NLP Lab Session
Week 13, April 17, 2013
Classification for the Final Projects

Getting Started

In this lab, we will be doing some work in the Python IDLE window and also running some stand-alone python programs. All of the programs and data are zipped together in one zip file on Blackboard:

LabExamplesWeek13FinalProject.zip

Download this zip file to your NLP class folder in the lab and unzip it there.

Creating data and using Weka for classification

In the NLTK, we have been using the Naïve Bayes classification algorithm, which has reasonable performance on many classification problems. However, on text classification problems, it is also often the case that the Support Vector Machines (SVM) algorithms have the best classification performance. For this, we can use the Weka software package, which is an open software application that collects together many data mining algorithms, including classification algorithms. Weka is particularly useful for classification experiments, for example, to compare different types of features (and less useful for creating a classifier for unlabeled data).

To prepare data for Weka, we will use the NLTK to create feature sets as usual, but then I have written a function that will write the feature sets to a file that can be used for classification testing in Weka. One file format that Weka can use is a csv (comma separated values) file where the first line should contain the names of the features (which Weka calls attributes) separated by commas and the remaining lines have one line per training example, with the feature values separated by commas. Note that you should not allow any feature name or value containing quotes, apostrophes, or commas. Also, Weka treats the class label (for example, 'pos' or 'neg') as one of the features, and it is customarily the last one.

In order to demonstrate this function, let's load the movie reviews in NLTK.

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

>>> 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 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]
```

We give the same feature definitions as before for all words, except that we make sure that no word of length 1 is used. This eliminates the punctuation, including quotes, apostrophes and commas.

```
>>> def document_features(document):
        document_words = set(document)
        features = {}
        for word in word_features:
          if len(word) > 1:
              features['contains(%s)' % word] = (word in document_words)
        return features
```

Define the feature sets for the documents.

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

Here is a function definition that takes featuresets and a path to where you want to write the csv file, and converts the featuresets to be a weka input file in csv format.

```
>>> def wekawrite(featuresets, outpath):
    # take featuresets defined in the nltk and convert them to weka input csv file
    #    and write the file to the outpath location
    # open outpath for writing - it should include the name of the csv file
    f = open(outpath, 'w')
    # get the feature names from the feature dictionary in the first featureset
    featurenames = featuresets[0][0].keys()
    # create the first line of the file as comma separated feature names
    #    with the word class as the last feature name
    featurenameline = ''
    for featurename in featurenames:
      featurenameline += featurename + ', '
    featurenameline += 'class'
    # write this as the first line in the csv file
    f.write(featurenameline)
    f.write('\n')
    # convert each feature set to a line in the file
    # with comma separated feature values, for booleans just write the words true and false
    for featureset in featuresets:
      featureline = ''
```

```
        for key in featurenames:
            featureline += str(featureset[0][key]) + ', '
        featureline += featureset[1]
        # write each feature set values to the file
        f.write(featureline)
        f.write('\n')
    f.close()
```

Now we need to make a path to the output file (which will be weka input). Include the name of the file and the file extension .csv. (On my Mac, it's
outpath =
"/Users/njmccrac1/AAAdocs/NLPspring2013/labs/LabExamplesWeek13FinalProject/weka/movies.csv")

>>> outpath = <put your path here for the lab>

Now we can call the function:

>>> wekawrite(featuresets, outpath)

And we can go to the file, open it either in Excel or in NotePad++ to see the contents.

Now we can classify in Weka; go to the Weka document for instructions.

Classifying Twitter Sentiment

There have been quite a few research papers in recent years that discuss various aspects of finding Twitter Sentiment. I have chosen to follow one by Koulompis, Wilson and Johanna Moore "Twitter Sentiment Analysis: The Good the bad and the OMG!", in ICWSM (International Conference on Weblogs and Social Media) in 2011. Note that Theresa Wilson is one of the researchers from the Subjectivity Lexicon project. This paper is available on-line and also in the Content -> Final Project Resources -> Twitter Sentiment Classification folder on Blackboard, along with two other papers on Twitter Sentiment.

For training data, we are going to use the Sentiment 140 Emoticon data set, available at http://help.sentiment140.com/, where you can click on the link "For Academics" to read about the data and their process. The data set was collected by searching Twitter for positive and negative emoticons. The emoticons were removed from the tweets, and each tweet was labeled positive (4) or negative (0) according to its emoticon. The data set has 1,600,000 tweets. This dataset is good for experiments, but note that in a practical situation where we want to detect sentiment in tweets, we would want to have training data for three polarities: positive, negative and neutral. Also, since this training set has emoticons removed, they can't be used for features.

In order to randomly select a smaller training set, I wrote a program called sentiment__classify_select_training_data.py that selects 8,000 tweets of each polarity and

saves them in a .csv file called training.16000.mixed.csv. I have included the program for your information and use if you wish to have larger or smaller training sets. (We can look at some examples in this file, either using Excel or NotePad.) Note that to run the program, you should use easy_install to install the python package csv.

First, if we don't already have easy_install, we should install it. Next we open a terminal (command prompt) window and type

% easy_install csv

This installs the python csv package in its folder of external packages.

Next, I have written a baseline sentiment classifier program that uses NLTK to train a classifier using word features and subjectivity features from the training set. The baseline classifier uses a twitter tokenizer from Tweet Motif and does some pruning to remove non-ascii characters, tokens of length 1 and tokens on a tweet stopword list. In order to run this program, open a terminal window (or command prompt window) and navigate to the folder where you put all the programs from this week's lab examples. Now run the classify train model program:

$ python sentiment__classify_train_model.py
(or whatever command you need to use on a PC)

This will take a little time to make the feature sets and train the model.

Now to work on this classification task for the final project, you can extend this program in several ways:

- Work on preprocessing to improve the tokens
- Work on features to add useful ones
- Experiment with different classifiers in weka

For ideas on preprocessing and features, we can look at a summary of what the Wilson et al paper did. Note that the baseline program already uses the three subjectivity lists as they did, but that you could experiment with using the LIWC or ANEW lists instead of, or in addition to, the subjectivity lexicon.

Here are the main points for successful preprocessing and features from the Wilson paper:

Preprocessing:

- Tokenize emoticons, hashtags, mentions and URLs and replace them by their token type (HAPPY, SAD, URL, etc.)
- Expand abbreviations to their meanings, brb -> "be right back"
- Normalize character repetitions to two, "happyyyyy" -> "happyy"
- Note intensifiers such as all caps, "I LOVE this show"

Features:

- Vocabulary: removed stopwords, replaced words that either directly preceded or directly succeeded a negation word with the word with NOT, “happy” -> “NOT_happy” and keep only the top occurring 1000 words
- Subjectivity Lexicon: three features denoting the presence or absence of positive, neutral or negative emotion words.
- (Part-of-speech features were tried, but didn’t add to the accuracy.)
- Micro-blogging features: emoticons, presence of intensifiers (using the Internet Lingo Dictionary and other internet slang dictionaries).

There are other ideas in the other two Twitter Sentiment papers that I placed in the Blackboard folder.

Note that if you want to try using positive and negative sentiment words from the LIWC lexicon, I have given a python program that will read them in sentiment__read_pos_neg_words.py in the zip file.