Using symbol clustering to improve probabilistic automaton inference

Pierre Dupont[1] and Lin Chase[2]

[1] EURISE, Université Jean Monnet
23, rue P. Michelon
42023 Saint-Etienne Cedex – France
pdupont@univ-st-etienne.fr
[2] LIMSI/CNRS
B.P.133
91403 Orsay Cedex – France
chase@limsi.fr

Abstract. In this paper we show that clustering alphabet symbols before PDFA inference is performed reduces perplexity on new data. This result is especially important in real tasks, such as spoken language interfaces, in which data sparseness is a significant issue. We describe the application of the ALERGIA algorithm combined with an independent clustering technique to the Air Travel Information System (ATIS) task. A 25 % reduction in perplexity was obtained. This result outperforms a trigram model under the same simple smoothing scheme.

1 Introduction

Inference of deterministic finite automaton (DFA) from positive and negative data can be solved by the RPNI algorithm, proposed independently by Trakhtenbrot *et al.* [16] and by Oncina *et al.* [13]. This algorithm was used by Lang in his extensive experimental study of learning random deterministic automata from sparse samples [10]. An adapted version of this algorithm proved to be successful in the recent Abbadingo competition [9]. However the use of the RPNI algorithm and other grammatical inference techniques requires several adaptations in order to be feasible for real applications such as speech recognition interfaces.

Since most databases do not include negative information, the learning can be controlled by probabilistic information instead. Several inference algorithms for probabilistic automata are known [15, 3, 14]. In particular, the ALERGIA algorithm proposed by Carrasco and Oncina [3] is the probabilistic version of the RPNI algorithm. As the purpose of the machines built by these algorithms is not to define hard decision boundaries, the quality of learning is not estimated by classification errors on new data. Instead we use a distance measure, such as the Kullback-Leibler divergence, between the learned distribution and the target distribution. Because the target distribution is unknown in practice, this distance measure is approximated by test set perplexity (see section 2 for more details).

Another problem arises from the fact that simulated learning curves do not accurately predict the performance on actual data sets of limited size. First, learning curves usually depend on the size of the target machine which is unknown. Second, real alphabets, at least for speech applications, typically contain up to several thousand symbols instead of just a few. Thus an extremely small fraction of all strings up to the maximal length observed is actually available. This data sparseness problem is a key concern to reserarchers who deal with human language applications.

In this context, clustering alphabet symbols can lead to a denser training set at the cost of a reduced ability to discriminate between individual symbols. The present work aims at showing that the performance of the ALERGIA algorithm improves when a clustering algorithm is used before the actual inference.

2 Definitions and notations

A *probabilistic DFA* (PDFA) is a 6-tuple $(Q, \Sigma, \delta, q_0, \gamma, \tau)$ where Q is a finite set of *states*, Σ is an *alphabet*, δ is a *transition function*, i.e. a mapping from QxΣ to Q, q_0 is the *initial state*, γ is the *next symbol probability function*, i.e. a mapping from QxΣ to $[0,1]$ and τ is the *end of string probability function*, i.e. a mapping from Q to $[0,1]$. The probability functions must satisfy the following constraints:

$$\begin{aligned} \gamma(q,a) &= 0 \text{ , if } \delta(q,a) = \emptyset \\ \sum_{a \in \Sigma} \gamma(q,a) + \tau(q) &= 1 \text{ ,} \forall q \in Q \end{aligned}$$

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

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

Our definition of probabilistic automaton is equivalent to a stochastic regular grammar used as string generator. Thus, $\sum_{x \in \Sigma^*} P_A(x) = 1$. Note that some works on the learning of discrete distributions use distributions defined on Σ^n (that is $\sum_{x \in \Sigma^n} P(x) = 1$, for any $n \geq 1$), instead of Σ^* (see for instance [1, 7]).

The probabilistic automaton A/π denotes the automaton *derived from* the probabilistic automaton A *with respect to the partition* π of Q, also called the *quotient automaton* A/π. It is obtained by merging states of A belonging to the same subset in π. When q results from the merging of the states q' and q'' , the following equalities must hold

$$\begin{aligned} \gamma(q,a) &= \gamma(q',a) + \gamma(q'',a) \text{ ,} \forall a \in \Sigma \\ \tau(q) &= \tau(q') + \tau(q'') \end{aligned}$$

Let $PPTA(I_+)$ denote the *probabilistic prefix tree acceptor* built from a positive sample I_+. Let $C(q)$ denote the count of state q, that is the number of times the state q was used while generating I_+ from $PPTA(I_+)$. Let $C(q, \lambda)$ denote the number of times a string of I_+ ended on q (λ denotes the empty string). Let $C(q, a)$ denote the count of the transition (q, a) in $PPTA(I_+)$. The $PPTA(I_+)$ is the maximal likelihood estimate built from I_+. In particular, for $PPTA(I_+)$ the probability estimates are $\hat{\gamma}(q, a) = \frac{C(q,a)}{C(q)}$ and $\hat{\tau}(q) = \frac{C(q,\lambda)}{C(q)}$.

Let $Lat(PPTA(I_+))$ denote the lattice of automata which can be derived from $PPTA(I_+)$.

Let A be a target PDFA and $\hat{A}$ a hypothesis PDFA, $D(P_A \parallel P_{\hat{A}})$ denotes the *Kullback-Leibler divergence* or *cross entropy*.

$$D(P_A \parallel P_{\hat{A}}) = \sum_{x \in \Sigma^*} P_A(x) \log \frac{P_A(x)}{P_{\hat{A}}(x)}$$

The divergence can be rewritten as follows.

$$\begin{aligned} D(P_A \parallel P_{\hat{A}}) &= \sum_{x \in \Sigma^*} P_A(x) \log \frac{1}{P_{\hat{A}}(x)} + \sum_{x \in \Sigma^*} P_A(x) \log P_A(x) \\ &= E_{P_A}\left[\log \frac{1}{P_{\hat{A}}(x)}\right] - H(P_A) \end{aligned}$$

The first term above denotes the expectation of $\log \frac{1}{P_{\hat{A}}(x)}$ according to the distribution P_A and $H(P_A)$ denotes the entropy of P_A. In other words, the divergence is the additional number of bits[1] needed to encode data generated from P_A when the optimal code is used for $P_{\hat{A}}$ instead of P_A. Note that the second term does not depend on the hypothesis. Thus the first term can be used to measure the *relative* quality of several hypotheses.

In practice, the target distribution is unknown. Instead, an independent sample S , which is assumed to have been drawn according to the distribution P_A, can be used. Let $\tilde{P}_S(x)$ denote the empirical distribution computed from the sample S containing $|S|$ strings x and $\|S\|$ symbols x_i. Letting $C_S(x)$ denote the count of the string x in S, one can write

$$\begin{aligned} E_{P_A}\left[\log \frac{1}{P_{\hat{A}}(x)}\right] &\simeq \sum_{x \in S} \tilde{P}_S(x) \log \frac{1}{P_{\hat{A}}(x)} \\ &= -\frac{1}{|S|} \sum_{x \in S} C_S(x) \log P_{\hat{A}}(x) \\ &= -\frac{1}{\|S\|} \sum_{i=1}^{\|S\|} \log P_{\hat{A}}(x_i | q^i) \end{aligned}$$

Thus the quality measure is the *average log-likelihood* of x according to the distribution $P_{\hat{A}}$ computed on the sample S. Most commonly used is the *sample perplexity* PP given by

$$2^{\left(-\frac{1}{\|S\|} \sum_{i=1}^{\|S\|} \log P_{\hat{A}}(x_i | q^i)\right)}$$

[1] Base 2 is assumed for the log function.

The minimal perplexity $PP = 1$ is reached when the next symbol is always predicted with probability 1 while $PP = |\Sigma|$ corresponds to random guessing from a lexicon of size $|\Sigma|$.

3 Probabilistic DFA Inference

```
Algorithm ALERGIA
input
I+                                          // A positive sample
α                                           // A precision parameter
output
a PDFA                                      // A probabilistic DFA

begin

// N is the number of states of PPTA(I+)

π ← {{0}, {1}, ..., {N − 1}}     // One block for each prefix in the order <
A ← PPTA(I+)

for i = 1 to |π| − 1                        // Loop on the blocks of partition π
   for j = 0 to i − 1                       // Loop on the blocks of lower rank
      if compatible (i, j, α) then
        π' ← π\{Bj, Bi}U{BiUBj}             // Merging of block Bi and block Bj
        A/π' ← derive(A, π')
        π'' ← determ_merge (A/π')
        A ← A/π''
        π ← π''
        break                               // Break j loop
      end if
   end for                                  // End j loop
end for                                     //End i loop
return A
end ALERGIA
```

Fig. 1. The ALERGIA algorithm.

The ALERGIA algorithm [3] is depicted in pseudocode in figure 1. It performs an ordered search in the lattice $Lat(PPTA(I_+))$. By construction each of the states of $PPTA(I_+)$ corresponds to a unique prefix. The prefixes may be sorted according to the standard order $<$ on strings[2]. This order also applies to the prefix tree states. A partition in the state set of the $PPTA(I_+)$ consists of an ordered block set, each block receiving the rank of its state of minimal rank. The ALERGIA algorithm proceeds in $N-1$ steps, where $N = \mathcal{O}(\|I_+\|)$ is the number of states of $PPTA(I_+)$. The partition $\pi(i)$ at step i is obtained by merging the

[2] According to the standard order, the first strings on the alphabet $\Sigma = \{a, b\}$ are $\lambda < a < b < aa < ab < ba < bb < aaa < \ldots$

two first blocks, according to the standard order, of the partition $\pi(i-1)$ at step $i-1$ such that $PTA(I_+)/\pi(i)$ is a compatible automaton.

The function *derive* (A, π') returns the quotient automaton A with respect to partition π'. The automaton A/π' may be non-deterministic. The function *determ_merge* (A/π') returns the partition π'' obtained by recursively merging all blocks of π', which creates a non-determinism. If A/π' is deterministic, the partition π'' is equal to partition π'.

The function *compatible* (i, j, α) controls the merging of states. It returns TRUE if the states i and j are compatible within the precision defined by the parameter α and FALSE otherwise. The compatibility measure derives from the Hoeffding bound [5]. Formally, two states q_1 and q_2 are α-compatible ($0 < \alpha \leq 1$) if the two following conditions hold

$$\left|\frac{C(q_1,a)}{C(q_1)} - \frac{C(q_2,a)}{C(q_2)}\right| < \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}\left(\frac{1}{\sqrt{C(q_1)}} + \frac{1}{\sqrt{C(q_2)}}\right), \forall a \in \Sigma \cup \lambda \quad (1)$$

$$\delta(q_1, a) \text{ and } \delta(q_2, a) \text{ are } \alpha\text{-compatible}, \forall a \in \Sigma \quad (2)$$

4 Clustering algorithm

At least two algorithms have been proposed to define classes automatically from data [2, 12]. Both algorithms can be shown to optimize the same criterion, the minimization of the training set perplexity of a class bigram. This is equivalent to the maximization of *average mutual information* between classes $I(g_1, g_2) = E_p\left[\log\frac{P(g_1,g_2)}{P(g_1)P(g_2)}\right]$ (see [2] for more details). While the algorithms use the same optimization function, however, they differ in the greedy search they follow. We have chosen to use the clustering algorithm proposed by Ney *et al.* [12] which is described in figure 2.

4.1 Definition of a class bigram

Given a deterministic symbol to class mapping $g : x_i \to g(x_i)$ a class bigram is defined as :

$$p(x_i|x_{i-1}) \stackrel{def}{=} p(x_i|g(x_i)).p(g(x_i)|g(x_{i-1}))$$

Let k denote the number of classes. The two limit cases are $k = 1$, which is equivalent to a unigram model, or $k = |\Sigma|$, which is equivalent to a bigram model at the symbol level.

4.2 Greedy reestimation algorithm

The objective of the clustering algorithm is to find the class mapping which minimizes the class bigram perplexity on the training set for a fixed number of classes k. Initially, each of the $k-1$ most frequent symbols is assigned to a 1-symbol class. All other symbols are assigned to the last class. At every iteration each symbol is moved from its original class to the class which corresponds to the local minimum of the class bigram perplexity. The reestimation is halted either when a predefined number of iterations (typically 100) is reached or when the difference between perplexities obtained at two consecutive iterations falls below a certain threshold (typically 0.01).

```
Algorithm CLUSTER
input
I_+                                                      // A positive sample
k                                                        // A fixed number of classes
output
g                                                        // A symbol to class mapping

begin
g ← Initilialize_mapping (I_+, k)                        // Initialize symbol to class mapping
while (not end)
    for all a ∈ Σ                                        // Loop on all symbols of the alphabet
        for j = 1 to k                                   // Loop on the k classes
            g(a) ← g^j                                   // Move a from g(a) to g^j
            PP(g^j) ← perplexity (I_+, g)                // Compute the resulting perplexity
        end for                                          // End j loop
        g(a) ← argmin_{g^j} PP(g^j)
    end for all                                          // End loop on Σ
end while                                                // Some stopping criterion is met
return g
end CLUSTER
```

Fig. 2. The clustering algorithm.

5 Infering from clustered data

Clustering the alphabet symbols allows us to reduce the alphabet size and thus to increase the training set density, relative to the set of strings defined on the class alphabet. After the symbol mapping has been performed, however, prediction of individual symbols can be more difficult. Trading off these effects one can find an optimal number of classes such that the resulting PDFA has a minimal perplexity on independent data. This is decribed in details in section 7.

The inference procedure can be summarized as follows.

1. Estimate a symbol to class mapping
2. Label the training set in terms of classes
3. Infer a PDFA from the relabeled training set
4. Expand the resulting PDFA by replacing each class by the symbols it contains
5. Interpolate the resulting PDFA with a unigram model

The first step requires choosing k, the number of classes. For the third step, the precision parameter α of the ALERGIA algorithm must be set. Expansion of the resulting PDFA consists in replacing each transition labeled with a class by a set of transitions labeled by its corresponding symbols. The probability of any transition (q, a) is then given by $\gamma(q, a) = \gamma(q, g(a))P(a|g(a))$. Finally, in order to guarantee that the probability of predicting any symbol from any given state is strictly positive, the PDFA must be smoothed. We use here linear interpolation with a unigram model $P_1(x)$:

$$P(x) = \beta P_{\hat{A}}(x) + (1 - \beta)P_1(x)$$

This smoothing technique is very rudimentary but, because it is so simple, it best reflects the quality of the PDFA itself. Thus our inference procedure depends on the free parameters k, α, β.

6 The ATIS task

The Air Travel Information System (ATIS) corpus [4] 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".

We use the ATIS-2 sub-corpus in the experiments reported here. This portion of the corpus was developed under a 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 is officially defined as containing a training set and two evaluation sets. The training set, which we used for estimating classes and for infering PDFA's, contains 13,186 utterances (130,773 tokens). The observed vocabulary contains 1,294 words. We used the first evaluation set (Feb92, 980 utterances, 10636 tokens) as a development set to tune our free parameters (k, α, β). The second evaluation set (Nov92, 1002 utterances, 11703 tokens) was used as our independent test set.

7 Results

In the context of these experiments, alphabet symbols represent *words* from the ATIS vocabulary and strings represent *utterances*. Our experiments aim at showing that the use of classes reduces perplexity on independent data.

The first three figures describe the original ALERGIA algorithm without using classes while the last three figures illustrate the inference behavior when clustering is performed beforehand. Figure 3 shows that the size of the inferred PDFA grows with the value of α. As α increases the compatibility criterion is harder to satisfy. Consequently fewer merging operations are performed.

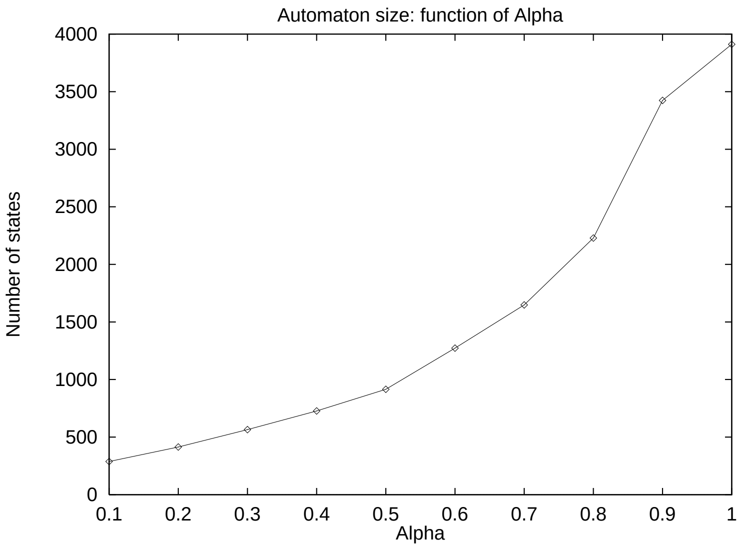

Fig. 3. Effect of α on the automaton size.

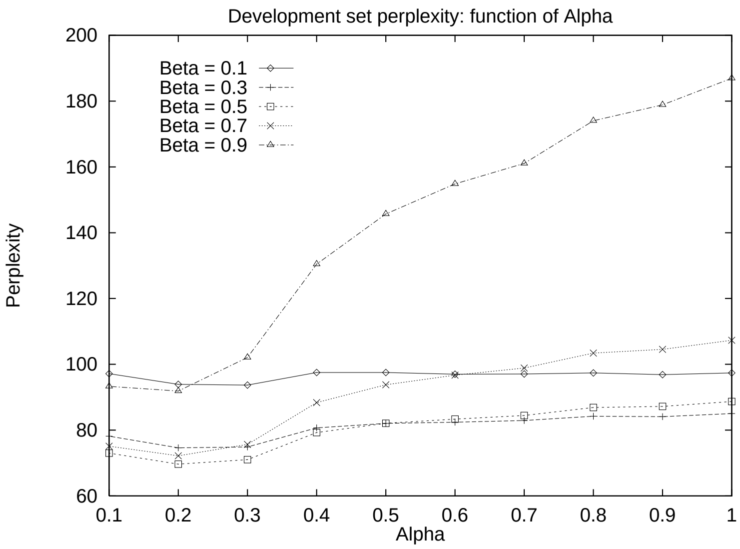

Fig. 4. Effect of α on development set perplexity with various interpolation parameters.

With the original ALERGIA algorithm the perplexity on new data depends only on the two parameters α and β. Figure 4 illustrates the relation between perplexity, α and β as measured on the development set. Optimal values are $\alpha = 0.2$ and $\beta = 0.5$. This result is detailed at figure 5 where the development set perplexity depends on α (for β fixed to its optimal value). The perplexity is minimal (PP=70) for small α's. However the limit case ($\alpha = 0$) results in the universal automaton which is equivalent to a unigram model. In this case the perplexity is 145.

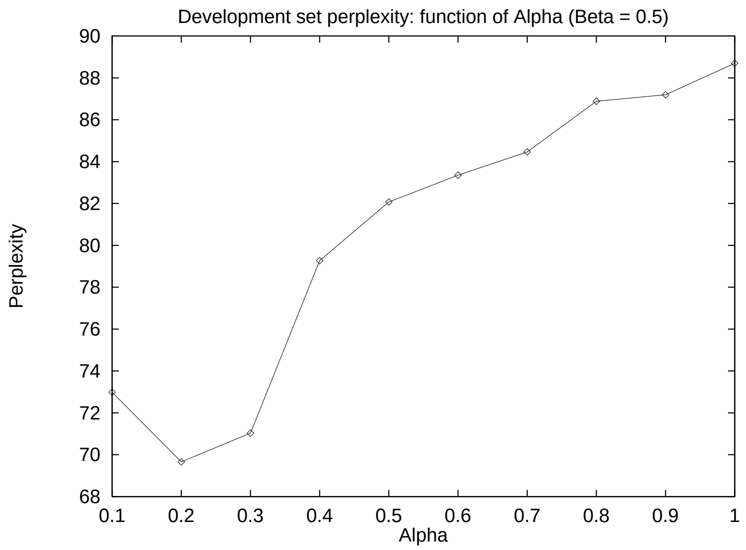

Fig. 5. Development set perplexity as a function of α with optimal interpolation ($\beta = 0.5$).

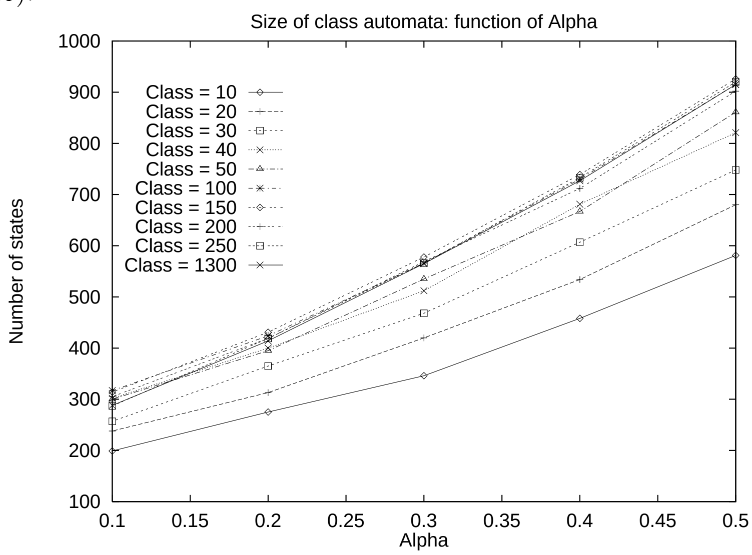

Fig. 6. Effect of α on the size of automata learned from classes.

Next we study how the use of classes modifies the previous results. In this case class-based automata refer to PDFA's obtained after expanding each class transition into its corresponding word transitions. Figure 6 shows that the size of the resulting automata follows the same dependence on α. Here the maximal number of classes ($k = 1300$) correponds to the reference model, as each word belongs to its own singleton class. Only small numbers of classes ($k \leq 50$) give rise to smaller automata for equal α's.

Figure 7 shows how the development set perplexities evolves as a function of α and k where β's are fixed to their optimal value for each class number. Note that the use of 40 classes reduces the development set perplexity to 51.

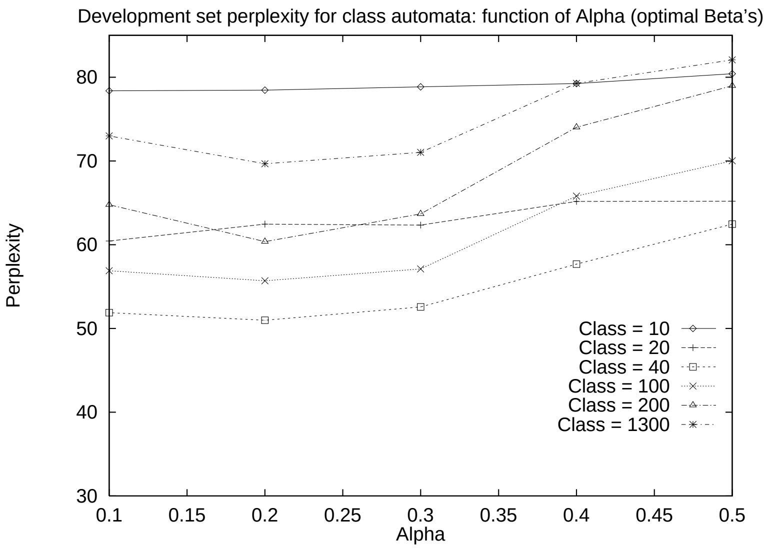

Fig. 7. Effect of α on the development set perplexity for class automata.

Figure 8 and table 1 summarize the best perplexity obtained for various number of classes (α and β are fixed to their optimal values in each case). The results in this table allow us to choose ($\alpha = 0.2, \beta = 0.5$) as optimal parameter settings for the reference model and ($k = 40, \alpha = 0.2, \beta = 0.8$) for the class model. Using these settings on the independent test data we obtain perplexity values of 71 for the reference model and 53 for the class model. These figures, which are very similar to those obtained on the development data, indicate that 25% reduction in perplexity has been achieved by clustering symbols before performing PDFA inference.

As a point of comparison we repeated the experiments using a trigram model with the same smoothing technique. Interpolating the unsmoothed trigram model with the unigram model (optimal $\beta = 0.5$) resulted in a test set perplexity of 59, a figure slightly inferior to the optimal value for the class model. The best known smoothed trigram model, due to Kneser and Ney [8], yields a perplexity of 14. This difference in performance indicates clearly that smoothing will be a key issue in improving the performance of learned automata.

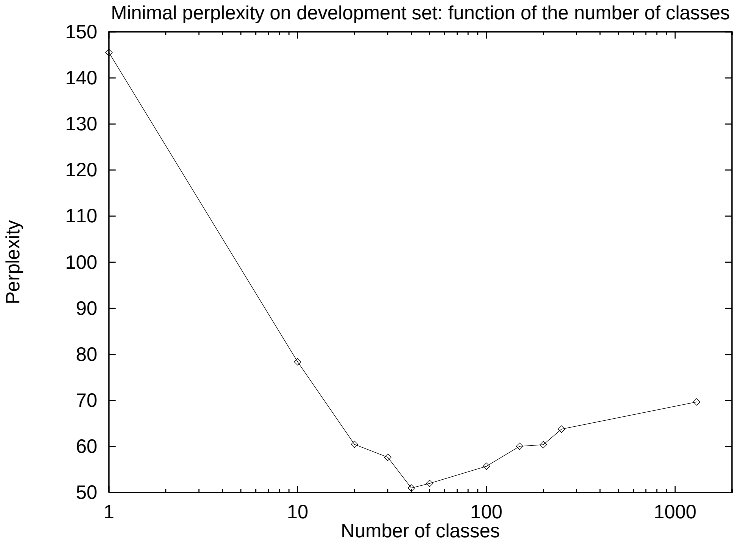

Fig. 8. Best development set perplexities as a function of the number of classes.

k	α	β	PP
1	-	0.0	145
10	0.1	0.8	78
20	0.1	0.8	60
30	0.1	0.8	58
40	0.2	0.8	51
50	0.1	0.8	52
100	0.2	0.7	56
150	0.2	0.6	60
200	0.2	0.6	60
250	0.2	0.6	64
1300	0.2	0.5	70

Table 1. Best development set perplexities depending on k, α, β.

8 Future work

We show in this paper that clustering alphabet symbols before PDFA inference reduces perplexity on new data. Here the clustering algorithm does not depend on the inference procedure. A potential extension would use the infered PDFA to update the clustering. Indeed compatible states define sets of compatible symbols. This property would allow to iteratively adapt the clustering while adapting the precision parameter of the ALERGIA algorithm.

Clustering the alphabet symbols can be considered as a particular type of smoothing. We believe that smoothing is the central issue if we are to improve the performance of learned PDFAs, especially for tasks which face serious data sparseness problems, as in the case of human language systems. Extensive work on smoothing N-gram models has been done in the last decade (see for instance [6, 8, 11]). Adaptation of these smoothing mechamisms to automata that cannot be reduced to N-grams is one of our major goals.

References

1. N. Abe and M. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.
2. P. Brown, V. Della Pietra, P. de Souza, J. Lai, and R. Mercer. Class-based N-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.
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. Springer Verlag, 1994.
4. L. Hirschman. Multi-site data collection for a spoken language corpus. In *Proc. of DARPA Speech and Natural Language Workshop*, pages 7–14, 1992.
5. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
6. S.M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustic, Speech and Signal Processing*, 35(3):400–401, 1987.
7. M.J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R.E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, pages 273–282, 1994.
8. R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *International Conference on Acoustic, Speech and Signal Processing*, pages 181–184, 1995.
9. K. Lang. Merge order counts. Technical report, NEC Research Institute, September 1997.
10. K.J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *5th ACM workshop on Computational Learning Theory*, pages 45–52, 1992.
11. H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.
12. H. Ney and R. Knesser. Improved clustering techniques for class-based statistical language modelling. In *European Conference on Speech Communication and Technology*, pages 973–976, Berlin, 1993.
13. 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, 1992.
14. D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic automata. to appear in Journal of Computer and System Sciences.
15. 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.
16. B. Trakhtenbrot and Ya. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Pub. Comp., Amsterdam, 1973.