Assignment 4: Boosting Algorithm

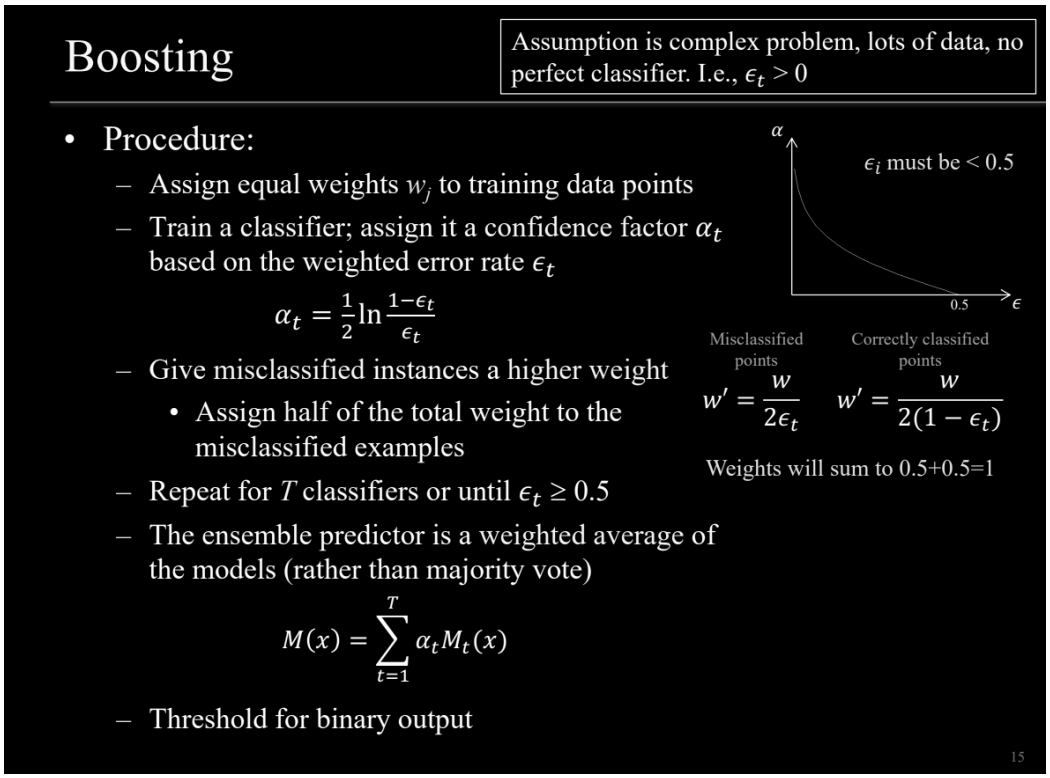
Cashton Holbert
CodaLab User: CashtonH

Part 1:

In this assignment I implemented a boosting meta algorithm on a basic linear classifier. Given training labels and training data, of which there were two classes, I made predictions on the data utilizing a linear classifier, and then ran the boosting algorithm that placed meta-weights on the data between new instances of model, halting after num_iterations or until the algorithm converged and correctly classified the training data.

How Does it work?

Boosting is a meta-algorithm that runs over the top of another classification algorithm. In this case, linear classification was utilized. The algorithm applies a weight to each data point and runs the classification model. If the model has not converged it changes the meta-weights of each data point according to whether or not it was misclassified. The math and general procedure can be seen below.



Procedure of the code?

The file that runs the code is boostit.py. It includes a class called BoostingClassifier that includes functions that run the linear classification model, perform boosting on the data set, and make predictions on the test set. Most of the code runs in fit(self, X, y), which takes in some training data X and some training labels y.

In fit I split the data into their respectives neg and pos classes according to their labels and then do the same with the meta-weights Wj according to the procedure above.

I then start the boosting algorithm, which in the case of this code loops over 5 iterations, I then perform the linear classification model. I then apply the weights to each respective data point. After this, I make classifications on the training data depending on their euclidean distance to each of the two classes' centroids. I go on to receive an error rate in which I use to calculate the confidence factor as outlined in the procedure above. From there I follow the procedure increasing or decreasing each weight depending on its classification all depending on if I haven't already jumped out of the loop if the model converged. If the model hasn't converged I continue this process until num_iterations or it does. After boosting training is completed I calculate the ensemble predictor as above.

The last function in the class is predict(self, X), which makes predictions of the respective classifiers on some data X and the model from fit.

Inputs Include:

- Dataset1
  - train.npy
    - Includes the labels and data points for training set
  - test.npy
    - Includes the labels and data points for testing set

Outputs include:

- Error rate on each iteration
- Alpha value on each iteration
- Factor to increase and decrease weights on each iteration
- Number of false positives
- Number of false negatives
- Error Rate on test set
- Accuracy on test set
- Precision on test set
- Recall on test set

Part 2: Code performance locally

```
PS C:\Users\casht\142 Hw\asgn4> python3 local_evaluation.py
Iteration 1:
Error = 0.09999999999999999
Alpha = 1.0986
Factor to increase weights = 5.0
Factor to decrease weights = 0.5556

Iteration 2:
Error = 0.5
Alpha = 0.0
Factor to increase weights = 1.0
Factor to decrease weights = 1.0

Testing:
False positives:  4
False negatives:  1
Error Rate:  6.25 %
Accuracy: 0.9375, F-measure: 0.9397590361445783, Precision: 0.9069767441860465, Recall: 0.975, Final Score: 93.86295180722891
```

Part 3: Code Performance on Basic Linear Model

```
PS C:\Users\casht\142 Hw\asgn4> python3 local_evaluation.py
Iteration 1:
Error = 0.09999999999999999
Alpha = 1.0986
Factor to increase weights = 5.0
Factor to decrease weights = 0.5556

Testing:
False positives:  4
False negatives:  1
Error Rate:  6.25 %
Accuracy: 0.9375, F-measure: 0.9397590361445783, Precision: 0.9069767441860465, Recall: 0.975, Final Score: 93.86295180722891
```

This output is the result of running the data through a linear classification model, the likes I implemented in assignment 2. With 1 iteration of the loop and meta-weights practically nullified as they are not updated, this is the output. It seems to be very similar to the boosting algorithm, which is likely because my model was already so accurate on the first iteration. With noisier and/or larger data, it is very likely that the algorithm would not perform as well as the linear classification with boosting model.