6.858/18.428 Machine Learning	Lecture 23: December 5, 1994
Lecturer: Ron Rivest	*Scribe: David Baggett*

Outline

This lecture discusses Dana Angluin's $L^\star$ algorithm for learning finite automata using membership and equivalence queries.

23.1 Finite automata

A deterministic finite automaton, or DFA, is a machine comprised of a set of states, a transition function, an alphabet, a start state, and a set of final states. Machine M runs on input w by examing each character of the input and traversing the arc out of the current state that is labeled with the input character. If this computation leaves the machine in a final state, the machine is said to accept w; otherwise, it rejects w.

Finite automata recognize a class of languages known as the *regular languages.* There is an extensive literature on this class of languages; see [2], [4], [5] for introductory material.

23.2 Active learning

Unfortunately, finite automata are probably not efficiently learnable within the PAC framework given certain cryptographic assumptions (see Kearns and Vazirani [3], chapter 7).

Hence it would seem that we need to give our learner more information. In particular, we'll give the learner a teacher who will answer two types of questions — membership queries ("is w accepted by the machine?") and equivalence queries ("is M equivalent to the machine?"). Furthermore, the teacher will give the learner a counterexample that proves that the learner's hypothesis is wrong with every negative equivalence query response. One requirement we'll add to the PAC model when using membership and equivalence queries is *exact* learning — the learner must get the DFA exactly right, not just correct within some given error bound.

Membership queries alone do not give efficient learning algorithms for DFAs. A simple example demonstrates this: consider trying to learn a "combination lock" DFA with one straight path from start to final state labeled with the "combination," and where all other transitions lead to the start state. The only way to learn such a DFA with membership queries is to get the whole combination right in a single query.

Angluin gives an algorithm for learning DFA's with membership and equivalence queries. This algorithm runs in time polynomial in the size of the minimal DFA equivalent to the target DFA and in the size of the longest counterexample the teacher provides. Note that equivalence queries do not place undue burden to the teacher, since determining whether two finite automata are equivalent (and giving a counterexample if they are not equivalent) takes polynomial time. Angluin [1] actually shows that within a probabilistic framework, a random sampling oracle can be used in place of the teacher for the equivalence queries.

In today's lecture, we assume that the teacher resets the target automaton before answering any membership queries. While this may at first seem like a trivial concern, learning in the absence of this feature is considerably more complicated, as we will see in the next lecture.

23.3 A general overview of the $L^\star$ algorithm

We maintain a table with rows corresponding to states and columns corresponding to experiments (input strings). We can think of the states as being named by the input strings over alphabet A that take the machine from the start state to the states themselves. Since we wish to find the smallest equivalent machine, we need to be able to recognize when two states are the same — we do this by comparing the rows corresponding to the states to be compared. If the rows are different, it means that there is some input that takes the machine from the two states to two different states, and hence the corresponding states can't be the same.

We fill in a table cell by taking the string for the cell's row, concatenating it with the string for the cell's column, and making a membership query on the compound — we store the yes/no answer to the query in the cell. Our goal is to have the experiments distinguish the states — i.e., to have some experiment (column) to differentiate any given pair of distinct states. Angluin calls the set of row labels (which is prefix-closed) S and the set of column labels (which is suffix-closed) E.

Aside from the rows corresponding to the states of our hypothesis DFA, we'll need another set of rows. Specifically, we need to have row for each hypothesis state concatenated with each alphabet symbol (i.e., $S \cdot A - S$). So given the following table:

T	λ	0
λ		
0		
1		
11		

We augment the table as follows:

T	λ	0
λ		
0		
1		
11		
00		
01		
10		
110		
111		

Note that there is a row in the bottom part of the table for each row in the top part of the table concatenated onto each letter of the alphabet (here $\{0, 1\}$). We do not put strings that are already in the top part into the bottom part (e.g., $1 \cdot 1 \Rightarrow 11$ in the example above). We call the rows in the bottom part of the table *successors*.

Some terminology will be useful in subsequent explanation. We say that a table is *closed* provided that every row in the bottom part of the table has a corresponding row in the top part of the table with identical numbers in all the columns. Intuitively, a table is closed when none of the rows in the bottom part could correspond to a new state given all the experiment strings we have so far.

For example, the following table is not closed, since row 0 is distinct from row λ:

T	λ
λ	1
0	0
1	0

Similarly, we say a table is *consistent* if every pair of rows in the top part of the table with identical experiment results (columns) also has identical experiment results when any alphabet symbol is added. (I.e., $(\forall s_1, s_2 \in S)$ if $\text{row}(s_1) = \text{row}(s_2)$ then $(\forall a \in A)\ \text{row}(s_1 \cdot a) = \text{row}(s_2 \cdot a)$.)

For example, the following table is *not* consistent, because row 0 = row 1 but row 00 $\neq$ row 10:

T	λ
λ	1
0	0
1	0
11	1
00	1
01	0
10	0
110	0
111	0

Intuitively, if a table is consistent, we know that any pair of states that we think are the same are still the same if we extend every experiment string with a single alphabet character.

23.3.1 The basic procedure

The basic idea behind L* is to repeatedly conjecture machines until the teacher tells us we've found an equivalent machine. We derive the conjectured machine from our current table as follows:

- Assign a unique state to each row in the top part of the table.
- Label as the start state the row labeled λ (i.e., corresponding to the empty string as input).
- Label as final any state whose λ column has a one in it (i.e., any row whose row label alone is accepted by the target machine).
- Make a transition from the state corresponding to row s for alphabet symbol a to the state corresponding to row $s \cdot a$ (s with a concatenated).

Before we can query the teacher about our machine, we have to ask membership queries to make the table closed and consistent. So while the table is not consistent, we keep picking inconsistent rows; i.e., two rows that agree in all columns, but that differ when alphabet symbol a is added. If s_1 and s_2 look the same (i.e., their rows are identical), but $s_1 \cdot a$ and $s_2 \cdot a$ don't look the same, then there must be some experiment e in the table (i.e., some column) that differentiates them. So then experiment $a \cdot e$ differentiates s_1 and s_2, so we add it to the table.

Similarly, we have to ensure that the table is closed before we can make an equivalence query. So while the table is not closed, we find a row label s and an alphabet a symbol such that the column values in the row labeled $s \cdot a$ differ from every other row in the top part of the table. We then move the row labeled $s \cdot a$ to the top part of the table, add the successors to the bottom part of the table (if necessary), and fill in the new columns by making membership queries.

Once we have a closed and consistent table, we make an equivalence query on the corresponding DFA. If the teacher answers "yes" then we're done. Otherwise we make a new row in the top part of the table for the teacher's counterexample *and all its prefixes*. We then fill out the table using membership queries and go back to ensuring consistency and closure again.

23.3.2 An example

Here we work through an example — the learner will learn a finite automaton over $A = \{0, 1\}$ that accepts any string with an even (or zero) number of zeros and an even (or zero) number of ones.

Initially, we assume just one state. We first ask membership queries for λ, 0, and 1. This gives us the following table:

T_1	λ
λ	1
0	0
1	0

This table is consistent but not closed, so we move the 0 row to the top of the table (S) and query its successors 00 and 01 to build T_2:

T_2	λ
λ	1
0	0
1	0
1	0
00	1
01	0

This table is closed and consistent, so we conjecture the machine described by the following table, where $q_{[1]}$ is the start state, and also a final state:

	0	1
$q_{[1]}$	$q_{[0]}$	$q_{[0]}$
$q_{[0]}$	$q_{[1]}$	$q_{[0]}$

The teacher responds with the counterexample 11. We incorporate the counterexample into the table by adding 11 and all its prefixes to the top of the table. Since λ is already in the top of the table, this means we add a row for 1 and a row for 11. We then make membership queries on 10, 110, and 111 to get the following table:

T_3	λ
λ	1
0	0
1	0
11	1
00	1
01	0
10	0
110	0
111	0

This table is closed but not consistent (consider rows 0 and 1, and successors 00 and 10), so we add a 0 column (i.e., we add 0 to E) and make membership queries on 000, 010, 100, 1100, and 1110 to get the following table:

T_4	λ	0
λ	1	0
0	0	1
1	0	0
11	1	0
00	1	0
01	0	0
10	0	0
110	0	1
111	0	0

Once again we have a closed and consistent table, and conjecture the machine described as follows, where state $q_{[10]}$ is the start state, and is also a final state:

	0	1
$q_{[10]}$	$q_{[01]}$	$q_{[00]}$
$q_{[01]}$	$q_{[10]}$	$q_{[00]}$
$q_{[00]}$	$q_{[00]}$	$q_{[10]}$

The teacher responds with the counterexample 011. We again add the counterexample and all its prefixes to S, the top of the table — in this case the prefixes 01 and 011 are the only ones not already in the table. To fill out the augmented table, we make membership queries on 0110, 0100, 01100, 0111, and 01110, to get table T_5:

T_5	λ	0
λ	1	0
0	0	1
1	0	0
11	1	0
01	0	0
011	0	1
00	1	0
10	0	0
110	0	1
111	0	0
010	0	0
0110	1	0
0111	0	0

T_5 is closed but not consistent (consider row 1 and row 01, and their successors 11 and 011), so we add a 1 column (i.e., add 1 to E) and query the strings 001, 101, 1101, 1111, 0101, 01101, 01111 to get table T_6:

T_6	λ	0	1
λ	1	0	0
0	0	1	0
1	0	0	1
11	1	0	0
01	0	0	0
011	0	1	0
00	1	0	0
10	0	0	0
110	0	1	0
111	0	0	1
010	0	0	1
0110	1	0	0
0111	0	0	0

Since this table is closed and consistent, we conjecture the corresponding machine, where $q_{[100]}$ is the start state and also a final state:

	0	1
$q_{[100]}$	$q_{[010]}$	$q_{[001]}$
$q_{[010]}$	$q_{[100]}$	$q_{[000]}$
$q_{[001]}$	$q_{[000]}$	$q_{[100]}$
$q_{[000]}$	$q_{[001]}$	$q_{[010]}$

The teacher confirms this conjecture, so we terminate and output the machine.

23.3.3 Pseudocode for $\mathbf{L}^\star$

Angluin [1] provides pseudocode for the algorithm; we reproduce it here since it is fairly readable. In Angluin's notation, S refers to the set of row labels, E refers to the set of experiment strings (column labels), T is a function on a string that returns 1 if the string is accepted by the target machine and 0 otherwise, and A is the set of alphabet symbols. The notation row(s) refers to the list of column values for row s. (Technically, Angluin defines this a function.) Finally, $M(\ldots)$ denotes the machine corresponding to the table.

Initialize S and E to $\{\lambda\}$.
Ask membership queries for λ and each $a \in A$.
Construct the initial observation table (S, E, T).

Repeat:
    While (S, E, T) is not closed or not consistent:
        If (S, E, T) is not consistent,
            then find s_1 and s_2 in S, $a \in A$, and $e \in E$ such that
            $\text{row}(s_1) = \text{row}(s_2)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$,
            add $a \cdot e$ to E,
            and extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.

        If (S, E, T) is not closed,
            then find $s_1 \in S$ and $a \in A$ such that
            $\text{row}(s_1 \cdot a)$ is different from $\text{row}(s)$ $\forall s \in S$,
            add $s_1 \cdot a$ to S,
            and extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.

    Once (S, E, T) is closed and consistent, let $M = M(S, E, T)$.
    Make the conjecture M.
    If the teacher replies with a counterexample t, then
        add t and all its prefixes to S
        and extend T to $(S \cup S \cdot A) \cdot E$ using membership queries.
Until the teacher replies *yes* to the conjecture M.
Halt and output M.

23.4 Conclusions

We have yet to prove that the algorithm that the algorithm is correct and converges quickly. We will address these issues in the next lecture. We'll also look at learning finite automata without a reset.

A final word: the algorithm given in chapter 8 of the text [3] is identical in spirit but somewhat more cleverly implemented.

References

[1] Dana Angluin. "Learning Regular Sets from Queries and Counterexample," *Information and Computation* **75**, pp. 87-106, 1987.

[2] John Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wessley, 1979.

[3] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1992.

[4] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

[5] Dominique Perrin. "Finite automata," *Handbook of Theoretical Science, Volume B* (J. van Leeuwen, ed.) MIT Press, 1994.