Position Models and Language Modeling*

Arnaud Zdziobeck and Franck Thollard

Université de Lyon
Université Jean Monnet, Saint-Étienne
Laboratoire Hubert Curien, UMR CNRS 5516
arnaud.zdziobeck@bvra.univ-st-etienne.fr,
thollard@univ-st-etienne.fr

Abstract. In statistical language modelling the classic model used is n-gram. This model is not able however to capture long term dependencies, *i.e.* dependencies larger than n. An alternative to this model is the probabilistic automaton. Unfortunately, it appears that preliminary experiments on the use of this model in language modelling is not yet competitive, partly because it tries to model too long term dependencies. We propose here to improve the use of this model by restricting the dependency to a more reasonable value. Experiments shows an improvement of 45% reduction in the perplexity obtained on the Wall Street Journal language modeling task.

1 Introduction

In statistical language modelling, the n-gram model is broadly used. This model allows to model dependencies of size n. By using non shadowing smoothing, the n-gram model will also capture dependencies smaller than its order n.

With the increase of the size of the corpora, researchers attempt to increase the size of the history in n-gram models. Unfortunately, the data sparseness problem is getting more an more important as the size of the order of the model (*i.e.* the history) augments. Moreover the size of the model dramatically increases, even if some pruning techniques can be applied with a small impact on the prediction power of the model [15,16].

As the improvements with the n-gram model tend to be harder and harder to obtain, another strategy could be to consider other models. An alternative is to use probabilistic automata, which, in general, have a greater expressive power. On the contrary to the non probabilistic case, non-deterministic automata have a greater power of expression than the deterministic one. See [20] for a presentation of probabilistic finite state machines and their associated algorithms. As the use of non-deterministic machines is much more complex (*e.g.* parsing for example is much efficient in the non deterministic case), most of the work has been concentrated on deterministic ones.

Even if many algorithms exist for inferring deterministic probabilistic finite state automata (DPFA for short), their successful application mainly appears in

* This work was supported by the BINGO2 project (ANR-07-MDCO 014-02).

N. da Vitora Lobo et al. (Eds.): SSPR&SPR 2008, LNCS 5342, pp. 76–85, 2008.

contexts with small vocabularies and short sentences (*e.g* character recognition [14], shallow parsing [18]). Unfortunately, work is still needed in the domain of statistical language modelling as size of the vocabulary is generally large, and more importantly, sentences length are, on average, quite long.

We think that DPFA can be accurate up to a moderate length, that is on part of sentences. We follow here a windowing approach in which the sentences to model are split into windows of a given size, and a model is learned on each window. The parsing is then made using all the automata.

Next section presents the notations used. We then present some preliminary experiments that demonstrate the relevance of modeling strings by chunks. Our experimental set up will follow together with the results. We then conclude.

2 Definitions and Notations

Let Σ be a *finite alphabet* and $\Sigma^\star$ (resp. Σ^+) be the set of all strings that can be built from Σ, including (resp. not including) the empty string denoted by λ. We denote $\Sigma' = \Sigma \cup \{\lambda\}$. A *language* is a subset of $\Sigma^\star$.

A *stochastic language* $\mathcal{D}$ is a probability distribution over $\Sigma^\star$. We denote by $P_\mathcal{D}(x)$ the probability [1] of a string $x \in \Sigma^\star$ under the distribution $\mathcal{D}$. The distribution must verify $\sum_{x\in\Sigma^\star} P_\mathcal{D}(x) = 1$. If the distribution is modeled by some syntactic machine $\mathcal{A}$, the probability of x according to the probability distribution defined by $\mathcal{A}$ is denoted $P_\mathcal{A}(x)$.

A *sample* S is a multi-set of strings: as samples are usually built through sampling, one string may appear more than once. The number of times a string x (resp a symbol x_i) appears in S is denoted $|S|_x$ (resp. $|S|_{x_i}$). The number of symbols $x_i \in \Sigma$ in S is denoted by $||S||$.

The goal of a statistical language model consists in providing a probability to strings of $\Sigma^\star$. If the length of x is know, this can be estimated using the chain rule:

$$P(x = x_1 x_2 \dots x_m) = P(x_1) \times \prod_{l=2}^{m} P(x_l \mid x_1 \dots x_{l-1}) \tag{1}$$

This decomposition requires the estimate of a long dependency term $P(x_l \mid x_1 \dots x_{l-1})$.

In the following, we recall the n-gram model in which a bound on the history is assumed, and the probabilistic automata which can model unbound dependencies.

2.1 The n-Gram Model

The n-gram assumption considers that

$$P(x_l \mid x_1 \dots x_{l-1}) \sim P(x_l \mid x_{l-(n-1)} \dots x_{l-1})$$

[1] As usual, we denote $P(x)$ as $P(X = x)$ for any discrete random variable X.

that is that the probability of a symbol depends only on the $n-1$ preceding ones.

Each probability is estimated using maximum likelihood:

$$P(x_l \mid x_{l-(n-1)} \dots x_{l-1}) = \frac{|S|_{x_{l-n+1}x_{l-n+2}\dots x_l}}{|S|_{x_{l-n+1}x_{l-n+2}\dots x_{l-1}}}$$

A special case of the n-gram model is the unigram model ($n = 1$), noted $\mathcal{U}$: for each symbol $x_i \in \Sigma$, $P_{\mathcal{U}}(x_i) = \frac{|S|_{x_i}}{||S||}$. This model can be represented as a one state automaton, with a looping transition for each symbol of the alphabet.

When dealing with large vocabularies, the denominator of the previous equation can have a null estimate. Hence a smoothing strategy needs to be set in order to provide a non null probability on each n-gram.

2.2 Probabilistic Automata

Definition 1. *A Deterministic Probabilistic Finite state Automaton* DPFA *is a tuple* $\mathcal{A}=\langle Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, q_0, p_{\mathcal{A}}\rangle$*, where:*

- $Q_{\mathcal{A}}$ *is a finite set of* states*;*
- $q_0 \in Q_{\mathcal{A}}$ *is the* initial state*;*
- Σ *is* the alphabet*;*
- $\delta_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma \to Q_{\mathcal{A}}$ *is a* transition function*;*
- $p_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma' \to \mathbb{R}^+$ *are* transition probabilities *such that*

$$\forall q \in Q_{\mathcal{A}}, \qquad \sum_{a \in \Sigma'} p_{\mathcal{A}}(q, a) \;=\; 1. \tag{2}$$

For each state q, the probability $p_{\mathcal{A}}(q, \lambda)$ is not associated with a transition. It represents the probability that the string ends at q.

Functions $\delta_{\mathcal{A}}$ and $p_{\mathcal{A}}$ can be extended recursively from Σ to $\Sigma^\star$.

The probability of a string s according to a DPFA $\mathcal{A}$ is then defined as: $P_{\mathcal{A}}(s) = p_{\mathcal{A}}(q_0, s) \times p_{\mathcal{A}}(\delta_{\mathcal{A}}(q_0, s), \lambda)$ if $s \neq \lambda$ and $P_{\mathcal{A}}(\lambda) = p_{\mathcal{A}}(q_0, \lambda)$ else. If (2) holds, these probabilities define a probability distribution $\mathcal{D}_{\mathcal{A}}$ over $\Sigma^\star$ [20].

This definition of DPFA considers only *complete* automata, *i.e.* functions p and δ are defined for all pairs of state and symbol. If an automaton is not complete, then transitions with zero probabilities can be added to get a complete automaton without changing its distribution $\mathcal{D}_{\mathcal{A}}$.

The automaton considered being deterministic and the size of the vocabulary being constant through the paper, the size of a DPFA $\mathcal{A}$ is defined as its number of states and will be noted $|\mathcal{A}|$.

Figure 1 presents a smoothed DPFA, smoothing being drawn using dash lines. The probability of string cb according to this automaton is $\frac{8}{32} \times \frac{16}{32} \times \frac{16}{16}$.

3 Building Language Models

As seen in section 2.1, the n-gram model is estimated by counting term frequencies in a training corpus. We address in this section the building of probabilistic

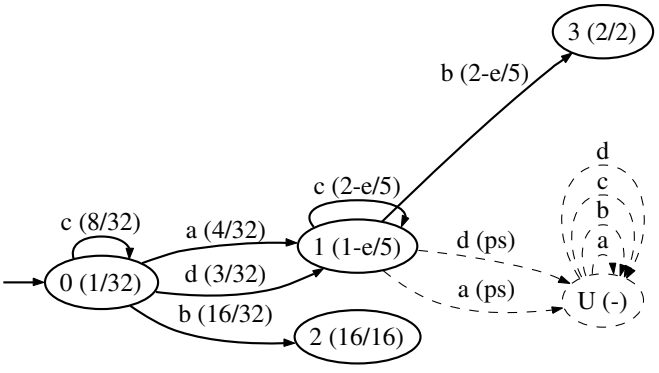

Fig. 1. Probabilistic automaton with a smoothed state

automata. In most of the probabilistic grammatical inference algorithm a first tree shape model is built that represents the maximum likelihood estimate of the training data. A second step is then used to generalize this model. This is done by merging states that are considered close according to some distance and some tolerance parameter α [2,10,19]. The algorithms differ in the distance used. We will consider here the MDI algorithm [19] as i) it uses a global criterion for comparing state and ii) the merging criterion favors small models. The automata produced are thus quite compact as compare to the ones output by the other algorithms. For our purpose, the MDI algorithm can be seen as a black box that takes as input a training set and a tuning parameter α and provides, as the output, a probabilistic automaton.

4 Smoothing Language Models

As mentioned above, smoothing language models is needed to handle the sparse data problem. This is mandatory either from a practical point of view [4,7] or from the theoretical one [13,14] . The main strategy consists in removing some probability mass from the seen events and redistribute them on the unseen ones. A back-off model, usually less accurate but more general, is used to provide a redistribution of the probability mass more realistic than, say, a uniform one. The classic smoothing scheme for n-gram is the Kneyser-Ney smoothing [11] while smoothing probabilistic automata is still an widely open problem (see [6,17] for some attempts).

In [17] smoothing a probabilistic automaton is done at a state level. A value ϵ is removed from each transition. This leaves a probability mass that can be redistributed, according to the unigram model, to estimate the probability of the symbol for which no transition exists. In order to keep the model compact, this smoothing scheme is done dynamically at parsing time. As an illustration, state 1 of figure 1 is explicitly smoothed: two smoothed transitions are added

and goes to the unigram state $\mathcal{U}$. The ps values are estimated with the unigram adequately renormalized by $\varepsilon/5 + \varepsilon/5 + \varepsilon/5$, as this represents the probability mass saved respectively from the stopping probability, and the outgoing original transitions "c" and "b".

5 Motivations

We made some preliminary experiments on the "*Air Travel Information System*" (ATIS) corpus[8]. Even though it is now considered a small corpus it is large enough to make relevant preliminary work. In Fig. 2 we compute the average probability estimated by the unigram model, a probabilistic automaton inferred by the MDI algorithm [19] and a trigram model smoothed by Kneyser-Ney smoothing [11]. As can be seen, apart from the very beginning and the very end of the sentences[2], it appears that, the unigram and the trigram performances are not much influenced by the position in the sentence. This is not the case however for the probabilistic automaton as its prediction tends to the unigram one as the position increases. What can be noticed however is the fact that both the trigram and the automaton have a peak of performance at the same point (namely at position 3). The curve labelled "*raw aut*" is obtained by averaging the non null probabilities provided by a non smoothed automaton, that is the probabilities obtained when no smoothing is required. It is interesting to see that this predictions are, on average, rather good.

This means that, when it can parse, the automaton is rather good. An interpretation of the difference between this curve and the one of the smoothed automaton, is that the automaton suffers from under-generalization, and the parsing at that point is usually done by the unigram model. Unfortunately, tuning the grammatical inference parameter in order to build a more general automaton will decrease the prediction performances at the beginning of the sentence. In order to circumvent this problem, we propose here to build a model for which the generalization will depend on the position. As the models inferred by MDI are quite compact (few hundred of states), we can combine several DPFA without fear of facing any storage problem.

6 Experiments

The analysis above were conduct on the small ATIS task. In order to evaluate the position model on a more up to date database, we now consider the Wall Street Journal database.

6.1 The Wall Street Journal Task

The data used are drawn from the Wall Street Journal database [12], a large syntactically annotated corpus subdivided in 25 sections. We followed the

[2] Before position 3, the trigram does not have information to model properly conditional probabilities. Above position 23, very few strings exists (namely 34) that makes any interpretation suspicious.

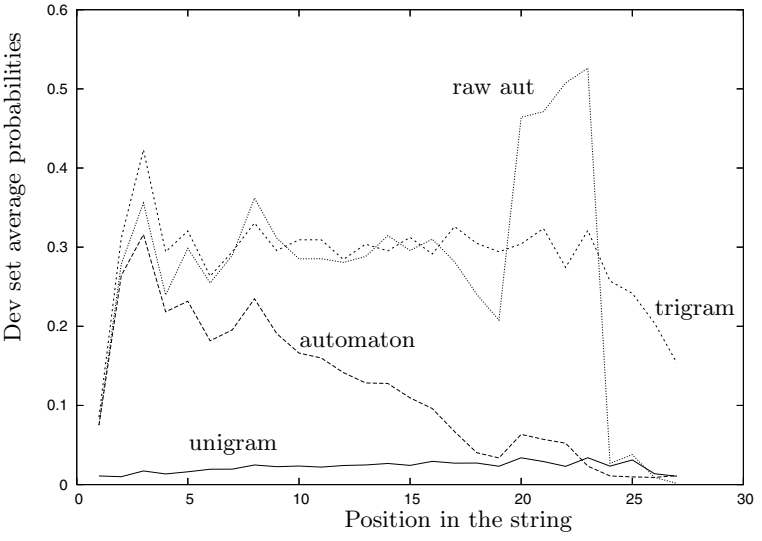

Fig. 2. Prediction power w.r.t the position

preprocessing used by [3]: Sections 0 to 20 were used as the training set (962,612 words), sections 21 and 22 were used as a development set (48,024 words) and sections 23 and 24 serve as the test (101,189 words). Digit numbers were replaced by a unique character. As in [3], the 10,000 most frequent words were kept and the remainder were transformed in a unique symbol `unknown`. In order to make the data more realistic for a speech to text task, the punctuation was removed and the words transformed in their lower case form. The average length of the sentences is 22 %($\pm$ 10) words and the size of Σ is around 10000.

6.2 Experimental Protocol

We present here the experimental protocol we followed. According to the Fig. 2 we consider that probabilistic automata are not able to handle cleverly histories of size greater than 10. Each string is thus split in different chunk (position 1 to 10, 8 to 15, ...). A model is inferred on each chunk and the final parsing will change the model as the position changes to another chunk. As can be seen on fig 2, the parsing at the very beginning of the sentence do not have enough evidence to provide good estimate. Moreover, as the position increases, smoothing is more likely to appear. For theses reasons, we propose to make an overlap between the chunks.

Defining the size of the overlap is not easy, as there is no reason that a good overlap size at the beginning of the string will be relevant at the end of the string. We therefore propose to have a lot of concurrent overlaps and to learn which one is relevant and which one is not.

The strings are thus split using different overlapping chunks (*e.g* 1-5, 1-6, 1-7, ..., 1-15, 3-11, 3-12, ...). For each chunk we inferred a probabilistic automaton[3]. A given position (say 8) will be covered by different chunks (*e.g.* 6-10 and 7-12). We make chunks of different size (ranging from 5 to 15 words) at different position (ranging from 0 to 58). Given the small amount of data at far position, a unique automaton is inferred from position 61. We will note C^i (resp. A^i) the set of chunks that covers position i (resp. the set of automata inferred on chunk C^i). Obviously, all the automata in A^i do not have the same accuracy in their probability estimate: as seen on Fig. 2 depending on the position in the automaton during the parsing, the probability estimated by the model do not have the same accuracy. The probability estimated at position i by our position model will be an interpolation of the probabilities provided by the automata in A^i. We note λ_j^i the weight of an automaton $A_j^i \in A^i$

More formally, the probability of a string will be computed as :

$$P(x = x_1..x_n) = \prod_{pos=i}^{n} \sum_{A_j^i \in A^i} \lambda_j^i \times P_{A_j^i}(x_i|x_1...x_{i-1})$$

The values of the λ_j^i coefficients are evaluated using an EM procedure.

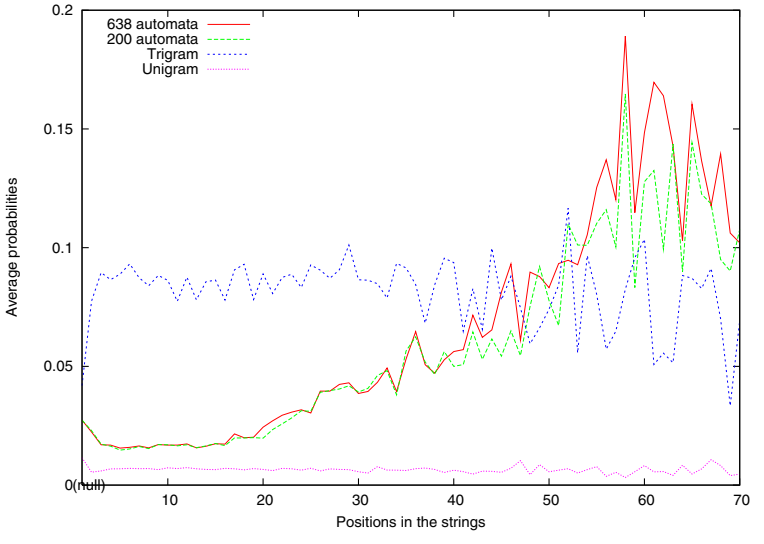

Fig. 3. Prediction power w.r.t the position

Fig. 3 shows the average probability of what we call the position model (curve named "638 auts") at different positions. As a point of comparison, the average probability is also provided for the trigram model and the automaton inferred

[3] Here again, a set of inferences is done by varying the MDI parameter α from 0.0008 to 0.1. The best automaton according to its dev set performance is selected.

on the full strings. It is interesting to notice that the position model starts to be relevant from position 20 and outperforms the trigram from position 40. Unfortunately, as the number of strings of length greater than 40 is quite small, this performance does not compensate the bad results obtained at the beginning: the position model is outperformed on average by the trigram. Another point should nevertheless be considered: despite the fact we used many automata, the overall size of our model is still quite reasonable as it is far smaller than the size of the trigram. Moreover, it appears that some of the λ_j^i are quite small as compare to other ones. We thus propose to reduce the size of the overall position model by iteratively removing the automaton that has the smallest weight.

Fig. 4 shows the behavior of the position model when we remove some automata. Note that on the contrary to the over figures, the quality criterion used is the perplexity: the smaller the perplexity on a given data set, the better the model predicts the strings of this set. This measure is linked, on the one hand, with the Kullback-Leibler divergence between the model and the development set, and on the other hand, on the inverse of the (geometric) average of the probabilities assigned by the model. See *e.g.* [20] for more detail on this measure and its links to the average on the test set string probabilities.

On Fig. 4 we can see that the our pruning strategy can save up to 50% of the size of the position model with a small impact on the development set perplexity (curve "200 auts" on fig 3).

For sake of comparison, the table 1 presents the *test set* perplexity of the position model as compare with the Trigram model (with a Kneyser-Ney smoothing).

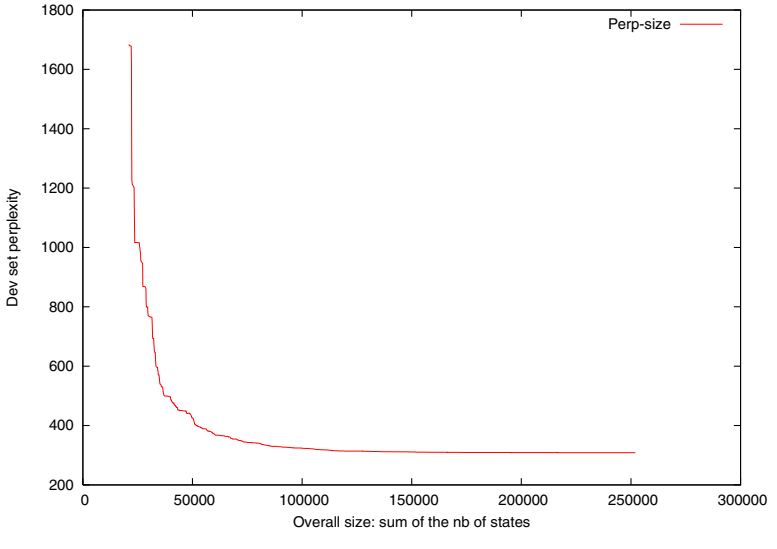

Fig. 4. Perplexity w.r.t the model size

We include the size of the model [4] and see first the that more compact model is the automaton inferred on the whole data set (name "full aut"). This model however performs rather badly as its perplexity is around 550 (as compare to the 150 of the trigram). The "All aut" model is the one in which all the 638 automata were kept. Using this model we obtain 45% of perplexity reduction, the size of the model being reasonable. When concern with the size of the model, we can see that the 200 auts model is a good tradeoff between size and prediction as the gain in size (w.r.t the All aut model) is interesting as regard to the prediction performances.

Table 1. Test set perplexity on the WSJ task

Model	#states	Perp (test)
3-grams	1,589,844	150
Full Aut	451	556
100 auts	32,609	399
200 auts	54,083	381
All aut	252,026	364

7 Conclusion and Further Work

We presented in this article a position model for statistical language modeling using DPFA. This position model is evaluated on the Wall Street Journal task. It achieve a test set perplexity reduction of 45% at the expense of having a bigger model. This model is still smaller than the trigram one.

In the near future, we want to consider the tradeoff between the size of the model and the prediction power. To this end, some trigram compression method (*e.g.* [9]) or trigram pruning methods (such as [16,15]) will be compare to the automaton's size, adequately compacted by compressing method [5]).

Another improvement can come from the use of other more recent probabilistic grammatical inference algorithms such as [1] and other smoothing schemes [6].

References

1. Callut, J., Dupont, P.: Learning partially observable markov models from first passage times. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS, vol. 4701, pp. 91–103. Springer, Heidelberg (2007)
2. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Second ICGI, pp. 139–152 (1994)
3. Charniak, E.: Immediate-head parsing for language models. In: 10th Conf. of the Association for Computational linguistic, ACL 2001 (2001)
4. Chen, S.F.: Building Probabilistic Models for natural Language. PhD thesis, Harvard University Cambridge Massachusetts (May 1996)

[4] The size of the trigram model is computed as the sum of the number of entries of the different hash tables.

5. Daciuk, J., van Noord, G.: Finite automata for compact representation of language models in NLP. In: Watson, B.W., Wood, D. (eds.) CIAA 2001. LNCS, vol. 2494, pp. 65–73. Springer, Heidelberg (2003)
6. Dupont, P., Amengual, J.C.: Smoothing probabilistic automata: an error-correcting approach. In: Oliveira, A.L. (ed.) ICGI 2000. LNCS, vol. 1891, pp. 51–64. Springer, Heidelberg (2000)
7. Goodman, J.: A bit of progress in language modeling. Technical report, Microsoft Research (2001)
8. Hirschman, L.: Multi-site data collection for a spoken language corpus. In: DARPA Speech and Natural Language Workshop, pp. 7–14 (1992)
9. Kenneth, C., Ted, H., Jianfeng, G.: Compressing trigram language models with Golomb coding. In: Joint EMNLP-CoNLL, pp. 199–207 (2007)
10. Kermorvant, C., Dupont, P.: Stochastic grammatical inference with multinomial tests. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS, vol. 2484, pp. 149–160. Springer, Heidelberg (2002)
11. Kneser, R., Ney, H.: Improved backing-off for m-gram language modeling. In: Intl. Conf. on Acoustic, Speech and Signal Processing, pp. 181–184 (1995)
12. Marcus, M., Santorini, S., Marcinkiewicz, M.: Building a large annotated corpus of English: the Penn treebank. Computational Linguistics 19(2), 313–330 (1993)
13. McAllester, D., Shapire, R.: On the convergence rate of the good-turing estimators. In: Conf. on Computational Learning Theory, pp. 1–66 (2000)
14. Ron, D., Singer, Y., Tishby, N.: On the learnability and usage of acyclic probabilistic finite automata. In: Proceedings of COLT 1995, pp. 31–40 (1995)
15. Siivola, V., Hirsimäki, T., Virpioja, S.: On growing and pruning kneser-ney smoothed n-gram models. IEEE Transactions on Audio, Speech and Language Processing 15(5), 1617–1624 (2007)
16. Stolcke, A.: Entropy-based pruning of backoff language models. In: DARPA Broadcast News Transcription and Understanding Workshop, pp. 270–274 (1998)
17. Thollard, F.: Improving probabilistic grammatical inference core algorithms with post-processing techniques. In: ICML 2001, pp. 561–568. Morgan Kaufmann, San Francisco (2001)
18. Thollard, F., Clark, A.: Shallow parsing using probabilistic grammatical inference. In: Adriaans, P.W., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS, vol. 2484, pp. 269–282. Springer, Heidelberg (2002)
19. Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: ICML (2000)
20. Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., Carrasco, R.C.: Probabilistic finite-state machines – Part I and II. PAMI 27(7) (2005)