Learning State Machine-based String Edit Kernels ☆

Aurélien Bellet, Marc Bernard, Thierry Murgue, Marc Sebban

Université de Lyon, F-42023, Saint-Étienne, France
CNRS, UMR 5516, Laboratoire Hubert Curien, F-42000, Saint-Étienne, France
Université de Saint-Étienne, Jean-Monnet, F-42000, Saint-Étienne, France

Abstract

During the past few years, several works have been done to derive string kernels from probability distributions. For instance, the Fisher kernel uses a generative model M (*e.g.* a hidden markov model) and compares two strings according to how they are generated by M. On the other hand, the *marginalized kernels* allow the computation of the joint similarity between two instances by summing conditional probabilities. In this paper, we adapt this approach to *edit distance*-based conditional distributions and we present a way to learn a new *string edit kernel*. We show that the practical computation of such a kernel between two strings x and x' built from an alphabet Σ requires (i) to learn edit probabilities in the form of the parameters of a stochastic state machine and (ii) to calculate an infinite sum over Σ^* by resorting to the intersection of probabilistic automata as done for *rational kernels*. We show on a handwritten character recognition task that our new kernel outperforms not only the state of the art string kernels and string *edit* kernels but also the standard edit distance used by a neighborhood-based classifier.

Key words: String kernel, marginalized kernel, learned edit distance.

☆This work is part of the ongoing ARA Marmota research project.

Email addresses: `Aurelien.Bellet@bvra.univ-st-etienne.fr` (Aurélien Bellet), `Marc.Bernard@univ-st-etienne.fr` (Marc Bernard), `Thierry.Murgue@univ-st-etienne.fr` (Thierry Murgue), `Marc.Sebban@univ-st-etienne.fr` (Marc Sebban)

1. Introduction

With the success of kernel-based learning methods [1], a number of structural kernels have been proposed in the literature to handle structured data such as strings. A first natural way to design string kernels consists in representing each sequence by a **fixed-length numerical vector**. In this context, the *spectrum kernel* proposed in [2] was one of the first string kernels. It relies on the principle that two strings should be considered as similar if they share a large number of contiguous subsequences. In the specific case of a *p-spectrum*, a string is represented by a histogram of frequencies of all contiguous subsequences of length p. Then, a dot product can be performed between two feature vectors to compute the kernel. To overcome the inconveniences of such an exact matching, the *mismatch kernel* [3] generalizes the spectrum one by comparing the frequencies of subsequences which are not exactly the same, *i.e.* allowing a given number of mismatches. Another extension of the spectrum kernel, called *subsequence kernel* [4], considers features made of possibly non-contiguous subsequences. Finally, in the specific context of protein remote homology detection, the distant segments kernel [5] extends the spectrum kernel to include positional information of polypeptide segments. Each component of the feature vector denotes the number of times a segment x is located at a given distance following a segment x'.

Another way to build string kernels while admitting error-tolerant matching consists of deriving kernels **from similarity measures**. For instance, Saigo et al. [6] designed a *local alignment kernel* to detect remote homologies in protein sequences. This kernel assesses the similarity between two sequences by summing up scores computed from local alignments allowing gaps. Other kernels have been derived from the *edit distance* [7], which corresponds to the minimal number of operations (in terms of insertion, deletion or substitution of symbols) to transform a string into another one. The edit distance is a metric and tightly related to the optimal alignment between two sequences. The resulting kernels based on the edit distance are usually called *string edit kernels*. Using this kind of kernels in classification tasks is interesting for a double reason: first, it enables us to better measure the dissimilarity between two strings by considering more possible structural distortions than the previously mentioned string kernels (*i.e.* spectrum, mismatch, subsequence kernels); second, it allows us to take advantage of powerful learning algorithms such as the support vector machines (SVM) to leave the constrained context of nearest-neighbor classifiers that are usually used

with the edit distance [8]. For these reasons, string edit kernels have seen an increasing amount of interest in recent years [8, 9, 10]. It is important to note that these kernel functions usually assume that the edit distance is negative definite. Indeed, to guarantee the convergence of the learning algorithm (*e.g.* SVM), this condition is required to ensure that the resulting kernel functions are valid kernels. However, in [9], the authors proved that the edit distance over a non-trivial alphabet Σ is not negative definite. This constitutes a theoretical drawback of the state of the art string edit kernels. Moreover, from a practical point of view, they usually use a standard version of the edit distance, *i.e.* the costs of the edit operations are a priori fixed (often to the same value). This is actually the case when no background knowledge is available. Parallel to this work on edit kernels, a recent line of research has investigated the ways to automatically learn the edit parameters from a learning sample in the form of state machines [11, 12, 13, 14, 15]. These edit parameters can then be used in neighborhood-based classifiers. Given a pair of strings (x, x'), the output of these models is not an edit cost anymore, but rather an edit probability to change x into x' by edit operations. It is important to note that these edit parameters no more allow us to obtain a true distance (the symmetry and reflexivity properties are often violated) that prevents us from using them in standard edit kernels such as those presented in [8, 9, 10]. To overcome this problem, we show in this paper how one can derive a new positive definite string edit kernel from these learned edit probabilities. Indeed, a third possibility to design string kernels is to exploit **probability distributions issued from stochastic models** as done in the Fisher kernel or its variants [16]. For instance, as shown by Haussler [17], the joint probability $p(x, x')$ of emitting two strings x and x' under a pair-Hidden Markov Model (pair-HMM) is a valid kernel that can be used to obtain by convolution new valid string kernels. In this paper, we show how to exploit edit probabilities that have been learned in the form of conditional state machines to create a new performing string edit kernel. We show that the computation of our new kernel requires to perform the intersection of two probabilistic automata, that is a special case of composition of stochastic transducers, as shown in [9] for designing *rational kernels.*

The rest of this article is organized as follows: Section 2 is devoted to the presentation of the state of the art string edit kernels. After having introduced in Section 3 our new string edit kernel, we briefly recall in Section 4 the main available approaches dealing with the learning of the edit parameters. Section 5 is devoted to the computation of our kernel based on a

specific case of composition of stochastic transducers. Finally, we present a large experimental study in Section 6, comparing our new learned edit kernel with the state of the art string kernels. We show that our approach provides significant improvements on a pattern recognition task.

2. Related work on string edit kernels

A common method used to define a string edit kernel from the standard edit distance e_d is the following [9, 10]:

$$k(x,x') = e^{-t \cdot e_d(x,x')}, \forall x, x' \in \Sigma^*, \tag{1}$$

where t is a positive real value. However, as mentioned in [9], there exists $t > 0$ for which this kernel is not positive definite, that does not guarantee the convergence of training for learning algorithms such as SVM. In [10], the authors show that t can be experimentally tuned according to the training data, but this remains a difficult task. Moreover, note that a slight modification of t can result in large differences of performance (see Section 6).

In [8], Neuhaus and Bunke present the following string edit kernel assuming the symmetry of the edit distance:

$$k(x,x') = k_{x_0}(x,x') = \frac{1}{2}\left(e_d(x,x_0)^2 + e_d(x_0,x')^2 - e_d(x,x')^2\right), \tag{2}$$

where x_0 is called a *zero string*. This kernel describes a measure of the squared distance from string x to x_0 and from x_0 to x' in relation to the squared distance from x to x'. Neuhaus and Bunke show that two more complex kernels can be obtained by selecting a set I of zero strings and combining the resulting kernels in a so-called *sum kernel* $k_I^+(x,x')$ and *product kernel* $k_I^*(x,x')$:

$$k_I^+(x,x') = \sum_{x_0 \in I} k_{x_0}(x,x'), \tag{3}$$

$$k_I^*(x,x') = \prod_{x_0 \in I} k_{x_0}(x,x'). \tag{4}$$

In these previous two edit kernels, the set I of zero strings plays an important role. In their experiments, the authors employ an iterative greedy selection strategy based on a validation set. Their experimental study shows

that this edit kernel outperforms a nearest-neighbor classifier using the standard edit distance. However, it is compared neither to other edit kernels nor to the state of the art string kernels.

In [6], Saigo et al. present a string alignment kernel specifically dedicated to detect *remote* homologies in biological sequences. This kernel, computed from a matrix of substitution between amino acids, is called LA (Local Alignment) kernel. It can be derived from Eq.1, where all the possible local alignments π for changing x into x' are taken into account and where an alignment score $s(x, x', \pi)$ is used instead of the edit distance e_d:

$$k_{LA}(x, x') = \sum_{\pi} e^{t \cdot s(x,x',\pi)}, \tag{5}$$

where t is a parameter and $s(x, x', \pi)$ is the corresponding score of π and defined as follows:

$$s(x, x', \pi) = \sum_{a,b} n_{a,b}(x, x', \pi) \cdot S(a, b) - n_{g_d}(x, x', \pi) \cdot g_d - n_{g_e}(x, x', \pi) \cdot g_e, \tag{6}$$

where $n_{a,b}(x, x', \pi)$ is the number of times that symbol a is aligned with letter b, $S(a, b)$ is the substitution score between a and b, g_d and g_e (and their corresponding number of occurrences $n_{g_d}(x, x', \pi)$ and $n_{g_e}(x, x', \pi)$) are two parameters dealing respectively with the opening and extension of gaps.

As mentioned in the introduction, another way to use the kernel described in Eq.1 is to compute it from edit probability distributions [10]. Indeed, the edit process can be viewed as a sequence of probabilistic events. In that way, the edit distance between x and x' can be computed from the summation of the negative logarithms of the transformation probabilities, as follows:

$$e_d(x, x') = -\sum_{i} \log p(x'_i|x_i), \forall x_i, x'_i \in \Sigma \cup \{\lambda\} \tag{7}$$

where λ is the empty symbol, $p(x'_i|x_i)$ is the probability to change the symbol x_i into x'_i at the ith operation of the optimal script changing x into x'. Plugging Eq.7 in Eq.1, we get the following edit kernel presented by Li & Jiang [10]:

$$k(x, x') = \left(\prod_{i} p(x'_i|x_i) \right)^t. \tag{8}$$

As mentioned in [10], this kernel assumes that $p(x'_i|x_i) = p(x_i|x'_i)$ which may not always be true. To guarantee the symmetry property, a solution consists of interpreting the edit distance as:

$$e_d(x, x') = -\frac{1}{2}\left(\sum_i \log p(x'_i|x_i) + \sum_i \log p(x_i|x'_i)\right).$$

Beyond the fact that our previous remark on the parameter t still holds (how t can be tuned to satisfy the property of positive definiteness?), defining a relevant value for $p(x'_i|x_i)$ is a difficult problem. In specific domains, some background knowledge is available allowing the use of such kernels. For instance, in molecular biology, some edit matrices have become standards to deal with amino acids (*e.g.* PAM [18] and BLOSUM [19]). These matrices describe similarity scores for amino acids and can be transformed (see [10]) to be used as relevant estimates of $p(x'_i|x_i)$. However, in many other domains, such an information is not available. A solution to overcome this problem is to automatically learn those edit probabilities from learning examples. We will present some approaches that deal with this problem in Section 4.

3. New edit distance-based kernel

Before that, let us introduce our new string edit kernel, which is a specific case of the marginalized kernels as described in [20]. Let $p(x, x', h)$ be the probability of observing jointly a hidden variable $h \in \mathcal{H}$ and two visible strings $x, x' \in \Sigma^*$. $p(x, x')$ can be obtained by marginalizing, *i.e.* summing over all variables $h \in \mathcal{H}$, the probability $p(x, x', h)$, such that:

$$p(x, x') = \sum_{h \in \mathcal{H}} p(x, x', h) = \sum_{h \in \mathcal{H}} p(x, x'|h) \cdot p(h).$$

A marginalized kernel computes this probability making the assumption that x and x' are conditionally independent given h:

$$k(x, x') = \sum_{h \in \mathcal{H}} p(x|h) \cdot p(x'|h) \cdot p(h). \tag{9}$$

Note that the computation of this kernel is possible since it is assumed that $\mathcal{H}$ is a *finite set*. Let us now suppose that $p(h|x)$ is known instead of $p(x|h)$. Then, as described in [16], we can use the following marginalized kernel:

$$k(x,x') = \sum_{h \in \mathcal{H}} p(h|x) \cdot p(h|x') \cdot K_z(z,z'), \tag{10}$$

where $K_z(z,z')$ is the joint kernel depending on combined variables $z = (x,h)$ and $z' = (x',h)$.

An interesting way to exploit kernel of Eq.10 as a *string edit kernel* is to use the following features:

- the finite set $\mathcal{H}$ of variables h is replaced by the *infinite set* of possible strings $y \in \Sigma^*$,
- as $p(y|x)$, we use $p_e(y|x)$ which is the conditional probability to transform a string x into a string y by using *edit operations*,
- the Dirac kernel (that returns 1, $\forall(z,z')$) is used as kernel $K_z(z,z')$.

Therefore, we obtain a new string edit kernel:

$$k(x,x') = \sum_{y \in \Sigma^*} p_e(y|x) \cdot p_e(y|x'), \tag{11}$$

which is positive definite as it corresponds to the dot product in the feature space defined by the following mapping: $\Phi : x \rightarrow \{p_e(y|x)\}_{y \in \Sigma^*}$. However, the practical use of this edit kernel raises two crucial questions:

1. Is it possible to learn a unique model that provides an estimate of the probability $p_e(y|x)$ for any pair of strings (x,y)?
2. If so, how can we compute from such a model the infinite sum over $y \in \Sigma^*$?

While recent papers have already answered the first question (see the next section for a survey), the reply to the second one is the matter of the rest of the article.

4. How to learn the edit parameters?

Let us recall that the edit distance between two strings x and y (built from an alphabet Σ) is the cost of the best sequence of edit operations that changes x into y. Typical edit operations are symbol deletion, insertion and substitution, and to each of them is assigned a cost. Since tuning these costs

can constitute a difficult task in many applications, supervised learning has been used during the last decade for learning the edit parameters.

In the context of the alignment of biological sequences, Saigo et al. [21] optimized from a learning set the BLOSUM substitution matrix by classical *gradient descent*. In order to control the optimization procedure, they designed an objective function that measures how well the local alignment kernel discriminates homologs from non-homologs. The learned substitution matrix $S(a, b)$ can then be incorporated in the LA kernel already presented in Eq.5. In order to avoid to call on positive and negative examples, another solution to learn the edit parameters is to use a probabilistic framework and control the optimization process by resorting to statistical constraints. In this context, that will be used in the rest of this paper, a widespread approach uses the *maximum likelihood* paradigm to learn probabilistic state machines (*e.g.* probabilistic automata, stochastic transducers, pair-hidden markov models) that can be used to model edit probability distributions. When there is no reason that the cost of a given edit operation changes according to the context where the operation occurs, memoryless machines (*i.e.* that contain only one state) are sufficient and very efficient from an accuracy and algorithmic point of view. In this case, training boils down to learning a single matrix of $|\Sigma \cup \{\lambda\}|^2$ parameters, *i.e.* a probability for each possible edit operation. A pioneer work that aimed to learn a memoryless transducer for modeling the edit probabilities has been presented by Ristadt and Yianilos in [12]. This generative model takes the form of a one state machine whose parameters (*i.e.* the edit probabilities) are learned by using an Expectation Maximization (EM)-based algorithm [22]. Using a *forward* procedure, it is then possible to deduce the joint edit probability $p_e(x, y)$ from the probabilities of the successive edit operations for changing x into y. However, to perform a classification task, one often requires $p_e(y|x)$ rather than $p_e(x, y)$. A solution consists in computing $p_e(y|x) = \frac{p_e(x,y)}{p(x)}$, but it is common knowledge that this can induce a statistical bias. To overcome this drawback, a relevant strategy consists in directly learning a conditional model (that allows an unbiased computation of $p_e(y|x)$ [23]) by adapting the maximization step of the EM algorithm, as done in [14]. In this context, the learned parameters are *conditional* edit probabilities. The authors of [14] show on a handwritten digit recognition task that such a discriminative model outperforms the generative one of Ristad and Yianilos.

Note that many applications can be treated by such memoryless models,

such as the correction of typing errors made with a computer keyboard, the recognition of images represented with Freeman codes, or in recognition of musical pieces [24].

However, learning a single matrix of edit costs can be viewed as insufficient in some applications, particularly when the operation plays a part more or less important in the string transformation according to its location. This is the case in molecular biology, where the edit operation on a symbol can depend on its presence in a transcription factor binding site. To deal with such situations, *non-memoryless* approaches have been proposed in the literature in the form of probabilistic state machines that are able to take into account the *string context*. They are mainly based on pair-HMM [11, 25], probabilistic deterministic automata (PDFA) [15], or stochastic transducers [26]. The string context is taken into account in each state by a specific statistical distribution over the edit operations. However, these models are generative and so do not allow the integration of constraints on the input string x. To overcome this limitation, two recent approaches have been presented to learn discriminative non memoryless edit models. In [13], McCallum et al. adapted the conditional random fields to the finite-state string edit distance. Note that this model, unlike the others, requires the use of positive and negative examples of matches of strings. More recently, in [27], the authors presented *constrained state machines* dedicated to the learning of edit distance satisfying domain constraints. The main difficulty of this approach is to have such background knowledge that can be expressed in the form of constraints.

Whatever the learning method we use, we have shown in this section that it is possible to learn a state machine T which models a distribution over the edit operations. As we noted before, memoryless transducers are suitable machines to deal with pattern recognition problems for whose the location of the edit operation does not play an important role. This is the case for the recognition of digits represented with Freeman codes as shown in [14]. Since we will use in this paper the same experimental framework to assess the relevance of our new edit kernel, we will consider in the rest of this paper that a memoryless transducer T is learned with the learning algorithm proposed in [14]. To ensure the self consistency of our paper, Annex 1 presents the main features of this learning algorithm that returns the machine T.

Such a machine can then be used to compute the joint or conditional edit probability between two strings. In the next section, we show how we can use T to compute the infinite sum of our string edit kernel. Rather

than computing each $p_e(y|x), \forall y \in \Sigma^*$, we show that one can deduce from T two probabilistic automata modeling respectively the conditional distribution $p_e(y|x)$ and $p_e(y|x')$, given x and x'. Then, drawing our inspiration from the rational kernels [9], we show that the infinite sum of Eq.11 can be efficiently calculated by calling on a composition of these two probabilistic automata.

5. How to compute the edit kernel over Σ^*?

While the original marginalized kernel of Eq.9 assumes that $\mathcal{H}$ is a finite set of variables [16], our string edit kernel computes an infinite sum over Σ^*, such that $k(x, x') = \sum_{y \in \Sigma^*} p_e(y|x) \cdot p_e(y|x')$. We show in the following that (i) given two strings x and x', $p_e(y|x)$ and $p_e(y|x')$ can be represented in the form of two probabilistic automata, (ii) the product $p_e(y|x) \cdot p_e(y|x')$ can be performed by intersecting the languages represented by those automata and (iii) the infinite sum over Σ^* can be computed by resorting to algebraic methods.

5.1. Definitions and Notations

Definition 1. *A weighted finite-state transducer (WFT) is an 8-tuple* $T = (\Sigma, \Delta, Q, S, F, w, \tau, \rho)$ *where* Σ *is the input alphabet,* Δ *the output alphabet,* Q *a finite set of states,* $S \subseteq Q$ *the set of initial states,* $F \subseteq Q$ *the set of final states,* $w : Q \times Q \times (\Sigma \cup \{\lambda\}) \times (\Delta \cup \{\lambda\}) \to \Re^+$ *the transition weight function,* $\tau : S \to \Re^+$ *the initial weight function, and* $\rho : F \to \Re^+$ *the final weight function. For convenience, we denote* $w(q_1, q_2, x_i, y_j)$ *by* $w_{q_1 \to q_2}(x_i, y_j)$*.*

Definition 2. *A joint probabilistic finite-state transducer (jPFT) is a WFT* $J = (\Sigma, \Delta, Q, S, F, w, \tau, \rho)$ *which defines a joint probability distribution over the pairs of strings* $\{(x, y) \in \Sigma^* \times \Delta^*\}$*. A jPFT satisfies the following three constraints:*

i) $\sum_{i \in S} \tau(i) = 1$;

ii) $\sum_{f \in F} \rho(f) = 1$;

iii)
$$\forall q_1 \in Q : \sum_{q_2 \in Q, x_i \in \Sigma \cup \{\lambda\}, y_j \in \Delta \cup \{\lambda\}} w_{q_1 \to q_2}(x_i, y_j) = 1.$$

Definition 3. *A conditional probabilistic finite-state transducer (cPFT) is a WFT $C = (\Sigma, \Delta, Q, S, F, w, \tau, \rho)$ which defines a conditional probability distribution over the output strings $y \in \Delta^*$ given an input string x. We denote the transition $w_{q_1 \to q_2}(x_i, y_j)$ in the conditional form $w_{q_1 \to q_2}(y_j|x_i)$. A cPFT satisfies the same first two constraints as those of a jPFT and the following third constraint (see [14] for a proof):*

$$\forall q_1 \in Q, \forall x_i \in \Sigma : \sum_{q_2 \in Q, y_j \in \Delta \cup \{\lambda\}} (w_{q_1 \to q_2}(y_j|x_i) + w_{q_1 \to q_2}(y_j|\lambda)) = 1.$$

In the following, since our string edit kernel is based on conditional edit probabilities, we will assume that a $cPFT$ has already been learned by one the previously mentioned methods (see Annex 1 for the details of a specific EM-based algorithm). Note that this transducer $cPFT$ is learned only one time and then is used to compute our edit kernel for any pair of strings. For instance, a memoryless $cPFT$ is described in Fig.1, where $\Sigma = \Delta = \{a, b\}$ and Q, S and F are composed of only one state of label 0. Note that if a generative learning model would have been used to learn the edit parameters, the resulting $jPFT$ could be a posteriori renormalized into a $cPFT$.

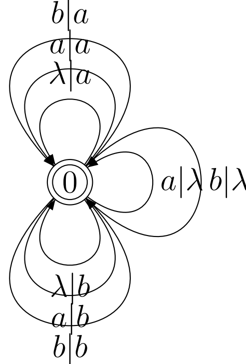

Figure 1: Memoryless $cPFT$. To each transition is assigned an edit probability (not shown here for the sake of legibility) which can be used to compute the edit conditional probability of any pair of strings.

5.2. Modeling $p_e(y|x)$ and $p_e(y|x')$ by probabilistic automata

Since our kernel $k(x, x')$ depends on two observable strings x and x', it is possible to represent in the form of probabilistic state machines the distributions $p_e(y|x)$ and $p_e(y|x')$, where only y is a hidden variable.

Given a *cPFT* C modeling the edit probabilities and a string x, we can define a new *cPFT* denoted by $C|x$ that models $p_e(y|x)$ while being driven by the specific observable string x.

Definition 4. *Let* $C = (\Sigma, \Delta, Q, S, F, w, \tau, \rho)$ *be a cPFT that models* $p_e(y|x)$, $\forall y \in \Delta^*, \forall x \in \Sigma^*$. *We define* $C|x$ *as a cPFT that models* $p_e(y|x), \forall y \in \Delta^*$ *but for a specific observable* $x \in \Sigma^*$. $C|x = (\Sigma, \Delta, Q', S', F', w', \tau', \rho')$ *with:*

- $Q' = \{[x]_i\} \times Q$ *where* $[x]_i$ *is the prefix of length* i *of* x *(note that* $[x]_0 = \lambda$*); In other words,* Q' *is a finite set of states labeled by the current prefix of* x *and its corresponding state during its parsing in of* C[1].
- $S' = \{(\lambda, q)\}$ *where* $q \in S$;
- $\forall q \in S, \tau'((\lambda, q)) = \tau(q)$;
- $F' = \{(x, q)\}$ *where* $q \in F$;
- $\forall q \in F, \rho'((x, q)) = \rho(q)$;
- *the following two rules are applied to define the transition weight function:*
  - $\forall a \in \Delta, \forall q_1, q_2 \in Q, w'_{([x]_i,q_1)\to([x]_{i+1},q_2)}(a|x_{i+1}) = w_{q_1\to q_2}(a|x_{i+1})$;
  - $\forall a \in \Delta, \forall q_1, q_2 \in Q, w'_{([x]_i,q_1)\to([x]_i,q_2)}(a|\lambda) = w_{q_1\to q_2}(a|\lambda)$.

As an example, given two strings $x = a$ and $x' = ab$, Fig.2(a) and Fig.2(b) show the *cPFT* $C|a$ and $C|ab$ constructed from the memoryless transducer C of Fig.1. Roughly speaking, $C|a$ and $C|ab$ model the output languages that can be respectively generated from x and x'. Therefore, from these state machines, we can generate output strings and compute the conditional edit probabilities $p_e(y|x)$ and $p_e(y|x')$ for any string $y \in \Delta^*$. Note that since we are in an edit-based framework, the cycles outgoing from each state model the different possible insertions before and after the reading of an input symbol.

Since the construction of $C|x$ and $C|x'$ are driven by the parsing of x and x' in the transducer C, we can omit the input alphabet Σ. Therefore,

[1] This specific notation is required to deal with non memoryless *cPFT*.

a transducer $C|x = (\Sigma, \Delta, Q, S, F, w, \tau, \rho)$ can be reduced to a finite-state automaton $A|x = (\Delta, Q, S, F, w', \tau, \rho)$. The transitions of $A|x$ are derived from w in the following way: $w'_{q_1 \to q_2}(a) = w_{q_1 \to q_2}(a|b), \forall b \in \Sigma \cup \{\lambda\}, \forall a \in \Delta \cup \{\lambda\}, \forall q_1, q_2 \in Q$. For example, Fig.3(a) and 3(a) are the resulting automata deduced from the *cPFT* $C|a$ and $C|ab$ of Fig.2(a) and 2(b).

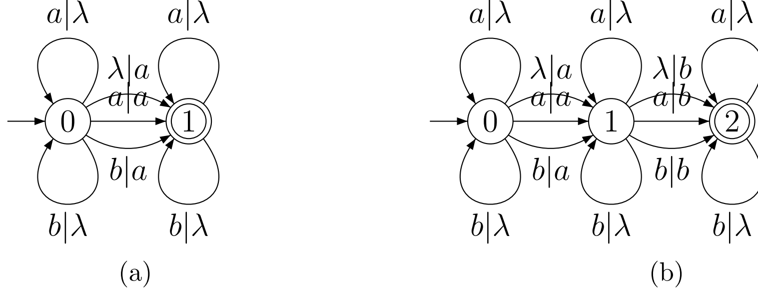

Figure 2: On the left: a *cPFT* $C|a$ that models the output distribution conditionally to an input string $x = a$; note that for the sake of legibility, state 0 stands for $(\lambda, 0)$, and state 1 for $(a, 0)$. On the right: a *cPFT* $C|ab$ that models the output distribution given $x' = ab$, here again, 0 stands for $(\lambda, 0)$, 1 for $(a, 0)$, and 2 for $(ab, 0)$.

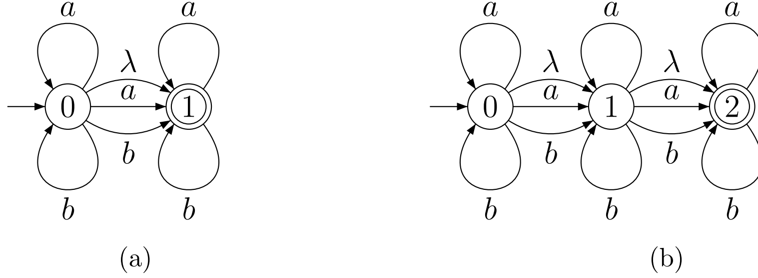

Figure 3: The transducers $C|a$ and $C|ab$ represented in the form of automata.

5.3. Computing the product $p_e(y|x) \cdot p_e(y|x')$

The next step for computing our kernel $k(x, x')$ requires the calculation of the product $p_e(y|x) \cdot p_e(y|x')$. This can be performed by modeling the language that describes the intersection of the automata modeling $p_e(y|x)$ and $p_e(y|x')$. This intersection can be obtained by performing a *composition* of transducers as described by Cortes et al. in [9]. As mentioned by the authors, composition is a fundamental operation on weighted transducers that

can be used to create complex weighted transducers from simpler ones. In this context, we can note that the intersection of two probabilistic automata (such as those of Fig.3(a) and 3(b)) is a special case of composition where the input and output label of transitions are identical. This intersection takes the form of a probabilistic automaton as defined below.

Definition 5. *Let C be a cPFT modeling conditional edit probabilities. Let x and x' be two strings of Σ^*. Let $A|x = (\Delta, Q, S, F, w, \tau, \rho)$ and $A|x' = (\Delta, Q', S', F', w', \tau', \rho')$ be the automata deduced from C given the observable strings x and x'. We define the intersection of $A|x$ and $A|x'$ as the automaton $A_{x,x'} = (\Delta, Q^A, S^A, F^A, w^A, \tau^A, \rho^A)$ such that:*

- $Q^A = Q \times Q'$;
- $S^A = \{(q, q')\}$ *with* $q \in S$ *and* $q' \in S'$;
- $F^A = \{(q, q')\}$ *with* $q \in F$ *and* $q' \in F'$;
- $w^A_{(q_1,q'_1)\to(q_2,q'_2)}(a) = w_{q_1\to q_2}(a) \cdot w'_{q'_1\to q'_2}(a)$;
- $\tau^A((q, q')) = \tau(q) \cdot \tau(q')$
- $\rho^A((q, q')) = \rho(q) \cdot \rho(q')$

Fig.4 describes the intersection automaton of automata of Fig.3(a) and 3(b).

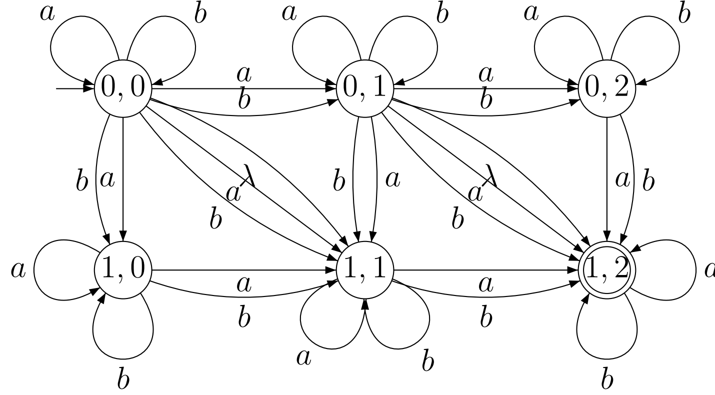

Figure 4: Resulting automaton modeling the intersection of the automata of Fig.3(a) and 3(b).

Let us now describe how this resulting automaton can be used to compute the infinite sum of our kernel over Σ^*.

5.4. Computing the sum over Σ^*

To simplify the notations, let $p(z) = p_e(y|x) \cdot p_e(y|x')$ be the distribution modeled by an intersection automaton $A = \{\Sigma, Q, S, F, w, \tau, \rho\}$. Let $\Sigma = \{a_1, \dots, a_{|\Sigma|}\}$ be the alphabet.

Let M_{a_k} be the square matrix defined over $Q \times Q$ and providing the probabilities $M_{a_k}(q_i, q_j) = w_{q_i \rightarrow q_j}(a_k), \forall a_k \in \Sigma$. In other words, $M_{a_k}(q_i, q_j)$ is the probability that the symbol $a_k \in \Sigma$ is emitted by the transition going from state $q_i \in Q$ to state $q_j \in Q$ in A. In this context, given a string $z = z_1...z_t$, $p(z)$ can be rewritten as follows:

$$p(z) = p(z_1...z_t) = \boldsymbol{\tau}^T M_{z_1}...M_{z_t}\boldsymbol{\rho} = \boldsymbol{\tau}^T M_z \boldsymbol{\rho}, \qquad (12)$$

where $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$ are two vectors of dimension $|Q|$ whose components are the values returned by the weight function τ ($\forall q \in S$) and ρ ($\forall q \in F$), and 0 otherwise, and where $M_z = M_{z_1}...M_{z_t}$.

¿From 12, we can deduce that:

$$\sum_{z \in \Sigma^*} p(z) = \sum_{z \in \Sigma^*} \boldsymbol{\tau}^T M_z \boldsymbol{\rho}. \qquad (13)$$

To take into account all the possible strings $z \in \Sigma^*$, Eq.13 can be rewritten according to the size of the string z.

$$\sum_{z \in \Sigma^*} p(z) = \sum_{i=0}^{\infty} \boldsymbol{\tau}^T (M_{a_1} + M_{a_2} + ... + M_{a_{|\Sigma|}})^i \boldsymbol{\rho} = \boldsymbol{\tau}^T \sum_{i=0}^{\infty} M^i \boldsymbol{\rho}, \qquad (14)$$

where $M = M_{a_1} + M_{a_2} + ... + M_{a_{|\Sigma|}}$. Let us replace $\sum_{i=0}^{\infty} M^i$ by B. Therefore,

$$B = I + M + M^2 + M^3 + \dots, \qquad (15)$$

where I is the identity matrix. Multiplying B by M we get

$$MB = M + M^2 + M^3 + \dots. \qquad (16)$$

Subtracting Eq.15 from Eq.16, we get:

$$B - MB = I \Leftrightarrow B = (I - M)^{-1}. \qquad (17)$$

Plugging Eq.17 in Eq.14, we get our kernel:

$$k(x, x') = \sum_{y \in \Sigma^*} p_e(y|x) \cdot p_e(y|x') = \boldsymbol{\tau}^T \sum_{i=0}^{\infty} M^i \boldsymbol{\rho} = \boldsymbol{\tau}^T (I - M)^{-1} \boldsymbol{\rho}. \qquad (18)$$

5.5. Tractability

Given two strings x and x', we deal here with the complexity of computing $k(x, x')$. As we just wrote, this involves the computation of an infinite sum computed by solving a matrix inversion.

Let us denote C the $cPFT$ that models the edit probabilities, and let be t the number of states of C. We denote by n the length of the string x and by m the length of the string x'.

First, recall that in the case of the classical edit distance with non learned costs, the complexity is $n \cdot m$ for the calculation of $d_e(x, x')$.

The weighted automaton $C|x$ describing $p_e(y|x)$ has $(n+1) \cdot t$ states, and $C|x'$ describing $p_e(y|x')$ has $(m+1) \cdot t$ states (*e.g.* see Fig.3). Thus the matrix $(I - M)$ is of dimension $(n + 1) \cdot (m + 1) \cdot t^2$. The computation cost of each element of this matrix linearly depends on the alphabet size $|\Sigma|$. Therefore, the computation cost of the entire matrix is $n^2 \cdot m^2 \cdot t^4 \cdot |\Sigma|$. Since M is triangular, the matrix inversion $(I - M)^{-1}$ can be performed by back substitution, avoiding the complications of general Gaussian elimination. The cost of the inversion is of order of the square of the matrix dimension, that is $n^2 \cdot m^2 \cdot t^4$. This leads to a cost of

$$n^2 \cdot m^2 \cdot t^4 \cdot |\Sigma|.$$

Note the factor t^4 stands for the size of C that models edit probabilities. In the case of memoryless models (that will be used in the experimental study of Section 6), we have $t = 1$ and thus a cost reduced to

$$n^2 \cdot m^2 \cdot |\Sigma|.$$

Therefore, in the case of a conditional memoryless transducer, and for small alphabet sizes, the computation cost of our edit kernel is “only” the square of that of the standard edit distance.

Despite the advantage of triangular matrices, the algorithmic complexity remains a problem in the case of learning processes with long strings and/or large alphabet sizes. To overcome this problem, we can efficiently approximate our kernel $k(x, x')$ by computing a finite sum only considering the strings y that belong to the learning sample. Therefore, we get

$$\hat{k}(x, x') = \sum_{y \in S} p_e(y|x) \cdot p_e(y|x')$$

where S is the learning sample. Since the computation of each probability $p_e(y|x)$ requires a cost of $n+|y|$, the average cost of each kernel computation is

$$(n+m+\overline{|y|}) \cdot |S|$$

where $\overline{|y|}$ is the average length of the learning strings.

In conclusion, even if our kernel is a priori rather costly from a complexity point of view, it has the advantage to be usable from any transducer modeling edit probabilities, and its calculation remains reasonable. Moreover, the analysis of this kernel must also take into account the gain in accuracy it implies in a pattern recognition task. This is the aim of the next section.

6. Experiments

6.1. Database and Experimental Setup

To assess the relevance of using learned edit distances in the design of a new string edit kernel, we carried out some experiments. We studied the behavior of our approach on the well known NIST Special Database 3 of the National Institute of Standards and Technology, describing a set of handwritten characters.

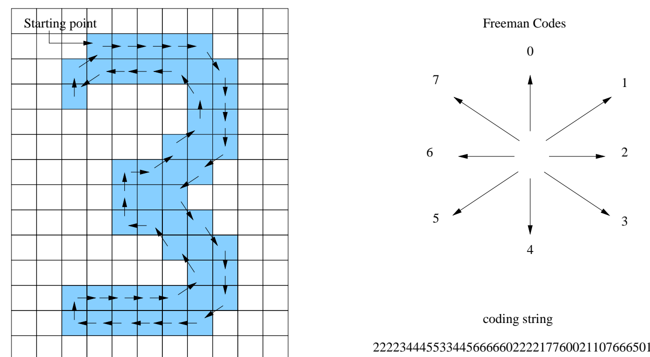

Figure 5: A digit and its string representation.

We focus on handwritten digits consisting of 128 $\times$ 128 bitmaps images. We use a learning set of about 8000 digits and a test sample of 2000 instances. Each instance is represented by a string. To code a given digit, the algorithm scans the bitmap left to right and starting from the top until encountering the first point. Then, the algorithm builds the string following the border of

the character until it returns to the starting pixel. The string is constructed considering the succession of Freeman codes as described in Fig.5.

To compare our approach with other string kernels, we used SVM-Light 6.02 [28]. In order to deal with our multi-class task, we learn 10 binary SVM models M_i. The model M_i, $\forall i \in \{0, ..., 9\}$, is learned from a positive class composed by all the handwritten characters labeled i and a negative class grouping all the other digits. Then, the class of an unknown example x is determined as follows: we compute for each model M_i the margin $\gamma_{M_i}(x)$ and the class of x is given by $\text{argmax}_i\, \gamma_{M_i}(x)$. Obviously, a high positive value of margin $\gamma_{M_i}(x)$ represents a high probability for x to be of class i.

6.2. Edit Kernels versus Edit Distance

As done by Neuhaus and Bunke [8], our first objective was to show the interest of our edit kernel in comparison with the edit distance used in a nearest-neighbor algorithm. To achieve this task, we first used a standard edit distance d_l with costs of all edit operations (deletion, insertion and substitution) set to 1. On the other hand, we used the SEDiL platform [29] to learn a matrix of edit probabilities from which one can deduce conditional edit probabilities $p_e(y|x)$ and compute a so-called stochastic edit distance

$$d_e(x, y) = -\log p_e(y|x).$$

We assessed the performance of a 1-nearest neighbor algorithm using both edit distances d_l and d_e, and compared them with our new string edit kernel (using SVM-Light 6.02) on a test set composed of 2000 instances. Note that the conditional probabilities $p_e(y|x)$ required in our edit kernel are the same as those used in $d_e(x, y)$. Results are presented in Fig.6 according to an increasing number of learning examples (from 100 to 8000).

We can make the following remarks:

- First, learning a stochastic edit distance d_e on this classification task leads to better results than using the standard edit distance d_l. Indeed, whatever the size of the learning set we use, the accuracy computed from d_e is always higher.
- Second, our string edit kernel outperforms not only the standard edit distance d_l, but also the stochastic edit distance d_e. This clearly shows the usefulness of our string edit kernel to take advantage of powerful classifiers such as SVMs. To estimate the statistical significance

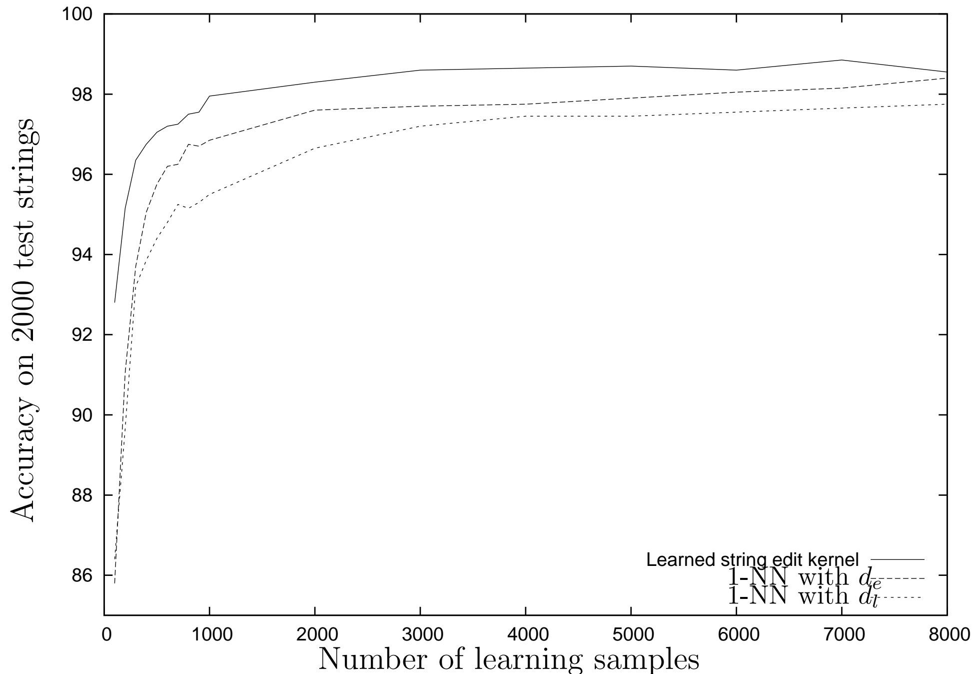

Figure 6: Comparison of our edit kernel with edit distances on a handwritten digit recognition task.

of these results, comparisons between the three approaches were performed using a Student paired-t test. Table 1 presents the so-called p-values obtained while comparing our kernel with d_e and d_l. Using a risk of 5%, a p-value of less than 0.05 means that the difference is significant in favor of our kernel (in bold font in the table). We can note that this is always the case except for the two learning samples of size 2000 and 6000.

Learning set size	1000	2000	3000	4000	5000	6000	7000	8000
EK vs learned ED	**1E-02**	6E-02	**2E-02**	**2E-02**	**2E-02**	9E-02	**3E-02**	**3E-01**
EK vs standard ED	**6E-06**	**4E-04**	**1E-03**	**3E-03**	**2E-03**	**8E-03**	**2E-03**	**3E-02**

Table 1: Statistical comparison between our edit kernel (EK) and a (learned or standard) Edit Distance (ED). Using a risk of 5%, a p-value of less than 5E-02 means that the difference is significant in favor of our kernel

However, these previous remarks are not sufficient to prove the relevance of our edit kernel. Indeed, as presented in Section 2, other string kernels have

been proposed in the literature that can also take advantage of SVMs. How does our edit kernel behave in comparison with those state of the art string kernels? This is the matter of the next section.

6.3. Comparison with State of the Art String Kernels

In this second series of experiments, we compared our learned string edit kernel with other string kernels presented in Section 1 and 2, that is, Li & Jiang kernel [10], Neuhaus & Bunke kernel [8], spectrum kernel [2] and subsequence kernel [4][2]. Even if these last two kernels have already been introduced in section 1, let us briefly present them and precise values of parameters we used in our experiments. The spectrum kernel considers a feature space of dimension $\mathbb{N}^k$ where k denotes the number of substrings over Σ^* of length smaller than a parameter p. In this space, each example x is represented by the vector of the number of occurrences of substrings. Thus, the spectrum kernel of x and y is trivially computed using the inner product between the two string representations. In our experiments, the parameter p has been fixed to 2.

The subsequences gap weighted kernel extends the spectrum kernel considering subsequences instead of substrings. Thus, this kernel is able to take into account long term dependencies information using sets of letters with gaps. The same parameter p is used to limit the size of subsequences. Moreover, a weight λ is defined in order to give less importance to subsequences with large gaps. In our experiments, λ has been set to 2 which leads in our case to the best results with this kernel.

Fig.7 shows the results we obtained with the different mentioned kernels. We can first note that the best results are obtained with edit distance based kernels. As we did before, Table 2 presents the p-values of the Student paired-t test. Except the kernel of Li & Jiang, our edit kernel significantly outperforms all the other string kernels. Second, even if their behaviors are quite similar, we can note that our kernel almost always outperforms that of Li & Jiang (even if the difference is not statistically significant). It is also important to note in this analysis that the parameter t required in the Li & Jiang kernel is very difficult to tune. In [10], the authors indicated that the best results were obtained with a value $t = 0.00195$. In order to optimally

[2]We did not use the LA kernel which is too specific (it searches for remote homologies between biological sequences by achieving local alignments) to be easily compared in this series of experiments with the state of the art string kernels.

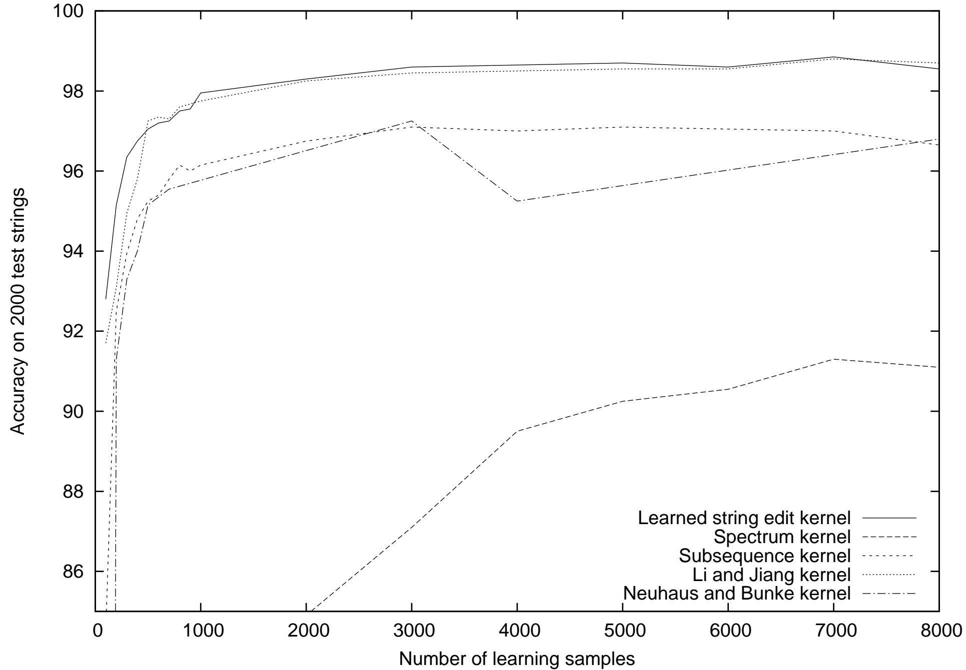

Figure 7: Comparison of our edit kernel with the state of the art edit kernels on a handwritten digit recognition task.

tune this parameter, we tested many values of t (from $t = 0$ to $t = 0.2$). Figure 8 shows that the best performances are obtained on 2000 test strings with a value of 0.02, while using $t > 0.1$ leads to a dramatic drop of the accuracy. Therefore, we used $t = 0.02$ throughout our comparison.

Learning set size	1000	2000	3000	4000	5000	6000	7000	8000
EK vs spectrum	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
EK vs subsequence	**4E-04**	**8E-04**	**5E-04**	**2E-04**	**2E-04**	**4E-04**	**2E-05**	**4E-05**
EK vs Li & Jiang	3E-01	4E-01	3E-01	3E-01	3E-01	4E-01	4E-01	7E-01
EK vs Neuhaus & B.	**6E-06**	**6E-06**	**1E-03**	**2E-10**	**6E-09**	**4E-08**	**4E-07**	**1E-04**

Table 2: Statistical comparison between our edit kernel (EK) and the state of art string kernels. Using a risk of 5%, a p-value of less than 5E-02 means that the difference is significant in favor of our kernel.

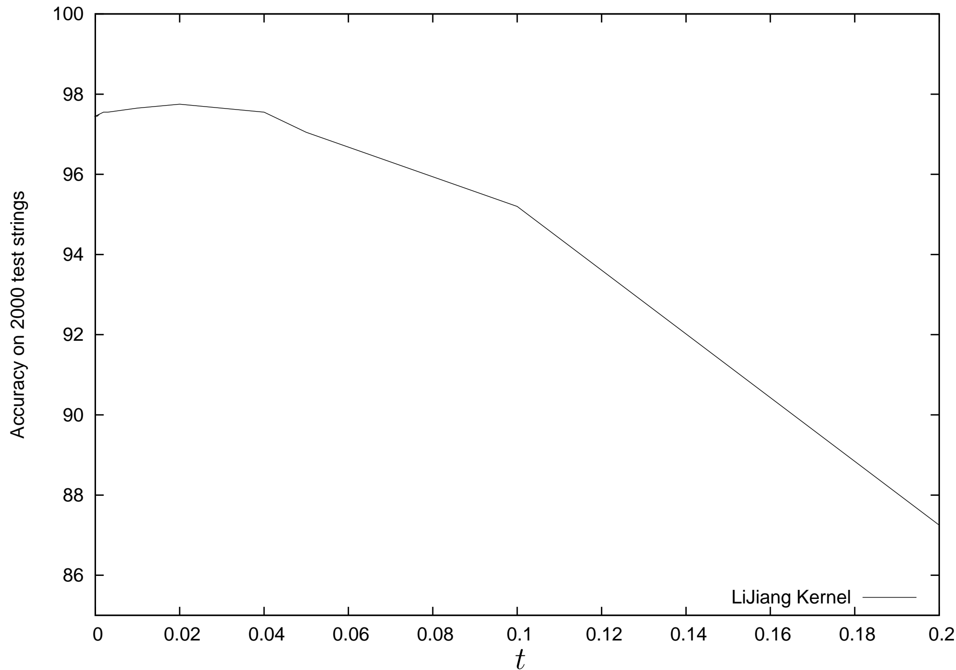

Figure 8: Tuning of the parameter t in Li & Jiang kernel.

7. Conclusion

The common way to use the edit distance in classification tasks consists in exploiting it in a classical k-Nearest-Neighbor algorithm. Recently, machine learning techniques have been used to automatically learn the probabilities of the edit operations to capture background knowledge and so improve the performance of the edit distance. On the other hand, in order to take advantage of the widely used framework of Support Vector Machines, recent works have been done to derive edit kernels from the edit distance. In this article, we suggested to embed the advantages of both approaches by designing a learned string edit kernel. We experimentally showed that such a strategy allows significant improvements in terms of classification accuracy.

A first perspective of our work is to improve the algorithmic complexity of our kernel. A crucial point in its calculation concerns the size of the product automaton allowing the computation of $p_e(y|x) \cdot p_e(y|x')$. To reduce this size, a possible solution would consist in simplifying the original conditional transducers $C|x$ and $C|x'$ by only considering the most probable transitions and states. A simplification of $C|x$ and $C|x'$ would have a direct impact on

the dimension of the matrix we have to invert.

A second natural perspective of this work relies on the extension of our method to the elaboration of *tree* edit kernels. Indeed, some recent papers dealing with the learning of tree edit distance in the form of stochastic models [30, 31] would allow us to generalize our method to the learning of more complex structured data-based edit kernels.

Annex 1

This annex describes the way to learn the parameters of a conditional memoryless transducer as proposed in [14].

Let us suppose that we have a learning set S of pairs of strings (x, y) and for the sake of simplicity that the input and the output alphabets are the same, noted Σ. Let Σ^* be the set of all finite strings over Σ. Let $x \in \Sigma^*$ be an arbitrary string of length $|x|$ over the alphabet Σ. 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).

Let c be the conditional probability function that returns for any edit operation $(b|a)$ the probability to output the symbol b given an input letter a. The aim of this annex is to show how we can automatically learn the function c from the learning sample S. The different values $c(b|a), \forall a \in \Sigma \cup \{\lambda\}, b \in \Sigma \cup \{\lambda\}$ represent the parameters of the memoryless machine T. These parameters are trained using an EM-based algorithm that calls on two auxiliary functions called *forward* and *backward*.

The conditional probability $p : \Sigma^* \times \Sigma^* \to [0, 1]$ of the string y given an input one was a x (noted $p(y|x)$) can be recursively computed by means of an auxiliary function (forward) $\alpha : \Sigma^* \times \Sigma^* \to \mathbb{R}^+$ as:

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

Using $\alpha(y|x)$, we get,

$$p(y|x) = \alpha(y|x) \cdot \gamma,$$

where γ is the probability of the termination symbol of a string (note that γ and $c(\lambda|\lambda)$ are synonyms). In a symmetric way, $p(y|x)$ can be recursively computed by means of an auxiliary function (backward) $\beta : \Sigma^* \times \Sigma^* \to \mathbb{R}^+$ as:

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

And we can deduce that,

$$p(y|x) = \beta(y|x) \cdot \gamma.$$

Both functions can be computed in $O(|x| \cdot |y|)$ time using a dynamic programming technique and will be used in the following to learn the current function c. In this model a probability distribution is assigned conditionally to each input string. Then

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

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

$$\gamma > 0, c(b|a), c(b|\lambda), c(\lambda|a) \geq 0 \quad \forall a \in \Sigma, b \in \Sigma \tag{19}$$

$$\sum_{b \in \Sigma} c(b|\lambda) + \sum_{b \in \Sigma} c(b|a) + c(\lambda|a) = 1 \quad \forall a \in \Sigma \tag{20}$$

$$\sum_{b \in \Sigma} c(b|\lambda) + \gamma = 1 \tag{21}$$

The expectation-maximization algorithm [22] can be used in order the find the optimal parameters of the function c. Given an auxiliary $(|\Sigma| + 1) \times$

$(|\Sigma|+1)$ matrix δ, the **expectation step** aims at computing the values of δ as follows: $\forall a \in \Sigma, b \in \Sigma$,

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

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

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

$$\delta(\lambda|\lambda) = \sum_{(x,y)\in S} \frac{\alpha(y|x)\cdot\gamma}{p(y|x)} = |S|.$$

The **maximization step** allows us to iteratively deduce the current edit costs.

$$c(b|\lambda) = \frac{\delta(b|\lambda)}{N} \text{ (insertion)}$$

$$\gamma = \frac{N - N(\lambda)}{N} \text{ (termination symbol)}$$

$$c(b|a) = \frac{\delta(b|a)}{N(a)} \cdot \frac{N - N(\lambda)}{N} \text{ (substitution)}$$

$$c(\lambda|a) = \frac{\delta(\lambda|a)}{N(a)} \cdot \frac{N - N(\lambda)}{N} \text{ (deletion)}$$

where:

$$N = \sum_{\substack{a\in\Sigma\cup\{\lambda\}\\ b\in\Sigma\cup\{\lambda\}}} \delta(b|a) \qquad N(\lambda) = \sum_{b\in\Sigma} \delta(b|\lambda) \qquad N(a) = \sum_{b\in\Sigma\cup\{\lambda\}} \delta(b|a)$$

References

[1] B. Scholkopf, A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, Cambridge, MA, USA, 2001.

[2] C. Leslie, E. Eskin, W. S. Noble, The spectrum kernel: a string kernel for svm protein classification, in: Proceedings of the Pacific Symposium on Biocomputing, 2002, pp. 564–575.

[3] C. Leslie, E. Eskin, A. Cohen, J. Weston, W. S. Noble, Mismatch string kernels for discriminative protein classification, Bioinformatics 20 (4) (2004) 467–476.

[4] H. Lodhi, C. Saunders, N. Cristianini, C. Watkins, B. Scholkopf, Text classification using string kernels, Journal of Machine Learning Research 2 (2002) 563–569.

[5] S. Boisvert, M. Marchand, F. Laviolette, J. Corbeil, Hiv-1 coreceptor usage prediction without multiple alignments: an application of string kernels., Retrovirology 5 (1) (2008) 110.

[6] H. Saigo, J.-P. Vert, N. Ueda, T. Akutsu, Protein homology detection using string alignment kernels, Bioinformatics 20 (11) (2004) 1682–1689.

[7] R. Wagner, M. Fischer, The string-to-string correction problem, Journal of the ACM (JACM) 21 (1974) 168–173.

[8] M. Neuhaus, H. Bunke, Edit distance-based kernel functions for structural pattern classification, Pattern Recognition 39 (10) (2006) 1852–1863.

[9] C. Cortes, P. Haffner, M. Mohri, Rational kernels: Theory and algorithms, Journal of Machine Learning Research 5 (2004) 1035–1062.

[10] H. Li, T. Jiang, A class of edit kernels for svms to predict translation initiation sites in eukaryotic mrnas, in: Proceedings of the eighth annual international conference on Research in computational molecular biology (RECOMB'04), ACM, 2004, pp. 262–271.

[11] R. Durbin, S. Eddy, A. krogh, G. Mitchison, Biological sequence analysis, Cambridge University Press, 1998, probalistic models of protein and nucleic acids.

[12] S. Ristad, P. Yianilos, Learning string-edit distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (5) (1998) 522–532.

[13] A. McCallum, K. Bellare, P. Pereira, A conditional random field for disciminatively-trained finite-state string edit distance, in: Proceedings of the 21th Conference on Uncertainty in Artificial Intelligence (UAI'2005), 2005, pp. 388–400.

[14] J. Oncina, M. Sebban, Learning stochastic edit distance: application in handwritten character recognition, Journal of Pattern Recognition 39 (9) (2006) 1575–1587.

[15] M. Bernard, J.-C. Janodet, M. Sebban, A discriminative model of stochastic edit distance in the form of a conditional transducer, in: 8th International Colloquium on Grammatical Inference (ICGI'06), LNCS 4201, 2006, pp. 240–252.

[16] K. Tsuda, T. Kin, K. Asai, Marginalized kernels for biological sequences, Bioinformatics 18 (90001) (2002) 268–275.

[17] D. Haussler, Convolution kernels on discrete structures, Technical Report, University of California, Santa Cruz.

[18] M. O. Dayhoff, R. M. Schwartz, B. C. Orcutt, A model of evolutionary change in proteins, in: Atlas of protein sequence and structure, Vol. 5, M. O. Dayhoff, National biomedical research foundation, Washington DC., 1978, pp. 345–358.

[19] S. Henikoff, J. G. Henikoff, Amino acid substitution matrices from protein blocks, in: National Academy of Sciences of USA, 1992, pp. 89(22):10915–10919.

[20] H. Kashima, K. Tsuda, A. Inokuchi, Marginalized kernels between labeled graphs, in: Proceedings of the Twentieth International Conference on Machine Learning, AAAI Press, 2003, pp. 321–328.

[21] S. Hiroto, V. Jean-Philippe, A. Tatsuya, Optimizing amino acid substitution matrices with a local alignment kernel, BMC Bioinformatics 7 (2006) 246.

[22] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the em algorithm, Journal of the Royal Statistical Society. Series B (Methodological) 39 (1) (1977) 1–38.

[23] G. Bouchard, B. Triggs, The tradeoff between generative and discriminative classifiers, in: IASC International Symposium on Computational Statistics (COMPSTAT), Prague, 2004, pp. 721–728.

[24] A. Habrard, M. Inesta, D. Rizo, M. Sebban, Melody recognition with learned edit distances, in: Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2008 and SPR 2008, 2008, pp. 86–96.

[25] M. Bilenko, R. J. Mooney, Adaptive duplicate detection using learnable string similarity measures, in: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003), 2003, pp. 39–48.

[26] J. Eisner, Parameter estimation for probabilistic finite-state transducers, in: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL'02), Morristown, NJ, USA, 2001, pp. 1–8. doi:http://dx.doi.org/10.3115/1073083.1073085.

[27] L. Boyer, A. Habrard, F. Muhlenbach, M. Sebban, Learning string edit similarities using constrained finite state machines, in: Proceedings of the 10th french conference in Machine Learning (CAp-2008), 2008, pp. 37–52.

[28] T. Joachims, Making large-scale SVM learning practical, in: B. Schlkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods - Support Vector Learning, MIT Press, 1999, pp. 169–184.

[29] L. Boyer, Y. Esposito, A. Habrard, J. Oncina, M. Sebban, Sedil: Software for edit distance learning, in: Proceedings of the 19th European Conference on Machine Learning (ECML 2008), LNCS 5212, 2008, pp. 672–677.

[30] M. Bernard, L. Boyer, A. Habrard, M. Sebban, Learning probabilistic models of tree edit distance, Pattern Recognition 41 (8) (2008) 2611–2629.

[31] L. Boyer, A. Habrard, M. Sebban, Learning metrics between tree structured data: Application to image recognition, in: Proceedings of the 18th European Conference on Machine Learning (ECML 2007), 2007, pp. 54–66.