CM146, Fall 2019
Problem Set 4: Learning Theory, Boosting, Multi-class Classification
Due December 8, 2019, 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 PAC Learning [25 pts]

Let $\mathcal{X}$ be a set of numbers {1,2,. . . , N}, and let $H_{Singleton} = \{h_z : z \in \mathcal{X}\} \cup \{h^-\}$ is the hypothesis space, where for each $z \in \mathcal{X}$, h_z is the function defined by $h_z(x) = 1$ if $x = z$ and $h_z(x) = 0$ if $x \neq z$. h^- is simply the all-negative hypothesis, namely, $\forall x \in X, h^-(x) = 0$. (For any hypothesis h, it only receives input $x \in \mathcal{X}$, which means the domain is discrete, consisting N numbers).

Assume a training dataset S_{train} consist of m data points drawn i.i.d from a uniform distribution D (each data point $x \in X$ has equal probability, i.e., $\frac{1}{N}$, to be sampled), and that the labels are provided from some target function $h^* \in H_{Singleton}$ (true hypothesis belongs to our learning hypothesis family). We simply choose 0-1 loss as train and generalization error metric.

(a) **[10 points]** Describe an algorithm A that learns a hypothesis $A(S_{train})$ from hypothesis family $H_{Singleton}$ on training set S_{train}, so that the training error is minimized (such an algorithm is Empirical Risk Minimization (ERM)).

(b) **[15 points]** Prove that if training set S_{train} has a sample size larger than $\frac{\log(1/\sigma)}{\epsilon}$, then the probability that the generalization error of $A(S_{train})$ is larger than ϵ is at most σ (This means that $H_{Singleton}$ is PAC learnable), i.e.:

$$\mathbf{P}\Big(L_{(D,h^*)}\big(A(S_{train})\big) > \epsilon\Big) \leq \sigma \tag{1}$$

where the generalization error is defined as:

$$\begin{aligned} & L_{(D,h^*)}\big(A(S_{train})\big) && (2)\\ = \ & \mathbf{E}_{x\sim\mathcal{D}}\Big(\text{0-1-}Loss(h^*(x), A(S_{train})(x))\Big) && (3)\\ = \ & \mathbf{P}_{x\sim\mathcal{D}}\Big(h^*(x) \neq A(S_{train})(x)\Big) && (4)\end{aligned}$$

Hint: Write down the generalization error of $A(S_{train})$, estimate the probability that it's larger than ϵ, and bound this probability by σ. The following inequality is useful: $1-x \leq e^{-x}$.

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

Table 1: Table for Boosting results

2 VC Dimension [15 pts]

We define the following hypothesis set:

$$H = \{sgn(ax^2 + bx + c); a, b, c, \in R\},$$

where $sgn(\cdot)$ is 1 when the argument $\cdot$ is positive, and 0 otherwise. What is the VC dimension of H? Prove your claim.

3 Boosting [40 pts]

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.

(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.

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

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

4 Multi-class classification [60 pts]

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?
  ii. How many examples do you use to learn each classifier within the scheme?
  iii. How will you decide the final class label (from {1, 2, ..., k}) for each example?
  iv. What is the computational complexity of the training process?

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

(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?

(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 $O(dn^2)$, 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?

(e) **[10 points]** We are now given another magical black-box binary classification algorithm (wow!) which has a learning time complexity of $O(d^2n)$, 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?

(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 $k(k-1)/2$ classifier ($w_i^T x = 0$ is the simple linear classifier hyperplane for each $i = [1, \cdots, k(k-1)/2]$)

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?