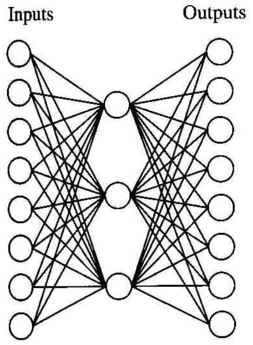
CS 6140: Machine Learning **Spring 2018**
College of Computer and Information Science
Northeastern University
Assignment 7 **April, 20**
Instructor: Bilal Ahmed **Due Date: April, 26**

Neural Networks and Boosting

1 Autoencoder Neural Network

Consider the following neural network (left graph), with 8 input units (for data with 8 features), 3 hidden units and 8 output units, and assume the nonlinear functions are all sigmoid.



Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Figure 1: An auto encoder neural network, all the hidden and output units use the sigmoid activation function. (*Taken from: TM*)

1. The 8 training data inputs are identical with the outputs, as shown in the right side table. Implement this network and the backpropagation algorithm to compute all the network weights; you should initialize the weights with nontrivial values (i.e not values that already minimize the erorr as shown in Figure-1). *HINT: on the trained network, you should obtain values for the hidden units somehow similar with the ones shown in the table (up to symmetry). Feel free to make changes to the algorithm that suit your implementation, but briefly document them.*

2. Since the outputs and inputs are identical for each datapoint, one can view this network as an encoder-decoder mechanism. (this is one of the uses of neural networks). In this context, explain the purpose of the training algorithm. (I expect a rather nontechnical –but well-explained– answer).

3. Train the same network with different number of hidden units $\in \{1, 2, 4\}$, and explain the effects of changing the size of the hidden layer on the training time, convergence of backpropagation, etc. With these changes can we still think of the network as an encoder/decoder?

4. Consider that the output units for the neural network shown in Figure-1 are linear units. What changes should be made to the backpropagation algorithm to train the network. Provide the mathematical details needed to change the algorithm and generalize it to training an arbitrary network whose hidden units are sigmoid while the output units are linear (i.e., their output is $\sum_{k \in output} w_{jk} x_{jk}$).

2 Boosting

Algorithm-1 lists a generic boosting algorithm. Boosting relies on training a weak (base) classifier on weighted data. In general, boosting provides a clean interface between boosting and the underlying weak learning algorithm: in each round, the boosting algorithm provides a weighted data set to the weak learner, and the weak learner provides a predictor in return. You may choose to keep this clean interface (which would allow you to run boosting using any weak learning algorithm) or you may choose to more simply incorporate the weak learning algorithm inside your boosting code. As boosting remains agnostic to the weak learner, it is known as a *meta-learning* algorithm.

The weighting scheme (how α is defined in Algorithm-1) can be varied to modify the overall boosting algorithms. In this question you will implement AdaBoost which chooses α as:

$$\alpha_t = \ln \frac{1+\gamma_t}{1-\gamma_t}$$

where γ is the so-called *edge* of $h_t(x)$, and defined as:

$$\gamma_t = \sum_{i=1}^{N} D_t(i) y_i h_t(x_i)$$

```
**Data:** Training data: $(x_1, y_1), \ldots, (x_N, y_N)$, where $x_i \in \mathbb{R}^m$ and $y_i \in \{-1, 1\}$
**Result:** Boosted Classifier: $H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$
Initialize $D_1(i) \leftarrow \frac{1}{N}$;
**for** *t=1 to T* **do**
    Train base classifier using distribution $D_t$;
    Get base classifier, $h_t : X \rightarrow \{-1, 1\}$;
    Choose $\alpha_t \in \mathbb{R}$;
    Update: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, where $Z_t$ is chosen so that $\sum_i D_{t+1}(i) = 1$;
**end**
```

Algorithm 1: A generic algorithm for boosting.

2.1 Decision Stumps

Each predictor will correspond to a decision stump, which is just a feature-threshold pair. Note that for each feature f_i, you may have many possible thresholds which we shall denote as τ. Given an input instance to classify, a decision stump corresponding to feature f_i and threshold τ will predict $+1$ if the input instance has a feature f_i value greater than the threshold τ; otherwise, it predicts -1.

Determining suitable threshold: To create the various thresholds for each feature f_i, you should:

1. sort the training examples by their f_i values
2. remove duplicate values
3. construct thesholds that are midway between successive feature values

You should also add two thresholds for each feature: one below all observed values for that feature and one above all values for that feature. By removing duplicate values we are trying to cut down the number of thresold we need to check. (*Note:* The search procedure is the same as used for selecting a threshold for defining a binary split on a contiuous feature in decision trees, except we are maximizing the training error instead of information gain or Gini index.)

Optimal Decision Stump: The optimal threshold τ^* for feature f_i corresponds to the threshold that maximizes:

$$\tau^* = \underset{\tau}{\operatorname{argmax}} \, |\frac{1}{2} - error(\tau)|$$

where, $error(.)$ is calculated as the sum of the weights ($D_t(i)$) of the misclassified instances. In this case since we are dealing with absolute values if a threshold has an error of 0.2 and another threshold produces an error of 0.9 we will prefer the latter one since it has a higher difference from 0.5 (for such decision stumps you would need to invert the prediction so that for instances with a feature value greater than τ you will predict -1 and $+1$ otherwise. You would need to remember this when you combine such decision stumps with others to evaluate $H(x)$).

2.2 Programming Tasks

1. Implement the AdaBoost algorithm, using optimal decision stumps.
2. Implement the AdaBoost algorithm, using random decision stumps (see below).

Random decision stumps on the other hand are completely random, where you will select the feature and threshold randomly.
Run your AdaBoost implementation on the Spambase, Diabetes and Breast Cancer datasets. For each dataset partition it into a training set (80%) and a test set (20%) (no ten fold cross-validation). After each round, you should compute:

1. The local "round" error for the decision stump returned, i.e., the error using h_t (the new weak learner being added to the ensemble) measured using D_t
2. The training error for the linear combination of decision stumps, i.e., the error using H_t (current ensemble i.e., prediction of the linear combination of all t weak learners) measured on training data (ignoring D_t)
3. The test error for the linear combination of decision stumps, i.e., the error using H_t measured on test data

2.3 Deliverables:

1. *Boosting with optimal decision stumps:* For each dataset provide three plots that show the local error, training error and test error.

2. *Boosting with random decision stumps:* For each dataset provide three plots that show the local error, training error and test error.

3. In both cases how did you determine that boosting has converged i.e., how did you set the parameter T in Algorithm-1.