Learning probabilistic languages by k-testable machines

Chu, W.; Bonsangue, M.M.; Aoki, T.; Li, Q.

Learning Probabilistic Languages by k-Testable Machines

Wenjing Chu* and Marcello Bonsangue
Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
* Corresponding author: chuw@liacs.leidenuniv.nl

2020 International Symposium on Theoretical Aspects of Software Engineering (TASE) | 978-1-7281-4086-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/TASE49443.2020.00026

***Abstract*—A k-testable machine is a finite automaton which recognizes a language L by only seeing a window of size k of each string in L. In this paper we use k-testable machines to recognize probabilistic languages and propose a novel algorithm to learn them. We work in the context of passive learning as our algorithm is based on a finite sample of strings belonging to the target language equipped with frequencies.**

Because our algorithm learns a probabilistic automaton, the resulting language is less sensitive to noise threshold than García's algorithm. When compared with the ALERGIA learning algorithm, our method provides a better result in the case of the target language being a k-testable language. In fact, in this case, for the given window k we can learn at the limit the target language exactly.

***Index Terms*—automata, k-testable language, probabilistic finite automata, passive learning**

I. Introduction

Probabilistic finite automata have been used to model a distribution over strings. They have been widely applied in several fields, such as computational biology [1] and speech recognition [2]. Therefore, learning probabilistic finite automata has been an important issue. In 1967, Gold [3] introduced the criterion of identification in the limit for successful learning a language and proved that regular languages cannot be identified by positive sample (text) only. However, Angluin proved that this is not the case for a wide range of distribution classes, including probabilistic regular languages [4], which can be identified from positive samples with probability 1. Since then, several algorithms of learning probabilistic regular languages have been published. The ALERGIA algorithm was developed by Rafael Carrasco and Jose Oncina [5], who also published a simpler version, the RLIPS algorithm, in 1999 [6]. Another algorithm for learning only acyclic automata was developed by Ron *et al.* [7].

Gold's results are very strict with respect to the class of regular languages. To cope with these, Angluin pointed out that it is possible to approach certain particular subclasses of languages [8]. Following Angluin, García introduced an algorithm for learning k-testable ($k - \mathcal{TSS}$) languages [9]. Practically, learning $k-$testable has been used in a wide range of applications [10]–[12], such as biological sequences, aural pattern recognition and sequence classification. Furthermore, k-testable probabilistic languages in the strict sense are directly related to order-k Markov source [13], which has been successfully utilized in pattern recognition system [14]. On the other hand, probabilistic languages can approximately characterize natural languages, and the accuracy increases as more data is taken into account [15], [16]. However, there is no room for noise in García's algorithm [17]: once a string is accepted by the learning automata it will always be in every refinements, even if the string do not belong to the language to be learn. In other words, the fault tolerance of García's algorithm is very low. Our research is motivated by this problem.

In this paper we provide an algorithm for passively learning probabilistic regular language given a finite set of strings together with some frequency (a sample), which allows for the identification of k-testable probabilistic languages. Our approach is different from active learning because it does not require any extra information about the sample. We show that it can learn a target language with probability 1 in the limit of infinite data. Our approach is based on a probabilistic extension of learning k-testable languages.

II. Preliminaries

In this section, we present some necessary background, mainly to fix the notation. Given an alphabet Σ, a language L is a subset of Σ^*. If s is a string in Σ^* and $s = tuv$ for three strings t, u and v, then t is said to be a prefix of s, v is a suffix of s, and u is a substring of s.

In this paper we are interested to learning regular languages, that is, languages recognized by a deterministic finite automaton.

Definition 1. ***Deterministic finite automaton*** *A deterministic finite automaton (DFA) is a tuple $A = \langle \Sigma, Q, q_\lambda, F_A, \delta \rangle$, where:*

- *Σ is the alphabet,*
- *Q is a finite set of states,*
- *$q_\lambda \in Q$ is the initial state,*
- *$\delta : Q \times \Sigma \to Q$ is a transition function,*
- *$F_A \subseteq Q$ is a set of final accepting states.*

The state $\delta(q, a)$ denotes the unique state in which a DFA goes to when reading the symbol a in a state q. As usual, we extend the transition function so to get as input strings rather than symbol s of the alphabet. We denote by $\delta^*(q, x)$ the state reached by a DFA after receiving a string x in a state q. The definition of the extended transition function is by induction on the length of the string:

978-1-7281-4086-5/20/$31.00 ©2020 IEEE
DOI 10.1109/TASE49443.2020.00026

Definition 2. *Let $A = \langle \Sigma, Q, q_\lambda, F_A, \delta \rangle$ be a finite automation. We define the extended transition function*

$$\delta^* : Q \times \Sigma^* \to Q \tag{1}$$

as follows:

- $\forall q \in Q$, $\delta^*(q, \lambda) = q$,
- $\forall q \in Q$, $\forall \omega \in \Sigma^*$, *and* $\forall a \in \Sigma$, $\delta^*(q, \omega a) = \delta(\delta^*(q, \omega), a)$

The extended transition function is used to define the language $L(A)$ recognized by a deterministic finite automaton A:

$$L(A) = \{\omega \in \Sigma^* | \delta^*(q_\lambda, \omega) \in F\}. \tag{2}$$

The class of languages recognized by a deterministic finite automaton is the one of regular languages [18]. In the sequel, we will be interested in a subset of regular languages, the $k-$testable language. Intuitively, these are languages that can be recognized by only observing no more than k consecutive symbols, either as prefix, suffix or substring. To characterize formally the class of k-testable language, [19] introduced a special machine:

Definition 3. ***k-testable machine*** *Given $k > 0$, a k-testable machine is a 5-tuple $Z_k = \langle \Sigma, I, F, T, C \rangle$ with:*

- *Σ is the alphabet,*
- *$I, F \subseteq \Sigma^{k-1}$ (prefixes of length $k-1$ and suffixes (or finals) of length $k-1$),*
- *$C \subseteq \Sigma^{<k}$ (short strings),*
- *$T \subseteq \Sigma^k$ (allowed segments).*

Given a k-testable machine $Z_k = \langle \Sigma, I, F, T, C \rangle$, the k-testable language recognized by it which can be defined by:

$$L(Z_k) = (I\Sigma^* \cap \Sigma^* F - \Sigma^*(\Sigma^k - T)\Sigma^*) \cup C$$

Informally, a k-testable language is a set of strings starting with strings in I, finishing with strings in F and containing strings in T, if their sizes are greater than or equal to k. Otherwise, they must belong to set C. There are, thus, two types of strings in $L(Z_k)$: strings of length less than k, that are defined by C, and strings of length greater or equal $k$that must contain substrings in the other sets I, T and F. Note that if $k = 1$, the language accepted by any $1-$testable machine is $\Sigma^\star$. This is because the sets I, F and C are $\{\lambda\}$, T equals Σ. See the example 1.

A k-testable language is a regular language for which its memory (i.e. the minimal number of states needed by a DFA to recognize it) can be bounded a priori. This follows from Definition 3, because the size of the window of visible symbols of a k-testable language is exactly k and the next symbol depend on the $k-1$ previous characters. In other words, k-testable languages are causal.

Even if all $k-$testable languages are regular languages, the converse is not true, for any k. For instance, consider the language which is defined by the regular expression $a\Sigma^* a + b\Sigma^* b$, which is not k-testable language (for any k), as the last symbol may depend on more than k previous one.

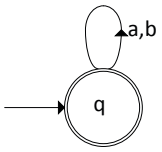

Fig. 1. A 1-testable language.

Example 1. *The 1-testable language recognized by the $1-$testable machine $Z_1 = \langle \Sigma = \{a, b\}, I = \{\lambda\}, F = \{\lambda\}, T = \{a, b\}, C = \{\lambda\} \rangle$ is the one recognized by the deterministic finite automaton in Figure 1 which basically accepts any strings.*

Example 2. *The automaton from Figure 2 recognises the language bb^*ab^*. This language is $3-$testable language because it can be recognized by a $3-$testable machine. But it is not a $2-$testable language, because with a window of size two, any 2-testable machine would accept the set of strings b^*.*

A probabilistic language D is a probability distribution over Σ^*. The probability of a string $x \in \Sigma^*$ under the distribution D is denoted by $P_D(x)$ and it must satisfy:

$$\sum_{x \in \Sigma^*} P_D(x) = 1\,. \tag{3}$$

Probabilistic regular languages are accepted by probabilistic finite automata [20].

Definition 4. ***Probabilistic finite automata*** *A deterministic probabilistic finite automaton(DPFA) is a tuple $A = \langle \Sigma, Q, q_\lambda, \mathbb{F}_p, \delta_p \rangle$, where:*

- *Σ is the alphabet,*
- *Q is a finite set of states,*
- *q_λ is the initial state,*
- *$\delta_p : Q \times \Sigma \times Q \to \mathbb{Q}^+ \cap [0, 1]$,*
- *$\mathbb{F}_p : Q \to \mathbb{Q}^+ \cap [0, 1]$ assigns final-state to probabilities,*
- *$\forall q \in Q, \forall a \in \Sigma, |\{q' : \delta_P(q, a, q') > 0\}| \leq 1$.*

The last conditions is about being deterministic, i.e. the automaton is deterministic when we assume that the transition function $\delta_p(q, a, q')$ is completely defined by q and a, in the sense that it is strictly positive for at most one state q'. A distribution over Σ^* said to be regular deterministic if it can be generated by DPFA.

To define the language, we need to introduce the extended transition function $\delta^* : Q \times \Sigma^* \rightharpoonup Q$ for deterministic probabilistic finite automata. It is defined by induction on the length of the string as follow:

- $\delta^*(q, \lambda) = q$
- $\delta^*(q, \omega a) = q'$, if $\delta(\delta^*(q, \omega), a, q') > 0$ and $\delta^*(q, \omega)$ is defined, else it is undefined.

Note that because of the deterministic assumption on a probabilistic automaton, the above function is well defined and partial.

The language accepted by a DPFA A is defined as follows:

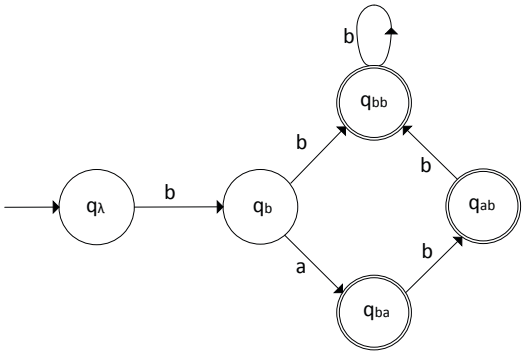

Fig. 2. A deterministic automaton for the $3-\mathcal{TSS}$ automaton $Z = \langle\{a,b\}, I=\{bb,ba\}, F=\{ab,bb,ba\}, T=\{bbb,bab,abb\}, C=\{ba\}\rangle$.

$$L(A) = \{\omega | \delta^*(q_\lambda, \omega) = q', \mathbb{F}_p(q') > 0, \}. \tag{4}$$

Next we describe how the probability associated to each string ω is computed, defining a distribution on P_D on Σ^* for a given PDFA. Basically, $P_D(\omega) = P_D(\omega, q_\lambda)$, where $P_D(\lambda, q) = F(q)$ and $P_D(a\omega) = \delta(q_\lambda, a, q') \cdot P_D(\omega, q')$. The assignment of probability to strings generalize to an assignment of probability to language by summing up the probability of each string in the language. This allows to define a distribution over Σ in a given context:

$$P_D(a|\omega) = \frac{P_D(\omega a \Sigma^*)}{P_D(\omega \Sigma^*)}. \tag{5}$$

Informally $P_D(a|\omega)$ is the probability of generating an a after having generated ω according to the distribution P_D.

Since our goal is to learn probabilistic finite automata, in practice we have access to frequencies instead of probabilities. For instance, we can get 10 out of 100 strings start with the letter a, not the probability $\frac{1}{10}$. For this propose, we introduce a deterministic frequency finite automaton (DFFA), which takes into account the number of times an event occurs.

Definition 5. ***Deterministic frequency finite automaton*** *A deterministic frequency finite automaton is a tuple* $A = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr} \rangle$ *where*

- Σ *is the alphabet,*
- Q *is a finite set of states,*
- $\mathbb{I}_{fr} : Q \to \mathbb{N}$ *(initial-state frequencies); since the automaton is deterministic there is exactly one state* q_λ *for which* $\mathbb{I}_{fr} \neq 0$*,*
- $\mathbb{F}_{fr} : Q \to \mathbb{N}$ *(final-state frequencies),*
- $\delta_{fr} : Q \times \Sigma \times Q \to \mathbb{N}$ *is the transition frequency function,*
- $\forall q \in Q, \forall a \in \Sigma, |\{q' : \delta_{fr}(q, a, q') > 0\}| \leq 1.$

The notation $\delta_{fr}(q, a, q') = n$ can be interpreted as "there is a transition from q to q' labelled with a that is used n times". In addition, there is a relationship between the frequencies of the transitions leading to a state and those leaving a state:

Definition 6. ***Consistency*** *A deterministic frequency finite automaton* $A = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}, \delta_A \rangle$ *is said to be consistent or well defined if* $\forall q \in Q$*,*

$$\mathbb{I}_{fr}(q) + \sum_{q' \in Q, a \in \Sigma} \delta_{fr}(q', a, q) = \mathbb{F}_{fr}(q) + \sum_{q' \in Q, a \in \Sigma} \delta_{fr}(q, a, q'). \tag{6}$$

When a deterministic frequency finite automaton is consistent, the number of strings entering and leaving a given state is identical. Consistency is defined as maintaining the flows: any string that enters a state (or starts in a state) has to leave it (or end there).

III. Inference Algorithm

In this section we introduce our learning algorithm. It consists of several steps. The first step is to construct a $A_{k-\mathcal{TSS}}$ machine from sample S. Let $S \subseteq \Sigma^*$ be a finite learning sample and $k \geq 1$, then we construct a k-testable machine $A_{k-\mathcal{TSS}}$ using an algorithm similar to [9]:

Algorithm 1 $A_{k-\mathcal{TSS}}$

Input: A sample S
Output: A $k-\mathcal{TSS}$ machine
1: Σ is the alphabet used in S
2: I(S) := $\Sigma^{k-1} \cap PREF(S)$
3: C(S) := $\Sigma^{<k} \cap S$
4: F(S) := $\Sigma^{k-1} \cap SUFF(S)$
5: T(S) := $\Sigma^k \cap \{v : uvw \in S, u, v, w \in \Sigma^\star\}$
6: **return** $\langle \Sigma, I(S), F(S), T(S), C(S) \rangle$

From the Algorithm 1, we see that the strings in the sample S of length less than k define the set C. While for those strings in S of length greater or equal to k, we cut all the prefixes of size exactly $k-1$ and put them in set I. Similarly, we cut all the suffixes of size $k-1$ and put them in set F. And we put all substrings of size k in the set T. In other words, this step of learning $k-$testable languages is to find the prefixes, substrings and suffixes of size $k-1$ that occur in the sample S.

Example 3. *Let us consider the sample* $S = \{a, aa, abba, ababa, ababab\}$*. Following the Algorithm 1, when* $k = 1$*, we get a machine* Z_1*:*

- $\Sigma = \{a, b\}$,
- $I(S) = \{\lambda\}$,
- $C(S) = \{\lambda\}$,
- $F(S) = \{\lambda\}$,
- $T(S) = \{a, b\}$.

However, when $k = 2$*, we get the more interesting machine* Z_2*:*

- $\Sigma = \{a, b\}$,
- $I(S) = \{a\}$,
- $C(S) = \{a\}$,
- $F(S) = \{a, b\}$,
- $T(S) = \{aa, ab, ba, bb\}$.

The next step is to construct a deterministic finite automaton from a k-testable machine that recognise the same language.

Algorithm 2 Building a DFA from a $k - \mathcal{TSS}$

Input: A $k - \mathcal{TSS}$ machine $\langle \Sigma, I, F, T, C \rangle$
Output: A DFA $\langle \Sigma, Q, q_\lambda, F_\mathcal{A}, \delta \rangle$

$Q := \emptyset$
$F_\mathcal{A} := \emptyset$
if k = 1 **then**
  $Q := \{q_\lambda\}$
  $F_\mathcal{A} := \{q_\lambda\}$
  for $a \in \Sigma$ **do**
    $\delta(q_\lambda, a) = q_\lambda$
  end for
else
  for $pu \in I \cup C, p, u \in \Sigma^\star$ **do**
    $Q := Q \cup \{q_u\}$
  end for
  for $au \in T, a \in \Sigma, u \in \Sigma^\star$ **do**
    $Q := Q \cup \{q_u\}$
  end for
  for $ua \in T, a \in \Sigma, u \in \Sigma^\star$ **do**
    $Q := Q \cup \{q_u\}$
  end for
  for $pau \in I \cup C, a \in \Sigma, p, u \in \Sigma^\star$ **do**
    $\delta(q_p, a) = q_{pa}$
  end for
  for $aub \in T, a, b \in \Sigma, u \in \Sigma^\star$ **do**
    $\delta(q_{au}, b) = q_{ub}$
  end for
  for $u \in F \cup C$ **do**
    $F_\mathcal{A} := F_\mathcal{A} \cup \{q_u\}$
  end for
end if
return $\langle \Sigma, Q, q_\lambda, F_\mathcal{A}, \delta \rangle$

Given a $k - \mathcal{TSS}$ machine, Algorithm 2 converts all the prefixes of strings in set I and C into states in Q, and for every string pau, we have a transition from p to pa to link two states q_p and q_{pa}. Similarly, all prefixes and suffixes of strings in T are converted into states of the automaton and for every string aub, we get a transition from au to ub to link the states q_{au} and q_{ub}. The final states in $F_\mathcal{A}$ corresponds to the strings in set F. Note that the sizes of the state space of the automaton is at most $k - 1$.

Example 4. *Let $S = \{a, c, abba, abbbba\}$ be our learning sample and suppose we choose $k = 3$. According to Algorithm 1, we get the 3-testable machine:*

- $\Sigma = \{a, b, c\}$,
- $I(S) = \{ab\}$,
- $F(S) = \{ba\}$,
- $T(S) = \{abb, bba, bbb\}$,
- $C(S) = \{a, c\}$.

Using Algorithm 2, the above machine in translated into the automaton of Figure 3. Hence the 3-testable language recognized by the above machine is $a + c + abb^*a$. Note that the sample S is included in the language recognized by the automaton.

Proposition 1. *Given a sample S, for a fixed $k > 0$, the language recognized by the k-testable machine generated by Algorithm 1 is the same as the language recognized by the deterministic automaton constructed by Algorithm 2. Furthermore, this language includes S.*

To learn a distribution on Σ^*, we will consider samples

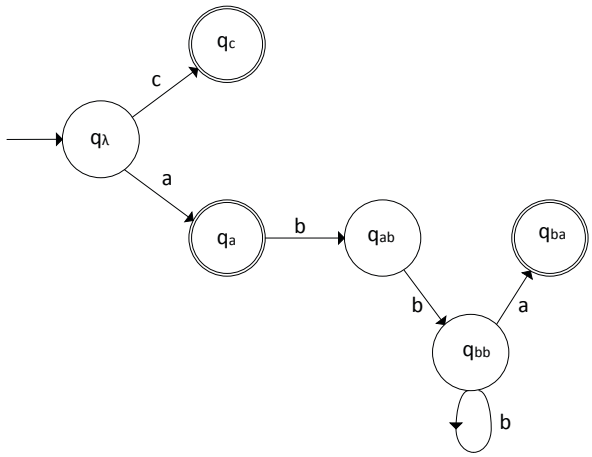

Fig. 3. A DFA generated from the sample $S = \{a, c, abba, abbbba\}$.

with frequencies. A sample with frequency is a pair (S, Fr), where S is a finite subset of Σ^* and Fr is a function associating to each element in S a positive number, denoting its frequency. The idea is that we observe a finite set of strings and their frequencies which are used to learn a distribution on Σ^* consistent with the sample S and close enough to the associated frequency.

Algorithm 3 Building a DFFA from a sample

Input: A sample S with frequency Fr
Output: A DFFA $\langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr} \rangle$

Build a $A_{k-\mathcal{TSS}}$ machine $\langle \Sigma, Q, q_\lambda, F_\mathcal{A}, \delta \rangle$ from a sample according to Algorithm 1
Build a DFA $\langle \Sigma, Q, q_\lambda, F_\mathcal{A}, \delta \rangle$ from a $A_{k-\mathcal{TSS}}$ machine according to Algorithm 2
$\mathbb{I}_{fr}(q_\lambda) := \sum_{x \in S} Fr(x)$
for $\forall q_p \in Q, \forall a \in \Sigma, u, p, x \in \Sigma^\star$ **do**
  if $|p| < k - 1$ **then**
    $\delta_{fr}(q_p, a, q_{pa}) = \sum_{pax \in S} Fr(pax)$
  else
    $\delta_{fr}(q_p, a, q_{pa}) = \sum_{upax \in S} Fr(upax)$
  end if
end for
for $\forall q \in F_\mathcal{A}, x \in S$ **do**
  $\mathbb{F}_{fr}(q) = \sum_{x, \delta(q_\lambda, x) = q} Fr(x)$
end for
return $\langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr} \rangle$

In Algorithm 3, q_λ is the initial state, which means the DFFA starts from q_λ, so the frequency of the initial state is the sum of the frequencies of all strings in the sample. For any other states $q_p \in Q$, if the length of string p is shorter than k, the frequency of this transition is equal to the sum of the frequencies of all the strings that start from q_p in the sample. Otherwise, the frequency is equal to the sum of the frequencies of all the strings which contain the substring p. For each state q in finial set, the frequency of transition is equal to the sum of the frequencies of strings which ended in state q.

Proposition 2. *The deterministic frequency finite automaton resulting from Algorithm 3 is consistent.*

Proof. For the state q_λ, the left hand side of Equation (6) is the sum of the frequencies of all strings in the sample. The right hand side, in turn, is the sum of the frequencies of strings with λ as prefix. Since all the strings have λ as prefix, the equation holds.

For any other state q_p, $\mathbb{I}_{fr}(q_p) = 0$, according to the Algorithm 3,and if the length of p is shorter than $k-1$ then $\sum_{q'\in Q,a\in\Sigma}\delta_{fr}(q',a,q_p) = \sum_{px\in S} Fr(px)$, $\mathbb{F}_{fr}(q_p) = \sum_{x\in S,\delta(q_\lambda,x)=q_p} Fr(x)$ and $\sum_{q'\in Q,a\in\Sigma}\delta_{fr}(q_p,a,q') = \sum_{pax\in S} Fr(pax)$. That is to say, the left of the equation equals the sum of the frequencies of all strings in S which have p has prefix, while the right of equation equals the sum of the frequency of the string p and of the frequencies of all strings in S having pa as prefix. Therefore, the equation is true. Similarly, if the length of p is equal to $k-1$, $\sum_{q'\in Q,a\in\Sigma}\delta_{fr}(q',a,q_p) = \sum_{upx\in S} Fr(upx)$, $\mathbb{F}_{fr}(q_p) = \sum_{x\in S,\delta(q_\lambda,x)=q_p} Fr(x)$ and $\sum_{q'\in Q,a\in\Sigma}\delta_{fr}(q_p,a,q') = \sum_{upax\in S} Fr(upax)$.

So the left of equation equals the sum of the frequency of all strings in S starting with upa and the right of equation equals the sum of the frequencies of all strings ending with p and the frequencies of all strings starting with $upab$. So they are equals, from which it follows that the DFFA we get from Algorithm 3 is a consistent DFFA. □

If the DFFA is consistent, then we can construct a DPFA. We use the following Algorithm 4 to translate frequencies into probabilities, in the most expected way.

Algorithm 4 Constructing a DPFA from a DFFA

Input: A consistent DFFA $A = \langle \Sigma, Q, \mathbb{I}_{fr}, \mathbb{F}_{fr}, \delta_{fr}\rangle$
Output: A DPFA $B = \langle \Sigma, Q, q_\lambda, \mathbb{F}_p, \delta_p\rangle$

1: **for** $q \in Q$ **do**
2: FREQ[q] := $\mathbb{F}_{fr}(q)$
3: **for** $a \in \Sigma$, $q' \in Q$ **do**
4: FREQ[q] := FREQ[q] + $\delta_{fr}(q,a,q')$
5: **if** FREQ[q] > 0 **then**
6: $\mathbb{F}_p(q) := \frac{\mathbb{F}_{fr}(q)}{FREQ[q]}$
7: **else**
8: $\mathbb{F}_p(q) := 0$
9: **end if**
10: **end for**
11: **for** $a \in \Sigma$ **do**
12: **if** FREQ[q] > 0 **then**
13: $\delta_p(q,a,q') := \frac{\delta_{fr}(q,a,q')}{FREQ[q]}$
14: **else**
15: $\delta_p(q,a,q') := 0$
16: **end if**
17: **end for**
18: **end for**
19: **return** $\langle \Sigma, Q, q_\lambda, \mathbb{F}_p, \delta_p\rangle$

In Algorithm 4, for every state q in Q, we sum up the frequencies of all transitions leaving the state q and entering it. We denote by $FREQ[q]$ the result of this summation. It follows that the positive probability assigned to each state is $\frac{\mathbb{F}_{fr}(q)}{FREQ[q]}$. and, using consistency, the probability associated to each transition from q to q' labeled by a is $\frac{\delta_{fr}(q,a,q')}{FREQ[q]}$. It is important to realize that the loop of line 3 can be optimized if we remember the state q' such that $\delta_{fr}(q,a,q') > 0$, because all other states do not add anything to the frequency. Note that all the transitions in DFFA are the same as ones in DPFA.

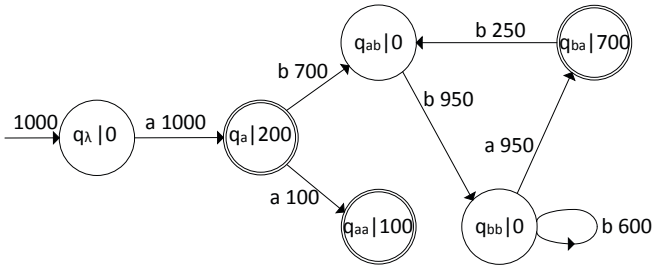

Fig. 4. A DFFA from sample $S = \{a, aa, abba, abbbba, abbabba\}$ and a frequency Fr such that $Fr(a) = 200$, $Fr(aa) = 100$, $Fr(abba) = 150$, $Fr(abbbba) = 300$ and $Fr(abbabba) = 250$.

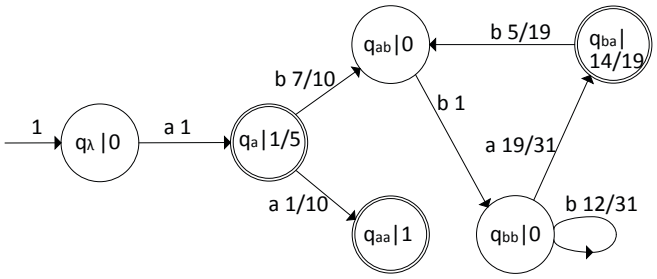

Fig. 5. A DPFA from sample $S = \{a, aa, abba, abbbba, abbabba\}$ and a frequency Fr such that $Fr(a) = 200$, $Fr(aa) = 100$, $Fr(abba) = 150$, $Fr(abbbba) = 300$ and $Fr(abbabba) = 250$.

In order to explain how our algorithm works, we need a sample with frequency. That is to say, a multiset that every event has the times of occurrence. Now we shall use one sample of 1000 strings:

Example 5. *Let $S = \{a, aa, abba, abbbba, abbabba\}$ and a frequency Fr such that $Fr(a) = 200$, $Fr(aa) = 100$, $Fr(abba) = 150$, $Fr(abbbba) = 300$ and $Fr(abbabba) = 250$ be our learning sample and suppose, we choose $k = 3$. According to Algorithm 1, we can get:*

- $\Sigma = \{a, b\}$
- $I(S) = \{ab, aa\}$
- $F(S) = \{aa, ba\}$
- $T(S) = \{abb, bab, bba, bbb\}$
- $C(S) = \{a, aa\}$

Using Algorithm 2, Algorithm 3 and Algorithm 4, then we can build the corresponding DFA, DFFA and DPFA from the above sample S. These machines are shown in Fig. 4 and Fig. 5.

IV. Analysis of the algorithm

A. Comparison with ALERGIA algorithm

When considering the language without probabilities, the language learned by our algorithm is the smallest k-testable language including the sample [17]. This property is called consistency.

Proposition 3. *Given a sample with frequency S, and $k > 0$, then $\mathcal{L}_k(A)$ is smallest language including $\bar{S} = \{\omega | (\omega, n) \in S\}$.*

Proof. If there were a smaller one then some prefix, suffix or substring should be absent. □

Note that if the length of a string is shorter than k, then the probability of the string in sample is the same as the probability of string we get from the learning automaton. Otherwise, if the length of string is equal to or greater than k, the probabilities may be different. In total, even if we normalize the probabilities of all strings in the sample from the learning automaton, we do not necessarily get the same probabilities as those in the sample. However, if k is large enough, which means k is greater than the longest string in the sample, the probabilities are the same.

If the target language is k-testable then in the large sample limit, our learning algorithm with k as window parameter will learn exactly the target language. This is an easy consequence of the consistency property above. However notice that for more general regular language (not necessarily k-testable), our learning algorithm will never learn the target language exactly, but it will learn the smallest k-testable language instead. This convergency result is different from probably approximately correct (PAC) learning [21], which is typically stronger.

Next we consider other methods for learning probabilistic languages. The most famous are those based on the ALERGIA algorithm [5]. The ALERGIA algorithm starts with the construction of a frequency prefix tree acceptor (FPTA), as a first basic approximation of the model of the language to be learnt. The learning algorithm then approximates the generating model by merging together states in the FPTA which are to be considered language equivalent.

Below, we present an example of a k-testable language for which our algorithm gets a better result when compared to the ALERGIA algorithm.

Let the target language be the one generated by the automaton in Fig. 6, and let $S = \{a, ab, abab, aaab, aabab\}$ and a frequency Fr such that $Fr(a) = 522$, $Fr(ab) = 174$, $Fr(abab) = 86$, $Fr(aaab) = 109$ and $Fr(aabab) = 109$ be the sample.

The automata generated by our algorithm and ALERGIA are shown in Figure 7 and Figure 8, respectively. The structure of target automaton and the automaton from our algorithm are the same, because the sample S is a characteristic set for the target language (i.e. there are enough strings in S to cover all transitions of the target automaton). This is not the case for the automaton learnt by ALERGIA. In fact, the string $aaba$ is not in the target language (L_T), but it is in the learnt language (L_A). Conversely, the string aab in the L_T, but not in L_A. Note that, if string aaa with some frequency would be included in the sample S, then the result from ALERGIA would be consistent with the target. In general, ALERGIA learn any regular language in the large sample limit [22].

Furthermore, ALERGIA complexity depends on merging of equivalent states. This problem is known to have polynomial

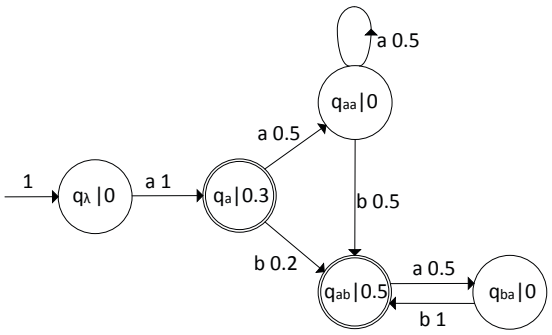

Fig. 6. The target DPFA which can recognize a $3-$testable language.

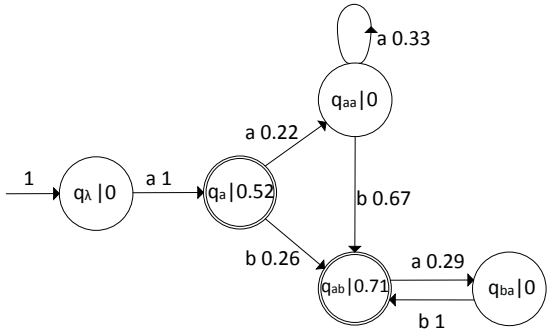

Fig. 7. The DPFA constructed by our algorithm from the sample $S = \{a, ab, abab, aaab, aabab\}$ and a frequency Fr such that $Fr(a) = 522$, $Fr(ab) = 174$, $Fr(abab) = 86$, $Fr(aaab) = 109$ and $Fr(aabab) = 109$.

time complexity [23]. Our algorithm, instead, is linear in the number of states and the size of sample. In general, for a fixed k, the number of states of our generated automaton is $\frac{1-|\Sigma|^k}{1-|\Sigma|}$, where $|\Sigma|$ is the length of Σ.

We conclude by showing a more complex example. Consider the probabilistic automaton shown in as Fig. 9 which accepts strings in $(aa + aab)^\star + b$ with probability strictly greater than 0. Let $S = \{b, aab, aabb, aaaab, aabaab, aabaabb\}$ be a sample with frequency Fr given by $Fr(b) = 188$, $Fr(aab) = 188$, $Fr(aabb) = 471$, $Fr(aaaab) = 94$, $Fr(aabaab) = 47$, $Fr(aabaabb) = 12$. Figures 10 and 11 show the automata learned by our algorithm with $k = 3$ and $k = 4$, respectively.

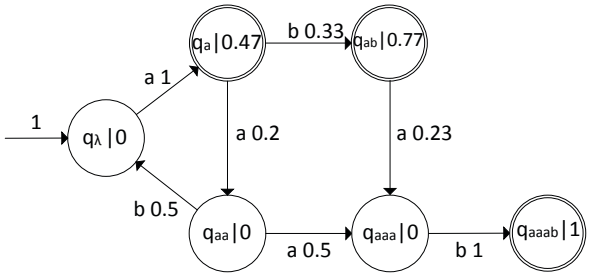

Fig. 8. The DPFA constructed by ALERGIA algorithm from the sample $S = \{a, ab, abab, aaab, aabab\}$ and a frequency Fr such that $Fr(a) = 522$, $Fr(ab) = 174$, $Fr(abab) = 86$, $Fr(aaab) = 109$ and $Fr(aabab) = 109$.

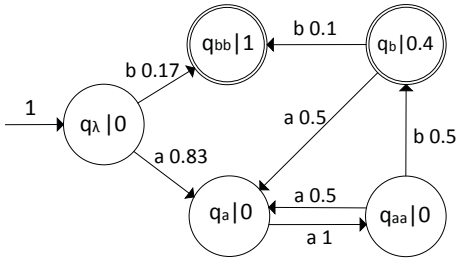

Fig. 9. The target DPFA.

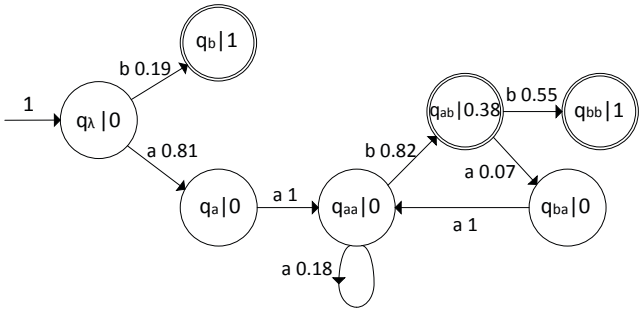

Fig. 10. The DPFA constructed by our algorithm from the sample $S = \{b, aab, aabb, aaaab, aabaab, aabaabb\}$ and a frequency Fr such that $Fr(b) = 188$, $Fr(aab) = 188$, $Fr(aabb) = 471$, $Fr(aaaab) = 94$, $Fr(aabaab) = 47$, $Fr(aabaabb) = 12$ with $k = 3$.

Fig. 12 shows the automaton learned by the ALERGIA algorithm. All three of them accept the strings of the sample with the probability greater than 0. In Table I, we can see that the string $aabaabaab$ is in the target language, but it cannot accepted by the automaton learned by ALERGIA. It is accepted by both automata learned by our algorithm. On the other hand, the string $aaab$ is not in the target language, but all three automata accept it. Even if the automaton with k is 4 accepted with the very low probability.

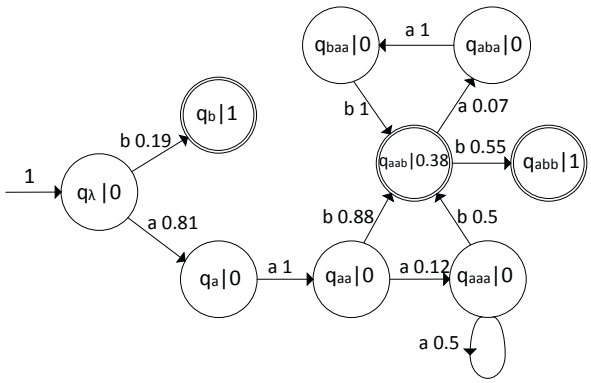

Fig. 11. The DPFA constructed by our algorithm from the sample $S = \{b, aab, aabb, aaaab, aabaab, aabaabb\}$ and a frequency Fr such that $Fr(b) = 188$, $Fr(aab) = 188$, $Fr(aabb) = 471$, $Fr(aaaab) = 94$, $Fr(aabaab) = 47$, $Fr(aabaabb) = 12$ with $k = 4$.

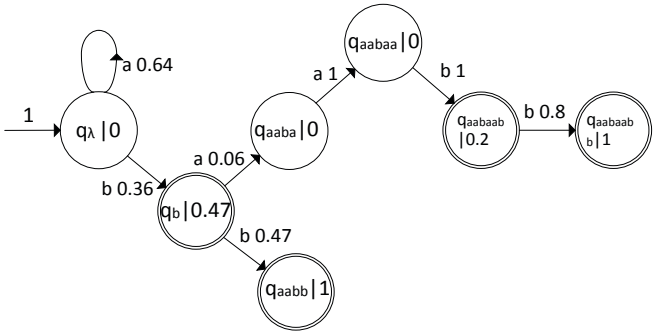

Fig. 12. The DPFA constructed by ALERGIA algorithm from the sample $S = \{b, aab, aabb, aaaab, aabaab, aabaabb\}$ and a frequency Fr such that $Fr(b) = 188$, $Fr(aab) = 188$, $Fr(aabb) = 471$, $Fr(aaaab) = 94$, $Fr(aabaab) = 47$, $Fr(aabaabb) = 12$.

strings	Target	$k = 3$	$k = 4$	ALERGIA
b	0.17	0.19	0.19	0.169
aab	0.166	0.249	0.271	0.069
aabb	0.042	0.365	0.392	0.147
aabaabb	0.01	0.021	0.027	0.007
aabaabaab	0.01	0.001	0.001	0
aaab	0	0.045	0.018	0.044
aaaab	0.083	0.008	0.009	0.044

TABLE I
THE COMPARISON TABLE OF OUTPUT PROBABILITIES

B. Comparison with García's algorithm

Now we compare our algorithm with García's algorithm in noise setting. The target automata is depicted as Fig. 13. We can get string a with probability 0, string aa with probability 0.5. Suppose there is a sample $S = \{b, aa, aaa, aaaa\}$ and a frequency Fr such that $Fr(b) = 1$, $Fr(aa) = 100$, $Fr(aaa) = 100$ and $Fr(aaaa) = 100$. The frequency of the string b is 1, apparently b is the "noise" here.

We choose the $k = 3$, according to our algorithm, we can get a DPFA as Fig. 14. Then we can get the string b with probability 0.003, the string aa with probability 0.499 and the string aaa with probability 0.249. Under these circumstances, we can tell the difference between the noise and other strings from probabilities. Then, we can recognize language by DPFA using a cut-point. Let $c \in \mathbb{R}$ and $0 \leq c < 1$, the language accepted by a DPFA A is defined as follows:

$$L_c(A) = \{\omega | p_A(\omega) > c\}. \quad (7)$$

The language $L_c(A)$ is then said to be accepted by A with respect to the cut-point c. If we set the cut-point c larger than the probability of getting string b, then we can avoid to accept string b.

García's algorithm is a $k-$length learner, for some $k \in \mathbb{N}$, learns languages defined entirely by good substrings of length n. It simply memorised all $k-$length it encounters, and accepts or generates strings that contain only $k-$length from the set it memorised. According to the García's algorithm, we can get a DFA as depicted in Fig. 15. This DFA will accept string b, which is supposed not to appear in the target

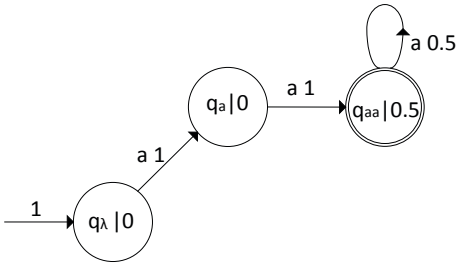

Fig. 13. The target DPFA.

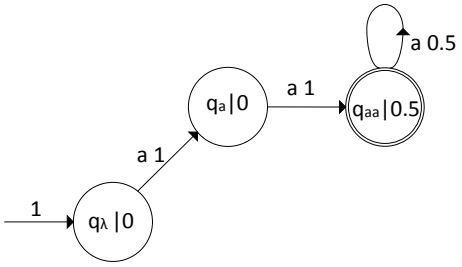

Fig. 14. A DPFA learnt from sample $S = \{b, aa, aaa, aaaa\}$ and a frequency Fr such that $Fr(b) = 1$, $Fr(aa) = 100$, $Fr(aaa) = 100$ and $Fr(aaaa) = 100$. The frequency of the string b is 1 according to our algorithm.

language. Therefore, the string b appears just once and the DFA is completely wrong. That is to say, it will introduce the wrong state in the automaton. Apparently, our algorithm is less sensible to the noise.

V. Conclusions

It is important to realize how to choose k is a crucial issue. If we choose a small k, the automaton will accept everything, running the risk of over-generalisation. If we choose k too large, one may not generalise enough. For example, if k is larger than the largest string in sample S, we just learn exactly the sample S itself, and nothing more.

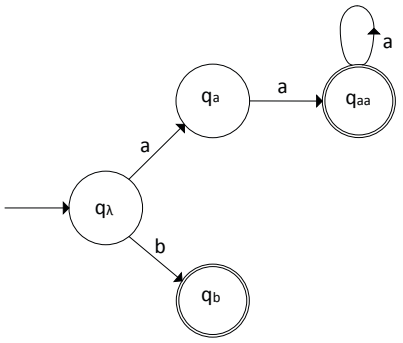

Fig. 15. A DFA learnt from sample $S = \{b, aa, aaa, aaaa\}$ and a frequency Fr such that $Fr(b) = 1$, $Fr(aa) = 100$, $Fr(aaa) = 100$ and $Fr(aaaa) = 100$. The frequency of the string b is 1 according to the García's algorithm.

Next we want to do some experiments to see how our algorithm performs in practical situations, using large text samples with frequency, and compare our results with those obtained by Alergia

VI. Acknowledgements

The research of the first author is supported by the China Scholarship Council.

References

[1] R. B. Lyngso, C. N. Pedersen, and H. Nielsen, "Metrics and similarity measures for hidden markov models," in *Proceedings of ISMB*, vol. 99, 1999, pp. 178–186.
[2] A. Nadas, "Estimation of probabilities in the language model of the ibm speech recognition system," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 4, pp. 859–861, 1984.
[3] E. M. Gold, "Language identification in the limit," *Information and control*, vol. 10, no. 5, pp. 447–474, 1967.
[4] D. Angluin, *Identifying languages from stochastic examples*. Yale University. Department of Computer Science, 1988.
[5] R. C. Carrasco and J. Oncina, "Learning stochastic regular grammars by means of a state merging method," in *International Colloquium on Grammatical Inference*. Springer, 1994, pp. 139–152.
[6] ——, "Learning deterministic regular grammars from stochastic samples in polynomial time," *RAIRO-Theoretical Informatics and Applications*, vol. 33, no. 1, pp. 1–19, 1999.
[7] D. Ron, Y. Singer, and N. Tishby, "On the learnability and usage of acyclic probabilistic finite automata," *Journal of Computer and System Sciences*, vol. 56, no. 2, pp. 133–152, 1998.
[8] D. Angluin and C. H. Smith, "Inductive inference: Theory and methods," *ACM Computing Surveys (CSUR)*, vol. 15, no. 3, pp. 237–269, 1983.
[9] P. Garcia, E. Vidal, and J. Oncina, "Learning locally testable languages in the strict sense." in *ALT*, 1990, pp. 325–338.
[10] F. Coste, "Learning the language of biological sequences," in *Topics in grammatical inference*. Springer, 2016, pp. 215–247.
[11] J. Rogers and G. K. Pullum, "Aural pattern recognition experiments and the subregular hierarchy," *Journal of Logic, Language and Information*, vol. 20, no. 3, pp. 329–342, 2011.
[12] F. Tantini, A. Terlutte, and F. Torre, "Sequences classification by least general generalisations," in *International Colloquium on Grammatical Inference*. Springer, 2010, pp. 189–202.
[13] N. Abramson, "Information theory and coding," 1963.
[14] P. Saranya and L. Thara, "Recongnition of complex human activities using visual and sequence pattern mining," *International Journal of Research in Computer Applications and Robotics*, vol. 3, no. 2, pp. 22–29, 2015.
[15] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
[16] ——, "Prediction and entropy of printed english," *Bell system technical journal*, vol. 30, no. 1, pp. 50–64, 1951.
[17] C. De la Higuera, *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
[18] J. C. Martin, *Introduction to Languages and the Theory of Computation*. McGraw-Hill NY, 1991, vol. 4.
[19] P. Garcia and E. Vidal, "Inference of k-testable languages in the strict sense and application to syntactic pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 9, pp. 920–925, 1990.
[20] M. O. Rabin, "Probabilistic automata," *Information and control*, vol. 6, no. 3, pp. 230–245, 1963.
[21] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
[22] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning deterministic probabilistic automata from a model checking perspective," *Machine Learning*, vol. 105, no. 2, pp. 255–299, 2016.
[23] F. Thollard, P. Dupont, C. de la Higuera *et al.*, "Probabilistic dfa inference using kullback-leibler divergence and minimality," in *ICML*, 2000, pp. 975–982.