15-850(C) Machine Learning Theory

Lecturer: Avrim Blum
Scribe: Carl Burch

Lecture 5: October 3, 1995

---

1 The Mistake Bound model

In the last lectures we discussed models for "batch" learning. We now turn to a model of on-line learning called the mistake-bound model. This model in some ways is much simpler than the PAC model because we will not need to work with distributions and accuracy and confidence parameters.

In this model, the following scenario is repeated indefinitely:

1. The algorithm receives an unlabeled example.
2. The algorithm predicts a classification of this example.
3. The algorithm is then told the correct answer.

We will call whatever is used to perform step (2), the algorithm's "current hypothesis."

Definition 1 *An algorithm A is said to* learn C in the mistake bound model *if for any concept $c \in C$, and for any ordering of examples consistent with c, the total number of mistakes ever made by A is bounded by $p(n, size(c))$, where p is a polynomial. We say that A is a* polynomial time *learning algorithm if its running time per stage is also polynomial in n and $size(c)$.*

Let us now examine a few problems that are learnable in the mistake bound model.

1.1 Conjunctions

Let us assume that we know that the target concept c will be a conjunction of a set of (possibly negated) variables, with an example space of n-bit strings.

Consider the following algorithm:

1. Initialize hypothesis h to be $x_1\overline{x_1}x_2\overline{x_2}\ldots x_n\overline{x_n}$.
2. Predict using $h(x)$.
3. If the prediction is **False** but the label is actually **True**, remove all the literals in h which are **False** in x. (So if the first mistake is on 1001, the new h will be $x_1\overline{x_2}\,\overline{x_3}x_4$.)

4. If the prediction is **True** but the label is actually **False**, then output "no consistent conjunction".

5. Return to step 2.

An invariant of this algorithm is that the set of literals in c will always be a subset of the set of literals in h.

The first mistake on a positive example will bring the size of h to n. Each subsequent such mistake will remove at least one literal from h, so that the maximum number of mistakes made will be at most $n + 1$.

In fact no deterministic algorithm can achieve a mistake bound better than n in the worst case. This can be seen by considering the sequence of n examples in which the ith example has all bits except the ith bit set to 1. The target concept c will be a monotone conjunction constructed by including x_i only if the algorithm predicts the ith example to be **True** (in which case the ith example's label will be **False**). (If the algorithm predicts the ith example to be **False**, then the target concept will not include x_i, and so the true label will be **True**.) The algorithm will have made n mistakes by the time all of these n examples are processed.

1.2 Hamiltonian paths

Say that the example space is the set of all graphs with n labeled vertices, and the target concept is a Hamiltonian path c on the complete graph, so that an example is positive if $c \subseteq G$ and negative otherwise.

The following algorithm will learn c in the mistake bound model:

1. Begin with hypothesis h as the complete graph over n labeled nodes.

2. Receive an example G.

3. Predict positive if $h \subseteq G$, negative otherwise.

4. If the prediction is positive but the actual label negative, take h to be $h \cap G$.

5. Return to step 2.

Note that the hypothesis space in this example is the set of graphs, whereas the concept space is the set of Hamiltonian paths, so H is not the same as C. In fact it is not clear that if we limit H to C that a solution exists.

Since the algorithm will only make mistakes by falsely predicting positively, and each time it does this the hypothesis will reduce in size by at least one, the number of mistakes is bounded by the initial size of h, $(n^2 - n)/2$.

This algorithm is clearly strongly analogous to the problem of discovering a conjunction, so let us look at a more interesting example.

1.3 Decision lists

Say that C is the set of decision lists on n variables. In our algorithm to learn decision lists, our hypotheses will again not belong to C. In particular, our hypothesis will be a list of *subsets* of $\{x_1 \Rightarrow \textbf{True}, x_1 \Rightarrow \textbf{False}, \overline{x_1} \Rightarrow \textbf{True}, \overline{x_1} \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{True}, \ldots \overline{x_n} \Rightarrow \textbf{False}, T \Rightarrow \textbf{True}, T \Rightarrow \textbf{False}\}$. The algorithm is the following.

1. Let h be the one-level list, whose level consists of the single set that includes all $4n+2$ possible terms.
2. Given an example x, look at the first set in h for which x satisfies the antecedents of at least one of the rules of the set. (I.e., select some rule that fires.) Predict based on this rule, breaking ties arbitrarily.
3. If the prediction is mistaken, move all the rules in that set which predicted incorrectly to the next set in the list.
4. Return to step 2.

As an example of running this algorithm, say that n is 2 and the target concept is:

$$x_1 \Rightarrow \textbf{False}, \text{ else } \overline{x_2} \Rightarrow \textbf{True}, \text{ else } \textbf{False}.$$

Initially h will be

$$\{ \text{ all rules } \}.$$

If the first example is 01 then the algorithm will see two possible predictions in the first set of h; say it chooses to predict positively. In fact, though, this is a negative example. So h will be changed to

$$\{x_1 \Rightarrow \textbf{True}, x_1 \Rightarrow \textbf{False}, \overline{x_1} \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{False}, \overline{x_2} \Rightarrow \textbf{True}, \overline{x_2} \Rightarrow \textbf{False}, T \Rightarrow \textbf{False}\},$$

$$\text{else } \{\overline{x_1} \Rightarrow \textbf{True}, x_2 \Rightarrow \textbf{True}, T \Rightarrow \textbf{True}\}.$$

Now say that we give an example of 00 to the algorithm. Then the algorithm might choose to predict that this will be negative, though in fact it is positive. So h will now be

$$\{x_1 \Rightarrow \textbf{True}, x_1 \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{False}, \overline{x_2} \Rightarrow \textbf{True}\},$$
$$\text{else } \{\overline{x_1} \Rightarrow \textbf{True}, \overline{x_1} \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{True}, \overline{x_2} \Rightarrow \textbf{False}, T \Rightarrow \textbf{True}, T \Rightarrow \textbf{False}\}.$$

And so on.

We can make several observations about this algorithm. First, with each mistake at least one term will move one level lower in h. Second, the ith rule of c will never move below the ith level of h. And finally, each rule will fall at most L levels (where L is the length of C). These three properties imply that with every mistake the algorithm always makes progress toward a correct solution. Furthermore, the number of mistaken predictions is bounded above by $(4n+2)L$. This is polynomial in n and the size of c, so this algorithm learns decision lists in the mistake bound model.

2 Relationship with PAC learning

One natural question to ask about mistake bound learning is its relationship with PAC learning. Given the hard bound on the number of mistakes, one might expect that a mistake bound learning algorithm is also a PAC learning algorithm. Should you expect this too, you are correct:

Theorem 1 *If algorithm $\mathcal{A}$ learns a concept class C in the mistake bound model, then $\mathcal{A}$ also learns C in the probably approximately correct model.*

In proving this we would like to assume that an algorithm is "conservative" or "lazy", meaning that the algorithm only changes its state (its state is its current hypothesis) when it makes a mistake. We can insure that it does this by taking a snapshot of its state before giving it an example; if the algorithm predicts correctly, we will restore the state it had before the prediction. In this case the algorithm will behave as it would were it only given examples on which it makes mistakes, so the number of mistakes is still bounded. Thus, given $\mathcal{A}$ learning C in the mistake bound model we can construct $\mathcal{A}'$ also learning C in the mistake bound model but only changing its hypothesis when it makes a mistake.

Transforming the algorithm to a lazy one is not strictly necessary, but it makes the proof simpler.

From $\mathcal{A}'$ we can construct the following algorithm $\mathcal{A}_{PAC}$ (where M is the mistake bound, which is bounded by a polynomial in n and the size of c):

> Run $\mathcal{A}'$ on the data seen, halting if any hypothesis h survives for more than $\frac{1}{\epsilon}\lg\frac{M}{\delta}$ subsequent examples. Return h as the hypothesis of the algorithm.

Since the number of mistakes made is bounded by M, the algorithm will terminate within $\frac{M}{\epsilon}\lg\frac{M}{\delta}$ examples.

The probability that the algorithm accepts a hypothesis with error greater than ϵ at most the chance that one of the at most M wrong hypotheses predicts $\frac{1}{\epsilon}\lg\frac{M}{\delta}$ examples correctly, which is at most

$$M(1-\epsilon)^{\frac{1}{\epsilon}\lg\frac{M}{\delta}} < M\frac{\delta}{M} = \delta$$

So the chance that an algorithm accepts a good hypothesis is at least $1-\delta$; hence $\mathcal{A}_{PAC}$ learns in the PAC model. ■

3 A general learning algorithm for finite concept sets

If we do not care about computation time or the hypothesis space H, and if C is a finite set, then we can achieve a mistake bound of $\lg|C|$. This is achieved by the following method called the "halving algorithm" (where H is the set of subsets of C):

1. Initialize h to C.
2. Given example x, predict according to the majority of the concepts in h.
3. Remove all the concepts in C that predicted wrongly.
4. Return to step 2.

Since with each mistake at least half of h is removed, the number of mistakes is bounded by $\lg |C|$.

Clearly this is a very nice general sort of algorithm to learn an arbitrary concept space in both the mistake bound model and the PAC model. Unfortunately, though, storing h and polling h in order to make a prediction is likely to be hard.

4 Learning with infinitely many attributes

Say that we want to learn a concept based on n attributes that range over Σ^* (i.e., attributes are string-valued). Or, suppose we have an infinite set of boolean attributes, but each example satisfies only n of them and an example is presented as a list of the attributes that are satisfied. (These are equivalent, because we can encode attribute i having a value of string s as a single boolean variable that is either true or false.) Once again, we would like to ensure a mistake bound polynomial in n and the size of the target concept c, with no dependence on the number of strings (or the number of boolean features).

4.1 Monotone conjunctions

Say that the concept will be of the form $(f_1 = \text{“hello”}) \wedge (f_2 = \text{“world”}) \wedge \ldots \wedge (f_n = \text{“foo”})$. This kind of concept is easy to learn by just deferring the construction of our initial hypothesis until the first positive examples is seen (that is, we begin by predicting negative until we make our first mistake). At that point we initialize our hypothesis to be the conjunction of rules $(f_i = s)$ satisfied by the example and cross them off as in the previous algorithm.

4.2 Monotone disjunctions

More difficult is learning monotone disjunctions in this model. The problem is that listing all infinitely many boolean attributes and crossing off the ones that appear in negative examples will take an unbounded amount of time. Instead, an algorithm that works follows (this is written from the point of view of having infinitely many boolean features):

1. Take h to be $\emptyset$.
2. Given an example x, if $x \cap h \neq \emptyset$ predict **True**, else predict **False**.

3. If the algorithm predicted **True** but the true answer was **False**, let h be $h \cap x$.

4. If the algorithm predicted **False** but the true answer was **True**, let h be $h \cup x$.

5. Return to step 2.

Say that c includes r attributes. The algorithm will predict **False** when the answer is **True** at most r times; each time up to n terms will be added to h, at least one of which will be in c. Thus the number of terms added to h is at most nr. At least one term, and no term from c, will be removed from h when the algorithm predicts **True** and the answer is **False**, so this occurs is at most $nr - r$ times. Hence the algorithm will make at most $nr - r + r = nr$ mistakes.

Below are some handouts on the topics from this class.

References

[1] A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9:373–386, 1992.

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