Stochastic Contextual Edit Distance and Probabilistic FSTs

Ryan Cotterell and **Nanyun Peng** and **Jason Eisner**
Department of Computer Science, Johns Hopkins University
{ryan.cotterell,npeng1,jason}@cs.jhu.edu

Abstract

String similarity is most often measured by weighted or unweighted edit distance $d(x, y)$. Ristad and Yianilos (1998) defined *stochastic* edit distance—a probability distribution $p(y \mid x)$ whose parameters can be trained from data. We generalize this so that the probability of choosing each edit operation can depend on contextual features. We show how to construct and train a probabilistic finite-state transducer that computes our stochastic contextual edit distance. To illustrate the improvement from conditioning on context, we model typos found in social media text.

1 Introduction

Many problems in natural language processing can be viewed as stochastically mapping one string to another: e.g., transliteration, pronunciation modeling, phonology, morphology, spelling correction, and text normalization. Ristad and Yianilos (1998) describe how to train the parameters of a stochastic editing process that moves through the input string x from left to right, transforming it into the output string y. In this paper we generalize this process so that the edit probabilities are conditioned on input and output context.

We further show how to model the conditional distribution $p(y \mid x)$ as a probabilistic finite-state transducer (PFST), which can be easily combined with other transducers or grammars for particular applications. We contrast our *probabilistic* transducers with the more general framework of *weighted* finite-state transducers (WFST), explaining why our restriction provides computational advantages when reasoning about unknown strings.

Constructing the finite-state transducer is tricky, so we give the explicit construction for use by others. We describe how to train its parameters when the contextual edit probabilities are given by a log-linear model. We provide a library for training both PFSTs and WFSTs that works with OpenFST (Allauzen et al., 2007), and we illustrate its use with simple experiments on typos, which demonstrate the benefit of context.

2 Stochastic Contextual Edit Distance

Our goal is to define a family of probability distributions $p_\theta(y \mid x)$, where $x \in \Sigma_x^*$ and $y \in \Sigma_y^*$ are input and output strings over finite alphabets Σ_x and Σ_y, and θ is a parameter vector.

Let x_i denote the i^{th} character of x. If $i < 1$ or $i > |x|$, then x_i is the distinguished symbol BOS or EOS ("beginning/end of string"). Let $x_{i:j}$ denote the $(j - i)$-character substring $x_{i+1}x_{i+2}\cdots x_j$.

Consider a stochastic edit process that reads input string x while writing output string y. Having read the prefix $x_{0:i}$ and written the prefix $y_{0:j}$, the process must stochastically choose one of the following $2|\Sigma_y| + 1$ edit operations:

- DELETE: Read x_{i+1} but write nothing.
- INSERT(t) for some $t \in \Sigma_y$: Write $y_{j+1} = t$ without reading anything.
- SUBST(t) for some $t \in \Sigma_y$: Read x_{i+1} and write $y_{j+1} = t$. Note that the traditional COPY operation is obtained as SUBST(x_{i+1}).

In the special case where $x_{i+1} = \text{EOS}$, the choices are instead INSERT(t) and HALT (where the latter may be viewed as copying the EOS symbol).

The probability of each edit operation depends on θ and is conditioned on the left input context $C_1 = x_{(i-N_1):i}$, the right input context $C_2 = x_{i:(i+N_2)}$, and the left output context $C_3 = y_{(j-N_3):j}$, where the constants $N_1, N_2, N_3 \geq 0$ specify the model's context window sizes.[1] Note that the probability cannot be conditioned on right output context because those characters have not yet been chosen. Ordinary stochastic edit distance (Ristad and Yianilos, 1998) is simply the case $(N_1, N_2, N_3) = (0, 1, 0)$, while Bouchard-Côté et al. (2007) used roughly $(1, 2, 0)$.

Now $p_\theta(y \mid x)$ is the probability that this process will write y as it reads a given x. This is the total probability (given x) of *all* latent edit operation sequences that write y. In general there are exponentially many such sequences, each implying a different alignment of y to x.

[1] If $N_2 = 0$, so that we do not condition on x_{i+1}, we must still condition on whether $x_{i+1} = \text{EOS}$ (a single bit). We gloss over special handling for $N_2 = 0$; but it is in our code.

This model is reminiscent of conditional models in MT that perform stepwise generation of one string or structure from another—e.g., string alignment models with contextual features (Cherry and Lin, 2003; Liu et al., 2005; Dyer et al., 2013), or tree transducers (Knight and Graehl, 2005).

3 Probabilistic FSTs

We will construct a **probabilistic finite-state transducer (PFST)** that compactly models $p_\theta(y \mid x)$ for all (x, y) pairs.[2] Then various computations with this distribution can be reduced to standard finite-state computations that efficiently employ dynamic programming over the structure of the PFST, and the PFST can be easily combined with other finite-state distributions and functions (Mohri, 1997; Eisner, 2001).

A PFST is a two-tape generalization of the well-known nondeterministic finite-state acceptor. It is a finite directed multigraph where each arc is labeled with an input in $\Sigma_\mathrm{x} \cup \{\epsilon\}$, an output in $\Sigma_\mathrm{y} \cup \{\epsilon\}$, and a probability in $[0, 1]$. (ϵ is the empty string.) Each state (i.e., vertex) has a halt probability in $[0, 1]$, and there is a single initial state q_I. Each path from q_I to a final state q_F has

- an input string x, given by the concatenation of its arcs' input labels;
- an output string y, given similarly;
- a probability, given by the product of its arcs' probabilities and the halt probability of q_F.

We define $p(y \mid x)$ as the total probability of all paths having input x and output y. In our application, a PFST path corresponds to an edit sequence that reads x and writes y. The path's probability is the probability of that edit sequence given x.

We must take care to ensure that for any $x \in \Sigma_\mathrm{x}^*$, the total probability of all paths accepting x is 1, so that $p_\theta(y \mid x)$ is truly a conditional probability distribution. This is guaranteed by the following sufficient conditions (we omit the proof for space), which do not seem to appear in previous literature:

- For each state q and each symbol $b \in \Sigma_\mathrm{x}$, the arcs from q with input label b or ϵ must have total probability of 1. (These are the available choices if the next input character is b.)
- For each state q, the halt action and the arcs from q with input label ϵ must have total probability of 1. (These are the available choices if there is no next input character.)
- Every state q must be co-accessible, i.e., there must be a path of probability > 0 from q to some q_F. (Otherwise, the PFST could lose some probability mass to infinite paths. The canonical case of this involves an loop $q \to q$ with input label ϵ and probability 1.)

We take the first two conditions to be part of the *definition* of a PFST. The final condition requires our PFST to be "tight" in the same sense as a PCFG (Chi and Geman, 1998), although the tightness conditions for a PCFG are more complex. In section 7, we discuss the costs and benefits of PFSTs relative to other options.

4 The Contextual Edit PFST

We now define a PFST topology that concisely captures the contextual edit process of section 2. We are given the alphabets $\Sigma_\mathrm{x}, \Sigma_\mathrm{y}$ and the context window sizes $N_1, N_2, N_3 \geq 0$.

For each possible context triple $C = (C_1, C_2, C_3)$ as defined in section 2, we construct an **edit state** q_C whose outgoing arcs correspond to the possible edit operations in that context.

One might expect that the $\textsc{subst}(t)$ edit operation that reads $s = x_{i+1}$ and writes $t = y_{j+1}$ would correspond to an arc with s, t as its input and output labels. However, we give a more efficient design where in the course of reaching q_C, the PFST has *already* read s and indeed the entire right input context $C_2 = x_{i:(i+N_2)}$. So our PFST's input and output actions are "out of sync": its read head is N_2 characters ahead of its write head. When the edit process of section 2 has read $x_{0:i}$ and written $y_{0:j}$, our PFST implementation will actually have read $x_{0:(i+N_2)}$ and written $y_{0:j}$.

This design eliminates the need for nondeterministic guessing (of the right context $x_{i:(i+N_2)}$) to determine the edit probability. The PFST's state is fully determined by the characters that it has read and written so far. This makes left-to-right composition in section 5 efficient.

A fragment of our construction is illustrated in Figure 1. An edit state q_C has the following outgoing **edit arcs**, each of which corresponds to an edit operation that replaces some $s \in \Sigma_x \cup \{\epsilon\}$ with some $t \in \Sigma_y \cup \{\epsilon\}$:

[2]Several authors have given recipes for finite-state transducers that perform a *single* contextual edit operation (Kaplan and Kay, 1994; Mohri and Sproat, 1996; Gerdemann and van Noord, 1999). Such "rewrite rules" can be individually more expressive than our simple edit operations of section 2; but it is unclear how to train a cascade of them to model $p(y \mid x)$.

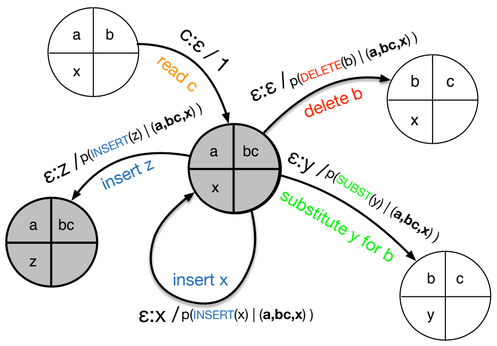
a
b
x
c:ε / 1
read c
ε:ε / p(DELETE(b) | (a,bc,x))
delete b
b
c
x
ε:z / p(INSERT(z) | (a,bc,x))
insert z
a
bc
x
ε:y / p(SUBST(y) | (a,bc,x))
substitute y for b
a
bc
z
insert x
ε:x / p(INSERT(x) | (a,bc,x))
b
c
y

Figure 1: A fragment of a PFST with $N_1 = 1, N_2 = 2, N_3 = 1$. Edit states are shaded. A state q_C is drawn with left and right input contexts C_1, C_2 in the left and right upper quadrants, and left output context C_3 in the left lower quadrant. Each arc is labeled with input:output / probability.

- A single arc with probability $p(\text{DELETE} \mid C)$ (here $s = (C_2)_1, t = \epsilon$)
- For each $t \in \Sigma_y$, an arc with probability $p(\text{INSERT}(t) \mid C)$ (here $s = \epsilon$)
- For each $t \in \Sigma_y$, an arc with probability $p(\text{SUBST}(t) \mid C)$ (here $s = (C_2)_1$)

Each edit arc is labeled with input ϵ (because s has *already* been read) and output t. The arc leads from q_C to $q_{C'}$, a state that moves s and t into the left contexts: $C'_1 = \text{suffix}(C_1 s, N_1)$, $C'_2 = \text{suffix}(C_2, N_2 - |s|)$, $C'_3 = \text{suffix}(C_3 t, N_3)$.

Section 2 mentions that the end of x requires special handling. An edit state q_C whose $C_2 = \text{EOS}^{N_2}$ only has outgoing $\text{INSERT}(t)$ arcs, and has a halt probability of $p(\text{HALT} \mid C)$. The halt probability at all other states is 0.

We must also build some **non-edit states** of the form q_C where $|C_2| < N_2$. Such a state does not have the full N_2 characters of lookahead that are needed to determine the conditional probability of an edit. Its outgoing arcs deterministically read a new character into the right input context. For each $s \in \Sigma_x$, we have an arc of probability 1 from q_C to $q_{C'}$ where $C' = (C_1, C_2 s, C_3)$, labeled with input s and output ϵ. Following such arcs from q_C will reach an edit state after $N_2 - |C_2|$ steps.

The initial state q_I with $I = (\text{BOS}^{N_1}, \epsilon, \text{BOS}^{N_3})$ is a non-edit state. Other non-edit states are constructed only when they are reachable from another state. In particular, a DELETE or SUBST arc always transitions to a non-edit state, since it consumes one of the lookahead characters.

5 Computational Complexity

We summarize some useful facts without proof. For fixed alphabets Σ_x and Σ_y, our final PFST, T, has $O(|\Sigma_x|^{N_1+N_2}|\Sigma_y|^{N_3})$ states and $O(|\Sigma_x|^{N_1+N_2}|\Sigma_y|^{N_3+1})$ arcs. Composing this T with deterministic FSAs takes time linear in the size of the *result*, using a left-to-right, on-the-fly implementation of the composition operator $\circ$.

Given strings x and y, we can compute $p_\theta(y \mid x)$ as the total probability of all paths in $x \circ T \circ y$. This acyclic weighted FST has $O(|x| \cdot |y|)$ states and arcs. It takes only $O(|x| \cdot |y|)$ time to construct it and sum up its paths by dynamic programming, just as in other edit distance algorithms.

Given only x, taking the output language of $x \circ T$ yields the full distribution $p_\theta(y \mid x)$ as a cyclic PFSA with $O(|x| \cdot \Sigma_y^{N_3})$ states and $O(|x| \cdot \Sigma_y^{N_3+1})$ arcs. Finding its most probable path (i.e., most probable aligned y) takes time $O(|\text{arcs}| \log |\text{states}|)$, while computing every arc's expected number of traversals under $p(y \mid x)$ takes time $O(|\text{arcs}| \cdot |\text{states}|)$.[3]

$p_\theta(y \mid x)$ may be used as a noisy channel model. Given a language model $p(x)$ represented as a PFSA X, $X \circ T$ gives $p(x, y)$ for all x, y. In the case of an n-gram language model with $n \le N_1 + N_2$, this composition is efficient: it merely reweights the arcs of T. We use Bayes' Theorem to reconstruct x from observed y: $X \circ T \circ y$ gives $p(x, y)$ (proportional to $p(x \mid y)$) for each x. This weighted FSA has $O(\Sigma_x^{N_1+N_2} \cdot |y|)$ states and arcs.

6 Parameterization and Training

While the parameters θ could be trained via various objective functions, it is particularly efficient to compute the gradient of *conditional log-likelihood*, $\sum_k \log p_\theta(y_k \mid x_k)$, given a sample of pairs (x_k, y_k). This is a non-convex objective function because of the latent x-to-y alignments: we do not observe *which* path transduced x_k to y_k. Recall from section 5 that these possible paths are represented by the small weighted FSA $x_k \circ T \circ y_k$.

Now, a path's probability is defined by multiplying the contextual probabilities of edit operations e. As suggested by Berg-Kirkpatrick et al. (2010), we model these steps using a conditional log-linear model, $p_\theta(e \mid C) \overset{\text{def}}{=} \frac{1}{Z_C} \exp\left(\theta \cdot \vec{f}(C, e)\right)$.

[3]Speedups: In both runtimes, a factor of $|x|$ can be eliminated from $|\text{states}|$ by first decomposing $x \circ T$ into its $O(|x|)$ strongly connected components. And the $|\text{states}|$ factor in the second runtime is unnecessary in practice, as just the first few iterations of conjugate gradient are enough to achieve good approximate convergence when solving the sparse linear system that defines the forward probabilities in the cyclic PFSA.

To increase $\log p_\theta(y_k \mid x_k)$, we must raise the probability of the edits e that were used to transduce x_k to y_k, relative to competing edits from the same contexts C. This means raising $\theta \cdot f(C, e)$ and/or lowering Z_C. Thus, $\log p_\theta(y_k \mid x_k)$ depends only on the probabilities of edit arcs in T that appear in $x_k \circ T \circ y_k$, and the competing edit arcs from the same edit states q_C.

The gradient $\nabla_\theta \log p_\theta(y_k \mid x_k)$ takes the form

$$\sum_{C,e} \mathrm{c}(C, e) \left[\vec{f}(C, e) - \sum_{e'} p_\theta(e' \mid C) \vec{f}(C, e') \right]$$

where $\mathrm{c}(C, e)$ is the *expected* number of times that e was chosen in context C given (x_k, y_k). (That can be found by the forward-backward algorithm on $x_k \circ T \circ y_k$.) So the gradient adds up the differences between observed and expected feature vectors at contexts C, where contexts are weighted by how many times they were likely encountered.

In practice, it is efficient to hold the counts $\mathrm{c}(C, e)$ constant over several gradient steps, since this amortizes the work of computing them. This can be viewed as a generalized EM algorithm that imputes the hidden paths (giving c) at the "E" step and improves their probability at the "M" step.

Algorithm 1 provides the training pseudocode.

Algorithm 1 Training a PFST T_θ by EM.

```
1:  while not converged do
2:      reset all counts to 0                          ▷ begin the "E step"
3:      for $k \leftarrow 1$ to $K$ do                  ▷ loop over training data
4:          $M = x_k \circ T_\theta \circ y_k$          ▷ small acyclic WFST
5:          $\vec{\alpha}$ = FORWARD-ALGORITHM($M$)
6:          $\vec{\beta}$ = BACKWARD-ALGORITHM($M$)
7:          for arc $A \in M$, from state $q \to q'$ do
8:              if $A$ was derived from an arc in $T_\theta$
                    representing edit $e$, from edit state $q_C$, then
9:                  $\mathrm{c}(C, e)$ += $\alpha_q \cdot \mathrm{prob}(A) \cdot \beta_{q'} / \beta_{q_I}$
10:     $\theta \leftarrow$ L-BFGS($\theta$, EVAL, max_iters=5) ▷ the "M step"
11: function EVAL($\theta$)        ▷ objective function & its gradient
12:     $F \leftarrow 0$; $\nabla F \leftarrow 0$
13:     for context $C$ such that $(\exists e)\mathrm{c}(C, e) > 0$ do
14:         $count \leftarrow 0$; $expected \leftarrow 0$; $Z_C \leftarrow 0$
15:         for possible edits $e$ in context $C$ do
16:             $F$ += $\mathrm{c}(C, e) \cdot (\theta \cdot \vec{f}(C, e))$
17:             $\nabla F$ += $\mathrm{c}(C, e) \cdot \vec{f}(C, e)$
18:             $count$ += $\mathrm{c}(C, e)$
19:             $expected$ += $\exp(\theta \cdot \vec{f}(C, e)) \cdot \vec{f}(C, e)$
20:             $Z_C$ += $\exp(\theta \cdot \vec{f}(C, e))$
21:         $F$ -= $count \cdot \log Z_C$; $\nabla F$ -= $count \cdot expected / Z_C$
22:     return $(F, \nabla F)$
```

7 PFSTs versus WFSTs

Our PFST model of $p(y \mid x)$ enforces a normalized probability distribution at each state. Dropping this requirement gives a **weighted FST (WFST)**, whose path weights $w(x, y)$ can be globally normalized (divided by a constant Z_x) to obtain probabilities $p(y \mid x)$. WFST models of contextual edits were studied by Dreyer et al. (2008).

PFSTs and WFSTs are respectively related to MEMMs (McCallum et al., 2000) and CRFs (Lafferty et al., 2001). They gain added power from hidden states and ϵ transitions (although to permit a *finite*-state encoding, they condition on x in a more restricted way than MEMMs and CRFs).

WFSTs are likely to beat PFSTs as linguistic models,[4] just as CRFs beat MEMMs (Klein and Manning, 2002). A WFST's advantage is that the probability of an edit can be *indirectly* affected by the weights of other edits at a distance. Also, one could construct WFSTs where an edit's weight *directly* considers local right output context C_4.

So why are we interested in PFSTs? Because they do not require computing a separate normalizing contant Z_x for every x. This makes it computationally tractable to use them in settings where x is uncertain because it is unobserved, partially observed (e.g., lacks syllable boundaries), or noisily observed. E.g., at the end of section 5, X represented an uncertain x. So unlike WFSTs, PFSTs are usable as the conditional distributions in noisy channel models, channel cascades, and Bayesian networks. In future we plan to measure their modeling disadvantage and attempt to mitigate it.

PFSTs are also more efficient to train under conditional likelihood. It is faster to compute the gradient (and fewer steps seem to be required in practice), since we only have to raise the probabilities of *arcs* in $x_k \circ T \circ y_k$ relative to competing arcs in $x_k \circ T$. We visit at most $|x_k| \cdot |y_k| \cdot |\Sigma_{\mathrm{y}}|$ arcs. By contrast, training a WFST must raise the probability of the *paths* in $x_k \circ T \circ y_k$ relative to the *infinitely many* competing paths in $x_k \circ T$. This requires summing around cycles in $x_k \circ T$, and requires visiting all of its $|x_k| \cdot |\Sigma_{\mathrm{y}}|^{N_3+1}$ arcs.

8 Experiments

To demonstrate the utility of contextual edit transducers, we examine spelling errors in social media data. Models of spelling errors are useful in a variety of settings including spelling correction itself and phylogenetic models of string variation

[4]WFSTs can also use a simpler topology (Dreyer et al., 2008) while retaining determinism, since edits can be scored "in retrospect" after they have passed into the left context.

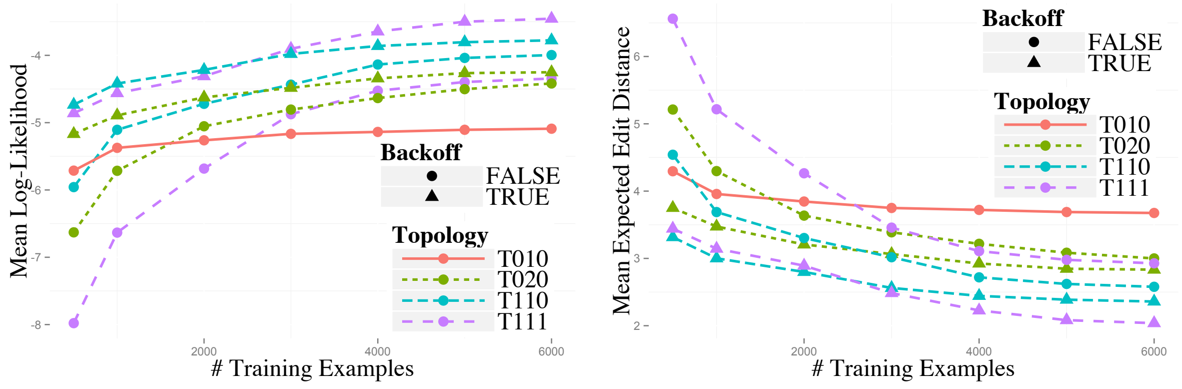

Figure 2: (a) Mean $\log p(y \mid x)$ for held-out test examples. (b) Mean expected edit distance (similarly).

(Mays et al., 1991; Church and Gale, 1991; Kukich, 1992; Andrews et al., 2014).

To eliminate experimental confounds, we use no dictionary or language model as one would in practice, but directly evaluate our ability to model $p(\text{correct} \mid \text{misspelled})$. Consider $(x_k, y_k) =$ (*feeel, feel*). Our model defines $p(y \mid x_k)$ for all y. Our training objective (section 6) tries to make this large for $y = y_k$. A *contextual* edit model learns here that $e \mapsto \epsilon$ is more likely in the context of *ee*.

We report on test data how much probability mass lands on the true y_k. We also report how much mass lands "near" y_k, by measuring the expected edit distance of the predicted y to the truth. Expected edit distance is defined as $\sum_y p_\theta(y \mid x_k) d(y, y_k)$ where $d(y, y_k)$ is the Levenshtein distance between two strings. It can be computed using standard finite-state algorithms (Mohri, 2003).

8.1 Data

We use an annotated corpus (Aramaki, 2010) of 50000 misspelled words x from tweets along with their corrections y. All examples have $d(x, y) = 1$ though we do not exploit this fact. We randomly selected 6000 training pairs and 100 test pairs. We regularized the objective by adding $\lambda \cdot ||\theta||_2^2$, where for each training condition, we chose λ by coarse grid search to maximize the conditional likelihood of 100 additional development pairs.

8.2 Context Windows and Edit Features

We considered four different settings for the context window sizes (N_1, N_2, N_3): (0,1,0)=stochastic edit distance, (1,1,0), (0,2,0), and (1,1,1).

Our log-linear edit model (section 6) includes a dedicated indicator feature for each contextual edit (C, e), allowing us to fit *any* conditional distribution $p(e \mid C)$. In our "backoff" setting, each (C, e) also has 13 binary backoff features that it shares with other (C', e'). So we have a total of 14 feature templates, which generate over a million features in our largest model. The shared features let us learn that certain *properties* of a contextual edit tend to raise or lower its probability (and the regularizer encourages such generalization).

Each contextual edit (C, e) can be characterized as a 5-tuple (s, t, C_1, C_2', C_3): it replaces $s \in \Sigma_x \cup \{\epsilon\}$ with $t \in \Sigma_y \cup \{\epsilon\}$ when s falls between C_1 and C_2' (so $C_2 = sC_2'$) and t is preceded by C_3. Then each of the 14 features of (C, e) indicates that a particular subset of this 5-tuple has a particular value. The subset always includes s, t, or both. It never includes C_1 or C_2' without s, and never includes C_3 without t.

8.3 Results

Figures 2a and 2b show the learning curves. We see that both metrics improve with more training data; with more context; and with backoff. With backoff, all of the contextual edit models substantially beat ordinary stochastic edit distance (at *all* training set sizes and especially at large sizes).

9 Conclusion

We have presented a trainable, featurizable model of *contextual* edit distance. Our main contribution is an efficient encoding of such a model as a tight PFST—that is, a WFST that is guaranteed to directly define conditional string probabilities without need for further normalization. We are releasing OpenFST-compatible code that can train both PFSTs and WFSTs (Cotterell and Renduchintala, 2014). We formally defined PFSTs, described their speed advantage at training time, and noted that they are crucial in settings where the input string is unknown. In future, we plan to deploy our PFSTs in such settings.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata*, pages 11–23. Springer.

Nicholas Andrews, Jason Eisner, and Mark Dredze. 2014. Robust entity clustering via phylogenetic inference. In *Proceedings of ACL*.

Eiji Aramaki. 2010. Typo corpus. Available at `http://luululu.com/tweet/#cr`, January.

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Proceedings of HLT-NAACL*, pages 582–590.

Alexandre Bouchard-Côté, Percy Liang, Thomas L. Griffiths, and Dan Klein. 2007. A probabilistic approach to language change. In *NIPS*.

Colin Cherry and Dekang Lin. 2003. A probability model to improve word alignment. In *Proceedings of ACL*, pages 88–95.

Zhiyi Chi and Stuart Geman. 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Kenneth W. Church and William A. Gale. 1991. Probability scoring for spelling correction. *Statistics and Computing*, 1(2):93–103.

Ryan Cotterell and Adithya Renduchintala. 2014. brezel: A library for training FSTs. Technical report, Johns Hopkins University.

Markus Dreyer, Jason R. Smith, and Jason Eisner. 2008. Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of EMNLP*, EMNLP '08, pages 1080–1089.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM Model 2. In *Proceedings of NAACL-HLT*, pages 644–648.

Jason Eisner. 2001. Expectation semirings: Flexible EM for learning finite-state transducers. In *Proceedings of the ESSLLI Workshop on Finite-State Methods in NLP*.

Dale Gerdemann and Gertjan van Noord. 1999. Transducers from rewrite rules with backreferences. In *Proceedings of EACL*.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Dan Klein and Christopher D. Manning. 2002. Conditional structure versus conditional estimation in NLP models. In *Proceedings of EMNLP*, pages 9–16.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Proc. of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.

Yang Liu, Qun Liu, and Shouxun Lin. 2005. Log-linear models for word alignment. In *Proceedings of ACL*, pages 459–466.

Eric Mays, Fred J. Damerau, and Robert L. Mercer. 1991. Context based spelling correction. *Information Processing & Management*, 27(5):517–522.

Andrew McCallum, Dayne Freitag, and Fernando Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of ICML*.

Mehryar Mohri and Richard Sproat. 1996. An efficient compiler for weighted rewrite rules. In *Proceedings of ACL*, pages 231–238.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Mehryar Mohri. 2003. Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(06):957–982.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532.