Homework 3: Sentiment Analysis [starter code]

Your goal for this homework is to perform **Sentiment Analysis**: classifying movie reviews as positive or negative. Recall from lecture that sentiment analysis can be used to extract people's opinions about all sorts of things (congressional debates, presidential speeches, reviews, blogs) and at many levels of granularity (the sentence, the paragraph, the entire document). Our goal in this task is to look at an entire movie review and classify it as positive or negative.

ALGORITHM

You will be using **Naïve Bayes** and Laplace smoothing. Your classifier will use words as features, add the logprob scores for each token, and make a binary decision between positive and negative. You will also implement the **binarized version** of the Naive Bayes classifier with boolean features. In addition to this, you will explore the effects of **stop-word filtering**. This means removing common words like "the", "a" and "it" from your train and test sets. We have provided a stop list with the starter code in the file:

```
data/english.stop
```

Finally, you will implement your own features and/or heuristics to improve performance (more on this below).
The algorithm is a simplified version of this paper from the required readings:

- **Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan.** 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 79-86.

ASSIGNMENT

Train a **Naïve Bayes** classifier on the **imdb1** data set provided with the starter code. This is the actual Internet Movie Database review data used in the original Pang and Lee paper. The starter code comes already set up for 10-fold cross-validation training and testing on this data. Recall that cross-validation involves dividing the data into

several sections (10 in this case), then training and testing the classifier repeatedly, with a different section as the held-out test set each time. Your final accuracy is the average of the 10 runs. When using a movie review for training, you use the fact that it is positive or negative (the hand-labeled "true class") to help compute the correct statistics. But when the same review is used for testing, you only use this label to compute your accuracy. The data comes with the cross-validation sections; they are defined in the file:

`data/poldata.README.2.0`

Task 1- Your first task is to implement the classifier training and testing code and evaluate them using the cross-validation mechanism.

Task 2- Next, evaluate your model again with the stop words removed. Does this approach affect average accuracy (for the current given data set)?

Task 3- Next, you'll implement a binarized version of the Naive Bayes classifier that uses feature-presence/absence instead of feature counts.

The last task is more open-ended. By this point,

- You'll have used only the (unigram) words in each review as features in your model. Can you **add other features** that can boost your classifier's performance?
- You'll want to try **feature selection**, which you'll recall from the reading means removing some possibly-redundant features. The stop-word-removal that you used in part 1 is a simple version of feature selection, but there are more sophisticated ones from the reading. See if there are other simple feature selection techniques that can be used here, to give better cross-validation scores.
- You might want to come up with clever ways of **weighting features**. For example, in the regular Naive Bayes model, you weighed each word (equivalently, feature) according to the number of times it occurred in the review. In the binarized version, however, you weighed the features differently, based only on presence/absence. Perhaps there are more clever things you can do.

These are some common techniques used for improving the performance of Naive Bayes classifiers. Here are some examples where they could potentially be useful-

- *I had heard this movie was very good, but I found it bad.*: There are two strong and opposing words in this sentence, 'good' (reflecting a positive review) and 'bad' (reflecting a negative review), but it can be inferred from the sentence that 'bad' determines its sentiment and not 'good'. Can you weigh your features for 'good' and 'bad' differently to reflect this?
- *Page's acting was neither inspired nor realistic.*: Your current feature set comprises individual words in a bag-of-words approach (since the information about word order is lost). Hence, 'inspired' and 'realistic', considered separately from their surrounding words in the sentence, would likely bias your classifier into labeling this sentence positive. Can you find a fix for this?
- *The locales are chosen meticulously; no wonder this comes across as a very beautiful film.*: You may want to tinker with the weight for the feature 'beautiful', since it is expressed more strongly than just saying it's a beautiful film. Is there something you can do?

Task 4- Experiment with different features and heuristics (and combinations of these) to boost your classifier's cross-validation score, and add to your code your best-performing system. Besides pointers from the aforementioned examples and your own insights from looking at the data (this is something you should get used to doing!), the suggested paper (Pang, Lee and Vaithyanathan, 2002) should prove immensely helpful. Based on your cross-validation scores on the imdb1 dataset and your score on the held-out test set (see Evaluation for more), the top-performing submissions would be eligible for extra credit.

A note on generalization- to get overall gains, it's important to come up with reasonably simple features that have some generality, as opposed to ones that fix just a few specific examples. In other words, if you try to boost your cross-validation score by using features that cater very strongly to examples in the training data, your cross-validation score may be high, but test data numbers may be quite bad. Be mindful of this, especially for task 4, since your classifier will be evaluated on both the training data and a held-out/unseen test set.

WHERE TO MAKE YOUR CHANGES

To ensure that your code works properly not only when run from the command line but also when executed by our autograder, you should limit your changes to addExample() and classify() methods within NaiveBayes.py. You're free to add other elements further invoked from addExample() or classify() (you would need to add some data structures for your work -- where to add them will be mentioned in the starter code), but note well that main() is **NOT** executed by the autograder so you cannot rely on anything added there.

Changes beyond addDocument() and classify(), if you choose to make any, should be done with caution. The autograder calls the following methods: crossValidationSplits(), trainSplit(), train(), test(). It also relies on the definitions of classes TrainSplit and Example and directly manipulates the boolean FILTER_STOP_WORDS. The role of the BOOLEAN_NB and BEST_MODEL booleans is the same as mentioned. If the BOOLEAN_NB switch is True, then your code should build a model based on the binarized version of the Naive Bayes classifier. Similarly, your code should build a model with all your added features and heuristics enabled if the flag BEST_MODEL is set. Note that the BOOLEAN_NB or FILTER_STOP_WORDS flags will not be set with the BEST_MODEL flag on, so if you wish to use a binarized model for your best model or include stopword filtering in your best model, write the code for this appropriately.

EVALUATION

Your classifier will be evaluated on two different data sets—both **imdb1** mentioned above (with cross validation) and a second held-out test set—and both with and without stop-word filtering, binarized Naive Bayes and your best model.

RUNNING THE CODE

In your terminal, execute

```
$ python NaiveBayes.py data/imdb1
```

Note that this allows you to specify other data sets in the process of your development, but for submission, our script **MUST**be able to find the imdb1 directory in its default location noted here.

Adding a flag (*-f*, *-b* or *-m*)...

```
$ python NaiveBayes.py –f data/imdb1
```

Flag -f involes stop word filtering

```
$ python NaiveBayes.py –b data/imdb1
```

Flag -b involes your binarized model implementation, and

```
$ python NaiveBayes.py –m data/imdb1
```

-m invokes your best model.

This will train all of the language models and output their performance.

Attention: Please use these flags (-f, -b and -m) one at at time. Using two or three together will lead to only one flag actually taking effect. This is done because this is how the assignment will be evaluated - stopword removal in isolation from binarization and best model, and so on.

If you're curious how your classifier peforms on other data sets, you can specify a second directory, in which case the program should train on the entirety of the first set (i.e., without cross-validation) then classify the entire held-out second set.

```
$ python NaiveBayes.py (–f or –b or –m) <train directory>
<test directory>
```

You can also classify one file by providing a file instead of a directory as the test argument.

SUBMITTING YOUR SOLUTION

Submit your assignment via **Canvas.** We expect the following files in your final submission:

- NaiveBayes.py
- Splitter.py

You will only be able to see your score sn the dev set. The test set scores will be released after the deadline.