NLP Lab Session
Week 11, November 10, 2011
Constructing Feature Sets for Opinion Classification in the NLTK

Getting Started

As usual, we will work together through a series of small examples using the IDLE window that will be described in this lab document. However, for purposes of using cut-and-paste to put examples into IDLE, the examples can also be found in a python file on the iLMS system, under Resources, or on the web server at classes.ischool.syr.edu/ist664/NLPFall2011/LabExamples

LabWeek11classifyopinion.py or LabWeek11classifyopinion.txt
Subjectivity.py or Subjectivity.txt
subjclueslen1-HLTEMNLP05.tff

Open an IDLE window. Use the File-> Open to open the Subjectivity.py file and the LabWeek11classifyopinion.py file. This should start other IDLE windows with the programs in them. **Each example line(s**) can be cut-and-paste to the IDLE window to try it out.

```
import nltk
```

Opinion Classification (using the Movie Review corpus)

In today's lab, we will look at three ways to add features that are sometimes used in various opinion classification problems. We will illustrate the process on the Movie Review corpus, even though not all these features are effective on that problem.

First, we restart by loading the movie reviews and getting the baseline performance of the unigram features. The features of each document will be the words contained in the document, out of a set of words that are frequent in the whole document collection.

```
>>> from nltk.corpus import movie_reviews
>>> import random
```

Movie reviews are given labels by human annotators that are either positive or negative.

```
>>> movie_reviews.categories()
```

The movie review documents are not labeled individually, but are separated into file directories by category. We first create the list of documents where each document is paired with its label.

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

Since the documents are in order by label, we mix them up for later separation into training and test sets.

```
>>> random.shuffle(documents)
```

We look at the first document, which will consist of all the words in the review, followed by the label. Since we did independent shuffles, each person should have a different document.

```
>>> documents[0]
```

We need to define the set of words that will be used for features. This is essentially all the words in the entire document collection, except that we will limit it to the 2000 most frequent words.

```
>>> all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
>>> word_features = all_words.keys()[:2000]
```

Look at the first 100.

```
>>> word_features[:100]
```

Now we can define the features for each document. The feature label will be 'contains(keyword)' for each keyword (aka word) in the word_features set, and the value of the feature will be Boolean, according to whether the word is contained in that document.

```
>>> def document_features(document):
        document_words = set(document)
        features = {}
        for word in word_features:
            features['contains(%s)' % word] = (word in document_words)
        return features
```

Define the feature sets for the documents. We can look at the first one, but remember that it contains 2000 words.

```
>>> featuresets = [(document_features(d), c) for (d,c) in documents]
>>> featuresets[0][:30]
```

We create the training and test sets, train a Naïve Bayes classifier, and look at the accuracy.

```
>>> train_set, test_set = featuresets[100:], featuresets[:100]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)
```

The function show_most_informative_features shows the top ranked features according to the ratio of one label to the other one. For example, if there are 20 times as many positive documents containing this word as negative ones, then the ratio will be reported as 20.00: 1.00 pos:neg.

```
>>> classifier.show_most_informative_features(20)
```

Subjectivity Count features

We will first read in the subjectivity words from the subjectivity lexicon file created by Janyce Wiebe and her group at the University of Pittsburgh. Although these words are often used as features themselves or in conjunction with other information, we

will create two features that involve counting the positive and negative subjectivity words present in each document.

Copy and past the definition of the readSubjectivity function from the Subjectivity.py module.

Create a path variable to where you stored the subjectivity lexicon file. Here is an example from my mac, making sure the path name goes on one line:

```
>>> nancymacpath =
"/Users/njmccrac/Desktop/Data/AAAdocs/research/subjectivitylexicon/hltemnlp05clues/subjclueslen1-HLTEMNLP05.tff"
```

Now run the function that reads the file. It creates a Subjectivity Lexicon that is represented here as a dictionary, where each word is mapped to a list containing the strength, POStag, whether it is stemmed and the polarity. (See more details in the Subjectivity.py file.)

```
>>> SL = readSubjectivity(nancymacpath)
```

Now we create a feature extraction function that has all the word features as before, but also has two features 'positivecount' and 'negativecount'. These features contains counts of all the positive and negative subjectivity words, where each weakly subjective word is counted once and each strongly subjective word is counted twice.

```
>>> def SL_features(document, SL):
        document_words = set(document)
        features = {}
        for word in word_features:
                features['contains(%s)' % 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
```

Now we create feature sets as before, but using this feature extraction function.

```
>>> SL_featuresets = [(SL_features(d, SL), c) for (d,c) in documents]

# features in document 0
>>> SL_featuresets[0][0]['positivecount']
99
>>> SL_featuresets[0][0]['negativecount']
56
>>> SL_featuresets[0][1]
'pos'
>>> train_set, test_set = SL_featuresets[100:], SL_featuresets[:100]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)
0.81
```

So in my random training, test split, these features did not improve the classification on this dataset (but in other opinion classifications, some way to count or use subjectivity or emotion words from LIWC has been effective).

(The LabWeek11 document has an experiment where only subjectivity counts are used as features, but with very poor results.)

Negation features

Negation of opinions is an important part of opinion classification. Here we try a simple strategy. We look for negation words "not", "never" and "no" and negation that appears in contractions of the form "doesn", "'", "t".

For example, my first document has the following words:
if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself',

One strategy with negation words is to negate the word following the negation word, while other strategies negate all words up to the next punctuation or use syntax to find the scope of the negation.

We follow the first strategy here, and we 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 1000 word features and 1000 Not word features set to false. If a negation occurs, add the following words as a Not word feature (if it's in the top 1000 feature words), and otherwise add it as a regular feature word.

```
>>> def NOT_features(document):
        features = {}
        for word in word_features:
            features['contains(%s)' % word] = False
        features['contains(NOT%s)' % 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 == 'not' or word == 'never' or word == 'no'):
            i += 1
            features['contains(NOT%s)' % document[i]]
                        = (document[i] in word_features)
        else:
            if ((i + 3) < len(document))  and
                  (word.endswith('n') and document[i+1] == "'" and document[i+2] == 't'):
                i += 3
                features['contains(NOT%s)' % document[i]]
                        = (document[i] in word_features)
            else:
                features['contains(%s)' % word]
                        = (word in word_features)
        return features
```

Create feature sets as before, using the NOT_features extraction funtion, train the classifier and test the accuracy.

```
>>> NOT_featuresets = [(NOT_features(d), c) for (d,c) in documents]
>>> NOT_featuresets[0][0]['contains(NOTlike)']
>>> NOT_featuresets[0][0]['contains(always)']

>>> train_set, test_set = NOT_featuresets[100:], NOT_featuresets[:100]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)

>>> classifier.show_most_informative_features(20)
```

When I did this experiment, I used all 2000 top word features + 2000 Not features for a total of 4000 features. This took a long time to train the classifier, but I got accuracy of 0.88.

POS features

There are a number of different ways that POS tags are used in features. For example, there are tasks for which using only adjectives is a good thing.

In the LabWeek1 examples is an experiment using words with POS tags instead of plain words. This does not really change the accuracy of the classifier, but it illustrates the use of POS tags.

There is no exercise this week.