10701/15781 Machine Learning, Spring 2005: Homework 2

Due: Monday, February 20, beginning of the class

1 [50 Points] Boosting [Anton]

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`

1.1 [20 Points] Parameter choice for boosting

In this question you will justify the choice of the parameter α_t in AdaBoost algorithm that you saw in class.

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):
$$\frac{1}{m}\sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \frac{1}{m}\sum_{i=1}^{m} \exp(-f(x_i)y_i),$$
where
$$\delta(H(x_i) \neq y_i) \equiv \begin{cases} 1, & H(x_i) \neq y_i \\ 0, & H(x_i) = y_i \end{cases}.$$
Hint: $e^{-x} \geq 1 \Leftrightarrow x \leq 0$.

2. Prove that
$$\frac{1}{m}\sum_{i=1}^{m} \exp(-f(x_i)y_i) = \prod_{t=1}^{T} Z_t,$$
where Z_t is the normalization factor for distribution D_{t+1}:
$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i)). \quad (1)$$
Hint 1: using (1), write $D_2(i)$ as a function of Z_1, m, α_1, y, h_1, then $D_3(i)$ as a function of $Z_1, Z_2, m, \alpha_1, \alpha_2, y, h_1, h_2$, and so on.
Hint 2: remember that $\sum_i D_{t+1}(i) = 1$.

3. We would like to minimize test set error, but it is hard to do so directly. We thus settle for greedily optimizing the upper bound on the error. You have showed that the error is bounded from above:
$$\epsilon_{training} \leq \prod_{t=1}^{T} Z_t.$$

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 Z_t from Equation 1 is minimized by picking α_t as:

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

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

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

For this proof, you will restrict yourself to a special class of weak classifiers: $h_t(x)$ will return exactly $+1$, if h_t thinks the example x is positive, and exactly -1 if h_t is classifying x as negative.

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 2.

1.2 [30 Points] Implementation

Implement the Adaboost algorithm with Naive Bayes as weak classifier.

Don't forget to weigh the training data when learning a weak classifier! For example, you may have the following dataset:

D_t	x	y
0.6	a	1
0.3	b	-1
0.1	a	-1

then Naive Bayes would learn the following values for $P(x|y=-1)$:

$$P(x=a|y=-1) = \frac{P(x=a|y=-1)}{P(y=-1)} = \frac{\sum_{i:x_i=a,y_i=-1} D_t(i)}{\sum_{i:y_i=-1} D_t(i)} = \frac{0.1}{0.1+0.3} = 0.25$$

$$P(x=b|y=-1) = \frac{P(x=b|y=-1)}{P(y=-1)} = \frac{\sum_{i:x_i=b,y_i=-1} D_t(i)}{\sum_{i:y_i=-1} D_t(i)} = \frac{0.3}{0.1+0.3} = 0.75$$

1. Give the pseudocode of your implementation. Submit the actual code to

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

2. Run your implementation on the voting dataset that you can download from course webpage. Use 2/3 of the dataset for training and 1/3 for testing. Average your results over 50 random test/training splits of the data. Limit the number of boosting iterations to 100. On the same figure (with X axis being the number of rounds of boosting) plot 3 curves: training set error, test set error, and the upper bound on training set error that you just have proved:

   $$\epsilon_{training} \leq \prod_{t=1}^{T} Z_t.$$

   Does training set error approach 0 for this dataset or does it get "stuck" above 0? Why?

2 [30+5 Points] Cross-validation [Stano]

In this question, we will look at a few interesting properties of the leave-one-out cross validation (LOOCV) error. Let $\mathcal{D} = \{(x_i, y_i), i = 1, 2, \ldots, N\}$ be a dataset of size N. Leave-one-out cross validation approximates the test error by comparing each output y_i to the prediction of the classifier if we leave out the data point (x_i, y_i) from the training for this output. Formally, let $h_{\mathcal{D}\setminus i}$ denote the classifier trained on data other than i and $h_{\mathcal{D}\setminus i}(x)$ denote the prediction of this classifier on input x. The LOOCV error is then

$$error_{LOO}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - h_{\mathcal{D}\setminus i}(x_i))^2. \tag{3}$$

2.1 [10 Points] LOOCV is unbiased

Let $\mathcal{D}'$ denote a dataset of size $N-1$. Show that

$$E[error_{LOO}(\mathcal{D})] = E[(Y - h_{\mathcal{D}'}(X))^2]. \tag{4}$$

Hint: Use the linearity of expectation, e.g., $E[A+B] = E[A] + E[B]$.

Therefore, we see that the LOOCV is *almost* unbiased: it is an unbiased estimator of the test error on a smaller dataset of size $N-1$. In particular, the LOOCV error will tend to be slightly pessimistic about the error of our estimator h (assuming that the estimator improves with more data and that we are not at a phase transition where an additional data point would make a big difference).

2.2 [20+5 Points] LOOCV is easy

An attractive property of the leave-one-out cross-validation error is that it can often be computed efficiently. Consider the task of computing the LOOCV error in linear regression. Recall that in linear regression, the goal is to approximate a function f using a vector of basis functions $\phi = [\phi_1,\ \phi_2,\ \ldots,\ \phi_K]^{\mathrm{T}}$ as

$$f(x) = \sum_j \phi_j(x) w_j = \phi(x)^{\mathrm{T}}\mathbf{w}, \tag{5}$$

where $\mathbf{w} \in \mathbb{R}^K$ are the weights we wish to learn. For example, with $\phi_i(x) = x^i$, this approach allows us to represent any polynomial of degree K that passes through the origin. As shown in the lecture, the optimal weights $\mathbf{w}^*$ are

$$\mathbf{w}^* = (\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{Y}, \tag{6}$$

where $\mathbf{H}$ is the data matrix

$$\mathbf{H} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_K(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_K(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_K(x_N) \end{bmatrix} = \begin{bmatrix} — \ \mathbf{h}_1^{\mathrm{T}} \ — \\ — \ \mathbf{h}_2^{\mathrm{T}} \ — \\ \vdots \\ — \ \mathbf{h}_N^{\mathrm{T}} \ — \end{bmatrix}, \tag{7}$$

and $\mathbf{Y} = [y_1,\ y_2,\ \ldots,\ y_N]^{\mathrm{T}}$ is the column vector of sample outputs. Each row $\mathbf{h}_i^{\mathrm{T}} = \phi(x_i)^{\mathrm{T}}$ in $\mathbf{H}$ is a vector of features for the i-th data point ($\mathbf{h}_i^{\mathrm{T}}$ is a row vector; $\mathbf{h}_i$ is a column vector). The optimal prediction at a new input x is then

$$\hat{y}(x) = \phi(x)^{\mathrm{T}}\mathbf{w}^* = \phi(x)^{\mathrm{T}}(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{Y}. \tag{8}$$

1. When computing the LOOCV error for linear regression, removing a datapoint i from the dataset corresponds to removing the i-th row from the data matrix $\mathbf{H}$ and the observation vector $\mathbf{Y}$. Using the notation $\mathbf{H}_{-i}$ and $\mathbf{Y}_{-i}$ to denote the resulting matrices (after we removed the i-th row from each matrix), write down a formula for the LOOCV error. What is the asymptotic computational complexity of evaluating your formula, assuming that the complexity of inverting an $M \times M$ matrix is $O(M^3)$?

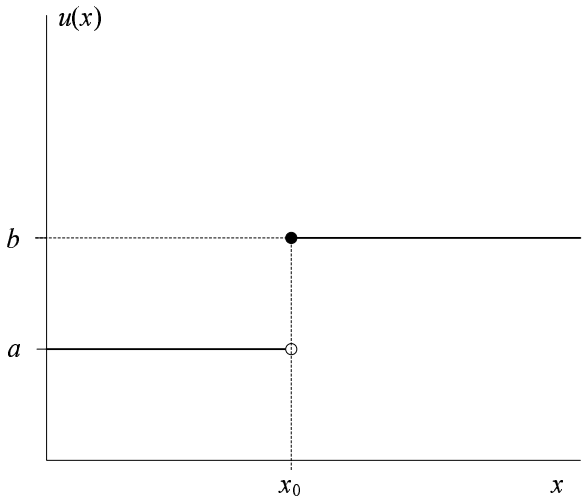

Figure 1: Step function in Question 3.

2. It turns out that for linear regression, the expression for the LOOCV error simplifies to

$$error_{LOO} = \frac{1}{N}\sum_{i}^{N}\left(\frac{y_i - \mathbf{h}_i^{\mathrm{T}}\mathbf{w}^*}{1 - \mathbf{h}_i^{\mathrm{T}}(\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{h}_i}\right)^2. \tag{9}$$

In other words, we do not need to recompute the optimal weights for each $\mathcal{D} \setminus i$. Instead we only need to compute optimal weights $\mathbf{w}^*$ based on all the data.

(a) First, prove Equation 9 when ϕ is a single basis function $\phi_1(x) = x, K = 1$.
Hint: show that the summands in Equations 3 and 9 are the same, that is,

$$(y_i - h_{\mathcal{D}\setminus i}(x_i))^2 = \left(\frac{y_i - \mathbf{h}_i^T\mathbf{w}^*}{1 - \mathbf{h}_i^T(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{h}_i}\right)^2,$$

where $h_{\mathcal{D}\setminus i}(x_i)$ denotes the prediction of the estimator trained with $\mathcal{D} \setminus i$ on the i-th input.

(b) **[5 Points extra credit]** How is $\mathbf{H}^{\mathrm{T}}\mathbf{Y}$ related to $\mathbf{H}_{-i}$, $\mathbf{h}_i$, $\mathbf{Y}_{-i}$, and y_i? Write down a formula and show why it is true. How about $\mathbf{H}^{\mathrm{T}}\mathbf{H}$, $\mathbf{H}_{-i}$, and $\mathbf{h}_i$? Now prove Equation 9 for an arbitrary number of basis functions.
Hint: you may find the Sherman-Morrison formula (also known as the Matrix Inversion Lemma) useful here. For an arbitrary invertible matrix A and suitable column vectors $\mathbf{u}$ and $\mathbf{v}$,

$$(A + \mathbf{u}\mathbf{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^T A^{-1}}{1 + \mathbf{v}^T A^{-1}\mathbf{u}}.$$

3. What is the computational complexity of evaluating the formula in Equation 9?

3 [20+15 Points] Neural Networks [Stano]

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. This question consists of two parts; the second part (3.2) is optional.

3.1 [20 Points] Approximating with hard thresholds

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

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

$$y = \begin{cases} 1 & \text{if } w_0 + \sum_i w_i x_i \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

1. Consider the step function $u(x)$ in Figure 1. 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, b, and x_0.

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

$$\forall x, x' \in [C; D), \quad |f(x') - f(x)| \leq L|x' - x|, \tag{11}$$

   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 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 attains the given accuracy ϵ?

   Hint: think of Riemann integrals.

3.2 [15 Points, extra credit] Approximating with soft thresholds

Threshold activation functions are rarely used in practice, since they are not differentiable, and thus difficult to learn. In the lecture, we discuss the sigmoid activation function

$$y = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}},$$

which provides a smooth approximation to the hard threshold in Equation 10. In both questions below, you need to prove that the neural networks you construct attain the given accuracy ϵ.

1. Prove that a sigmoid cannot approximate the following simple threshold function:

$$y = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{otherwise.} \end{cases}$$

   in the interval $[-1, 1]$ within any $\epsilon < 0.5$ in max-norm, where the max-norm is defined as $\forall x \in [-1, 1]$, $| f(x) - \hat{f}(x) | \leq \epsilon$.

2. Suppose that you are allowed to use linear and sigmoid activation functions (but not hard threshold functions). Approximate the step function in Figure 1 within $\epsilon > 0$ for x arbitrarily close to x_0. That is, given $\delta > 0$ and $\epsilon > 0$, construct a neural network with output $\hat{f}(x)$, such that $|f(x) - \hat{f}(x)| \leq \epsilon$ for $x \leq x_0 - \delta$ or $x \geq x_0 + \delta$. Draw the structure of your network and specify the parameters as a function of a, b, x_0, ϵ, and δ.

3. Based on your solutions to the previous two parts, construct a two-layer neural network that approximates any Lipschitz continuous function on $[C; D)$ within $\epsilon > 0$, using only linear and sigmoid activation functions. As before, the structure and parameters should be specified in terms of C, D, L, ϵ, and $f(x)$ evaluated at a finite number of points.