CS57300: Homework 3

Due date: Monday March 9, 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 are two sample datasets (`funny_data.csv` and `stars_data.csv`) to use for this assignment. Both csv files use the same format and each contains information for 5,000 reviews. The only difference between the two datasets are the target class labels (see description below). For each dataset, we randomly selected 2,500 reviews with each class label value—to produce a 50/50 binary classification problem. (Note: there may be duplicate reviews between the two datasets.)

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 detail how to use turnin to submit your code and assignment on data.cs.purdue.edu. For HW2 we accepted some submissions via email/blackboard for people who had technical difficulties using turnin—but we expect you to work out the issues in time to submit HW3. **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 "`text`" column, covert to lowercase, strip punctuation, and determine the set of unique words. Count the number of times each word occurs in the set of reviews. Sort the words in descending order by frequency. Discard the top 200 words with highest frequency (these will correspond roughly to "stop" words). Then construct features for the next 2,000 most frequent words (i.e., 201-2200), 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 two different binary class labels Y (for the data in `funny_data.csv` and `stars_data.csv` respectively) based on the "`funny`" and "`stars`" columns—*isFunny* (funny>0 vs. funny$=0$), *isPositive* (stars$=5$ vs. stars$=1$).

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 four arguments as input.

1. *trainingDataFilename*: corresponds to a subset of the Yelp data (in the same format as `funny_data.csv` and `stars_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.

3. *classLabelIndex*: an column index to specify the classification task (funny= 5 and stars= 7).

4. *printTopWords*: a flag to indicate whether to print out the top word features; a value of 1 indicates that you should print out the top ten words from the reviews used in the model (i.e., words 201-210).

Your code should read in the training/test sets from the csv file, learn a NBC model from the training set for the appropriate classification task, apply the learned model to the test set, and evaluate the predictions with zero-one loss.

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

```
$ python nbc.py train-set.dat test-set.dat 5 1
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. (10 pts)

   (a) Compute binary features for the 201-2,200 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) If indicated by the command line argument `printTopWords`, output the top ten selected words with highest frequency (i.e., words 201-210).

   (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.

   (d) If the command line argument `classLabelIndex` is 5 (i.e., classification task is *isFunny*), you will also need to construct a binary class label from the "`funny`" column to differentiate values of 0 from those with value > 0.

2. Implement a Naive Bayes Algorithm. (25 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)

   For each classification task (*isFunny*, *isPositive*), using the relevant dataset for the task:

   (a) For each % in [10,50,90]:
   - 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 learning curves for each class label (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.

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 HW3 your_folder_name

   (e.g. your prof would use: turnin -c cs57300 -p HW3 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 HW3

   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).