DS-GA 1003: Machine Learning and Computational Statistics
Homework 5: Trees and Boosting

Due: Monday, April 4, 2016, at 6pm (Submit via NYU Classes)

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single file, either HTML or PDF. 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 Dataset description

2 Decision Trees

2.1 Trees on the Banana Dataset

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.

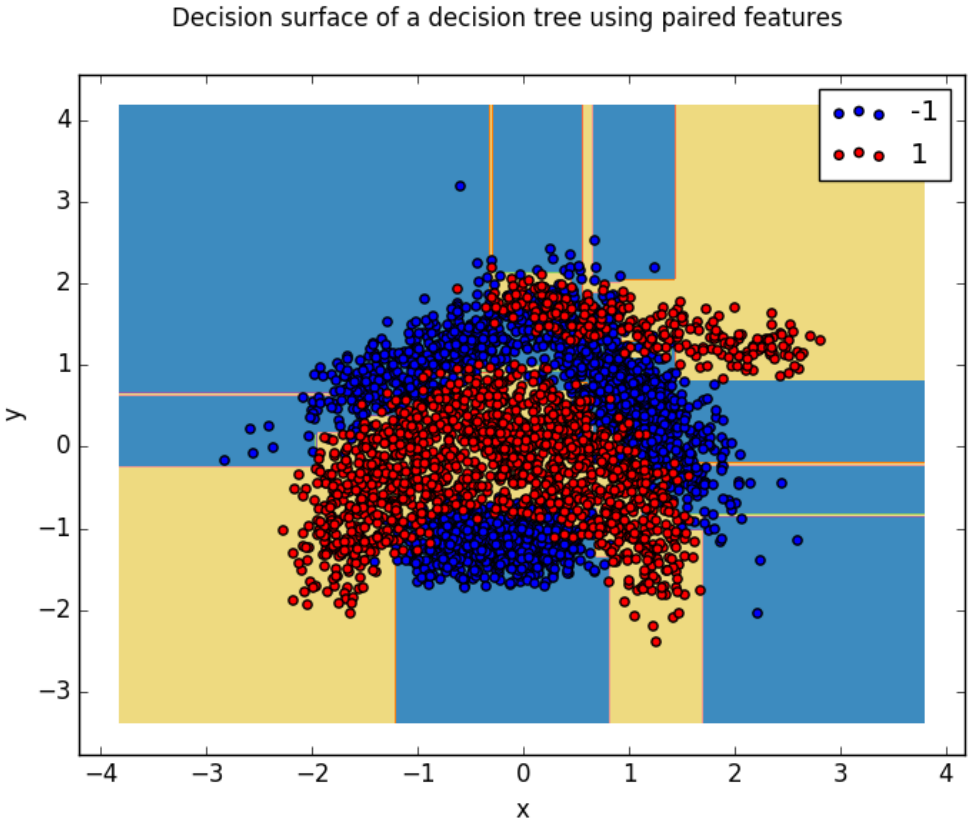
```
n_classes = 2
plot_colors = "br"
plot_step = 0.02

df = pd.read_csv('D:\Academics\Courses\MachineLearning\hw5-boosting\data\\banana_tr
X_train = df.values[:,1:]
y_train = df.values[:,0]
y_train = np.asarray([0 if (y_train[i] == -1) else 1 for i in range(len(y_train))])


df = pd.read_csv('D:\Academics\Courses\MachineLearning\hw5-boosting\data\\banana_te
X_test = df.values[:,1:]
y_test = df.values[:,0]
y_test = np.asarray([(0 if (y_test[i] == -1) else 1) for i in range(len(y_test))])
```

```
# Load data


# Shuffle
idx = np.arange(X_train.shape[0])
np.random.seed(13)
np.random.shuffle(idx)
X_train = X_train[idx]
y_train = y_train[idx]

 # Standardize
mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
#
 # Train
clf = DecisionTreeClassifier(max_depth=10).fit(X_train, y_train)

 # Plot the decision boundary
plt.subplot(1, 1, 1)

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

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

plt.xlabel('x')
plt.ylabel('y')
plt.axis("tight")

target = ['-1', '1']
# Plot the training points
for i, color in zip(range(n_classes), plot_colors):
    idx = np.where(y_train == i)
    plt.scatter(X_train[idx, 0], X_train[idx, 1], c=color, label=target[i],cmap=plt
plt.axis("tight")

plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend()
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.]

Below is the plot obtained for tree depth = 10. As we increase the tree depth from 1 to 10, we see more decision boundaries. There are more lines of division in the graph drawn. This explains the depth of the tree. Also, point to be noted is that as we add more decision lines, the loss decreases.

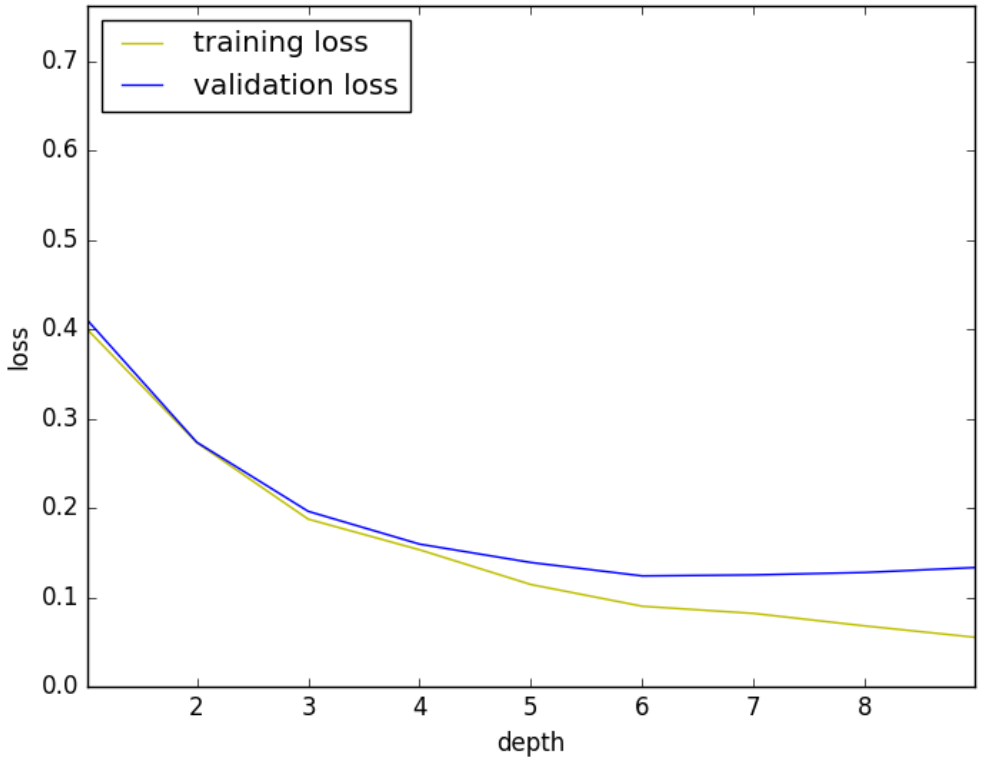


3. Plot the train and test errors as a function of the depth. Again, give a brief description of your observations.

   From the graph drawn below, it is evident that as the depth of the tree increase the loss reduces. Depth implies more decision makers and hence better prediction.

```
all_trainingloss = np.zeros(10)
all_validationloss = np.zeros(10)
x_values = np.zeros(10)
i = 1
while i < 10:
    clf = DecisionTreeClassifier(max_depth=i).fit(X_train, y_train)
    x_values[i] = i
    Z = clf.predict(X_test)
    all_validationloss[i] = compute_loss(y_test, Z)
```

```
        Z = clf.predict(X_train)
        all_trainingloss[i] = compute_loss(y_train, Z)
        i = i + 1

fig, ax = plt.subplots()
ax.plot(x_values, all_trainingloss, '-y', label='training loss')
ax.plot(x_values, all_validationloss, '-b',label='validation loss')
plt.legend(loc='upper left')
plt.axis([1, 9, 0, 100])
plt.xlabel('depth')
plt.ylabel('loss')
plt.show()
```



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

Couple of things I tried here are: Changing the *criterion* to "entropy" Changing $min_samples_split$ Changing $max_features$.

I am not sure how I can be sure that the value reached is of minimum error. One way would be

to increase the max_depth of tree which will reduce the training loss to as low as possible but that would overfit the training data.

3 AdaBoost

3.1 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 2.1. The `fit` function of `DecisionTreeClassifier` has a parameter `sample_weight`, which you can use to weigh training examples differently during various rounds of AdaBoost.

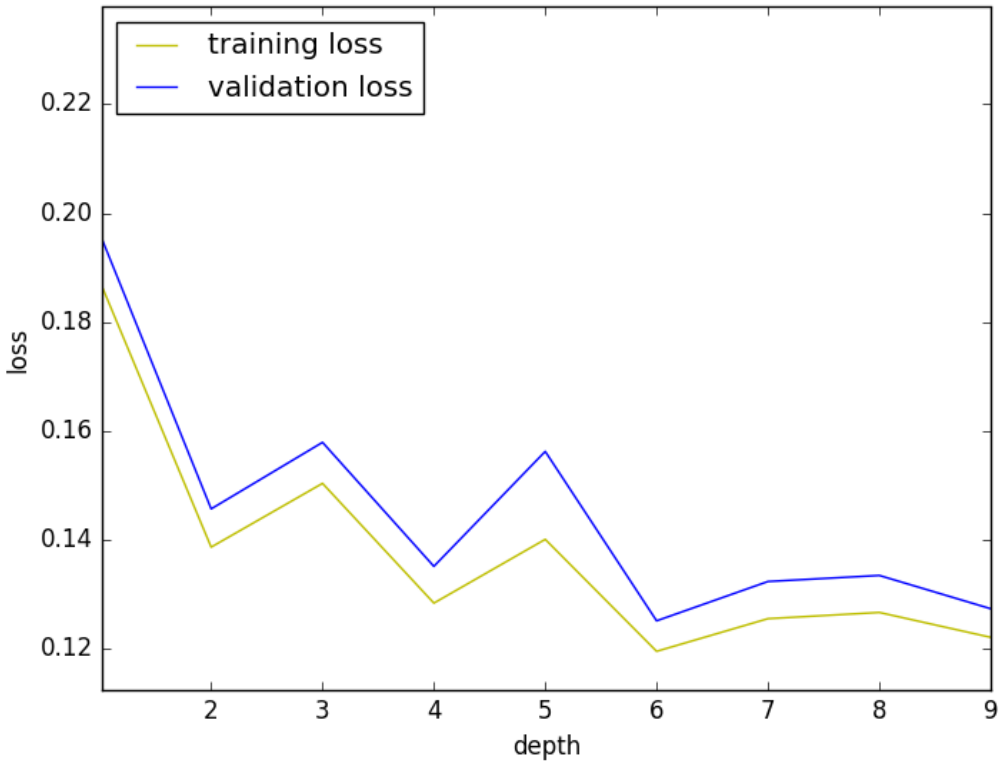
```
y_test = np.asarray([(-1 if (y_test[i] == 0) else 1) for i in range(len(y_test))])
y_train = np.asarray([-1 if (y_train[i] == 0) else 1 for i in range(len(y_train))])


final_prediction = []
total_train_input = X_train.shape[0]
total_test_input = X_test.shape[0]
weights = np.ones(total_train_input)/(X_train.shape[0] * 1.0)
y_final_train = np.zeros(total_train_input)
y_final_test = np.zeros(total_test_input)
loss_train = np.zeros(10)
loss_test = np.zeros(10)
x_values = np.zeros(10)
j = 0
while j < 10:
    clf = DecisionTreeClassifier(max_depth=3).fit(X_train, y_train, weights)
    Z_train = clf.predict(X_train)
    Z_test = clf.predict(X_test)
    err = compute_err(y_train, Z_train, weights)
    loss = compute_loss(Z_train, y_train)
    alpha = 0.5 * math.log((1-err)/err)
    for i in range(0, total_train_input):
        if(Z_train[i] != y_train[i]):
            weights[i] = weights[i] * np.exp(alpha)

    final_prediction.append([clf, alpha])
    y_final_train = y_final_train + alpha * Z_train
    y_final_test = y_final_test + alpha * Z_test

    loss_train[j] = compute_loss(y_final_train, y_train)
```

```
        loss_test[j] = compute_loss(y_final_test, y_test)
        x_values[j] = j

        #err_on_combined = compute_err_combined(X_test, y_test, final_prediction)
        j = j+1

    fig, ax = plt.subplots()
    ax.plot(x_values, loss_train, '-y', label='training loss')
    ax.plot(x_values, loss_test, '-b', label='validation loss')
    plt.legend(loc='upper left')
    plt.axis([1, 9, 0, 0.5])
    plt.xlabel('depth')
    plt.ylabel('loss')
    plt.show()
```

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.

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.

   Below is the graph of validation loss and training loss as function of rounds. Note that even though there are ups and down in adaboost, eventually the training loss is reducing over rounds. The same pattern can be seen with respect to validation loss also.



4 Gradient Boosting Machines

Recall the general gradient boosting algorithm[1], 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 $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute:

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

[1] 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`.

(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

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

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.]

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 $\mathcal{F}$.

5 From Margins to Conditional Probabilities[2]

Let's consider the classification setting, in which $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{-1, 1\}$ are sampled i.i.d. from some unknown distribution. For a prediction function $f : \mathcal{X} \to \mathbf{R}$, we define the **margin**

[2]This problem is based on Section 7.5.3 of Schapire and Freund's book *Boosting: Foundations and Algorithms*.

on an example (x, y) to be $m = yf(x)$. Since our class predictions are given by $\text{sign}(f(x))$, we see that a prediction is correct iff $m(x) > 0$. We have said we can interpret the magnitude of the margin $|m(x)|$ as a measure of confidence. However, it is not clear what the "units" of the margin are, so it is hard to interpret the magnitudes beyond saying one prediction is more or less confident than another. In this problem, we investigate how we can translate the margin into a conditional probability, which is much easier to interpret. In other words, we are looking for a mapping $m(x) \mapsto p(y = 1 \mid x)$.

In this problem we will consider margin losses. A loss function is a **margin loss** if it can be written in terms of the margin $m = yf(x)$. We are interested in how we can go from an empirical risk minimizer of a margin loss, $\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(y_i f(x_i))$, to a conditional probability estimator $\hat{\pi}(x) \approx p(y = 1 \mid x)$. Our approach will be to find the mapping in terms of the Bayes prediction function[3], and then apply the same mapping to the empirical risk minimizer. While there is plenty that can go wrong with this "plug-in" approach (primarily, the empirical risk minimizer from a hypothesis space $\mathcal{F}$ may be a poor estimate for the Bayes prediction function), it is at least well-motivated, and it can work well in practice. And please note that we can do better than just hoping for success: if you have enough validation data, you can directly assess how well "calibrated" the predicted probabilities are. This blog post has some discussion of calibration plots: https://jmetzen.github.io/2015-04-14/calibration.html.

It turns out it is straightforward to find the Bayes prediction function f^* for margin losses, at least in terms of the data-generating distribution: For any given $x \in \mathcal{X}$, we'll find the best possible prediction $\hat{y}$. This will be the $\hat{y}$ that minimizes

$$\mathbb{E}_y \left[\ell(y\hat{y}) \mid x \right].$$

If we can calculate this $\hat{y}$ for all $x \in \mathcal{X}$, then we will have determined $f^*(x)$. We will simply take

$$f^*(x) = \arg\min_{\hat{y}} \mathbb{E}_y \left[\ell(y\hat{y}) \mid x \right].$$

It may be intuitively obvious that this f^* is the risk minimizer. Here's a mathematical proof:

$$\begin{aligned}
\min_f \mathbb{E}_{x,y} \ell(yf(x)) &= \min_f \mathbb{E}_x \left[\mathbb{E}_y \left[\ell(yf(x)) \mid x \right] \right] \\
&\geq \mathbb{E}_x \left[\min_{\hat{y}} \mathbb{E}_y \left[\ell(y\hat{y}) \mid x \right] \right] \\
&= \mathbb{E}_x \left[\mathbb{E}_y \left[\ell(yf^*(x)) \mid x \right] \right] \\
&= \mathbb{E}_{x,y} \ell(yf^*(x)).
\end{aligned}$$

But of course we must also have $\min_f \mathbb{E}_{x,y} \ell(yf(x)) \leq \mathbb{E}_{x,y} \ell(yf^*(x))$. So the inequality must actually be an equality, and thus the minimum is attained at f^*.

Below we'll calculate f^* for several loss functions. It will be convenient to let $\pi(x) = \mathbb{P}(y = 1 \mid x)$ in the work below.

1. Write $\mathbb{E}_y \left[\ell(yf(x)) \mid x \right]$ in terms of $\pi(x)$ and $\ell(f(x))$. [Hint: Use the fact that $y \in \{-1, 1\}$.]

[3] In this context, the Bayes prediction function is often referred to as the "population minimizer." The term "population" makes most sense in a context where we are using a sample to approximate some statistic of an entire population. In our case, "population" referes to the fact that we are minimizing with respect to the true distribution, rather than a sample.

2. Show that the Bayes prediction function $f^*(x)$ for the exponential loss function $\ell(y, f(x)) = e^{-yf(x)}$ is given by

$$f^*(x) = \frac{1}{2} \ln \left(\frac{\pi(x)}{1 - \pi(x)} \right)$$

and, given the Bayes prediction function f^*, we can recover the conditional probabilities by

$$\pi(x) = \frac{1}{1 + e^{-2f^*(x)}}.$$

[Hint: Differentiate the expression in the previous problem with respect to $f(x)$. If this is confusing, you may find it more comforting to change variables a bit: Fix an $x \in \mathcal{X}$. Then write $p = \pi(x)$ and $\hat{y} = f(x)$. After substituting these into the expression you had for the previous problem, you'll want to find $\hat{y}$ that minimizes the expression. Use differential calculus. Once you've done it for a single x, it's easy to write the solution as a function of x.]

3. Show that the Bayes prediction function $f^*(x)$ for the logistic loss function $\ell(y, f(x)) = \ln\left(1 + e^{-yf(x)}\right)$ is given by

$$f^*(x) = \ln \left(\frac{\pi(x)}{1 - \pi(x)} \right)$$

and the conditional probabilities are given by

$$\pi(x) = \frac{1}{1 + e^{-f^*(x)}}.$$

Again, we may assume that $\pi(x) \in (0, 1)$.

4. [Optional] Show that the Bayes prediction function $f^*(x)$ for the hinge loss function $\ell(y, f(x)) = \max(0, 1 - yf(x))$ is given by

$$f^*(x) = \text{sign} \left(\pi(x) - \frac{1}{2} \right).$$

Note that it is impossible to recover $\pi(x)$ from $f^*(x)$ in this scenario. However, in practice we work with an empirical risk minimizer, from which we may still be able to recover a reasonable estimate for $\pi(x)$. An early approach to this problem is known as "Platt scaling": https://en.wikipedia.org/wiki/Platt_scaling.

6 AdaBoost Actually Works [Optional]

Introduction

Given training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where y_i's are either $+1$ or -1, suppose we have a weak learner G_t at time t and we will perform AdaBoost T times. Initialize observation weights uniformly by setting $W^1 = (w_1^1, \ldots, w_n^1)$ and $w_i = 1/n$ for $i = 1, 2, \ldots, n$. For $t = 1, 2, \ldots, n$:

1. Fit the weak learner at time t to weighted samples: G_t that depends on (D, W^t)
2. Compute the weighted misclassifications: $\text{err}_t = \sum_D w_i^t 1(G_t(x_i) \neq y_i) / \sum_i w_i^t$

3. Compute the contribution coefficient for the weak learner: $\alpha_t = 1/2 \log(\frac{1}{\text{err}_t} - 1)$

4. Update the weights: $w_i^{t+1} = w_i^t \exp(-\alpha_t y_i G_t(x_i))$

After T steps, the cumulative contributions of weak learners is $G(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t G_t(x))$ as the final output. We will prove that with a reasonable weak learner the error of the output decreases exponentially fast with the number of iterations.

Exponential bound on the training loss

More precisely, we will show that the training error $L(G, D) = \frac{1}{n} \sum_{i=1}^{n} 1(G(x_i) \neq y_i) \leq \exp(-\gamma^2 T)$ where the error of the weak learner is less than $1/2 - \gamma$ for some $\gamma > 0$. To start, let's denote two cumulative variables: the output at time t as $f_t = \sum_{s \leq t} \alpha_s G_s$ and $Z_t = \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i f_t(x_i))$.

1. For any function g into $\{-1, +1\}$, show that $1(g(x) \neq y) < \exp(-yg(x))$.

2. Use this to show $L(G, D) < Z_T$

3. Show that $w_i^{t+1} = \exp(-y_i f_t(x_i))$

4. Use part 3 to show $\frac{Z_{t+1}}{Z_t} = 2\sqrt{\text{err}_{t+1}(1 - \text{err}_{t+1})}$ (Hint: use the definition of weight updates and separate the sum on where G_t is equal to 1 and -1.)

5. Show that the function $g(a) = a(1 - a)$ is monotonically increasing on $[0, 1/2]$. Show that $1 - a \leq \exp(-a)$. And use the assumption on the weak learner to show that $\frac{Z_{t+1}}{Z_t} \leq \exp(-2\gamma^2)$

6. Conclude the proof!

7 AdaBoost is FSAM With Exponential Loss [Optional]

Introduction

The main function in AdaBoost, $G(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t G_t(x))$, is an additive expansion in a set of 'basis' functions, $f(x) = \sum_{t=1}^{T} \alpha_t G_t(x)$. The function f is similar to the way of representing a vector as a linear combination of the basis vectors in linear algebra: Given a set of basis elements, find the correct coefficients. Here we have $G_t(x)$'s as basis functions and α's as coefficients.

In the additive model, the algorithm starts by initializing $f_0(x) = 0$, and then for $t = 1, \ldots, T$ iterate over the following for some loss function L:

1. Compute $(\alpha_t, G_t) = \text{argmin}_{\alpha, G} \sum_{i=1}^{n} L(y_i, f_{t-1}(x_i) + \alpha G(x_i))$

2. Find the expansion at time t: $f_t(x) = f_{t-1}(x) + \alpha_t G_t(x)$

In the next problem, show that using exponential loss will lead to AdaBoost.

Exponential loss and AdaBoost

Consider the loss function $L(y, f(x)) = \exp(-yf(x))$.

1. Write the first step of the additive model using the exponential loss function. Show that it can be written as:
$$(\alpha_t, G_t) = \text{argmin}_{\alpha,G} \sum_{i=1}^{n} w_i^t \exp(-\alpha y_i G(x_i)))$$

2. Show that for fixed positive alpha:
$$G_t = \text{argmin}_G \sum_{i=1}^{n} w_i^t 1(G(x_i) \neq y_i)$$
(Hint: split the sum in part 1 for $y_i = G(x_i)$ and otherwise.)

3. Plug this G_t back into the first equation and solve for α to obtain $\alpha_t = \frac{1}{2} \log \frac{1}{\text{err}_t} - 1$

4. Show that the weight iterations are given by:
$$w_i^{t+1} = w_i^t \exp(-\alpha_t y_i G_t(x_i))$$
And conclude the equivalence.