CE 314/887 Using NLTK for text classification

1: Text classification with NLTK

To do this, we're going to start by trying to use the movie reviews database that is part of the NLTK corpus. From there we'll try to use words as "features" which are a part of either a positive or negative movie review. The NLTK corpus movie_reviews data set has the reviews, and they are labelled already as positive or negative. This means we can train and test with this data.

First, let's explore our data.

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

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

random.shuffle(documents)

print(documents[1])

all_words = []
for w in movie_reviews.words():
    all_words.append(w.lower())

all_words = nltk.FreqDist(all_words)
print(all_words.most_common(15))
print(all_words["stupid"])
```

After importing the data set we want, you see:

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

The above code is translated to: In each category (we have pos or neg), take all of the file IDs (each review has its own ID), then store the word_tokenized version (a list of words) for the file ID, followed by the positive or negative label in one big list.

Next, we use random package to shuffle our documents. This is because we're going to be training and testing. If we left them in order, chances are we'd train on all of the negatives, some positives, and then test only against positives. We don't want that, so we shuffle the data.

Then, just so you can see the data you are working with, we print out documents[1], which is a big list, where the first element is a list the words, and the 2nd element is the "pos" or "neg" label.

Next, we want to collect all words that we find, so we can have a massive list of typical words. From here, we can perform a frequency distribution, to then find out the most common words. As you will see, the most popular "words" are actually things like punctuation, "the," "a" and so on, but quickly we get to legitimate words. We intend to store a few thousand of the most popular words, so this shouldn't be a problem.

```
print(all_words.most_common(15))
```

The above gives you the 15 most common words. You can also find out how many occurences a word has by doing:

```
print(all_words["stupid"])
```

Next up, we'll begin storing our words as features of either positive or negative movie reviews.

2： Converting word to features with NLTK

In this part, we're going to be building off the previous video and compiling feature lists of words from positive reviews and words from the negative reviews to hopefully see trends in specific types of words in positive or negative reviews.

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

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

random.shuffle(documents)

all_words = []

for w in movie_reviews.words():
    all_words.append(w.lower())

all_words = nltk.FreqDist(all_words)

word_features = list(all_words.keys())[:3000]
```

Mostly the same as before, only with now a new variable, word_features, which contains the top 3,000 most common words. Next, we're going to build a quick function that will find these top 3,000 words in our positive and negative documents, marking their presence as either positive or negative:

```
def find_features(document):
    words = set(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)

    return features
```

Next, we can print one feature set like:

```
print((find_features(movie_reviews.words('neg/cv000_29416.txt')))
)
```

Then we can do this for all of our documents, saving the feature existence booleans and their respective positive or negative categories by doing:

```
featuresets = [(find_features(rev), category) for (rev, category)
in documents]
```

Awesome, now that we have our features and labels, what is next? Typically the next step is to go ahead and train an algorithm.

3: Naive Bayes Classifier with NLTK

Now it is time to choose an algorithm, separate our data into training and testing sets, and press go! The algorithm that we're going to use first is the Naive Bayes classifier. This is a pretty popular algorithm used in text classification, so it is only fitting that we try it out first. Before we can train and test our algorithm, however, we need to go ahead and split up the data into a training set and a testing set.

You could train and test on the same dataset, but this would present you with some serious bias issues, so you should never train and test against the exact same data. To do this, since we've shuffled our data set, we'll assign the first 1,900 shuffled reviews, consisting of both positive and negative reviews, as the training set. Then, we can test against the last 100 to see how accurate we are.

This is called supervised machine learning, because we're showing the machine data, and telling it "hey, this data is positive," or "this data is negative." Then, after that training is done, we show the machine some new data and ask the computer, based on what we taught the computer before, what the computer thinks the category of the new data is.

We can split the data with:

```
# set that we'll train our classifier with
training_set = featuresets[:1900]

# set that we'll test against.
testing_set = featuresets[1900:]
```

Next, we can define, and train our classifier like:

```
classifier = nltk.NaiveBayesClassifier.train(training_set)
```

First we just simply are invoking the Naive Bayes classifier, then we go ahead and use .train() to train it all in one line.

Easy enough, now it is trained. Next, we can test it:

```
print("Classifier accuracy
percent:",(nltk.classify.accuracy(classifier, testing_set))*100)
```

Boom, you have your answer. In case you missed it, the reason why we can "test" the data is because we still have the correct answers. So, in testing, we show the computer the data without giving it the correct answer. If it guesses correctly what we know the answer to be, then the computer got it right. Given the shuffling that we've done, you and me might come up with varying accuracy, but you should see something from 60-75% in accuracy on average.

Next, we can take it a step further to see what the most valuable words are when it comes to positive or negative reviews:

```
classifier.show_most_informative_features(15)
```

This is going to vary again for each person, but you should see something like:

Most Informative Features:

```
insulting = True neg : pos = 10.6 : 1.0
ludicrous = True neg : pos = 10.1 : 1.0
winslet = True pos : neg = 9.0 : 1.0
detract = True pos : neg = 8.4 : 1.0
breathtaking = True pos : neg = 8.1 : 1.0
silverstone = True neg : pos = 7.6 : 1.0
excruciatingly = True neg : pos = 7.6 : 1.0
warns = True pos : neg = 7.0 : 1.0
tracy = True pos : neg = 7.0 : 1.0
insipid = True neg : pos = 7.0 : 1.0
freddie = True neg : pos = 7.0 : 1.0
damon = True pos : neg = 5.9 : 1.0
debate = True pos : neg = 5.9 : 1.0
ordered = True pos : neg = 5.8 : 1.0
lang = True pos : neg = 5.7 : 1.0
```

What this tells you is the ratio of occurrences in negative to positive, or visa versa, for every word. So here, we can see that the term "insulting" appears 10.6 more times as often in negative reviews as it does in positive reviews. Ludicrous, 10.1.