# CS6140 Machine Learning

# HW4 Boosting and Bagging

Make sure you check the syllabus for the due date. Please use the notations adopted in class, even if the problem is stated in the book using a different notation.

---

## PROBLEM 1 Adaboost code [50 points]

Implement the boosting algorithm as described in class. Note that the specification of boosting provides a clean interface between boosting (the meta-learner) and the underlying weak learning algorithm: in each round, boosting provides a weighted data set to the weak learner, and the weak learner provides a predictor in return. You may choose to keep this clean interface (which would allow you to run boosting over most any weak learner) or you may choose to more simply incorporate the weak learning algorithm inside your boosting code.

- **Decision Stumps**

Each predictor will correspond to a *decision stump*, which is just a feature-threshold pair. Note that for each feature $f_i$, you may have many possible thresholds which we shall denote $t_{i,j}$.

Given an input instance to classify, a decision stump corresponding to feature $f_i$ and threshold $t_{i,j}$ will predict +1 if the input instance has a feature $f_i$ value exceeding the threshold $t_{i,j}$; otherwise, it predicts -1.

To create the various thresholds for each feature $f_i$, you should

- sort the traiing examples by their $f_i$ values
- remove duplicate values, and
- construct thesholds that are midway between successive feature values.

You should also add two thresholds for each feature: one below all values for that feature and one above all values for that feature. Note that by removing duplicate values, you will have fewer thresholds than examples for any given feature, and possible far fewer.

- **Weak Learning via "Optimal" Decision Stumps**

Create a weak learner that returns the "best" decision stump with respect to the weighted training set given. Here, the "best" decision stump $h$ is the one whose error is as far from 1/2 as possible; in other words, your goal is to maximize

$$|1/2 - error(h)|.$$

Thus, a decision stump whose error is 0.9 will be favored over one whose error rate is 0.2. Why? Because boosting will *negatively* weight the former decision stump, effectively flipping its predictions and turning it

into a predictor with error rate 0.1.

You should think carefully about how to efficiently search for such a decision stump so that your code runs in a reasonable amount of time.

- **Weak Learning via "Randomly Chosen" Decision Stumps**

Create a weak learner that returns a "random" decision stump, independent of the weighted training set given.

Note that you would almost certainly *never* do this in practice, but the point of this exercise is to demonstrate that boosting can leverage *any* weak predictor given, even one chosen at random.

- **Boosting with Decision Stumps**

Run your Adaboost code on the Spambase dataset (10 folds Cross validation). You can work with the preconditioned or non- preconditioned data; it should make little difference when boosting via decision stumps. (Consider why this is so...)

- **"Optimal" Decision Stumps:** Run your implementation of boosting with "optimal" decision stumps on the training data. After each round, you should compute (1) the local "round" error for the decision stump returned, (2) the current training error for the weighted linear combination predictor at this round, (3) the current testing error for the weighted linear combination predictor at this round, and (4) the current test AUC for the weighted linear combination predictor at this round.
    - Create three plots: One for the local "round" error (which should go up as rounds increase), one for the training and test error (which should both go down as rounds increase), and one for the test AUC (which should go up as the rounds increase). You should boost until you see "convergence" in test error or AUC.
    - For the final weighted linear combination that is produced, create an ROC curve on the test data and compare your results to those you obtained in previous assignments.

  You should think carefully about how you can efficiently generate the required results above. For example, I would suggest keeping a running weighted linear prediction value (before thresholding at zero) for each training and testing instance: when each new round predictor is created, you can simply update your running weighted linear prediction value and then easily compute training and testing error rates (by thresholding these values at zero), as well as testing AUCs (by ranking the instances by these values).

- **"Randomly Chosen" Decision Stumps:** Repeat the procedure above for "randomly chosen" decision stumps. Note that you will almost certaily have to boost for more rounds to "converge".

## PROBLEM 2 [50 points] Adaboost on UCI datasets

UCI datasets: AGR, BAL, BAND, CAR, CMC, CRX, MONK, NUR, TIC, VOTE. (These are archives which I downloaded a while ago. For more details and more datasets visit http://archive.ics.uci.edu/ml/). The relevant files in each folder are only two:
* .config : # of datapoints, number of discrete attributes, # of continuous (numeric) attributes. For the discrete ones, the possible values are provided, in order, one line for each attribute. The next line in the config file is the number of classes and their labels.
* .data: following the .config convention the datapoints are listed, last column are the class labels.

You should write a parser that given the .config file, reads the data from the .data file.

**A.** Run the Adaboost code on the UCI data and report the results. The datasets CRX, VOTE are required.

**B.** Run the algorithm for each of the required datasets using c% of the datapoints chosen randomly for training, for several c values: 5, 10, 15, 20, 30, 50, 80. Test on a fixed fold (not used for training). For statistical significance, you can repeat the experiment with different randomly selected data or you can use cross-validation.

**C(extra credit)** Run boosting on the other UCI datasets. Some of them are multiclass, so you will have to have a multiclass-boosting implementation. The easiest "multiclass" is to run binary boosting one-vs-the-others separately for each class.

## PROBLEM 3 [50 points] Active Learning

Run your code from PB1 on Spambase dataset to perform Active Learning. Specifically:

- start with a training set of about 5% of the data (selected randomly)

- iterate M episodes: train the Adaboost for T rounds; from the datapoints not in the training set, select the 2% ones that are closest to the separation surface (boosting score F(x) closest to ) and add these to the training set (with labels). Repeat until the size of the training set reaches 50% of the data.

How is the performance improving with the training set increase? Compare the performance of the Adaboost algorithm on the c% randomly selected training set with c% actively-built training set for several values of c : 5, 10, 15, 20, 30, 50. Perhaps you can obtain results like these

## PROBLEM 4 [70 points] Error Correcting Output Codes

Run Boosting with ECOC functions on the 20Newsgroup dataset with extracted features. The zip file is called 8newsgroup.zip because the 20 labels have been grouped into 8 classes to make the problem easier. The features are unigram counts, preselected by us to keep only the relevant ones.

ECOC are a better muticlass approach than one-vs-the-rest. Each ECOC function partition the multiclass dataset into two labels; then Boosting runs binary. Having K ECOC functions means having K binary boosting models. On prediction, each of the K models predicts 0/1 and so the prediction is a "codeword" of length K 11000110101... from which the actual class have to be identified.

You can use the following setup for 20newsgroup data set.

- Use the exhaustive codes with 127 ECOC functions as described in the ECOC paper, or randomly select 20 functions.

- Use all the given features

- For each ECOC function, train an AdaBoost with decision stumps for 200 or more iterations

The above procedure takes a few minutes (Cheng's optimized code, running on a Haswell i5 laptop) and gives us at least 70% accuracy on the test set.

## PROBLEM 5 [20 points]

What is the VC dimension for the class of hypothesis of:

a) unions of two rectangles

b) circles

c) triangles

d) multidimensional "sphere" given by f(x) = sign [(x-c)(x-c) -b] in the Euclidean space with m dimensions $\mathbb{R}^m$. Justify your answers !

## PROBLEM 6 [50 points] Bagging

**Bagging setup**:

- Training: Run your Decision Tree classifier separately (from scratch) T=50 times. Each Decision Tree is trained on a sample-with-replacement set from the training dataset (every Decision Tree has its own sampled-training-set). You can limit the depth of the tree in order to simplify computation.
- Sampling with replacement: Say there are N datapoints in the training set. Sampling with replacement, uniformly, for a sample of size N, works in the following way: in a sequence, independently of each other, select randomly and uniformly N times from the training datapoints. Once a datapoint is selected, it is still available for further sampling (hence "with replacement" methodology). Each sampled-training-set will have N datapoints; some points will be sampled overall more than once (duplicates) while other datapoints will not be sampled at all.
- Testing: for a test datapoint, will run all T Decision Trees and average the predictions to obtain the final prediction.

**Run bagging on Spambase dataset**. Compare results with boosting.

## PROBLEM 7 [Extra Credit]

Run Boosting with ECOC functions on the Letter Recognition Data Set (also a multiclass dataset).

## PROBLEM 8 [Extra Credit] Boosted Decision Trees

Do PB1 with weak learners being full decision trees instead of stumps. The final classifier is referred as "boosted trees". Compare the results. Hints: there are two possible ways of doing this.

- Option 1. The weights are taken into account when we decide the best split, like in Adaboost. This requires you to change the decision tree training : when looking for best split at each node, the split criteria has to account for current datapoints weights as assigned by the boosting.

- Option 2. We can simulate the weights by sampling. In each round, when we want to train the decision tree, we construct a new data set based on the old one. We sample from the old data set k times with replacement. In each sampling, each data point x_i in the old data set has probability w_i of being chosen into the new data set. k can be as large as the size old data set, or smaller. We only need to make sure there are enough data points sampled for a decision tree to be trained reliably. Then we train a decision tree on the new data set without considering weights. Weights are already considered in sampling. In this way, you don't need to modify the decision tree training algorithm. More generally, any weak learner, whether it can handle weights naturally or not, can be trained like this. Once the decision tree is trained, the new data set is discarded. The only use of the newly constructed data set is in building the tree. Any other computation is based on the original data set.

## PROBLEM 9 [Extra Credit] Rankboost

- Implement rankboost algorithm following the rankboost paper and run it on TREC queries.

## PROBLEM 10 [Extra Credit] Gradient Boosted Trees

Run gradient boosting with regression stumps/trees on 20Newsgroup dataset dataset.