Smoothing probabilistic automata: an error-correcting approach

Pierre Dupont[1] and Juan-Carlos Amengual[2]

[1] EURISE, Université Jean Monnet
23, rue P. Michelon
42023 Saint-Etienne Cedex – France
pdupont@univ-st-etienne.fr
[2] Universidad Jaume I de Castellón
Campus de Riu Sec
12071 Castellón – Spain
jcamen@inf.uji.es

Abstract. In this paper we address the issue of smoothing the probability distribution defined by a probabilistic automaton. As inferring a probabilistic automaton is a statistical estimation problem, the usual data sparseness problem arises. We propose here the use of an error correcting technique for smoothing automata. This technique is based on a symbol dependent error model which guarantees that any possible string can be predicted with a non-zero probability. We detail how to define a consistent distribution after extending the original probabilistic automaton with error transitions. We show how to estimate the error model's free parameters from independent data. Experiments on the ATIS travel information task show a 48 % test set perplexity reduction on new data with respect to a simply smoothed version of the original automaton.

1 Introduction

The goal of learning a probabilistic deterministic finite automaton (PDFA) is to induce a DFA structure from data and estimate its constituent transition probabilities. As the structure itself constrains the probability distribution over the set of possible strings, the inference procedure can be considered to be a single problem of statistical estimation. Several learning algorithms for probabilistic automata have been proposed [16, 3, 15], but the smoothing issue has not been addressed. In particular, when probabilistic automata are used for modeling real data, as in the case of natural language interfaces, the usual problem of data sparseness arises. In other words only a few strings are actually observed in the training sample and many strings that could be observed receive a zero probability of being generated even after the generalization introduced by the inference algorithm.

Smoothing the probability distribution fundamentally requires us to discount a certain probability mass from the seen events and to distribute it over unseen events which would otherwise have a zero probability. Considering that a string with zero probability is a string for which there is no path between the initial state and an

accepting state in a probabilistic automaton[1], error-correcting techniques [2] can be used towards this end.

Error-correcting techniques extend automata to allow acceptance of, in principle, any string. Using error correction allows us to compute the probability of accepting the string with minimal error. Several criteria can be used to guide this process. For instance, we can look for the minimal number of editing operations necessary to accept a string. Alternatively, we can search for the accepting path of maximal probability in a probabilistic error-correcting parser. In the latter case, the error model parameters need to be estimated and possibly smoothed as well.

Definitions and notations are given in section 2.1. The ALERGIA algorithm, which will be used for PDFA inference, is briefly presented in section 2.2. The criterion for evaluating the quality of a PDFA, that is the perplexity computed on an independent test sample, is detailed in section 2.3. We present in section 3 our baseline smoothing technique using linear interpolation with a unigram model.

The formal definition of the proposed error-correcting model and the method for estimating its free parameters are fully described in section 4. Experiments on the ATIS task, a spoken language interface to a travel information database, were performed in order to assess the proposed smoothing techniques. The task is presented in section 5. Finally we show how error-correcting techniques improve the baseline perplexity. These experiments are detailed in section 6.

2 Preliminaries

In this section we detail the formal definition of a *probabilistic DFA* (PDFA). Next, we review briefly the ALERGIA algorithm which will be used in our experiments to infer PDFA. Finally we present the measure for estimating the quality of PDFA inference and smoothing.

2.1 Definitions

A PDFA is a 5-tuple $(Q, \Sigma, \delta, q_0, \gamma)$ in which Q is a finite set of *states*, Σ is a finite *alphabet*, δ is a *transition function*, i.e. a mapping from $Q\mathrm{x}\Sigma$ to Q, q_0 is the *initial state*, γ is the *next symbol probability function*, i.e. a mapping from $Q\mathrm{x}\Sigma \cup \{\#\}$ to $[0, 1]$. A special symbol $\#$, not belonging to the alphabet Σ, denotes *the end of string symbol*. Hence $\gamma(q, \#)$ represents the probability of ending the generation process in state q and q is an accepting state if $\gamma(q, \#) > 0$. The probability function must satisfy the following constraints:

$$\begin{array}{rl} \gamma(q,a) & = 0 \text{ , if } \delta(q,a) = \emptyset, \forall a \in \Sigma \\ \sum_{a \in \Sigma \cup \{\#\}} \gamma(q,a) & = 1 \text{ , } \forall q \in Q \end{array}$$

[1] We assume here that no existing transition in an automaton has a zero probability.

The probability $P_A(x)$ of *generating* a string $x = x_1 \ldots x_n$ from a PDFA $A = (Q, \Sigma, \delta, q_0, \gamma)$ is defined as

$$P_A(x) = \begin{cases} \left(\prod_{i=1}^{n} \gamma(q^i, x_i)\right) \gamma(q^n, \#) \\ \quad \text{if } \delta(q^i, x_i) \neq \emptyset \text{ with } q^{i+1} = \delta(q^i, x_i), \\ \quad \text{for } 1 \leq i < n \text{ and } q^1 = q_0 \\ 0, \quad \text{otherwise} \end{cases}$$

The language $L(A)$ generated by a PDFA A is made of all strings with non-zero probability:

$$L(A) = \{x \mid P_A(x) > 0\}$$

Our definition of probabilistic automaton is equivalent to a stochastic deterministic regular grammar used as a string generator. Thus, $\sum_{x \in \Sigma^*} P_A(x) = 1$. Note that some work on the learning of discrete distributions uses distributions defined on Σ^n (that is $\sum_{x \in \Sigma^n} P(x) = 1$, for any $n \geq 1$), instead of Σ^*.

Let I_+ denote a *positive sample*, *i.e.* a set of strings belonging to a probabilistic language we are trying to model. Let $PTA(I_+)$ denote the *prefix tree acceptor* built from a positive sample I_+. The prefix tree acceptor is an automaton that only accepts the strings in the sample and in which common prefixes are merged together resulting in a tree shaped automaton. Let $PPTA(I_+)$ denote the *probabilistic prefix tree acceptor*. It is the probabilistic extension of the $PTA(I_+)$ in which each transition has a probability proportional to the number of times it is used while generating, or equivalently parsing, the positive sample.

2.2 PDFA inference

Several inference algorithms for probabilistic automata have been proposed [16, 3, 15] but only Carrasco and Oncina's ALERGIA algorithm, a stochastic extension of the RPNI algorithm [14], is free from the restriction to the learning of acyclic automata. This algorithm has been applied to information extraction from text [7] or structured documents [17], speech language modeling [5] and probabilistic dialog modeling [10].

The ALERGIA algorithm performs an ordered search in a lattice of automata $Lat(PPTA(I_+))$. This lattice is the set of automata that can be derived from $PPTA(I_+)$ by merging some states. The specific merging order, that is the order in which pair of states are considered for merging, is explained in detail and fully motivated in [4]. At each step of this algorithm, two states are declared compatible for merging, if the probability of any of their suffixes are similar within a certain threshold α. This parameter α indirectly controls the level of generalization of the inferred PDFA.

2.3 Evaluation criterion

Evaluation of non-probabilistic inference methods is usually based on correct classification rates of new positive and negative data [12]. In the case of PDFA inference,

the model quality can no longer be measured by classification error rate, as the fundamental problem has become the estimation of a probability distribution over the set of possible strings.

The quality of a PDFA $A = (Q, \Sigma, \delta, q_0, \gamma)$ can be measured by the *per symbol log-likelihood* of strings x belonging to a test sample according to the distribution defined by the solution $P_A(x)$ computed on a test sample S:

$$LL = \left(-\frac{1}{\|S\|} \sum_{j=1}^{|S|} \sum_{i=1}^{|x|} \log P(x_i^j | q^i) \right)$$

where $P(x_i^j | q^i)$ denotes the probability of generating x_i^j, the i-th symbol of the j-th string in S, given that the generation process was in state q^i. This average log-likelihood is also related to the Kullback-Leibler divergence between an unknown target distribution and the proposed solution by considering the test sample as the empirical estimate of the unknown distribution (see e.g. [5]).

The *test sample perplexity* PP is most commonly used for evaluating language models of speech applications. It is given by $PP = 2^{LL}$. The minimal perplexity $PP = 1$ is reached[2] when the next symbol x_i^j is always predicted with probability 1 from the current state q^i (i.e. $P(x_i^j | q^i) = 1$) while $PP = |\Sigma|$ corresponds to random guessing from an alphabet of size $|\Sigma|$.

3 Interpolation with a unigram model

In this section we present the basic smoothing technique which will serve as our reference model for smoothing probabilistic automata.

A *unigram model* is a probabilistic model in which the probability of any symbol a from Σ is independent from its context. It can be simply estimated by computing the frequency $C(a)$ of a in a training sample containing N tokens. The probability $P(a)$ is given by

$$P(a) = \frac{C(a)}{N}$$

and the probability $P_1(x)$ of a string $x = x_1 \ldots x_{|x|}$ is given by

$$P_1(x) = \prod_{i=1}^{|x|} P(x_i)$$

In general not all symbols are observed in the training sample and the unigram distribution is smoothed according to a *discounting parameter* d [13]:

$$\widehat{P}(a) = \begin{cases} \frac{C(a)-d}{N} & \text{, if } C(a) > 0 \\ \frac{D}{N_0} & \text{otherwise} \end{cases} \quad (1)$$

[2] Such a perfectly informed model cannot be constructed in general.

where D is the total discounted probability mass

$$D = \sum_{\{a \mid C(a)>0\}} \frac{d}{N}$$

and N_0 is the number of unseen symbols in the training sample

$$N_0 = \sum_{\{a \mid C(a)=0\}} 1.$$

A smoothed unigram model is guaranteed to assign a non-zero probability to any string which will be denoted $\widehat{P}_1(x)$. It is equivalent to the universal automaton built from the alphabet Σ with transitions probabilities defined according to equation (1). If $P_A(x)$ denotes the (possibly null) probability assigned to a string x by a PDFA A, a smoothed distribution is obtained by linear interpolation with the smoothed unigram model:

$$\widehat{P}(x) = \beta \cdot P_A(x) + (1 - \beta) \cdot \widehat{P}_1(x) \quad \text{, with } 0 \leq \beta < 1.$$

This smoothing technique is very rudimentary but, because it is so simple, it best reflects the quality of the PDFA itself. This smoothed probabilistic distribution serves as our reference model. In the sequel we study whether error-correcting techniques can improve over this reference model, that is whether a probabilistic model with smaller perplexity on independent data can be obtained.

4 Error-correcting model

Given A, a PDFA, and its language $L(A)$, error transitions can be added in order to make it possible to accept any possible string from Σ^* with a non-zero probability. This error model is fully described in section 4.1.

The problem of estimating the error-correcting model free parameters, which are the probabilities of error transitions, is detailed in section 4.2. Once the error model has been estimated from data, there may still be some string which cannot be generated with a non-zero probability. This is due to the fact that some error transitions may not have been seen during the estimation of the error model. Smoothing of the error model is then required as explained in section 4.3.

The adaptation of the original PDFA distribution in order to include the error transition probabilities to build a consistent model is described in section 4.4. Once the error model has been defined and its parameters estimated, the probability $\widetilde{P}(x)$ of generating any string x from the original PDFA extended with error transitions can be computed. An efficient algorithm to compute a path of maximal probability in any probabilistic automaton A (i.e. an automaton possibly including cycles and possibly non-deterministic) was recently proposed [1]. This algorithm is briefly presented in section 4.5. We use this algorithm here to reestimate iteratively the error model as described in section 4.6.

4.1 Model definition

Our error model is based on the addition of error transitions to an existing PDFA resulting in an *extended* automaton. These error transitions accounts for the possibility of inserting any symbol at any state, of substituting any existing transition labelled by a symbol a by any other symbol from the alphabet or of deleting the transition (or equivalently substituting a by the empty string λ).

Figure 1 illustrates the addition of error transitions to a PDFA. Initially there are only two transitions from state q labeled by a and b, respectively. The original automaton is extended with *insertion* transitions, *substitution* transitions and *deletion* transitions. Note that for this example the alphabet is assumed to have two symbols, $\Sigma = \{a, b\}$.

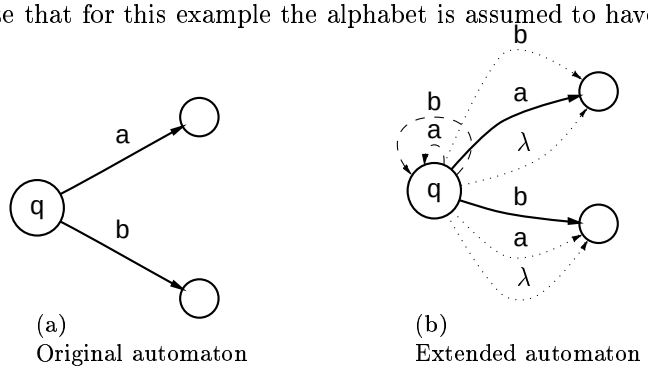


Fig. 1. Addition of error transitions to a PDFA

The parameters of the general error model are the following:

- $P(\lambda \to a \mid q)$ which denotes the probability of inserting symbol a while being in state q
- $P(a \to b \mid q,\ q')$ which denotes the probability of substituting a by b while going from q to q'. In particular $P(a \to a \mid q,\ q')$ denotes the probability of substituting a by a, that is of taking the original transition labeled by a from state q.
- $P(a \to \lambda \mid q,\ q')$ which denotes the probability of deleting a while going from q to q'.

Estimating an error model consists of estimating the error transitions probabilities. In order to minimize the number of free parameters, these probabilities can be made dependent on the symbol but independent of the transitions (or the state) they apply to. The parameters of the symbol dependent error model now become:

- $P(\lambda \to a)$ which denotes the probability of inserting symbol a in any state.
- $P(a \to b)$ which denotes the probability of substituting a by b while taking any transition labeled by a.
- $P(a \to \lambda)$ which denotes the probability of deleting a while taking any transition labeled by a.

Alternatively, the error model can be made state dependent instead of symbol dependent. In our case, we adopt a symbol dependent error model as the alphabet is usually known before the automaton inference process. State independence also allows us to merge several error models as described in section 6.4.

4.2 Estimation of an error model

Once a PDFA is given or inferred from a training sample, the parameters of the error model can be estimated on an independent sample. For any string x from this independent sample, the probability of generating the string can be computed. This requires that a consistent probability distribution can be defined for the extended automaton as detailed in section 4.4. Note also that after the extension of the original automaton with error transitions, the new automaton is no longer deterministic. Following a Viterbi criterion, the probability of generating x can be approximated by the probability of the most likely path to generate x. An efficient algorithm to compute this path is described in section 4.5.

The set of editing operations used while generating the independent sample from the extended automaton can be stored and the associated counts can be computed:

- $C(\lambda, a)$ denotes the number of insertions of the symbol a.
- $C(a, b)$ denotes the number of substitutions of the symbol a by b. In particular, $C(a, a)$ denotes the number of times the symbol a was asserted, that is, not substituted while parsing the independent sample.
- $C(a, \lambda)$ denotes the number of deletions of the symbol a.
- $C(\#)$ denotes the number of (end of) strings.

As the proposed error model is *state independent* several estimates of its parameters for various underlying automata can be computed. Combining these estimates simply amounts to summing the respective error counts. This property will be used in our experiments as explained in section 6.4.

4.3 Smoothing of the error counts

Some counts associated with error transitions may be null after estimating the error model. This is the case when some error transitions are never used along any of the most likely paths computed while parsing the independent sample. This problem can be solved by adding real positive values to the error counts. We use four additional parameters $\varepsilon_{ins}, \varepsilon_{sub}, \varepsilon_{del}$ and ε_{noerr} to smooth the error counts :

- $\widehat{C}(\lambda, a) = C(\lambda, a) + \varepsilon_{ins}$
- $\widehat{C}(a, b) = C(a, b) + \varepsilon_{sub}$ if $a \neq b$
- $\widehat{C}(a, a) = C(a, a) + \varepsilon_{noerr}$
- $\widehat{C}(a, \lambda) = C(a, \lambda) + \varepsilon_{del}$

4.4 Definition of the extended PDFA distribution

In the original PDFA $P(a|q) = \gamma(q, a)$ denotes the probability of generating the symbol a from state q whenever such a transition exists. This transition probability can be estimated from the training sample from which the PDFA was built. The maximum likelihood estimate for $\gamma(q, a)$ is given by

$$\gamma(q, a) = \frac{C_q(a)}{C_q}$$

where $C_q(a)$ denotes the number of times the symbol a was generated from state q and C_q denotes the number of times the state q was observed while parsing the training sample. We can assume that the counts $C_q(a)$ and C_q are strictly positive, as any transition (or state) which would not satisfy this constraint would be initially removed from the original PDFA.

The probability distribution of the extended automaton can be defined as follows. The total *insertion count* C_{ins} is defined as

$$C_{ins} = \sum_{a \in \Sigma} \widehat{C}(\lambda, a)$$

and its complementary count $C_{\overline{ins}}$ is defined as

$$C_{\overline{ins}} = \sum_{a \in \Sigma} \sum_{b \in \Sigma \cup \{\lambda\}} \widehat{C}(a, b) \quad + C(\#)$$

Let $P_{ins} = \frac{C_{ins}}{C_{ins} + C_{\overline{ins}}}$ denote the probability of inserting any symbol. The probability of the transitions from any state q in the extended automaton are computed as follows:

- the probability of inserting a while being in state q:

$$P(\lambda \to a|q) \doteq P(\lambda \to a) = P_{ins} \cdot \frac{\widehat{C}(\lambda, a)}{C_{ins}} \tag{2}$$

- the probability of substituting a by b, for any symbol b in the alphabet Σ (including the case where $a = b$), from state q :

$$P(a \to b|q) \doteq P(a \to b) \cdot \gamma(q, a) = \left[(1 - P_{ins}) \cdot \frac{\widehat{C}(a, b)}{\sum_{b \in \Sigma \cup \{\lambda\}} \widehat{C}(a, b)} \right] \cdot \gamma(q, a) \tag{3}$$

- the probability of deleting a from state q :

$$P(a \to \lambda|q) \doteq P(a \to \lambda) \cdot \gamma(q, a) = \left[(1 - P_{ins}) \cdot \frac{\widehat{C}(a, \lambda)}{\sum_{b \in \Sigma \cup \{\lambda\}} \widehat{C}(a, b)} \right] \cdot \gamma(q, a) \tag{4}$$

- the probability of generating the end of string symbol # from state q :

$$P(\# \mid q) \doteq (1 - P_{ins}) \cdot \gamma(q, \#) \tag{5}$$

4.5 Computation of the most likely path

The general problem of finite-state parsing with no error correction can be formulated as a search for the most likely path or equivalently the minimum cost[3] path through a *trellis diagram* associated to the PDFA A and the string x to be parsed. This trellis

[3] *Sums of negative log* probabilities rather than products of probabilities are used.

is a directed *acyclic* multistage graph, where each node q_k^j corresponds to a state q_j in a stage k. The stage k is associated with a symbol x_k of the string to be parsed and every edge of the trellis $t_k = (q_k^i, q_{k+1}^j)$ stands for a transition between the state q_i in stage k and the state q_j in stage $k+1$ (Fig. 2 (a)). Thanks to the acyclic nature of this graph, dynamic programming can be used to solve the search problem, leading to the well-known Viterbi algorithm [6].

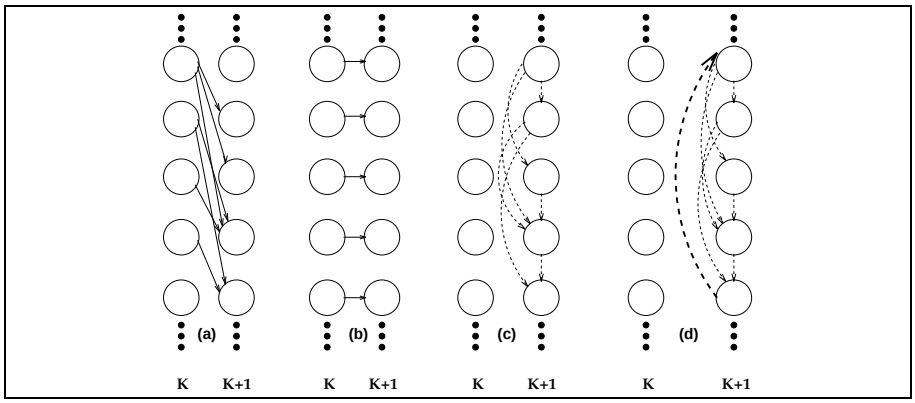


Fig. 2. Trellis with: a) Substitution and proper PDFA transitions b) Insertion transitions c) Deletion transitions in an *acyclic* PDFA d) Deletion transitions in a *cyclic* PDFA. Every edge is labeled with a symbol of Σ.

The trellis diagram can be extended in a straightforward fashion to parse errors produced by *substitution* and *insertion* actions. Efficient error correcting parsing can be implemented because such an extended trellis diagram still has the shape of a directed *acyclic* multistage graph (Fig. 2 (a),(b)). However, the extension of the trellis diagram to parse errors produced by *deletion* of one or more (consecutive) symbol(s) in the original string results in a graph form that includes edges between nodes belonging to the same stage k (Fig. 2 (c)). In particular, when the automaton A has cycles dynamic programming can no longer be used as the problem becomes one of finding a minimum cost path through a *general* directed *cyclic* graph (Fig. 2 (d)). As noted in [8], we can still take advantage of the fact that most edges, for this kind of graph, still have a *left-to-right* structure and consider each column as a separate stage like in the Viterbi algorithm.

An efficient algorithm for computing the most likely acceptance path that includes error operations in general automata was proposed in [1]. This algorithm can be considered as an extension of the Viterbi algorithm. The main difference lies in the fact that an order has to be defined when parsing deletion transitions (see Fig. 2 (c), (d)) for adequately performing the required computations during local (state) minimizations. In particular, it is based on the definition of a *pseudo-topological* state ordering, that is an extension to cyclic graphs of the usual topological ordering. This pseudo-topological ordering is computed and efficiently stored in a hash table during a preprocessing stage which detects the backward edges, i.e. those edges which produce cycles in A. This leads

to a fixed order for the traversal of the list of nodes (states of the PDFA) at any stage of the parsing process in order to update the cumulated costs whenever required.

Full details of the computation of this state ordering, the resulting parsing algorithm and practical evaluations are presented in [1]. We use here this algorithm to compute the most likely path of generating a string from the extended PDFA.

4.6 Greedy reestimation of the smoothed distribution

Computing the most likely path using the technique described in section 4.5 is equivalent to computing the path of minimum cumulated cost. For example the cost $D_q(a \to b)$ of substituting a by b from state q is given by $D_q(a \to b) = -\log P(a \to b|q)$. Thus the maximization of a product of probabilities becomes a minimization of additive costs.

The *initial* error model can not be derived from the probabilistic error model described in section 4.4, as the error counts are initially unknown and the extended (smoothed) PDFA distribution can not be computed. However a set of editing costs can be defined *a priori*, for instance according to the Levenshtein distance [11]: $D_q(\lambda \to a) = 1$, $D_q(a \to b) = 1$ if $a \neq b$, $D_q(a \to a) = 0$ and $D_q(a \to \lambda) = 1$. Once the initial editing costs are defined, the counts of insertions, substitutions and deletions that minimize the Levenshtein distance criterion on an independent sample can be computed as described in section 4.1. Note that, in this particular case, only the structure of the PDFA is required. A new error model can then be derived from these error counts, and this estimation can be iterated with a true probabilistic error model. This reestimation process is performed until a maximum number of iterations is reached (typically 10) or until the relative change of perplexities computed on two consecutive iterations falls below a certain threshold (typically 1%).

During this iterative procedure, the original PDFA distribution can also be reestimated by adding to the original counts, C_q and $C_q(a)$, their values computed on the independent sample and by modifying accordingly the estimate of $\gamma(q, a)$. This will be referred to as *reestimation of non-error transitions*.

5 The ATIS task

The Air Travel Information System (ATIS) corpus [9] was developed under a DARPA speech and natural language program that focussed on developing language interfaces to information retrieval systems. The corpus consists of speakers of American English making information requests such as,

"Uh, I'd like to go from, uh, Pittsburgh to Boston next Tuesday, no wait, Wednesday".

Each user was given several goal scenarios to work with, in which he or she had to try to make travel arrangements between multiple cities in North America. A database containing information from the Official Airline Guide was at the heart of the system. Users could ask questions about a wide variety of items in the database, ranging from flight information to aircraft equipment descriptions and even meals served on particular flights. They could speak naturally to the machine, as there was no fixed interaction language or required sequence of events. Spoken language phenomena such

as truncated words, hesitations, false starts, and verbal error recovery are common in the corpus. It is commonplace to find multiple turn interactions (and thus multiple utterances from a user) between the user and machine for solving each scenario.

6 Experiments

6.1 Data sets

We use the ATIS-2 sub-corpus in the experiments reported here. This portion of the corpus was developed under Wizard-of-Oz conditions in which a human being secretly replaced the speech recognition component of an otherwise fully automated dialogue system. The ATIS-2 collection is officially defined as containing a *training set* and two *evaluation sets*. The training set, which we used for inferring PDFAs, contains 13,044 utterances (130,773 tokens). The vocabulary contains 1,294 words. We used the first evaluation set (Feb92, 974 utterances, 10636 tokens) as a *validation set* to estimate the baseline perplexity and an error model. The second evaluation set (Nov92, 1001 utterances, 11703 tokens) was used as our independent *test set*. In the context of these experiments, alphabet symbols represent *words* from the ATIS vocabulary and strings represent *utterances*.

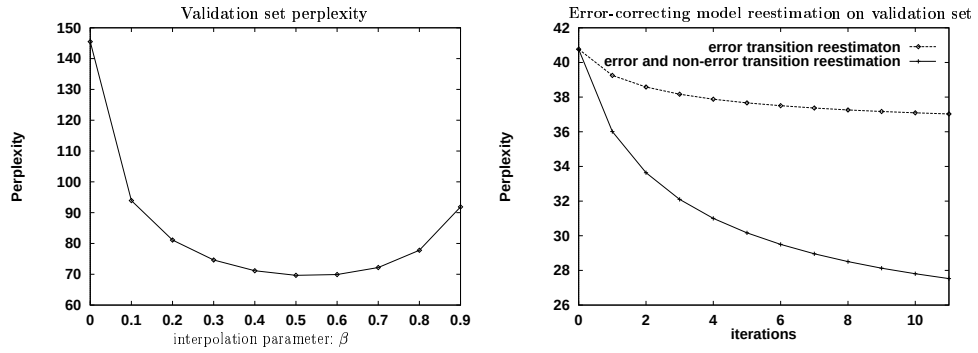
6.2 Baseline perplexity

A PDFA is inferred from the training set using the ALERGIA algorithm. The resulting PDFA consists of 414 states and 12,303 transitions. It accepts 55 % (532 strings) of the validation set, illustrating the need for smoothing the PDFA distribution. In particular the validation set perplexity is infinite without smoothing. Figure 3(a) shows the perplexity obtained after smoothing by interpolating with a unigram model as explained in section 3. The optimal perplexity (70) is obtained for β equal to 0.5.

6.3 Validation set perplexity with error model

The initial error model parameters are estimated from training and validation sets by counting the observed editing operation frequencies so as to minimize the Levenshtein distance (see section 4.6). As some error transitions are not observed during this process, the initial error table is then smoothed (see section 4.3). The additional smoothing parameters (ε_{ins}, ε_{sub}, ε_{del} and ε_{noerr}) are adjusted in order to minimize the perplexity on the last 10 % of the validation set while estimating the error model only on the first 90 % of the validation set. Their optimal values are $\varepsilon_{ins} = 0.1$, $\varepsilon_{sub} = 0.1$, $\varepsilon_{del} = 0.1$ and $\varepsilon_{noerr} = 0.0$.

Figure 3(b) shows the perplexity obtained on the validation set during reestimation of the error model. The initial perplexity (41) is achieved after the initial estimation of error parameters, based on the counts of the editing operations which minimize Levenshtein distance. In the first case (type I model), only error transitions are reestimated resulting in a 10% relative perplexity improvement (from 41 to 37). In the second case (type II model), error and non-error transitions probabilities are reestimated. The perplexity obtained after 10 iterations is 28.



(a) (b)

Fig. 3. Perplexity results

6.4 Estimating the error model by cross-validation

In the experiments described in section 6.3, the error model was constructed and reestimated on the validation set. The training set, which represents about 13 times more data, was not used for estimating the error model as the original automaton is guaranteed to accept all training strings without errors. However a better estimate of the error model can be obtained using cross-validation. This procedure can be summarized as follows:

- Concatenate training and validation set in a single data set.
- Construct N (typically 10) different partitions of this data set.
- For each partition, infer a PDFA on the first part (typically 90 % of the data set) and estimate an error model on the second part (typically the remaining 10 %) following the greedy procedure described in section 4.6.
- Merge all error models by summing up the error counts obtained on each partition.

Merging of several error models is simple in our case, as these models are symbol dependent but do not depend on the structure of the underlying automaton. Once the error model is estimated by cross-validation, a final reestimation on the validation set can be performed using the original automaton constructed on the training set only.

6.5 Independent test set perplexity

Table 1 summarizes the final results computed on an independent test set. The reference model is a PDFA interpolated with a unigram using the optimal interpolation parameter estimated on the validation set ($\beta = 0.5$). Type I error model refers to the model obtained after reestimating only the error transition probabilities on the validation set. Type II error model refers to the model obtained after reestimating both error and non error transitions probabilities. In both cases the error model probabilities may

be simply computed on the validation set or can be estimated by cross-validation (CV) following the procedure described in section 6.4.

Table 1. Test set perplexity

Model	Perplexity
Unigram smoothing ($\beta = 0.5$)	71
Type I error model	40
Type II error model	41
CV + Type I error model	37
CV + Type II error model	37

The reestimation of non-error transitions does not improve the perplexity of the extended PDFA on an independent test set. The significant perplexity decrease on the validation set, as seen in figure 3(b), is thus a result of overfitting to the validation data. On the other hand, cross-validation allows for up to 10 % relative perplexity reduction. Finally these results show a 48 % relative perplexity reduction as compared to the perplexity obtained by interpolating with a unigram model.

7 Conclusions and future work

We have examined the issues of smoothing probabilistic automata by adding error transitions to an original probabilistic automaton structure. The probability distribution of the extended automaton is such that any possible string can be predicted with non-zero probability. We explained how to define a consistent error model and how to estimate its free parameters from independent data.

Practical experiments on the ATIS travel information task show a 48 % test set perplexity reduction on new data with respect to a simply smoothed version of the original automaton.These experiments illustrate the risk of overfitting when both the error model and the initial non error transitions are reestimated. On the other hand, cross-validation allows us to estimate a more reliable error model which results in significant perplexity reduction on new data.

The error model proposed here is symbol dependent but state independent. In particular, the probability of inserting a given symbol a does not depend on where this symbol is inserted. In order to refine the error model without significantly increasing the number of free parameters, the relative weight of error versus non-error transitions could also be estimated for each state.

We presented here the error correcting approach as a method for extending a probabilistic deterministic automaton. Most existing inference algorithms produce deterministic machines which after extension with error transitions become non-deterministic. The techniques presented here handle this non-determinism. Thus, smoothing of automata which are non-deterministic from the start is also something we can pursue.

Clustering alphabet symbols before PDFA inference was shown to reduce perplexity on new data [5]. Combination of this technique with error correcting will also be investigated in the future.

References

1. J.-C. Amengual and E. Vidal. Efficient error-correcting viterbi parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-20(10), October 1998.
2. J.-C. Amengual, E. Vidal, and J.-M. Benedí. Simplifying language through error-correcting techniques. In *International Conference on Spoken Language Processing*, pages 841–844, 1996.
3. R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 139–150, Alicante, Spain, 1994. Springer Verlag.
4. R. Carrasco and J. Oncina. Learning deterministic regular gramars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, 33(1):1–19, 1999.
5. P. Dupont and L. Chase. Using symbol clustering to improve probabilistic automaton inference. In *Grammatical Inference, ICGI'98*, number 1433 in Lecture Notes in Artificial Intelligence, pages 232–243, Ames, Iowa, 1998. Springer Verlag.
6. G.D. Forney. The Viterbi algorithm. *IEEE Proceedings*, 3:268–278, 1973.
7. D. Freitag. Using grammatical inference to improve precision in information extraction. In *Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, Fourteenth International Conference on Machine Learning*, Nashville, Tennessee, 1997.
8. G.W. Hart and A. Bouloutas. Correcting dependent errors in sequences generated by finite-state processes. *IEEE Trans. on Information Theory*, 39(4):1249–1260, July 1993.
9. L. Hirschman. Multi-site data collection for a spoken language corpus. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 7–14, Arden House, NY, 1992.
10. K. Kita, Y. Fukui, M. Nagata, and T. Morimoto. Automatic acquisition of probabilistic dialogue models. In *Proceedings of ISSD96, workshop of the International Conference on Spoken Language Processing*, pages 196–199, Philadelphia, October 1996.
11. J.B. Kruskal. An overview of sequence comparison. In D. Sankoff and J.B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, pages 1–44. Addison-Wesley, Reading, Massachusetts, 1983.
12. K.J. Lang, B.A. Pearlmutter, and R.A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, number 1433 in Lecture Notes in Artificial Intelligence, pages 1–12, Ames, Iowa, 1998. Springer-Verlag.
13. H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.
14. J. Oncina and P. García. Inferring regular languages in polynomial update time. In N. Pérez de la Blanca, A. Sanfeliu, and E.Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, Singapore, 1992.
15. D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic automata. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pages 31–40, Santa Cruz, CA, 1995. ACM Press.
16. H. Rulot and E. Vidal. An efficient algorithm for the inference of circuit-free automata. In G. Ferratè, T. Pavlidis, A. Sanfeliu, and H. Bunke, editors, *Advances in Structural and Syntactic Pattern Recognition*, pages 173–184. NATO ASI, Springer-Verlag, 1988.
17. M. Young-Lai and F. Tompa. Stochastic grammatical inference of text database structure. To appear in *Machine Learning*, 2000.