COS 511: Foundations of Machine Learning

Rob Schapire
Scribe: Brian Tsang

Lecture #21
April 22, 2003

1 Active Learning

Now we turn to a new type of learning where the algorithm gets to ask for particular examples, and where it has to come up with an exact answer.

Figure 1: The target concept c is a "black box" for which we must find an equivalent function.

1.1 Definition Formalization

Let c, the target concept, be a function taking our example space, X, to $\{0, 1\}$. We wish to identify $h \in \mathcal{C}$ which is exactly equivalent to c. That is, $\forall x \in X, c(x) = h(x)$.

1.2 Query Types

Our algorithm is actually allowed two types of queries:

The Membership Query:

In this type of query, the algorithm provides an example, $x \in X$, and the membership oracle returns the label of that example, $c(x)$.

The Equivalence Query:

In this type of query, the algorithm provides a hypothesis, $h \in \mathcal{C}$, and the equivalence oracle returns "Yes" if h is equivalent to c. If not, the equivalence oracle returns "No" and provides a counterexample (i.e. some $x_c \in X$ such that $c(x_c) \neq h(x_c)$).

2 The Significance of Active Learning

There are some problems in which active learning learns where an algorithm which is merely provided with m examples according to some distribution, $\mathcal{D}$, cannot. Here we describe one such problem: determining an n-state Finite State Automaton. Kearns and Valiant showed that it is computationally hard to even weakly PAC learn an FSA using any hypothesis space.

2.1 Problem Formanlization

Our target concept c is an n-state FSA where:

$$
\begin{aligned}
Q &\equiv \text{the set of states} && (1)\\
q_0 \in Q &\equiv \text{the start state} && (2)\\
\delta : Q \times \{a, b\} \to Q &\equiv \text{the state transition function} && (3)\\
v : Q \to \{0, 1\} &\equiv \text{the label that the FSA returns for any} && (4)\\
&\quad \text{sequence ending at the given state} && (5)\\
n &\equiv |Q| && (6)
\end{aligned}
$$

Our example space is the set of all strings formed with the letters a and b, where the empty string is denoted by λ.

2.2 Example

Consider the FSA drawn below:

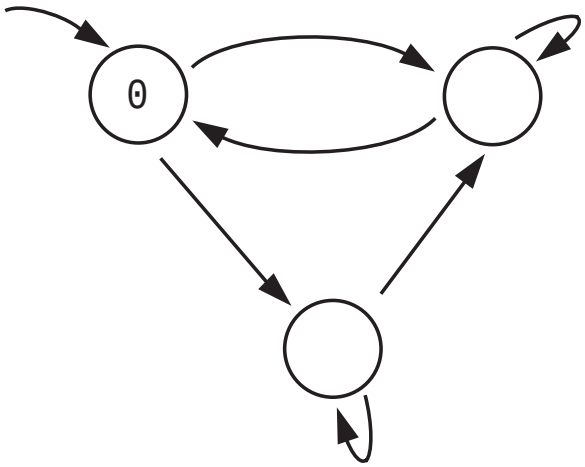


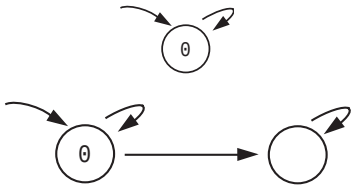
Figure 2: This FSA is equivalent to "not empty and not ending with ab" or "$(a|b)^*ab$".

The following are some example queries on the FSA.

Membership Queries:

$$
\begin{aligned}
aba &\longrightarrow 1\\
bbab &\longrightarrow 0\\
\lambda &\longrightarrow 0
\end{aligned}
$$

Equivalence Queries:



$\longrightarrow$ `No;` "a" $\to 1$

$\longrightarrow$ `No;` "a" $\to 1$

2.3 The Necessity of Equivalence Queries

It seems that equivalence queries are rather powerful, but are they necessary? In this problem we can definitley spot an instance where they are necessary. Consider the combination lock FSA which matches only to a certain sequence of n characters; you would have to query $\Omega(2^n)$ examples to differentiate this from an FSA which always returns 0!

2.4 Distinguishing States

First, let us adopt a shorthand for state transitions. Let $q \in Q$ and $e \in \{a, b\}$.

$$qe \equiv \delta(q, e) \tag{7}$$

We can repeatedly apply this operation so that qs is the state the FSA will be in after reading the string s given that it was in state q just before it was fed s.

Now for the definitions: q_1 and $q_2 \in Q$ are said to be distinguished by a string s if $v(q_1 s) \neq v(q_2 s)$.

A set of strings, T, is a distinguishing set of our FSA if and only if for every pair of states in Q at least one string in T distinguishes them. More precisely:

$$\forall q_1, q_2 \in Q, q_1 \neq q_2 \Rightarrow \exists t \in T : v(q_1 t) \neq v(q_2 t) \tag{8}$$

2.5 Example

Consider the FSA drawn below:

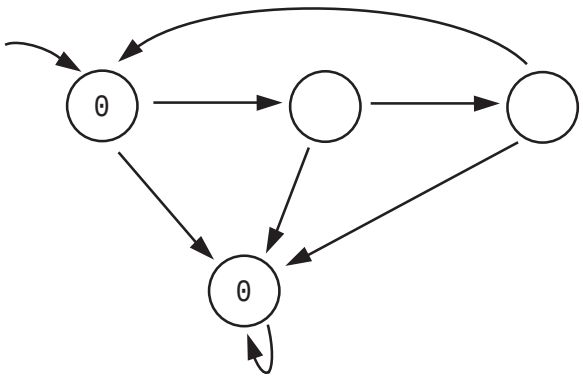


Figure 5: This FSA is equivalent to "no b's and only a number of a's which is eqivalent to 1 or 2 modulo 3" or "$[(aaa)^*a]|[(aaa)^*aa]$".

A minimal set of disginguishing strings for this DFA is $\{\lambda, aa\}$.

2.6 Angluin's Algorithm: Part I

Suppose we are given a set of distinguishing strings, $T = \{t_1, t_2, \ldots, t_m\}$. We claim that, we can find an FSA equivalent to c using only membership queries!

Consider the following fact: Suppose you have a string s_1 and a string s_2, and you want to

know if they end up in different states when fed into the FSA (i.e. Does $q_0 s_1 = q_0 s_2$?). Well, you could append each of our distinguishing strings, to s_1 and s_2 and query the results. If for any distinguishing string, s_d, we find that $v(q_0 s_1 s_d) \neq v(q_0 s_2 s_d)$, then and only then will our two states be different.

By this fact, we can find a unique identifier for each state, q: the ordered m-tuple, $(v(qt_1), v(qt_2), \ldots, v(qt_m))$.

We can use this to take a methodical approach to determining the FSA. First, presume that all strings yield different states. Then, find the identifier for each state, and merge states with the same identifier.

Let's formalize this into a high-level algorithm:

1. Initialize a stack of states to be analyzed.
2. Add q_0 to the stack.
3. While there is a state q in the stack:
   (a) Pop q off the stack.
   (b) Determine the identifier of q.
   (c) If the identifier has been encountered before, in state q_i, merge the states q and q_i.
   (d) Otherwise, push qa and qb onto the stack (and define the appropriate transitions from q).

2.7 Example

The following is the result of running the algorithm on the FSA in Figure 5. Note how the two are isomorphic:

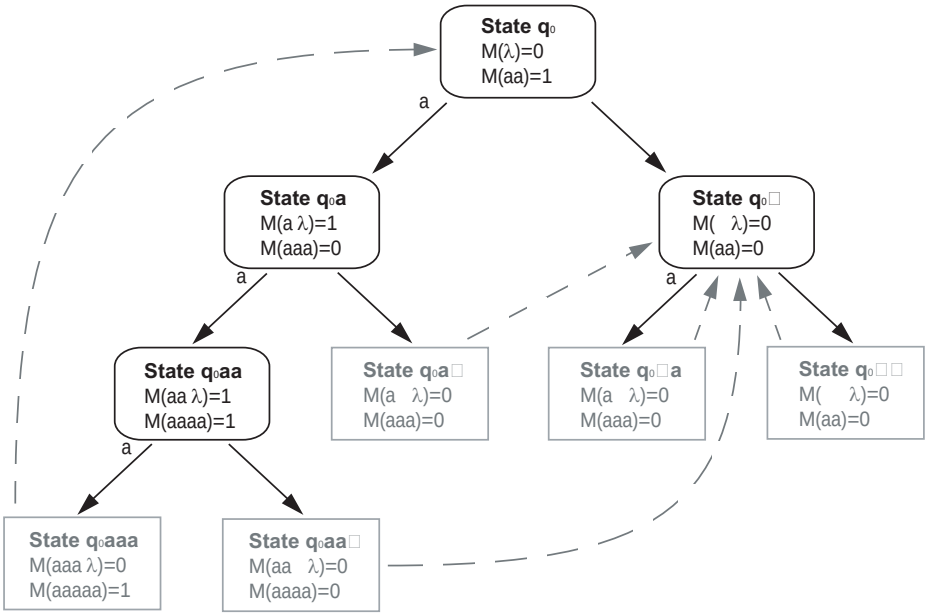


Figure 6: The result of running the algorithm on the FSA in Figure 5. Dotted arrows indicate that two states have been merged. $M(s)$ is a membership query on string s (i.e. $M(s) = v(q_0 s)$).

2.8 Next Time (Algluin's Algorithm: Part II)

Of course, we are not given T. So how do we fix this hole in our algorithm?

Next time we will discuss a way to start out with some guess for T_i (where our initial guess would be $T_0 = \{\lambda\}$), build a tree and model M_i, run an equivalence query on M_i, and use the counterexample to add a string to T_i, repeating this process until the equivalence query returns "Yes".