Extra Credit: Adaboost **Chao Shi**
CS7646, Summer 2016 GTID#: 902694979

1 Algorithm

I followed the algorithm outlined in Harris Drucker(1997) [1], a modified version of *Adaboost.R* (Freund&Schapire,1996 [2]). I summarize the algorithm here below.

Initially, we assign a uniform weight $w_i = 1,\ i = 1, ..., N_1$ to each data point. Then repeat the following:
1. Calculate the probability of each data point being sampled: $p_i = w_i / \sum w_i$. Then sample the original data set *with replacement* according to the probability distribution.
2. Construct a regression machine(here I use KNN) t from the training set sampled in step 1. Each machine makes a prediction: $h_t : x \rightarrow y$
3. Use this machine to predict on every x in the *original* training set, obtain $y_i^p(x_i), i = 1, ...N_1$.
4. Calculate a loss for each prediction $L_i = L(|y_i^{(p)} - y_i|)$. I choose the loss function L as

$$L_i = \frac{|y_i^{(p)}(x_i) - y_i|}{D} \tag{1}$$

$$D = \sup |y_i^{(p)}(x_i) - y_i|,\ i = 1, ..., N_1 \tag{2}$$

5. Calculating an average loss: $\bar{L} = \sum_{i=1}^{N_1} L_i p_i$
6. Calculate confidence of this predictor: $\beta = \bar{L}/(1 - \bar{L})$
7. Update the weights: $w_i \rightarrow w_i * \beta^{1-L_i}$. Here we see that the larger the loss, the more weight to put on this point. Then in next round this point will have a better chance to be chosen and get larger update, in order to reduce the overall RMSE.
8. Query. Step 1 to 7 finished training. For query on a particular x, we adopt an weight median approach. Each of the T machines makes prediction $h_t(x), t = 1, ..., T$. Then final prediction from the whole ensemble h_f is:

$$h_f = \inf \left\{ y \in Y : \sum_{t:\ h_t \le y} \log \frac{1}{\beta_t} \ \ge\ \frac{1}{2} \sum_t \log \frac{1}{\beta_t} \right\} \tag{3}$$

2 Implementation

I implement this algorithm in my `BagLearner.py` , as an extension of the Bagging method, since there is already a default `boost` parameter in the function parameter list. The source code file is identical with the one I submitted to Project 2. Here I paste the core part to illustrate my implementation.

```
class BagLearner(object):
        def __init__(...):
        ...
        pass

    def addEvidence(self, dataX, dataY):
        ...
        if boost:
                self.betas = []
            indices = np.array(range(self.dataX.shape[0]))
            # initialize weight
            W = np.ones(self.dataX.shape[0])  # all weights
            prob = W/self.dataX.shape[0]      # probability

            # Create bins and sample according to
            # the probability distribution
            bins = np.cumsum(prob)
            sample = indices[np.digitize\
                (np.random.random(self.dataX.shape[0]), bins)]
            learner = self.Learner(**self.kwargs)
            learner.addEvidence(dataX[sample],dataY[sample])
            predY = learner.query(dataX)
            L = np.abs(predY - dataY)/\
                (np.abs(predY - dataY)).max() # loss function
            Lmean = (L*prob).sum()
            beta = Lmean/(1 - Lmean)
            self.betas.append(beta)
            self.learners.append(learner)
            W = W*np.power(beta, 1 - L)
            Maxloop= 10
            loop = 0

            #This is the most time-consuming part
            while Lmean < 0.5 and loop < Maxloop:
                ...
                #repeat the code segment before
                #this loop and after the previous if
            ...

        def query(self, points):
        .....# omit boost = False part
        if self.boost:
            predYs = np.zeros((self.betas.shape[0], points.shape[0]))
```

```
Y = np.zeros(points.shape[0])

for i, learner in enumerate(self.learners):
    predYs[i] = learner.query(points)
bsum = 0.5*np.sum(np.log(1./self.betas))

# The for loop below find out the weighted
# median. Pretty time consuming part, since
# I need to sort each column individually
for j in range(points.shape[0]):
    relabel = predYs[:,j].argsort()
    beta_cumsum = np.cumsum(np.log\
            (1./self.betas[relabel]))
    Y[j] = predYs[:,j][relabel]\
            [np.where(beta_cumsum >= bsum)[0][0]]
return Y
```

3 Experiment

3.1 How to use

The implementation above has the same API as `BagLearner` without boosting. Simply assigning `True` to the default `boost` keyword argument will trigger Boosting instead of Bagging, which means in this case, the `bags` parameter is ignored, because it won't be used in boosting.

3.2 Result

I run boosting against the same 10 test cases as tested against Bagging with KNN. 8 of them gave more or less the same result as Bagging, but significantly slower because of the while loop and query.
2 cases failed because the prediction of the initial machine is exactly correct, zero RMSE, resulting in zero value in the denominator when calculating the loss function. See equation (1) and (2).

3.3 Discussion

Now I have a general picture about the implementation and theory of Boosting Algorithm. But I still have many questions. Such as:

1. In several papers/notes online stated that the iteration of adding new learner has a natural exit condition, which is $\bar{L} \geq 0.5$. I think it means no matter how you sample the original training set, you can not get average loss below 0.5. I don't understand because as the iteration going on, shouldn't the loss function be decreasing since we're improving the performance of the learner? Then we shouldn't expect the loop end with acceptable number of iterations, which is actually the case for my implementation. That's why I added a `Maxloop` variable to cut off the loop at some point. I think there is certainly problem with my implementation, but I haven't figured out why yet. I think in order to understand these, I need some background knowledge about Learning theory, such as VC class and dimension, and converging criterion of a machine learning algorithm, and etc. I haven't got time to dig into those yet, but I definitely will in the future.

2. On a more detailed level, for those 2 failed test case, how should I avoid or work around these cases? All the loss function I found need the difference of two values at denominator.

3. Do you have any suggestions on which way I should proceed at this point? I have collected many notes, textbooks and MOOC courses about Machine Learning and data science, too many actually. Not quite sure where to start.

References

[1] Harris Drucker. Improving regressors using boosting techniques. In *ICML*, volume 97, pages 107–115, 1997.

[2] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. *ICML*, 96:148–156, 1996.