Learning Finite State Machines*

Colin de la Higuera

Université de Lyon, F-42023 Saint-Étienne,
CNRS UMR5516, Laboratoire Hubert Curien,
Université de Saint-Etienne - Jean Monnet
cdlh@univ-st-etienne.fr

Abstract. The terms *grammatical inference* and *grammar induction* both seem to indicate that techniques aiming at building grammatical formalisms when given some information about a language are not concerned with automata or other finite state machines. This is far from true, and many of the more important results in grammatical inference rely heavily on automata formalisms, and particularly on determinism. We survey here some of the main ideas and results in the field.

1 Introduction

The terms *grammatical inference* and *grammar induction* refer to the techniques allowing to rebuild a grammatical formalism for a language of which only partial information is known. These techniques are both inspired and have applications in fields like computational linguistics, pattern recognition, inductive inference, computational biology and machine learning [1, 2].

2 Some of the key ideas

We describe first some of the ideas that seem important to us.

2.1 You can't 'have learned'

The question posed by grammatical inference is that of building a grammar or automaton for an unknown language, given some data about this language. But it is essential to explain that, in a certain sense, this question can never be settled. In other words, one cannot hope to be able to state "I have learned" just because, given the data, we have built the best automaton for some combinatorial criterion (typically the one with the least number of states). Indeed, this would be similar to having used a random number generator, looking at the result, and claiming that "the returned number is random". In both cases, the issue is about the building process itself [3].

* This work was partially supported by the IST Programme of the European Community, under the Pascal 2 Network of Excellence, Ist–2006-216886.

2.2 Identification in the limit

One essential concept in grammatical inference, in order to study the process of learning languages, is due to Gold: *identification in the limit*.

Identification in the limit [4, 5] describes a situation where learning is a never ending process. The learner is given information, builds a hypothesis, receives more information, updates his hypothesis, and so on, forever. This setting may seem unnatural, completely abstract, and far from concrete learning situation. We argue differently and believe Identification in the limit provides several useful insights.

First, identification in the limit requires the notion of 'target'. A target language, from which the data is extracted, pre-exists to the learning process. Therefore the task is really about *finding* this hidden target.

Second, even if more typical learning situations are not incremental, it is still worth studying a learning session as part of a process. It may be that at some particular moment the learning algorithm returned a good hypothesis, but unless we have some completeness guarantee about the data we have got, there is no way we can be sure. In other words, one can study the fact that we are learning a language, but not that we have learned one.

Third, even knowing that our algorithm does identify in the limit will not give us any guarantee in a specific situation; what we do know is that if the algorithm does not identify in the limit there is necessarily a *hidden bias*: there is at least one possible language which, for some unknown or undeclared reason, is not learnable.

The advantage of using an algorithm that has the propriety of identifying in the limit is that you can argue, if the algorithm fails: "*Don't blame me, blame the data*" [3].

Definition 1. *Let $\mathcal{L}$ be a class of languages. A* presentation *is a function $\phi : \mathbb{N} \to \boldsymbol{X}$ where $\boldsymbol{X}$ is some set. The set of all admitted presentations for $\mathcal{L}$ is denoted by $\boldsymbol{Pres}(\mathcal{L})$ which is a subset of $\left[\mathbb{N} \to \boldsymbol{X}\right]$.*

In some way these presentations denote *languages from $\mathcal{L}$, i.e. there exists a function* Yields *: $\boldsymbol{Pres}(\mathcal{L}) \to \mathcal{L}$. If $L = \textsc{Yields}(\phi)$ then we will say that ϕ is a presentation of L. We also denote by $\boldsymbol{Pres}(L)$ the set $\{\phi \in \boldsymbol{Pres}(\mathcal{L}) : \textsc{Yields}(\phi) = L\}$.*

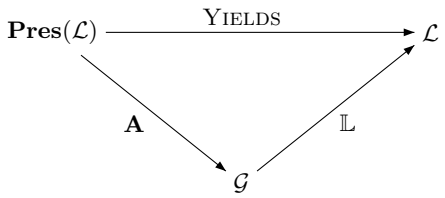


Fig. 1. The learning setting.

We summarise these notions in Figure 1. The general goal of learning (in the framework of identification in the limit) is to find a learning algorithm $\mathbf{A}$ such that $\forall \phi \in \mathbf{Pres}(\mathcal{L}), \exists n \in \mathbb{N} : \ \forall m \geq \ n, \mathbb{L}(\mathbf{A}(\phi_m)) = \textsc{Yields}(\phi)$. When this is true, we will say that class $\mathcal{L}$ is identifiable in the limit by presentations in $\mathbf{Pres}(\mathcal{L})$.

Gold [5] proved the first and essential results in this setting:

Theorem 1. *Any recursively enumerable class of recursive languages is identifiable in the limit from an informant.*
No super-finite class of languages is identifiable in the limit from text.

A *super-finite* language class is a class that contains all finite languages and at least one infinite language. Of course, the theorem holds for the usual classes of languages from the Chomsky hierarchy, and typically for regular languages.

2.3 Complexity matters

The general setting described in the previous section is close to what is typically called inductive inference. But if we want to apply the theoretical algorithms in practice, it is necessary to come up with algorithms whose resources are bounded in some way: one can (try to) bound the number of examples needed before convergence, in a best case or in a worse case, the number of mind changes the learning algorithm may make, or the number of times the learning algorithm fails to classify the next example it sees [6–8]. In fact, no unique notion of complexity covers all the possible learning scenarii.

3 Some basic tools

We describe some essential finite state tools that are of great use when devising or studying automata learning algorithms.

3.1 Three types of states

We shall be dealing here with *learning samples* composed of labelled strings. Let Σ be an alphabet. An *informed learning sample* is made of two sets S_+ and S_- such that $S_+ \cap S_- = \emptyset$. The The sample will be denoted as $S = \langle S_+, S_- \rangle$. If $S = \langle S_+, S_- \rangle$ is sampled from a language L, then $S_+ \subset L$ and $S_- \cap L = \emptyset$.A text sample is made of just one set S_+. Let us consider sample:

$$S_+ = \{\mathtt{a}, \mathtt{aa}\}$$
$$S_- = \{\mathtt{ab}, \mathtt{aaba}\}.$$

Now if we look at Dfa from Figure 2, we notice that state q_ε is reached by no string in the sample. Therefore, this state could just as well be accepting or rejecting, at least if we only have this data to decide upon. In order to not take premature decisions, we therefore manage three types of states: the final accepting states (double line), the final rejecting states (thick grey) and the other states, yet to be labelled.

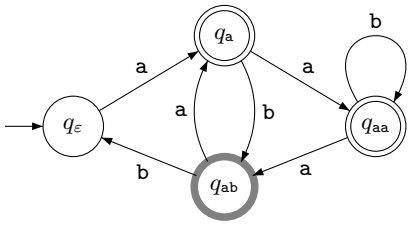


Fig. 2. A DFA.

3.2 The prefix tree acceptor (PTA)

A *prefix tree acceptor* (PTA) is a tree-like DFA built from the learning sample by taking all the prefixes in the sample as states and constructing the smallest DFA which is a tree ($\forall q \in Q|,\ \{q' : \delta(q', a) = q\}| \leq 1\}$). The PTA is useful as a good starting point for state merging algorithms and helps to see the learning problem like a search question [9].

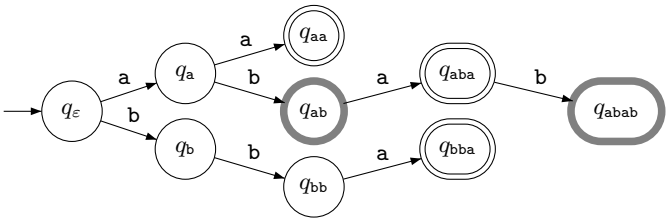


Fig. 3. PTA($\{(\mathtt{aa}, 1), (\mathtt{aba}, 1), (\mathtt{bba}, 1), (\mathtt{ab}, 0), (\mathtt{abab}, 0)\}$).

Note that we can also build a PTA from a set of positive strings only. This corresponds to building PTA($\langle S_+, \emptyset \rangle$). In that case, and for the same sample we would get the PTA represented in Figure 4.

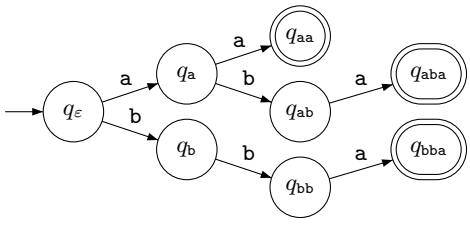


Fig. 4. PTA($\{(\mathtt{aa}, 1), (\mathtt{aba}, 1), (\mathtt{bba}, 1)\}$).

3.3 Red, Blue and White states

In order not to get lost in the process (and to avoid undoing merges that have been made some time ago) it will be interesting to divide the states into three categories [10]:

- The RED states which correspond to states that have been analysed and which will not be revisited; they will be the states of the final automaton.
- The BLUE states which are the *candidate* states: they have not been analysed yet and it should be from this set that a state is drawn in order to consider merging it with a RED state.
- The WHITE states, which are all the others. They will in turn become BLUE and then RED.

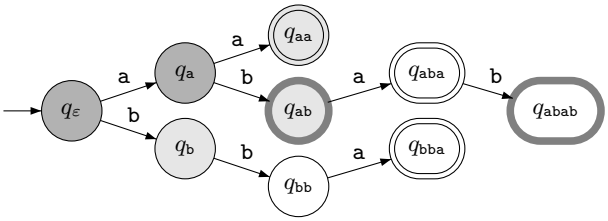


Fig. 5. Colouring of states: RED = $\{q_\varepsilon, q_\mathtt{a}\}$, BLUE = $\{q_\mathtt{b}, q_\mathtt{aa}, q_\mathtt{ab}\}$, all the other states are WHITE.

We conventionally draw the RED states in dark grey and the BLUE ones in light grey as in Figure 5, where RED = $\{q_\varepsilon, q_\mathtt{a}\}$ and BLUE = $\{q_\mathtt{b}, q_\mathtt{aa}, q_\mathtt{ab}\}$.

3.4 Merge and fold

Merging states (starting perhaps from the PTA) is the key to most algorithms that deal with inferring automata. The idea is that by merging states a generalisation of the language is taking place (thus some form of induction).

In order to avoid problems with non-determinism, the merge of two states is immediately followed by a folding operation: the merge always occurs between a RED state and a BLUE state. The BLUE states have the following properties:

– if q is a BLUE state, it has exactly one predecessor, *i.e.* whenever $\delta(q_1, a_1) = \delta(q_2, a_2) = q \in$ BLUE, then necessarily $q_1 = q_2$ and $a_1 = a_2$;
– $q \in$ BLUE is the root of a tree, *i.e.* if $\delta(q, u)$=$\delta(q, v)$ then necessarily $u = v$.

Algorithm MERGE takes as arguments a RED state q and a BLUE state q'. It first finds the unique pair (q_f, a) such that $q' = \delta_{\mathcal{A}}(q_f, a)$. This pair exists and is unique because q' is a BLUE state and therefore the root of a tree. MERGE then redirects $\delta(q_f, a)$ to q. After that, the tree rooted in q' (which is therefore disconnected from the rest of the DFA) is folded (FOLD) into the rest of the DFA. The possible intermediate situations of non-determinism are dealt with during the recursive calls to FOLD. This two-step process is shown in Figures 6 to 9. We want to merge state $q_{\mathtt{aa}}$ with q_ε.

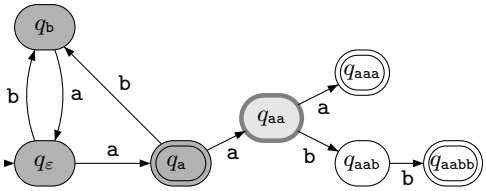


Fig. 6. Before merging $q_{\mathtt{aa}}$ with q_ε and redirecting transition $\delta_{\mathcal{A}}(q_{\mathtt{a}}, \mathtt{a})$ to q_ε.

Once the redirection has taken place (Figure 7), state $q_{\mathtt{aa}}$ is effectively merged (Figure 8) with q_ε. In dashed lines the recursive merges that are still to be done.

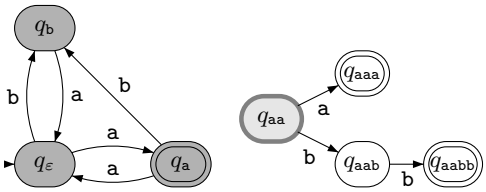


Fig. 7. Before folding $q_{\mathtt{aa}}$ into q_ε.

Then $q_{\mathtt{aaa}}$ is folded into $q_{\mathtt{a}}$, and finally $q_{\mathtt{aab}}$ into $q_{\mathtt{b}}$. The result is represented in Figure 9.

Traditional state merging techniques rely on a "merge and determinise" procedure [11]. The presentation given here avoids the cumbersome nondeterministic phase [12].

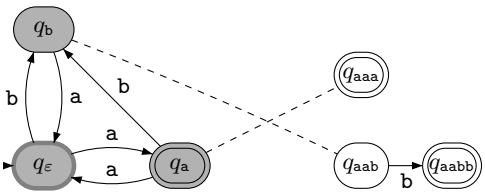
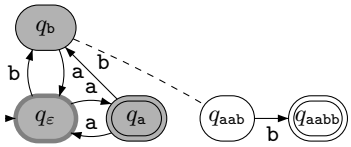
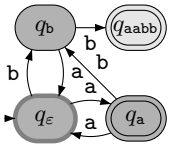


Fig. 8. Folding in $q_{\mathtt{aa}}$.



(a) Folding in $q_{\mathtt{aaa}}$



(b) Folding in $q_{\mathtt{aab}}$.

Fig. 9. The end of the merge and fold process

4 Some techniques and results

4.1 Dfa and Nfa

Deterministic finite automata (Dfa) have been learnt by a variety of techniques and in a number of settings:

- In the case of learning from text (only positive examples are available), theory tells us that identification in the limit cannot be achieved [4, 13]. But for certain subclasses, like the k-testable languages [14] or the k-reversible languages [15], algorithms that do identify in the limit have been built.
- When learning from an informant (this time both positive and negative examples are available), the picture is more positive. But if we are concerned with (too) optimistic convergence criteria, like those of Pac learning or of identification with only a polynomial number of implicit prediction errors, we should not hope for success [6, 16, 17]. If what we hope is to be able to only need polynomial characteristic samples [7], then algorithm Rpni [11] fulfils the conditions.

– Active learning consists in learning from an Oracle to whom queries are made. In this setting Angluin proved that equivalence queries alone, or membership queries alone were insufficient [18, 19]. On the other hand, algorithm L* makes use of a minimally adequate teacher [20] to learn Dfa in polynomial time.

4.2 Probabilistic automata

Another line of research concerns learning probabilistic finite automata. State merging techniques have also been devised in this case. Algorithm Alergia uses the same principles as Rpni and merges two states when the distributions look alike [21]. Algorithm Mdi [22] uses distances between distributions to take the decisions about merging and algorithm Dsai [23] merges if there is a string of significant weight that appears in the two distributions with very different relative frequencies.

Finally, Algorithm Dees solves systems of equations and uses multiplicity automata in order to learn probabilistic finite automata that may not be deterministic [24].

4.3 Transducers

Algorithm Ostia was developed to learn subsequencial transducers from examples consisting of pairs: each pair is composed of one string from the original language and its translation [25]. An extension of Ostia to allow the use of extra knowledge (typically about the domain and range of the function) has been proposed [26], and an active learning version, where the goal is to learn from translation queries exists [27].

5 Some open problems and challenges

The fact that there are a number of existing algorithms that allow to infer automata and transducers should not deter the potential researcher. Indeed, there are so many possible types of data from which to learn, or specific conditions posed on the automata we want to learn, or even on the criteria used to measure the quality of the learning process makes it of interest to pursue research. As an alternative starting point, one may find some open problems relating to learning finite state machines in [28], and an extensive presentation can be found in [12].

References

1. Sakakibara, Y.: Recent advances of grammatical inference. Theoretical Computer Science **185** (1997) 15–45
2. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition **38** (2005) 1332–1348

3. de la Higuera, C.: Data complexity issues in grammatical inference. In Basu, M., Ho, T.K., eds.: Data Complexity in Pattern Recognition. Springer-Verlag (2006) 153–172
4. Gold, E.M.: Language identification in the limit. Information and Control **10**(5) (1967) 447–474
5. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37** (1978) 302–320
6. Pitt, L.: Inductive inference, DFA's, and computational complexity. In: Analogical and Inductive Inference. Number 397 in LNAI. Springer-Verlag (1989) 18–44
7. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning Journal **27** (1997) 125–138
8. de la Higuera, C., Janodet, J.C., Tantini, F.: Learning languages from bounded resources: the case of the DFA and the balls of strings. In Clark, A., Coste, F., Miclet, L., eds.: Grammatical Inference: Algorithms and Applications, Proceedings of ICGI '08. Volume 5278 of LNCS., Springer-Verlag (2008) 43–56
9. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? [29] 25–37
10. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In Honavar, V., Slutski, G., eds.: Grammatical Inference, Proceedings of ICGI '98. Number 1433 in LNAI, Springer-Verlag (1998) 1–12
11. Oncina, J., García, P.: Identifying regular languages in polynomial time. In Bunke, H., ed.: Advances in Structural and Syntactic Pattern Recognition. Volume 5 of Series in Machine Perception and Artificial Intelligence. World Scientific (1992) 99–108
12. de la Higuera, C.: Grammatical inference: learning automata and grammars. Cambridge University Press (2009) (to appear).
13. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control **45** (1980) 117–135
14. García, P., Vidal, E.: Inference of K-testable languages in the strict sense and applications to syntactic pattern recognition. Pattern Analysis and Machine Intelligence **12**(9) (1990) 920–925
15. Angluin, D.: Inference of reversible languages. Journal of the Association for Computing Machinery **29**(3) (1982) 741–765
16. Angluin, D., Kharitonov, M.: When won't membership queries help? In: Proceedings of 24th ACM Symposium on Theory of Computing, New York, ACM Press (1991) 444–454
17. Kearns, M.J., Vazirani, U.: An Introduction to Computational Learning Theory. MIT press (1994)
18. Angluin, D.: Queries and concept learning. Machine Learning Journal **2** (1987) 319–342
19. Angluin, D.: Negative results for equivalence queries. Machine Learning Journal **5** (1990) 121–150
20. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Control **39** (1987) 337–350
21. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. [29] 139–150
22. Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: Proceedings of the 17th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA (2000) 975–982

23. Ron, D., Singer, Y., Tishby, N.: Learning probabilistic automata with variable memory length. In: Proceedings of COLT 1994, New Brunswick, New Jersey, ACM Press (1994) 35–46
24. Denis, F., Esposito, Y.: Learning classes of probabilistic automata. In Shawe-Taylor, J., Singer, Y., eds.: Proceedings of COLT 2004. Volume 3120 of LNCS., Springer-Verlag (2004) 124–139
25. Oncina, J., García, P., Vidal, E.: Learning subsequential transducers for pattern recognition interpretation tasks. Pattern Analysis and Machine Intelligence **15**(5) (1993) 448–458
26. Oncina, J., Varó, M.A.: Using domain information during the learning of a subsequential transducer. [30] 301–312
27. Vilar, J.M.: Query learning of subsequential transducers. [30] 72–83
28. de la Higuera, C.: Ten open problems in grammatical inference. In Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T., Tomita, E., eds.: Grammatical Inference: Algorithms and Applications, Proceedings of ICGI '06. Volume 4201 of LNAI., Springer-Verlag (2006) 32–44
29. Carrasco, R.C., Oncina, J., eds.: Grammatical Inference and Applications, Proceedings of ICGI '94. In Carrasco, R.C., Oncina, J., eds.: Grammatical Inference and Applications, Proceedings of ICGI '94. Number 862 in LNAI, Springer-Verlag (1994)
30. Miclet, L., de la Higuera, C., eds.: Proceedings of ICGI '96. In Miclet, L., de la Higuera, C., eds.: Proceedings of ICGI '96. Number 1147 in LNAI, Springer-Verlag (1996)