CS 165B – Machine Learning, Spring 2017

Assignment #5
Due Friday, June 9 by 4:30pm

Notes:

- *This assignment is to be done individually. You may discuss the problems at a general level with others in the class (e.g., about the concepts underlying the problem, or what lecture or reading material may be relevant), but the work you turn in must be solely your own.*
- Be sure to re-read the "Policy on Academic Integrity" on the course syllabus.
- Be aware of the late policy in the course syllabus – i.e., *late submissions will not be accepted*, so turn in what you have by the due time.
- Any updates or corrections will be posted on the Assignments page (of the course web site), so check there occasionally.
- Your code must be well commented – graders should not have to work to figure out the purpose or method of any section or line or code.
- Turning in the assignment:
  - There is no hardcopy to turn in for this homework.
  - Turn in your source code and output file (as described in the problems) via GauchoSpace.

Problem #1 [100 points]

In this problem you'll create a binary classifier using boosting, based on a variation of the simple linear classifier (similar to what you implemented in HW2, problem 6, but with two classes instead of three). You may use and modify code from the posted solutions or from your own solution to HW2, problem 6.

Write a Python program called **`boostit.py`** that implements boosting for binary classification, using the following modification of the basic linear classifier as the model. Instead of computing each class exemplar by taking the mean of the class training points, use the weighted mean. I.e., instead of

class_exemplar = $\frac{1}{k}\sum_{i=1}^{k} x_i$ (where k is the number of training points in the class)

use

class_exemplar = $\frac{1}{\sum w_i}\sum_{i=1}^{k} w_i x_i$ (where w_i is the weight associate with point x_i)

As described in the boosting algorithm, these weights w_i are initially evenly distributed among all of the training points (each weight is initialized to 1/|D|, where |D| is the number of training points), and then are updated at each iteration of the boosting algorithm. As before, the discrimination surface of each model is halfway between the two class exemplars, perpendicular to the vector connecting the exemplars. Once the boosting algorithm is run on the training points, classify the test points based on the weighted average of the T models $M_t(x)$. For classifying test points, use $M(x) > 0$ to predict the positive class (and thus $M(x) \leq 0$ to predict the negative class).

The syntax of your program should be:

```
% python boostit.py T train_pos train_neg test_pos test_neg
```

where **`T`** is the ensemble size (the number of models $M(x)$ to produce) and **`train_pos`** and **`train_neg`** are filenames containing the training data for positive and negative instances, respectively. (Similarly, **`test_pos`** and **`test_neg`** comprise the testing data set.) **`T`** can be any positive integer.

You can assume that the two classes are not linearly separable, so you will never get an error of zero for an iteration over the training data. (Otherwise, you'll get a divide-by-zero error!)

As output, your program should print out information about the boosting iterations and about the classification of the test data, as in this example:

```
% python boostit.py 5 train_pos train_neg test_pos test_neg
Iteration 1:
Error = 0.15
Alpha = 0.8673
Factor to increase weights = 3.3333
Factor to decrease weights = 0.5882
Iteration 2:
Error = 0.22
Alpha = 0.6328
Factor to increase weights = 2.2727
Factor to decrease weights = 0.6410
(...etc... for all iterations of boosting)

Testing:
False positives: 10
False negatives: 7
Error rate: 9%
```

The training and testing data sets have two files each (for positive and negative instances), and all have the same format, each comprising M N-dimensional points:

```
M N
P11 P12 … P1N
P21 P22 … P2N
…
PM1 PM2 … PMN
```

where M, N are integers specifying the number of points in the file and the dimensionality of the points, respectively (in non-homogeneous coordinates), and Pij is real-valued, describing the j^{th} component of point i. The values of N must be the same in all files.

Run the program on the provided data sets with T=5, and save the output from these examples in different text files (named **`output1.txt`** and **`output2.txt`**, etc.) to be turned in with your code.

Although not part of the assignment, you are encouraged to compare this boosting classifier with the original linear classifier and also to compare results using different values of T. You are also encouraged to visualize the data and results (training data points, linear classifiers trained, and training data points and their classification). These are all optional, however.