Learning Probabilistic Subsequential Transducers from Positive Data

Hasan Ibne Akram[1] and Colin de la Higuera[2]
[1]*Technische Universität München, Munich, Germany*
[2]*University of Nantes, Nantes, France*

Keywords: Probabilistic Transducers, Weighted Transducers, Transducer Learning, Grammatical Inference, Machine Learning.

Abstract: In this paper we present a novel algorithm for learning probabilistic subsequential transducers from a randomly drawn sample. We formalize the properties of the training data that are sufficient conditions for the learning algorithm to infer the correct machine. Finally, we report experimental evidences to backup the correctness of our proposed algorithm.

1 INTRODUCTION

Probabilistic transducers or weighted transducers are widely applied in the fields of natural language processing, machine translation, bioinformatics, IT security and many other areas. The generality of probabilistic transducers allow them capturing other existing probabilistic models such as hidden Markov models (HMM) (Rabiner, 1990) or pair hidden Markov models (PHMM) (Durbin et al., 1998). Due to the widespread use and applicability of probabilistic transducers, the learnability issue of such models is an important problem.

Probabilistic subsequential transducers (PSTs) are deterministic finite state machines *w.r.t.* the input symbols, *i.e.*, there can be no two outgoing edges from a given state with same input label having different output labels and/or different destination states. Due to its deterministic property, PSTs have less expressive power than PHMM and probabilistic transducers in general. Despite of the low expressive power of PSTs, the computational complexity (*e.g.*, translation of a given string, computing the probability of a translation pair) is linear *w.r.t.* the size of the input string. Therefore, there is an interesting trade-off between PHMM or probabilistic transducers and PSTs in terms of expressive power and computational complexity.

The problem of learning PSTs in an identification of the limit model has been investigated by Akram *et al.* in (2012), where they have proposed the algorithm APTI (Algorithm for Learning Probabilistic Transducers). APTI uses a combination of state merging approach and probabilistic queries to infer the target machine.

In this paper we propose a novel algorithm (APTI2) for learning PSTs that learns from an empirical distribution of positive examples. The main idea of our proposed algorithm is inspired by previous work in subsequential transducer learning (Oncina and García, 1991; Oncina et al., 1993) and probabilistic finite automata (PFA) learning such as (Carrasco and Oncina, 1994; Thollard et al., 2000; Carrasco and Oncina, 1999).

This paper aims to utilize the lessons learnt from subsequential transducer learning and PFA learning and attempts to solve the problem of learning PSTs from positive examples.

The fact that APTI2 makes no use of an oracle, makes it usable in practice. We present an analysis of the algorithm to illustrate the theoretical boundaries. Finally, we present experimental results based on artificially generated datasets.

2 DEFINITIONS AND NOTATIONS

Let $[n]$ denote the set $\{1,\ldots,n\}$ for each $n \in \mathbb{N}$. An *alphabet* Σ is a non-empty set of symbols and the symbols are called *letters*. Σ^* is a *free-monoid* over Σ. Subsets of Σ^* are known as (*formal*) *languages* over Σ. A *string* w over Σ is a finite sequence $w = a_1 \ldots a_n$ of letters. Let $|w|$ denote the length of the string w. In this case we have $|w| = |a_1 \ldots a_n| = n$. The empty

Ibne Akram H. and de la Higuera C..
Learning Probabilistic Subsequential Transducers from Positive Data.
DOI: 10.5220/0004359904790486
In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence* (LAFLang-2013), pages 479-486
ISBN: 978-989-8565-38-9

string is denoted by ε. For every $w_1, w_2 \in \Sigma^*$, $w_1 \cdot w_2$ is the concatenation of w_1 and w_2. The concatenation of ε and a string w is given by $\varepsilon \cdot w = w$ and $w \cdot \varepsilon = w$. When decomposing a string into substrings, we will write $w = w_1, \ldots, w_n$ where $\forall i \in [n], w_i \in \Sigma^*$. If $w = w_1 w_2$ is a string, then w_1 is a *prefix*. Given a language $\mathcal{L} \subseteq \Sigma^*$, the *prefix set* of $\mathcal{L}$ is defined as $Pref(\mathcal{L}) = \{u \in \Sigma^* : \exists v \in \Sigma^*, uv \in \mathcal{L}\}$. $Pref(w)$ is defined as the set of all the substrings of w that are prefixes of w. The *longest common prefix* of $\mathcal{L}$ is denoted as $lcp(\mathcal{L})$, where $lcp(\mathcal{L}) = w \iff w \in \bigcap_{x \in \mathcal{L}} Pref(x) \wedge \forall w' \in \bigcap_{x \in \mathcal{L}} Pref(x) \Rightarrow |w'| \leq |w|$. Less formally, *lcp* is a function that returns the longest possible string which is the prefix of all the strings in a given set of strings. For example, for $\mathcal{L} = \{aabb, aab, aababa, aaa\}$ the $lcp(\mathcal{L})$ is aa.

2.1 Stochastic Transductions

In order to represent *transductions* we now use two alphabets, not necessarily distinct, Σ and Ω. We use Σ to denote the input alphabet and Ω to denote the output alphabet. For technical reasons, to denote the end of an input string we use a special symbol $\sharp$ as an end marker.

A *stochastic transduction* $\mathcal{R}$ is given by a function $Pr_{\mathcal{R}} : \Sigma^*\sharp \times \Omega^* \to \mathbb{R}_+$, such that :

$$\sum_{u \in \Sigma^*\sharp} \sum_{\mathbf{v} \in \Omega^*} Pr_{\mathcal{R}}(u, \mathbf{v}) = 1,$$

where $Pr_{\mathcal{R}}(u, \mathbf{v})$ is the joint probability of u and $\mathbf{v}$. Otherwise stated, a stochastic transduction $\mathcal{R}$ is the joint distribution over $\Sigma^*\sharp \times \Omega^*$. Let $\mathcal{L} \subset \Sigma^*\sharp$ and $\mathcal{L}' \subset \Omega^*$;

$$Pr_{\mathcal{R}}(\mathcal{L}, \mathcal{L}') = \sum_{u \in \mathcal{L}} \sum_{\mathbf{v} \in \mathcal{L}'} Pr_{\mathcal{R}}(u, \mathbf{v}).$$

Example 1. *The transduction* $\mathcal{R} : \Sigma^*\sharp \times \Omega^* \to \mathbb{R}_+$ *where* $Pr_{\mathcal{R}}(a^n\sharp, \mathbf{1}^n) = \frac{1}{2^n}, \forall n > 0$, *and* $Pr_{\mathcal{R}}(u, \mathbf{v}) = 0$ *for every other pair.*

In the sequel, we will use $\mathcal{R}$ to denote a stochastic transduction and T to denote a transducer. Note that the end marker $\sharp$ is needed for technical reasons only. The probability of generating a $\sharp$ symbol is equivalent to the stopping probability of an input string.

2.2 Probabilistic Subsequential Transducers

A transduction scheme can be modeled by transducers or probabilistic transducers. In this section, we will define *probabilistic subsequential transducers* (PST) that can be used to model a specific subclass of stochastic transductions.

Definition 1. *A* ***probabilistic subsequential transducer*** *(*PST*) defined over the probability semiring* $\mathbb{R}_+$ *is a 5-tuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E \rangle$ *where:*

- Q *is a non-empty finite set of states,*
- $q_0 \in Q$ *is the unique initial state,*
- Σ *and* Ω *are the sets of input and output alphabets,*
- $E \subseteq Q \times \Sigma \cup \{\sharp\} \times \Omega^* \times \mathbb{R}_+ \times Q$, *and given* $e = (q, a, \mathbf{v}, \alpha, q')$ *we denote:* $prev[e] = q$, $next[e] = q'$, $i[e] = a$, $o[e] = \mathbf{v}$, *and* $prob[e] = \alpha$,
- *the following conditions hold:*
  - $\forall q \in Q$, $\forall (q, a, \mathbf{v}, \alpha, q'), (q, a', \mathbf{v}', \beta, q'') \in E,\ a = a' \Rightarrow \mathbf{v} = \mathbf{v}',\ \alpha = \beta,\ q' = q''$,
  - $\forall q \in Q, \sum_{a \in \Sigma \cup \{\sharp\}, q' \in Q} Pr(q, a, q') = 1$,
  - $\forall (q, a, \mathbf{v}, \alpha, q') \in E, a = \sharp \Rightarrow q' = q_0$.

3 THE LEARNING SAMPLE

When learning, the algorithm will be given a randomly drawn sample: the pairs of strings will be drawn following the joint distribution defined by the target PST. Therefore, such a sample is a multiset, since more frequent translation pairs may occur more than once and is called a *learning sample*. The formal definition of a learning sample is the following:

Definition 2. *A* ***learning sample*** *is a multiset* $S_n \langle X, f \rangle$ *where* $X = \{(u, \mathbf{v}) : u \in \Sigma^*\sharp, \mathbf{v} \in \Omega^*\}$, $f : (u, \mathbf{v}) \to [n]$, *and* $f(u, \mathbf{v})$ *is the multiplicity or number of occurrence of* $(u, \mathbf{v})$ *in* X.

For simplicity, for a given $S_n \langle X, f \rangle$, if $(u\sharp, \mathbf{v}) \in X$, we will write $(u\sharp, \mathbf{v}) \in S_n$ unless the context requires to be more specific. We assume that the learning sample obeys the following property: $\forall (u\sharp, \mathbf{v}), (u\sharp, \mathbf{v}') \in S_n \Rightarrow \mathbf{v} = \mathbf{v}'$, *i.e.,* the translation pairs are *unambiguous.*

4 FREQUENCY TRANSDUCERS

In our proposed solution, the learner utilizes the relative frequency of the observed data. It starts by building a tree like transducer that incorporates the frequencies of the training data and is an exact representation of the training data. In the next phase of the algorithm, the learner makes iterative state merges. The merge acceptance decision is based on the consistency of translation *w.r.t.* the training data and statistical test using the relative frequencies. At this point

we will define a new type of transducer that incorporates the frequencies of translations, called *frequency subsequential transducer*.

Definition 3 (Frequency Finite Subsequential Transducer)**.** *A frequency finite subsequential transducer (*FFST*) is a 6-tuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E, F_r \rangle$ *where* $\psi(T) = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E \rangle$ *is an* ST *and* F_r *is the frequency function defined as* $F_r : e \to \mathbb{N}$*, where* $e \in E$*.*

An FFST is *well defined* or *consistent* if the following property holds:

$$\forall q \in Q \backslash \{q_0\}, \sum_{e \in E : next[e] = q} F_r(e) = \sum_{e \in E[q]} F_r(e) \quad (1)$$

Since q_0 is the only initial state, the sum of the frequencies of the outgoing edges is assumed to be number of frequency entering the initial state, and therefore q_0 is treated specifically. Figure 1(a) depicts an example of a consistent FFST.

Intuitively, an FFST is an object where the weights are the frequencies of the transitions instead of probabilities. $F_r(e) = n$ should be interpreted as: the edge e is used n times. FFSTs can be converted to equivalent PSTs.

Next we define a prefix tree transducer that is an exact representation of the observed sample S_n and holds the frequencies of the strings.

Definition 4 (Frequency Prefix Tree Subsequential Transducer)**.** *A frequency prefix tree subsequential transducer (*FPTST*) is a 6-tuple* $T = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E, F_r \rangle$ *where* $\psi(T) = \langle Q, \Sigma \cup \{\sharp\}, \Omega, \{q_0\}, E, F_r \rangle$ *is an* FFST *and* T *is built from a training sample* S_n *such that:*

- $Q = \bigcup_{(u,\mathbf{v}) \in S_n} \{q_x : x \in Pref(u)\}$,
- $E = \{e \mid prev[e] = q_u, next[e] = q_v \Rightarrow v = ua, a \in \Sigma, i[e] = a, o[e] = \varepsilon\}$,
- $\forall q_u \in Q, \forall e \in E[q],\ i[e] = \sharp, o[e] = \mathbf{v}$ *if* $(u,\mathbf{v}) \in S_n$, $\perp$ *otherwise,*
- $\forall e \in E[q_0], F_r(e) = |\{a : i[e] = a, a \in Pref(\{u : (u,\mathbf{v}) \in S_n\})\}|$,
- $\forall q_u \in Q \setminus \{q_0\}, \forall e \in E[q_u], F_r(e) = |\{a : ua \in Pref(\{x : (x,\mathbf{y}) \in S_n\})\}|$.

An FPTST is said to be in an *onward* form if the following condition holds:

$$\forall q \in Q \backslash \{q_0\}, lcp\left(\bigcup_{e \in E[q]} \{o[e]\}\right) = \varepsilon.$$

An FPTST is essentially an exact representation of the training data augmented with the observed frequencies of the training data. The frequencies can be utilized to make statistical tests: relative frequencies can be compared by means of Hoeffding tests (Hoeffding, 1963) during the state merging phase of the algorithm.

Let f_1, n_1 and f_2, n_2 be two pairs of observed frequencies of symbols and number of trials respectively.

$$\left| \frac{f_1}{n_1} - \frac{f_2}{n_2} \right| < \sqrt{\frac{1}{2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right) \log \frac{2}{\delta}}. \quad (2)$$

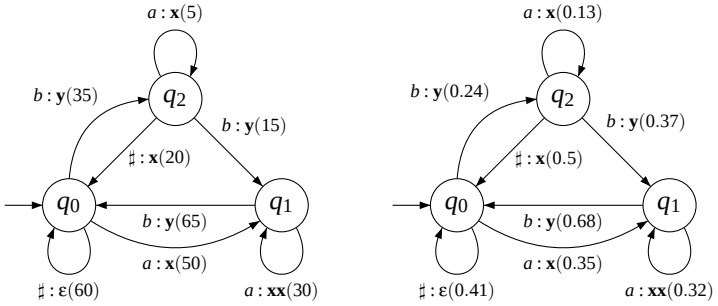
It is noteworthy that the Hoeffding bound is relatively weak or bad approximation and there are better alternatives. However, to demonstrate the proof of concept in our algorithm we will only use the Hoeffding bound.

5 THE ALGORITHM

The proposed algorithm APTI2 executes in three phases: initialization, state merging and conversion of the inferred FFST to an equivalent PST. In the initialization phase, the algorithm builds an FPTST in an onward form, from the learning sample S_n. This is achieved by the routine ONWARDFPTST (Algorithm 1, line 1). Next, the state merging phase of APTI2 begins and is similar to the OSTIA algorithm, with a modified state merging strategy. The details of OSTIA can be found in (Oncina et al., 1993; Castellanos et al., 1998; de la Higuera, 2010). Here we follow the recursive formalism given in (de la Higuera, 2010). The algorithm APTI2 (Algorithm 1) selects a candidate pair of RED and BLUE states in lex-length order using the $\text{CHOOSE}_{<_{lex}}$ function. The MERGE function merges the two selected states and recursively performs a cascade of folding of a number of edges (see (de la Higuera, 2010) for details). As a result of the onwarding process, some of the output strings may come too close to the initial state. During the run of the algorithm these strings or the suffixes of these strings are pushed back in order to make state merging possible by deferring the translations; this is done in the standard way as in OSTIA. During the recursive fold operation, it is actually decided whether a merge is accepted or not. The algorithm APTI2 employs the following condition for a merge to be accepted:

- $\forall e_b \in E[q_b], \forall e_r \in E[q_r], i[e_b] = i[e_r] \Rightarrow$
  1. $o[e_b] = o[e_r]$,
  2. inequality (2) holds for $f_1 = F_r(e_b)$, $n_1 = \sum_{e \in E[q_b]} F_r(e)$ and $f_2 = F_r(e_r)$, $n_2 = \sum_{e \in E[q_r]} F_r(e)$.

Condition 1 is essentially similar to the one used in OSTIA. The second condition is to check whether the relative frequencies related to two states are close



(a) (b)

Figure 1: Figure 1(a) shows an example of an FFST. Figure 1(b) shows an equivalent PST of the FFST shown in Figure 1(a). Notice that the FFST in Figure 1(a) is consistent as per condition (1).

enough. In this case the Hoeffding bound is used as a statistical test. The state merging phase terminates whenever there are no more BLUE states to be merged. Finally, in the third phase, the inferred FFST is converted to an equivalent PST using the hypothetical routine CONVERTFFSTTOPST (Algorithm 1, line 13)

Algorithm 1: APTI2.

Input: a sample S_n
Output: a PST T

1 $T \leftarrow$ ONWARDFPTST(S_n);
2 RED $\leftarrow \{q_\varepsilon\}$;
3 BLUE $\leftarrow \{q_a : a \in \Sigma \cap Pref(\{u : (u,\mathbf{v}) \in S_n)\})\}$;
4 **while** BLUE $\neq \emptyset$ **do**
5 $\quad q =$ CHOOSE$_{<_{lex}}$(BLUE);
6 $\quad$ **for** $p \in$ RED *in lex-length order* **do**
7 $\quad\quad (T', \texttt{IsAccept}) \leftarrow$ MERGE(T, p, q);
8 $\quad\quad$ **if** `IsAccept` **then**
9 $\quad\quad\quad T \leftarrow T'$;
10 $\quad\quad$ **else**
11 $\quad\quad\quad$ RED $\leftarrow$ RED $\cup \{q\}$;
12 $\quad\quad\quad$ BLUE $\leftarrow \{q : \forall p \in$ RED$, \forall e \in E[p], q = next[e] \wedge q \notin$ RED$\}$;
13 CONVERTFFSTTOPST(T);
14 **return** T;

6 ANALYSIS OF THE ALGORITHM

We recall the definition of a learning sample $S_n \langle X, f \rangle$ where the frequency of a translation pair $(u,\mathbf{v})$ is given by $f(u,\mathbf{v})$. Similarly, we define the *prefix frequency* F_{S_n} *w.r.t.* a learning sample S_n as the following: $F_{S_n}(u\Sigma^*,\mathbf{v}) = |\{(w,\mathbf{x}) : (w,\mathbf{x}) \in S_n \wedge u \in Pref(w)\}|$. Less formally, $F_{S_n}(u\Sigma^*,\mathbf{v})$ is the number of translation pairs where the input string starts with the substring u.

We define the *prefix set* ($\mathbb{PR}$) and the *short prefix set* ($\mathbb{SP}$) with respect to a stochastic transduction $\mathcal{R}$ as the following: $\mathbb{PR}(\mathcal{R}) = \{u \in \Sigma^* | (u,\mathbf{v})^{-1}\mathcal{R} \neq \emptyset, \mathbf{v} \in \Omega^*\}$ and $\mathbb{SP}(\mathcal{R}) = \{u \in \mathbb{PR}(\mathcal{R}) | (u,\mathbf{v})^{-1}\mathcal{R} = (w,\mathbf{x})^{-1}\mathcal{R} \Rightarrow |u| \leq |w|\}$. The *kernel set* ($\mathbb{K}$) of $\mathcal{R}$ is defined as follows: $\mathbb{K}(\mathcal{R}) = \{\varepsilon\} \cup \{ua \in \mathbb{PR}(\mathcal{R}) | u \in \mathbb{SP}(\mathcal{R}) \wedge a \in \Sigma\}$. Note that $\mathbb{SP}$ is included in $\mathbb{K}$.

We use the above definitions to define the sufficient conditions a learning sample must obey in order to obtain a correct PST. The sufficient condition for a learning sample $S_n \langle X, f \rangle$ in order to learn the syntactic machine *w.r.t.* an SDRT $\mathcal{R}$ are the following:

1. $\forall u \in \mathbb{K}, \exists (uw,\mathbf{vx}) \in X$ such that $w \in \Sigma^*, \mathbf{vx} \in \Omega^*, (w,\mathbf{x}) \in (u,\mathbf{v})^{-1}\mathcal{R}$,
2. $\forall u \in \mathbb{SP}, u' \in \mathbb{K}$, if $(u,\mathbf{v})^{-1}\mathcal{R} \neq (u',\mathbf{v}')^{-1}\mathcal{R}$ where $\mathbf{v},\mathbf{v}' \in \Omega^*$, then any one of the following holds:
   (a) $\exists (uw,\mathbf{vx}), (u'w,\mathbf{v}'\mathbf{x}'), (ur,\mathbf{vy}), (u'r,\mathbf{v}'\mathbf{y}') \in X$ such that for a given value of δ:
   $$\left| \frac{\left|f(uw,\mathbf{vx}) - f(ur,\mathbf{vy})\right|}{F_{S_n}(u\Sigma^*,\mathbf{z})} - \frac{\left|f(u'w,\mathbf{v}'\mathbf{x}') - f(u'r,\mathbf{v}'\mathbf{y}')\right|}{F_{S_n}(u'\Sigma^*,\mathbf{z}')} \right| < \sqrt{\frac{1}{2}\log\frac{2}{\delta}\left(\frac{1}{F_{S_n}(u\Sigma^*,\mathbf{z})} + \frac{1}{F_{S_n}(u'\Sigma^*,\mathbf{z}')}\right)}$$
   (b) $\exists (uw,\mathbf{vx}), (u'w,\mathbf{v}'\mathbf{x}'), \in X$ such that: $(w,\mathbf{x}) \in (u,\mathbf{v})^{-1}\mathcal{R}, (w,\mathbf{x}') \in (u',\mathbf{v}')^{-1}\mathcal{R}, x \neq x'$
3. $\forall u \in \mathbb{K}, \exists (uw,\mathbf{vx})(uw',\mathbf{vx}') \in X$ such that $lcp(\mathbf{x},\mathbf{x}') = \varepsilon \wedge w \neq w'$.

With the conditions above, the properties of a learning sample S_n have been formalized in order to guarantee the algorithm to learn correctly. The condition 1, 2(b), and 3 are essentially similar to its non-probabilistic counterpart OSTIA (Oncina et al., 1993). Condition 1 is to ensure that there are at least as many states in the learning phase as the target transducer. Condition 2(b) is for making merge decisions *w.r.t.* the output strings. Condition 3 ascertains the alignments of the output strings by factorizing them during

the state merging phase. Condition 2(b) is not guaranteed if the transduction scheme is not a total function (Oncina et al., 1993), *i.e.*, it is not sufficient to make the merge decision if the transduction scheme is a partial function. In order to overcome this issue, in our proposed algorithm we have used the frequencies of the given data. Condition 2(a) ensures that the relative frequencies of the observed data is sufficient to distinguish non-mergable states by means of Hoeffding bound. The inequality shown in condition 2(a) is from inequality 2.

Notice that condition 2(a) depends on the value of δ. Carrasco and Oncina have discussed how a large or a small value of δ affects the merge decision in (Carrasco and Oncina, 1994). Basically, if the size of the learning sample S_n is significantly large, one could keep δ negligibly small. On the contrary, for a relatively small size of learning sample S_n, δ requires to be sufficiently high.

The runtime complexity of algorithm APTI2 is given by $O((\|S_n\|)^3(m+n)+\|S_n\|mn)$, where: $\|S_n\| = \sum_{(u,v)\in S_n}|u|$ and $m = max\{|u| : (u,\mathbf{v}) \in S_n\}$. The FPTST can be built in linear time *w.r.t.* $\|S_n\|$. We will now analyze the outermost while loop in the APTI2 algorithm. Being pessimistic, there will be at most $\|S_n\|$ number of states in the FPTST. In the worse case, if no merges are accepted, there will be $O(\|S_n\|)$ executions of the outermost while loop and $O(\|S_n\|^2)$ executions of the inner for loop, resulting $O(\|S_n\|^3)$ executions of the core algorithm. In each of these executions, *lcp* operation can be implemented in $O(m)$ times and the pushback operation in $O(n)$ times. Assuming that all arithmetic operations are computed in unit time, the total core operation of APTI2 can be bounded by $O((\|S_n\|)^3(m+n)+\|S_n\|mn)$. This runtime complexity is pessimistic and the runtime of APTI2 is much lower in practice. Experimental evidence of runtime of APTI2 is presented in the next section.

7 EXPERIMENTAL RESULTS

We conduct our experiments with two types of data sets: 1) artificial data sets generated from random transducers 2) data generated from the Miniature Language Acquisition (MLA) task (Feldman et al., 1990) adapted to English-French translations.

For the artificial data sets, we first generate a random PST with m states. The states are numbered from q_0 to q_{m-1} where state q_0 is the initial state. The states are connected randomly; labels on transitions preserve the deterministic property. Then the unreachable states are removed. The outputs are assigned as random strings drawn from a uniform distribution over $\Omega^{\leq k}$, for an arbitrary value of k. The probabilities of the edges are randomly assigned making sure the following condition holds:

$$\forall q_i \in Q, \sum_{e\in E[q_i]} prob[e] = 1 \qquad (3)$$

Using the target PST, the training sample is generated following the paths of the PST. The test data is also generated in the similar manner. In order to test the algorithm with unseen examples, we make sure that the test set and the training set are disjoint.

As a measure of correctness we compute two metrics: word error rate (WER) and sentence error rate (SER). Intuitively, WER is the percentage of symbol errors in the hypothesis translation *w.r.t.* the reference translation. For each test pair, the Levenshtein distance (Levenshtein, 1966) between the reference translation and hypothesis translation is computed and divided by the length of the reference string. The mean of the scores computed for each test pair is reported as the WER. On the other hand the SER is more strict; it is the percentage of wrong hypothesis translations *w.r.t.* the reference translations.

The objectives of the first experiment (see Figure 2(a) and Figure 2(b)) are to demonstrate the correctness of our algorithm and to study the practical runtime. Figure 2(a) shows experiments conducted on randomly generated PSTs with 5 states and $|\Sigma| = |\Omega| = 2$. We start with a training sample size 200, we keep incrementing the training size by 200 up to a size of 20000. For every training size the experiment is repeated 10 times by generating new datasets. The mean of these 10 experimental results is reported. We have conducted the experiment for 10 random PSTs. Thus, in total we have conducted 10000 trials. Figure 2(a) shows the mean of the results obtained from 10 random PSTs. As the Figure 2(a) shows, the error rate is approaching zero. As expected, the WER in most cases remains below SER. The execution time for this experiment is reported in Figure 2(b). The results of this experiment tell us that APTI2 shows acceptable error rate with 5000 training examples, and from a training size of 10000 the error rate is close to zero. The runtime in practice is much lower than the theoratical bound and almost linear.

In our second experiment (Figures 2(c) and 2(d)), we aim to learn slightly larger PSTs. In this second experiment we have taken a randomly generated PSTs with 10 states and $|\Sigma| = |\Omega| = 2$. We start with a training sample size 1000, we keep incrementing the training size by 1000 up to a size of 40000. Similar to the previous experiment, experiments with each training sample size are repeated 10 times. The experiment is conducted for 10 random PSTs. The results of this ex-

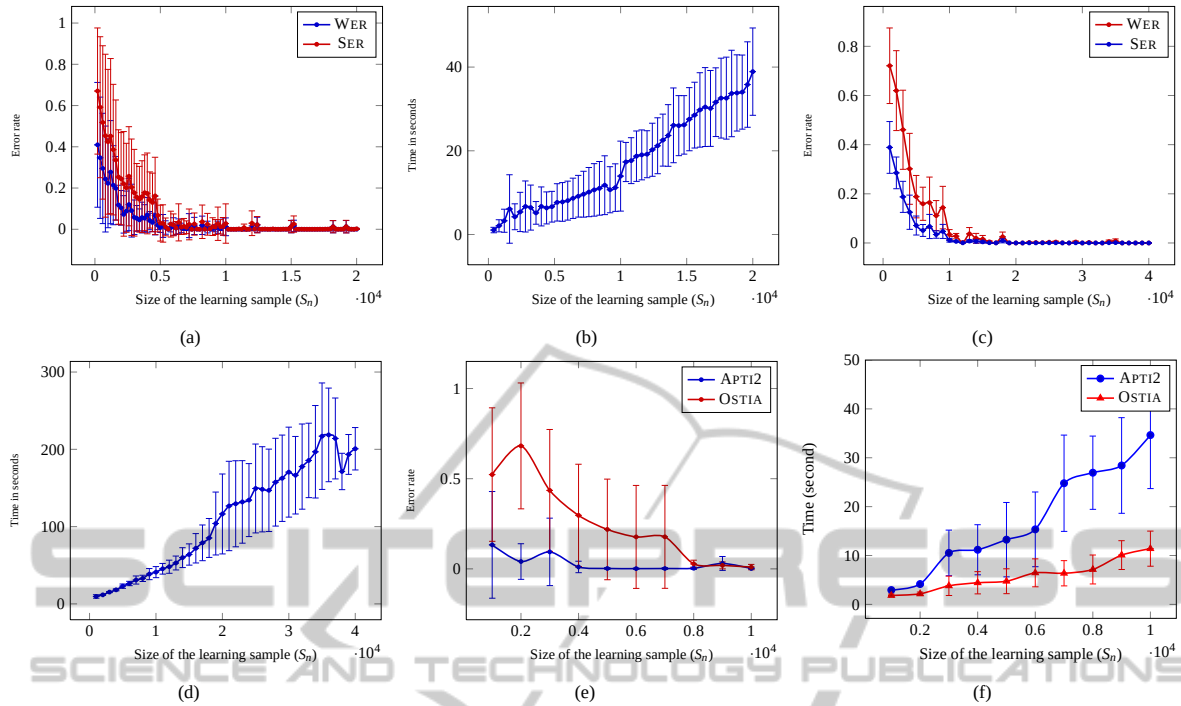


Figure 2: Performance of APTI2 reported for the artificially generated data set (Figures 2(a), 2(b), 2(c), and 2(d)) and the data set for the MLA task (Figures 2(e), 2(f)).

periment show us APTI2 requires larger size of learning sample when the number of states of the target machine is bigger and again, the execution time remains reasonable. However, a target size of 10 is still not large enough for many practical tasks.

Finally, we have conducted another experiment with the Feldman dataset. The target PST consists of 22 states. The objectives of this experiment were to make a comparison of prediction accuracy and of runtime for OSTIA and APTI2. Here we start with 1000 training pairs and incremented by 1000 till obtaining 10000 training pairs. Each data point is repeated 10 times for statistical significance. The results are depicted in Figure 2(e) and 2(f). The results of Figure 2(e) demonstrate significant improvement in prediction accuracy of APTI2 in comparison with OSTIA. APTI2 attains almost perfect accuracy with only 4000 training examples while OSTIA continues to make an error of about 30%. As we increase the number of training data, at one point, when the number of training examples is about 8000, OSTIA also performs as well as APTI2. However, APTI2 attains this accuracy rate with only half of the number of training examples. Figure 2(f) shows that the execution time of APTI2 is higher than that of OSTIA, yet it always remains reasonable (below 40 seconds).

We have implemented APTI2 using an open source C++ library for weighted transducers (Allauzen et al., 2007). Figures 2(b), 2(d), and 2(f) show the execution times for APTI2 using our implementation. Although, the theoretical worst case runtime complexity of APTI2 is cubic, in practice APTI2 exhibits much lower execution time.

8 CONCLUSIONS

We have presented a learning algorithm APTI2 that learns any PST provided a characteristic training sample is given. We have also presented experimental results based on synthetic data to proof the correctness of our algorithm. Moreover, based on our implementation, we have reported that the runtime complexity of APTI2 in practice is much lower than the theoretical worst case runtime complexity.

The limitation of our work is twofold: first, our model is restricted to regular stochastic bi-grammar, and hence not capable of capturing many practical scenarios, *e.g.*, natural languages. Second, as a statistical test we have used a basic Hoeffding bound. We believe that more sophisticated statistical tests will lead to better accuracy of the algorithm.

REFERENCES

Akram, H. I., de la Higuera, C., and Eckert, C. (2012). Actively learning probabilistic subsequential transducers. In *Proceedings of ICGI*, volume 21 of *JMLR:Workshop and Conference Proceedings*. MIT Press.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: A general and efficient weighted finite-state transducer library. In *Proceedings of CIAA*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer.

Carrasco, R. C. and Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *Proceedings of ICGI*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer.

Carrasco, R. C. and Oncina, J. (1999). Learning deterministic regular grammars from stochastic samples in polynomial time. *Rairo - Theoratical Informatics and Applications*, 33(1):1–20.

Castellanos, A., Vidal, E., Varó, M. A., and Oncina, J. (1998). Language understanding and subsequential transducer learning. *Computer Speech & Language*, 12(3):193–228.

de la Higuera, C. (2010). *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Feldman, J. A., Lakoff, G., Stolcke, A., and Weber, S. H. (1990). Miniature language acquisition: A touchstone for cognitive science. Technical Report TR-90-009, International Computer Science Institute, Berkeley CA.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707.

Oncina, J. and García, P. (1991). Inductive learning of subsequential functions. Technical report, Univ. Polit'ecnica de Valencia.

Oncina, J., García, P., and Vidal, E. (1993). Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):448–458.

Rabiner, L. R. (1990). Readings in speech recognition. chapter A tutorial on hidden Markov models and selected applications in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc.

Thollard, F., Dupont, P., and de la Higuera, C. (2000). Probabilistic dfa inference using Kullback-Leibler divergence and minimality. In *Proceedings of ICML*, pages 975–982. Morgan Kaufmann.

APPENDIX

An Example Run of the APTI2 Algorithm

As an example of a stochastic regular bi-language, let us consider the following:

Example 2. *The transduction* $\mathcal{R} : \Sigma^*\sharp \times \Omega^* \to \mathbb{R}_+$ *where* $Pr_{\mathcal{R}}(a^n b a^m \sharp, \mathbf{x}^n \mathbf{y} \mathbf{x}^m) = \frac{1}{2(3^n 4^m)}, \forall n, m \geq 0$. *and* $Pr_{\mathcal{R}}(u, \mathbf{v}) = 0$ *for every other pairs.*

Figure 3 shows the PST in the onward and minimal form (for details about the minimal form see (Akram et al., 2012)) that generates $\mathcal{R}$. We will consider the PST shown in Figure 3 as the target PST. The training data *w.r.t.* the target PST (Figure 3) is given in Table 1.

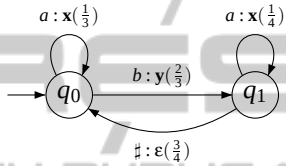


Figure 3: The PST in canonical normal form that generates $\mathcal{R}$ defined in Example 2.

Table 1: Training data for the target PST in Figure 3.

input	output	frequency
$b\sharp$	**y**	500
$ab\sharp$	**xy**	160
$ba\sharp$	**x**	120
$aab\sharp$	**xxy**	50
$aba\sharp$	**xyx**	40
total		870

Next, an onward FPTST is built from the data given in Table 1. The FPTST is shown in Figure 4. The states q_0 is initiated as a RED state and states q_1 and q_2 are initiated as BLUE states. Here for the Hoeffding bound test, the value of δ is set arbitrarily to 0.5.

The first merge candidate pair of states are q_0 and q_1. The merge between them is accepted and the resulting transducer is shown in Figure 5.

Next, the algorithm tries to merge q_0 and q_0 (Figure 6). This merge is rejected because the Hoeffding bound test (Equation 2) returns false. The state q_2 is promoted to RED and consequently the states q_5 and q_6 are added as RED states (Figure 7).

Then the Algorithm tries to merge the states q_0 and q_5 which is also rejected because the Equation 2 returns false. Now the next candidate merge pair is q_2 and q_5. This merge is accepted (Figure 8).

The next candidate pair of merge is q_0 and q_6 which is accepted and the resulting transducer is depicted in Figure 9.

Finally, the FFST is converted into a PST (Figure 10) and the algorithm terminates. The inferred PST is shown in Figure 10.

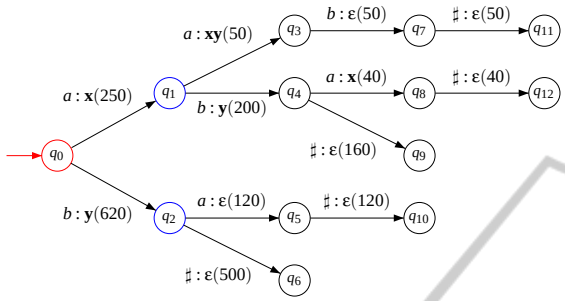


Figure 4: An onward FPTST built from stochastic sample given in Table 1.

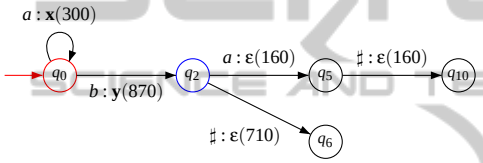


Figure 5: After merging and folding q_0 and q_1.

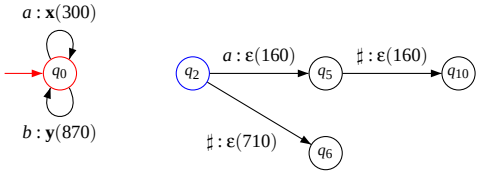


Figure 6: Merge between q_0 and q_2 is rejected.

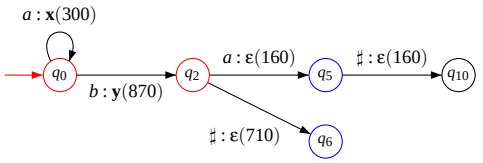


Figure 7: q_2 is promoted to RED and as a consequence of that q_5 and q_6 are added to BLUE.

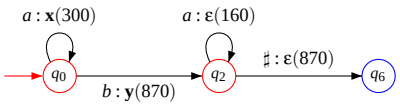


Figure 8: After merging and folding q_2 and q_5.

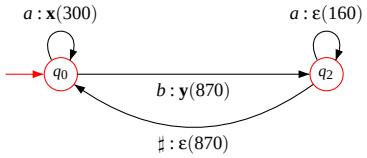


Figure 9: After merging and folding q_0 and q_6.

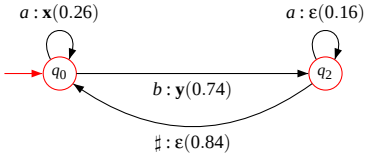


Figure 10: After converting the FFST shown in Figure 7 to a PST.