Automata learning: from probabilistic to quantum

Chu, W.

Chapter 3

Passive Learning Probabilistic Automata

Automata learning techniques aim to automatically infer automata models from observations. We can distinguish two different types of algorithms: active and passive learning. Active learning involves interactions between the learner and the system being learned. Typically this is done via queries to gather information that helps in learning an automaton. Passive learning, in contrast, refers to the process of learning an automaton by observing its behavior passively. In this setting, the learner is not allowed to interact with the system but uses a finite subset available of the observable behavior of the system to infer the underlying automaton. Passive learning is particularly useful when the system under study is inaccessible or difficult to interact with actively. It has applications in various domains, including protocol analysis, language recognition, and software verification.

In this chapter, we focus on passive learning probabilistic automata. The goal is to construct a PFA as a representation of an unknown regular distribution D over Σ^* by observing only a finite number of strings independently drawn from Σ^* according to D. A sample considering only strings with a strictly positive probability (or above a fixed cut point) is called positive. Selecting the string according to a uniform distribution over Σ^* allows us to consider strings with a null probability according to D. Every such finite set is called a negative sample, and the strings can be considered counterexamples, as they do not belong to the language of the underlying NFA.

Regular languages (and distributions) cannot be learned from positive samples only, even if we let the size of the sample increase [46]. The way out is to look for subclasses of regular languages or to have some extra information available besides the positive sample [3]. Here we look at both cases and present a novel technique for passive learning PFAs based on a finite set of strings, each associated with a frequency, and a parameter k determining the length of the history or context that one remembers. Specifically, we look at k-testable machines that can remember only the last k symbols of the input sequence it has encountered.

This restriction makes it possible to infer (an approximation of) a subclass of DPFA from a finite sample of its distribution and a fixed parameter k. The larger the sample, the closer is the learned DPFA to the original system, for a correct guess of the parameter k. In the next chapter, we will abandon the guess for the parameter k and use positive and negative samples to learn an even larger class of PFAs.

We compare our algorithm with ALERGIA [24] another popular method used for learning DPFAs that focuses solely on positive samples. ALERGIA is an incremental algorithm that starts from a tree-like automaton accepting exactly the sample and iteratively merges states to create a more general and compact model. The probabilities of the transitions are updated based on the information in the positive sample. Different than our algorithms, ALERGIA uses statistical tests to decide which states to merge.

3.1 From k-testable machines to deterministic automata

Our learning approach is based on a probabilistic extension of learning k-testable languages, a proper subclass of the regular languages. Intuitively, a k-testable language is a language that can be recognized or generated by a machine, which has limited memory allowing it to observe only no more than k consecutive symbols, either as a prefix, suffix, or substring of the input sequence.

To fix the notation, for any string u and any language L, we define $PREF(u) = \{v \in \Sigma^* | \exists w \in \Sigma^*, vw = u\}$ to be the set of prefixes of u and $PREF(L) = \bigcup_{u \in L} PREF(u)$ to be the prefix set of L, and the suffix set of L is denoted by $SUFF(L) = \{v \in \Sigma^* | \exists u \text{ such that } uv \in L\}$. To characterize formally the class of k-testable languages, [44] introduced a special type of machine:

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

- *Σ is a finite set, the alphabet,*
- *$I, F \subseteq \Sigma^{k-1}$ are the sets of prefixes of length $k-1$ and of suffixes (or finals) of length $k-1$),*
- *$C \subseteq \Sigma^{<k}$ is the set short strings, and*
- *$T \subseteq \Sigma^k$ is the set of 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 lengths are greater than or equal to k. Otherwise, they must belong to set C. Thus, there are two types of strings in $L(Z_k)$: strings of length less than k, that are defined by C, and strings of length greater than or equal to k, which 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 either $\emptyset$ or Σ^*. This is because I, F and C are subsets of $\{\lambda\}$ and T equals Σ. See the Example 3 for an illustration.

A k-testable language is a regular language whose memory (i.e., the minimal number of states needed by a DFA to recognize it) can be bounded a priori. This follows from Definition 5 because the size of the window of visible symbols of a k-testable language is exactly k. The following symbol in a string depends on the previous $k-1$ characters. In other words, k-testable languages are causal.

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

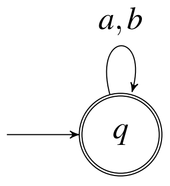

Fig. 3.1 A 1-testable language.

Example 3. *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 3.1 which accepts all strings.*

Example 4. *The DFA from Figure 3.2 recognises the language* $\{ba,bb\}\{b\}^*$. *This language is* 3*-testable language because it can be recognized by the* 3*-testable machine* $Z_3 = \langle \{a,b\}, I = \{bb,ba\}, F = \{ab,bb,ba\}, T = \{bbb,bab,abb\}, C = \{ba,bb\} \rangle$. *It is, however, not a* 2*-testable language, because, with a window of size two, any* 2*-testable machine would accept the set of strings* $\{b\}^*$.

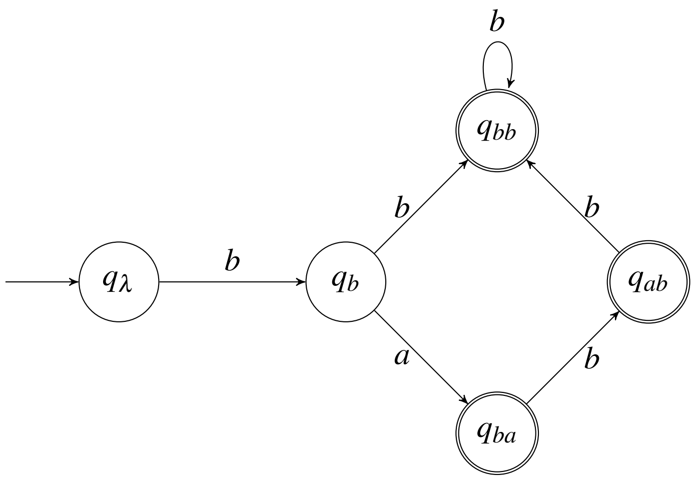

Fig. 3.2 A deterministic automaton for the language $\{ba, bb\}\{b\}^*$.

To construct a DFA from a k-testable machine Z_k, each state in the DFA represents a unique history of at most k-symbols observable in Z_k, and transitions between states are determined based on the observed input symbols and the previous history, resulting in a DFA that captures exactly the deterministic behavior of the k-testable machine.

Given a k-testable machine, Algorithm 4 converts all the prefix x of strings in I and C into states $q_x \in Q$ of the DFA. For every string pau, there is an a-transition 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. For every string aub, there is a b-transition to link the states q_{au} and q_{ub}, where the first symbol a is forgotten because it is outside the range k of history visibility of the automaton. The final states of the DFA correspond to the states indexed by strings in F in the k-TM. Note that the state space size of the automaton is at most $k-1$.

Example 5. *Let $\Sigma = \{a,b,c\}$ and consider the 3-testable machine $Z_3 = \langle \Sigma, I, F, T, C \rangle$ with $I = \{ab\}$, $F = \{ba\}$, $T = \{abb, bba, bbb\}$, and $C = \{a, c\}$. Using Algorithm 4, we can build the deterministic automaton of Figure 3.5. It is now easy to see that 3-testable language recognized by both machines is $\{a,c\} \cup \{ab\}\{b\}^*\{a\}$.*

The following proposition is not hard to prove. It shows that the above construction is correct in the sense that for every k-TM the constructed DFA recognizes the same language. Since we already know that not all regular languages are k-testable, it is not obvious how to find syntactic restrictions for DFAs so that they correspond exactly to k-TMs, for a given k.

Proposition 2. *For any fixed value $k > 0$ let A be DFA returned by Algorithm 6 given a k-TM Z_k as input. Then $L(A) = L(Z_k)$.*

Algorithm 4 Building a DFA from a k-testable machine

```
Input: A $k$-TM $\langle \Sigma, I, F, T, C \rangle$
Output: A DFA $\langle \Sigma, Q, I_D, F_D, \delta \rangle$
 1: $Q = \emptyset$
 2: $F_D = \emptyset$
 3: if k = 1 then
 4:     $Q = \{q_\lambda\}$
 5:     $F_D(q_\lambda) = 1$
 6:     $I_D(q_\lambda) = 1$
 7:     for $a \in \Sigma$ do
 8:         $\delta(q_\lambda, a)(q_\lambda) = 1$
 9:     end for
10: else
11:     for $pu \in I \cup C, p, u \in \Sigma^\star$ do
12:         $Q = Q \cup \{q_u\}$
13:     end for
14:     for $au \in T, a \in \Sigma, u \in \Sigma^\star$ do
15:         $Q = Q \cup \{q_u\}$
16:     end for
17:     for $ua \in T, a \in \Sigma, u \in \Sigma^\star$ do
18:         $Q = Q \cup \{q_u\}$
19:     end for
20:     for $pau \in I \cup C, a \in \Sigma, p, u \in \Sigma^\star$ do
21:         $\delta(q_p, a)(q_{pa}) = 1$
22:     end for
23:     for $aub \in T, a, b \in \Sigma, u \in \Sigma^\star$ do
24:         $\delta(q_{au}, b)(q_{ub}) = 1$
25:     end for
26:     for $u \in F \cup C$ do
27:         $F_D(q_u) = 1$
28:     end for
29: end if
30: return $\langle \Sigma, Q, I_D, F_D, \delta \rangle$
```

3.2 From frequency automata to probabilistic ones

Our goal is to learn deterministic regular distributions represented by probability automata based on the number of times that each string in a sample occurs. This frequency deals with the observed occurrences in a sample and it differs from the probability assigned to the string a PFA to be learned because the latter quantifies the likelihood of the string to be generated based on the underlying distribution represented by the PFA. For this purpose, instead of learning directly a PFA from a sample with frequency, it will be easier to first build a deterministic frequency finite automaton (DFFA) and then transform it into a PFA [34].

Definition 6. ***Deterministic frequency finite automaton*** *A deterministic frequency finite automaton (DFFA) is a tuple $F = \langle \Sigma, Q, I_v, F_v, \delta_v \rangle$ where*

- *Σ is a finite set representing the alphabet,*

- *Q is a finite set of states,*
- *$I_V : Q \to \mathbb{N}$ is the initial-state frequency map with exactly one state $q_\lambda \in Q$ for with $I_V(q_\lambda) \neq 0$,*
- *$F_V : Q \to \mathbb{N}$ is the final-state frequency map,*
- *$\delta_V : Q \times \Sigma \to \mathbb{N}^Q$ is the transition frequency function, such that*

$$|\{q' \in Q \mid \delta_V(q,a)(q') > 0\}| \leq 1$$

for all $q \in Q$ and $a \in \Sigma$.

The transition function $\delta_V(q,a)(q') = n$ can be interpreted as the number of occurrences of a needed when taking a transition from state q to state q'.

A DFFA is said to be *consistent* if the frequencies of the transitions leading to a state are related to those leaving it, that is, $\forall q \in Q$, the following equation holds:

$$I_V(q) + \sum_{q' \in Q, a \in \Sigma} \delta_V(q',a)(q) = F_V(q) + \sum_{q' \in Q, a \in \Sigma} \delta_V(q,a)(q'). \tag{3.1}$$

The above constraint ensures the preservation of flow, meaning that the number of strings entering and leaving a given state must be identical: any string that enters or starts in a state has to leave it or end there. For any state $q \in Q$, we denote by $FREQ[q]$ either the left or right-hand side of the above constraints. Figure 3.3 shows an example of a consistent DFFA.

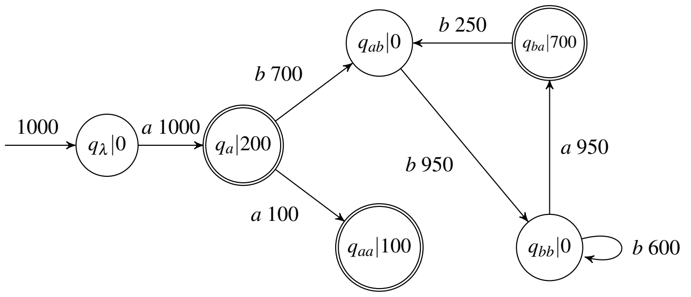

Fig. 3.3 A consistent DFFA

Consistent DFFAs are important because they can be easily translated into DPFA, as shown in Algorithm 5 where frequencies are mapped to corresponding probabilities. Algorithm 5 computes $FREQ[q]$, the frequency of each state in a DFFA, by summing up the frequencies of all transitions that leave the state and enter it. It follows that the positive probability assigned to each state is $\frac{F_V(q)}{FREQ[q]}$. Similarly, the probability associated with each transition from state q to state q' labeled by symbol a is $\frac{\delta_V(q,a)(q')}{FREQ[q]}$. It is important to realize that the loop at line 3 can be optimized if we remember the state q' such that $\delta_V(q,a)(q') > 0$ because all other states do not add anything to the frequency.

Algorithm 5 Constructing a DPFA from a consistent DFFA

```
Input: A consistent DFFA $A = \langle \Sigma, Q, I_V, F_V, \delta_V \rangle$
Output: A DPFA $B = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$
 1: for $q \in Q$ do
 2:     FREQ[q] = $F_V(q)$
 3:     for $a \in \Sigma$, $q' \in Q$ do
 4:         FREQ[q] = FREQ[q] + $\delta_V(q,a)(q')$
 5:         if FREQ[q] > 0 then
 6:             $F_p(q) = \frac{F_V(q)}{FREQ[q]}$
 7:         else
 8:             $F_p(q) = 0$
 9:         end if
10:     end for
11:     for $a \in \Sigma$ do
12:         if FREQ[q] > 0 then
13:             $\delta_p(q,a)(q') = \frac{\delta_V(q,a)(q')}{FREQ[q]}$
14:         else
15:             $\delta_p(q,a)(q') = 0$
16:         end if
17:     end for
18: end for
19: return $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$
```

Note that the structure of the DFFA (states, strictly positive transition, initial state, accepting ones) are the same as the ones in the resulting DPFA, which is thus deterministic. Because of consistency, it is not hard to see that the automata resulting from Algorithm 5 are indeed probabilistic. Given the consistent DFFA in Figure 3.3, we have, for example, that $FREQ[q_{bb}] = 600 + 950 = 1550$ and thus $\delta_p(q_{bb}, b)(q_{bb}) = 600/1550 = 12/31$ and $\delta_p(q_{bb}, a)(q_{bb}) = 950/1550 = 19/31$. Since $F_V(q+bb) = 0$ so is F_p. The full DPFA resulting from this DFFA is given in Figure 3.4.

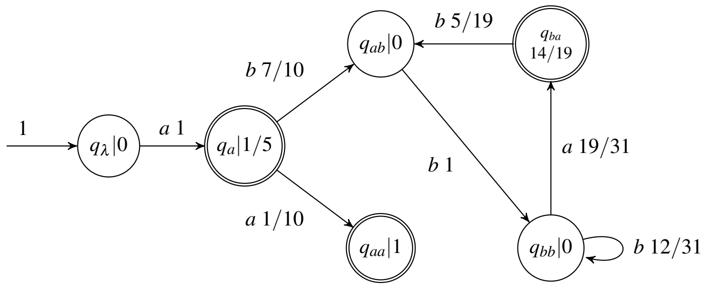

Fig. 3.4 The DPFA resulting from the DFFA in Figure 3.3

3.3 Learning DPFAs using k-testable machines

We can finally present our algorithm for learning a DPFA from a sample (S, Fr), where S is a finite subset of strings in Σ^* and $Fr : S \to \mathbb{N}$ is a function associating to each element in S a strictly positive number representing its frequency. To find the distribution on Σ^* of which we only have a sample (S, Fr), we proceed as follows. For a given parameter k, first, we learn a k-testable machine from the set S. Then we build a DFA that recognizes the same language used as the structure of a DFFA with frequency built following the sample. We finally translate the DFFA into a DPFA.

Given a finite set of strings $S \subseteq \Sigma^*$ and a parameter $k \geq 1$, we construct a k-testable machine Z_k using an algorithm similar to [45]:

Algorithm 6 Building a k-testable machine from a sample

```
Input: A finite set $S \subseteq \Sigma^*$, a positive integer $k$
Output: A $k$-testable machine $Z_k$
1: $\Sigma$ is the alphabet used in S
2: I $= \Sigma^{k-1} \cap PREF(S)$
3: C $= \Sigma^{<k} \cap S$
4: F $= \Sigma^{k-1} \cap SUFF(S)$
5: T $= \Sigma^k \cap \{v : uvw \in S, u, v, w \in \Sigma^\star\}$
6: return $\langle \Sigma, I, F, T, C \rangle$
```

Learning k-testable languages involves identifying all the prefixes, substrings, and suffixes of length $k-1$ that appear in the sample S. In fact, in Algorithm 6 the strings in the sample S that are shorter than k define the set C of short strings of Z_k. For strings that are longer than or equal to k, we extract all the prefixes of length exactly $k-1$ and place them in set I. Similarly, we extract all the suffixes of length $k-1$ and place them in set F. Additionally, we extract all substrings of length k and place them in T.

Example 6. *Let us consider the set of strings $S = \{a, aa, abba, ababa, ababab\}$ over the alphabet $\Sigma = \{a, b\}$. By applying Algorithm 6, with $k = 1$, we get a machine $Z_1 = \langle \Sigma, I = \{\lambda\}, C = \{\lambda\}, F = \{\lambda\}, T = \{a, b\} \rangle$ recognizing every string in Σ. However, for $k = 2$, we get the more interesting machine $Z_2 = \langle \Sigma, I = \{a\}, C = \{a\}, F = \{a, b\}, T = \{aa, ab, ba, bb\} \rangle$ which recognizes all strings starting with 'a'.*

By using Algorithm 4 immediately after Algorithm 6 we can construct a deterministic finite automaton from a set of strings. For example, let $S = \{a, c, abba, abbbba\}$ and assume we choose $k = 3$ as the learning parameter. Then Algorithm 6, returns the 3-testable machine Z_3 from Example 5 that we have seen corresponding to the DFA given in Figure 3.5. Note that for any k the set S is always included in the language recognized by the resulting DFA, as it should be.

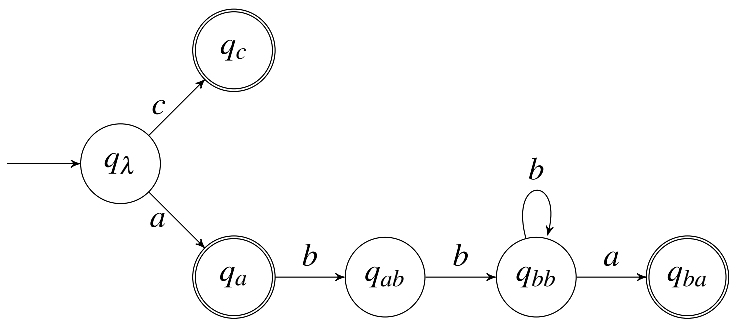

Fig. 3.5 The DFA generated from the $S = \{a, c, abba, abbbba\}$.

The next step is to add frequencies to the transitions of a DFA generated from a sample (S, Fr) for a certain learning parameter $k > 0$. The idea is to associate to the initial state q_λ the sum of the frequencies of all strings in S. For any other states $q_p \in Q$, if the length of the string p is shorter than $k-1$ then the frequency of transition labeled by a leaving q_p is equal to the sum of all frequencies of the strings in S with pa as a prefix. Otherwise, the length of p is $k-1$ and the frequency of a transition labeled by a leaving q_p is equal to the sum of all frequencies of the strings n S having pa as a substring. Finally, each state q in the final set F gets as frequency the sum of the frequencies of all strings in S that ended in that state. Algorithm 7 presents the code for generating a DFFA from a sample.

Algorithm 7 Building a DFFA from a sample (S, Fr)

```
Input: A finite sample $(S, Fr)$ and $k \geq 0$
Output: A DFFA $\langle \Sigma, Q, I_V, F_V, \delta_V \rangle$
 1: Build a $k$-testable machine $Z_k$ using Algorithm 6
 2: Build the DFA $\langle \Sigma, Q, I, F, \delta \rangle$ from $Z_k$ using Algorithm 4
 3: $I_V(q_\lambda) = \sum_{x \in S} Fr(x)$
 4: for $\forall q_p \in Q, \forall a \in \Sigma, u, p, x, p' \in \Sigma^*, \delta(q_p, a)(q_{p'}) = 1$ do
 5:     if $|p| < k-1$ then
 6:         $\delta_V(q_p, a)(q_{p'}) = \sum_{pax \in S} Fr(pax)$
 7:     else
 8:         $\delta_V(q_p, a)(q_{p'}) = \sum_{upax \in S} Fr(upax)$
 9:     end if
10: end for
11: for $\forall q \in F, x \in S$ do
12:     $F_V(q) = \sum_{x, \delta(q_\lambda, x)(q) = 1} Fr(x)$
13: end for
14: return $\langle \Sigma, Q, I_V, F_V, \delta_V \rangle$
```

Proposition 3. *For any sample (S, Fr) and $k > 0$, the deterministic frequency finite automaton resulting from Algorithm 7 is consistent.*

Proof. For the initial state q_λ, the left-hand side of Equation (3.1) is the sum of the frequencies of all strings in S. The right-hand side, in turn, is the sum of the frequencies of strings with λ as a prefix, thus all strings in S, and the equation holds. For any other state q_p, $I_v(q_p) = 0$,

Consider a state q_p with $p \neq \lambda$ and with shorter than $k-1$. Then

$$\sum_{q' \in Q, a \in \Sigma} \delta_v(q', a)(q_p) = \sum_{px \in S} Fr(px),$$
$$F_v(q_p) = \sum_{x \in S, \delta(q_\lambda, x)(q_p)=1} Fr(x),$$
$$\sum_{q' \in Q, a \in \Sigma} \delta_v(q_p, a)(q') = \sum_{pax \in S} Fr(pax).$$

That is to say, the left-hand side of the equation equals the sum of the frequencies of all strings in S which have p as a prefix, while the right-hand side of the equation equals the sum of the frequency of the string p and the frequencies of all strings in S having pa as a prefix. Therefore, the equation is true.

Similarly, if the length of p is equal to $k-1$ then $\sum_{q' \in Q, a \in \Sigma} \delta_v(q', a)(q_p) = \sum_{upx \in S} Fr(upx)$, $F_v(q_p) = \sum_{x \in S, \delta(q_\lambda, x) = q_p} Fr(x)$ and $\sum_{q' \in Q, a \in \Sigma} \delta_v(q_p, a)(q') = \sum_{upax \in S} Fr(upax)$.

In this case, the left-hand side of the equation equals the sum of the frequencies of all strings in S starting with upa, and the right-hand side equals the sum of the frequencies of all strings ending with p and the frequencies of all strings starting with $upab$. Therefore, the two sides are equal, from which it follows that the DFFA is consistent. □

The final step is to transform the learned DFFA into a PDFA using Algorithm 5. Consider for a sample (S, Fr) of 1000 strings, with $S = \{a, aa, abba, abbbba, abbabba\}$ and $Fr(a) = 200$, $Fr(aa) = 100$, $Fr(abba) = 150$, $Fr(abbbba) = 300$ and $Fr(abbabba) = 250$. If we set our learning parameter $k = 3$, Algorithm 6 returns the 3-testable machine $\langle \Sigma = \{a, b\}, I = \{ab, aa\}, F = \{aa, ba\}, T = \{abb, bab, bba, bbb\}, C = \{a, aa\} \rangle$. Using Algorithm 4, Algorithm 7, and Algorithm 5, then we can build the corresponding DFA, DFFA, and DPFA from the above sample S. These machines are shown in Figure 3.3 and Figure 3.4, respectively.

3.4 An analysis of the algorithm

The k-testable machine constructed from a finite set of string S recognizes the smallest k-testable language, including the sample[34]. If there were a smaller one, then some prefixes, suffixes, or substrings should be absent. As a consequence, if the target language is itself k-testable, the larger is the size of the sample the closer is the learned language to the target one in terms of subset inclusion. In the limit thus, one can learn precisely the target language. In general, however, one does not know if the target language is k-testable. Since not all regular languages are k-testable for any k, it follows that some languages will never be learned exactly, but only over-approximating it. As such, the learning framework we described is different from the probably approximately correct (PAC) learning framework [118] which guarantees the learned language is likely correct within a certain probability and approximation bound.

Another problem comes from the selection of the parameter k. For a given set of finite string S, the smallest k-testable language L_k including it is itself included in the smallest $k+1$-testable language L_{k+1} including S, that is $S \subseteq L_{k+1} \subseteq L_k$. As an extreme case, for k larger than the largest string in sample S, the learned k-testable language is exactly S. While if we choose $k = 1$ then every string is accepted by the learned language. The choice of the learning parameter k is not always straightforward and depends on the size of the strings in S, the memory available, and the required efficiency in learning. In general, it may require some experimentation.

The way we add frequencies to the learned DFFA implies that, for strings shorter than k, the probability assigned by the resulting PDFA converges to that of the distribution to be learned when we increase the size of the sample, under the assumption that the distribution to be learned is regular and deterministic. However, for strings with a length greater than k, the probabilities may differ because the learned probabilistic automaton will contain loops that may not exist in the PDFA to be learned. Even if we normalize the probabilities of all strings in the sample given their frequencies, the learned DPFA will not necessarily get the exact probabilities as those in the sample. However, if the underlying language is k-testable and the distribution to be learned is regular and deterministic, then the learned distribution will converge to the one to be learned for larger samples.

3.4.1 Comparison with ALERGIA algorithm

Next, we compare our learning algorithm with the ALERGIA algorithm [24] that starts with constructing a frequency prefix tree acceptor (FPTA) as the first basic approximation of the model of the language to be learned and then keeps refining the current model by merging states that can be considered close enough in the distribution they recognize. ALERGIA's complexity depends on the merging of equivalent states. This problem is known to have polynomial time complexity [115]. Instead, our algorithm is linear in the number of states and the sample size. In general, for a fixed k, the number of states of our generated automaton is $\frac{1-|\Sigma|^k}{1-|\Sigma|}$.

Similar to our algorithm, given an infinite sequence of positive samples generated by a DPFA, ALERGIA will converge to the distribution generated by that PDFA [73]. However, differently than our algorithm, for ALERGIA this holds for all regular deterministic distributions instead of only those with an underlying k-testable language. Next, we present two small examples where, for a fixed sample (S, Fr) originated from a regular distribution with an underlying k-testable language for which our algorithm gives better results than ALERGIA.

Let the target distribution be the one generated by DPFA in Figure 3.6. Note that the underlying language is a 3-testable language. Consider the sample (S, Fr) with $S = \{a, ab, abab, aaab, aabab\}$ and a frequency Fr such that $Fr(a) = 522$, $Fr(ab) = 174$, $Fr(abab) = 86$, $Fr(aaab) = 109$ and $Fr(aabab) = 109$.

The automata generated by our algorithm for $k = 3$ and ALERGIA are shown in Figure 3.7 and Figure 3.8, respectively. The structure of the target DPFA and the automaton resulting 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 learned by ALERGIA. The string *aaba* is not in the underlying target language (L_T) but belongs to the underlying language learned by ALERGIA (L_A). Conversely, the string *aab* is in the L_T, but not in L_A. If we extend the sample by adding the string *aaa* to S with some positive frequency, the result from ALERGIA would be consistent with the target.

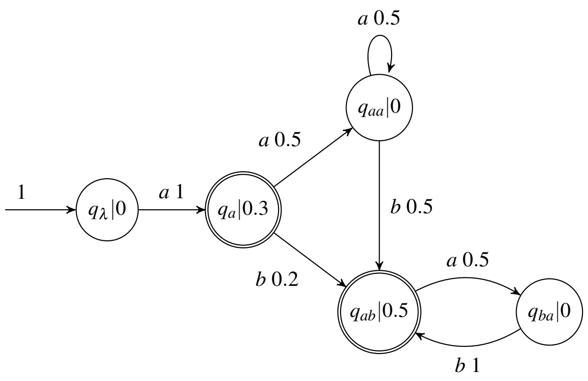

Fig. 3.6 A DPFA which can recognize a 3-testable language.

We conclude by showing a more complex example. Consider the probabilistic automaton shown in Figure 3.9 which accepts strings in $\{aa, aab\}^* \cup \{b\}$ 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 3.10 and 3.11 show the automata learned by our algorithm with

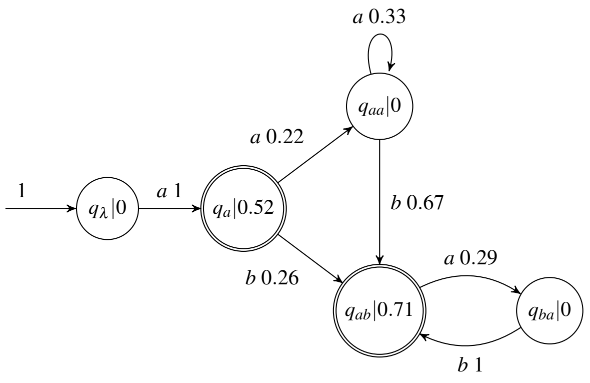

Fig. 3.7 The DPFA returned by our algorithm with $k = 3$.

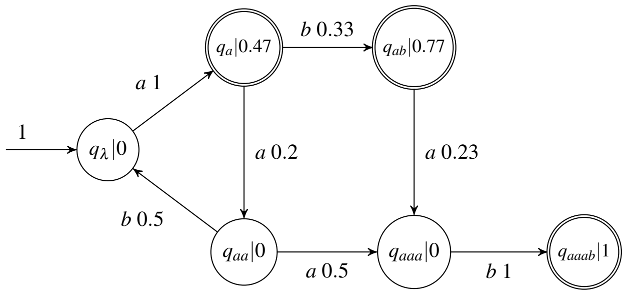

Fig. 3.8 The DPFA returned by the ALERGIA algorithm.

$k = 3$ and $k = 4$, respectively. Fig. 3.12 shows the automaton learned by the ALERGIA algorithm. All three accept the sample strings with a probability greater than 0. In Table 3.1, we can see that the string *aabaabaab* is in the target language, but the automaton learned by ALERGIA cannot accept it. Both automata learned by our algorithm accept it. On the other hand, the string *aaab* is not in the target language, but all three automata accept it even if the automaton with $k = 4$ accepts it with a very low probability.

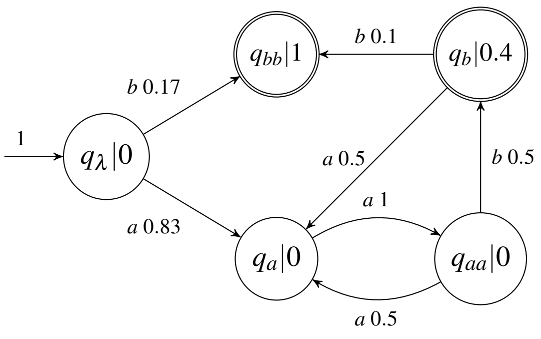

Fig. 3.9 Another target DPFA.

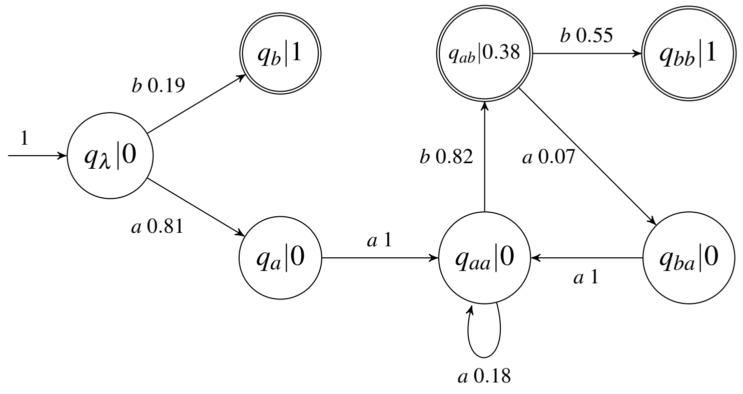

Fig. 3.10 The DPFA constructed by our algorithm for $k = 3$.

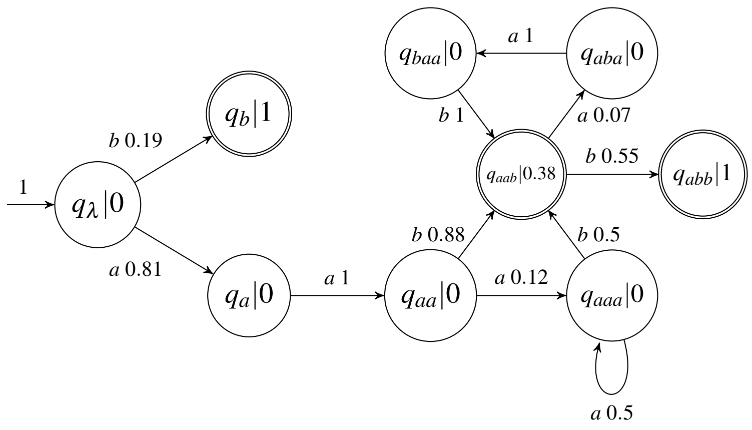

Fig. 3.11 The DPFA returned by our algorithm for $k = 4$.

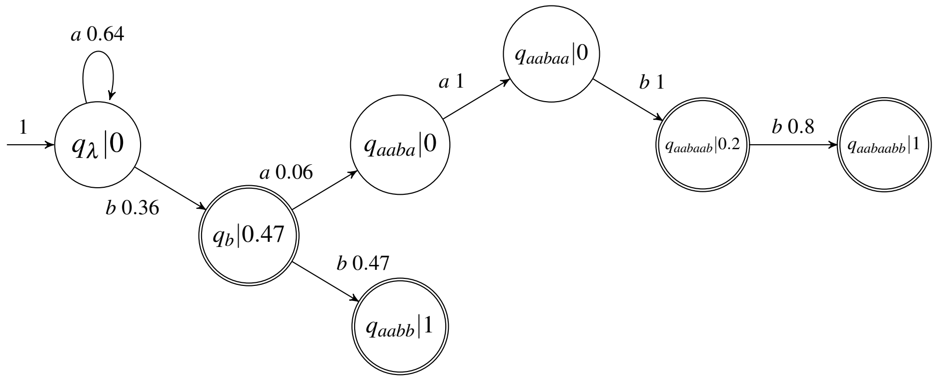

Fig. 3.12 The DPFA returned by the ALERGIA algorithm with $\alpha = 0.5$.

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 3.1 Comparing the probabilities for a few strings

The Table. 3.1 displays the probabilities for each string under different automata. The "Target" column represents the anticipated probabilities, while the "$k = 3$" and "$k = 4$" columns detail our experimentally determined probabilities. Additionally, a comparison with the "ALERGIA" column highlights noteworthy results.

3.5 Summary

In this chapter, we introduced passive learning of probabilistic finite automata based on positive samples. We presented a novel method for passive learning that is based on learning the structure of automata via k-testable machines and then adding the probabilities using frequency automata. The algorithm guarantees convergence to the distribution to be learned under the assumption it is regular, deterministic and its underlying language is k-testable, with k known.