15-859(B) Machine Learning
Learning finite state environments

Avrim Blum

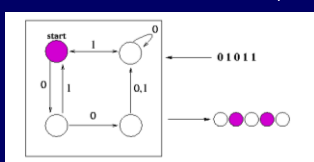
04/02/12

Consider the following setting

- Say we are a baby trying to figure out the effects our actions have on our environment...
  - Perform actions
  - Get observations
  - Try to make an internal model of what is happening.

A model: learning a finite state environment

- Let's model the world as a DFA. We perform actions, we get observations.
- Our actions can also change the state of the world. # states is finite.



Another way to put it

- We have a box with buttons and lights.

- Can press the buttons, observe the lights.
  lights = f(current state)
  next state = g(button, current state)
- **Goal: learn predictive model of device.**

Learning a DFA

In the language of our standard models...

- Asking if we can learn a DFA from Membership Queries.
  - Issue of whether we have counterexamples (Equivalence Queries) or not.
    [for the moment, assume not]
  - Also issue of whether or not we have a reset button.
    [for today, assume yes]

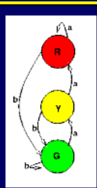
Learning DFAs

This seems really hard. Can't tell for sure when world state has changed.

Let's look at an easier problem first: state = observation.

An example w/o hidden state

2 actions: a, b.



Generic algorithm for lights=state:
- Build a model.
- While not done, find an unexplored edge and take it.

Now, let's try the harder problem!

Some examples

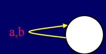
Example #1 (3 states)

Example #2 (3 states)

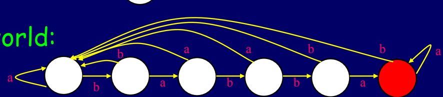
Can we design a procedure to do this in general?

One problem: what if we always see the same thing? How do we know there isn't something else out there?

Our model:

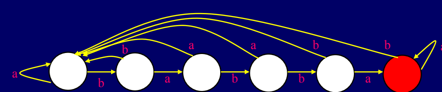


Real world:



Called "combination-lock automaton"

Can we design a procedure to do this in general?



Combination-lock automaton: basically simulating a conjunction.

This means we can't hope to efficiently come up with an exact model of the world from just our own experimentation. (I.e., MQs only).

How to get around this?

- Assume we can propose model and get counterexample. (MQ+EQ)
- Equivalently, goal is to be predictive. Any time we make a mistake, we think and perform experiments. (MQ+MB)
- Goal is not to have to do this too many times. For our algorithm, total # mistakes will be at most # states.

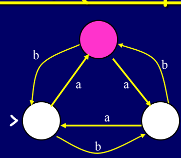
Algorithm by Dana Angluin

(with extensions by Rivest & Schapire)

- To simplify things, let's assume we have a RESET button. [Back to basic DFA problem]
- Can get rid of that using something called a "homing sequence" that you can also learn.

The problem (recap)

- We have a DFA:



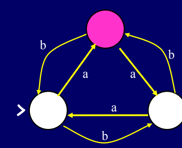
  - *observation = f(current state)*
  - *next state = g(button, prev state)*
- Can feed in sequence of actions, get observations. Then resets to start.
- Can also propose/field-test model. Get counterexample.

Key Idea

Key idea is to represent the DFA using a state/experiment table.

experiments

		λ	a
states	λ	□	■
	a	■	□
	b	□	□
trans-itions	aa	□	□
	ab	□	■
	ba	□	■
	bb	■	□



Key Idea

Key idea is to represent the DFA using a state/experiment table.

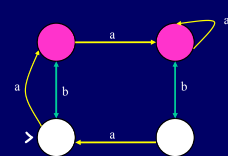
experiments

		λ	a
states	λ	□	■
	a	■	□
	b	□	□
trans-itions	aa	□	□
	ab	□	■
	ba	□	■
	bb	■	□

Guarantee will be: either this is correct, or else the world has > n states. In that case, need way of using counterexs to add new state to model.

The algorithm

We'll do it by example...



Algorithm (formally)

Begin with $S = \{\lambda\}, E = \{\lambda\}$.

1. Fill in transitions to make a hypothesis FSM.
2. While exists $s \in SA$ such that no $s' \in S$ has $row(s') = row(s)$, add s into S, and go to 1.
3. Query for counterexample z.
4. Consider all splits of z into (p_i, s_i), and replace p_i with its predicted equivalent $\alpha_i \in S$.
5. Find $\alpha_i r_i$ and $\alpha_{i+1} r_{i+1}$ that produce different observations.
6. Add r_{i+1} as a new experiment into E. go to 1.