**CS 585 Homework:**

**Naive Bayes Text Classification**

### Instructions

The goal of this homework is for you to execute what you have learned in the class and implement the naive Bayes algorithm. Depending on the efficiency of your implementation the experiments required to complete the assignment may take some time to run, so it is a good idea to start now. If you have any questions, the best way is to ask on Piazza.

**Note: You are to write all code specified in this assignment by yourself. Any copying or other form of code reuse is plagiarism and will be treated accordingly.**

In this assignment you will implement and evaluate the naive Bayes algorithm for text classification. You will train your models on a (provided) dataset of positive and negative movie reviews and report prediction accuracy on a test set. We provide you with starter Python code to help read in the data and evaluate the results of your model’s predictions. Please finish by yourself and turn in the following in Blackboard:

* Your (commented) code for NaiveBayes.py
* A brief writeup that includes the metrics and evaluation described below

Hand in both files in a gzipped tar file with the name *<CWID>*-HW1.tar.gz, where *<CWID>* is your CWID. Make sure your name and CWID are at the top of your writeup as well.

### The Code

The provided code in imdb.py reads the data into a document-term matrix using scipy’s csr matrix format (see [http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.sparse.csr\_matrix.html - scipy.sparse.csr\_matrix](http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix)). You need to work with log probabilities instead of multiplying them directly to avoid the possibility of floating point underflow (see: <https://en.wikipedia.org/wiki/List_of_logarithmic_identities>).

You can run the sample code like so:

python NaiveBayes.py data/aclImdb 1.0

python NaiveBayes.py ../../data/aclImdb 1.0

### Classification and Evaluation (40 Points)

The first two methods you will need to implement are NaiveBayes.Train and NaiveBayes.PredictLabel. Before you do this, the classifier in the starter code always predicts +1 (positive). Once you have implemented these methods, the code will print out accuracy. Try running with different values of the smoothing hyperparameter ([ALPHA](https://en.wikipedia.org/wiki/Additive_smoothing)) (suggested values to try: 0.1, 0.5, 1.0, 5.0. 10.0), and record the evaluation results for your report.

### Probability Prediction (20 Points)

Then you will need to implement two methods NaiveBayes.LogSum and NaiveBayes.PredictProb. You would need to work with log probabilities and use the log-sum-exp trick to prevent potential numerical underflows (see [this on StackOverflow](http://stats.stackexchange.com/questions/105602/example-of-how-the-log-sum-%20exp-trick-works-in-naive-bayes)). Record the probability estimated for the first 10 reviews in the test data for your report.

### Precision and Recall (20 Points)

Now, use the PredictProb method to produce precision/recall curves for the data, by adjusting the probability threshold for determining whether a review is classified as positive or negative. First, implement methods EvalPrecisiosn and EvalRecall that compute the precision and recall for a given class (positive or negative). Then change PredictLabel to take a parameter probThresh such that it predicts positive only if the probability of positive is greater than probThresh. Graph precision vs. recall for the positive and negative classes by varying the threshold. What relationship do you see?

### Features (20 points)

Print out the 20 most positive and 20 most negative words in the vocabulary sorted by their weight according to your model This will require a bit of thought how to do because

1. the words in each document have been converted to IDs (see Vocab.py) so you will need to convert them back, and
2. you will need to compute the linear feature weight for each word based on the condition probabilities in the model.

The output should look something like so:

word1\_pos weight1 word2\_pos weight2 word3\_pos weight3

...

word1\_neg weightk word2\_neg weightk+1 word3\_neg weightk+2

...

Where wordn pos and wordn neg are the top 20 positive and negative words. (Hint: you might find the [numpy.argsort method](http://docs.scipy.org/doc/numpy/reference/generated/numpy.%20argsort.html)  useful). Please include this output in your report.

Give your best explanation of why the particular positive and negative words you see appear.