NLP Lab Session
Week 10

Sentiment Analysis in NLTK and Python

Getting Started

For this lab session, download the examples and put them in your class folder for copy/pasting examples.

LabWeek10.sentimentfeatures.sents.txt
Subjectivity.py
subjclueslen1-HLTEMNLP05.tff.zip

Unzip the subjclues file and remember the location and start your Python interpreter session.

>>> import nltk

The goal of this week's lab is to show how to conduct a sentiment analysis in the NLTK.

SentiWordNet, a sentiment lexicon in NLTK

Let's look at SentiWordNet, a sentiment lexicon in NLTK. [here shows the NLTK resources http://www.nltk.org/howto/corpus.html] The documentation for SentiWordNet is on this HowTo page: http://www.nltk.org/howto/sentiwordnet.html

In this sentiment lexicon, each word is judged to be made up of partly positive, negative and objective meaning, and 3 scores are given as to how much of each, where the scores must sum to 1.

>>>from nltk.corpus import wordnet as wn
>>> from nltk.corpus import sentiwordnet as swn

Note that there are not very many functions listed, but the trick is that the synsets in SentiWordNet are the same as in WordNet, so we can use the functions to find the synonyms, definitions and examples in wordnet.

>>> list(swn.senti_synsets('breakdown'))
>>> wn.synsets('breakdown')

Following the example in the HowTo page, we look at the third sense of the word breakdown

>>> breakdown3 = swn.senti_synset('breakdown.n.03')

The print function for senti_synsets will give the positive and negative scores.
>>> print (breakdown3)

There are functions to access these two scores separately, and a third function to access the objective scores.

```
>>> breakdown3.pos_score()
>>> breakdown3.neg_score()
>>> breakdown3.obj_score()
```

Now let's look at this term *swn.senti_synset('breakdown.n.03')* closely. In this naming convention *'breakdown.n.03'*, the first item is the word we are interested in, the second item *n* stands for noun, and the third item shows that this synset is the third one in the synset list for *breakdow*n synsets. Below is a list that shows how different word classes are represented in this naming convention.

```
n - NOUN
v - VERB
a – ADJECTIVE
s – ADJECTIVE SATELLITE
r - ADVERB
```

You can find some discussions about the meaning of "Adjective Satellite" (also called "Adjective Cluster"): http://stackoverflow.com/questions/18817396/what-part-of-speech-does-s-stand-for-in-wordnet-synsets.

Recall that you can find here a list of POS tags for the default tagger of nltk.pos_tag(text) (text is a list of tokens)
http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

In Python environment, you can also use the following command to find the tagset.

```
>>>nltk.help.upenn_tagset()
```

Lab Exercise
Use SentiWordNet to get the senti_synset of the sense of a word that you pick. Show the positive, negative and objective sentiment scores for that word, if any.

Sentiment Classification – Words as Features
Now let's look at two ways to add features that are sometimes used in various sentiment or opinion classification problems. We will illustrate the process on a corpus of sentences from the Movie Review corpus, where each sentence has been labeled with the 'positive' or 'negative' tag.

We start by loading the sentence_polarity corpus and creating a list of documents where each document represents a single sentence with the words and its label.
```
>>> from nltk.corpus import sentence_polarity
>>> import random
```

Look at sentences from the entire list of sentences.
```
>>> sentences = sentence_polarity.sents()
>>> len(sentences)
>>> type(sentences)
>>> sentence_polarity.categories()
```

The movie review sentences are not labeled individually, but can be retrieved by category. We first create the list of documents where each document(sentence) is paired with its label.

```
>>> documents = [(sent, cat) for cat in sentence_polarity.categories()
        for sent in sentence_polarity.sents(categories=cat)]
```

In this list, each item is a pair (sent,cat) where sent is a list of words from a movie review sentence and cat is its label, either 'pos' or 'neg'.
```
>>> documents[0]
>>> documents[-1]
```

(Note: negative index in Python means you count from the end of the list, so [-1] is the last element in the list)

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

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

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. Note that we lowercase the words, but do not do stemming or remove stopwords.
```
>>> all_words_list = [word for (sent,cat) in documents for word in sent]
>>> all_words = nltk.FreqDist(all_words_list)
>>> word_items = all_words.most_common(2000)
>>> word_features = [word for (word, freq) in word_items]
```

Now we can define the features for each document, using just the words, sometimes called the BOW or unigram features. 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, word_features):
    document_words = set(document)
    features = { }
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

Define the feature sets for the documents.
```
>>> featuresets = [(document_features(d,word_features), c) for (d,c) in documents]
```

We create the training and test sets, train a Naïve Bayes classifier, and look at the accuracy, and this time we'll do a 90/10 split of our approximately 10,000 documents.

```
>>> train_set, test_set = featuresets[1000:], featuresets[:1000]
>>> 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(30)

Sentiment Classification - Subjectivity Count features

Let's look at another way of adding features for sentiment classification. 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 in the MPQA project. 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 paste the definition of the readSubjectivity function from the Subjectivity.py module. We'll look at the function to see how it reads the file into a dictionary.

Create a path variable to where you stored the subjectivity lexicon file. If you started your python in the same directory, you can just type

>>>SLpath = '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(SLpath)

Now the variable SL (for Subjectivity Lexicon) is a dictionary where you can look up words and find the strength, POS tag, whether it is stemmed and polarity. We can try out some words.
>>> SL['absolute']
>>> SL['shabby']
Or we can use the Python multiple assignment to get the 4 items:
>>> strength, posTag, isStemmed, polarity = SL['absolute']

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. Note that this is only one of the ways in which people count up the presence of positive, negative and neutral words in a document.

>>> def SL_features(document, word_features, 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, word_features, SL), c) for (d,c) in documents]
```

```
# features in document 0
>>> SL_featuresets[0][0]['positivecount']
>>> SL_featuresets[0][0]['negativecount']
>>> SL_featuresets[0][1]
>>>  SL_features[0][0]['contains(bother)']
>>> train_set, test_set = SL_featuresets[1000:], SL_featuresets[:1000]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)
```

In my random training, test split, these particular sentiment features did improve the classification on this dataset. But also note that there are several different ways to represent features for a sentiment lexicon, e.g. instead of counting the sentiment words, we could get one overall score by subtracting the number of negative words from positive words, or other ways to score the sentiment words. Also, note that there are many different sentiment lexicons to try.

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.

Here is one list of negation words, including some adverbs called "approximate negators":
no, not, never, none, rather, hardly, scarcely, rarely, seldom, neither, nor, couldn't, wasn't, didn't, wouldn't, shouldn't, weren't, don't, doesn't, haven't, hasn't, won't, hadn't

The form of some of the words is a verb followed by n't. Now in the Movie Review Corpus itself, the tokenization has these words all split into 3 words, e.g. "couldn", "'", and "t". (and I have a NOT_features definition for this case). But in this sentence_polarity corpus, the tokenization keeps these forms of negation as one word ending in "n't".

```
>>> for sent in list(sentences)[:50]:
...    for word in sent:
...      if (word.endswith("n't")):
...        print(sent)

>>> negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']
```

Start the feature set with all 2000 word features and 2000 Not word features set to false. If a negation occurs, add the following word as a Not word feature (if it's in the top 2000 feature words), and otherwise add it as a regular feature word.

```
>>> def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = False
        features['contains(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['contains(NOT{})'.format(document[i])] = (document[i] in
word_features)
        else:
            features['contains({})'.format(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, word_features, negationwords), c) for (d, c) in documents]
>>> NOT_featuresets[0][0]['contains(NOTlike)']
>>> NOT_featuresets[0][0]['contains(always)']

>>> train_set, test_set = NOT_featuresets[200:], NOT_featuresets[:200]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)

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

In my random split, using the negation features did improve the classification.

Now let's use this classifier to classify the polarity of a new sentence.

```
>>>text = 'I had a good day at the office'
>>>texttokens = nltk.word_tokenize(text)
>>>inputfeatureset = NOT_features(texttokens, word_features, negationwords)
>>>classifier.classify(inputfeatureset)
```

Sentiment Classification - Other features

There are other types of possible features. For example, sometimes people use bigrams in addition to just words/unigrams or use the words together with a POS tagger. Also, there are many other forms of negation features.

For some problems, the word features can be pruned with a stop word list, but care should be taken that the list doesn't remove any negation or useful function words. A very small stop word list is probably better than a large one.