NLP Lab Session
Week 11, April 8, 2010
Classification and Feature Sets 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.

Labweek11classify.py

Installing NLTK Toolkit
Reinstall nltk-2.0b7.win32.msi and Copy and Paste nltk_data from H:\ nltk_data to C:\ nltk_data

Open an IDLE window. Use the File-> Open to open the labweek11classify.py file. This should start another IDLE window with the program in it. **Each example line(s)** can be cut-and-paste to the IDLE window to try it out.

These examples and others appear in Chapter 6 of the NLTK book.

POS Tagging Classifier

Although we have already talked about better ways to do POS tagging, we first use the example of POS tagging in order to show how to build a feature set in the NLTK and to run a classifier.

For each item to be classified, in this case a single word, in NLTK we build the features of that item as a dictionary that maps each feature name to a value, which can be a Boolean, a number or a string. A feature set is the feature dictionary together with the label of the item to be classified, in this case the POS tag.

We start by looking at suffixes of words and building features, which we will name ‘endswith(s)’, where s can be any suffix. The value of the feature will be True or False, depending on whether the word ends with that suffix.

To get a set of suffixes to use for features, we will select the 100 most frequent ones in our corpus, using suffixes of lengths 1, 2 or 3.

```
>>> suffix_fdist = nltk.FreqDist()
>>> for word in brown.words():
        word = word.lower()
        suffix_fdist.inc(word[-1:])
        suffix_fdist.inc(word[-2:])
        suffix_fdist.inc(word[-3:])

>>> common_suffixes = suffix_fdist.keys()[:100]
>>> print common_suffixes
```

Now we write a function that will take a word and create the features for that word.

```
>>>def pos_features(word):
        features = {}
        for suffix in common_suffixes:
                features['endswith(%s)' % suffix]=word.lower().endswith(suffix)
        return features
```

We can test this on some words.

```
>>> pos_features('lovely')
>>> pos_features('expansion')
```

In the NLTK book chapter, they define a decision tree classifier using just these single word features, but we'll go on to define additional features using the word context of the word we're classifying.

Instead of using the suffixes for feature names and giving Boolean values, we now use feature names of ‘suffix(1)’, ‘suffix(2)’, and ‘suffix(3)’ and the values of the these features will be the string that contains the suffix letters of lengths 1, 2, and 3. Furthermore, we add context by having a feature that is the previous word of the sentence.

```
>>> def pos_features(sentence, i):
        features = {"suffix(1)": sentence[i][-1:],
                    "suffix(2)": sentence[i][-2:],
                    "suffix(3)": sentence[i][-3:]}
        if i == 0:
                features["prev-word"] = "<START>"
    else:
                features["prev-word"] = sentence[i-1]
        return features
```

Look at the features of the first 8 words of the first sentence in the Brown corpus:

```
>>> brown.sents()[0]
>>>pos_features(brown.sents()[0], 8)
```

Now we take all the sentences in the news portion of Brown and apply our function to get the POS features, as a dictionary, of each (untagged) word. Then we pair that with the tag to get feature sets.

```
>>> tagged_sents = brown.tagged_sents(categories='news')
>>> featuresets = []
>>> for tagged_sent in tagged_sents:
        untagged_sent = nltk.tag.untag(tagged_sent)
        for i, (word, tag) in enumerate(tagged_sent):
                featuresets.append( (pos_features(untagged_sent, i), tag) )
```

Look at the first 10 words.

```
>>> for f in featuresets[:10]:
        print f
```

Finally we separate our corpus into training and test sets and use these feature sets to train a Naïve Bayes classifier and look at the accuracy.

```
>>> size = int(len(featuresets) * 0.1)
>>> train_set, test_set = featuresets[size:], featuresets[:size]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> nltk.classify.accuracy(classifier, test_set)
```

Text Classification (aka Text Categorization)

For a different type of classification problem, we next look at text classification. In this problem, the items to be classified are documents. Most widely known are datasets that label each document with a topic category (hence the name categorization), but we will look at documents from the NLTK Movie Review corpus, where each document is labeled either 'pos' for positive or 'neg' for negative, according to the opinion of the review.

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.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]
```

(optional)

```
>>> featuresets[0]
```

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)
```

Exercise:

We will assign each group/person a number of words to use for features, instead of the 2000 used here. Each group will re-run the movie review text classification problem and post their accuracy to the discussion list. If we increase the number of words by some increment, we should get an idea of how the accuracy increases according to the number of words used.