stem(t) for t in all_words_list]

    all_words = nltk.FreqDist(w.lower() for w in all_words_list)
    word_features = [fpair[0] for fpair in list(all_words.most_common(2
    return word_features

def generate_review_features(negids, posids):
    # Generate features for positive and negative reviews
    negfeats = [(document_features(movie_reviews.words(fileids=[f]), wo
    posfeats = [(document_features(movie_reviews.words(fileids=[f]), wo
    return [negfeats, posfeats]


def generate_train_test_split(negfeats, posfeats, train, test):
    # Generate a train-test split
    combined_feats = negfeats + posfeats;
    trainfeats = [ combined_feats[i] for i in train ]
    testfeats = [ combined_feats[i] for i in test ]
    #print 'train on %d instances, test on %d instances' % (len(trainfe
    return [trainfeats, testfeats]

def train_classifier(trainfeats):
    # Train a classifier
    classif = SklearnClassifier(LinearSVC())
    classifier = classif.train(trainfeats)
    return classifier

def evaluate_performance(classifier, testfeats):
    # Evaluate classifier
    return nltk.classify.util.accuracy(classifier, testfeats)
    #classifier.show_most_informative_features()

[neg_fileids, pos_fileids] = load_movie_review_data()
word_features = compute_preprocessing_features()

[neg_features, pos_features] = generate_review_features(neg_fileids, po
```

```
print "Completed preprocessing"

foldnum=0
review_results = pd.DataFrame()
for train, test in cross_validation.KFold(len(neg_features)+len(pos_fea
                                          shuffle=True, n_folds=5):
    foldnum+=1
    [review_train, review_test] = generate_train_test_split(neg_feature
    clfr = train_classifier(review_train)
    review_results.loc[foldnum,'accuracy'] = evaluate_performance(clfr,

print review_results.mean()
```

```
1583820
After stop words removal
964269
After removing punctuation
717292
Completed preprocessing
accuracy    0.666
dtype: float64
```

Document Similarity

In this section, you'll build the basic components for document retrieval and relevancy by computing cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity). This measure is actually fairly straightforward to compute: it's simply the dot product of the two document vectors normalized by the document length. Commonly, the document vectors aren't simply word counts, but TF-IDF features for the documents to put greater emphasis on salient words.

For example the cosine similarity, after TF-IDF normalization, of:

- **`a little bird`** and **`a little bird`** is 1
- **`a little bird`** and **`a little bird chirps`** is 0.71
- **`a little bird chirps`** and **`a big dog barks`** is 0 (think about why this is the case, even though they have "a" in common)

You'll be using this same principle to find similar news articles in a large corpus of news data.

Question 2: Finding Similar Documents (15 points)

1. Create a Text Collection from the Reuters Corpus (hint: use the function `TextCollection()`)
2. Tokenize each document in the Reuters Corpus into words
3. Remove punctuation

4. Remove stop words
5. Stem the words using PorterStemmer
6. Compute TF-IDF features for all of the pre-processed documents in the Reuters corpus.
7. Write a function that computes the cosine-similarity between two documents using TF-IDF features. The function should be named cosine_sim and should take two text documents as input -- e.g., cosine_sim(text1, text2).
8. Create a training set of fileids that contain 'train' (similar to what we did for 'pos' and 'neg' in the previous question)
9. Find and report the most similar documents in the training set for the following fileids:
   - test/14826
   - test/14998
   - test/15110
   - test/15197
   - test/15348

Answers 2

1. See bode below for the preprocessing stage

In [24]:
```
from nltk.corpus import reuters

document_ids = reuters.fileids()
doc_words = nltk.TextCollection(reuters)
```

```
1720901
```

In [ ]:
```
# ------ REMOVE PUNCTUATION
doc_words = [''.join(c for c in s if c not in string.punctuation) for s
doc_words = filter(None, doc_words)

# ------ REMOVE STOP WORDS
stop = stopwords.words('english')
doc_words = [word for word in doc_words if word not in stop]

# ------ PORTER STEMMER
porter = nltk.PorterStemmer()
doc_words = [porter.stem(t) for t in doc_words]
```

In [ ]:
```
import sys
reload(sys)
sys.setdefaultencoding('utf-8')

def document_features(document, word_features):
    tfidf_vectorizer = TfidfVectorizer()
    document_words = tfidf_vectorizer.fit_transform(document)
    features = {}

    for word in word_features:
        count = tfidf_vectorizer.vocabulary_.get(word)
        if count == None:
            count = 0
        features['contains({})'.format(word)] = count

    return features

doc_features = []
for fileid in document_ids:
    doc_features.append(document_features(reuters.words(fileid), doc_wo

print len(doc_words)
print len(doc_features)
```

In [ ]:
```
```