Machine Learning and Computational Statistics, Spring 2015
Homework 5: Trees and Ensemble Methods

Qingxian Lai(ql516)

Due: Wednesday, March 25, 2015, at 4pm (Submit via NYU Classes)

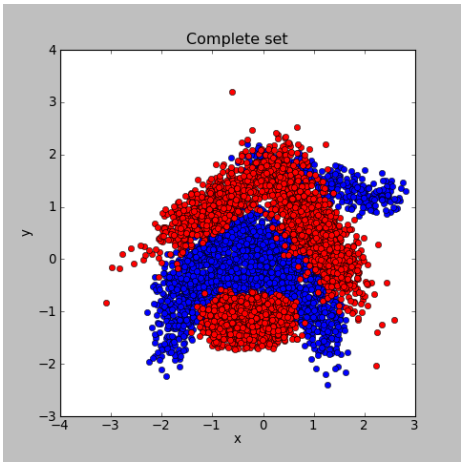
Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. You may include your code inline or submit it as a separate file. You may either scan hand-written work or, preferably, write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython).

1 Introduction

In this problem set, you will work with decision trees and ensemble methods. You will be using the decision tree implementation from `sklearn`, and implementing `AdaBoost` from scratch. You'll also work on some simple theoretical problems that highlight interesting properties of decision trees and ensemble methods.

2 Dataset description

You will be working with a simple two-feature binary dataset, known as the Banana dataset[1], which can be visualized as follows:



[1] `http://mldata.org/repository/data/viewslug/banana-ida/`

(Source: http://adessowiki.fee.unicamp.br/adesso/wiki/courseIA368Q1S2012/eri_test_2/view/)

The data consists of 5,300 instances, which have been split into 3,500 training points and 1,800 test points for this assignment. The csv files are included in the `data` directory. Each row corresponds to a data point - the first entry of the row gives the class label, and the next two entries give the values of the attributes.

3 Decision Trees

3.1 Building Trees by Hand[2]

In this problem we're going to be build a a small decision tree by hand for predicting whether or not a mushroom is poisonous. The training dataset is given below:

Poisonous	Size	Spots	Color
N	5	N	White
N	2	Y	White
N	2	N	Brown
N	3	Y	Brown
N	4	N	White
N	1	N	Brown
Y	5	Y	White
Y	4	Y	Brown
Y	4	Y	Brown
Y	1	Y	White
Y	1	Y	Brown

We're going to build a binary classification tree using the Gini index as the node impurity measure. The feature "Size" should be treated as numeric (i.e. we should find real-valued split points). For a given split, let R_1 and R_2 be the sets of data indices in each of the two regions of the split. Let $\hat{p}_1$ be the proportion of poisonous mushrooms in R_1, and let $\hat{p}_2$ be the proportion in R_2. Let N_1 and N_2 be the total number of training points in R_1 and R_2, respectively. Then the Gini index for the first region is $Q_1 = 2\hat{p}_1(1-\hat{p}_1)$ and $Q_2 = 2\hat{p}_2(1-\hat{p}_2)$ for the second region. When choosing our splitting variable and split point, we're looking to minimize the weighted impurity measure:

$$N_1 Q_1 + N_2 Q_2.$$

1. What is the first split for a binary classification tree on this data, using the Gini index? Work this out "by hand", and show your calculations. [Hint: This should only require calculating 6 weighted impurity measures.]

   Answer: The first splite point should minimize the weighted impurity measure. And from the data we can see we only have six possible splite at this stage:

[2]Based on Homework #4 from David Sontag's DS-GA 1003, Spring 2014.

(a) Splite criterion: { Size ≤ 1; Size > 1 }; then we have $N_1 = 3$, $N_2 = 8$; $\widehat{p_1} = \frac{2}{3}$, $\widehat{p_2} = \frac{3}{8}$; $Q_1 = \frac{4}{9}$, $Q_2 = \frac{30}{64}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 5.083$

(b) Splite criterion: { Size ≤ 2; Size > 2 }; then we have $N_1 = 5$, $N_2 = 6$; $\widehat{p_1} = \frac{2}{5}$, $\widehat{p_2} = \frac{3}{6}$; $Q_1 = \frac{12}{25}$, $Q_2 = \frac{18}{36}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 5.4$

(c) Splite criterion: { Size ≤ 3; Size > 3 }; then we have $N_1 = 6$, $N_2 = 5$; $\widehat{p_1} = \frac{2}{6}$, $\widehat{p_2} = \frac{3}{5}$; $Q_1 = \frac{16}{36}$, $Q_2 = \frac{12}{25}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 5.066$

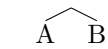
(d) Splite criterion: { Size ≤ 4; Size > 4 }; then we have $N_1 = 9$, $N_2 = 2$; $\widehat{p_1} = \frac{4}{9}$, $\widehat{p_2} = \frac{1}{2}$; $Q_1 = \frac{40}{81}$, $Q_2 = \frac{2}{4}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 6.444$

(e) Splite criterion: { Spots = N; Spots = Y}; then we have $N_1 = 4$, $N_2 = 7$; $\widehat{p_1} = 0$, $\widehat{p_2} = \frac{5}{7}$; $Q_1 = 0$, $Q_2 = \frac{20}{49}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 2.857$

(f) Splite criterion: { Color = White; Color = Brown}; then we have $N_1 = 5$, $N_2 = 6$; $\widehat{p_1} = \frac{2}{5}$, $\widehat{p_2} = \frac{3}{6}$; $Q_1 = \frac{12}{25}$, $Q_2 = \frac{18}{36}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 5.4$

As a result, the first split is whether it has spots.

2. Compute the full decision tree by hand, building until all terminal nodes are either completely pure, or we cannot split any further.
Answer: From the first question, I already know the first split. And the current tree structure is :



Note that all the left children of parent nodes are the "true" nodes, and all the right children is "false" nodes. Next, I look for the split of point A as I did in question 1, and there are only 5 possible split:

(a) Splite criterion: { Size ≤ 1; Size > 1 }; then we have $N_1 = 2$, $N_2 = 5$; $\widehat{p_1} = 1$, $\widehat{p_2} = \frac{3}{5}$; $Q_1 = 0$, $Q_2 = \frac{12}{25}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 2.4$

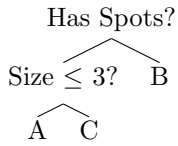
(b) Splite criterion: { Size ≤ 2; Size > 2 }; then we have $N_1 = 3$, $N_2 = 4$; $\widehat{p_1} = \frac{2}{3}$, $\widehat{p_2} = \frac{3}{4}$; $Q_1 = \frac{4}{9}$, $Q_2 = \frac{6}{16}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 2,833$

(c) Splite criterion: { Size ≤ 3; Size > 3 }; then we have $N_1 = 4$, $N_2 = 3$; $\widehat{p_1} = \frac{2}{4}$, $\widehat{p_2} = 1$; $Q_1 = \frac{8}{16}$, $Q_2 = 0$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 = 2$

(d) Splite criterion: { Size ≤ 4; Size > 4 }; then we have $N_1 = 6$, $N_2 = 1$; $\widehat{p_1} = \frac{4}{6}$, $\widehat{p_2} = 1$; $Q_1 = \frac{16}{36}$, $Q_2 = 0$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 2.66666$

(e) Splite criterion: { Color = White; Color = Brown}; then we have $N_1 = 3$, $N_2 = 4$; $\widehat{p_1} = \frac{2}{3}$, $\widehat{p_2} = \frac{3}{4}$; $Q_1 = \frac{4}{9}$, $Q_2 = \frac{6}{16}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 2.883$

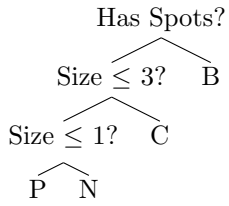
As a result, the split at point A is whether the size is less equal than 3 or not . The current tree is:



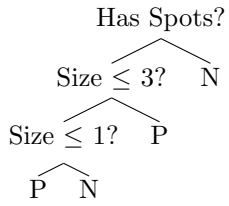
Then I move on to find the splite on point A:

(a) Splite criterion: { Size ≤ 1; Size > 1 }; then we have $N_1 = 2$, $N_2 = 0$; $\widehat{p_1} = 1$, $\widehat{p_2} = 0$; $Q_1 = 0$, $Q_2 = 0$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 = 0$

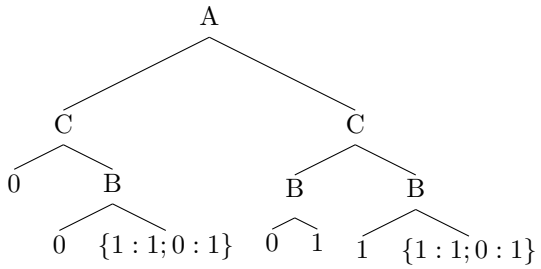
The split at point A is whether the size is less equal than 1 or not . The current tree is:



Then I find point B and C are completely pure : B is not poisonous; C is poisonous. So the final Tree structure is:



3. Suppose we built the same type of tree described above (binary, Gini criterion, terminal nodes are either pure or cannot be split further) on the dataset given below. What would the training error be, given as a percentage? Why? [Hint: You can do this by inspection, without any significant calculations.]
Answer: In this part, I built a tree by inspection, note that all left child is the result when parent node equals to 0, and the right child is the result when parent node equals to 1:

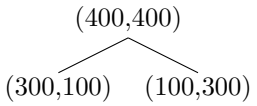


There are two points being incorrect. So the training percent error is: $\frac{2}{11} \approx 0.18$

Y	A	B	C
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	0
0	0	1	1
1	0	1	1
0	1	0	0
1	1	0	1
1	1	1	0
0	1	1	1
1	1	1	1

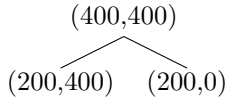
3.2 Investigating Impurity Measures[3]

1. Consider a data set with 400 data points from class C_1 and 400 data points from class C_2 . Suppose that a tree model A splits these into $(300, 100)$ at the first leaf node and $(100, 300)$ at the second leaf node, where (n, m) denotes that n points are assigned to C_1 and m points are assigned to C_2 . Similarly, suppose that a second tree model B splits them into $(200, 400)$ and $(200, 0)$. Show that the misclassification rates for the two trees are equal, but that the cross-entropy and Gini impurity measures are both lower for tree B than for tree A.
   Answer: The first tree is:



   The misclassification rate is $\frac{200}{800} = 0.25$. The cross-entropy is $-\frac{3}{4}\log\frac{3}{4} - \frac{3}{4}\log\frac{3}{4} \approx 0.43$. Then I calculate the Gini impurity measures: $N_1 = 400$, $N_2 = 400$; $\widehat{p_1} = \frac{3}{4}$, $\widehat{p_2} = \frac{3}{4}$; $Q_1 = \frac{6}{16}$, $Q_2 = \frac{6}{16}$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 = 300$

   For the second tree:



[3]From Bishop's *Pattern Recognition and Machine Learning*, Problem 14.11

The misclassification rate is $\frac{200}{800} = 0.25$. The cross-entropy is $-\frac{2}{3}\log\frac{2}{3} - 0 \approx 0.27$. Then I calculated the Gini impurity measures: $N_1 = 600$, $N_2 = 200$; $\widehat{p_1} = \frac{2}{3}$, $\widehat{p_2} = 1$; $Q_1 = \frac{4}{9}$, $Q_2 = 0$. So the weighted impurity measures $WIM = N_1Q_1 + N_2Q_2 \approx 266.67$

Compare the results of the two tree model: although their misclassification rates are equal, the cross-entropy and Gini impurity measure are both lower for the second tree than the first tree.

3.3 Trees on the Banana Dataset

The official `sklearn` documentation provides code that constructs a decision tree and visualizes the decision boundary on the "Iris[4] dataset" (`http://scikit-learn.org/stable/auto_examples/tree/plot_iris.html#example-tree-plot-iris-py`). Note that the `sklearn` implementation of decision trees is a bit different from that described in lecture: they just build to a certain depth, without a pruning step.

1. Modify the code referenced above to work on the Banana dataset. The default class labels are -1 and 1 in the given data files, but for the visualization code snippet to work, you will have to modify the class labels to 0 and 1. Note that the Iris dataset is a multiclass problem with 3 classes, while the Banana dataset is a binary dataset.

   code

```
# data load and precessing
train_data_set = pd.read_csv("data/banana_train.csv",header=None)
test_data_set = pd.read_csv("data/banana_test.csv",header=None)

train_data_X = train_data_set.iloc[:,1:3]
train_data_y = train_data_set.iloc[:,0]

test_data_X = test_data_set.iloc[:,1:3]
test_data_y = test_data_set.iloc[:,0]

clf = DecisionTreeClassifier(max_depth=5).fit(train_data_X,train_data_y)
plt.figure(figsize=(12,10))
plot_step = 0.01

x_min, x_max = train_data_X.iloc[:, 0].min()-0.1, train_data_X.iloc[:,
    0].max()+0.1
y_min, y_max = train_data_X.iloc[:, 1].min()-0.1, train_data_X.iloc[:,
    1].max()+0.1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
```

[4] `https://archive.ics.uci.edu/ml/datasets/Iris`

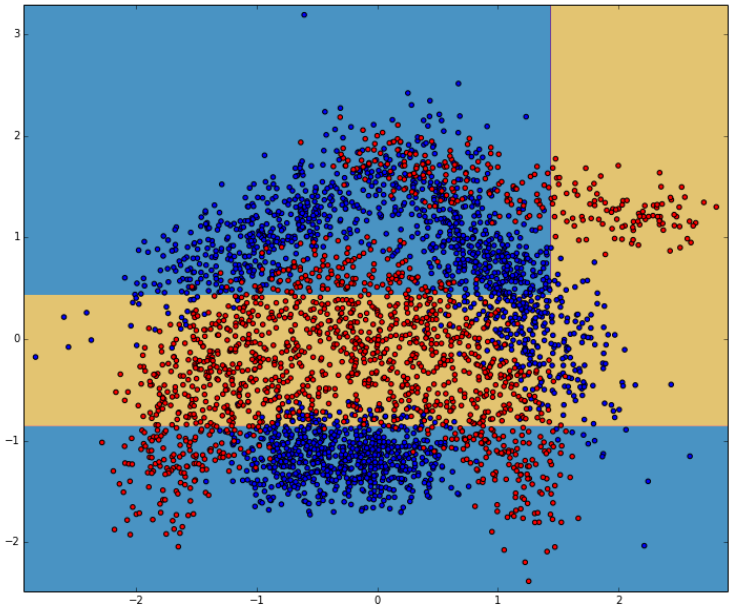


Figure 1: Decision tree with max depth 2

```
plt.axis("tight")
for i, color in [(-1,'b'),(1,'r')]:
   idx = np.where(train_data_y == i)
   plt.scatter(train_data_X.iloc[idx[0], 0], train_data_X.iloc[idx[0], 1],
       c=color,
                cmap=plt.cm.Paired)


plt.show()
```

---

2. Run your code for different depths of decision trees, from 1 through 10, and briefly describe your observations of the decision surface visualization. [Use the default values for all other parameters.]

   Answer: When Max depth = 2, see figure 1.

   When Max depth = 5, see figure 2.

   When Max depth = 7, see figure 3.

   When Max depth = 10, see figure 4.

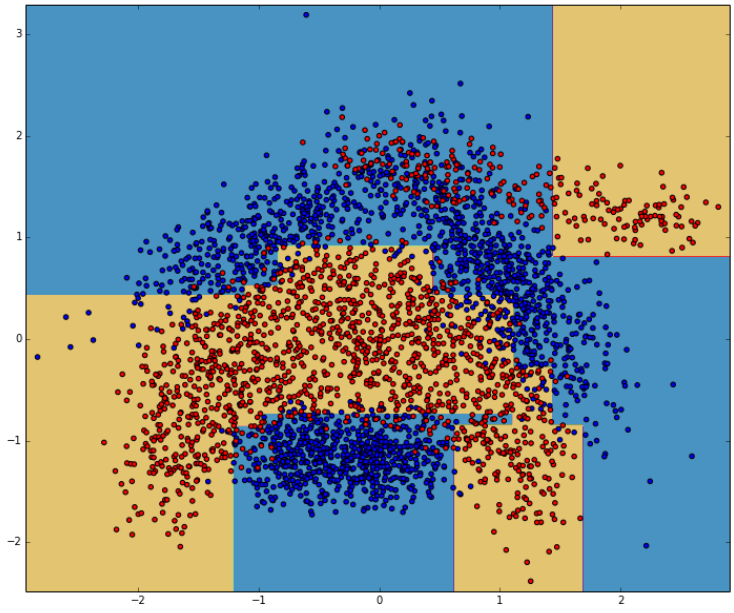


Figure 2: Decision tree with max depth 5

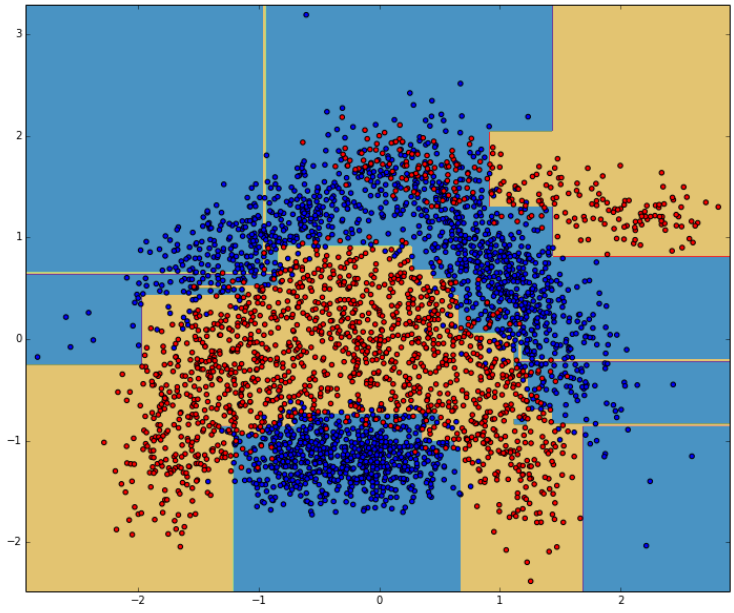


Figure 3: Decision tree with max depth 7

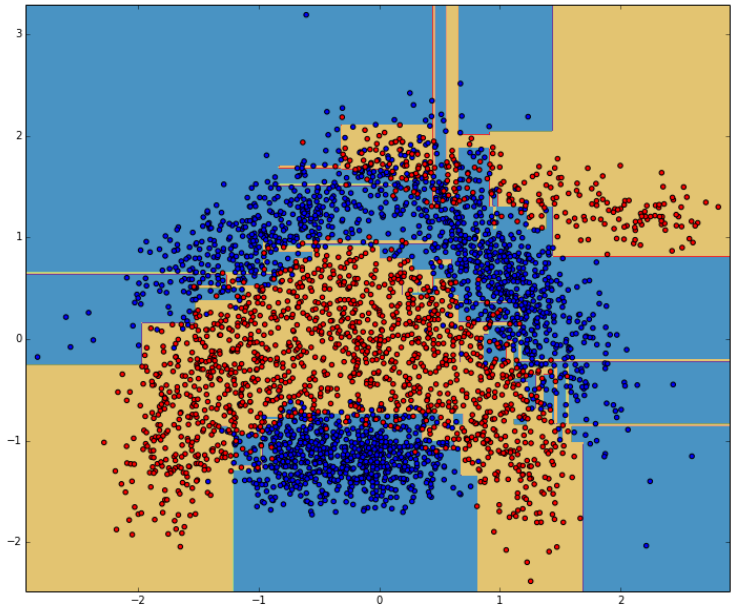


Figure 4: Decision tree with max depth 10

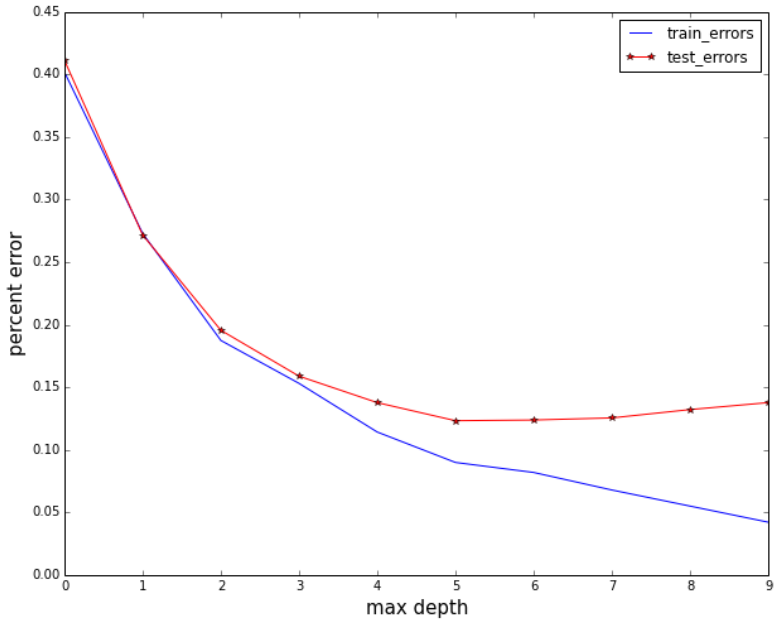


Figure 5: Train and test error against different max depthes

As is shown above, when max depth geting bigger, the tree model become more complex, and more accurate .

3. Plot the train and test errors as a function of the depth. Again, give a brief description of your observations.

   Answer: As is shown in figure 5, train error and test error are very close when max depth is smaller than 3, and they all went down until max depth equals to 5. After that, the train error keeps going down but the test error stop to go up. From this plot, we know the minimum test error is approximate 12%.

4. [Optional] Experiment with the other hyperparameters provided by `DecisionTreeClassifier` and find the combination giving the smallest test error. Summarize what you learn.

4 Bagging[5] [Optional Problem]

Consider a regression problem where we wish to learn function $y(\mathbf{x})$. Suppose we learn M functions $\hat{y}_1(\mathbf{x}), \ldots, \hat{y}_M(\mathbf{x})$. The predictions of each of these functions can be expressed as the sum of the

[5]Based on a problem from Bishop's *Pattern Recognition and Machine Learning*.

true prediction plus an error term

$$\hat{y}_m(\mathbf{x}) = y(\mathbf{x}) + \epsilon_m(\mathbf{x})$$

The expected squared-error of the function is then given by $\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})^2]$. The average squared-error of the models acting individually is therefore

$$E_{av} = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})^2]$$

Bagging involves constructing the final function as an average over the M functions:

$$\hat{y}_{bag}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M}\hat{y}_m(\mathbf{x})$$

The error of bagging is therefore

$$\epsilon_{bag}(\mathbf{x}) = \hat{y}_{bag}(\mathbf{x}) - y(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M}\epsilon_m(\mathbf{x})$$

The expected squared-error is

$$E_{bag} = \mathbb{E}_{\mathbf{x}}[\epsilon_{bag}(\mathbf{x})^2]$$

1. [Optional] Assuming that the individual errors $\epsilon_m(\mathbf{x})$ have mean zero and are uncorrelated, that is, $\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})] = 0$ and $\mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})\epsilon_l(\mathbf{x})] = 0$ for $m \neq l$, show that

$$E_{bag} = \frac{1}{M}E_{av}$$

Answer:

$$\begin{aligned}
E_{bag} &= E_X[\epsilon_{bag}(X^2)] \\
&= E_X[\left(\frac{1}{M}\sum_{m=1}^{M}\epsilon_m(X)\right)^2] \\
&= \frac{1}{M^2}E_X\left[\sum_{m=1}^{M}\sum_{m=1}^{M}\epsilon_m(X)\epsilon_n(X)\right] \\
&= \frac{1}{M^2}E_X\left[\sum_{m=1}\epsilon_m(X)^2\right] + \frac{1}{M^2}\sum_{m\neq n}E_X[\epsilon_m(X)\epsilon_n(X)] \\
&= \frac{1}{M^2}\sum_{m=1}E_X[\epsilon_m(X)^2] \\
&= \frac{1}{M}E_{av}
\end{aligned}$$

2. [Optional] In practice, however, the errors may be highly correlated. Nevertheless, using Jensen's inequality for the special case of the convex function $f(x) = x^2$, show that the average expected squared-error E_{av} of the individual functions and the expected error of bagging E_{bag} satisfy $E_{bag} \leq E_{av}$, without any assumptions on $\epsilon_m(\mathbf{x})$.

   Jensen's inequality can be stated as follows: For a convex function $f(x)$, $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$.
   Answer:

$$\begin{aligned} E_{bag} &= E_X[\epsilon_{bag}(X)^2] \\ &= E_X\left[\left(\frac{1}{M}\sum_{m=1}^{M}\epsilon_m(X)\right)^2\right] \\ &\leq E_X\left[\frac{1}{M}\sum_{m=1}^{M}\epsilon_m^2(X)\right] \\ &= \frac{1}{M}\sum_{m=1}^{M}E_X[\epsilon_m(X)^2] \\ &= E_{av} \end{aligned}$$

5 AdaBoost

5.1 Implementation

In this problem, you will implement AdaBoost, one of the most popular techniques in ensemble methods.

1. Implement AdaBoost for the Banana dataset with decision trees of depth 3 as the weak classifiers (also known as "base classifiers"). Use the decision tree implementation from `sklearn` as in 3.3. The `fit` function of `DecisionTreeClassifier` has a parameter `sample_weight`, which you can use to weigh training examples differently during various rounds of AdaBoost. **Code:**

```
train_err = []
test_err = []

for rounds in range(1,11):

   w = np.array([1.0/len(train_data_X)]*len(train_data_X))
   train_predict = []
   test_predict = []

   for i in range(rounds):

      clf =
          DecisionTreeClassifier(max_depth=3).fit(train_data_X,train_data_y,sample_weight=w)

      W = np.sum(w)
      err = 0
```

```
        train_predict_value = clf.predict(train_data_X)

        for j in range(len(train_data_y)):
            if (train_predict_value[j] != train_data_y[j]):
                err += w[j]/float(W)

        alpha = np.log((1-err)/err)
        for j in range(len(w)):
            if (train_predict_value[j]!=train_data_y[j]):
                w[j] = w[j]*((1-err)/err)


        train_predict.append(alpha * clf.predict(train_data_X))
        test_predict.append(alpha * clf.predict(test_data_X))


    train_predict_sum = np.sum(np.array(train_predict),axis=0)
    test_predict_sum = np.sum(np.array(test_predict),axis=0)
    train_err.append(np.sum(0.5*np.abs(np.sign(train_predict_sum)-train_data_y.values))/
    float(len(train_data_X)))
    test_err.append(np.sum(0.5*np.abs(np.sign(test_predict_sum)-test_data_y.values))/
    float(len(test_data_X)))
```

2. [Optional] Visualize the AdaBoost training procedure for different numbers of rounds from 1 through 10. Plot the decision surface, and the training examples, such that training samples with larger weights in any round are represented as larger points compared to those with smaller weights. Provide a brief description of your observations.

   Answer: Figure 6 shows that, the training procedure has some fluctuations. This is because of the changing weight of sample points. The misclassified point in the previous round will be given a high weight so that it has more influnce on the next round classification which may lead to the misclassification of some other points.

3. Plot the train and test errors as a function of the number of rounds from 1 through 10. Again, give a brief description of your observations.

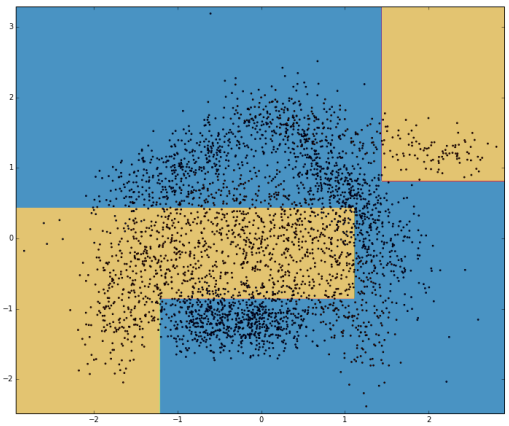
   Answer: The Figure 7 shows that the train and test error is decreased as the adaboost rounds increase. The lowest test error is approximate 12%.

6 Gradient Boosting Machines

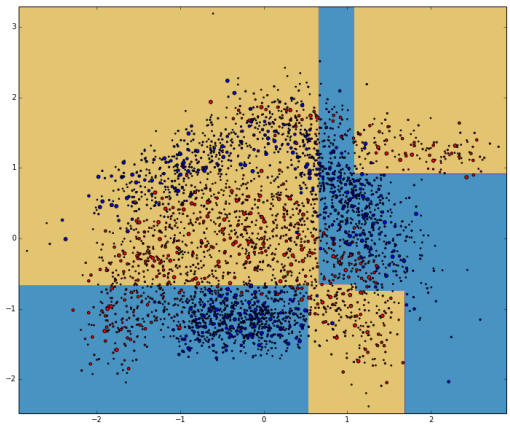
Recall the general gradient boosting algorithm[6], for a given loss function ℓ and a hypothesis space $\mathcal{F}$ of regression functions (i.e. functions mapping from the input space to $\mathbf{R}$):

1. Initialize $f_0(x) = 0$.

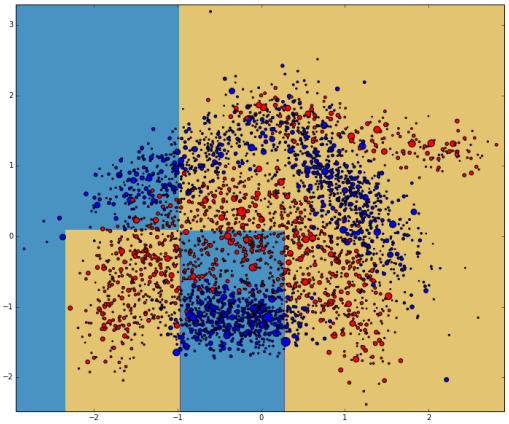
[6]Besides the lecture slides, you can find an accessible discussion of this approach in `http://www.saedsayad.com/docs/gbm2.pdf`, in one of the original references `http://statweb.stanford.edu/~jhf/ftp/trebst.pdf`, and in this review paper `http://web.stanford.edu/~hastie/Papers/buehlmann.pdf`.



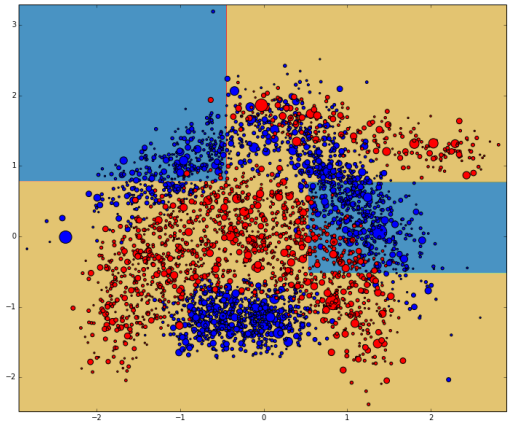
(a) Round 1



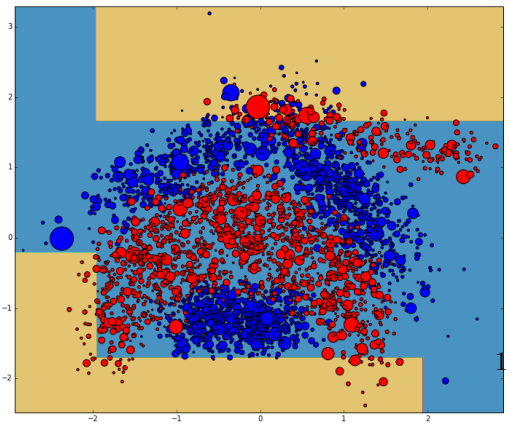
(b) Round 3



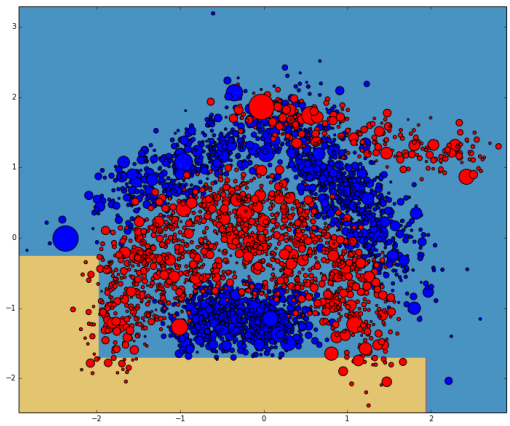
(c) Round 5



(d) Round 7



(e) Round 9



(f) Round 11

Figure 6: AdaBoost train procedure

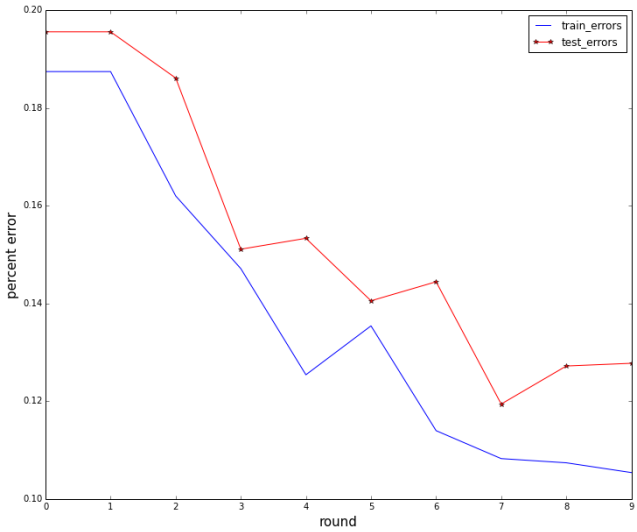


Figure 7: Train & Test Error against Number of Adaboost Rounds

2. For $m = 1$ to M:

   (a) Compute:

$$\mathbf{g}_m = \left(\frac{\partial}{\partial f(x_i)} \sum_{i=1}^{n} \ell\left\{y_i, f(x_i)\right\} \Big|_{f(x_i)=f_{m-1}(x_i)} \right)_{i=1}^{n}$$

   (b) Fit regression model to $-\mathbf{g}_m$:

$$h_m = \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left((-\mathbf{g}_m)_i - h(x_i) \right)^2 .$$

   (c) Choose fixed step size $\nu_m = \nu \in (0, 1]$, or take

$$\nu_m = \arg\min_{\nu > 0} \sum_{i=1}^{n} \ell\left\{y_i, f_{m-1}(x_i) + \nu h_m(x_i)\right\} .$$

   (d) Take the step:

$$f_m(x) = f_{m-1}(x) + \nu_m h_m(x)$$

3. Return f_M.

In this problem we'll derive two special cases of the general gradient boosting framework: L_2-Boosting and BinomialBoost.

1. Consider the regression framework, where $\mathcal{Y} = \mathbf{R}$. Suppose our loss function is given by

$$\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2,$$

and at the beginning of the m'th round of gradient boosting, we have the function $f_{m-1}(x)$. Show that the h_m chosen as the next basis function is given by

$$h_m = \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} [(y_i - f_{m-1}(x_i)) - h(x_i)]^2.$$

In other words, at each stage we find the weak prediction function $h_m \in \mathcal{F}$ that is the best fit to the residuals from the previous stage. [Hint: Once you understand what's going on, this is a pretty easy problem.]

Answer:

$$\begin{aligned}
(g_m)_i &= \frac{\partial}{\partial f(x_i)} \sum_{j=1}^{n} \frac{1}{2}(f(x_j) - y_j)^2 |_{f(x_i)=f_{m-1}(x_i)} \\
&= f_{m-1}(x_i) - y_i \\
h_m &= argmin \sum_{i=1}^{n} (y_i - f_{m-1}(x_i) - h(x_i))^2
\end{aligned}$$

2. Now let's consider the classification framework, where $\mathcal{Y} = \{-1, 1\}$. In lecture, we noted that AdaBoost corresponds to forward stagewise additive modeling with the exponential loss, and that the exponential loss not very robust to outliers (i.e. outliers can have a large effect on the final prediction function). Instead, let's consider instead the logistic loss

$$\ell(m) = \ln\left(1 + e^{-m}\right),$$

where $m = yf(x)$ is the margin. Similar to what we did in the L_2-Boosting question, write an expression for h_m as an argmin over $\mathcal{F}$.

Answer:

$$\begin{aligned}
(g_m)_i &= \frac{\partial}{\partial f(x_i)} \sum_{j=1}^{n} ln(1 + e^{-y_j f(x_j)}) |_{f(x_i)=f_{m-1}(x_i)} \\
&= \frac{-y_i e^{-y_i f(x_i)}}{1 + e^{-y_i f(x_i)}} |_{f(x_i)=f_{m-1}(x_i)} \\
&= \frac{-y_i e^{-y_i f_{m-1}(x_i)}}{1 + e^{-y_i f_{m-1}(x_i)}} \\
h_m &= argmin \sum_{i=1}^{n} \left(\frac{y_i e^{-y_i f_{m-1}(x_i)}}{1 + e^{-y_i f_{m-1}(x_i)}} - h(x_i) \right)^2
\end{aligned}$$