Learning Languages from Bounded Resources: The Case of the DFA and the Balls of Strings[⋆]

Colin de la Higuera and Jean-Christophe Janodet and Frédéric Tantini

Universities of Lyon, 18 r. Pr. Lauras, F-42000 St-Etienne
{cdlh,janodet,frederic.tantini}@univ-st-etienne.fr

Abstract. Comparison of standard language learning paradigms (identification in the limit, query learning, Pac learning) has always been a complex question. Moreover, when to the question of converging to a target one adds computational constraints, the picture becomes even less clear: how much do queries or negative examples help? Can we find good algorithms that change their minds very little or that make very few errors? In order to approach these problems we concentrate here on two classes of languages, the topological balls of strings (for the edit distance) and the deterministic finite automata (Dfa), and (re-)visit the different learning paradigms to sustain our claims.

1 Introduction

The study of the properties of the learning algorithms, particularly those in grammatical inference, can be either empirical (based on experiments from datasets), or theoretical. In the latter, the goal is to study the capacity of the algorithm to retrieve, exactly or approximately, a target language. Often, the goal is also to measure the resources (time, amount of data) necessary to achieve this task. Different paradigms have been proposed to take into account notions of convergence from bounded resources, but none has really imposed itself, and few comparison exists between these definitions.

In this paper, we visit standard criteria for *polynomial* identification and compare them by considering two fundamentally different classes of languages: the regular languages represented by deterministic finite automata, and the balls of strings *w.r.t.* the edit distance [1].

When aiming to prove that a class of languages is learnable, there are typically three different settings. The first one, identification in the limit [2], mimes the cognitive process of a child that would acquire his native language by picking up the sentences that are broadcasted in his environment. More formally, information keeps on arriving about a target language and the Learner keeps making new hypotheses. We say that convergence takes place if there is a moment when the process is stationary and the hypothesis is correct.

⋆ This work was supported in part by the IST Programme of the European Community, under the Pascal Network of Excellence, IST-2006-216886. This publication only reflects the authors' views.

The second one, query learning [3], looks like a game of riddles where the Learner (pupil) can asks questions (queries) to an Oracle (teacher) about the target language. The game ends when the Learner guesses the target. Of course, the learning results strongly depends on the sort of queries that the Learner is allowed to ask. Both previous paradigms are probably at least as interesting for the negative results they induce as for the positive ones. Indeed, concerning query learning, if a Learner cannot identify an unknown concept by choosing and testing examples, how could he hope to succeed to learn from examples that are imposed by an application?

The last paradigm, Pac learning (for Probably Approximately Correct) [4] is intended to be a more pragmatic setting. It formalizes a situation where one tries to build automatically a predictive model from the data. In this setting, one assumes that there is a (unknown) distribution $\mathcal{D}$ over the strings of the target language, which is used to sample learning and testing examples. Two parameters are fixed: ϵ is related to the error of the model (*i.e.*, the probability of a string to be misclassified) and δ is related to the confidence one has in the sampling. Ideally, a good Pac-Learner returns, with high confidence ($> 1 - \delta$), hypotheses that have small error rates ($< \epsilon$).

The three settings are usually difficult to compare, in particular when complexity issues are discussed. Some exceptions are the work by Angluin comparing Pac-learning and using equivalence queries [5], the work by Pitt relating equivalence queries and implicit prediction errors [6], comparisons between learning with characteristic samples, simple Pac [7] and Mat in [8]. Other analysis of polynomial aspects of learning grammars, automata and languages can be found in [6, 9–11]. If the customary comparative approach is to introduce a learning paradigm and survey a variety of classes of languages for this paradigm, we choose here to fix the classes of languages and to proceed to a horizontal analysis of their learnability by visiting the paradigms systematically.

Concerning the Dfa, we complete a long list of known results. Concerning the balls of strings, our results are generally negative: identification in the limit from examples and counter-examples is impossible in most cases, even from membership and equivalence queries. Pac-learning is also impossible in polynomial time, unless $\mathcal{RP} = \mathcal{NP}$. Yet, the errors are usually due to the counter-examples. Hence, we show that it is sometimes (and surprisingly) easier to learn from positive examples only than from positive and negative examples.

Section 2 is devoted to preliminary definitions. In Sections 3, 4 and 5, we focus on the so-called good balls and on the Dfa, and we present the results concerning Pac-learning, query learning and polynomial identification in the limit, respectively. We conclude in Section 6.

2 Definitions

An *alphabet* Σ is a finite nonempty set of symbols called *letters*. In the sequel, we suppose that $|\Sigma| \geq 2$. A *string* $w = a_1 \cdots a_n$ is any finite sequence of letters. We write λ for the empty string and $|w|$ for the length of w. Let $\Sigma^\star$ denote the set of all strings over Σ. We say that u is a *subsequence* of v, denoted $u \preceq v$, $\mathit{if}_{\mathit{def}}$ $u = a_1 \cdots a_n$ and there exist $u_0, \cdots, u_n \in \Sigma^\star$ *s.t.* $v = u_0 a_1 u_1 \cdots a_n u_n$. We introduce the set $lcs(u,v)$ of all longest common subsequences of u and v. We also introduce the *hierarchical order*: $u \trianglelefteq v$ $\mathit{if}_{\mathit{def}}$ $|u| < |v|$ or ($|u| = |v|$ and $u \leq_{lex} v$). A *language* is any subset $L \subseteq \Sigma^\star$. Let $\mathbb{N}$ denote the set of non negative integers. For all $k \in \mathbb{N}$, let $\Sigma^{\leq k}$ (respectively $\Sigma^{>k}$) be the set of all strings of length at most k (respectively of length more than k). We define $A \oplus B = (A \setminus B) \cup (B \setminus A)$.

Grammatical inference aims at learning the languages of a fixed class $\boldsymbol{\mathcal{L}}$ represented by the grammars of a class $\boldsymbol{\mathcal{G}}$. $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{G}}$ are related by a naming function $\mathbb{L} : \boldsymbol{\mathcal{G}} \to \boldsymbol{\mathcal{L}}$ that is total ($\forall G \in \boldsymbol{\mathcal{G}}, \mathbb{L}(G) \in \boldsymbol{\mathcal{L}}$) and surjective ($\forall L \in \boldsymbol{\mathcal{L}}, \exists G \in \boldsymbol{\mathcal{G}}$ *s.t.* $\mathbb{L}(G) = L$). For any string $w \in \Sigma^\star$ and language $L \in \boldsymbol{\mathcal{L}}$, we shall write $L \models w$ $\mathit{if}_{\mathit{def}}$ $w \in L$. Concerning the grammars, they may be understood as any piece of information allowing some parser to recognize the strings. For any string $w \in \Sigma^\star$ and grammar $G \in \boldsymbol{\mathcal{G}}$, we shall write $G \vdash w$ if the parser recognizes w. Basically, the parser must be sound and complete *w.r.t.* the semantics: $G \vdash w \iff \mathbb{L}(G) \models w$. In the following, we will mainly consider learning paradigms subject to complexity constraints. In their definitions, $\|G\|$ will denote the size of the grammar G (*e.g.*, the number of states in the case of Dfa). Moreover, given a set X of strings, we will write $|X|$ for the cardinality of X and $\|X\|$ for the sum of the lengths of the strings in X.

The *edit distance* $d(w, w')$ is the minimum number of *primitive edit operations* needed to transform w into w' [1]. The operation is either (1) a *deletion*: $w = uav$ and $w' = uv$, or (2) an *insertion*: $w = uv$ and $w' = uav$, or (3) a *substitution*: $w = uav$ and $w' = ubv$, where $u, v \in \Sigma^\star$, $a, b \in \Sigma$ and $a \neq b$. *E.g.*, $d(abaa, aab) = 2$ since $a\underline{b}aa \to aa\underline{a} \to aab$ and the rewriting of $abaa$ into aab cannot be achieved with less than two steps. $d(w, w')$ can be computed in $\mathcal{O}(|w| \cdot |w'|)$ time by dynamic programming [12].

The edit distance is a metric, so we can introduce the *balls* over Σ. The *ball of centre* $o \in \Sigma^\star$ *and radius* $r \in \mathbb{N}$, denoted $B_r(o)$, is the set of all strings whose distance is at most r from o: $B_r(o) = \{w \in \Sigma^\star : d(o, w) \leq r\}$. *E.g.*, if $\Sigma = \{a, b\}$, then $B_1(ba) = \{a,b,aa,ba,bb,aba,baa,bab,bba\}$ and $B_r(\lambda) = \Sigma^{\leq r}$ for all $r \in \mathbb{N}$. We will write $\boldsymbol{\mathcal{BALL}}(\Sigma)$ for the family of all the balls.

To the purpose of grammatical inference, we are going to represent any ball $B_r(o)$ by the pair (o, r) that will play the role of a grammar. Indeed, its size is $|o| + \log r$ (which corresponds to the number of bits necessary to encode the grammar[1]). Moreover, the parser able to decide whether $w \in B_r(o)$ or not is simple: (1) it computes $d(o, w)$ and (2) it checks if this distance is $\leq r$, that can be achieved in time $\mathcal{O}(|o| \cdot |w| + \log r)$. Finally, as $|\Sigma| \geq 2$, we can show that (o, r) is a unique thus *canonical* grammar of $B_r(o)$ [14]. In consequence, we shall also denote by $\boldsymbol{\mathcal{BALL}}(\Sigma)$ the class of grammars associated to the balls.

A ball $B_r(o)$ is called *good* $\mathit{if}_{\mathit{def}}$ $r \leq |o|$. The advantage of using good balls is that there is a polynomial relation between the size of the centre and the size of

[1] Notice that $|o| + r$ *is not* a correct measure of the size as implicitly it would mean encoding the radius in unary, something unreasonable [13].

the longest strings in the ball. We will write $\boldsymbol{\mathcal{GB}}(\Sigma)$ for the class of all the good balls (and that of the corresponding grammars).

A *deterministic finite automaton* (DFA) is a 5-tuple $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ *s.t.* Q is a set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states and $\delta : Q \times \Sigma \to Q$ is a transition function. Every DFA can be *completed* with one sink state *s.t.* δ is a total function. As usual, δ is extended to $\Sigma^\star$. The *language recognized by* A is $\mathbb{L}(A) = \{w \in \Sigma^\star : \delta(q_0, w) \in F\}$. The size of A is $|Q|$. We will write $\boldsymbol{\mathcal{DFA}}(\Sigma)$ for the class of all DFA over the alphabet Σ.

The inference of DFA has been intensively studied for forty years at least [2, 6, 15]. On the other hand, the learnability of the balls is a recent issue [14] motivated by the problem of identifying languages from noisy data. However, our approach raises a preliminary question: if any ball whose grammar would have size n could be recognized by a DFA with $p(n)$ states (for some polynomial $p()$), then one could deduce learnability results on the former from (known) learnability results on the latter. Yet it is generally believed (albeit still an open question) that the transformation of a ball into a DFA is not polynomial [16].

3 PAC-learnability

The PAC paradigm [4] has been widely used in machine learning. It aims at building, with high confidence, good approximations of an unknown concept.

Definition 1 (ϵ-good hypothesis). *Let G be the target grammar and H be a hypothesis grammar. Let $\mathcal{D}$ be a distribution over $\Sigma^\star$ and $\epsilon > 0$. We say that H is an ϵ-*good hypothesis *w.r.t. G if$_{def}$ $Pr_{\mathcal{D}}(x \in \mathbb{L}(G) \oplus \mathbb{L}(H)) < \epsilon$.*

The PAC-learnability of grammars from strings of unbounded size has always been tricky [17, 10, 18]. Indeed, with the standard definition, a PAC-Learner can ask an Oracle to return a sample randomly drawn according to the distribution $\mathcal{D}$. However, in the case of strings, there is always the risk (albeit small) to sample a string too long to account for in polynomial time. In order to avoid this problem, we will sample from a distribution restricted to strings shorter than a specific value given by the following lemma:

Lemma 1. *Let $\mathcal{D}$ be a distribution over $\Sigma^\star$. Then $\forall \epsilon, \delta > 0$, with probability at least $1 - \delta$, if one draws a sample X of at least $\frac{1}{\epsilon} \ln \frac{1}{\delta}$ strings following $\mathcal{D}$, then the probability of any new string x to be longer than all the strings of X is less than ϵ. Formally, let $\mu_X = \max\{|y| : y \in X\}$, then $Pr_{x \sim \mathcal{D}}(|x| > \mu_X) < \epsilon$.*

Proof. Let ℓ be the smallest integer *s.t.* $Pr_{\mathcal{D}}(\Sigma^{>\ell}) < \epsilon$. A sufficient condition for $Pr_{\mathcal{D}}(|x| > \mu_X) < \epsilon$ is that we take a sample X large enough to be nearly sure (with probability $> 1 - \delta$) to have one string $\geq \ell$. Basically, the probability of drawing n strings in X of length $< \ell$ is $\leq (1 - \epsilon)^n$. So the probability of getting at least one string of length $\geq \ell$ is $> 1 - (1 - \epsilon)^n$. In order to build X, we thus need $1 - (1 - \epsilon)^n > 1 - \delta$, that is to say, $(1 - \epsilon)^n < \delta$. As $(1 - \epsilon)^n \leq e^{-n\epsilon}$, it is sufficient to take $n \geq \frac{1}{\epsilon} \ln \frac{1}{\delta}$ to reach a convenient value for μ_X. □

An algorithm is now asked to learn a grammar given a *confidence* parameter δ and an *error* parameter ϵ. The algorithm must also be given an upper bound n on the size of the target grammar and an upper bound m on the length of the examples it is going to get (perhaps computed using Lemma 1). The algorithm can query an Oracle for an example randomly drawn according to the distribution $\mathcal{D}$. The query of an example or a counter-example will be denoted Ex(). When the Oracle is only queried for a positive example, we will write Pos-Ex(). And when the Oracle is only queried for string of length $\leq m$, we will write Ex(m) and Pos-Ex(m) respectively. Formally, the Oracle will then return a string drawn from $\mathcal{D}$, or $\mathcal{D}(\mathbb{L}(G))$, or $\mathcal{D}(\Sigma^{\leq m})$, or $\mathcal{D}(\mathbb{L}(G) \cap \Sigma^{\leq m})$, respectively, where $\mathcal{D}(L)$ is the restriction of $\mathcal{D}$ to the strings of L: $Pr_{\mathcal{D}(L)}(x) = Pr_{\mathcal{D}}(x)/Pr_{\mathcal{D}}(L)$ if $x \in L$, 0 otherwise. $Pr_{\mathcal{D}(L)}(x)$ is not defined if $L = \emptyset$.

Definition 2 (Polynomial Pac-learnability). *Let $\boldsymbol{\mathcal{G}}$ be a class of grammars. $\boldsymbol{\mathcal{G}}$ is* Pac-learnable *if$_{def}$ there exists an algorithm $\mathfrak{A}$ s.t. $\forall \epsilon, \delta > 0$, for any distribution $\mathcal{D}$ over $\Sigma^\star$, $\forall n \in \mathbb{N}$, $\forall G \in \boldsymbol{\mathcal{G}}$ of size $\leq n$, for any upper bound $m \in \mathbb{N}$ on the size of the examples, if $\mathfrak{A}$ has access to* Ex*(), ϵ, δ, n and m, then with probability $> 1 - \delta$, $\mathfrak{A}$ returns an ϵ-good hypothesis w.r.t. G. If $\mathfrak{A}$ runs in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n and m, we say that $\boldsymbol{\mathcal{G}}$ is* polynomiallly Pac-learnable.

Typical techniques proving non Pac-learnability depend on complexity assumptions [19]. Let us recall that $\mathcal{RP}$ (Randomised Polynomial Time) is the complexity class of decision problems for which a probabilistic Turing machine exists which (1) runs in time polynomial in the input size, (2) on a negative instance, always returns No and (3) on a positive instance, returns Yes with probability $> \frac{1}{2}$ (otherwise, it returns No). The algorithm is randomised: it is allowed to flip a random coin while it is running. The algorithm does not make any error on negative instances, and it is important to remark that on positive instances, since the error is $< \frac{1}{2}$, by repeating the run of the algorithm as many times as necessary, the actual error can be brought to be as small as one wants. We will use the strong belief and assumption that $\mathcal{RP} \neq \mathcal{NP}$ [13].

We are going to show that the good balls are not polynomially Pac-learnable. The proof follows the classical lines for such results: we first prove that the associated consistency problem is $\mathcal{NP}$-hard, through reductions from a well known $\mathcal{NP}$-complete problem (*Longest Common Subsequence*). Then it follows that if a polynomial Pac-learning algorithm for balls existed, this algorithm would provide us with a proof that this $\mathcal{NP}$-complete problem would also be in $\mathcal{RP}$.

Lemma 2. *The following problems are $\mathcal{NP}$-complete:*

1. ***Longest Common Subsequence** (*Lcs*): Given n strings $x_1, \ldots, x_n$ and an integer k, does there exist a string w which is a subsequence of each x_i and is of length k?*
2. ***Longest Common Subsequence of Strings of a Given Length** (*Lcssgl*): Given n strings $x_1, \ldots, x_n$ all of length $2k$, does there exist a string w which is a subsequence of each x_i and is of length k?*
3. ***Consistent ball** (*Cb*): Given two sets X_+ and X_- of strings, does there exist a good ball containing X_+ and which does not intersect X_-?*

Proof. (1) See [20]. (2) See [21, page 42], Problem LCS0. (3) We use a reduction of Problem LCSSGL. We take the strings of length $2k$, and put these with string λ into the set X_+. We build X_- by taking each string of length $2k$ and inserting every possible symbol once only (hence constructing at most $n(2k+1)|\Sigma|$ strings of size $2k+1$). It follows that a ball that contains X_+ but no element of X_- has necessarily a centre of length k and a radius of k (since we focus on good balls only). The centre is then a subsequence of all the strings of length $2k$ that were given. Conversely, if a ball is built using a subsequence of length k as centre, this ball is of radius k, contains also λ, and because of the radius, contains no element of X_-. Finally the problem is in $\mathcal{NP}$, since given a centre o, it is easy to check if $\max_{x \in X_+} d(o,x) < \min_{x \in X_-} d(o,x)$. □

Theorem 1. *Unless $\mathcal{RP} = \mathcal{NP}$, $\boldsymbol{\mathcal{GB}}(\Sigma)$ is not polynomially* PAC*-learnable.*

Proof. Suppose that $\boldsymbol{\mathcal{GB}}(\Sigma)$ is polynomially PAC-learnable with $\mathfrak{A}$ and take an instance $\langle X_+, X_- \rangle$ of Problem CB. We write $h = |X_+| + |X_-|$ and define over $\Sigma^\star$ the distribution $Pr(x) = \frac{1}{h}$ if $x \in X_+ \cup X_-$, 0 if not. Let $\epsilon = \frac{1}{h+1}$, $\delta < \frac{1}{2}$, $m = n = \max\{|w| : w \in X_+\}$. Let $B_r(o)$ be the ball returned by $\mathfrak{A}(\epsilon, \delta, n, m)$ and test if ($X_+ \subseteq B_r(o)$ and $X_- \cap B_r(o) = \emptyset$). If there is no consistent ball, then $B_r(o)$ is inconsistent with the data, so the test is false. If there is a consistent ball, then $B_r(o)$ is ϵ-good, with $\epsilon < \frac{1}{h}$. So, with probability at least $1 - \delta > \frac{1}{2}$, there is no error at all and the test is true. This procedure runs in polynomial time in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n and m. So if the good balls were PAC-learnable, there would be a randomized algorithm for the $\mathcal{NP}$-complete CB Problem (by Lemma 2). □

Concerning the PAC-learnability of the DFA, a lot of studies have been done [6, 10, 18, 19] The associated consistency problem is hard [22] and an efficient learning algorithm could be used to invert the RSA encryption function [10]:

Theorem 2 ([10]). $\boldsymbol{\mathcal{DFA}}(\Sigma)$ *is not polynomially* PAC*-learnable.*

In certain cases, it may even be possible to PAC-learn from positive examples only. In this setting, during the learning phase, the examples are sampled following POS-EX() whereas during the testing phase, the sampling is done following EX(), but in both cases the distribution is identical. Again, we can sample using POS-EX(m), where m is obtained by using Lemma 1 and little additional cost. For any class $\boldsymbol{\mathcal{L}}$ of languages, we get:

Lemma 3. *If $\boldsymbol{\mathcal{L}}$ contains 2 languages L_1 and L_2 s.t. $L_1 \cap L_2 \neq \emptyset$, $L_1 \not\subset L_2$ and $L_2 \not\subset L_1$, then $\boldsymbol{\mathcal{L}}$ is not polynomially* PAC*-learnable from positive examples only.*

Proof. Let $w_1 \in L_1 - L_2$, $w_2 \in L_2 - L_1$ and $w_3 \in L_1 \cap L_2$. Consider the distribution $\mathcal{D}_1$ *s.t.* $\Pr_{\mathcal{D}_1}(w_1) = \Pr_{\mathcal{D}_1}(w_3) = \frac{1}{2}$ and the distribution $\mathcal{D}_2$ *s.t.* $\Pr_{\mathcal{D}_2}(w_2) = \Pr_{\mathcal{D}_2}(w_3) = \frac{1}{2}$. Basically, if one learns either L_1 from positive examples drawn according to $\mathcal{D}_2$, or L_2 from positive examples drawn according to $\mathcal{D}_1$, only the string w_3 will be used. However, the error will be $\geq \frac{1}{2}$. □

Theorem 3. *(1) $\boldsymbol{\mathcal{GB}}(\Sigma)$ and (2) $\boldsymbol{\mathcal{DFA}}(\Sigma)$ are not polynomially* PAC*-learnable from positive examples only.*

Proof. An immediate consequence of Lemma 3 with $L_1 = B_1(a)$, $L_2 = B_1(b)$, $w_1 = aa$, $w_2 = bb$ and $w_3 = ab$. □

4 Query learning

Learning from queries involves the Learner (he) being able to interrogate the Oracle (she) using queries from a given set [3]. The goal of the Learner is to identify a grammar of an unknown language L. The Oracle knows L and properly answers to the queries (*i.e.*, she does not lie). Below, we will use three kinds of queries. With the Membership Queries (MQ), the Learner submits a string w to the Oracle and she answers YES if $w \in L$, NO otherwise. With the Equivalence Queries (EQ), he submits (the grammar of) a language K and she answers YES if $K = L$, and a string belonging to $K \oplus L$ otherwise. With the Correction Queries based on the Edit Distance (CQ$_{\text{EDIT}}$), he submits a string w and she answers YES if $w \in L$, and any correction $z \in L$ at minimum edit distance of w otherwise.

Definition 3. *A class $\boldsymbol{\mathcal{G}}$ is* polynomially identifiable from queries *if$_{def}$ there is an algorithm $\mathfrak{A}$ able to identify every $G \in \boldsymbol{\mathcal{G}}$ s.t. at any call of a query, the total number of queries and of time used up to that point by $\mathfrak{A}$ is polynomial both in $\|G\|$ and in the size of the information presented up to that point by the Oracle.*

In the case of good balls, we have shown:

Theorem 4 ([14]). *(1) $\boldsymbol{\mathcal{GB}}(\Sigma)$ is not polynomially identifiable from* MQ *and* EQ. *(2) $\boldsymbol{\mathcal{GB}}(\Sigma)$ is polynomially identifiable from* CQ$_{\text{EDIT}}$.

Notice however that if the Learner is given one string from a good ball, then he can learn using a polynomial number of MQ only.

Concerning the class of the DFA, we get:

Theorem 5. *(1) $\boldsymbol{\mathcal{DFA}}(\Sigma)$ is polynomially identifiable from* MQ *and* EQ *[15], but is not from (2)* MQ *only [3], nor (3)* EQ *only [5], nor (4)* CQ$_{\text{EDIT}}$ *only.*

Proof. (4) Let $\mathcal{A}_w$ denote the DFA that recognizes $\Sigma^\star \setminus \{w\}$. Let $n \in \mathbb{N}$ and $\boldsymbol{\mathcal{DFA}}_{\leq n} = \{\mathcal{A}_w : w \in \Sigma^{\leq n}\}$. Following [5], we describe an Adversary that maintains a set X of all the possible DFA. At the beginning, $X = \boldsymbol{\mathcal{DFA}}_{\leq n}$. Each time the correction of any string w is demanded, the Adversary answers YES and eliminates only one DFA of X: $\mathcal{A}_w$. As there is $\Omega(|\Sigma|^n)$ DFA in $\boldsymbol{\mathcal{DFA}}_{\leq n}$, identifying one of them will require $\Omega(|\Sigma|^n)$ queries in the worst case. □

5 Polynomial identification in the limit

Identification in the limit [2] is standard: a Learner receives an infinite sequence of information (presentation) that should help him to find the grammar $G \in \boldsymbol{\mathcal{G}}$ of an unkown target language $L \in \boldsymbol{\mathcal{L}}$. The set of admissible presentations is denoted by **Pres**, each presentation being a function $\mathbb{N} \to X$ where X is any set. Given $\boldsymbol{f} \in$ **Pres**, we will write $\boldsymbol{f}_m$ for the $m+1$ first elements of $\boldsymbol{f}$, and $\boldsymbol{f}(n)$ for its n^{th} element. Below, we will consider two sorts of presentations. When **Pres**=TEXT, all the strings in L are presented: $\boldsymbol{f}(\mathbb{N}) = \mathbb{L}(G)$. When **Pres**=INFORMANT, a presentation is of labelled pairs (w, l) where $(w \in L \Rightarrow l = +)$ and $(w \notin L \Rightarrow l = -)$: $\boldsymbol{f}(\mathbb{N}) = \mathbb{L}(G) \times \{+\} \cup \overline{\mathbb{L}(G)} \times \{-\}$; we will write **Pres** = PRESENTATION for all the results that concern both TEXT and INFORMANT.

Definition 4. *We say that $\boldsymbol{\mathcal{G}}$ is* identifiable in the limit from **Pres** *if_{def} there exists an algorithm $\mathfrak{A}$ s.t. for all $G \in \boldsymbol{\mathcal{G}}$ and for any presentation $\boldsymbol{f}$ of $\mathbb{L}(G)$, there exists a rank n s.t. for all $m \geq n$, $\mathfrak{A}(\boldsymbol{f}_m) = \mathfrak{A}(\boldsymbol{f}_n)$ and $\mathbb{L}(\mathfrak{A}(\boldsymbol{f}_n)) = \mathbb{L}(G)$.*

This definition yields a number of learnability results. However, the absence of efficiency constraints often leads to unusable algorithms. Firstly, it seems reasonable that the amount of time an algorithm has to learn should be bounded:

Definition 5 (Polynomial Update Time). *An algorithm $\mathfrak{A}$ is said to have* polynomial update time *if_{def} there is a polynomial $p()$ s.t., for every presentation $\boldsymbol{f}$ and every integer n, computing $\mathfrak{A}(\boldsymbol{f}_n)$ requires $\mathcal{O}(p(\|\boldsymbol{f}_n\|))$ time.*

It is known that polynomial update time is not sufficient [6]: a Learner could receive an exponential number of examples without doing anything but wait, and then use the amount of time he saved to solve any $\mathcal{NP}$-hard problem... Polynomiality should also concern the minimum amount of data that any Learner requires:

Definition 6 (Polynomial Characteristic Sample). *We say that $\boldsymbol{\mathcal{G}}$ admits* polynomial characteristic samples *if_{def} there exist an algorithm $\mathfrak{A}$ and a polynomial $p()$ s.t. for all $G \in \boldsymbol{\mathcal{G}}$, there exists* $\textsc{Cs} \subseteq X$ *s.t. (1) $\|\textsc{Cs}\| \leq p(\|G\|)$, (2) $\mathbb{L}(\mathfrak{A}(\textsc{Cs})) = \mathbb{L}(G)$ and (3) for all $\boldsymbol{f} \in$ **Pres**, for all $n \geq 0$, if $\textsc{Cs} \subseteq \boldsymbol{f}_n$ then $\mathfrak{A}(\boldsymbol{f}_n) = \mathfrak{A}(\textsc{Cs})$. Such a set* Cs *is called a* characteristic sample *of G for $\mathfrak{A}$. If $\mathfrak{A}$ exists, we say that $\boldsymbol{\mathcal{G}}$* is identifiable in the limit in Cs polynomial time.

Lastly, polynomiality may concern either the number of implicit prediction errors [6] or the number of mind changes (Mc) [23] done by the learner:

Definition 7 (Implicit Prediction Errors). *We say that an algorithm $\mathfrak{A}$ makes an* implicit prediction error *(*Ipe*) at time n of a presentation $\boldsymbol{f}$ if_{def} $\mathfrak{A}(\boldsymbol{f}_{n-1}) \not\vdash \boldsymbol{f}(n)$. $\mathfrak{A}$ is called* consistent *if_{def} it changes its mind each time a prediction error is detected with the new presented element.*

$\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit in Ipe *polynomial time if_{def} (1) $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit, (2) $\mathfrak{A}$ has polynomial update time and (3) $\mathfrak{A}$ makes a polynomial number of implicit prediction errors: let $\#\textsc{Ipe}(\boldsymbol{f}) = |\{k \in \mathbb{N} : \mathfrak{A}(\boldsymbol{f}_k) \not\vdash \boldsymbol{f}(k+1)\}|$; there exists a polynomial $p()$ s.t. for each $G \in \boldsymbol{\mathcal{G}}$ and each presentation $\boldsymbol{f}$ of $\mathbb{L}(G)$, $\#\textsc{Ipe}(\boldsymbol{f}) \leq p(\|G\|)$.*

Definition 8 (Mind Changes). *We say that an algorithm $\mathfrak{A}$* changes its mind *(*Mc*) at time n of presentation $\boldsymbol{f}$ if_{def} $\mathfrak{A}(\boldsymbol{f}_n) \neq \mathfrak{A}(\boldsymbol{f}_{n-1})$. $\mathfrak{A}$ is called* conservative *if_{def} it never changes its mind when the current hypothesis is consistent with the new presented element.*

$\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit in Mc *polynomial time if_{def} (1) $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in the limit, (2) $\mathfrak{A}$ has polynomial update time and (3) $\mathfrak{A}$ makes a polynomial number of mind changes: Let $\#\textsc{Mc}(\boldsymbol{f}) = |\{k \in \mathbb{N} : \mathfrak{A}(\boldsymbol{f}_k) \neq \mathfrak{A}(\boldsymbol{f}_{k+1})\}|$; there exists a polynomial $p()$ s.t. for each $G \in \boldsymbol{\mathcal{G}}$ and each presentation $\boldsymbol{f}$ of $\mathbb{L}(G)$, $\#\textsc{Mc}(\boldsymbol{f}) \leq p(\|G\|)$.*

Concerning both last notions, one can notice that if an algorithm $\mathfrak{A}$ is consistent then $\#\textsc{Ipe}(\boldsymbol{f}) \leq \#\textsc{Mc}(\boldsymbol{f})$ for every presentation $\boldsymbol{f}$. Likewise, if $\mathfrak{A}$ is conservative then $\#\textsc{Mc}(\boldsymbol{f}) \leq \#\textsc{Ipe}(\boldsymbol{f})$. So we deduce the following lemma:

Lemma 4. *If $\mathfrak{A}$ identifies the class $\boldsymbol{\mathcal{G}}$ in* Mc *polynomial time and is consistent, then $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in* Ipe *polynomial time. Conversely, if $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in* Ipe *polynomial time and is conservative, then $\mathfrak{A}$ identifies $\boldsymbol{\mathcal{G}}$ in* Mc *polynomial time.*

5.1 Polynomial identification from text

The aim of this section is to show the following result:

Theorem 6. *$\boldsymbol{\mathcal{GB}}(\Sigma)$ is identifiable in the limit from* Text *in (1)* Mc *polynomial time, (2)* Ipe *polynomial time and (3)* Cs *polynomial time.*

Notice that as the Dfa recognize a *superfinite* class of languages (*i.e.*, containing all the finite languages and at least one infinite language), it is impossible to identify the class in the limit from positive examples only:

Theorem 7 ([2]). *$\boldsymbol{\mathcal{DFA}}(\Sigma)$ is not identifiable in the limit from* Text.

In order to prove Theo. 6, we will need to build the minimum consistent good ball containing a set $X = \{x_1, \ldots, x_n\}$ of strings (sample). This problem is $\mathcal{NP}$-hard but some instances are efficiently solvable. Let X^{max} (*resp.* X^{min}) denote the set of all longest (*resp.* shortest) strings of X. A *minimality fingerprint* is a subset $\{u, v, w\} \subseteq X$ *s.t.* (1) $u, v \in X^{max}$, (2) $w \in X^{min}$, (3) $|u| - |w| = 2r$ for some $r \in \mathbb{N}$, (4) u and v have only one longest common subsequence, that is, $lcs(u, v) = \{o\}$ for some $o \in \Sigma^\star$, (5) $|o| = |u| - r$ and (6) $X \subseteq B_r(o)$.

Checking if X contains a minimality fingerprint, and computing o and r can be achieved in polynomial time (in $\|X\|$). Indeed, the only critical point is that the cardinal of $lcs(u, v)$ may be $> 1.442^n$ [24] (where $n = |u| = |v|$); nevertheless, a data structure such as the Lcs-graph [25] allows one to conclude polynomially. Moreover, the minimality fingerprints are meaningful. Indeed, only the properties of the edit distance are needed to show that if X contains a minimality fingerprint $\{u, v, w\}$ for the ball $B_r(o)$ and $X \subseteq B_{r'}(o')$, then either $r' > r$, or ($r' = r$ and $o' = o$). In other words, $B_r(o)$ is the smallest ball containing X *w.r.t.* the radius.

We can now consider Algo. 1. This algorithm does identify $\boldsymbol{\mathcal{GB}}(\Sigma)$ in the limit since if $B_r(o)$ denotes the target ball, then at some point, the algorithm will meet the strings $u = a^r o, v = b^r o$, and some w of length $|o| - r$ that constitute a minimality fingerprint for $B_r(o)$. Moreover, it obviously has a polynomial update time. Finally, it makes a polynomial number of Mc. Indeed, it only changes its hypothesis in favour of a valid ball if the ball has a larger radius than all the valid balls it has ever conjecture, that may happen $\leq r$ times. And it only changes its hypothesis in favour of a junk ball if either it must abandon a valid ball, or if the actual junk ball does not contain all the examples, that may happen $\leq r + 2r$ times. So the total number of Mc is $\leq 4r$. So Claim (1) holds.

Concerning Claim (2), note that Algo. 1 is consistent (thanks to the use of the junk balls), thus Claim (2) holds by Lemma 4. Lastly, every minimality fingerprint is a characteristic set that makes Algo. 1 converge, so Claim (3) holds.

Algorithm 1: Identification of good balls from text.

```
Data: A text $\boldsymbol{f} = \{x_1, x_2, \ldots\}$
read($x_1$); $c \leftarrow x_1$; output $(x_1, 0)$;
while true do
    read($x_i$);
    if $\boldsymbol{f}_i$ is a minimality fingerprint for $B_r(o)$ then
        output $(o, r)$ (* valid ball *)
    else
        if $c \notin \boldsymbol{f}_i^{max}$ then $c \leftarrow$ any string in $\boldsymbol{f}_i^{max}$;
        output $(c, |c|)$ (* junk ball *)
    end
end
```

5.2 Polynomial identification from informant

Theorem 8. *(1) $\mathcal{GB}(\Sigma)$ is not identifiable from* Informant *in* Ipe *polynomial time, but is identifiable in (2)* Mc *polynomial time and (3)* Cs *polynomial time.*

Proof. (1) Similar proof as that of [6] for the Dfa: if $\mathcal{GB}(\Sigma)$ was identifiable in Ipe polynomial time from Informant, then $\mathcal{GB}(\Sigma)$ would be polynomially identifiable from Eq, that contradicts Theo. 4. (2) As the hypotheses are not necessarily consistent with the data, one can use Algo. 1, ignoring the negative examples. (3) Same characteristic sets as those of Theo. 6, Claim (3). □

Theorem 9. *(1) $\mathcal{DFA}(\Sigma)$ is not identifiable from* Informant *in* Ipe *polynomial time [6], but is identifiable in (2)* Mc *polynomial time and (3)* Cs *polynomial time [26, 27].*

Let us prove Claim (2) with minimality fingerprints again. We say that $X = \langle X_+, X_- \rangle$ contains a *minimality fingerprint* if_{def} the following conditions hold: (1) let $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ be the Dfa computed by Rpni [27] on X possibly completed with one hole state; (2) for all $q \in Q$, the smallest string w_q *w.r.t.* the hierarchical order $\trianglelefteq$ *s.t.* $\delta(q_0, w_q) = q$ belongs to either X_+ if $q \in F$, or X_- if $q \notin F$; (3) for all $q \in Q, a \in \Sigma$, $w_q a$ belongs to either X_+ if $\delta(q, a) \in F$, or X_- if $\delta(q, a) \notin F$; (4) for all $p, q, r \in Q, a \in \Sigma$ *s.t.* $\delta(p, a) = q \neq r$, there exists $f \in \Sigma^\star$ *s.t.* either $(w_p a f \in X_+, w_r f \in X_-)$, or $(w_p a f \in X_-, w_r f \in X_+)$.

Notice that not all the Dfa have minimality fingerprints. Moreover, every fingerprint contains a characteristic sample of A for Rpni [27], plus new information: actually, *all* the states and the transitions of A are determined by the fingerprint, so any other complete Dfa A' compatible with X necessarily has more states than A. A is thus the unique complete minimal Dfa compatible with X. Lastly, checking if X contains a minimality fingerprint, and computing A is achievable in polynomial time (in $\|X\|$).

We can now define a Learner $\mathfrak{A}$. At each step, $\mathfrak{A}$ tests if $\boldsymbol{f}_i$ contains a minimality fingerprint. If yes, $\mathfrak{A}$ changes its mind in favour of the Dfa A_i returned by Rpni on $\boldsymbol{f}_i$. If no, $\mathfrak{A}$ returns the previous hypothesis (that may not be consistent). Clearly, the number of states of A_i strictly increases (thanks to the fingerprints). As this number is bounded by the number of states of the target Dfa A, we get $\#\text{Mc}(\boldsymbol{f}) \leq \|A\|$. Moreover, at some point, a fingerprint (thus a characteritic set of A) will appear in the data, and then $\mathfrak{A}$ will converge. $\square$

6 Conclusion

In this paper, we have performed a systematic study of two classes of languages whose definitions is based on very different principles (see Table 1). Clearly, the goal of this work was not to prove that any paradigm is equivalent or better than any other: comparing two classes is not sufficient. Nevertheless, we have shown that several hints were wrong. For instance, it is wrong to think that the identification in Mc polynomial time implies the identification in Ipe polynomial time. It is also wrong to think that it is easier to learn from positive and negative examples (Informant) than from positive examples only (Text) (because in some paradigm, misclassifying negative examples is expensive in terms of complexity). Finally, one can note that the good balls are not learnable from a polynomial number of Mq and Eq, that is the case of the Dfa. As the balls are finite languages, they are recognizable with Dfa. Thus a subclass of a learnable class could be non learnable! We conjecture, following [28, 16], that this is because the size of the minimal Dfa recognizing a ball is exponential in the size of the ball.

Table 1. A synthetic view of the results presented in this paper. † marks the theorems proved above. Due to lack of space, the results concerning $\mathcal{BALL}(\Sigma)$ are just claimed.

Criterion	$\mathcal{GB}(\Sigma)$		$\mathcal{DFA}(\Sigma)$		$\mathcal{BALL}(\Sigma)$
Pac Inform.	No†	Theo. 1	No	Theo. 2	No
Pac Text	No†	Theo. 3 (1)	No†	Theo. 3 (2)	No
Ipe Inform.	No†	Theo. 8 (1)	No	Theo. 9 (1)	No
Ipe Text	Yes†	Theo. 6 (2)	No	Theo. 7	No
Mc Inform.	Yes†	Theo. 8 (2)	Yes†	Theo. 9 (2)	Yes
Mc Text	Yes†	Theo. 6 (1)	No	Theo. 7	No
Cs Inform.	Yes†	Theo. 8 (3)	Yes	Theo. 9 (3)	No
Cs Text	Yes†	Theo. 6 (3)	No	Theo. 7	No
Mq (or Eq)	No	Theo. 4 (1)	No	Theo. 5 (2,3)	No
Mq and Eq	No	Theo. 4 (1)	Yes	Theo. 5 (1)	No
CQ_{Edit}	Yes	Theo. 4 (2)	No	Theo. 5 (4)	No

References

1. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4) (1965) 845–848
2. Gold, E.M.: Language identification in the limit. Information and Control **10**(5) (1967) 447–474
3. Angluin, D.: Queries and concept learning. Machine Learning Journal **2** (1987) 319–342

4. Valiant, L.G.: A theory of the learnable. Communications of the ACM **27**(11) (1984) 1134–1142
5. Angluin, D.: Negative results for equivalence queries. Machine Learning Journal **5** (1990) 121–150
6. Pitt, L.: Inductive inference, DFA's, and computational complexity. In: Proc. Analogical and Inductive Inference (AII'89), LNAI 397 (1989) 18–44
7. Li, M., Vitanyi, P.: Learning simple concepts under simple distributions. Siam J. of Comput. **20** (1991) 911–935
8. Parekh, R.J., Honavar, V.: On the relationship between models for learning in helpful environments. In: Proc. ICGI'00, LNAI 1891 (2000) 207–220
9. Haussler, D., Kearns, M.J., Littlestone, N., Warmuth, M.K.: Equivalence of models for polynomial learnability. Information and Computation **95**(2) (1991) 129–161
10. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. In: 21st ACM Symp. on Theory of Computing. (1989) 433–444
11. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning Journal **27** (1997) 125–138
12. Wagner, R., Fisher, M.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–178
13. Papadimitriou, C.M.: Computational Complexity. Add.–Wesley, New York (1994)
14. Becerra-Bonache, L., de la Higuera, C., Janodet, J.C., Tantini, F.: Learning balls of strings with correction queries. In: Proc. ECML'07, LNAI 4701 (2007) 18–29
15. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Control **39** (1987) 337–350
16. Navarro, G.: A guided tour to approximate string matching. ACM computing surveys **33**(1) (2001) 31–88
17. Warmuth, M.: Towards representation independence in *pac*-learning. In: Proc. Analogical and Inductive Inference (AII'89), LNAI 397 (1989) 78–103
18. Kearns, M., Vazirani, U.: An Introduction to Computational Learning Theory. MIT press (1994)
19. Pitt, L., Valiant, L.G.: Computational limitations on learning from examples. Journal of the ACM **35**(4) (1988) 965–984
20. Maier, D.: The complexity of some problems on subsequences and supersequences. Journal of the ACM **25** (1977) 322–336
21. de la Higuera, C., Casacuberta, F.: Topology of strings: Median string is NP-complete. Theoretical Computer Science **230** (2000) 39–48
22. Pitt, L., Warmuth, M.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the ACM **40**(1) (1993) 95–142
23. Angluin, D., Smith, C.: Inductive inference: theory and methods. ACM computing surveys **15**(3) (1983) 237–269
24. Greenberg, R.I.: Bounds on the number of longest common subsequences. Technical report, Loyola University (2003) http://arXiv.org/abs/cs/0301030v2.
25. Greenberg, R.I.: Fast and simple computation of all longest common subsequences. Technical report, Loyola University (2002) http://arXiv.org/abs/cs.DS/0211001.
26. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37** (1978) 302–320
27. Oncina, J., García, P.: Identifying regular languages in polynomial time. In: Advances in Structural and Syntactic Pattern Recognition. Volume 5 of Series in Machine Perception and Artificial Intelligence. World Scientific (1992) 99–108
28. Schulz, K.U., Mihov, S.: Fast string correction with Levenshtein automata. Int. Journal on Document Analysis and Recognition **5**(1) (2002) 67–85