Learning Regular Sets

Qingjun Wu (Wade, qiwu5520@colorado.edu)

May 2017

Abstract

This paper is inspired by Learning Regular Sets from Queries and Counterexamples by DANA ANGLUIN[1] and is the final project of class Computation Theory. The main contribution of this paper includes comprehensive understanding of the original , providing more comprehensive proofs of the lemmas in the paper, implementing the original algorithm using JavaScript, running the algorithm using a different example, researching on applying the algorithm to different languages like Context-Free Grammar, Structually Reversible Context-Free Grammar, Very Simple Grammar, and also parallel algorithm.

1 Introduction

Learning general rules from random or arbitrary examples has been well studied before ANGLUIN's paper[1]. Gold[5] had shown that the problem of finding a DFA of a minimum number of states compatible with a given set of positive and negative examples in NP-hard. In this paper ANGLUIN makes an assumption that there's a Minimally Adequate Teacher (MAT) that can correctly answer two kinds of questions in polynomial time: it executes membership query to decide if an element is in the regular set; and it can also verify equivalence between a conjecture given by Learner algorithm and this is under the hypothesis the Teacher has already known what the unknown regular set is. The details of the method including its Teacher and Learner algorithms (especially the latter) will be elaborated below. A different example of regular set are also used to run the algorithm. Please note that this is under hypothesis that the Teacher knows the regular set and is able to answer question in polynomial time. But in the really world this is not true, the time for membership query is NP-Hard like we mentioned above. The author provided an approximation algorithm makes polynomial number of calls to a random sampling oracle to return a counterexample or none.

2 Algorithm

ANGLUIN's algorithm includes two roles: Teacher and Learner. And each role has its own algorithm.

2.1 Teacher

Assumably the Minimally Adequate Teacher(MAT) is a pair of two oracles: MEMBER(x) which executes membership query to decide if x in the unknown regular set and EQUIV(M) which decide if conjecture M is equivalent to the hypothesis of the regular set that the Teacher has already known.

- MEMBER(x) is straight forward. One of the solution for MEMBER(x) is that, since the Teacher is assumed to know the regular set, or it has got a predefined DFA. Starting for the initial state and each step takes one symbol from x. If the last state is in final states of the DFA, returns true otherwise false. But there's still another solution: a collection of representative samples.

- EQUIV(M) The Hopcroft-Karp algorithm[7, 4] can be used to verify equivalence of two Finite Automatons (FA). The basic idea is: Starting from initial state pair $(q_0, q_0^{'})$, both FAs transitions to next states $(q_n, q_n^{'})$ in parallel, if q_n and $q_n^{'}$ are not in one set, merge them. If one set has both final states and non-final states, the two FAs are not equivalent.

2.2 Learner

Observation Table

Observation Table is the heart of ANGLUIN's Leaner algorithm. Observation Table can be constructed using prefix table S, suffix table E and membership query T, so let's denote observation table as (S, E, T), where S is the rows, E is the columns of the table, and T is membership query defined as *T(u)* if u is in the unknown regular set returns 1 else 0.

The Observation Table (S, E, T) exposes two important properties:

- CLOSED

  An observation table (S, E, T) is called closed provided that

  for $\forall t \in S \cdot A$, $\exists s \in S$, s.t., $row(t) = row(s)$

- CONSISTENT

  An observation table is called consistent provided that

  $\forall s_1 \in S,\ \forall s_2 \in S, if\ row(s_1) = row(s_2)$

  $\forall a \in A\ \ row(s_1, a) = row(s_2, a)$

Acceptor

If observation table (S, E, T) is both closed and consistent, we can define a corresponding acceptor $M(S, E, t)$:

$Q = \{row(s) : s \in S\}$

$q_0 = row(\lambda)$

$F = \{row(s) : s \in Sand\ T(s) = 1\}$

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

Acceptor $M(S, E, T)$ exposes very important property defined in THEROM 1.

THEOREM 1. *M(S, E, T) is consistent with T, and also M is minimum acceptor.*

This theorem is approved by a sequence of lemmas.

- LEMMA 2. *Assume that (S, E, T) is a closed, consistent observation table. For the acceptor* $M(S, E, T)$ *and* $\forall s \in (S \cup S \cdot A)$, $\delta(q_0, s) = row(s)$

  Proof: This lemma is approved by induction on length of s *len(s)*.

  Base case: if*len(s)* = 0, i.e., $s = \lambda$, by definition of acceptor M, $q_0 = row(\lambda)$ and $\delta(q_0, \lambda) = q_0$, it shows that $\delta(q_0, s) = row(s)$.

  Assume $\forall s \in (S \cup S \cdot A)$, and $len(s) \leq k$, $\delta(q_0, s) = row(s)$

  And let $t \in (S \cup S \cdot A)$, and $t = s \cdot a, where a \in A$ for some $s \in S$, then

  $\delta(q_0, t) = \delta(\delta(q_0, s), a)$
  $= \delta(row(s), a)$
  $= row(s \cdot a)$
  $= row(t)$.

  Proved.

  This lemma shows that acceptor $M(S, E, T)$ is complete or closed.

- LEMMA 3. $M(S, E, T)$ is consistent with T. That is $\forall s \in (S \cup S \cdot A) and \forall e \in E, \delta(q_0, s \cdot e) \in F\ iff\ T(s \cdot e) = 1$.

  Proof:

  This is also proved by induction of length of e.

  Base case: if $e = \lambda$, by LEMMA 2, $\delta(q_0, s \cdot e) = row(s)$. If $s \in S$, by definition of F, then $row(s) \in F\ iff\ T(s) = 1$. If $sinS \cdot A$, since observation table is closed, $\exists s_1 \in S$, such that $row(s) = row(s_1)$. And $row(s) \in F\ iff\ T(s) = 1$.

  Induction step: Supposed $\forall e \in E, andlen(e) \leq k, \delta(q_0, s \cdot e) \in FiffT(s) = 1$. And then $lete \in E$, and $|e| = k + 1$, since observation table is closed, then $\exists a \in A, \exists e_1 \in E$ and $|e_1| \leq k, e = a \cdot e1$.

  $$\delta(q_0, s \cdot e) = \delta(\delta(q_0, s), a \cdot e_1) \tag{1}$$

  And based on LEMMA 2,

  $$\delta(q_0, s \cdot e) = row(s) \tag{2}$$

  Since observation table is closed, $\exists s_1 \in S$ such that

  $$row(s_1) = row(s) \tag{3}$$

Combine (1), (2), (3), it shows that $\delta(q_0, s \cdot e) =$
$\delta(row(s_1, a \cdot e_1)) =$
$\delta(\delta(row(s_1), a), e_1)$, by definition of δ
$= \delta(\delta(q_0, s_1 \cdot a), e_1)$
$= \delta(q_0, s_1 \cdot a \cdot e_1)$
$\delta(q_0, s_1 \cdot a \cdot e_1) \in F iff T(s_1 \cdot a \cdot e_1) = 1$
Since $row(s) = row(s_1), and a \cdot e_1 = e \in E$, it shows that

$$T(s_1 \cdot a \cdot e_1) = T(s \cdot a \cdot e_1) = T(s \cdot e) \tag{4}$$

Which shows $\delta(q_0, s \cdot e) \in F\ iff\ T(s \cdot e) = 1$ as claimed.
Proved.

- LEMMA 4. If there's another acceptor $M^{'} = (Q^{'}, q_0^{'}, F^{'}, \delta^{'})$ with fewer or equal number of states than $M(S, E, T)$, then $M^{'}$ is isomorphic to $M(S, E, T)$ and $M^{'}$ has exactly the number of states as $M(S, E, T)$.
  Based on LEMMA 3, $\forall s \in (S \cup S \cdot A), and \forall e \in E, \delta^{'}(q_0^{'}, s \cdot e) \in F^{'} iff T(s \cdot e) = 1$,
  it shows that $row(\delta^{'}(q_0^{'}, s)) = row(s)$. so $M^{'}$ has exactly the same number of states as $M(S, E, T)$.
  So there's one-to-one mapping from $M^{'}$ to M. We define a mapping function $\phi(row(s)) = \delta^{'}(q_0^{'}, s)$.
  For $q_0 and q_0^{'}$, it shows that

$$\phi(q_0) = \phi(row(\lambda)) = \delta^{'}(q_0^{'}, \lambda), by\ definition\ of\ \delta^{'} = q_0^{'} \tag{5}$$

  Which means q_0 can be mapped to $q_0^{'}$.
  And $\forall s \in S$, let $s_1 \in$ such that $row(s \cdot a) = row(s_1)$, it shows

$$\phi(\delta(row(s), a)) = \phi(row(s \cdot a)) = \phi(row(s_1)) = \delta^{'}(q_0^{'}, s1) \tag{6}$$

  And also,

$$\delta^{'}(\phi(row(s), a)) = \delta^{'}(\delta^{'}(q_0^{'}, s), a) = \delta^{'}(q_0^{'}, s \cdot a) \tag{7}$$

3 Correctness

Let n be different values of $row(s)\ for\ s \in S$, E.G., $row(s_1)! = row(s_2),\ s_1, s_2 \in S$. According to LEMMA 2, $\delta(q_0, s_1) \neq \delta(q_1, s_2)$. So acceptor $M(S, E, T)$ has at least n states.

And note that n is also the number of states of minimum acceptor. In the Learner's algorithm, at each iteration, if the observation is not closed or consistent, the distinct number of $row(s)$ will be increased at least by one. So the algorithm will eventually terminate. And the maximum number of acceptors generated by the algorithm is also n.

4 Complexity

The complexity of the Learn's algorithm depends on S and E. The cardinality of E is n, where n is number of different states of S; E is increased by ≤ 1 at each iteration of the algorithm. The cardinality of S is $n + m(n-1)$ where $m = maximumlengthofs, s \in S$. $\forall t \in S \cup S \cdot \Sigma maximumlengthoft = maximumlengthofs + maximumlengthofe, wheres \in S, e \in E$. Put all there together, the total complexity is $O(m^2n^2 + mn^3)$.

5 Approximation

So far the algorithm is under assumption that the Teacher knows the pattern of the regular set and can query membership/equivalence in polynomial time and thus gain polynomial time complexity in total. But in the real world, the unknown regular set is really completely UNKNOWN. But the Learner can still obtain polynomial time complexity by substituting the EQUIV(x) with a random sampling oracle $EX()$ which returns a pair (t, d) at a time. If $d = yes$ and t is rejected by acceptor $M(S, E, T)$ or vice versa, then t is counterexample, and the Learner will continue to update acceptor like the old Learner algorithm. The new Leaner makes polynomial time of calls to $EX()$, if none of this calls return counterexample, the algorithm terminates.

6 Examples

Let's run an example different from ANGLUIN's paper. Define regular language :
$L = \{w \in \{a, b\}^* : thenumberofb^{'}sinwiscongruentto3modulo4\}$
$\Sigma = \{0, 1\}$
Initially $S = [\lambda]$, and $E = [\lambda]$, and ask membership queries for λ and $\forall a \in \Sigma$, we gain the below observation table (S, E, T):

T_1	λ
λ	0
a	0
b	0

This table is both closed and consistent, and we can generate the acceptor $M1(S, E, T)$ like table below:

δ_1	a	b
q_0	q_0	q_0

The Teacher will execute equivalence query against the acceptor $M1(S,E,T)$. It rejects $M1$ and returns a counter example *bbb*. The Learner algorithm will add prefixes of *bbb* into S: λ, *b*, *bb*, *bbb*. Since λ and *b* are already in S, only *bb* and *bbb* will be added into S, and T will be extended to include *bba*, *bbba*, *bbb* and *bbbb*, and it shows observation table $M2(S,E,T)$ as

T_2	λ
λ	0
a	0
b	0
bb	0
bbb	1
ba	0
bba	0
bbba	1
bbba	0

This observation table is not consistent, since $row(b) = row(bb)$, but $row(b \cdot b) \neq row(bb \cdot b)$. So b is added into E, and execute membership queries to extend T, it shows

T_3	λ	b
λ	0	0
a	0	0
b	0	0
bb	0	1
bbb	1	0
ba	0	0
bba	0	1
bbba	1	0
bbbb	0	0

This table is closed but not consistent, since $row(a) = row(b)$, but $row(a \cdot b) \neq row(b \cdot b)$. And the current $E = \{\lambda, b\}$, so both b and bb will be added into E, but since b is already included in E, so only bb is added, and membership queries are also executed to extend T, it shows

T_4	λ	b	bb
λ	0	0	0
a	0	0	0
b	0	0	1
bb	0	1	0
bbb	1	0	0
ba	0	0	1
bba	0	1	0
bbba	1	0	0
bbbb	0	0	0

Now this table is both closed and consistent, and acceptor $M^2(S, E, T)$ is like below table.

δ_1	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_3
q_3	q_3	q_0

And also in this table, $q_3 \in F$ since $q_3 = row(bbb)$, and $T(bbb) = 1$.

7 Implementation

In ANGLUIN's paper, pseudo code of the Learner's algorithm has been given. A JavasScript version of this algorithm is implemented while this write-up is being written.

Membership query MEMBER($transitionTable$, $finalStates$, x), which queries if element x is in the unknown regular set S using $transitionTable$ and $finalStates$, and Equivalence query EQUIV(dfa_1, dfa_2), which queries if the two DFA are equivalent, are not given by ANGLUIN's paper. So more details are given here.

7.1 MEMBER(x)

In my implementation, this function is taking three parameters:

- $transitionTable$, which is actually the acceptor $M(S, E, T)$, e.g., the δ_* table which presents the transition functions. It's also another format of DFA.
- $finalStates$, this is corresponding to F in the definition of DFA.
- x is the input element to be queried.

The code is quite straightforward: it starts with the q_0, and transition to next state according to each symbol in the input string sequentially. If it terminates in F, then return true else false.

```
function MEMBER(transitionTable, q0, F, symbols, x) {
    var state = q0;
    for (var i = 0; i < x.length; ++i) {
        for (var j = 0; j < symbols.length; ++j) {
            if (x[i] === symbols[j]) {
                state = transitionTable[state][j];
            }
        }
    }
    return F.has(state);
}
```

7.2 EQUIV(dfa_1, dfa_2)

Hopcroft-Karp Algorithm [7] is used to verify equivalence of two DFAs, pseudo code looks like this:

```
function EQUIV(dfa1, dfa2, Q1, Q2, q1_0, q2_0, F1, F2, symbols) {
    var allsets = [];
    for(let a of Q1.values()) {
        var s1 = new Set();
        s1.add(a);
        allsets.push(s1);
    }

    for(let a of Q2.values()) {
        var s1 = new Set();
        s1.add(a);
        allsets.push(s1);
    }

    var stack = [];
    stack.push([q1_0, q2_0]);
    while(stack.length > 0){
        var top = stack.pop();
        for (var i = 0; i < symbols.length; ++i) {
            var n1 = dfa1[top[0]][i];
            var n2 = dfa1[top[1]][i];
            var r1 = findSet(allsets, n1);
            var r2 = findSet(allsets, n2);

            if (r1 !== r2) {
                for (let e in allsets[r2].values()) {
                    allsets[r1].add(e);
                }
                allsets[r2].clear();
                var count = 0;
                for (let e in allsets[r1].values()) {
                    if (F1.has(e) || F2.has(e)) {
                        ++count;
                    }
                }
                if (count > 0 && count < F1.length + F2.length) return false;
                stack.push([n1, n2]);
            }
        }
    }
    return true;
}
```

7.3 Demo

The entire algorithm is implemented using JavaScript and the code is available in github [6].

8 Application and Related Work

ANGLUIN's algorithm has been widely applied to different languages and also further researched since it's published to the public. This paper is going to list some of them.

8.1 Context-Free Grammar

ANGLAUIN's also applied the algorithm against Context-Free Grammer (CFG) with modification. The Teacher still answers two types of questions.

- $MEMBER(x, A)$ determines whether x can be derived from A using rules from the unknown Context-Free Grammar, where x is a string and A is a non-terminal symbol.
- EQUIV(H) tests whether the conjectural Context-Free Grammar H generates the same terminal symbols as the unknown Context-Free Grammar. This is again an approximation. It's a undecidable problem to determine if two CFGs are equivalent.

The modified algorithm initially creates supper set of all the possible productions (rules), let's call it P, which is actually the first conjecture. At each iteration of the while loop, the algorithm will verify the equivalence of the two CFGs by calling EQUIV(H), if *yes* halts; otherwise, it finds a counterexample t, and remove the corresponding production (rule) from P.

The Learner algorithm firstly generates a parse tree for t, where the start symbol S is the root of the tree. And check each sub-tree if it is in G. If the answer is *yes*, it will recursively check the sub-tree of the current sub-tree; otherwise it will check its next sibling. Eventually the algorithm will one or more productions (rules) which are not existing in the unknown CFG G, and remove all these incorrect rules from P.

The algorithm will terminate with a final P equivalent to G.

8.2 Non-deterministic Finite Automata

YOKOMORI [8] adapted ANGLUIN's algorithm to make it work on Non-deterministic Finite Autmata (NFA).

The algorithm queries Minimally Adequate Teacher with the counterexample.

If the counterexample is in the unknown regular set, it is positive else negative. Positive counterexamples are used to generate new transition rules, while negative ones are used to remove incorrect transition rules.

- Generating new rules
  $Q(w) = \{row(x) | w = x \cdot y for some y \in \Sigma^*\}$
- Constructing new candidate transition rules
  $\delta_{new} = \{p- > q | p, q \in Q \cup Q(w), a \in \Sigma, and at least one of p, q \in Q(w)\}$

- Removing incorrect conjectured rules. This is due to the conjectured NFA may have some incorrect transition rules introduced by the previous step. These rules can be determined by negative counterexamples.

  The algorithm is following the below routine to remove incorrect transition rules:

  $Define transition Trans(M, p^{'}, w') : p^{'} - > p - > q, where q \in F, w^{'} = aw, a \in \Sigma, w \in \Sigma^{*} and p = row(y), y \in \Sigma^{*}$

  $if\ w^{'} = a, a \in \Sigma, transition rule r : p^{'} \xrightarrow{\mathrm{w}} q is incorrect and should be removed$ $else if yw \notin L recursively diagnose Tran(M, p, w); otherwise rule r : p^{'} \xrightarrow{\mathrm{a}} p is incorrect and needs be removed$

8.3 Very Simple Grammar

A Context Free Grammar is called *very simple* if the righthand side of each rule starts with a distinct terminal symbol. T. Yokomori [9] also applied ANGLUIN's algorithm on Very Simple Grammar and gained polynomial time complexity. By exploring the properties of Very Simple Grammar, the author developed an algorithm to generate grammar rules which are both closed and consistent. The input of the algorithm is a collection of positive examples. The conjectured grammar rules keep being updated by each input positive example.

8.4 Structually Reversible Context Free Grammar

In learning Context-Free Grammars (CFG), BURAGO [3] used positive examples to generate the rules.

The role of Minimally Adequate Teacher answers two questions:

- **Queries**: The Teacher tells if a string is in the language or not;
- **Conjecture**: If a conjectural grammar is equivalent to the hypothesis returns yes otherwise returns a counterexample.
- get a string $w \in L$. Let $E = E \cup w$ be the set of all string in L ;
- $\forall e \in E$, create two lists: a list of all substrings of all strings in E, e.g., $K = e[i, j] | e \in E, 0 < i \leq j \leq |e|$, and

  a list of positions where keys can be inserted, e.g.,

  $P = ((e[1 : i - 1], e[j + 1]) | e \in E, 0 < i \leq j \leq |e|)$.

  for key k, if $p_1 k p_2 \in L, p_1, p_2 \in P$, we say that key k matches position $(p_1 p_2)$. A key k is said to be **minimal** if non of its subkeys matches position $p, where p = (p1, P2), p1, p2 in \in P$.

  $Let P^{'} = \{p | k matches p, \forall k \in K\ , p \in P\}$.

  $\forall p \in P^{'}$, remove all keys which has subkeys matching it. So each position p associates with a list of minimal keys, we call it **characteristic list**.

Positions having the identical characteristic lists are equivalent, and they are actually non-terminal alphabet of conjectural grammar (CG), we also call it a **Class** of CG.

To generate the rule for a class C, if its original key (not subkeys) k has only one symbol, then the rule is $C \xrightarrow{k}$; otherwise split k to $k = as_1$ or $k = as_1s_2$, either way s_i is minimal keys for their corresponding positions. Find out the corresponding classes of s_i and generate the rule as $C \xrightarrow{a} C_1$ or $C \xrightarrow{a} C_1C_2$, depending on how k is splitted.

The conjectural grammar for the context-free grammar is generated

8.5 Parallel Algorithm

Jose L. Balcazar studied the parallelism of learning DFA [2]. He proved that there's no Concurrent-Read, Concurrent-Write Parallel Random Access Machine CRCW PRAM algorithm that can use polynomially many processors to learn DFA in $O(n/log\ n)$ time, and even in the strong use of parallelism the learning DFA must take close to linear time. However using approximation method in Angluin's paper this parallel algorithm parallelly queries a polynomial number of examples in constant time, so they were able to obtain a $n \cdot log(n+m)/log(n)$ time solution in Parallel Architecture Core (PAC) mode.

9 Conclusion and Future Work

The idea of learning from representative samples and using approximation is suggestive to solve hard problems in the real world. Take Machine Learning for example, this helps to avoid unnecessary over-fitting problems.

Learning languages is still hard problem. Nowadays strong assumptions are made to solve relative simple questions. But the power of machine keeps increasing, the rising of quantum computer will help solve more and more hard problems in this area.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples.

[2] Jose L. Balcazar. An optimal paralle algorithm for learning dfa. *Journal of Univeral Computer Science*, 2(3):97–112, 1996.

[3] Andrew Burago. Learning structually reversible context-free grammars from queries and counterexamples in polynomial time.

[4] Vijeth D Chintan Rao. A linear algorithm for testing equivalence of finite automata.

[5] E. M. Gold. *Complexity of automation identification from given data.* Addison-Wesley, 1978.

[6] qingjun wu. Implementation of angluin algorithm in javascript.

[7] wikipedia. Hopcroft–karp algorithm.

[8] T. Yokomori. Learning non-deterministic finite automata from queries and counterexamples.

[9] T. Yokomori. Polynomial-time identification of very simple grammars from positive data.