

Machine Learning: Learning finite state environments



Avrim Blum
15-451 lecture 12/07/06

Machine Learning

A big topic in Computer Science. We'd like programs that learn with experience.

- Because it's hard to program up complicated things by hand.
- Want software that personalizes itself to users needs.
- Because it's a necessary part of anything that is going to be really intelligent.

What ML can do

- Learn to steer a car.  Pomerleau NHAA
- Learn to read handwriting, recognize speech, detect faces.  Schneiderman Kanade
- Learn to play backgammon (best in world).
- Identify patterns in databases.

Generally, program structure developed by hand. Learning used to set (lots of) parameters. ML as programmer's assistant.

More conceptually...

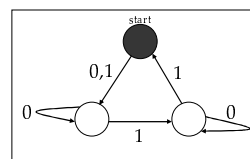
- Can we use CS perspective to help us understand what learning is?
 - Think about learning as a computational task just like multiplying?
 - How does a baby learn to figure out its environment? To figure out the effect of its actions?
- Lots of parts to all this. Today: one problem that captures some small piece of it.

Imagine...

- Say we are a baby trying to figure out the effects our actions have on our environment...
- Sometimes actions have effects we can notice right away, sometimes effects are more long-term.

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.

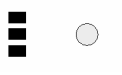


← Actions 0 and 1.
Observations white or purple.

Learning a DFA

Another way to put it:

- We have a box with buttons and lights.



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

Learning DFAs



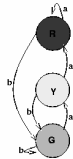
This seems really hard. Can't tell for sure when world state has changed.

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)

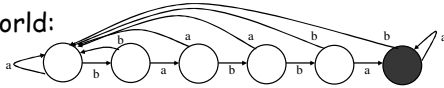
Can we design a procedure to do this in general?

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

Our model:



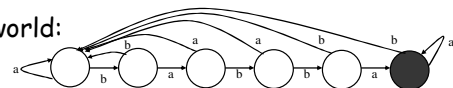
Real world:



Called "combination-lock automaton"

Can we design a procedure to do this in general?

Real world:



Called "combination-lock automaton"

This is a serious problem. It means we can't hope to efficiently come up with an exact model of the world from just our own experimentation.

How to get around this?

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

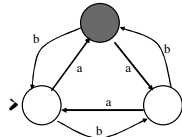
Today: a really cool algorithm by Dana Angluin

(with extensions by R.Rivest & R.Schapire)

- To simplify things, let's assume we have a RESET button.
- If time, we'll see how to get rid of that.

The problem (recap)

- We have a DFA:

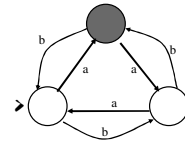


- $observation = f(current\ state)$
- $next\ state = g(button, prev\ state)$
- Can feed in sequence of actions, get observations. Then resets to start.
- Can also propose/field-test model. Get counterexample.

Key Idea

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

		experiments	
		λ	a
states	λ	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	a	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	b	<input type="checkbox"/>	<input type="checkbox"/>
transitions	aa	<input type="checkbox"/>	<input type="checkbox"/>
	ab	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	ba	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	bb	<input checked="" type="checkbox"/>	<input type="checkbox"/>



Either $aa=b$ or else aa is a totally new state and we need another expt to distinguish.

Key Idea

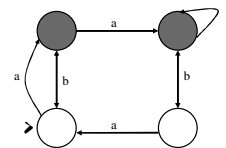
Key idea is to represent the DFA using a state/experiment table.

		experiments	
		λ	a
states	λ	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	a	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	b	<input type="checkbox"/>	<input type="checkbox"/>
transitions	aa	<input type="checkbox"/>	<input type="checkbox"/>
	ab	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	ba	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	bb	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Guarantee will be: either model 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.

Summary / Related problems

- All states look distinct: easy.
- Not all look distinct:
 - can do with counterex.
- All distinct but probabilistic transitions?
 - Markov Decision Process(MDP) / Reinforcement Learning.
 - Usual goal: maximize discounted reward (like probabilistic shortest path). DP-based algs.
- Not all distinct & probabilistic transitions?
 - POMDP. hard.