10701/15781 Machine Learning, Spring 2007: Homework 2

Due: Wednesday, February 21, beginning of the class

Instructions

There are 4 questions on this assignment. The second question involves coding. Do *not* attach your code to the writeup. Instead, copy your implementation to

`/afs/andrew.cmu.edu/course/10/701/Submit/your_andrew_id/HW2`

Refer to the webpage for policies regarding collaboration, due dates, and extensions. To write in this directory, you need a kerberos instance for andrew, or you can log into, for example, unix.andrew.cmu.edu.

1 Decision Trees vs. Linear Classifiers [Andy, 15 points]

Our favorite billionaire wants to hire a team to work on a new operating system called Goosoft. He wants to classify job applicants into good programmers (worth hiring) and bad programmers based on their GPA and top score in Minesweeper. He wants to use a classifier, but cannot choose which one, so he needs the help of a machine learning expert. You happily offer your help.

1.1 Single-node Decision Tree As Linear Classifier [3 points]

When the features are continuous, a decision tree with one node (a depth 1 decision tree) can be viewed as a linear classifier (such degenerate trees, consisting of only one node and therefore using only one variable to split the data, are also called *decision stumps*). What is the difference between the decision boundary for this classifier and other linear classifiers you have learned in class, such as logistic regression (LR)?

Provide a small dataset (no more than 10 2-dimensional points), for which a perfect classifier can be trained using LR, but no one-node decision tree is a perfect classifier. Qualitatively show the decision boundary of the classifier learned using LR and one-node decision tree. Justify your answer.

1.2 General Decision Trees vs. Linear Classifiers [5 points]

If we allow decision trees to have depth more than 1, they can capture more complex separation surfaces. Provide a small 2D dataset for which there exists a decision tree that is a perfect classifier, but no perfect linear classifier exists. Show (again qualitatively) the resulting decision tree, and linear classifier learned by LR.

1.3 Using Linear Classifiers In Decision Trees [7 points]

Now that we saw that there are cases in which decision trees are preferable, and there are cases in which linear classifiers are preferable, let us combine their advantages. Your goal is to sketch an algorithm that would train a linear-classifier-augmented decision tree: at each node, instead of testing some particular feature to split the decision tree, we now use the output of some linear classifier. Positively classified examples will go to one branch of the tree, while negatively classified ones will follow the other branch.

Assume that you have a black box that takes a training set as input and gives you an optimal (but not necessarily perfect!) linear classifier as the output. Do not worry about the stopping criteria - grow the tree until the classifier is perfect for the training set.

Provide a dataset, for which no perfect linear classifier exist, and your algorithm outputs a decision tree with depth smaller than the depth of a standard decision tree. Qualitatively show both resulting trees, and justify your answer.

2 Boosting [Purna/Brian, 50 points]

The details of Adaboost are in *Robert E. Schapire. The boosting approach to machine learning: An overview.* In Nonlinear Estimation and Classification. Springer, 2003. `http://www.cs.princeton.edu/~schapire/uncompress-papers.cgi/msri.ps`

The alogrithm details of Schapire's tutorial differ slightly from those in the textbook. Both will yield the same results, but the internal values of weights will differ. The proofs of 2.1 follow Schapire's tutorial. Please use Schapire's algorithm for Problem 2.1 and 2.2. You may implement either and should obtain identical results for Problem 2.3.

2.1 [20 Points] Analyzing the training error of boosting

Consider the AdaBoost algorithm you saw in class. In this question we will try to analyze the training error of Boosting.

1. Let $h_t(x)$ be the weak classifier obtained at step t, and let α_t be its weight. Recall that the final classifier is
$$H(x) = \text{sign}(f(x)), \text{ where } f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$
Show that the training set error of the final classifier can be bounded from above by an exponential loss function (y_i is the true class of x_i):
$$\frac{1}{m}\sum_{i=1}^{m}(H(x_i) \neq y_i) \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-f(x_i)y_i),$$
Hint: $e^{-x} \geq 1 \Leftrightarrow x \leq 0$.

2. Remember that
$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$
Use this recursive definition to prove the following.
$$\frac{1}{m}\sum_{i=1}^{m}\exp(-f(x_i)y_i) = \prod_{t=1}^{T} Z_t, \tag{1}$$
where Z_t is the normalization factor for distribution D_{t+1}:
$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i)). \tag{2}$$
Hint :Remember that $e^{\sum_i g_i} = \prod_i e^{g_i}$,$D_1(i) = \frac{1}{m}$, *and that* $\sum_i D_{t+1}(i) = 1$.

3. Equation 1 suggests that the training error can be reduced rapidly by greedily optimizing Z_t at each step. You have showed that the error is bounded from above:
$$\epsilon_{training} \leq \prod_{t=1}^{T} Z_t.$$

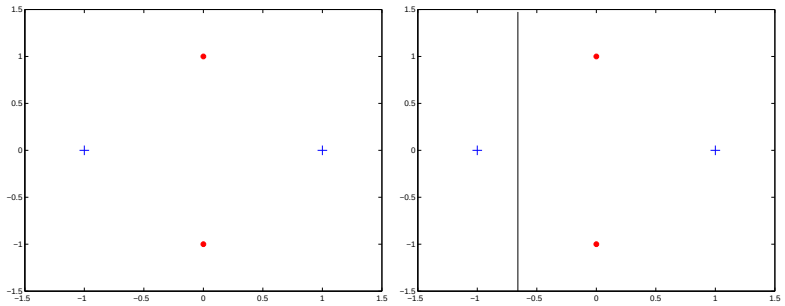

Figure 1: 1 a)Toy data in Question 2. b) h_1 in Question 2

Observe that $Z_1, \ldots, Z_{t-1}$ are determined by the first $(t-1)$ rounds of boosting, and we cannot change them on round t. A greedy step we can take to minimize training set error bound on round t is to minimize Z_t.

In this question, you will prove that for binary base classifiers Z_t from Equation 2 is minimized by picking α_t as:

$$\alpha_t^* = \frac{1}{2} \log \left(\frac{1 - \epsilon_t}{\epsilon_t} \right), \tag{3}$$

where ϵ_t is the training set error of weak classifier h_t for weighted dataset:

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) I(h_t(x_i) \neq y_i).$$

where I is the indicator function. For this proof, only consider the simplest case of binary classifiers, i.e. the range of $h_t(x)$ is binary, $\{-1, +1\}$.

For this special class of classifiers, first show that the normalizer Z_t can be written as:

$$Z_t = (1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t).$$

Hint: consider the sums over correctly and incorrectly classified examples separately.

Now, prove that the value of α_t that minimizes this definition of Z_t is given by Equation 3.

4. Prove that for the above value of α_t

$$Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$$

5. In class you will see that

$$\epsilon_{training} \leq \prod_t Z_t \leq exp(-2 \sum_t \gamma_t^2) \tag{4}$$

Where $\gamma_t = \frac{1}{2} - \epsilon_t$. The above equations shows that , if each base classifier is slightly better than random, so that $\gamma_t \geq \gamma$, for some $\gamma > 0$, then the training error drops exponentially fast in T.

Show that $\epsilon_t \leq 0.5$. Also show that for $\epsilon_t = 0.5$ the training error can get "stuck" above zero. *Hint: the $D_t(i)$'s do not change over t.*

2.2 [5 Points] Adaboost on a toy dataset

Now we will apply adaboost to classify a toy dataset. Consider the following dataset in Figure 1a). The dataset consists of 4 points , $(X_1 : 0, -1, -)$, $(X_2 : 1, 0, +)$, $(X_3 : -1, 0, +)$ and $(X_4 : 0, 1, -)$.

1. Use simple decision stumps as weak base classifiers. Now for $T = 4$, show how Adaboost works for this dataset. For each timestep remember to compute the following :

$$\epsilon_t, \alpha_t, Z_t, D_t(i) \forall i,$$

Also for each timestep draw your weak classifier. For example h_1 can be as shown in 1b).

2. What is the training error of Adaboost?

3. Is the above dataset linearly separable ? Explain why Adaboost does better than a decision stump in the above dataset.

2.3 [25 Points] Implementation

Implement the AdaBoost algorithm (page 658) using a decision stump as the weak classifier.

AdaBoost trains a sequence of classifiers. Each classifier is trained on the same set of training data $\{\mathbf{X}, \mathbf{Y}\}$, but with the significance $D_t(i)$ of each example $\{\mathbf{x_i}, y_i\}$ weighted differently. At each iteration, a classifier, $f_m(\mathbf{x}) \rightarrow \{-1, 1\}$, is trained to minimize the weighted classification error, $\sum_i D_t(i) * I(f_m(\mathbf{x_i}) \neq y_i)$, where I is the indicator function (0 if the predicted and actual label match, and 1 otherwise). The overall prediction of the AdaBoost algorithm is a linear combination of these classifiers, $F_M(\mathbf{x}) = sign(\sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}))$. *Note:* The textbook uses $w_i \equiv D_t(i)$.

A decision stump is a decision tree with a single node. It corresponds to a single threshold in one of the axes of the features $\mathbf{X}$, and predicts the class for examples falling above and below the threshold, $f_m(\mathbf{x}) = C_1 I(x_j \geq T) + C_2 I(x_j < T)$. Unlike in class, where we split on Information Gain, for this algorithm split the data based on the weighted classification accuracy described above, and find the class assignments $C_1, C_2 = \{-1, 1\}$, threshold T, and feature choice j that maximizes this accuracy.

1. Submit your source code to:
   /afs/andrew.cmu.edu/course/10/701/Submit/your_andrew_id/HW2

2. Evaluate your AdaBoost implementation on the Bupa Liver Disorder dataset that is available for download from the course website. The classification problem is to predict whether an individual has a liver disorder (indicated by the selector feature) based on the results of a number of blood tests and levels of alcohol consumption. Use 90% of the dataset for training and 10% for testing. Average your results over 50 random splits of the data into testing sets and training sets. Limit the number of boosting iterations to 100. In a single plot show the:

   - Training set error average after each boosting iteration
   - Testing set error average after each boosting iteration
   - Training error bounds ($\prod_t Z_t$) average after each boosting iteration

3. Using all of the data for training, display the selected axis (i.e. feature j) and threshold (T) of the decision stump classifier, $f_m(\mathbf{x})$ used in each of the first 10 boosting interations ($m = 1, 2, ..., 10$).

3 Linear Regression and LOOCV [Jon, 20 points]

In class, you learned about using cross validation as a way to estimate the true error of a learning algorithm. The preferred solution is *Leave-One-Out Cross Validation*, which provides an almost unbiased estimate of this true error, but it can take a really long time to compute. In this problem, you will derive an algorithm for efficiently computing the leave-one-out cross validation error (LOOCV) for linear regression using the *Hat Matrix*. (This is the *cool trick* alluded to in the slides!)

Assume that there are r given training examples, $(X_1, Y_1), (X_2, Y_2), \ldots, (X_r, Y_r)$, where each input data point X_i, has n real valued features. The goal of regression is to learn to predict Y from X. The linear regression model assumes that the output Y is a linear combination of the input features plus Gaussian noise with weights given by β.

We can write this in matrix form by stacking the datapoints as the rows of a matrix X so that x_{ij} is the j-th feature of the i-th datapoint. Then writing Y, β and ϵ as column vectors, we can write the matrix form of the linear regression model as:

$$Y = X\beta + \epsilon$$

where:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_r \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_r \end{bmatrix}, \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \text{and } X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{r1} & x_{r2} & \dots & x_{rn} \end{bmatrix}$$

Assume that ϵ_i is normally distributed with variance σ^2. We saw in class that the maximum likelihood estimate of the model parameters β (which also happens to minimize the sum of squared prediction errors) is given by the *Normal equation*:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Define $\hat{Y}$ to be the vector of predictions using $\hat{\beta}$ if we were to plug in the original training set X:

$$\begin{aligned} \hat{Y} &= X\hat{\beta} \\ &= X(X^T X)^{-1} X^T Y \\ &= HY \end{aligned}$$

where we define $H = X(X^T X)^{-1} X^T$ (H is often called the *Hat Matrix*).

As mentioned above, $\hat{\beta}$, also minimizes the sum of squared errors:

$$\text{SSE} = \sum_{i=1}^{r} (Y_i - \hat{Y}_i)^2$$

Now recall that the Leave-One-Out Cross Validation score is defined to be:

$$\text{LOOCV} = \sum_{i=1}^{r} (Y_i - \hat{Y}_i^{(-i)})^2$$

where $\hat{Y}^{(-i)}$ is the estimator of Y after removing the i-th observation (i.e., it minimizes $\sum_{j \neq i} (Y_j - \hat{Y}_j^{(-i)})^2$).

1. *(3 points)* What is the complexity of computing the LOOCV score naively? (The naive algorithm is to loop through each point, performing a regression on the $r - 1$ remaining points at each iteration.)

   Hint: The complexity of matrix inversion for a $k \times k$ matrix is $O(k^3)$.

2. *(3 points)* Write $\hat{Y}_i$ in terms of H and Y.

3. *(5 points)* Show that $\hat{Y}^{(-i)}$ is also the estimator which minimizes SSE for Z where

$$Z_j = \begin{cases} Y_j, & j \neq i \\ \hat{Y}_i^{(-i)}, & j = i \end{cases}$$

4. *(5 points)* Show that $\hat{Y}_i^{(-i)} = \hat{Y}_i - H_{ii} Y_i + H_{ii} \hat{Y}_i^{(-i)}$, where H_{ii} denotes the i-th element along the diagonal of H.

5. *(4 points)* Show that

$$LOOCV = \sum_{i=1}^{r} \left(\frac{Y_i - \hat{Y}_i}{1 - H_{ii}} \right)^2$$

   What is the algorithmic complexity of computing the LOOCV score using this formula?

   Note: We see from this formula that the diagonal elements of H somehow indicate the impact that each particular observation has on the result of the regression.

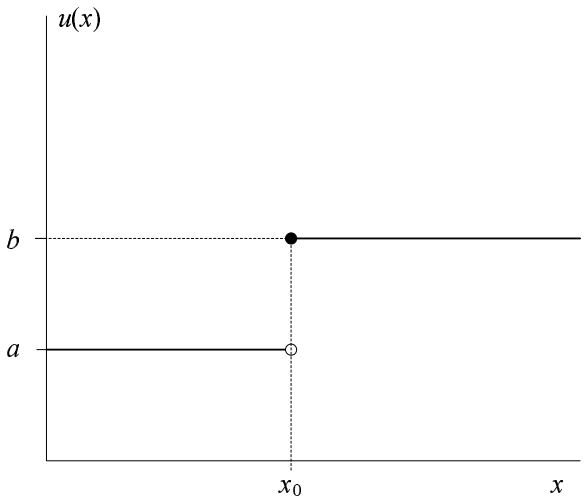

Figure 2: Step function in Question 4.

4 Neural Networks [Andy, 15 points]

In this question, you will prove that a neural network with a single hidden layer can provide an arbitrarily close approximation to any 1-dimensional bounded smooth function.

Suppose that you have two types of activation functions at hand:

- linear $y = w_0 + \sum_i w_i x_i$,
- hard threshold

$$y = \begin{cases} 1 & \text{if } w_0 + \sum_i w_i x_i \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

1. Consider the step function $u(x)$ in Figure 2. Construct a neural network with one input x and one hidden layer whose response is $u(x)$. That is, if $x < x_0$, the output of your network should be a, whereas if $x \geq x_0$, the output should be b. Draw the structure of the neural network and specify the parameters as a function of a, b, and x_0.

2. Now consider the indicator function $\delta(x \in [a, b))$:

$$\delta(x \in [a, b)) = \begin{cases} 1, & \text{if } x \in [a, b); \\ 0, & \text{otherwise.} \end{cases}$$

   Construct a neural network with one input x and one hidden layer whose response is $\delta(x \in [a, b))$. Draw the structure of the neural network and specify the parameters as a function of a and b. (Note: we previously asked for the parameters as a function of a, b, and x_0. x_0 was removed because it is irrelevant for this part of the problem).

3. You are now given any function $f(x)$ whose domain is $[C; D)$. Suppose that the function is Lipschitz continuous[1], that is,

$$\forall x, x' \in [C; D), \quad |f(x') - f(x)| \leq L|x' - x|, \tag{6}$$

   for some Lipschitz constant $L \geq 0$. Use the intuition from the previous part to construct a neural network with one hidden layer that approximates this function within $\epsilon > 0$, that is, $\forall x \in [C; D), \ |f(x) - \hat{f}(x)| \leq \epsilon$, where $\hat{f}(x)$ is the output of the neural network for the input x. Your network should use only the linear and hard threshold activation functions from parts 1 and 2 of this exercise. The structure

[1] Lipschitz continuity is a smoothness condition that limits how fast a function can change.

and parameters should be specified in terms of C, D, L, ϵ, and $f(x)$ evaluated at a finite number of points. Why does your neural network attain the given accuracy ϵ?

Hint: think of Riemann sum.