CMPS 143 - Assignment 3

Due Monday, January 28 at 11:59 PM

1 Overview

In part 3 of the previous assignment we discussed using language models to categorize text into two distinct categories: `positive` opinion and `negative` opinion. You then began constructing a language model for each category of text. In this assignment you will use the language models you built in the previous assignment applying them to a text classification task.

1.1 Text Classification

Text classification requires assigning a category from a predefined set to each document of interest. For this assignment, you will build off of the previous assignment to model the reviews as a feature vector. You will then use these feature vector representations to make predictions about unlabeled (unknown rating) reviews. To do this, you will need to build and train a classifier.

1.2 Data

The assignment operates on a corpus restaurant reviews that have been split into `training`, `development`, and `testing` files for you. The training set consists of 1200 reviews, 600 that have been given a positive rating (an overall score = 5 out of 5) and 600 that have been given a negative rating (an overall score <= 3). The development set consists of 140 positive and 140 negative reviews. Both the training and the development set come with heir correct classification category label (positive or negative). The test set is also provided: the idea is that the test set is *unseen* during training, and you will be testing how well your trained model predict the correct classification for the test set. There are 400 additional reviews (200 positive and 200 negative) in the test set that we will use to evaluate your models.

How the dataset was made. The dataset was built from a list of user reviews where each review has the following format:

```
Review: INTEGER
FACET_NAME_1 = VALUE_1
FACET_NAME_2 = VALUE_2
...
FACET_NAME_N = VALUE_N
```

The *FACET_NAME* is a string. The *VALUE* could be an integer, string or a label from a closed set (e.g., {*Yes*, *No*}. Not all reviews have all facets. However, we are only interested in the facets *Overall*, which specifies whether the review is positive or negative, and *Text*, which gives the textual review. Each review is separated by a period character (.). We are providing some stub code that will process the reviews for you to extract the text and the *Overall* score. The *Overall* score is the basis of the dependent variable whose value you will be predicting (pos or neg) with your trained classifier.

2 Feature Vector Representation of Text

For each review, create a feature vector representation of the text. Each element of this vector represents a key/value pair. The *key* is the *name* of the feature. The name of the feature can be anything you'd like, but it should give an indication of what property you are extracting from the text. The *value* can be any real number. Binary features are a common special case of features when the values are restricted to the set {0,1}. These are useful for indicating whether a feature is present or not in the document. For example, we could model the presence or absence of the word *throw* with a feature named *UNI_throw* and a value of 1 to indicate we've seen it and a 0 to indicate we have not. We name the feature *UNI_throw* with the *UNI* prefix to indicate that its provenance is our unigram features, and *throw* to indicate what unigram it is counting. In some cases, we can also use the value to give an estimate of the strength or frequency of the feature. For example we might want to take into account that a word was seen more than once in a document, and assign feature values that increase as the frequency of the feature increases.

In NLTK, feature vectors are represented using the built-in *dict* class, which is Python's implementation of a hash table.

In this part of the assignment, **create three different feature sets of the text**:

1. word_features: unigram and bigram word features
2. word_pos_features: unigram and bigram word features and unigram and bigram part-of-speech features
3. word_pos_liwc_features: unigram and bigram word features and unigram and bigram part-of-speech features and the liwc category features.

You should use binning to assign value to your features. As we'll see in class, binning is a method for dealing with data which is skewed by a small number of features having extremely high frequencies. Specifically we want to transform our numerical features so that we combine and collapse counts that roughly represent frequencies. Imagine the word *the* occurred 10 times, *food* occurs 100 times, and the bigram *the food* occurs 1 times. This is the binning strategy you should use where count is the number of occurrences for an arbitrary feature.

```
return count if count < 2 else 3
```

The value returned by this binning function should then be joined into a string, which is a binary feature. To reiterate, this is called a binary feature because either the string is going to be present in the feature vector, or it's going to be absent from the feature vector. The binary features which will be added to your feature vector in the previous example would be the following strings:

```
UNI_the:3
UNI_food:3
BIGRAM_the_food:1
```

For each feature set and dataset, **write the labels and feature vectors to a file named** `FEATURE_SET-DATASET-features.txt`**, where each line represents a document.** FEATURE_SET should be one of the following: `word_features, word_pos_features, word_pos_liwc_features`. DATASET should be one of the following: `training, development, testing`. The first token on the line should be the class label (`positive` or `negative` for the restaurant reviews). Each additional token should be a feature **name:value** pair separated by whitespace.

The feature name and value should be concatenated by a colon (`:`). For example, your file should look something like this (ignore the values mentioned here) for two reviews, when you create your feature set for `word_features`:

```
positive UNI_taste:3 UNI_good:2 ...  BIGRAM_taste_good:1
negative UNI_taste:3 UNI_bad:3 ...  BIGRAM_taste_bad:1
```

For `word_pos_features`, it would look like this:

```
positive UNI_taste:3 UNI_good:2 ...  BIGRAM_taste_good:1 ...  UNI_POS_NN:3 ...  BI_POS_NN_POS:2
negative UNI_taste:3 UNI_bad:3 ...  BIGRAM_taste_bad:1 ...  UNI_POS_JJ:3 ...  BI_POS_DT_NN:2
```

For the feature file created out of the training data, there should be 1200 such lines (600 positive and 600 negative), one for each review.

2.1 LIWC

In class, you learned about LIWC [1] categories. For this assignment we will use LIWC categories as another feature. We have included a Python module that will take a tokenized input string and return a `Counter` [2] object with various counts and features of the input text. The stub code provided with this assignment uses only two categories of LIWC as features. You need to extend the features to include all of the LIWC categories and apply a binning strategy to determine their values. You can see how they perform individually, and in combination with the other features.

To use the LIWC module:

1. Place `word_category_counter.py` into the same directory as your Python script for this assignment.
2. Create a subfolder in the directory where your script is located and name it `data`.
3. Place the two `dic` files in this `data` directory.

More information about LIWC can be found in the manual [3].

3 Baseline Sentiment Classification

In this part of the assignment we will use the feature vectors to train a Naive Bayes classifier to automatically label unseen reviews as positive or negative.

1. Write a program that trains a Naive Bayes model using the features you extracted in the previous part of the assignment by:
   (a) Creating the training instances from the features you created previously. This is a list of tuples. The first element of the tuple is the feature vector of the document (i.e., the dictionary of feature/value pairs). The second element is the name of the category (i.e., `positive` or `negative`). There should be one feature set for each off `word_features`, `word_pos_features`, and `word_pos_liwc_features`.
   (b) Train three Naive Bayes classifiers with the training instances created in the previous step: `classifier = nltk.classify.NaiveBayesClassifier.train(train)`
   (c) Use the `show_most_informative_features()` method to find the most informative features in each case and write the results to a file named `FEATURE_SET-DATASET-informative-features.txt`.
2. Write a program `restaurant-competition-P1.py` that classifies reviews using your trained models. It should take two arguments:
   (a) The file with the reviews in it to be classified.
   (b) The second should be the name of a file to write predictions to. When saving predictions, each predicted label should be on a separate line in the output file, in the same order as the input file. This file should be the output of a function called `evaluate`. The `evaluate` function should also calculate the accuracy and confusion matrix if it is supplied with the example labels.
3. Run your classification program on the development data using both of the classifiers.
4. Make a document called `all-results.txt` and include the accuracy and confusion matrix of the word_features classifier and `word_pos_features` classifier on the development data.

[1] `http://www.liwc.net/`
[2] `https://docs.python.org/2/library/collections.html`
[3] `http://www.liwc.net/LIWC2007LanguageManual.pdf`

4 Sentiment Classification Competition

Train and test your classifier from the previous section using different combinations of features, similar to the feature selection that was illustrated in class. Make the best classifier you can using only the features described above.

We will provide you with a set of test samples which are unlabeled. You'll need to use your best classifier to make your prediction on this file and write that to a file called `restaurant-competition-model-P1-predictions.txt`.

5 What to Turn In

Turn in a single zipped file including all the files below.

1. In the document `all-results.txt` that you made above in Section 3 part 4, describe any feature combinations and additional features that you tried beyond the word_features, word_pos_features. Also summarize the different accuracies that you were able to get for each different feature set. Describe the feature set that resulted in the best model and include the accuracy of your best classifier on the development data. Turn in this file as a part of your solution.
2. The predictions of your best classifier on the test data `restaurant-competition-model-P1-predictions.txt`.
3. The output of the `evaluate` function on the development data for word_features, word_pos_features, word_pos_liwc_features and the best classifier (as in Section 3, part 2 (c)). Call these files `output-ngrams.txt`, `output-pos.txt`, `output-liwc.txt`, and `output-best.txt`.
4. Feature files from Section 2: `FEATURE_SET-DATASET-features.txt`.
5. Your program `restaurant-competition-P1.py`.
6. Informative features (`FEATURE_SET-DATASET-informative-features.txt`) from Section 3.