10-701/15-781 Machine Learning, Fall 2007: Homework 2

Due: Wednesday, October 17th, beginning of the class

Instructions There are 4 questions on this assignment. The last 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`

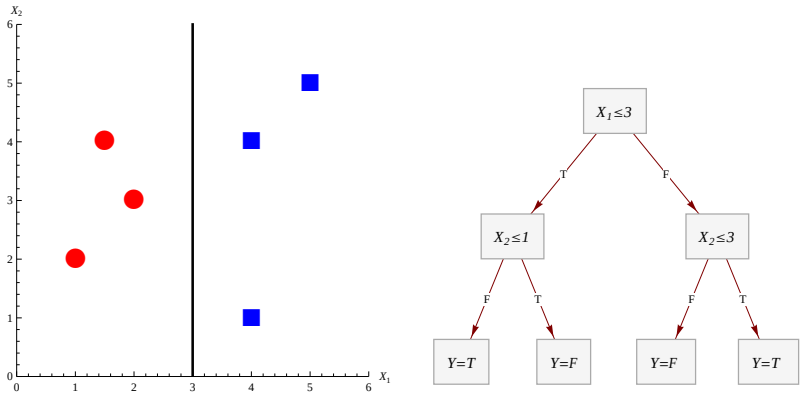
To write in this directory, you need a kerberos instance for andrew, or you can log into, for example, unix.andrew.cmu.edu. Please submit each problem *seperately* with your name and userid on each problem. Refer to the webpage for policies regarding collaboration, due dates, and extensions.

1 Decision Surfaces [Joseph, 15 points]

Your favorite billionaire wants to hire a team to work on a new operating system called Goosoft OSXSun. He wants to classify job applicants into good programmers (worth hiring) and bad programmers based on their 10701 Grade ($X_1 \in [0, 5]$) and top score in Minesweeper ($X_2 \in [0, 5]$). He wants to use a classifier, but cannot choose which one, so he needs the help of a machine learning expert. We happily offer your help.

1.1 Single-node Decision Tree As Linear Classifier [2 points]

When the features are continuous, a decision tree with one node (a depth 1 decision tree) can be viewed as a linear classifier. These degenerate trees, consisting of only one node and therefore using only one variable to split the data, are called *decision stumps*). See Figure 1 part A for an example of a decision stump which completely separates the data by splitting on $X_1 \leq 3$. What is the difference between the decision boundary for this classifier and other linear classifiers (i.e., logistic regression) you have learned in class?



(A) (B)

Figure 1: (A) A single decision stump which completely seperates the data by splitting at $X_1 \leq 3$. (B) An example decision tree. The decision tree in (B) does not correspond to the decision surface obtained by the single decision stump drawn on the left. You should understand why.

Sketch or plot a small data set of programmers (no more than 10 2-dimensional points), which is completely separable using a linear classifier but not completely separable using decision stumps. Include in your sketch a linear classifier that completely separates the data. Briefly explain why you cannot construct a decision stump which completely seperates the data you provided.

1.2 General Decision Trees vs. Linear Classifiers [3 points]

Decision trees of arbitrary depth, can capture more complex decision surfaces. Sketch or plot a small 2D data set which is completely separable using decision trees of arbitrary depth (using decision stumps at each node) but cannot be completely separated using a single linear classifier. Include in your sketch the decision surface obtained by the decision tree. Explain why no linear classifier can completely separate the data. Also, draw the decision tree annotating each node with the decision rule and each edge with True or False (indicating the outcome of the decision rule). For an example of how to draw such a decision tree look at Figure 1 part B.

1.3 Using Linear Classifiers In Decision Trees [5 points]

Now that we see 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 completely seperates the training set.

Sketch or plot a data set, which is not completely separable by a linear classifier, and your algorithm outputs a decision tree with depth smaller than the depth of a decision tree using simple decision stumps. Sketch the linear classifiers obtained by your algorithm.

1.4 K-Nearest Neighbors [5 points]

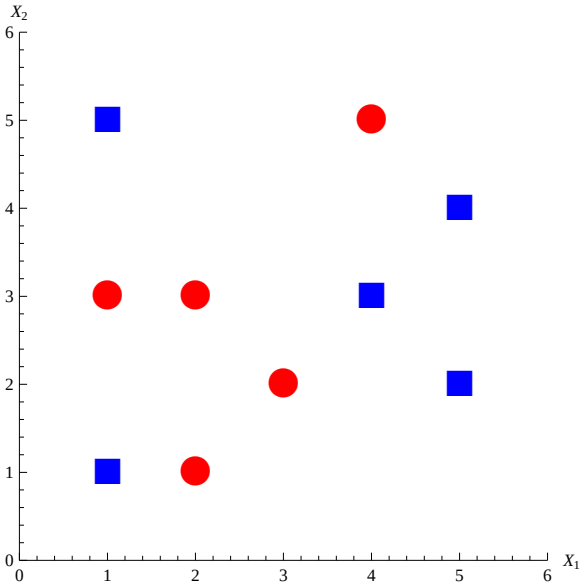


Figure 2: Plot of the Nearest Neighbors data. Blue square programmers are are true (good) and red circle programmers are false (bad).

Decision trees (without constraints on the number of nodes) are an example of a nonparametric method. Reflect on this for a moment (Approximately 1 minute). Another common nonparametric method is the K-Nearest Neighbor algorithm. In this problem you will have the opportunity to compare the complexity of the decision surface obtained using $K = 1$ and $K = 3$.

For the developers (data) in Figure 2, roughly sketch the decision surface obtained by applying the K-Nearest Neighbors Algorithm with $K = 1$ and $K = 3$. Feel free to write a program to automatically generate the plots. What can you say about the complexity of the decision surfaces and the tendency of the K-Nearest Neighbors algorithm to *overfit* as a function of K.

2 Boosting [Jingrui, 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. Given a set of m examples, (x_i, y_i) (y_i is the class label of x_i), $i = 1, \ldots, m$, 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 error of the final classifier can be bounded from above by an exponential loss function:

$$\frac{1}{m} \sum_{i=1}^{m} I(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^{m} \exp(-f(x_i) y_i),$$

   where $I(a = b)$ is the indicator function. It is equal to 1 if $a = b$, and 0 otherwise

   Hint: $e^{-x} \geq 1 \Leftrightarrow x \leq 0$.

2. Remember that

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

   Start with 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 shown 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 the training error bound on round t is to minimize Z_t.

In this question, you will prove that for binary weak 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 error of weak classifier h_t for the 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 output 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

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

Furthermore, let $\epsilon_t = \frac{1}{2} - \gamma_t$, prove that

$$Z_t \leq \exp(-2\gamma_t^2)$$

Hint: $\log(1 - x) \leq -x$ *for* $0 < x \leq 1$.

Therefore

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

Finally prove that, if each weak 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, i.e.

$$\epsilon_{training} \leq \exp(-2T\gamma^2)$$

5. Show that in each round of boosting, there always exists a weak classifier h_t such that its training error on the weighted dataset $\epsilon_t \leq 0.5$. Also show that for $\epsilon_t = 0.5$ the training error can get "stuck" above zero.

Hint: $D_t(i)$s do not change over t.

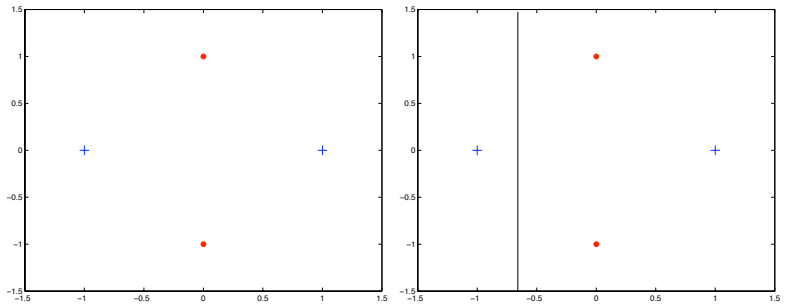


Figure 3: a) Toy data in Question 2. b) h_1 in Question 2

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 3a). 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 classifiers. (For description of decision stumps, refer to Problem 2.3) Now for $T = 4$, show how Adaboost works for this dataset. For each timestep remember to compute the following numbers:
$$\epsilon_t, \alpha_t, Z_t, D_t(i) \ \forall i,$$
Also for each timestep draw your weak classifier. For example h_1 can be as shown in 3b).

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 $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, m$, but with the significance $D_t(i)$ of each example $\{\mathbf{x}_i, y_i\}$ weighted differently. At each iteration, a classifier, $h_t(\mathbf{x}) \rightarrow \{-1, 1\}$, is trained to minimize the weighted classification error, $\sum_{i=1}^{m} D_t(i) \cdot I(h_t(\mathbf{x}_i) \neq y_i)$, where I is the indicator function (0 if the predicted and actual labels match, and 1 otherwise). The overall prediction of the AdaBoost algorithm is a linear combination of these classifiers, $H_T(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$. *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 features, and predicts the class for examples falling above and below the threshold respectively, $h_t(\mathbf{x}) = C_1 I(x^j \geq c) + C_2 I(x^j < c)$, where x^j is the j^{th} component of the feature vector $\mathbf{x}$. 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 $C_1, C_2 \in \{-1, 1\}$, threshold c, 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 training sets and test sets. Limit the number of boosting iterations to 100. In a single plot show:

- average training error after each boosting iteration
- average test error after each boosting iteration

3. Using all of the data for training, display the selected feature component j, threshold c, and class label C_1 of the decision stump $h_t(\mathbf{x})$ used in each of the first 10 boosting iterations ($t = 1, 2, ..., 10$)

4. Using all of the data for training, in a single plot, show the empirical cumulative distribution functions of the margins $y_i f_T(\mathbf{x}_i)$ after 10, 50 and 100 iterations respectively, where $f_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$. Notice that in this problem, before calculating $f_T(\mathbf{x})$, you should normalize the α_ts so that $\sum_{t=1}^{T} \alpha_t = 1$. This is to ensure that the margins are between -1 and 1.

   hint: the empirical cumulative distribution function of a random variable X at x is the proportion of times $X \leq x$.

3 Linear Regression and LOOCV [Sue Ann, 20 points]

In class, you learned about using cross validation as a way to estimate the true error of a learning algorithm. A solution that provides an almost unbiased estimate of this true error is *Leave-One-Out Cross Validation* (LOOCV), but it can take a really long time to compute the LOOCV error. In this problem, you will derive an algorithm for efficiently computing the LOOCV error for linear regression using the *Hat Matrix*. [1] (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 data points as the rows of a matrix X so that x_{ij} is the j-th feature of the i-th data point. 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:

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

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:

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

[1] Unfortunately, such an efficient algorithm may not be easily found for other learning methods.

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 time 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].
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. *(1 point)* Write $\hat{Y}_i^{(-i)}$ in terms of H and Z. By definition, $\hat{Y}_i^{(-i)} = Z_i$, but give an answer that includes both H and Z.
5. *(4 points)* Show that $\hat{Y}_i - \hat{Y}_i^{(-i)} = H_{ii} Y_i - H_{ii} \hat{Y}_i^{(-i)}$, where H_{ii} denotes the i-th element along the diagonal of H.
6. *(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.

4 Neural Networks [Steve, 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, and that such a network can be learned from i.i.d. data.

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

- identity

$$g_I(x) = x,$$

- step function

$$g_s(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

So, for example, the output of a neural network with one input x, a single hidden layer with K units having step function activations, and a single output with identity activation can be written as $out(x) = g_I(w_0 + \sum_{k=1}^{K} w_k g_s(w_0^{(k)} + w_1^{(k)} x))$, and can be drawn as in Figure 4.

4.1 Representation

1. *(2 points)* Consider the step function $u(x)$ in Figure 5. Construct a neural network with one input x and one hidden layer whose response is $u(x)$. That is, if $x < a$, the output of your network should be 0, whereas if $x \geq a$, the output should be y. Draw the structure of the neural network, specify the activation function for each unit (either g_I or g_s), and specify the values for all weights (in terms of a and y).

[2] There are faster algorithms out there but for simplicity we'll assume that we are using the naive $O(k^3)$ algorithm.

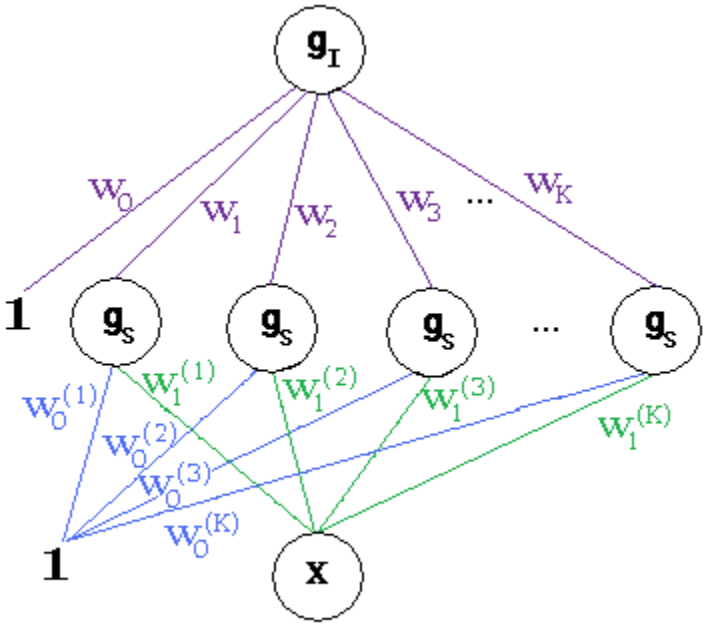


Figure 4: Neural net drawing in Question 4.

2. *(2 points)* Now consider the indicator function $\mathbb{1}_{[a,b)}(x)$:

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

Construct a neural network with one input x and one hidden layer whose response is $y\mathbb{1}_{[a,b)}(x)$, for given real values y, a, and b; that is, its output is y if $x \in [a,b)$, and 0 otherwise. Draw the structure of the neural network, specify the activation function for each unit (either g_I or g_s), and specify the values for all weights (in terms of a, b, and y).

3. *(6 points)* You are now given any function $f(x)$ whose domain is $[C,D)$, for real values $C < D$. Suppose that the function is Lipschitz continuous[3]; that is,

$$\forall x, x' \in [C,D), \quad |f(x') - f(x)| \leq L|x' - x|, \tag{5}$$

for some constant $L \geq 0$ (called the Lipschitz constant of f). 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, $\forall x \in [C,D), \quad |f(x) - out(x)| \leq \epsilon$, where $out(x)$ is the output of your neural network given input x. Your network should use only the activation functions g_I and g_s given above. You need to specify the number K of hidden units, the activation function for each unit, and a formula for calculating each weight w_0, w_k, $w_0^{(k)}$, and $w_1^{(k)}$, for each $k \in \{1, 2, \ldots, K\}$; these may be specified in terms of C, D, L, and ϵ, as well as the values of $f(x)$ evaluated at a finite number of x values of your choosing (please explicitly specify which x values you use). You do *not* need to draw the network or explicitly write the $out(x)$ function. Why does your neural network attain the given accuracy ϵ?

Hint: Lipschitz continuous means that points close together have similar f values. Can you find a set of points so that every $x \in [C,D)$ is close to at least one of them? How close does it need to be to guarantee the f values are within ϵ of each other?

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

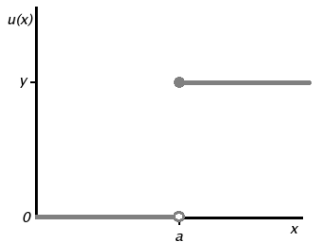


Figure 5: Step function in Question 4.

4.2 Universal Learning

Suppose there is a distribution $\mathcal{D}_X$ with domain $[C, D)$, and $X_1, X_2, \ldots, X_n$ are i.i.d. random variables, with distribution $\mathcal{D}_X$. In this part, rather than direct access to $f(x)$ at x values of your choosing, you are given the sequence of labeled examples

$$\mathcal{L}_n = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}, \text{ where } Y_i = f(X_i).$$

In this setting, suppose we are interested in achieving a small value of $\mathbb{E}_X[|f(X) - out(X)|]$, where X is a random variable with distribution $\mathcal{D}_X$, independent of $X_1, X_2, \ldots, X_n$, and $out(x)$ is the output of a neural network.[4] In this problem, you will design a learning algorithm that outputs a neural network $out_{\mathcal{L}_n}(x)$ with the property that for any $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}_{\mathcal{L}_n}\{\mathbb{E}_X[|f(X) - out_{\mathcal{L}_n}(X)|] > \epsilon\} = 0.$$

In other words, the algorithm will achieve an arbitrarily good expected approximation to any smooth function if we give it enough data.

1. *(2 points)* Given the data points $\mathcal{L}_n = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$, construct a neural network $out_{\mathcal{L}_n}(x)$ such that for any $x \in [C, D)$,

   $$out_{\mathcal{L}_n}(x) = Y_{j_x}, \text{ where } j_x = \arg\min_{i\in\{1,2,\ldots,n\}} |x - X_i|.$$

   That is, $out_{\mathcal{L}_n}(x)$ labels x according to the Y_i value of the closest X_i in $\mathcal{L}_n$.[5]

   Your network should have one input layer, one hidden layer, and one output layer, and should use only the activation functions g_I and g_s given above. You need to specify the number K of hidden units, the activation function for each unit, and a formula for calculating each weight w_0, w_k, $w_0^{(k)}$, and $w_1^{(k)}$ for each $k \in \{1, 2, \ldots, K\}$; these should be specified in terms of n, the training data $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$, and nothing else. In particular, unlike the previous part, you are not allowed to use the values $f(x)$ at arbitrary points x of your choosing: only at the data points $X_1, X_2, \ldots, X_n$ given to you.

2. *(2 point)* Next, use the Lipschitz smoothness of f to show that

   $$\mathbb{E}_X[|f(X) - out_{\mathcal{L}_n}(X)|] \le L\mathbb{E}_X\left[\min_{i\in\{1,2,\ldots,n\}} |X - X_i|\right]$$

   (note that the expectation is only over X, not the X_i variables), and argue that this implies

   $$\forall \epsilon > 0, \quad \mathbb{P}_{\mathcal{L}_n}\{\mathbb{E}_X[|f(X) - out_{\mathcal{L}_n}(X)|] > \epsilon\} \le \mathbb{P}_{\mathcal{L}_n}\left\{\mathbb{E}_X\left[\min_{i\in\{1,2,\ldots,n\}} |X - X_i|\right] > \frac{\epsilon}{L}\right\}.$$

[4] We use the following notation: $\mathbb{E}_X$ denotes the expectation with respect to X, (e.g., $\mathbb{E}_X[X]$ is the mean of X); $\mathbb{P}_{\mathcal{L}_n}$ denotes the probability with respect to the i.i.d. sample $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$, (e.g., $\mathbb{P}_{\mathcal{L}_n}\{\max_{i\in\{1,2,\ldots,n\}} X_i \le z\}$ denotes the probability the largest training example is at most z.

[5] You may break ties in the arg min however you like.

3. *(1 point)* In this part, you may take the following observation as given.

$$\forall \epsilon > 0, \quad \mathbb{P}_{\mathcal{L}_n}\left\{\mathbb{E}_X\left[\min_{i\in\{1,2,\ldots,n\}} |X - X_i|\right] > \frac{\epsilon}{L}\right\} \leq \alpha_\epsilon e^{-n\beta_\epsilon}, \tag{6}$$

where α_ϵ and β_ϵ are finite positive constants that depend only on ϵ, L, C, D, and $\mathcal{D}_X$.

Use this to prove that for any $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}_{\mathcal{L}_n}\left\{\mathbb{E}_X\left[|f(X) - out_{\mathcal{L}_n}(X)|\right] > \epsilon\right\} = 0.$$

4. (BONUS [+5 pts extra credit]) Prove the observation in Formula (6) from the previous part. You can prove it however you like, but if you want, you can proceed as follows.

   - First pick an explicit set of points $\{z_1, z_2, \ldots, z_m\}$ such that every point $x \in [C, D)$ has

   $$\min_{j\in\{1,2,\ldots,m\}} |x - z_j| \leq \frac{\epsilon}{2L}.$$

   - Then you can let $J = \{j \in \{1, 2, \ldots, m\} : \mathbb{P}_X\left\{|X - z_j| \leq \frac{\epsilon}{2L}\right\} > 0\}$, and prove that

   $$\mathbb{E}_X\left[\min_{i\in\{1,2,\ldots,n\}} |X - X_i|\right] \leq \mathbb{E}_X\left[\min_{j\in\{1,2,\ldots,m\}} |z_j - X|\right] + \max_{j\in J} \min_{i\in\{1,2,\ldots,n\}} |z_j - X_i|.$$

   In particular, because by construction $\mathbb{E}_X\left[\min_{j\in\{1,2,\ldots,m\}} |z_j - X|\right] \leq \frac{\epsilon}{2L}$, this means

   $$\mathbb{P}_{\mathcal{L}_n}\left\{\mathbb{E}_X\left[\min_{i\in\{1,2,\ldots,n\}} |X - X_i|\right] > \frac{\epsilon}{L}\right\} \leq \mathbb{P}_{\mathcal{L}_n}\left\{\max_{j\in J} \min_{i\in\{1,2,\ldots,n\}} |z_j - X_i| > \frac{\epsilon}{2L}\right\}.$$

   - Next, show that for any $z \in [C, D)$, (and in particular, for any z_j)

   $$\mathbb{P}_{\mathcal{L}_n}\left\{\min_{i\in\{1,2,\ldots,n\}} |z - X_i| > \frac{\epsilon}{2L}\right\} \leq \left(1 - \mathbb{P}_X\left\{|z - X| \leq \frac{\epsilon}{2L}\right\}\right)^n.$$

   (Hint: first try proving it for $n = 1$; then recall that for any independent Bernoulli random variables A and B, we know that $\mathbb{P}(A = 1 \wedge B = 1) = \mathbb{P}(A = 1)\mathbb{P}(B = 1)$.)

   - Next, use the fact that for any $p \in [0, 1]$, $1 - p \leq e^{-p}$.
   - Finally, notice that for any two Bernoulli random variables A and B (not necessarily independent), $\mathbb{P}(A = 1 \vee B = 1) \leq \mathbb{P}(A = 1) + \mathbb{P}(B = 1)$. This should allow you to bound $\mathbb{P}_{\mathcal{L}_n}\left\{\max_{j\in J} \min_{i\in\{1,2,\ldots,n\}} |z_j - X_i| > \frac{\epsilon}{2L}\right\}$ in terms of a sum over $|J|$ terms.
   - Using the fact that this sum is at most $|J|$ times its largest term should allow you to obtain the bound, with $\beta_\epsilon = \min_{j\in J} \mathbb{P}_X\left\{|z_j - X| \leq \frac{\epsilon}{2L}\right\}$ and $\alpha_\epsilon = |J| \leq m$.

5. (Optional Exercise [0 pts]) [Warning: This problem could take some extra effort]
   Think about how the learning problem might change if we add zero-mean independent random noise to the labels $Y_i = f(X_i) + \epsilon_i$, where $\epsilon_i \sim N(0, \sigma^2)$, with $\sigma^2 < \infty$.[6] How could you modify your learning algorithm to still guarantee

   $$\lim_{n\to\infty} \mathbb{P}_{\mathcal{L}_n}\left\{\mathbb{E}_X\left[|f(X) - out(X)|\right] > \epsilon\right\} = 0?$$

   Hint: Instead of just one X_i close to X, can we expect k_n values $\{X_{i_1}, X_{i_2}, \ldots, X_{i_{k_n}}\}$ close to X, for some number k_n that depends on n?

   Note: Being able to solve this problem is not necessary for this class. But it's a fun exercise to think about in your spare time, if you're interested.

[6] In fact, it is possible to modify the algorithm to guarantee this even for non-Gaussian independent noise, $Y_i = f(X_i) + \epsilon_i(X_i)$, as long as $\forall x, \mathbb{E}[\epsilon_i(x)] = 0$ and $\mathbb{E}[Y^2] < \infty$.