CS57300: Homework 2

Due date: Thursday February 16, midnight (submit via turnin)

Bag of Words Naive Bayes

In this programming assignment you will implement a naive Bayes classification (NBC) algorithm and use it on a sample of the reviews from the Yelp data set—to predict characteristics of the reviews based on the words in the reviews.

There is a sample dataset (`yelp_data.csv`) to use for this assignment. The csv file contains information for 2,000 reviews in tab-separated format; the columns consists of `reviewID, classLabel, reviewText`.

You will implement algorithms in Python to learn (i.e., estimate) and apply (i.e., predict) the NBC to textual data. Note that although there are many data mining algorithms available online, for the assignment you must design and implement your own versions of the algorithm. **Do not use any publicly available code–your code will be checked against public implementations.** Also, there will not test cases provided—you are required to design your own tests to ensure that your code runs correctly and meets the specifications below.

Instructions below describe the details of how to submit your code and assignment using turnin on data.cs.purdue.edu. **Alternate submissions (i.e., outside the turnin system) will not be accepted.**

Algorithm Details

The NBC uses a bag-of-words representation to model documents as unordered collections of words. For a given dictionary of size p (i.e., possible words), the bag-of-words representation for a document D with class label C consists of a binary vector $\mathbf{X}$ of size p, with a random variable X_i for each word in the dictionary.

Features: You will first need to construct features for $\mathbf{X}$ based on the words that occur in the reviews. Preprocess the data to parse the reviews from the "`reviewText`" column with the following process,

1. Covert to lowercase (e.g., Purdue $\rightarrow$ purdue)
2. Strip all punctuation (e.g., don't $\rightarrow$ dont)
3. Split based on white space to determine "words" (e.g., to construct $\rightarrow$ {to, construct})

Determine the set of unique words and count the number of times each word occurs in the set of reviews. Sort the words in descending order by frequency. Discard the top 100 words with highest frequency (these will correspond roughly to "stop" words). Then construct features for the next 500 most frequent words (i.e., 101-600), for each review. Specifically construct a binary word feature X_i for each selected word—with $x_i = 1$ indicating that the word X_i appears in the review D, and $x_i = 0$ otherwise.

Class label: You will consider the binary class label Y in the "`classLabel`" column. This records whether the review was positive or not (i.e., $Y = 1$ implies rating is positive, and $Y = 0$ implies rating is negative).

Smoothing: Implement Laplace smoothing in the parameter estimation. For an attribute X_i with k values, Laplace correction adds 1 to the numerator and k to the denominator of the maximum likelihood estimate for $P(X_i = x_i|Y)$.

Evaluation: When you apply the learned NBC to predict the class labels of the examples the test set, use zero-one loss to evaluate the predictions.

Code specification

Your python script should take two arguments as input.

1. *trainingDataFilename*: corresponds to a subset of the Yelp data (in the same format as `yelp_data.csv`) that should be used as the *training set* in your algorithm.
2. *testDataFilename*: corresponds to another subset of the Yelp data (again in the same format) that should be used as the *test set* in your algorithm.

Your code should do the following:

- Read in the training/test sets from the csv files.
- Calculate and print out the top ten word feature in the training set (i.e., words 101-110).
- Learn a NBC model from the training set, apply the learned model to the test set.
- Apply the learned model to the test set.
- Evaluate the predictions with zero-one loss and print out the results.

Name your file nbc.py. Then the input and output should look like this:

```
$ python nbc.py train-set.dat test-set.dat
WORD1 love
WORD2 best
...
WORD9 place
WORD10 service
ZERO-ONE-LOSS 0.3106
```

Programming assignment

You should implement your solution using Python. You may **not** use anybody else's code; you **must** implement your own version! You should submit your source code files along with your typed HW report. The TAs should be able to compile and run your code.

1. Write code to transform a given input file to a bag-of-words representation. (5 pts)
   (a) Compute binary features for the 101-600 most frequently occurring words as described above. Note that the frequencies should be computed (and selected) from the training set, not the entire data.

(b) Output the top ten selected words with highest frequency (i.e., words 101-110). (Make sure you count every word once in each review (even though it appear multiple times) since we are using bag of words representation.)

(c) Construct the bag of words features each review in both the training and the test set, based on the words selected from the training set.

2. Implement a Naive Bayes Algorithm. (20 pts)

   (a) Write code to read in training data and learn the NBC model.

   (b) Write code to read in test data to apply the learned NBC and evaluate the resulting predictions with zero-one loss.

3. Learn and apply the algorithm (15 pts)

   (a) For each % in [1, 5, 10, 20, 50, 90]:
      - Randomly sample % of the data to use for training.
      - Use the remaining (1-%) of the data for testing.
      - Learn a model from the training data and apply it to test data.
      - Measure the performance on the test data using zero-one loss.
      - Repeat ten times with different random samples (using the same %).
      - Record avg. and st. dev. of zero-one loss across the ten trials.

   (b) Plot a learning curve for the results (training set size vs. zero-one loss). Compare to the baseline *default* error that would be achieved if you just predicted the most frequent class label. Discuss the results.

4. Explore effect of feature space (10 pts)

   (a) For each W in [10, 50, 250, 500, 1000, 4000]:
      - Drop the 100 most frequent words and construct features for the next W most frequent words (i.e., you will vary how many features are used in the model). Randomly sample 50% of the data to use for training.
      - Use the remaining (50%) of the data for testing.
      - Learn a model from the training data and apply it to test data.
      - Measure the performance on the test data using zero-one loss.
      - Repeat ten times with different random samples (using the same %).
      - Record avg. and st. dev. of zero-one loss across the ten trials.

   (b) Plot a learning curves for the results (feature size vs. zero-one loss). Again compare to the baseline *default* error that would be achieved if you just predicted the most frequent class label. Discuss the results.

Submission Instructions:

After logging into data.cs.purdue.edu, please follow these steps to submit your assignment:

1. Make a directory named *'yourName_yourSurname'* and copy all of your files there.

2. While in the upper level directory (if the files are in /homes/neville/jennifer_neville, go to /homes/neville), execute the following command:

   turnin -c cs57300 -p HW2 your_folder_name

   (e.g. your prof would use: turnin -c cs57300 -p HW2 jennifer_neville to submit her work) Keep in mind that old submissions are overwritten with new ones whenever you execute this command.

   You can verify the contents of your submission by executing the following command:

   turnin -v -c cs57300 -p HW2

   Do not forget the -v flag here, as otherwise your submission would be replaced with an empty one.

Your submission should include the following files:

1. The source code in python.

2. Your evaluation & analysis in .pdf format. Note that your analysis should include learning curve graphs as well as a discussion of results.

3. A README file containing your name, instructions to run your code and anything you would like us to know about your program (like errors, special conditions, etc).