CSE 143 - Assignment 3

Due Thursday, October 24 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

You will be given a total of 500 movie reviews: 250 that have been given a positive rating (an overall score > 5 out of 10) and 250 that have been given a negative rating (an overall score < 5). These reviews have been split into `training`, `development`, and `testing` files for you. We have held out 100 additional reviews that we will use to evaluate your models.

Each dataset contains a list of reviews. Each review has two facets, *Overall*, which specifies whether the review is positive or negative, based on the score from 1-10 and *Text*, which gives the textual review. Each review is separated by a period character (`.`).

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.

In some cases, we can also use the value to give an estimate on the strength of the feature. For example we might want to take into account that a word was seen more than once in a document. Although any value is possible, most text classifiers work best when the values are between ± 1 or between 0 and 1. To ensure this we will normalize our values to be within this range.

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 four different feature sets of the text**:

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

4. word_pos_opinion_features: unigram, bigram and trigram word features and unigram, bigram and trigram part-of-speech features and the opinion lexicon binary features.

The value of the feature should be the **relative frequency of that feature within the document, except for LIWC and Opinion Lexicon features**. For example, if the word *the* occurred 10 times and there were 200 words overall in the document, then the feature value for `UNI_the` should be $\frac{10}{200} = 0.05$. If the bigram feature `BIGRAM_the_food` was seen once out of 60 bigrams in the document, the value should be $\frac{1}{60} = 0.0167$.

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`, `word_pos_opinion_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 imdb 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:0.01 UNI_good:0.01 ...  BIGRAM_taste_good:0.001
negative UNI_taste:0.01 UNI_bad:0.01 ...  BIGRAM_taste_bad:0.001
```

For `word_pos_features`, it would look like this:

```
positive UNI_taste:0.01 UNI_good:0.01 ...  BIGRAM_taste_good:0.001 ...  'UNI_POS_NN': 0.466 ...
'BI_POS_NN_POS': 0.071
negative UNI_taste:0.01 UNI_bad:0.01 ...  BIGRAM_taste_bad:0.001 ...  UNI_POS_JJ': 0.333 ...
'BI_POS_DT_NN': 0.041
```

For the feature file created out of the training data, there should be 300 such lines, one for each review.

2.1 LIWC and Opinion Lexicon

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 extend the features to include additional LIWC categories. You must add 5 categories of your choosing. 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].

The Opinion Lexicon is a set of positive words and a set of negative words [4]. For the features using this lexicon each word in both the positive and negative list of words will be a binary feature. So if the word appears in the review it will have a value of 1 and if it does not it will have a value of 0. (For the challenge you are not restricted to using the Opinion Lexicon as a binary feature).

[1] `http://www.liwc.net/`
[2] `https://docs.python.org/2/library/collections.html`
[3] `http://www.liwc.net/LIWC2007LanguageManual.pdf`
[4] https://www.kaggle.com/nltkdata/opinion-lexicon

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 of `word_features`, `word_pos_features`, `word_pos_liwc_features`, and `word_pos_opinion_features`.
   (b) Train four Naive Bayes classifiers with the training instances created in the previous step: `classifier = nltk.classify.NaiveBayesClassifier.train(train)`
   (c) Save each classifier to disk using the following code snippet.

   ```
   import pickle
   f = open('classifier.pickle', 'wb')
   pickle.dump(classifier, f)
   f.close()
   ```

   (d) 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 `imdb-competition-P1.py` that classifies reviews using your trained models. It should take three arguments:
   (a) The first should be the trained classifier model (one of the pickled models).
   (b) The second should be the file with the reviews in it to be classified.
   (c) The third should be the output file to write the classification results. Each predicted label should be on a separate line in the output in the same order as the input file. This file should be the output of a function called `evaluate` and you should turn it in. The `evaluate` function should also calculate the accuracy, probability and confusion matrix.

   You can read in your trained classifier using the following code snippet.

   ```
   import pickle
   f = open('classifier.pickle', 'r')
   classifier = pickle.load(f)
   f.close()
   ```

3. Run your classification program on the development data and testing 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 both the development data and the testing data.

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. You will turn in the best classifier as a pickle file. We will then be able to test your classifier on imdb reviews that we have held out and see who can obtain the best results.

Save your best classifier model to a file named `imdb-competition-model-P1.pickle`.

OPTIONAL. You may also explore alternatives to using the relative frequency of features. You can try binning the feature counts as we discussed in class.

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, word_pos_liwc_features and word_pos_opinion_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 both the development data and the testing data. Turn in this file as a part of your solution.

2. The pickle files for word_features, word_pos_features, word_pos_liwc_features, word_pos_opinion_features and your best classifier:

   - `imdb-word_features-model-P1.pickle`
   - `imdb-word_pos_features-model-P1.pickle`
   - `imdb-word_pos_liwc_features-model-P1.pickle`
   - `imdb-word_pos_opinion_features-model-P1.pickle`
   - `imdb-competition-model-P1.pickle`

3. The output of the `evaluate` function on the test data for word_features, word_pos_features, word_pos_liwc_features, word_pos_opinion_features and the best classifier (as in Section 3, part 2 (c)). Call these files `output-ngrams.txt`, `output-pos.txt`, `output-liwc.txt`, `output-opinion.txt`, and `output-best.txt`.

4. Feature files from Section 2: `FEATURE_SET-DATASET-features.txt`.

5. Your program `imdb-competition-P1.py`.

6. Informative features (`FEATURE_SET-DATASET-informative-features.txt`) from Section 3.