MULTICLASS COST-SENSITIVE LEARNING

1. Problem Model

In multiclass learning, given an input X from some arbitrary-but-fixed space, we want to produce an action $a \in \{1, ..., A\}$. After choosing the label, we incur some cost (or loss), l_a.

Suppose that there is some unknown distribution D, generating examples, on the space $X \times R^{+l}$. Each example has the form $(x, l_1, l_2, ..., l_A)$. The goal in cost-sensitive multiclass learning is to produce a (multiclass) classifier c which minimizes the expected cost, $E_{(x,l_1,l_2,...,l_A)\sim D} l_{c(x)}$.

Our goal is to reduce thie multiclass cost sensitive learning problem to the multiclass learning problem. In the multiclass learning problem, the distribution D is over $X \times \{1, ..., A\}$. In the multiclass learning problem we seek to find a classifier c which minimizes $E_{x,a\sim D} I(c(x) \neq a)$.

2. Method of Reduction

Our primary tool for accomplishing a reduction is shifting the distribution over examples. The basic difficulty with using this tool is that it is unclear how we should shift the distribution, as there are multiple ways that a classifier could err. Should we err against the first failure mode, the second, or the $A-1$th? The correct answer appears fundamentally dependent upon the classifier learning algorithm and the example. Given this dependence, we must extract information from the learning algorithm in order to find the correct weighting. The proposed method for doing this is an iterative procedure.

Cost_to_importance(multiclass learning algorithm A, multiclass cost sensitive dataset S)

Initialization

(1) For every $(x, l_1, l_2, ..., l_A) \in S$, set
(2) $i_x \leftarrow \max_a l_a - \min_a l_a$
(3) $k_x \leftarrow 1$
(4) $a_x \leftarrow \operatorname{argmin}_a l_a$

Iteration

(1) Let $\hat{S} = \{((x, a_x), i_x) : (x, l_1, ..., l_A) \in S\}$.
(2) Let $c = A(\hat{S})$.
(3) For each $(x, l_1, l_2, ..., l_A) \in S$
  (a) if $c(x) \neq a_x$
    (i) $i_x \leftarrow \frac{i_x k + l_{c(x)} - \min_a l_a}{k+1}$
    (ii) $k \leftarrow k + 1$
(4) If the l_1 distance between the old $i_x$and the new i_x is small then terminate.
(5) Goto 1

Termination

(1) Output the classifer c.

3. Things you can Prove

We want to prove that this is a good reduction to importance weighted classification. Very little can be said theoretically. In particular, if the importance weighted classification algorithm is inimical, then **Cost_to_importance** may never converge. Even when converges happens, we can not generally guaratnee that the values i_x are the "right" values. However, we can show these are the "right" values for every example where the final solution errs.

Assume that the loss rate for any importance weighted problem is:

$$\epsilon = E_{x,y,i\sim D} i I(c(x) \neq y)$$

There is one obvious problem with this statement: it isn't normalized. In particular, with no constraint on the value of i, doubling the value of every i while keeping ϵ reduces the number of errs. It seems unreasonable to believe that good learning algorithms halve (in some sense) the quantity of errors when i is doubled. To remove this effect, let's assume that the above condition holds only when:

$$E_{x,y,i\sim D} i = 1$$

Theorem 3.1. *If A has loss rate ϵ and **Cost_to_importance** converges, then:*

$$E_{x,c_1,c_2,\ldots,c_l\sim D} l_{c(x)} - \min_a l_a = E_{x,y,i_x\sim \hat{D}} i_x = \epsilon$$

Proof. The strong learning assumption for A implies that for all i_x:

$$E_{(x,y,i_x)} i_x I(c(x) \neq y) = \epsilon$$

Convergence, implies that for every example where $c(x) \neq y$, we have $l_{c(x)} - \min_a l_a = i_x$. Furthermore, when $c(x) = y$, we have $0 = l_{c(x)} - \min_a l_a$.

Since $i_x I(c(x) \neq y) = l_{c(x)} - \min_a l_a$ regardless of whether $c(x) \neq y$, the expectations are the same. $\square$

From the perspective of the proof, we could instead use the method:

(1) Guess a choice of i_x
(2) Check that it is consistent with the observed losses.

Using the proposed update mechanism is hopefully more efficient than guess-and-check, and if the learning algorithm cooperates, i_x may converge.

What is a natural way to specify "the algorithm is nice and so **Cost_to_importance** converges"?

Nonconvergence may motivate averaging over the nonconvergent solutions.