# CSE 5525 Homework 1: Text Classification

Alan Ritter

In this assignment you will implement naïve bayes and perceptron algorithms for sentiment classification. You will train your models on a (provided) dataset of 25,000 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. You are *strongly* encouraged to make use of the provided code. If you really prefer to implement everything from scratch for some reason, please talk to the instructor first. Your submitted code should run on the command line in a unix-like environment (e.g. Linux, OSX, Cygwin).

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 early.

## Naïve Bayes (3 points)

First, implement a naïve bayes classifier. 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` for details). I would recommend working 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
```

The two methods you will need to implement are `NaiveBayes.Train` and `NaiveBayes.Predict`. Before you do this, the classifier 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`) (suggested values to try: 0.1, 0.5, 1.0, 10.0), and record the results for your report.

## Perceptron (3 points)

Next, implement the perceptron classification algorithm (analagous starter code is provided in `Perceptron.py`). The only hyperparameter is the number of iterations. Run the classifier and report results on the test set with various numbers of iterations (for exaple: 1, 10, 50, 100, 1000).

## Parameter Averaging (2 points)

Modify the perceptron code to implement parameter averaging. Instead of using parameters from the final iteration, $w_n$, to classify test examples, use the average of the parameters from every iteration, $\sum_{i=1}^{N} w_i$. A nice trick for doing this efficiently is described in section 2.1.1 of Hal Daume's thesis (`http://www.umiacs.umd.edu/~hal/docs/daume06thesis.pdf`).

## Features (2 points)

Print out the 20 most positive and 20 most negative words in the volcabulary sorted by their weight according to your model This will require a bit of thought how to do because the words in each document have been converted to IDs (see `Vocab.py`). 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 useful - `http://docs.`

`scipy.org/doc/numpy/reference/generated/numpy.argsort.html`). Please include this output in your report.

## What to Turn In

Please turn in the following to the dropbox on Carmen:

1. Your code
2. A brief writeup that includes the numbers / evaluation requested above.