CS 395T Computational Complexity of Machine Learning	
Lecture 20: April 4, 2005	Lecturer: Adam Klivans
Scribe: Lily Mihalkova	

20.1 Angluin's Algorithm and Applications

The following result due to Angluin [1] was stated in the previous lecture:

Theorem 1 *Deterministic finite state automata (DFAs) of size t are learnable in the exact model with membership and equivalence queries in time polynomial in t.*

We saw how this fact can be used to learn $O(\log n)$-term DNFs in polynomial time via DFAs. Here we present a second application—learning intersections of halfspaces via DFAs. In addition, we provide a sketch of Angluin's algorithm.

20.1.1 Learning intersections of halfspaces via DFAs

Recall that a halfspace $f(x)$ is defined as

$$f(x) = sign(\sum_{i=1}^{n} \alpha_i \chi_i - \theta).$$

Theorem 2 *Intersections of k halfspaces can be learned in the exact model with membership and equivalence queries via DFAs in time polynomial in $(2nW)^k$, where W is an upper bound on the weight of any particular halfspace; i.e. $\sum_{i=1}^{n} |\alpha_i| + |\theta| < W$.*

Note that when $W = n^{O(1)}$ and $k = O(1)$, this gives an algorithm that is polynomial in n.

To prove this theorem, we will use two lemmas:

Lemma 1 *Let f and g be DFAs with a and b states respectively. Then, we can construct a DFA for $f \wedge g$ that has $a \cdot b$ states.*

Proof: The proof describes how to construct the new DFA e that computes $f \wedge g$. Let the set of states of f be $F = \{F_1, F_2, \ldots, F_a\}$ and that of g be $G = \{G_1, G_2, \ldots, G_b\}$. The set of states of e will be $E = F \times G$. Thus $|E| = a \cdot b$. The initial state of e is $S_{init} = (F_{init}, G_{init})$. Now suppose that the current state is (F_i, G_j) and the next symbol is c. In this case e transitions to the state (F_k, G_l) where F_k is reached in f from F_i upon observing c and G_l is defined analogously. Thus e simulates f and g simultaneously. The set of accepting states of e consists of all states (F_i, G_j) where F_i is accepting in f and G_j is accepting in g. Therefore, e will result in an accepting state if and only if both the simulation of f and the simulation of g result in an accepting state. Thus, e is a DFA for $f \wedge g$ that has $a \cdot b$ states. ■

Lemma 2 *A halfspace with weight W can be computed by a DFA of size* $(2W + 1)n + 1$.

Proof: The desired DFA will have $(2W + 1)n + 1$ states. For ease of exposition, imagine that all states except for the initial one are arranged in rows 1 through n, each containing $2W + 1$ states. The states in a given row correspond to the numbers in $[-W, W]$ and are arranged accordingly. The initial state is placed in row 0 above the state corresponding to 0 in the first row. The transition function is defined as follows. If the i-th input bit is 0, the DFA transitions to the state directly below the current state (i.e. the one that is in the same column but in row i). If the i-th input bit is 1, the DFA transitions to a state on row i that is $|\alpha_i|$ columns over to the right (left) of the current column if $\alpha_i \geq 0$ ($\alpha < 0$). The accepting states are those to the right of the state corresponding to θ in row n. ■

We are now ready to prove Theorem 2.

Proof: Combining the results of Lemmas 1 and 2, we get that an intersection of k halfspaces, each of weight at most W, can be computed by a DFA of size $((2W + 1)n + 1)^k$. From Theorem 1 it follows that the intersection of k weight-W halfspaces can be learned in time polynomial in $(2Wn)^k$.
■

20.1.2 Sketch of Angluin's algorithm

This section provides a sketch of Angluin's algorithm for learning DFAs. We will use the following notation:

- A is the alphabet over which the unknown DFA is defined
- S is a set of strings that contains the prefixes
- E is a set of strings that contains the suffixes
- $S \cdot A = \{s \cdot a | s \in S \text{ and } a \in A\}$ where $\cdot$ is the concatenation operator.

The algorithm keeps an observation table T in which the rows are indexed by strings from $S \cup S \cdot A$, and the columns are indexed by strings from E (i.e. suffixes). For $s \in S \cup S \cdot A$ and $e \in E$, $T(s, e)$ contains the value given by the automaton on the string $s \cdot e$ (1 if the string was accepted and 0 otherwise). Initially, both S and E contain only the empty string: $S = \{\epsilon\}$ and $E = \{\epsilon\}$. For example, if the DFA computes the regular language ab^* over the alphabet $\{a, b\}$, initially the table will be

	ϵ
ϵ	0
a	1
b	0

Definition 1 *A table is* ***closed*** *if for every $s \in S$ and every $a \in A$, there exists a $t \in S$ such that $row(t) = row(s \cdot a)$.*

Definition 2 *A table is* ***consistent*** *if for any $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$, we have $row(s_1 \cdot a) = row(s_2 \cdot a)$.*

Using the above definitions, the algorithm proceeds as follows:

Until the oracle says "Exactly correct," do:

1. Check if T is consistent. If it is not, we found strings s_1, s_2, and e and symbol a, such that $row(s_1) = row(s_2)$ and $T(s_1 \cdot a, e) \neq T(s_2 \cdot a, e)$. In this case, add $a \cdot e$ to E (in other words, add a column to T and fill in this column using membership queries). Doing this corrects the inconsistency because now $row(s_1) \neq row(s_2)$ because the unknown DFA gives two different results when run on the strings $s_1 \cdot a \cdot e$ and $s_2 \cdot a \cdot e$.
2. Otherwise, check if T is closed. If it is not, we found a string $s \in S$ and a symbol $a \in A$ such that $row(s \cdot a) \neq row(t)$ for every $t \in S$. In this case, add $s \cdot a$ to S, hence adding a new row for each symbol in A. Fill those rows in using membership queries.
3. Otherwise, if T is both closed and consistent, ask the teacher whether the current hypothesis is correct. If it is not, the learner receives a counterexample c. If this is the case, add c and all of its prefixes to S. Again, the new rows are filled in using membership queries.

In the last step of this algorithm, the learner needs to convert its table to a DFA. This is done as follows. The set of states of the DFA consists of all rows $row(s)$ where $s \in S$. Thus, if for two strings $s_1, s_2 \in S$, $row(s_1) = row(s_2)$, then $row(s_1)$ and $row(s_2)$ map to the same state. The initial state corresponds to $row(\epsilon)$ (i.e. the first row in the table). Because of how T is initialized, the state corresponding to $row(\epsilon)$ always exists. Let the current state correspond to $row(s)$ for $s \in S$ and let the current symbol be a. Then the transition function is defined as:

$$\delta(row(s), a) = row(s \cdot a) = row(t)$$

Because T is closed and consistent, this transition function is well defined. Since T is closed, there exists some $t \in S$ that satisfies the second equality above. Therefore, the DFA will contain a state that corresponds to $row(t)$. Consistency guarantees that for any $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$, $row(s_1 \cdot a) = row(s_2 \cdot a)$. This is important in order to ensure that the transition function is deterministic—because $row(s_1) = row(s_2)$, these two rows map to the same state in the DFA and consistency guarantees that the transition function specifies a single next state for any particular symbol.

The accept states are those corresponding to $row(s)$ for all $s \in S$ such that $T(s, \epsilon) = 1$.

If the algorithm halts, then the automaton will be correct. To prove that this happens before too many rows and columns are added to the table, we use the fact that this algorithm constructs the minimal DFA. The detailed proof of this fact can be found in Angluin's paper [1]. A discussion of the proof is also available in [2].

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. Information and Computation, 75:87-106, November 1987.

[2] Sally Goldman (lecturer) and Ellen Witte (scribe). Topic 13: Learning regular sets. Can be found at http://userfs.cec.wustl.edu/ cs582/scribe-notes.html.