IST 664 Natural Language Processing

Homework 4

Harper He

xhe128@syr.edu

Table of Contents

Dataset

In this assignment, I analyzed the review contents from Amazon Product Data provided by Julian McAuley at http://jmcauley.ucsd.edu/data/amazon/ which contains product reviews and metadata from Amazon, including 142.8 million reviews spanning May 1996 – July 2014.

I chose the 5-core subset of the category "Baby / Clothing" for the analysis. 5-core subsets mean that all users and items in the dataset have at least 5 reviews. The dataset was modified into text file and was downloaded from the Assignment folder in the course web site.

Here is the screenshot of the modified review file:

```
reviewerID:A19K65VY14D13R
asin:097293751X
reviewerName:angela
helpful:[0, 0]
reviewText:This book is such a life saver.  It has been so helpful to be able to go back to track trends, answer pediatrician questions, or communicate with
each other when you are up at different times of the night with a newborn.  I think it is one of those things that everyone should be required to have before
they leave the hospital.  We went through all the pages of the newborn version, then moved to the infant version, and will finish up the second infant book
(third total) right as our baby turns 1.  See other things that are must haves for baby at [...]
overall:5.0
summary:Should be required for all new parents!
unixReviewTime:1372464000
reviewTime:06 29, 2013

reviewerID:A2LL1TGG90977E
asin:097293751X
reviewerName:Carter
helpful:[0, 0]
reviewText:Helps me know exactly how my babies day has gone with my mother in law watching him while I go to work.  It also has a section for her to write
notes and let me know anything she may need.  I couldn't be happier with this book.
overall:5.0
summary:Grandmother watching baby
unixReviewTime:1395187200
reviewTime:03 19, 2014
```

Here is the dictionary of the fields provide in the file:

- reviewerID: ID of the reviewer

- asin: ID of the product

- reviewerName: name of the reviewer

- helpful: helpfulness rating of the review, e.g. 2/3

- reviewText: text of the product

- overall: rating of the product

- summary: summary of the review

- unixReviewTime: time of the review (unix time)

- reviewTime: time of the review (raw)

Among all these fields, I used "reviewText" and "overall" for my analysis.

Data Pre-processing

Extract review texts

To extract only review texts, I used the Python code below.

```
# extracts only review texts
reviews=[]
with open("baby.txt") as baby:
    for line in baby:
        if(line.startswith("reviewText:")):
            each_line = line.split("reviewText:")[1]
            each_line = each_line.replace('\n','')
            reviews.append(each_line)
print(len(reviews))
reviews[:3]
```

Below is the sample screenshot of the output.

```
["Perfect for new parents. We were able to keep track of baby's feeding, sleep and diaper change schedule for the fir
st two and a half months of her life. Made life easier when the doctor would ask questions about habits because we ha
d it all right there!",
 'This book is such a life saver.  It has been so helpful to be able to go back to track trends, answer pediatrician
questions, or communicate with each other when you are up at different times of the night with a newborn.  I think it
is one of those things that everyone should be required to have before they leave the hospital.  We went through all
the pages of the newborn version, then moved to the infant version, and will finish up the second infant book (third
total) right as our baby turns 1.  See other things that are must haves for baby at [...]',
 "Helps me know exactly how my babies day has gone with my mother in law watching him while I go to work.  It also ha
s a section for her to write notes and let me know anything she may need.  I couldn't be happier with this book."]
```

Sentence Tokenization

To tokenize each sentence, I used “sent_tokenize” and “word_tokenize” function, below is the output after these manipulations.

```
# sentence tokenization
sent_tokens=sent_tokenize(review_str)
print(len(sent_tokens))
print(sent_tokens[:1])
```

```
743494
['perfect for new parents.']
```

```
# word level tokenization for each sentence
sent_word_token = [word_tokenize(word) for word in sent_tokens]
sent_word_token[:1]
```

```
[['perfect', 'for', 'new', 'parents', '.']]
```

Label review texts

To decide the sentiment of each reviews, I referred to the field “overall”, which indicated the rating of the product. I used regular expression processing to extract the field “overall”.

```
# tag the review as 'pos' or 'neg' based on the user's rating of the product
# Extract overall part from the file
baby_overall = re.findall(r'overall:(.*)',baby_str)
baby_overall[:10]
```

```
['5.0', '5.0', '5.0', '5.0', '4.0', '4.0', '5.0', '5.0', '3.0', '5.0']
```

Then I created a list to store the sentiment of each sentence. The criterion of categories is “rating 4-5: positive, rating 1-2: negative, rating 3: neutral”.

```
# create a tag list based on the rating
senti_tag=[]
for a,b in baby_comb:
    if b == '5.0' or b == '4.0':
        senti_tag==senti_tag.append('pos')
    elif b == '1.0' or b == '2.0':
        senti_tag==senti_tag.append('neg')
    elif b == '3.0':
        senti_tag==senti_tag.append('neu')
print(len(senti_tag))
```

```
160792
```

```
# combine the sentence token and the tag
review_senti=list(zip(sent_word_token, senti_tag))
print(review_senti[:1])
```

```
[(['perfect', 'for', 'new', 'parents', '.'], 'pos')]
```

Since the classification in this assignment was binary, so I removed the neutral reviews and got 2 lists of sentences: one was marked as negative and the other as positive. As we can see, there were 142527 sentence tokens in total and 126525 of them were positive while only 17012 were negative.

```
# remove those neutral tokens
rev_senti=[(token,tag) for (token,tag) in review_senti if tag=='pos' or tag=='neg']
print(len(rev_senti))
print(rev_senti[:1])
# this is my final token to analyze
```

```
143537
[(['perfect', 'for', 'new', 'parents', '.'], 'pos')]
```

```
# provide two lists of sentences, pos and neg
pos_rev = [(token,tag) for (token,tag) in review_senti if tag=='pos']
neg_rev = [(token,tag) for (token,tag) in review_senti if tag=='neg']
print(len(pos_rev))
print(len(neg_rev))
```

```
126525
17012
```

Sample

One challenge I met in the analysis was the limitation of space. Although the process of “bag-of-words” was smooth, the kernel kept dying in the processing of sentiment analysis using subjectivity and representing negation. To ensure the consistency when comparing the performance of different classifiers, I used “random.sample” function to get a sample with 30,000 reviews for analysis.

Stop word filtering

To check if stop words removal was helpful in the analysis, I pruned the word features with a stop word list and made sure that the list didn’t remove any negation or useful function words. Then I rerun the classifiers with different features.

```
###  Classification 4: Stop words removal
# import the stop words list
stopwords = nltk.corpus.stopwords.words('english')
print(len(stopwords))
```

```
179
```

```
# remove some negation words
negationwords.extend(['ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn',
                      'ma', 'mightn', 'mustn', 'needn', 'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn'])
newstopwords = [word for word in stopwords if word not in negationwords]
```

```
# remove stop words from the all words list
new_all_words_list = [word for (sent,cat) in documents for word in sent if word not in newstopwords]
```

```
# continue to define a new all words dictionary, get the 2000 most common as new_word_features
new_all_words = nltk.FreqDist(new_all_words_list)
new_word_items = new_all_words.most_common(2000)
new_word_features = [word for (word,count) in new_word_items]
```

Sentiment Analysis

"Bag-of-words" features

For the classification task and the experiments, I started with the "bag-of-words" features where I collected and selected the most frequent 2000 words in the "sentence_polarity" corpus as the word features. Before building the classifier, I divided the dataset into training and testing set, the former contained nearly 90% of the data while the latter had 10% of the data.

Then I used Naïve Bayes classifier to train and test a classifier on the feature sets. The accuracy of this classifier was 0.8731. Below were top ranked 30 features according to the ratio of one label to the other one.

```
Most Informative Features
              V_ordinary = True              neg : pos    =     12.5 : 1.0
               V_project = True              neg : pos    =     12.5 : 1.0
                V_lesson = True              neg : pos    =     12.5 : 1.0
                V_beauty = True              neg : pos    =     12.5 : 1.0
             V_substance = True              neg : pos    =      7.5 : 1.0
               V_capable = True              neg : pos    =      7.5 : 1.0
               V_channel = True              neg : pos    =      7.5 : 1.0
             V_painfully = True              neg : pos    =      7.5 : 1.0
         V_old-fashioned = True              neg : pos    =      7.5 : 1.0
           V_experiences = True              neg : pos    =      7.5 : 1.0
                 V_rent = True              neg : pos    =      7.5 : 1.0
              V_carries = True              neg : pos    =      7.5 : 1.0
                V_drags = True              neg : pos    =      7.5 : 1.0
             V_nonsense = True              neg : pos    =      7.5 : 1.0
             V_southern = True              neg : pos    =      7.5 : 1.0
              V_monster = True              neg : pos    =      7.5 : 1.0
                V_skill = True              neg : pos    =      7.5 : 1.0
              V_theater = True              neg : pos    =      7.5 : 1.0
                 V_dull = True              neg : pos    =      7.5 : 1.0
              V_capture = True              neg : pos    =      7.5 : 1.0
             V_audience = True              neg : pos    =      7.5 : 1.0
                V_stale = True              neg : pos    =      7.5 : 1.0
                  V_die = True              neg : pos    =      7.5 : 1.0
             V_presents = True              neg : pos    =      7.5 : 1.0
               V_merely = True              neg : pos    =      7.5 : 1.0
              V_tribute = True              neg : pos    =      7.5 : 1.0
           V_performance = True              neg : pos    =      7.5 : 1.0
                V_battle = True              neg : pos    =      7.5 : 1.0
                 V_lame = True              neg : pos    =      7.5 : 1.0
             V_ensemble = True              neg : pos    =      7.5 : 1.0
```

Subjectivity Count features

I read in the subjectivity words from the subjectivity lexicon file created by Janyce Wiebe and her group at the University of Pittsburgh in the MPQA project. Then I created two features that involved counting the positive and negative subjectivity words present in each document.

Then I created a feature extraction function that had all the word features as before, but also had two features ‘positivecount’ and ‘negativecount’. These features contained counts of all the positive and negative subjectivity words, where each weakly subjective word was counted once and each strongly subjective word was counted twice.

Then I used Naïve Bayes classifier to train and test a classifier on the feature sets. The accuracy of this classifier was 0.8697. As we can see, these particular sentiment features did not improve the classification on this dataset.

Negation features

Since negation of opinions was an important part of opinion classification. I negated the word following the negation word "not", "never" and "no" and negation that appears in contractions of the form "doesn", "'", "t".

By using the function defined as below, I went through the document words in order. If a negation occurred, the following words would be added as a not word feature, otherwise it would be added as a regular feature word.

Then I created feature sets as before, used the NOT_features extraction function to train the Naïve Bayes classifier and test the accuracy. The accuracy of this classifier was 0.7817. As we can see, these particular sentiment features did not improve the classification on this dataset. Below were top ranked 30 features according to the ratio of one label to the other one.

```
Most Informative Features
             V_decathlon = False              neg : pos    =     17.4 : 1.0
            V_inevitably = False              neg : pos    =     17.4 : 1.0
          V_unattractive = False              neg : pos    =     17.4 : 1.0
                 V_20lbs = False              neg : pos    =     17.4 : 1.0
               V_NOTseen = True               neg : pos    =     13.9 : 1.0
               V_NOTdoes = True               neg : pos    =     13.5 : 1.0
                  V_ouml = False              neg : pos    =     13.5 : 1.0
              V_meanwhile = False              neg : pos    =     12.5 : 1.0
                 V_speaks = False              neg : pos    =     12.5 : 1.0
                  V_merry = False              neg : pos    =     12.5 : 1.0
                 V_lesson = True               neg : pos    =     12.5 : 1.0
                 V_select = False              neg : pos    =     12.5 : 1.0
              V_estimated = False              neg : pos    =     12.5 : 1.0
                 V_tended = False              neg : pos    =     12.5 : 1.0
              V_oversized = False              neg : pos    =     12.5 : 1.0
                    V_cue = False              neg : pos    =     12.5 : 1.0
                    V_hoo = False              neg : pos    =     12.5 : 1.0
                 V_NOTdue = True               neg : pos    =     12.5 : 1.0
                 V_doubts = False              neg : pos    =     12.5 : 1.0
               V_hampers = False               neg : pos    =     12.5 : 1.0
              V_marketed = False               neg : pos    =     12.5 : 1.0
          V_nursery.this = False               neg : pos    =     12.5 : 1.0
              V_playmats = False               neg : pos    =     12.5 : 1.0
                   V_dvd = False               neg : pos    =     12.5 : 1.0
               V_project = True                neg : pos    =     12.5 : 1.0
          V_misalignment = False               neg : pos    =     12.5 : 1.0
               V_slumber = False               neg : pos    =     12.5 : 1.0
                  V_sunk = False               neg : pos    =     12.5 : 1.0
                V_policy = False               neg : pos    =     12.5 : 1.0
            V_assortment = False               neg : pos    =     12.5 : 1.0
```

Comparison

As mentioned above, the sentiment analysis was rerun after the stop words were removed. And the accuracies of different classifier were listed below.

	Accuracy	Accuracy (Filter by stop words)
"Bag-of-words" features	0.8713	0.8720
Subjectivity Count features	0.8697	0.8703
Negation features	0.7817	0.7697

As we can see, when 30,000 sample was used and stop words were not removed, the "bag-of-words" features had the highest accuracy, about 87.13%. While negation features performed worse than the other two features with accuracy as 78.17%.

After the stop words were removed, the accuracy of the "bag-of-words" and subjectivity count features increased while the accuracy of negation features decreased.

For this specific case where a sample of 30,000 was considered, I can conclude that the "bag-of-words" features with stop words removed gave us the highest accuracy.

Regarding the future development of the analysis, except using the whole dataset instead of sample, we can use one of the other sentiment lexicons like LIWC or other classifiers like Support Vector Machines. Also, we can extract types of features like bigrams in addition to just words/unigrams or use the words together with a POS tagger.

Appendix: Python Code & Output

In [ ]:

```
# IST 664 Natural Language Processing
# Homework 4
# Harper He
# xhe128
```

In [1]:

```
# import packages
import nltk
from nltk.tokenize import *
from nltk.corpus import *
import random
```

In [2]:

```
# open the file and check the file type
baby=open('baby.txt').readlines()
print(type(baby))
```

```
<class 'list'>
```

In [3]:

```
# convert the list back to string
baby_str=''
for i in baby:
    baby_str=baby_str+i
print(baby_str[:200])
```

```
reviewerID:A1HK2FQW6KXQB2
asin:097293751X
reviewerName:Amanda Johnsen "Amanda E. Johnsen"
helpful:[0, 0]
reviewText:Perfect for new parents. We were able to keep track of baby's feeding, sleep and dia
```

In [4]:

```
# check the first 10 lines
baby[:10]
```

Out[4]:

```
['reviewerID:A1HK2FQW6KXQB2\n',
 'asin:097293751X\n',
 'reviewerName:Amanda Johnsen "Amanda E. Johnsen"\n',
 'helpful:[0, 0]\n',
 "reviewText:Perfect for new parents. We were able to keep track of baby's feeding, sleep and diap
er change schedule for the first two and a half months of her life. Made life easier when the doct
or would ask questions about habits because we had it all right there!\n",
 'overall:5.0\n',
 'summary:Awesine\n',
 'unixReviewTime:1373932800\n',
 'reviewTime:07 16, 2013\n',
 '\n']
```

In [5]:

```
# extracts only review texts
reviews=[]
with open("baby.txt") as baby:
    for line in baby:
        if(line.startswith("reviewText:")):
            each_line = line.split("reviewText:")[1]
            each_line = each_line.replace('\n','')
            reviews.append(each_line)
print(len(reviews))
```

```
print(len(reviews))
reviews[:3]
```

```
160792
```

Out[5]:

```
["Perfect for new parents. We were able to keep track of baby's feeding, sleep and diaper change s
chedule for the first two and a half months of her life. Made life easier when the doctor would as
k questions about habits because we had it all right there!",
 'This book is such a life saver.  It has been so helpful to be able to go back to track trends,
answer pediatrician questions, or communicate with each other when you are up at different times o
f the night with a newborn.  I think it is one of those things that everyone should be required to
have before they leave the hospital.  We went through all the pages of the newborn version, then m
oved to the infant version, and will finish up the second infant book (third total) right as our b
aby turns 1.  See other things that are must haves for baby at [...]',
 "Helps me know exactly how my babies day has gone with my mother in law watching him while I go t
o work.  It also has a section for her to write notes and let me know anything she may need.  I co
uldn't be happier with this book."]
```

In [6]:

```
# lowercase the list
review_lower = [w.lower() for w in reviews]
```

In [7]:

```
# convert list to string for tokenization
review_str=''
for i in review_lower:
    review_str=review_str+i
review_str[:200]
```

Out[7]:

```
"perfect for new parents. we were able to keep track of baby's feeding, sleep and diaper change sc
hedule for the first two and a half months of her life. made life easier when the doctor would ask
ques"
```

In [8]:

```
# sentence tokenization
sent_tokens=sent_tokenize(review_str)
print(len(sent_tokens))
print(sent_tokens[:1])
```

```
743494
['perfect for new parents.']
```

In [9]:

```
# word level tokenization for each sentence
sent_word_token = [word_tokenize(word) for word in sent_tokens]
sent_word_token[:1]
```

Out[9]:

```
[['perfect', 'for', 'new', 'parents', '.']]
```

In [10]:

```
# tag the review as 'pos' or 'neg' based on the user's rating of the product
# Extract overall part from the file
baby_overall = re.findall(r'overall:(.*)',baby_str)
baby_overall[:10]
```

Out[10]:

```
['5.0', '5.0', '5.0', '5.0', '4.0', '4.0', '5.0', '5.0', '3.0', '5.0']
```

In [11]:

```
# combine the sentence token and the rating
baby_comb = list(zip(sent_word_token, baby_overall))
print(len(baby_comb))
print(baby_comb[:1])
```

```
160796
[(['perfect', 'for', 'new', 'parents', '.'], '5.0')]
```

In [12]:

```
# create a tag list based on the rating
senti_tag=[]
for a,b in baby_comb:
    if b == '5.0' or b == '4.0':
        senti_tag==senti_tag.append('pos')
    elif b == '1.0' or b == '2.0':
        senti_tag==senti_tag.append('neg')
    elif b == '3.0':
        senti_tag==senti_tag.append('neu')
print(len(senti_tag))
```

```
160792
```

In [13]:

```
# combine the sentence token and the tag
review_senti=list(zip(sent_word_token, senti_tag))
print(review_senti[:1])
```

```
[(['perfect', 'for', 'new', 'parents', '.'], 'pos')]
```

In [14]:

```
# remove those neutral tokens
rev_senti=[(token,tag) for (token,tag) in review_senti if tag=='pos' or tag=='neg']
print(len(rev_senti))
print(rev_senti[:1])
# this is my final token to analyze
```

```
143537
[(['perfect', 'for', 'new', 'parents', '.'], 'pos')]
```

In [16]:

```
rev_senti=random.sample(rev_senti,30000)
```

In [17]:

```
# provide two lists of sentences, pos and neg
pos_rev = [(token,tag) for (token,tag) in review_senti if tag=='pos']
neg_rev = [(token,tag) for (token,tag) in review_senti if tag=='neg']
print(len(pos_rev))
print(len(neg_rev))
```

```
126525
17012
```

In [18]:

```
### Classification 1
### "Bag of words" features in the sentence_polarity corpus
# get the sentence corpus and look at some sentences
sentences = sentence_polarity.sents()
print(len(sentences))
print(type(sentences))
```

```
print(sentence_polarity.categories())
```

```
10662
<class 'nltk.corpus.reader.util.ConcatenatedCorpusView'>
['neg', 'pos']
```

In [19]:

```
# look at the sentences by category to see how many positive and negative
pos_sents = sentence_polarity.sents(categories='pos')
print(len(pos_sents))
neg_sents = sentence_polarity.sents(categories='neg')
print(len(neg_sents))
```

```
5331
5331
```

In [20]:

```
# setup the movie reviews sentences for classification
# create a list of documents, each document is one sentence as a list of words paired with categor
y
documents = [(sent, cat) for cat in sentence_polarity.categories()
             for sent in sentence_polarity.sents(categories=cat)]
# look at the first and last documents - consists of all the words in the review
# followed by the category
print(documents[0])
print(documents[-1])
```

```
(['simplistic', ',', 'silly', 'and', 'tedious', '.'], 'neg')
(['provides', 'a', 'porthole', 'into', 'that', 'noble', ',', 'trembling', 'incoherence', 'that', '
defines', 'us', 'all', '.'], 'pos')
```

In [21]:

```
# randomly reorder documents
random.shuffle(documents)
```

In [22]:

```
# get all words from all movie_reviews and put into a frequency distribution
all_words_list = [word for (sent,cat) in documents for word in sent]
all_words = nltk.FreqDist(all_words_list)
# get the 2000 most frequently appearing keywords in the corpus
word_items = all_words.most_common(2000)
word_features = [word for (word,count) in word_items]
print(word_features[:50])
```

```
['.', 'the', ',', 'a', 'and', 'of', 'to', 'is', 'in', 'that', 'it', 'as', 'but', 'with', 'film',
'this', 'for', 'its', 'an', 'movie', "it's", 'be', 'on', 'you', 'not', 'by', 'about', 'one', 'more
', 'like', 'has', 'are', 'at', 'from', 'than', '"', 'all', '--', 'his', 'have', 'so', 'if', 'or',
'story', 'i', 'too', 'just', 'who', 'into', 'what']
```

In [23]:

```
# define features (keywords) of a document by using just the words, or a BOW/unigram baseline
# each feature is 'contains(keyword)' and is true or false depending
# on whether that keyword is in the document
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    return features
```

In [24]:

```
# get features sets for a document, including keyword features and category feature
```

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

In [25]:

```
word_train_set,word_test_set = featuresets[3000:],featuresets[:3000]
```

In [26]:

```
classifier=nltk.NaiveBayesClassifier.train(word_train_set)
print(nltk.classify.accuracy(classifier,word_test_set))
```

```
0.8713333333333333
```

In [27]:

```
# show which features of classifier are most informative
classifier.show_most_informative_features(30)
```

```
Most Informative Features
              V_ordinary = True              neg : pos    =     12.5 : 1.0
               V_project = True              neg : pos    =     12.5 : 1.0
                V_lesson = True              neg : pos    =     12.5 : 1.0
                V_beauty = True              neg : pos    =     12.5 : 1.0
             V_substance = True              neg : pos    =      7.5 : 1.0
               V_capable = True              neg : pos    =      7.5 : 1.0
               V_channel = True              neg : pos    =      7.5 : 1.0
             V_painfully = True              neg : pos    =      7.5 : 1.0
          V_old-fashioned = True             neg : pos    =      7.5 : 1.0
            V_experiences = True             neg : pos    =      7.5 : 1.0
                  V_rent = True              neg : pos    =      7.5 : 1.0
               V_carries = True              neg : pos    =      7.5 : 1.0
                 V_drags = True              neg : pos    =      7.5 : 1.0
              V_nonsense = True              neg : pos    =      7.5 : 1.0
              V_southern = True              neg : pos    =      7.5 : 1.0
               V_monster = True              neg : pos    =      7.5 : 1.0
                 V_skill = True              neg : pos    =      7.5 : 1.0
               V_theater = True              neg : pos    =      7.5 : 1.0
                  V_dull = True              neg : pos    =      7.5 : 1.0
               V_capture = True              neg : pos    =      7.5 : 1.0
              V_audience = True              neg : pos    =      7.5 : 1.0
                 V_stale = True              neg : pos    =      7.5 : 1.0
                   V_die = True              neg : pos    =      7.5 : 1.0
              V_presents = True              neg : pos    =      7.5 : 1.0
                V_merely = True              neg : pos    =      7.5 : 1.0
               V_tribute = True              neg : pos    =      7.5 : 1.0
            V_performance = True             neg : pos    =      7.5 : 1.0
                 V_battle = True             neg : pos    =      7.5 : 1.0
                  V_lame = True              neg : pos    =      7.5 : 1.0
              V_ensemble = True              neg : pos    =      7.5 : 1.0
```

In [28]:

```
### Classification 2 - add features
### Using a sentiment lexicon with scores or counts: Subjectivity
SLpath="subjclueslen1-HLTEMNLP05.tff"
def readSubjectivity(path):
    flexicon = open(path, 'r')
    # initialize an empty dictionary
    sldict = { }
    for line in flexicon:
        fields = line.split()   # default is to split on whitespace
        # split each field on the '=' and keep the second part as the value
        strength = fields[0].split("=")[1]
        word = fields[2].split("=")[1]
        posTag = fields[3].split("=")[1]
        stemmed = fields[4].split("=")[1]
        polarity = fields[5].split("=")[1]
        if (stemmed == 'y'):
            isStemmed = True
        else:
            isStemmed = False
        # put a dictionary entry with the word as the keyword
```

```
        #     and a list of the other values
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict
```

In [29]:

```
SL = readSubjectivity(SLpath)
```

In [30]:

```
# how many words are in the dictionary
len(SL.keys())
```

Out[30]:

```
6885
```

In [31]:

```
def SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    # count variables for the 4 classes of subjectivity
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
```

In [34]:

```
SL_featuresets = [(SL_features(d, word_features, SL), c) for (d, c) in rev_senti]
```

In [36]:

```
# show just the two sentiment lexicon features in document 0
print(SL_featuresets[0][0]['positivecount'])
print(SL_featuresets[0][0]['negativecount'])
```

```
3
1
```

In [37]:

```
# retrain the classifier using these features
train_set, test_set = SL_featuresets[3000:], SL_featuresets[:3000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

Out[37]:

```
0.8696666666666667
```

In [38]:

In [38]:

```
###  Classification 3: Negation words - add features
# Negation words "not", "never" and "no"
# Not can appear in contractions of the form "doesn't"
for sent in list(sentences)[:50]:
   for word in sent:
     if (word.endswith("n't")):
       print(sent)
```

```
['there', 'is', 'a', 'difference', 'between', 'movies', 'with', 'the', 'courage', 'to', 'go',
'over', 'the', 'top', 'and', 'movies', 'that', "don't", 'care', 'about', 'being', 'stupid']
['a', 'farce', 'of', 'a', 'parody', 'of', 'a', 'comedy', 'of', 'a', 'premise', ',', 'it', "isn't",
'a', 'comparison', 'to', 'reality', 'so', 'much', 'as', 'it', 'is', 'a', 'commentary', 'about',
'our', 'knowledge', 'of', 'films', '.']
['i', "didn't", 'laugh', '.', 'i', "didn't", 'smile', '.', 'i', 'survived', '.']
['i', "didn't", 'laugh', '.', 'i', "didn't", 'smile', '.', 'i', 'survived', '.']
['most', 'of', 'the', 'problems', 'with', 'the', 'film', "don't", 'derive', 'from', 'the',
'screenplay', ',', 'but', 'rather', 'the', 'mediocre', 'performances', 'by', 'most', 'of', 'the',
'actors', 'involved']
['the', 'lack', 'of', 'naturalness', 'makes', 'everything', 'seem', 'self-consciously', 'poetic',
'and', 'forced', '.', '.', '.', "it's", 'a', 'pity', 'that', "[nelson's]", 'achievement',
"doesn't", 'match', 'his', 'ambition', '.']
```

In [39]:

```
# this list of negation words includes some "approximate negators" like hardly and rarely
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', '
scarcely', 'rarely', 'seldom', 'neither', 'nor']
```

In [40]:

```
# One strategy with negation words is to negate the word following the negation word
#   other strategies negate all words up to the next punctuation
# Strategy is to go through the document words in order adding the word features,
#   but if the word follows a negation words, change the feature to negated word
# Start the feature set with all 2000 word features and 2000 Not word features set to false
def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or (word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in word_features)
        else:
            features['V_{}'.format(word)] = (word in word_features)
    return features
```

In [41]:

```
# define the feature sets
NOT_featuresets = [(NOT_features(d, word_features, negationwords), c) for (d, c) in rev_senti]
# show the values of a couple of example features
print(NOT_featuresets[0][0]['V_NOTcare'])
print(NOT_featuresets[0][0]['V_always'])
```

```
False
False
```

In [42]:

```
train_set, test_set = NOT_featuresets[3000:], NOT_featuresets[:3000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
```

Out[42]:

```
0.7816666666666666
```

In [43]:

```
classifier.show_most_informative_features(30)
```

```
Most Informative Features
             V_decathlon = False              neg : pos    =     17.4 : 1.0
            V_inevitably = False              neg : pos    =     17.4 : 1.0
          V_unattractive = False              neg : pos    =     17.4 : 1.0
                 V_20lbs = False              neg : pos    =     17.4 : 1.0
               V_NOTseen = True               neg : pos    =     13.9 : 1.0
               V_NOTdoes = True               neg : pos    =     13.5 : 1.0
                  V_ouml = False              neg : pos    =     13.5 : 1.0
              V_meanwhile = False              neg : pos    =     12.5 : 1.0
                 V_speaks = False              neg : pos    =     12.5 : 1.0
                 V_merry = False               neg : pos    =     12.5 : 1.0
                 V_lesson = True               neg : pos    =     12.5 : 1.0
                 V_select = False              neg : pos    =     12.5 : 1.0
              V_estimated = False              neg : pos    =     12.5 : 1.0
                 V_tended = False              neg : pos    =     12.5 : 1.0
              V_oversized = False              neg : pos    =     12.5 : 1.0
                    V_cue = False              neg : pos    =     12.5 : 1.0
                    V_hoo = False              neg : pos    =     12.5 : 1.0
                 V_NOTdue = True               neg : pos    =     12.5 : 1.0
                 V_doubts = False              neg : pos    =     12.5 : 1.0
                V_hampers = False              neg : pos    =     12.5 : 1.0
               V_marketed = False              neg : pos    =     12.5 : 1.0
          V_nursery.this = False               neg : pos    =     12.5 : 1.0
              V_playmats = False               neg : pos    =     12.5 : 1.0
                   V_dvd = False               neg : pos    =     12.5 : 1.0
               V_project = True                neg : pos    =     12.5 : 1.0
          V_misalignment = False               neg : pos    =     12.5 : 1.0
               V_slumber = False               neg : pos    =     12.5 : 1.0
                  V_sunk = False               neg : pos    =     12.5 : 1.0
                V_policy = False               neg : pos    =     12.5 : 1.0
            V_assortment = False               neg : pos    =     12.5 : 1.0
```

In [44]:

```
###  Classification 4: Stop words removal
# import the stop words list
stopwords = nltk.corpus.stopwords.words('english')
print(len(stopwords))
```

```
179
```

In [47]:

```
# remove some negation words
negationwords.extend(['ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn',
                      'ma', 'mightn', 'mustn', 'needn', 'shan', 'shouldn', 'wasn', 'weren', 'won',
wouldn'])
newstopwords = [word for word in stopwords if word not in negationwords]
```

In [48]:

```
# remove stop words from the all words list
new_all_words_list = [word for (sent,cat) in documents for word in sent if word not in newstopwords
]
```

In [50]:

```
# continue to define a new all words dictionary, get the 2000 most common as new_word_features
new_all_words = nltk.FreqDist(new_all_words_list)
new_word_items = new_all_words.most_common(2000)
new_word_features = [word for (word,count) in new_word_items]
```

In [60]:

```
### rerun the "bag of words"
```

```
def document_new_features(document, new_word_features):
    document_words = set(document)
    features = {}
    for word in new_word_features:
        features['V_{}'.format(word)] = (word in document_words)
    return features
```

In [61]:

```
# get features sets for a document, including keyword features and category feature
new_featuresets = [(document_new_features(d, new_word_features), c) for (d, c) in rev_senti]
```

In [62]:

```
new_word_train_set,new_word_test_set = new_featuresets[3000:],new_featuresets[:3000]
```

In [63]:

```
new_classifier=nltk.NaiveBayesClassifier.train(new_word_train_set)
print(nltk.classify.accuracy(new_classifier,new_word_test_set))
```

```
0.872
```

In [56]:

```
# show which features of classifier are most informative
classifier.show_most_informative_features(30)
```

```
Most Informative Features
             V_decathlon = False              neg : pos    =     17.4 : 1.0
           V_inevitably = False              neg : pos    =     17.4 : 1.0
         V_unattractive = False              neg : pos    =     17.4 : 1.0
                V_20lbs = False              neg : pos    =     17.4 : 1.0
              V_NOTseen = True               neg : pos    =     13.9 : 1.0
              V_NOTdoes = True               neg : pos    =     13.5 : 1.0
                 V_ouml = False              neg : pos    =     13.5 : 1.0
             V_meanwhile = False              neg : pos    =     12.5 : 1.0
                V_speaks = False              neg : pos    =     12.5 : 1.0
                 V_merry = False              neg : pos    =     12.5 : 1.0
                V_lesson = True               neg : pos    =     12.5 : 1.0
                V_select = False              neg : pos    =     12.5 : 1.0
             V_estimated = False              neg : pos    =     12.5 : 1.0
                V_tended = False              neg : pos    =     12.5 : 1.0
             V_oversized = False              neg : pos    =     12.5 : 1.0
                   V_cue = False              neg : pos    =     12.5 : 1.0
                   V_hoo = False              neg : pos    =     12.5 : 1.0
               V_NOTdue = True               neg : pos    =     12.5 : 1.0
               V_doubts = False              neg : pos    =     12.5 : 1.0
              V_hampers = False              neg : pos    =     12.5 : 1.0
             V_marketed = False              neg : pos    =     12.5 : 1.0
         V_nursery.this = False              neg : pos    =     12.5 : 1.0
             V_playmats = False              neg : pos    =     12.5 : 1.0
                  V_dvd = False              neg : pos    =     12.5 : 1.0
              V_project = True               neg : pos    =     12.5 : 1.0
         V_misalignment = False              neg : pos    =     12.5 : 1.0
              V_slumber = False              neg : pos    =     12.5 : 1.0
                 V_sunk = False              neg : pos    =     12.5 : 1.0
               V_policy = False              neg : pos    =     12.5 : 1.0
           V_assortment = False              neg : pos    =     12.5 : 1.0
```

In [57]:

```
### rerun the subjectivity to compare the accuracy
def new_SL_features(document, new_word_features, SL):
    document_words = set(document)
    features = {}
    for word in new_word_features:
        features['V_{}'.format(word)] = (word in document_words)
    # count variables for the 4 classes of subjectivity
    weakPos = 0
    strongPos = 0
```

```
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
```

In [58]:

```
new_SL_featuresets = [(new_SL_features(d, new_word_features, SL), c) for (d, c) in rev_senti]
```

In [59]:

```
# retrain the classifier using these features
new_train_set, new_test_set = new_SL_featuresets[3000:], new_SL_featuresets[:3000]
classifier = nltk.NaiveBayesClassifier.train(new_train_set)
nltk.classify.accuracy(classifier, new_test_set)
```

Out[59]:

```
0.8703333333333333
```

In [64]:

```
### rerun the representing negation
# Start the feature set with all 2000 word features and 2000 Not word features set to false
def new_NOT_features(document, new_word_features, negationwords):
    features = {}
    for word in new_word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or (word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in new_word_features)
        else:
            features['V_{}'.format(word)] = (word in new_word_features)
    return features
```

In [65]:

```
# define the feature sets
new_NOT_featuresets = [(new_NOT_features(d, new_word_features, negationwords), c) for (d, c) in rev
_senti]
# show the values of a couple of example features
print(new_NOT_featuresets[0][0]['V_NOTcare'])
print(new_NOT_featuresets[0][0]['V_always'])
```

```
False
False
```

In [66]:

```
new_train_set, new_test_set = new_NOT_featuresets[3000:], new_NOT_featuresets[:3000]
classifier = nltk.NaiveBayesClassifier.train(new_train_set)
nltk.classify.accuracy(classifier, new_test_set)
```

Out[66]:

0.7696666666666667

In [67]:

```
classifier.show_most_informative_features(30)
```

```
Most Informative Features
             V_decathlon = False              neg : pos    =     17.4 : 1.0
            V_inevitably = False              neg : pos    =     17.4 : 1.0
          V_unattractive = False              neg : pos    =     17.4 : 1.0
                 V_20lbs = False              neg : pos    =     17.4 : 1.0
               V_NOTseen = True               neg : pos    =     13.9 : 1.0
                  V_ouml = False              neg : pos    =     13.5 : 1.0
               V_NOTdoes = False              neg : pos    =     13.5 : 1.0
             V_meanwhile = False              neg : pos    =     12.5 : 1.0
                 V_merry = False              neg : pos    =     12.5 : 1.0
                V_lesson = True               neg : pos    =     12.5 : 1.0
                V_select = False              neg : pos    =     12.5 : 1.0
             V_estimated = False              neg : pos    =     12.5 : 1.0
                V_tended = False              neg : pos    =     12.5 : 1.0
             V_oversized = False              neg : pos    =     12.5 : 1.0
                   V_cue = False              neg : pos    =     12.5 : 1.0
                   V_hoo = False              neg : pos    =     12.5 : 1.0
                V_NOTdue = True               neg : pos    =     12.5 : 1.0
                V_speaks = True               neg : pos    =     12.5 : 1.0
                V_doubts = False              neg : pos    =     12.5 : 1.0
               V_hampers = False              neg : pos    =     12.5 : 1.0
              V_marketed = False              neg : pos    =     12.5 : 1.0
          V_nursery.this = False              neg : pos    =     12.5 : 1.0
              V_playmats = False              neg : pos    =     12.5 : 1.0
                   V_dvd = False              neg : pos    =     12.5 : 1.0
               V_project = True               neg : pos    =     12.5 : 1.0
          V_misalignment = False              neg : pos    =     12.5 : 1.0
               V_slumber = False              neg : pos    =     12.5 : 1.0
                  V_sunk = False              neg : pos    =     12.5 : 1.0
                V_policy = False              neg : pos    =     12.5 : 1.0
            V_assortment = False              neg : pos    =     12.5 : 1.0
```