Using Learned Conditional Distributions as Edit Distance

Jose Oncina, Marc Sebban

Using Learned Conditional Distributions as Edit Distance*

Jose Oncina[1] and Marc Sebban[2]

[1] Dep. de Lenguajes y Sistemas Informáticos, Universidad de Alicante (Spain)
oncina@dlsi.ua.es
[2] EURISE, Université de Saint-Etienne, (France)
marc.sebban@univ-st-etienne.fr

Abstract. In order to achieve pattern recognition tasks, we aim at learning an *unbiased* stochastic edit distance, in the form of a finite-state transducer, from a corpus of (*input,output*) pairs of strings. Contrary to the state of the art methods, we learn a transducer independently on the marginal probability distribution of the *input* strings. Such an unbiased way to proceed requires to optimize the parameters of a *conditional* transducer instead of a *joint* one. This transducer can be very useful in pattern recognition particularly in the presence of noisy data. Two types of experiments are carried out in this article. The first one aims at showing that our algorithm is able to correctly assess simulated theoretical target distributions. The second one shows its practical interest in a handwritten character recognition task, in comparison with a standard edit distance using *a priori* fixed edit costs.

1 Introduction

Many applications dealing with sequences require to compute the similarity of a pair (*input,output*) of strings. A widely-used similarity measure is the well known *edit distance*, which corresponds to the minimum number of operations, *i.e. insertions, deletions*, and *substitutions*, required to transform the *input* into the *output*. If this transformation is based on a random phenomenon and then on an underlying probability distribution, edit operations become random variables. We call then the resulting similarity measure, the *stochastic edit distance*.

An efficient way to model this distance consists in viewing it as a stochastic transduction between the input and output alphabets [1]. Stochastic finite-state transducers suffer from the lack of a training algorithm. To the best of our knowledge, the first published algorithm to automatically learn the parameters of a stochastic transducer has been proposed by Ristad and Yianilos [2, 1]. They provide a stochastic model which allows us to learn a stochastic edit distance, in the form of a memoryless transducer (*i.e.* with only one state), from a corpus of similar examples, using the Expectation Maximization (EM) algorithm. During the last few years, the algorithm EM has also been used for learning other transducer-based models [3–5].

* This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

Ristad and Yianilos define the stochastic edit distance between two strings x and y as (the minus logarithm of) the *joint* probability of the pair (x, y). In this paper, we claim that it would be much more relevant to express the stochastic edit distance from a *conditional* probability.

First, in order to correctly compute the edit distance, we think that the probabilities of edit operations over a symbol must be independent of those computed over another symbol. In other words, if the transformation of a string x into another one y does not require many edit operations, it is expected that the probability of the substitution of a symbol by itself should be high. But, as the sum of the probabilities of all edit operations is one, then the probability of the substitution of another symbol by itself can not obviously be too large. Thus, by using a joint distribution (summing to 1), one generates an awkward dependence between edit operations.

Moreover, we think that the primitive edit costs of the edit distance must be independent of the *a priori* distribution $p(x)$ of the input strings. However, $p(x)$ can be directly deduced from the joint distribution $p(x, y)$, as follows: $p(x) = \sum_{y \in Y^*} p(x, y)$, where Y^* is the set of all finite strings over the output alphabet Y. This means that this information is totally included in the joint distribution. By defining the stochastic edit distance as a function of the joint probability, as done in [1], the edit costs are then dependent of $p(x)$. However, if we use a conditional distribution, this dependence is removed, since it is impossible to obtain $p(x)$ from $p(y|x)$ alone.

Finally, although it is sensible and practical to model the stochastic edit distance by a memoryless transducer, it is possible that the *a priori* distribution $p(x)$ may not be modeled by such a very simple structure. Thus, by learning a transducer defining the joint distribution $p(x, y)$, its parameters can converge to compromise values and not to the true ones. This can have dramatic effects from an application standpoint. Actually, a widely-used solution to find an optimal output string y according to an input one x consists in first learning the joint distribution transducer and later deducing the conditional transducer dividing by $p(x)$ (more precisely by its estimates over the learning set). Such a strategy is then irrelevant for the reason we mentioned above.

In this paper we have developed a way to learn directly the conditional transducer. After some definitions and notations (Section 2), we introduce in Section 3 the learning principle of the stochastic edit distance proposed by Ristad and Yianilos [2, 1]. Then, by simulating different theoretical joint distributions, we show that the *unique way*, using their algorithm, to find them consists in sampling a learning set of (x, y) pairs according to the marginal distribution (*i.e.* over the input strings) of the target joint distribution itself. Moreover, we show that for any other *a priori* distribution, the difference between the target and the learned model increases. To free the method from this bias, one must *directly* learn at each iteration of the algorithm EM the conditional distribution $p(y|x)$. Achieving this task requires to modify Ristad and Yianilos's framework. That is the goal of Section 4. Then, we carry out experiments that show that it is possible to correctly estimate a target distribution whatever the *a priori* distribution we use. Section 5 is devoted to compare both models (along with two versions of the classic edit distance) in a character recognition task.

2 Notation

An alphabet X is a finite nonempty set of symbols. X^* denotes the set of all finite strings over X. Let $x \in X^*$ be an arbitrary string of length $|x|$ over the alphabet X. In the following, unless stated otherwise, symbols are indicated by $a, b, \ldots$, strings by $u, v, \ldots, z$, and the empty string by λ. $\mathbb{R}^+$ is the set of non negative reals. Let $f(\cdot)$ be a function, from which $[f(x)]_{\pi(x,\ldots)}$ is equal to $f(x)$ if the predicate $\pi(x, \ldots)$ holds and 0 otherwise, where x is a (set of) dummy variable(s).

3 Stochastic Edit Distance and Memoryless Transducers

A joint memoryless transducer defines a joint probability distribution over the pairs of strings. It is denoted by a tuple (X, Y, c, γ) where X is the input alphabet, Y is the output alphabet, c is the *primitive* joint probability function, $c : E \rightarrow [0, 1]$ and γ is the probability of the termination symbol of a string. As $(\lambda, \lambda) \notin E$, in order to simplify the notations, we are going to use $c(\lambda, \lambda)$ and γ as synonyms.

Let us assume for the moment that we know the probability function c (in fact, we will learn it later). We are then able to compute the joint probability $p(x, y)$ of a pair of strings (x, y). Actually, the joint probability $p : X^* \times Y^* \rightarrow [0, 1]$ of the strings x, y can be recursively computed by means of an auxiliary function (forward) $\alpha : X^* \times Y^* \rightarrow \mathbb{R}^+$ or, symmetrically, by means of an auxiliary function (backward) $\beta : X^* \times Y^* \rightarrow \mathbb{R}^+$ as:

$$\begin{aligned}
\alpha(x, y) = {} & [1]_{x=\lambda \wedge y=\lambda} \\
& + [c(a, b) \cdot \alpha(x', y')]_{x=x'a \wedge y=y'b} \\
& + [c(a, \lambda) \cdot \alpha(x', y)]_{x=x'a} \\
& + [c(\lambda, b) \cdot \alpha(x, y')]_{y=y'b}.
\end{aligned}
\qquad
\begin{aligned}
\beta(x, y) = {} & [1]_{x=\lambda \wedge y=\lambda} \\
& + [c(a, b) \cdot \beta(x', y')]_{x=ax' \wedge y=by'} \\
& + [c(a, \lambda) \cdot \beta(x', y)]_{x=ax'} \\
& + [c(\lambda, b) \cdot \beta(x, y')]_{y=by'}.
\end{aligned}$$

And then, $p(x, y) = \alpha(x, y)\gamma$ or $p(x, y) = \beta(x, y)\gamma$.

Both functions (forward and backward) can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique. This model defines a probability distribution over the pairs (x, y) of strings. More precisely,

$$\sum_{x \in X^*} \sum_{y \in Y^*} p(x, y) = 1,$$

that is achieved if the following conditions are fulfilled [1],

$$\gamma > 0, c(a, b), c(\lambda, b), c(a, \lambda) \geq 0 \quad \forall a \in X, b \in Y$$

$$\sum_{\substack{a \in X \cup \{\lambda\} \\ b \in Y \cup \{\lambda\}}} c(a, b) = 1$$

Given $p(x, y)$, we can then compute, as mentioned in [1], the stochastic edit distance between x and y. Actually, the stochastic edit distance $d_s(x, y)$ is defined as being

Table 1. Target joint distribution $c^*(a, b)$ and its corresponding marginal distribution $c^*(a)$.

$c^*(a,b)$	λ	a	b	c	d	$c^*(a)$
λ	0.00	0.05	0.08	0.02	0.02	0.17
a	0.01	0.04	0.01	0.01	0.01	0.08
b	0.02	0.01	0.16	0.04	0.01	0.24
c	0.01	0.02	0.01	0.15	0.00	0.19
d	0.01	0.01	0.01	0.01	0.28	0.32

the negative logarithm of the probability of the string pair $p(x, y)$ according to the memoryless stochastic transducer.

$$d_s(x, y) = -\log p(x, y), \forall x \in X^*, \forall y \in Y^*$$

Let S be a finite set of (x, y) pairs of *similar* strings. Ristad and Yianilos [1] propose to use the expectation-maximization (EM) algorithm to find an optimal joint stochastic transducer. The EM algorithm consists in two steps (expectation and maximization) that are repeated until a convergence criterion is achieved.

Given an auxiliary $(|X| + 1) \times (|Y| + 1)$ matrix δ, the expectation step can be described as follows: $\forall a \in X, b \in Y$,

$$\delta(a, b) = \sum_{(xax', yby') \in S} \frac{\alpha(x, y)c(a, b)\beta(x', y')\gamma}{p(xax', yby')} \quad \delta(\lambda, b) = \sum_{(xx', yby') \in S} \frac{\alpha(x, y)c(\lambda, b)\beta(x', y')\gamma}{p(xx', yby')}$$

$$\delta(a, \lambda) = \sum_{(xax', yy') \in S} \frac{\alpha(x, y)c(a, \lambda)\beta(x', y')\gamma}{p(xax', yy')} \quad \delta(\lambda, \lambda) = \sum_{(x,y) \in S} \frac{\alpha(x, y)\gamma}{p(x, y)} = |S|,$$

and the maximization as:

$$c(a, b) = \frac{\delta(a, b)}{N} \quad \forall a \in X \cup \{\lambda\}, \forall b \in Y \cup \{\lambda\} \text{ where } N = \sum_{\substack{a \in X \cup \{\lambda\} \\ b \in Y \cup \{\lambda\}}} \delta(a, b).$$

To analyze the ability of Ristad and Yianilos's algorithm to correctly estimate the parameters of a target joint memoryless transducer, we carried out a series of experiments.

We simulated a target joint memoryless transducer from the alphabets $X = Y = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, such as $\forall a \in X \cup \{\lambda\}, \forall b \in Y \cup \{\lambda\}$, the target model is able to return the primitive theoretical joint probability $c^*(a, b)$. The target joint distribution we used is described in Table 1[3]. The marginal distribution $c^*(a)$ can be deduced from this target such that: $c^*(a) = \sum_{b \in X \cup \{\lambda\}} c^*(a, b)$.

Then, we sampled an increasing set of learning input strings (from 0 to 4000 sequences) of variable length generated from a given probability distribution $p(a)$ over

[3] Note that we carried out many series of experiments with various target joint distributions, and all the results we obtained follow the same behavior as the one presented in this section.

the input alphabet X. In order to simplify, we modeled this distribution in the form of an automaton with only one state[4] and $|X|$ output transitions with randomly chosen probabilities.

We used different settings for this automaton to analyze the impact of the input distribution $p(a)$ on the learned joint model. Then, given an input sequence x (generated from this automaton) and the target joint distribution $c^*(a, b)$, we sampled a corresponding output y. Finally, the set S of generated (x, y) pairs was used by Ristad and Yianilos's algorithm to learn an estimated primitive joint distribution $c(a, b)$.

We compared the target and the learned distributions to analyze the behavior of the algorithm to correctly assess the parameters of the target joint distribution. We computed an average difference between the both, defined as follows:

$$d(c, c^*) = \frac{\sum_{a \in X \cup \{\lambda\}} \sum_{b \in Y \cup \{\lambda\}} |c(a, b) - c^*(a, b)|}{2}$$

Normalized in this way, $d(c, c^*)$ is a value in the range $[0, 1]$. Figure 1 shows the behavior of this difference according to various configurations of the automaton. We can note that the unique way to converge towards a difference near from 0 consists in using the marginal distribution $c^*(a)$ of the target for generating the input strings. For all the other ways, the difference becomes very large.

As we said at the beginning of this article, we can easily explain this behavior. By learning the primitive joint probability function $c(a, b)$, Ristad and Yianilos learn at the same time the marginal distribution $c(a)$. The learned edit costs (and the stochastic edit distance) are then dependent of the *a priori* distribution of the input strings, that is obviously awkward. To free of this statistical bias, we have to learn the primitive conditional probability function independently of the marginal distribution. That is the goal of the next section.

4 Unbiased Learning of a Conditional Memoryless Transducer

A conditional memoryless transducer is denoted by a tuple (X, Y, c, γ) where X is the input alphabet, Y is the output alphabet, c is the primitive conditional probability function $c : E \rightarrow [0, 1]$ and γ is the probability of the termination symbol of a string. As in the joint case, since $(\lambda, \lambda) \notin E$, in order to simplify the notation we use γ and $c(\lambda|\lambda)$ as synonyms.

The probability $p : X^* \times Y^* \rightarrow [0, 1]$ of the string y assuming the input one was a x (noted $p(y|x)$) can be recursively computed by means of an auxiliary function (forward) $\alpha : X^* \times Y^* \rightarrow \mathbb{R}^+$ or, in a symmetric way, by means of an auxiliary function (backward) $\beta : X^* \times Y^* \rightarrow \mathbb{R}^+$ as:

$$\begin{aligned}
\alpha(y|x) = {} & [1]_{x=\lambda \wedge y=\lambda} \\
& + [c(b|a) \cdot \alpha(y'|x')]_{x=x'a \wedge y=y'b} \\
& + [c(\lambda|a) \cdot \alpha(y|x')]_{x=x'a} \\
& + [c(b|\lambda) \cdot \alpha(y'|x)]_{y=y'b}.
\end{aligned}
\qquad
\begin{aligned}
\beta(y|x) = {} & [1]_{x=\lambda \wedge y=\lambda} \\
& + [c(b|a) \cdot \beta(y'|x')]_{x=ax' \wedge y=by'} \\
& + [c(\lambda|a) \cdot \beta(y|x')]_{x=ax'} \\
& + [c(b|\lambda) \cdot \beta(y'|x)]_{y=by'}.
\end{aligned}$$

[4] Here also, we tested other configurations leading to the same results.

And then, $p(y|x) = \alpha(y|x)\gamma$ and $p(y|x) = \beta(y|x)\gamma$.

As in the joint case, both functions can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique. In this model a probability distribution is assigned conditionally to each input string. Then

$$\sum_{y \in Y^*} p(y|x) \in \{1, 0\} \quad \forall x \in X^*.$$

The 0 is in the case the input string x is not in the domain of the function[5]. It can be show that the normalization of each conditional distribution can be achieved if the following conditions over the function c and the parameter γ are fulfilled,

$$\gamma > 0, c(b|a), c(b|\lambda), c(\lambda|a) \geq 0 \quad \forall a \in X, b \in Y \tag{1}$$

$$\sum_{b \in Y} c(b|\lambda) + \sum_{b \in Y} c(b|a) + c(\lambda|a) = 1 \quad \forall a \in X \tag{2}$$

$$\sum_{b \in Y} c(b|\lambda) + \gamma = 1 \tag{3}$$

As in the joint case, the expectation-maximization algorithm can be used in order to find the optimal parameters. The expectation step deals with the computation of the matrix δ:

$$\delta(b|a) = \sum_{(xax', yby') \in S} \frac{\alpha(y|x)c(b|a)\beta(y'|x')\gamma}{p(yby'|xax')} \quad \delta(b|\lambda) = \sum_{(xx', yby') \in S} \frac{\alpha(y|x)c(b|\lambda)\beta(y'|x')\gamma}{p(yby'|xx')}$$

$$\delta(\lambda|a) = \sum_{(xax', yy') \in S} \frac{\alpha(y|x)c(\lambda|a)\beta(y'|x')\gamma}{p(yy'|xax')} \quad \delta(\lambda|\lambda) = \sum_{(x,y) \in S} \frac{\alpha(y|x)\gamma}{p(y|x)} = |S|.$$

In order to do the maximization step, we begin by normalizing the insertion cost because it appears in both normalization equations (eq. 2 and eq. 3). Then:

$$c(b|\lambda) = \frac{\delta(b|\lambda)}{N} \quad \text{where} \quad N = \sum_{\substack{a \in X \cup \{\lambda\} \\ b \in Y \cup \{\lambda\}}} \delta(b|a)$$

The value of γ is now fixed by eq. 3 as:

$$\gamma = \frac{N - N(\lambda)}{N} \quad \text{where} \quad N(\lambda) = \sum_{b \in Y} \delta(b|\lambda)$$

and $c(b|a)$ and $c(\lambda|a)$ are obtained working out the values in eq. 2 and distributing the probability proportionally to their respective expectations $\delta(b|a)$ and $\delta(\lambda|a)$. Then

$$c(b|a) = \frac{\delta(b|a)}{N(a)} \frac{N - N(\lambda)}{N} \quad c(\lambda|a) = \frac{\delta(\lambda|a)}{N(a)} \frac{N - N(\lambda)}{N} \text{ where } N(a) = \sum_{b \in Y \cup \{\lambda\}} \delta(b|a).$$

[5] If $p(x) = 0$ then $p(x, y) = 0$ and as $p(y|x) = \frac{p(x,y)}{p(x)}$ we have a $\frac{0}{0}$ indeterminism. We chose to solve it taking $\frac{0}{0} = 0$, in order to maintain $\sum_{y \in Y^*} p(y|x)$ finite.

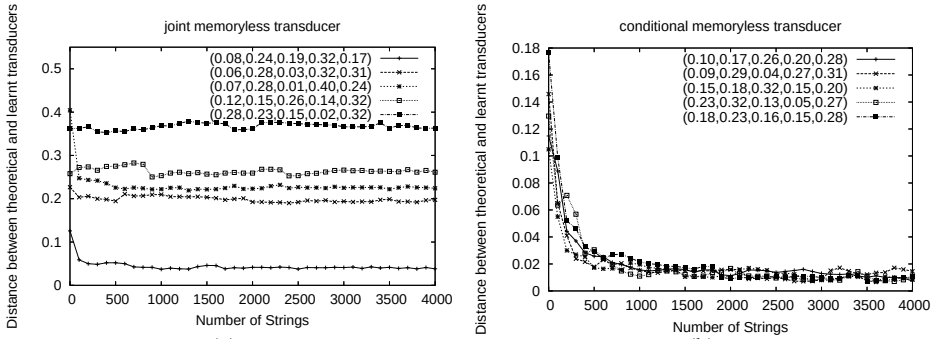

Fig. 1. Average difference between the target and the learned distributions according to various generations of the input strings using a joint (a) and a conditional (b) memoryless transducer. The tuples $(p_a, p_b, p_c, p_d, p_\#)$ represents the probabilities of the symbols a, b, c, d and the probability of ending in the stochastic automaton used to generate the input strings.

We carried out experiments to assess the relevance of our new learning algorithm to correctly estimate the parameters of target transducers. We followed exactly the same experimental setup as the one of the previous section, except to the definition of our difference $d(c, c^*)$. Actually, our new framework estimates $|X|$ conditional distributions. So $d(c, c^*)$ is defined as:

$$d(c, c^*) = \frac{(A + B\,|X|)}{2\,|X|}$$

where $A = \sum_{a \in X} \sum_{b \in Y \cup \{\lambda\}} |c(b|a) - c^*(b|a)|$ and $B = \sum_{b \in Y \cup \{\lambda\}} |c(b|\lambda) - c^*(b|\lambda)|$.

The results are shown in Figure 1. We can make the two following remarks. First, the different curves clearly show that the convergence toward the target distribution is independent of the distribution of the input strings. Using different parameter configurations of the automaton, the behavior of our algorithm remains the same, *i.e* the difference between the learned and the target conditional distributions tends to 0. Second, we can note that $d(c, c^*)$ rapidly decreases, *i.e.* the algorithm requires few learning examples to learn the target.

5 Application to the handwritten character recognition

In order to assess the relevance of our model in a pattern recognition task, we applied it on the real world problem of handwritten digit classification. We used the NIST Special Database 3 of the National Institute of Standards and Technology, already used in several articles such as [6–8]. This database consists in 128 $\times$ 128 bitmap images of handwritten digits and letters. In this series of experiments, we only focus on digits written by 100 different writers. Each class of digit (from 0 to 9) has about 1,000 instances, then the whole database we used contains about 10,000 handwritten digits. Since our model handles strings, we coded each digit as contour chain following the feature extraction algorithm proposed in [6].

As presenting throughout this article, our method requires a set of (input,output) pairs of strings for learning the probabilistic transducer. While it is rather clear that pairs in the form of (noisy,unnoisy) strings constitute the most relevant way to learn an edit distance useful in a noise correction model, what must they represent in a pattern recognition task, with various classes, such as in handwritten digit classification? As already proposed in [1], a possible solution consists in building pairs of "similar" strings that describe the possible variations or distortions between instances of each class. In this series of experiments, we build pairs of (input,output) strings, where the input is a learning string, and the output is the corresponding nearest-neighbor in the learning set. The objective is then to learn a stochastic transducer that allows to optimize the conditional probabilities $p(output/input)$.

In the following series of experiments, we aim at comparing our approach (i) to the one of Ristad and Yianilos, and (ii) to the classic edit distance. Note that for the latter, we used two different matrices of edit costs. The first one is the most classic one, *i.e.* each edit operation has the same cost (here, 1). According to [7], a more relevant strategy would consist in taking costs proportionally to the relative angle between the directions used for describing a digit.

In order to assess each algorithm, the number of learning strings varied from 200 (20 for each class of digits) to 6,000 (600 for each class), with a step of 20 strings per class (resulting in 30 step iterations). The test accuracy was computed with a test set containing always 2,000 strings. For each learning size, we run 5 times each algorithm using 5 different randomly generated learning sets and we computed the average.

From Fig. 2, we can make the following remarks. First of all, learning an edit distance in the form of a conditional transducer is indisputably relevant to achieve a pattern recognition task. Whatever the size of the learning set, the test accuracy obtained using the stochastic edit distance is higher than the others. However, note that the difference decreases logically with the size of the learning set. Whatever the distance we choose, when the number of examples increases, the nearest-neighbor of an example x tends to be x itself. Interestingly, we can also note that for reaching approximately the same accuracy rate, the standard edit distance (using proportional costs) needs much more learning strings, and therefore requires a higher time complexity, than our approach.

Second, when the number of learning string pair is small, all the drawbacks with Ristad and Yianilos's method we already mentioned in the first part of this paper occur. Actually, while a nearest-neighbor is always a string belonging to the learning set, many learning strings are not present in the current (small) set of nearest-neighbors. Therefore, while all these strings (inputs and outputs) come from the same set of digits, the distribution over the outputs (the nearest-neighbors) is not the same as the distribution over the inputs (the learning strings). Of course, this bias decreases with the rise of the learning set size, but not sufficiently in this series of experiments for improving the performances of the classic edit distance.

To assess the level of stability of the approaches, we have computed a measure of dispersion on the results provided by the standard edit distance (with proportional costs) and our learned distance. Fig. 2 shows the behavior of the variance of the test accuracy throughout the iterations. Interestingly, we can note that in the large majority of the cases, our method gives a smaller variance.

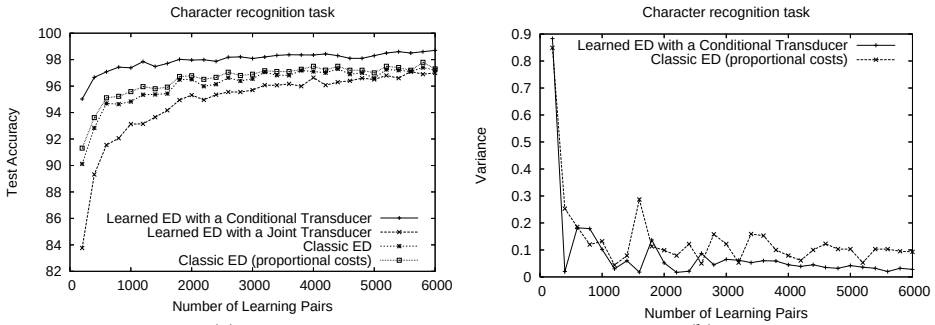

Fig. 2. Evolution of the accuracy (a) and the variance (b) throughout the iterations in the character recognition task.

6 Conclusion

In this paper, we proposed a relevant approach for learning the stochastic edit distance in the form of a memoryless transducer. While the standard techniques aim at learning a joint distribution over the edit operations, we showed that such a strategy induces a bias in the form of a statistical dependence on the input string distribution. We overcame this drawback by directly learning a conditional distribution of the primitive edit costs. The experimental results bring to the fore the interest of our approach. We think that our model is particularly suited for dealing with noisy data.

References

1. Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **20**(5) (1998) 522–532
2. Ristad, E.S., Yianilos, P.N.: Finite growth models. Technical Report CS-TR-533-96, Princeton University Computer Science Department (1996)
3. Casacuberta, F.: Probabilistic estimation of stochastic regular syntax-directed translation schemes. In: Proceedings of th VIth Spanish Symposium on Pattern Recognition and Image Analysis. (1995) 201–207
4. Clark, A.: Memory-based learning of morphology with stochastic transducers. In: Proceedings of the Annual meeting of the association for computational linguistic. (2002)
5. Eisner, J.: Parameter estimation for probabilistic finite-state transducers. In: Proceedings of the Annual meeting of the association for computational linguistic. (2002) 1–8
6. Gómez, E., Micó, L., Oncina, J.: Testing the linear approximating eliminating search algorithm in handwritten character recognition tasks. In: VI Symposium Nacional de reconocimiento de Formas y Análisis de Imágenes. (1995) 212–217
7. Micó, L., Oncina, J.: Comparison of fast nearest neighbour classifiers for handwritten character recognition. Pattern Recognition Letters **19** (1998) 351–356
8. Rico-Juan, J.R., Micó, L.: Comparison of aesa and laesa search algorithms using string and tree-edit-distances. Pattern Recognition Letters **24** (2003) 1417–1426