Homework #4

ECE 461/661: Introduction to Machine Learning
Prof. Carlee Joe-Wong and Prof. Gauri Joshi
Due: Wednesday March 18th, 2020 at 8:59PM PT / 11:59PM ET

Please remember to show your work for all problems and to write down the names of any students that you collaborate with. The full collaboration and grading policies are available on the course website: https://www.andrew.cmu.edu/course/18-661/.

Your solutions should be uploaded to Gradescope (https://www.gradescope.com/) in PDF format by the deadline. We will not accept hardcopies. If you choose to hand-write your solutions, please make sure the uploaded copies are legible. Gradescope will ask you to identify which page(s) contain your solutions to which problems, so make sure you leave enough time to finish this before the deadline. We will give you a 30-minute grace period to upload your solutions in case of technical problems.

1 k-Nearest Neighbors Decision Boundary *[15 points]*

For some applications, it may make sense to perform k-nearest neighbors with respect to a distance other than the usual Euclidean distance. In this problem, we will look at the k-nearest neighbor problem when the distance between the points is the following modified form of the Euclidean distance. Given two vectors $v_1 = (x_1, y_1), v_2 = (x_2, y_2)$, the modified distance measure $d_M(v_1, v_2)$ is defined as:

$$d_M(v_1, v_2) = \sqrt{\frac{1}{2}(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Consider the following two labelled training datasets, each of which consists of three data points with two input features:

$$\mathcal{D}_1 = \{((0,0),1), ((2,2),2), ((4,0),3)\}$$
$$\mathcal{D}_2 = \{((0,0),1), ((1,1),1), ((-1,1),2)\}.$$

a. First consider the 1-nearest neighbor classifier on the data points in $\mathcal{D}_1$ with respect to the usual Euclidean distance. Draw the decision boundaries for the classifier in the (x_1, x_2) plane (note that there are three possible labels, $1, 2, 3$; and thus three possible classes). Write down the equations of the decision boundaries. Clearly mark each region in your drawing with the label assigned by the classifier to a test example in this region.

b. Now, consider the 1-nearest neighbor classifier on the data points in $\mathcal{D}_1$ with respect to the **modified** Euclidean distance d_M. In a separate figure, draw the decision boundaries for this classifier in the (x_1, x_2) plane. Write down the equations for the different segments of the decision boundary, and clearly mark each region in your drawing with the label assigned by the classifier to a test example in this region.

c. Repeat part (a), i.e. draw the decision boundary with respect to Euclidean distance in the (x_1, x_2) plane, for training data points in $\mathcal{D}_2$. Note that points in $\mathcal{D}_2$ have two possible labels (1 and 2).

d. Repeat part (b), i.e. draw the decision boundary with respect to the modified Euclidean distance in the (x_1, x_2) plane, for training data points in $\mathcal{D}_2$.

2 Entropy and KL Divergence *[10 points]*

When discussing decision trees, we introduced the concepts of entropy $H(X)$ and information gain $H(Y) - H(Y|X)$ for discrete random variables X, Y that take on values in a set $\mathcal{X}$ and a set $\mathcal{Y}$, respectively. In information theory, the information gain is also referred to as mutual information $I(X;Y)$. We can define:

$$H(X) = -\sum_{x \in \mathcal{X}} \Pr(X = x) \log_2 \Pr(X = x) \tag{1}$$

$$H(Y|X) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \Pr(X = x, Y = y) \log_2 \Pr(Y = y|X = x) \tag{2}$$

In this question, we will derive some useful properties of mutual information and relate it to the relative entropy. Using the above definitions, answer the following questions:

2.1 KL Divergence *[5 points]*

The Kullback–Leibler divergence $D(P(X)||Q(X))$ between two probability distributions $P(X)$ and $Q(X)$ over a discrete random variable X taking values in $\mathcal{X}$ is defined as:

$$D(P(X)||Q(X)) = \sum_{x \in \mathcal{X}} P(X = x) \log_2 \left(\frac{P(X = x)}{Q(X = x)} \right)$$

This quantity is also known as relative entropy and measures the similarity of the P and Q distributions.

a. Show that

$$D(P(X)||Q(X)) = H(P, Q) - H(P)$$

where we define $H(P, Q) = -\sum_{x \in \mathcal{X}} P(X = x) \log_2 Q(X = x)$, called the cross entropy between the distributions P and Q.

b. Prove that the KL-divergence is always non-negative (i.e., $D(P(X)||Q(X)) \geq 0$).

Hint: Jensen's inequality is a useful result that states that for any convex function $f(x)$, $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$. You may use this result without proving it.

2.2 Chain Rule *[5 points]*

Given the joint probability distributions $P(X = x, Y = y)$ and $Q(X = x, Y = y)$ of two discrete random variables X and Y, the conditional divergence between the two corresponding conditional probability distributions $P(Y = y|X = x)$ and $Q(Y = y|X = x)$ is obtained by computing the divergence between P and Q for all possible values of $x \in X$ and then averaging over these values of x. Formally it is defined as:

$$D(P(Y|X)||Q(Y|X)) = \sum_{x \in \mathcal{X}} P(X = x) \sum_{y \in \mathcal{Y}} P(Y = x|X = x) \log \frac{P(Y = y|X = x)}{Q(Y = y|X = x)},$$

where we define $P(X = x) = \sum_{y \in \mathcal{Y}} P(X = x, Y = y)$ as the probability distribution of X corresponding to the joint distribution $P(X = x, Y = y)$.

Show that

$$D(P(X, Y)||Q(X, Y)) = D(P(X)||Q(X)) + D(P(Y|X)||Q(Y|X))$$

where $P(X, Y), P(Y|X), P(X)$ and $Q(X, Y), Q(Y|X), Q(X)$ are the joint, conditional and marginal distributions induced by P and Q respectively.

3 Boosting *[25 points]*

We learned about boosting in lecture, and the topic is covered in Murphy 16.4. On page 555 Murphy claims that "it was proved that one could boost the performance (on the training set) of any weak learner arbitrarily high, provided the weak learner could always perform slightly better than chance." We will now verify this statement in the AdaBoost framework.

(1) *[5 points]* Given a set of N observations (x^j, y^j) where y^j is the label $y^j \in \{-1, 1\}$, let $h_t(x)$ be the weak classifier at step t and let β_t be its weight. First we note that the final classifier after T steps is defined as:

$$H(x) = \text{sgn}\left\{\sum_{t=1}^{T} \beta_t h_t(x)\right\} = \text{sgn}\{f(x)\},$$

where

$$f(x) = \sum_{t=1}^{T} \beta_t h_t(x).$$

Show that:

$$\epsilon_{\text{Training}} = \frac{1}{N}\sum_{j=1}^{N} 1_{\{H(x^j)\neq y^j\}} \leq \frac{1}{N}\sum_{j=1}^{N} \exp(-f(x^j)y^j),$$

where $1_{\{H(x^j)\neq y^j\}}$ is 1 if $H(x^j) \neq y^j$ and 0 otherwise.

(2) *[8 points]* The weight for each data point j at step $t+1$ can be defined recursively by:

$$w_j^{(t+1)} = \frac{w_j^{(t)} \exp(-\beta_t y^j h_t(x^j))}{Z_t},$$

where Z_t is a normalizing constant ensuring the weights sum to 1:

$$Z_t = \sum_{j=1}^{N} w_j^{(t)} \exp(-\beta_t y^j h_t(x^j)).$$

Show that:

$$\frac{1}{N}\sum_{j=1}^{N} \exp(-f(x^j)y^j) = \prod_{t=1}^{T} Z_t$$

(3) By combining your results to parts a and b, you have shown that the training error $\epsilon_{\text{Training}}$ is bounded above by $\prod_{t=1}^{T} Z_t$. At step t the values $Z_1, Z_2, \ldots, Z_{t-1}$ are already fixed. Therefore, at step t we can choose β_t to minimize Z_t. Let

$$\epsilon_t = \sum_{j=1}^{m} w_j^{(t)} 1_{\{h_t(x^j)\neq y^j\}}$$

be the weighted training error for weak classifier $h_t(x)$. Then we can re-write the formula for Z_t as:

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

(a) *[4 points]* First, find the value of β_t that minimizes Z_t. Then **show that:**

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

(b) *[4 points]* Assume we choose β_t to have this optimal value, and thus that $Z_t = Z_t^{opt}$. We can express the training error $\epsilon_t = \frac{1}{2} - \gamma_t$, so that $\gamma_t > 0$ implies that the weak classifier $h_t(x)$ performs better than random (i.e., the weighted training error $\epsilon_t < \frac{1}{2}$) and $\gamma_t < 0$ implies that it performs worse than random. Then **show that:**

$$Z_t^{opt} = Z_t \leq \exp(-2\gamma_t^2).$$

You may take as given the fact that $\ln(1-x) \leq -x$ for $0 \leq x < 1$.

Putting together this result with your results from parts a and b, you have shown that:

$$\epsilon_{\text{training}} \leq \prod_{t=1}^{T} Z_t \leq \exp(-2\sum_{t=1}^{T} \gamma_t^2)$$

(c) *[4 points]* Finally, use the inequality above to show that if each classifier is better than random (i.e., there exists a constant $\gamma > 0$ such that $\gamma_t \geq \gamma$ for all t) then:

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

which shows that the training error can be made arbitrarily small with enough steps (a large enough T).

4 Programming: AdaBoost Algorithm *[25 points]*

In this section, you will implement the AdaBoost algorithm with decision stumps, i.e., 1-level decision trees. You will need to implement the adaboost algorithm from scratch and train and test the classifier on the dataset provided. We provide suggestions for designing your code and a corresponding template as noted below, but you are not required to follow these suggestions.

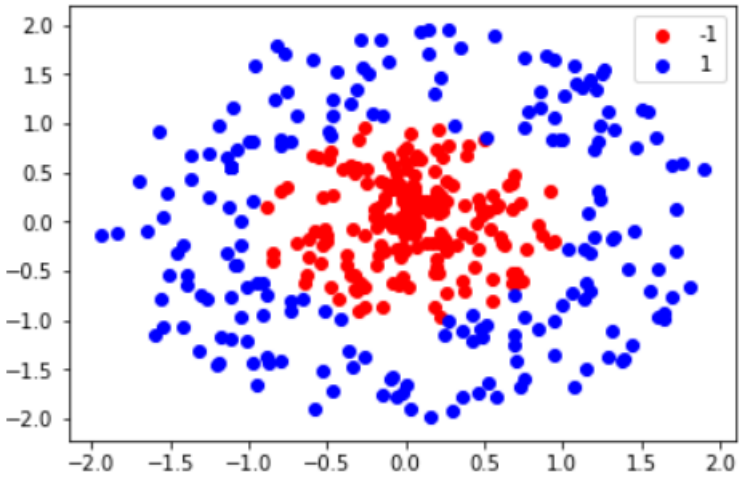

Figure 1: Figure 2: Example dataset. Each point $(x_1, x_2) \in [-2, 2] \times [-2, 2]$ and $y_i \in \{-1, 1\}$

The dataset for this task is synthetically generated and has two entries $\mathbf{x}_i = (x_1, x_2)$ and y. Here $\mathbf{x}_i = (x_1, x_2) \in [-2, 2] \times [-2, 2]$ is the i-th instance and $y_i \in \{-1, 1\}$ is its corresponding label. The training and test dataset are in "train-adaboost.csv" and "test-adaboost.csv" respectively.

a. **Suggested structure for the programming assignment**: Consider writing the following functions:

(a) **read_data**: This function reads the data from the input file.

(b) **weak_classifier**: This function finds the best weak classifier, which is a 1-level decision tree. Defining n as number of training points and d as the number of features per data point, the inputs to the function should be the input data ($n \times d$ array), the true labels ($n \times 1$ array) and the current distribution D ($n \times 1$ array) and you should return the best weak classifier with the best split based on the error. Some of the things to include in the output can be: best feature index,best split value, label, value of β_t and predicted labels by the best weak classifier. Note: β_t should be a ($T \times 1$ array, for $t = 1 \dots T$, and T denotes the number of iterations that you choose to run the boosting algorithm)

(c) **update_weights**: This function computes the updated distribution D_{t+1}, The inputs to this function should be current distribution D ($n \times 1$ array), the value of β_t, the true target values ($n \times 1$ array) and the predicted target values ($n \times 1$ array). And the function should output the updated distribution D.

(d) **adaboost_predict**: This function returns the predicted labels for each weak classifier. The inputs: the input data ($n \times d$ array), the array of weak classifiers ($T \times 3$ array) and the array of β_t for $t = 1 \dots T$ ($T \times 1$ array) and output the predicted labels ($n \times 1$ array)

(e) **eval_model**: This function evaluates the model with test data by measuring the accuracy. Assuming we have m test points, the inputs should be the test data ($m \times d$ array), the true labels for test data ($m \times 1$ array), the array of weak classifiers ($T \times 3$ array), and the array of β_t for $t = 1 \dots T$ ($T \times 1$ array). The function should output: the predicted labels ($m \times 1$ array) and the accuracy.

(f) **adaboost_train**: This function trains the model by using AdaBoost algorithm.
Inputs:
- The number of iterations (T)
- The input data for training data ($n \times d$ array)
- The true labels for training data ($n \times 1$ array)
- The input features for test data ($m \times 2$ array)
- The true labels for test data ($m \times 1$ array)

Output:
- The array of weak classifiers ($T \times 3$ array)
- The array of β_t for $t = 1 \dots T$ ($T \times 1$ array)

(g) **main**:

```
def main():
    num_iter = 400

    X_train, y_train = read_data("train_adaboost.csv")
    X_test, y_test = read_data("test_adaboost.csv")

    hlist, alphalist = train(num_iter, X_train, y_train, X_test, y_test)
    final_pred, final_acc = eval_model(X_test, y_test, hlist, alphalist)
```

NOTE: You may use the function signatures mentioned above or this code template (linked) that uses the same functions but as a class and thus makes use of class attributes. You **do not** necessarily have to use the suggested function signatures or code template. A different working code structure is acceptable.

a. Test your classifier by training with varying the number of iteration(num_iter) from 1 to 400; and plot the test accuracy vs the number of iterations. Also report the final test accuracy of the classifier when trained for 400 iterations.