CS 395T Computational Learning Theory

Lecture 1: September 06, 2006 — Lecturer: Adam Klivans

Scribe: Tuyen Huynh

1.1 Mistake Bound Model of Learning

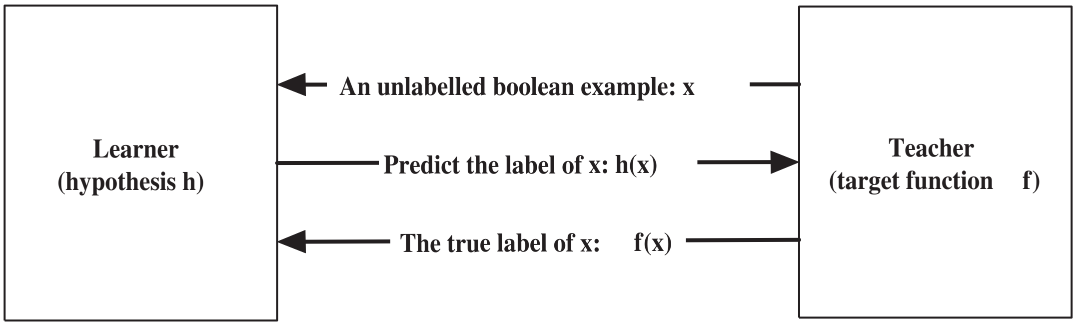

Figure 1.1: Mistake Bound model of learning

The Mistake Bound (MB) model of learning, figure 1.1, was introduced in [1]. In this model, the learner tries to learn a target function f through a series of trials. In each trial, the following steps take place:

- Step 1: The learner is provided an unlabelled boolean example $x \in \{0,1\}^n$ from the teacher.
- Step 2: Based on its current hypothesis h, the learner predicts the label of x as $h(x)$.
- Step 3: The learner receives the correct label $f(x)$ from the teacher. If the learner made a mistake ($h(x)! = f(x)$), it updates its current hypothesis h.

To evaluate a learning algorithm in this model, we consider how many mistakes an algorithm makes. The smaller mistakes an algorithm incurs, the better it is. Note that, all the learning algorithms in this model are online and incremental learning algorithms, i.e they can update their model incrementally every time they see a new training example. Following are some important definitions.

Definition 1 *A concept c over $X = \{0,1\}^n$ is a boolean function over X, i.e. c is a mapping from X to $\{0,1\}$. A concept class C is a set of concepts having the same representations as c.*

Some examples of concept class:

- Disjunctions: $f(X_1, X_2, ..., X_n) = X_1 \vee X_2 \vee \bar{X}_3 \vee ... \vee X_n$

- Conjunctions: $f(X_1, X_2, ..., X_n) = \bar{X}_1 \wedge X_2 \wedge ... \wedge \bar{X}_n$

Definition 2 *A learning algorithm A has a mistake bound $M(c)$ on a target concept c if for any sequence of examples (possibly infinite), A make at most $M(c)$ mistakes while learning c. And the mistake bound of a learning algorithm A on a concept class C is the worst number of mistakes made by A across all function $f \in C$.*

Definition 3 *A concept class C over $\{0,1\}^n$ is efficiently learnable in MB if there exists an algorithm A such that for any concept $c \in C$, A has a mistake bound $M(c) = poly(n, size(c))$ and its running time in each trial is also $poly(n, size(c))$ where $size(c)$ is the description length of c under some encoding scheme. In addition, if the mistake bound is only $poly(size(c)) \times polylog(n)$ then A is said to be an attribute efficient learning algorithm.*

1.2 Learning Decision List in Mistake Bound model

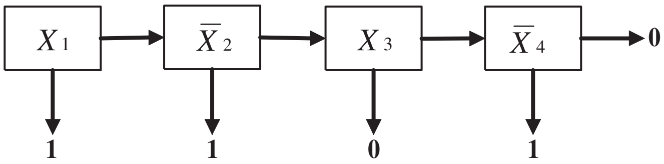

Figure 1.2: A decision list

A decision list of length k over n Boolean variables $X_1, ..., X_n$ is a list L of k pairs and a bit:

$$(t_1, v_1), (t_2, v_2), ..., (t_k, v_k), v_{k+1}$$

where each t_i is a literal and each v_i is either 1(true) or 0(false). In figure 1.2, the literal inside each box corresponds to the literal t_i, and the number below each box is the value of the corresponding v_i. A decision list L defines a Boolean function from $\{0,1\}^n$ to $\{0,1\}$. For any $x \in \{0,1\}^n$, the value of $L(x)$ is v_i where i is the least index such that t_i is made true by x; if no t_i is satisfied by x then $L(x) = v_{k+1}$. To compute the value of $L(x)$ for a given x, we start from the head of the list, evaluate the literal inside the box. if the literal is satisfied by x then output the number below that box as the value of $L(x)$, otherwise move to the next box and repeat the process. From this point of view, a decision list corresponds to a nested *if-then-else* statements. This kind of representation was introduced in [2].

Next, we present an algorithm for learning this concept class, decision lists of length k, with a mistake bound $O(nk)$. We define a basic block of decision list to be a singe box, figure 1.3, where the inside literal can be X_i or $\bar{X}_i$ and the output value can be either 0 or 1. This basic block corresponds to a *if-then-else* rule. With n variables, we have $4n$ different basic blocks.

Algorithm:

1. Create k bags correspond to k levels of a decision list. Put all possible basic blocks into the first bag (the head of the decision list), and let other bags empty.

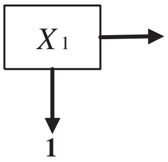

Figure 1.3: A component of a decision list

2. Given an example x, start from the first bag, find a basic blocks such that its literal is satisfied by x, then use it output value as the label for x. If there are multiple blocks satisfied by x, pick one arbitrarily. If none of the blocks in the current bag are satisfied then move to next bag and repeat the search step. Continue until a satisfied block is found.

3. If the prediction is incorrect, demote the chosen block to the next bag.

Note that, the above algorithm may not produce a decision list since it may happen that there are more than one component in some bags but the hypothesis formed by those bags is consistent with the target concept.

Claim 1 *The true basic blocks of the target concept will never be demoted past its correct position.*
Proof: We prove this by induction.

- Base case: $k = 1$. Since the algorithm only demote a block when it makes a mistake, the block corresponding to the head of the target decision list will never be demoted to the second bag.
- Induction step. Assume that the claim is hold for all true blocks from 1 to $k - 1$. Consider a new example that cannot be satisfied by any basic blocks in any bags from 1 to $k - 1$. By the inductive hypothesis, this example is not satisfied by any of the first $k - 1$ true basic blocks. Therefore to predict the label of this example, we need to use a block in bag k. If the block chosen is the correct one then the prediction is correct, so the algorithm will not demote this block to the next bag. In other words, the correct block for the $k - th$ level will never be demoted past the $k - th$ bag.

■

Claim 2 *Every time the algorithm makes a mistake, some blocks are demoted.*

Proof: This claim is obvious from the algorithm. Every time it makes a mistake, the chosen block will be demoted. ■

From these two claims, we can see that at each level the algorithm makes $O(n)$ mistakes. There are k levels, so in total the algorithm makes $O(nk)$ mistakes.

References

[1] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[2] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.