Learning Automata and Grammars

P. Černo

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. The problem of learning or inferring automata and grammars has been studied for decades and has connections to many disciplines, including bioinformatics, computational linguistics and pattern recognition. In this paper we present a short survey of basic models and techniques related to the grammatical inference and try to outline some new promising approaches which we expect to bring new light into this subject. For illustration, we introduce delimited string-rewriting systems as a sample model for grammatical inference and sketch the simplified version of the learning algorithm LARS for these systems.

1. Introduction

The central notion in the *formal language theory* is a (formal) *language*, which is a finite or infinite set of words. A *word* is a finite sequence consisting of zero or more letters, whereby the same letter may occur several times. The sequence of zero letters is called the *empty word*, written λ. In defining words (and languages) we usually restrict ourselves to some specific finite nonempty set of letters, called the *alphabet*. The set of all words (all nonempty words, respectively) over an alphabet Σ is denoted by Σ^* (Σ^+, respectively). If x and y are words over Σ, then so is their *concatenation* xy (or $x \cdot y$), obtained by juxtaposition, that is, writing x and y one after another. Concatenation is an associative operation and the empty word λ acts as an identity element: $w\lambda = \lambda w = w$ holds for all words w.

In formal language theory in general, there are two major types of mechanisms for defining languages: *acceptors* and *generators*. Acceptors are usually defined in terms of *automata*, which work as follows: they are given an input word and after some processing they either accept or reject this input word. For instance, the so-called *finite automata* consist of a finite set of internal states and a set of rules that govern the change of the current state when reading a given input symbol. The finite automaton reads a given input word from left to right starting in a specific *starting state*. After reading the input word it accepts only if it ends in the so-called *accepting state*, otherwise it rejects. Finite automata recognize the family of *regular languages*, which plays a central role in the whole formal language theory. Regular languages and finite automata have had a wide range of applications. Their most celebrated application has been lexical analysis in programming language compilation and user-interface translations. Other notable applications include circuit design, text editing, and pattern matching.

Generators, on the other hand, usually generate the language using some finite set of rules. Typically they are defined in terms of grammars. One of the most famous is the classical *Chomsky hierarchy* of grammars (and corresponding languages), which consists of *phrase-structure*, *context-sensitive*, *context-free*, and *regular* grammars (they are also called type 0, type 1, type 2, and type 3 grammars, respectively). Type 0 grammars and languages are equivalent to *computability*: what is in principle computable. Thus, their importance is beyond any question. The same or almost the same can be said about regular grammars and languages. They correspond to strictly finitary computing devices (finite automata). The remaining two classes lie in-between. The class of context-sensitive languages has turned out to be of smaller importance than the other classes. The particular type of context-sensitivity combined with linear work-space is perhaps not the essential type, it has been replaced by various complexity hierarchies. The Chomsky hierarchy still constitutes a testing ground often used: new classes are compared with those in the Chomsky hierarchy. However, it is not any more the only testing ground in the language theory.

In this short survey we are interested mainly in the problem of learning automata and grammars under a suitable learning regime. This topic is associated with many different fields, and also under a number of names, such as *grammar learning*, *automata inference*, *grammar identification*, but principally *grammar induction* and *grammatical inference*. By grammar induction we mean finding a grammar (or automaton) that can explain the data, whereas grammatical inference relies on the fact that there is a (true) target grammar (or automaton), and that the quality of the learning process has to be measured relatively to this target.

2. Importance of learnability

Alexander Clark in Clark [2010] emphasized the importance of learnability of the representation classes of formal languages. He proposed that one way to build learnable representations is by making them objective or empiricist: the structure of the representation should be based on the structure of the language. He illustrated this approach with three classes corresponding to the lowest three levels of the Chomsky hierarchy. All these classes were efficiently learnable under suitable learning paradigms.

In defining these representation classes the author followed a simple slogan: "Put learnability first!" It means that we should design representations from the ground to be learnable. Rather than defining a representation, and then defining a function from the representation to the language, we should start by defining the map from the language to the representation. The basic elements of such formalism, whether they are states in an automaton, or non-terminals in a phrase-structure grammar, must have a clear definition in terms of sets of strings.

In the conclusive remarks the author suggested that the representations, which are both efficiently learnable and capable of representing mildly context-sensitive languages seem to be good candidates for models of human linguistic competence.

The fact that the human being should be able to discover the syntactic representations of language was the key motivation for introduction of many formalisms to define formal languages. However, as Alexander Clark pointed out, if we first define a representation and then a function from this representation to the language, we are likely to encounter difficult obstacle to going in the reverse direction. Imagine a context-free language L, and a grammar G such that $L(G) = L$. If N is a non-terminal in G, what constraints are there on the language $L(N)$? We can say literally nothing about this set, except that it is a context-free language.

In our opinion, the problems with efficient learnability arise whenever the representation makes use of any of the auxiliary elements such as states, non-terminals, auxiliary symbols, stack, working tapes etc. The complexity lies not only in the fact that the hidden structure between these elements in our target model can be very intricate and complex, but sometimes even if we know exactly the concrete structure of our model, there may still be some very hard (or even algorithmically undecidable) questions concerning the model. Although there is a number of methods and algorithms to learn regular languages, moving up the Chomsky hierarchy is proving to be a challenging task. Indeed, there are several theoretical barriers which make the class of context-free languages hard to learn. Alexander Clark has tried to avoid these obstacles by defining the basic elements of his formalisms in terms of sets of strings. Another promising way to tackle these barriers is to change the way we represent the languages. One approach is to consider models which do not use auxiliary elements at all. Typical representatives of this category are contextual grammars in Marcus [1969], pure grammars in Maurer et al. [1980], and of course locally testable languages in Salomaa [1987]. More recent associative language description model in Cherubini et al. [2002] was the main inspiration for introducing our own model of so-called clearing restarting automata in Černo and Mráz [2009, 2010]. Our model is also quite similar to the so-called delimited string-rewriting systems introduced in Eyraud et al. [2007]. Kutrib et al. [2009, 2010] introduced the so-called stateless restarting automata, which are in fact restarting automata (Jančar et al. [1995]; Otto [2006]) restricted to use only one single state. However, they can still use auxiliary symbols. For brevity, we omit definitions of these models and refer the interested reader to the references.

In a typical grammatical inference scenario we are concerned with learning language representations based on some source of information, which can be text, examples and counter-examples, or anything that should provide us insight about the elements of the language being sought. In the following Section 3 we provide a discussion concerning the general setting for the problem of identifying languages. In Section 4 we introduce delimited string-rewriting systems as a sample model for grammatical inference. In Section 5 we sketch the simplified version of the algorithm LARS for delimited string-rewriting systems. We believe that the study of delimited string-rewriting systems will help us better understand our own model of clearing restarting automata, and perhaps we will find a similar algorithm to LARS adapted to our own model.

The main source for this whole article is de la Higuera [2010], which provides a nice and comprehensive survey of the techniques and results concerning the grammatical inference, and Rozenberg and Salomaa [1997], which rigorously covers the whole formal language theory.

3. General setting

There are some problems whose tractability is of great importance in grammatical inference. Let $\mathcal{L}$ be a language class, $\mathcal{G}$ be a class of representation of objects for $\mathcal{L}$ and $L : \mathcal{G} \to \mathcal{L}$ be the *naming function*, i.e. $L(G)$ is the language denoted, accepted, recognized or represented by $G \in \mathcal{G}$. In the following we fix the alphabet Σ.

The first problem concerns the fact whether the following *membership problem* is decidable: given $w \in \Sigma^*$ and $G \in \mathcal{G}$, is $w \in L(G)$? It is well known that, for instance, for context-free grammars the membership problem is decidable in polynomial time.

The second problem is the *equivalence problem*: given $G, G' \in \mathcal{G}$, do we have $L(G) = L(G')$? For context-free grammars, the equivalence problem is undecidable. It is decidable for finite automata, but the complexity depends on whether the automata are deterministic or not.

The aforementioned problems are well mathematically defined and formalized. However, considering machine learning as a field where good ideas can be tested and lead to algorithms, we encounter more difficult problems with no clear or established notion. How do we know that the method or algorithm in use is able to infer a reasonable model for our target language? The trick we will use is to consider that the problem we are really interested in is not about discovering the model that would explain the data, but about identifying the *hidden target model*. An alternative formalism for convergence may be that there is no target: the idea is just to induce a grammar from the data in such a way as to minimize some statistical criterion. But again, whether explicitly or implicitly, there is somewhere, hidden, an ideal solution that we can call a target. This discussion leads us to an important notion of the so-called *identification in the limit*. We will not delve much into the technical details of this notion, but only sketch informally the basic idea.

A *presentation* ϕ is an enumeration of elements, which represents a source of information about some specific language $L \in \mathcal{L}$. An example of presentation can be, for instance, the enumeration of all positive and negative samples of L (in some order). A *learning algorithm* $\mathbf{A}$ is a program that takes the first n elements of a presentation (denoted as ϕ_n) and returns some object $G \in \mathcal{G}$. We say that $\mathcal{G}$ is identifiable in the limit if there exists a learning algorithm $\mathbf{A}$ such that for any target object $G \in \mathcal{G}$ and any presentation ϕ of $L(G)$ there exists a rank n such that for all $m \geq n$, $\mathbf{A}(\phi_m)$ does not change and $L(\mathbf{A}(\phi_m)) = L(G)$. Notice that the above definition does not force us to learn the target object G, but only to learn an object equivalent to the target. However, there are some complexity issues with the identification in limit, since it neither tells us how we know when we have found what we are looking for nor how long it is going to take.

4. Rewritting systems

Languages can be defined by using string-rewriting systems, which are usually specified by some rewriting mechanism and a base of simple (accepted) words. In order to study a class containing all the regular languages, we use the so-called delimited string-rewriting systems.

Let us introduce two new symbols ¢ and \$, called the *sentinels*, that do not belong to the alphabet Σ. We will be concerned with languages that are subsets of $¢ \cdot \Sigma^* \cdot \$$. As for the rewrite rules, they will be made of pairs of *terms* partially marked; a term is a string from $T(\Sigma) = \{\lambda, ¢\} \cdot \Sigma^* \cdot \{\lambda, \$\}$.

Terms in $T(\Sigma)$ can be of the following *types*: type 1: $w \in \Sigma^*$, type 2: $w \in ¢ \cdot \Sigma^*$, type 3: $w \in \Sigma^* \cdot \$$, and type 4: $w \in ¢ \cdot \Sigma^* \cdot \$$. For $w \in T(\Sigma)$ the *root* of w is w without the sentinels ¢ and \$, e.g. $root(¢aab) = aab$. We define a specific order relation over $T(\Sigma)$: $u < v \Leftrightarrow root(u) <_{lex-length} root(v) \vee (root(u) = root(v) \wedge type(u) < type(v))$, where $w_1 <_{lex-length} w_2 \Leftrightarrow |w_1| < |w_2| \vee (|w_1| = |w_2| \wedge w_1 <_{lex} w_2)$. For instance, $ab < ¢ab < ab\$ < ¢ab\$ < ba$.

A *rewrite rule* ρ is an ordered pair of terms $\rho = (l, r)$, generally written as $\rho = l \vdash r$. The term l is called the *left-hand side* of ρ and r is *right-hand side* of ρ. We say that $\rho = l \vdash r$ is a *delimited rewrite rule* if l and r are of the same type. By a *delimited string-rewriting system* (DSRS), we mean any finite set $\mathcal{R}$ of delimited rewrite rules. The order relation extends to rules: $(l_1, r_1) < (l_2, r_2)$ if $l_1 < l_2$ or $(l_1 = l_2) \wedge (r_1 < r_2)$.

A system is *deterministic* if no two rules share a common left-hand side. Given a system $\mathcal{R}$ and string w, there may be several rules applicable upon w. Nevertheless, only one rule is eligible. This is the rule having the smallest left-hand side. The same rule might be eligible in different places, but we systematically privilege the leftmost position.

Given a DSRS $\mathcal{R}$ and two strings $w_1, w_2 \in T(\Sigma)$, we say that w_1 *rewrites in one step into* w_2, written $w_1 \vdash_{\mathcal{R}} w_2$ or simply $w_1 \vdash w_2$, if there exists an eligible rule $(l \vdash r) \in \mathcal{R}$ for w_1, and there are two strings $u, v \in T(\Sigma)$ such that $w_1 = ulv$ and $w_2 = urv$, and furthermore u is shortest for this rule. A string w is *reducible* if there exists w' such that $w \vdash w'$, and *irreducible* otherwise. Let $\vdash^*_{\mathcal{R}}$ (or simply $\vdash^*$) denote the reflexive and transitive closure of $\vdash_{\mathcal{R}}$. We say that w_1 *reduces to* w_2 or that w_2 is *derivable from* w_1 if $w_1 \vdash^*_{\mathcal{R}} w_2$.

Given a DSRS $\mathcal{R}$ and an irreducible string $e \in \Sigma^*$ we define the language $L(\mathcal{R}, e)$ as the set of strings that reduce to e using the rules of $\mathcal{R}$: $L(\mathcal{R}, e) = \{w \in \Sigma^* \mid ¢w\$ \vdash^*_{\mathcal{R}} ¢e\$\}$. Deciding whether a string w belongs to a language $L(\mathcal{R}, e)$ consists of trying to obtain e from w by a rewriting derivation. We will denote by $\mathsf{Apply}_{\mathcal{R}}(w)$ the string obtained by applying the different rules in $\mathcal{R}$ until no more rules can be applied. We extend the notation to a set of strings: $\mathsf{Apply}_{\mathcal{R}}(S) = \{\mathsf{Apply}_{\mathcal{R}}(w) \mid w \in S\}$.

Example 4.1 *Let $\Sigma = \{a, b\}$.*

1. $L(\{ab \vdash \lambda\}, \lambda)$ *is the* Dyck language*. The single rule erases substring ab, as is illustrated in the following example of derivation:*

$$¢aabb\underline{ab}\$ \vdash ¢a\underline{ab}b\$ \vdash ¢\underline{ab}\$ \vdash ¢\lambda\$.$$

2. $L(\{ab \vdash \lambda; ba \vdash \lambda\}, \lambda)$ *is the language* $\{w \in \Sigma^* \mid |w|_a = |w|_b\}$*, because every rewriting step erases one a and one b.*

3. $L(\{aabb \vdash ab; ¢ab\$ \vdash ¢\$\}, \lambda)$ *is the language* $\{a^n b^n \mid n \geq 0\}$*. For instance,*

$$¢aa\underline{aabb}bb\$ \vdash ¢a\underline{aabb}b\$ \vdash ¢\underline{aabb}\$ \vdash ¢\underline{ab}\$ \vdash ¢\lambda\$.$$

4. $L(\{¢ab \vdash ¢\}, \lambda)$ *is the regular language* $(ab)^*$*. It can be shown that given any regular language L there is a system $\mathcal{R}$ such that $L(\mathcal{R}, \lambda) = L$.*

5. Algorithm LARS

Learning Algorithm for Rewriting Systems (LARS) introduced in Eyraud et al. [2007] generates the possible rules among those that can be applied over the positive samples S_+, tries using them and keeps them if they do not create inconsistency (using the negative samples S_- for that). Algorithm LARS calls the function NewRule, which generates the next possible rule to be checked.

For this, one should choose *useful* rules, i.e. those that can be applied on at least one string from S_+. One might also consider useful a rule that allows us to diminish the size of the set S_+: a rule which, when added, has the property that two different strings rewrite into an identical string. The goal of usefulness is to avoid an exponential explosion in the number of rules to be checked. The function Consistent checks that by adding the new rule to the system, one does not rewrite a positive example and a negative example into a same string.

Algorithm 5.1 *LARS*

Input: S_+, S_-.

Output: $\mathcal{R}$.

Description:

```
(1)     $\mathcal{R} \leftarrow \emptyset$; $\rho \leftarrow (\lambda \vdash \lambda)$;
(2)     while $|S_+| > 1$ do
(2.1)     $\rho \leftarrow \mathsf{NewRule}(S_+, \rho)$;
(2.2)     if $\mathsf{Consistent}(S_+, S_-, \mathcal{R} \cup \{\rho\})$ then
(2.2.1)     $\mathcal{R} \leftarrow \mathcal{R} \cup \{\rho\}$;
(2.2.2)     $S_+ \leftarrow \mathsf{Apply}_{\mathcal{R}}(S_+)$;
(2.2.3)     $S_- \leftarrow \mathsf{Apply}_{\mathcal{R}}(S_-)$;
(2.3)     end
(3)     end
(4)     return $\mathcal{R}$
```

The goal is to be able to learn any DSRS with LARS. The simplified version proposed here can be used as basis for that, and does identify in the limit any DSRS. But, a formal study of the qualities of the algorithm is beyond scope of this article. We refer the interested reader to the article Eyraud et al. [2007].

6. Conclusion

Our own model of clearing restarting automata is quite similar to the delimited string rewriting systems with the exception that we only allow rules of the form $\rho = (xzy \vdash xy)$. In the extended model, called the Δ-clearing restarting automata, we allow also rules $\rho = (xzy \vdash x\Delta y)$, where Δ is a special auxiliary symbol. However, algorithm LARS cannot be directly applied to clearing restarting automata, since we do not specify any order relation on terms or rules. This makes our model implicitly nondeterministic. It would be therefore interesting to investigate the deterministic version of clearing restarting automata obtained by using the same order relation as used in delimited string rewriting systems.

Acknowledgments. The author thanks RNDr. František Mráz, CSc. for his advices and guidance. The present work was partially supported by the Grant Agency of Charles University under Grant-No. 272111/MFF/A-INF and by the Czech Science Foundation under Grant-No. P103/10/0783 and Grant-No. P202/10/1333.

References

Černo, P. and Mráz, F., Clearing restarting automata, in *Workshop on Non-Classical Models for Automata and Applications (NCMA)*, edited by H. Bordinh, R. Freund, M. Holzer, M. Kutrib, and F. Otto, vol. 256 of *books@ocg.at*, pp. 77–90, Österreichisches Computer Gesellschaft, 2009.

Černo, P. and Mráz, F., Clearing restarting automata, *Fundamenta Informaticae*, *104*, 17–54, 2010.

Cherubini, A., Reghizzi, S. C., and San Pietro, P., Associative language descriptions, *Theor. Comput. Sci.*, *270*, 463–491, 2002.

Clark, A., Three learnable models for the description of language, in *Language and Automata Theory and Applications*, edited by A.-H. Dediu, H. Fernau, and C. Martn-Vide, vol. 6031 of *Lecture Notes in Computer Science*, pp. 16–31, Springer Berlin / Heidelberg, 2010.

de la Higuera, C., *Grammatical Inference: Learning Automata and Grammars*, Cambridge University Press, New York, NY, USA, 2010.

Eyraud, R., de la Higuera, C., and Janodet, J.-C., Lars: A learning algorithm for rewriting systems, *Machine Learning*, *66*, 7–31, 2007.

Jančar, P., Mráz, F., Plátek, M., and Vogel, J., Restarting automata, in *FCT'95*, edited by H. Reichel, vol. 965 of *LNCS*, pp. 283–292, Springer, Dresden, Germany, 1995.

Kutrib, M., Messerschmidt, H., and Otto, F., On stateless deterministic restarting automata, in *SOFSEM*, edited by M. Nielsen, A. Kučera, P. B. Miltersen, C. Palamidessi, P. Tůma, and F. D. Valencia, vol. 5404 of *LNCS*, pp. 353–364, Springer, 2009.

Kutrib, M., Messerschmidt, H., and Otto, F., On stateless deterministic restarting automata, *Acta Inf.*, *47*, 391–412, 2010.

Marcus, S., Contextual grammars, in *Proceedings of the 1969 conference on Computational linguistics*, COLING '69, pp. 1–18, Association for Computational Linguistics, Stroudsburg, PA, USA, 1969.

Maurer, H., Salomaa, A., and Wood, D., Pure grammars, *Information and Control*, *44*, 47 – 72, 1980.

Otto, F., Restarting automata, in *Recent Advances in Formal Languages and Applications*, edited by Z. Ésik, C. Martín-Vide, and V. Mitrana, vol. 25 of *Studies in Computational Intelligence*, pp. 269–303, Springer, Berlin, 2006.

Rozenberg, G. and Salomaa, A., eds., *Handbook of Formal Languages (3 volumes)*, Springer, 1997.

Salomaa, A., *Formal languages*, Academic Press Professional, Inc., San Diego, CA, USA, 1987.