CM146, Winter 2018
Problem Set 4: Boosting, Multi-class Classification
Due March 8, 2018, 11:59pm

Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

1 Boosting - 40 points

Consider the following examples $(x, y) \in \mathbb{R}^2$ (i is the example index):

i	x	y	Label
1	0	8	−
2	1	4	−
3	3	7	+
4	-2	1	−
5	-1	13	−
6	9	11	−
7	12	7	+
8	-7	-1	−
9	-3	12	+
10	5	9	+

In this problem, you will use Boosting to learn a hidden Boolean function from this set of examples. We will use two rounds of AdaBoost to learn a hypothesis for this data set. In each round, AdaBoost chooses a weak learner that minimizes the error ϵ. As weak learners, use hypotheses of the form (a) $f_1 \equiv [x > \theta_x]$ or (b) $f_2 \equiv [y > \theta_y]$, for some integers θ_x, θ_y (either one of the two forms, not a disjunction of the two). There should be no need to try many values of θ_x, θ_y; appropriate values should be clear from the data. When using log, use base 2.

(a) **[10 points]** Start the first round with a uniform distribution D_0. Place the value for D_0 for each example in the third column of Table 1. Write the new representation of the data in terms of the *rules of thumb*, f_1 and f_2, in the fourth and fifth columns of Table 1.

(b) **[10 points]** Find the hypothesis given by the weak learner that minimizes the error ϵ for that distribution. Place this hypothesis as the heading to the sixth column of Table 1, and give its prediction for each example in that column.

i	Label	Hypothesis 1 (1st iteration)				Hypothesis 2 (2nd iteration)			
		D_0	$f_1 \equiv [x > 2]$	$f_2 \equiv [y > 6]$	$h_1 \equiv [x > 2]$	D_1	$f_1 \equiv [x > 10]$	$f_2 \equiv [y > 11]$	$h_2 \equiv [y > 11]$
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
1	$-$	0.1	$-$	$+$	$-$	$\frac{1}{16}$	$-$	$-$	$-$
2	$-$	0.1	$-$	$-$	$-$	$\frac{1}{16}$	$-$	$-$	$-$
3	$+$	0.1	$+$	$+$	$+$	$\frac{1}{16}$	$-$	$-$	$-$
4	$-$	0.1	$-$	$-$	$-$	$\frac{1}{16}$	$-$	$-$	$-$
5	$-$	0.1	$-$	$+$	$-$	$\frac{1}{16}$	$-$	$+$	$+$
6	$-$	0.1	$+$	$+$	$+$	$\frac{1}{4}$	$-$	$-$	$-$
7	$+$	0.1	$+$	$+$	$+$	$\frac{1}{16}$	$+$	$-$	$-$
8	$-$	0.1	$-$	$-$	$-$	$\frac{1}{16}$	$-$	$-$	$-$
9	$+$	0.1	$-$	$+$	$-$	$\frac{1}{4}$	$-$	$+$	$+$
10	$+$	0.1	$+$	$+$	$+$	$\frac{1}{16}$	$-$	$-$	$-$

Table 1: Table for Boosting results

(c) **[10 points]** Now compute D_1 for each example, find the new best weak learners f_1 and f_2, and select hypothesis that minimizes error on this distribution, placing these values and predictions in the seventh to tenth columns of Table 1.

$$\alpha_0 = \frac{1}{2}\log_2\left(\frac{1-\epsilon_0}{\epsilon_0}\right) = \frac{1}{2}\log_2\left(\frac{1-\frac{2}{10}}{\frac{2}{10}}\right) = \frac{1}{2}\log_2 4 = 1$$

$$\sum_i D_1(i) = \sum_i \frac{D_0(i)}{Z_0}\, 2^{-\alpha_0 y_i h_t(x_i)} = \frac{\frac{1}{10}(2^{-1}\times 8 + 2\times 2)}{Z_0} = 1$$

$$\text{Hence,} \quad Z_0 = \frac{4}{5} \quad \text{and} \quad D_1(i) = \frac{1}{8}\times 2^{-y_i h_t(x_i)}$$

(d) **[10 points]** Write down the final hypothesis produced by AdaBoost.

$$\epsilon_1 = 4\times\frac{1}{16} = \frac{1}{4}$$

$$\alpha_1 = \frac{1}{2}\times\log_2\left(\frac{1-\frac{1}{4}}{\frac{1}{4}}\right) = \frac{1}{2}\log_2 3$$

$$H_{final}(x) = \text{sgn}\left(h_1(x) + \left(\frac{1}{2}\log_2 3\right)h_2(x)\right)$$

What to submit: Fill out Table 1 as explained, show computation of α and $D_1(i)$, and give the final hypothesis, H_{final}.

2 Multi-class classification - 60 points

Consider a multi-class classification problem with k class labels $\{1, 2, \ldots k\}$. Assume that we are given m examples, labeled with one of the k class labels. Assume, for simplicity, that we have m/k examples of each type.

Assume that you have a learning algorithm L that can be used to learn Boolean functions. (E.g., think about L as the Perceptron algorithm). We would like to explore several ways to develop learning algorithms for the multi-class classification problem.

There are two schemes to use the algorithm L on the given data set, and produce a multi-class classification:

- **One vs. All:** For every label $i \in [1, k]$, a classifier is learned over the following data set: the examples labeled with the label i are considered "positive", and examples labeled with any other class $j \in [1, k], j \neq i$ are considered "negative".
- **All vs. All:** For every pair of labels $\langle i, j \rangle$, a classifier is learned over the following data set: the examples labeled with one class $i \in [1, k]$ are considered "positive", and those labeled with the other class $j \in [1, k], j \neq i$ are considered "negative".

(a) **[20 points]** For each of these two schemes, answer the following:

i. How many classifiers do you learn?
One vs. All: k,
times All vs. All: $\frac{k(k-1)}{2}$.

ii. How many examples do you use to learn each classifier within the scheme?
One vs. All: m,
All vs. All: $2m/k$.

iii. How will you decide the final class label (from {1, 2, . . . , k}) for each example?
One vs. All: $y^{\text{test}} = \arg\max_{y_i \in [1,k]} W_{y_i}^T X^{\text{test}}$, which means that take the classifer with largest product of all the k weight vectors and feature vector to be predicted.
All vs. All: either by majority stategy, where label i wins on X^{test} more often than any other labels, or by tournament, where start with $m/2$ pairs and continue with winners.

iv. What is the computational complexity of the training process?
According to questions on Piazza, the computational complexity here is time complexity.
One vs. All: $O(mk)$,
All vs. All: $O\left(\frac{k(k-1)}{2} \times \frac{2m}{k}\right) = O(m(k-1)) = O(mk)$.

(b) **[5 points]** Based on your analysis above of two schemes individually, which scheme would you prefer? Justify.

If the only critierion here is computational time complexity, both schemes are equivalent. If we consider the space complexity, then One vs. All is better.

(c) **[5 points]** You could also use a KernelPerceptron for a two-class classification. We could also use the algorithm to learn a multi-class classification. Does using a KernelPerceptron change your analysis above? Specifically, what is the computational complexity of using a KernelPerceptron and which scheme would you prefer when using a KernelPerceptron?

In kernal perception, $y_j\left(\sum_{1\ldots n} \alpha_i y_i K(X_i, X_j)\right)$ is computed for each iteration, where n is number of examples and α_i is the count of examples as constituents in weight vector. As the algorithm proceeds, more α_i will be nonzero, so the computational complexity increases for 1 single classifier, which is bounded by $O(d + 2d + ... + nd) = O(n^2 d)$, where d is the

dimension of feature vectors. For One vs. All, $n = m$; for All vs. All, $n = 2m/k$. Clearly, All vs. All has better computational time complexity of the overall training process.

(d) **[10 points]** We are given a magical black-box binary classification algorithm (we dont know how it works, but it just does!) which has a learning time complexity of $\mathrm{O}(dn^2)$, where n is the total number of training examples supplied (positive+negative) and d is the dimensionality of each example. What are the overall training time complexities of the all-vs-all and the one-vs-all paradigms, respectively, and which training paradigm is most efficient?

One vs. All: $O\left(dm^2k\right)$,
All vs. All: $O\left(d\left(\frac{2m}{k}\right)^2 \times \frac{k(k-1)}{2}\right) = O(dm^2)$.
Hence, as long as k is not a constant, them All vs. All is better.

(e) **[10 points]** We are now given another magical black-box binary classification algorithm (wow!) which has a learning time complexity of $\mathrm{O}(d^2n)$, where n is the total number of training examples supplied (positive+negative) and d is the dimensionality of each example. What are the overall training time complexities of the all-vs-all and the one-vs-all paradigms, respectively, and which training paradigm is most efficient, when using this new classifier?

One vs. All: $O\left(d^2mk\right)$,
All vs. All: $O\left(d^2 \times \frac{2m}{k} \times \frac{k(k-1)}{2}\right) = O(d^2m(k-1)) = O(d^2mk)$.
Both schemes are equivalent in terms of time complexity.

(f) **[10 points]** Suppose we have learnt an all-vs-all multi-class classifier and now want to proceed to predicting labels on unseen examples.

We have learnt a simple linear classifier with a weight vector of dimensionality d for each of the $m(m-1)/2$ classes ($w_i^T x = 0$ is the simple linear classifier hyperplane for each $i = [1, \cdots, m(m-1)/2]$)

We have two evaluation strategies to choose from. For each example, we can:

- **Counting**: Do all predictions then do a majority vote to decide class label
- **Knockout**: Compare two classes at a time, if one loses, never consider it again. Repeat till only one class remains.

What are the overall evaluation time complexities per example for Counting and Knockout, respectively?

Counting: $O(\frac{k(k-1)}{2} * d) = O(k^2d)$,
Knockout: $O\left(\left(k + \frac{k}{2} + ... + 1\right) \times d\right) = O(kd)$ Knockout is more efficient.