Shallow Parsing using Probabilistic Grammatical Inference

Franck Thollard and Alexander Clark

EURISE, Université Jean Monnet,23, Rue du Docteur Paul Michelon, 42023 Saint-Etienne Cédex 2, France
ISSCO/TIM, Université de Genève 40, Bvd du Pont d'Arve CH-1211, Genève 4, Switzerland

Abstract. This paper presents a machine learning approach to shallow parsing using techniques of grammatical inference. We first learn a deterministic probabilistic automaton that models the joint distribution of chunk and Part-of-speech tags, and then use this automaton as a transducer to find the most likely chunk tag sequence using a dynamic programming algorithm. The resulting transducers can also be combined with statistical *POS* taggers. We also discuss an efficient means of incorporating lexical information together with an application of *bagging* that improve our results.

Keywords: Probabilistic Grammatical Inference, Shallow Parsing, Bagging

Shallow parsing of natural language is the task of dividing sentences into a sequence of simple phrases. Many applications of this can be found. Among them are (i) *indexing web pages*: as users mainly enter noun phrases into internet search engines, detecting noun phrases leads to a better indexing of web pages and (ii) as a preliminary to full parsing, *i.e.* building a complete constituent structure tree.

We consider as input a sequence of words together with their part of speech *POS* tags; the task of shallow parsing is then one of adding an additional tag that indicates the beginning and ends of the various types of constituents (Noun Phrase, Verb Phrase, etc.). These tags will be called *Chunk* tags. Initially we shall consider just the mapping from POS tags to *Chunk* tags; later we will consider how to incorporate lexical knowledge.

Our approach is to infer a deterministic probabilistic automaton that models the joint sequence of POS tags and *Chunk* tags. We use a grammatical inference algorithm for this task. We then consider it as a probabilistic transducer that models the mapping from POS tags to *Chunk* tags. This interpretation will be done with the Viterbi algorithm [9]. Since for each word we have one POS tag, and one *Chunk* tag, the resulting transducers are fully aligned which simplifies the algorithms considerably.

We start by presenting the data we will use; we then describe our approach in general terms. The probabilistic grammatical inference algorithms are then introduced. We describe how these automata can be used as transducers, and then present the experimental results we have obtained. We conclude with a brief discussion.

1 The problem

This task consists in constructing a **flat** bracketing of the sentences. For the sentence *"He reckons the current account deficit will narrow to only # 1.8 billion in September."*, the goal will be to define the following brackets[1] *[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September] .*

The data used are drawn from the Wall Street Journal database [14], a large syntactically annotated corpus. Section 15 to 18 were used as the learning set (211727 words) and section 20 as the test set (47377 words). The most frequent *Chunks* are Noun Phrases (51%), Verb Phrases (20%) and Prepositional Phrases (20%). The data is constructed by taking the words, adding the *Chunk* tags that can be deduced from the parse trees in the corpus, and using POS tags obtained by the Brill tagger [4], not the "correct" POS tags drawn from the corpus. The motivation for this is that it mimics more closely real-world situations. The task then is to identify the *Chunk* tag given the word and its POS tag. The interpretation of the *Chunk* tags is as follows: for a *Chunk* C, B-C will mean Begin *Chunk* C and I-C In *Chunk* C. Beginning a given *Chunk* will automatically ends

[1] NP stands for Noun Phrase, VP for Verb Phrase et PP for Preposition.

the previous one. In addition, the symbol O means that the given word is outside any *Chunk*. There are many different approaches [19]. Here is a concrete example:

He	PRP	B-NP	....		
reckons	VBZ	B-VP	billion	CD	I-NP
the	DT	B-NP	in	IN	B-PP
current	JJ	I-NP	Sept	NNP	B-NP
....			.	.	O

2 Our point of view

The long-term goal is to build a complete system that combines the different levels of structure: the lexicon, the POS, the *Chunks*, and the other levels. This strategy is used for example in [1]. In our approach, each level of representation is dealt with by *one* model. Here we assume we already have the POS tags and concern ourselves with building the POS to *Chunk* model.

We will first study the task using just the POS tags to predict the *Chunk* ones. The inclusion of the lexical information is addressed later on in the paper. POS tagging has been widely studied in the literature (see *e.g* [20, 4, 2] for different approches to POS tagging). The problem is then to find a mapping between the POS sequence and the *Chunk* sequence.

If $T_{1,n}$ (resp. $C_{1,n}$) denotes the POS 1 to n, (resp. *Chunks* 1 to n), the goal is to find the most probable set of *Chunks* 1 to n given the POS 1 to n. This can be written:

$$Chunk(T_{1,n}) = \arg\max_{C_{1,n}} P(C_{1,n}|T_{1,n}) = \arg\max_{C_{1,n}} \frac{P(C_{1,n},T_{1,n})}{P(T_{1,n})} = \arg\max_{C_{1,n}} P(C_{1,n},T_{1,n})$$

The problem is then to estimate the joint probability $P(C_{1,n},T_{1,n})$. A possible decomposition of $P(C_{1,n},T_{1,n}) = P_{1,n}$ is:

$$\begin{aligned}
P_{1,n} =&\ P(T_1)\ P(C_1|T_1)\ P(T_2|C_1,T_1)\ P(C_2|T_{1,2},C_1) \times \\
&\ldots \times P(T_n|T_{1,n-1},C_{1,n-1})\ P(C_n|T_{1,n},C_{1,n-1}) \\
=&\ \textstyle\prod_{i=1}^{n} P(C_i|C_{1,i-1},T_{1,i})\ P(T_i|T_{1,i-1},C_{1,i-1}) \\
=&\ \textstyle\prod_{i=1}^{n} P(C_i|C_{1,i-1},T_{1,i-1},T_i)\ P(T_i|C_{1,i-1},T_{1,i-1}) \\
=&\ \textstyle\prod_{i=1}^{n} \frac{P(C_i,T_i|C_{1,i-1},T_{1,i-1})}{P(T_i|C_{1,i-1},T_{1,i-1})}\ P(T_i|C_{1,i-1},T_{1,i-1}) \\
=&\ \textstyle\prod_{i=1}^{n} P(C_i,T_i|C_{1,i-1},T_{1,i-1})
\end{aligned}$$

We therefore aim at finding the set of *Chunks* such that the pairs (POS, *Chunk*) are the most probable given the history. We can denote each pair by joining the POS and the *Chunks* in a single symbol.

If we use the "+" symbol as the separator, our learning data will look like :
`PRP+B-NP VBZ+B-VP DT+B-NP JJ+I-NP ...CD+I-NP IN+B-PP NNP+B-NP .+O`

Many models exist to estimate the probability of a symbol given a history. An example is the n-gram model and all the improvements applied to it: smoothing techniques (see [6] for a recent survey) and variable memory models [20, 16].

Another approach would be to infer Hidden Markov Models [21] or probabilistic automata [11, 5, 17, 22]. Among these techniques only four infer models in which the size of the dependency is not bound [21, 5, 22]. The first one is too computationally expensive to be feasible with our data. The others have been used in the context of natural language modeling. We have applied these algorithms to this task but only the best performing one (namely DDSM [22]) will be presented here. Reassuringly, the algorithm that performs best on the natural language modeling task is also the one that performs best on the noun phrase chunking task.

We now present formally the model followed by the description of the DDSM algorithm.

3 The learning algorithm

We first present the formal definition of the model and the inference algorithm.

A *Probabilistic Finite Automaton* (PFA) A is a 5-tuple $(\Sigma, Q^A, Q^{I^A}, \xi^A, F^A)$ where Σ is the alphabet (a finite set of symbols), Q^A is the *set of states*, $Q^{I^A} \subseteq Q^A$ is the *initial* states, ξ^A is a multiset of probabilistic transitions $t \in Q^A \times \Sigma \times Q^A \times (0,1]$. F^A: $Q^A \to [0,1]$ is the "end of parsing" probabilistic function.

We require that for all states q, $\sum_\sigma \xi^A(q, \sigma, q') + F^A(q) = 1$. We assume that all states can generate at least one string with a strictly positive probability. This then defines a distribution over Σ^*.

We now define δ^A and γ^A from $Q^A \times \Sigma$ to Q^A and $]0, 1]$ respectively:

$$\begin{aligned} \delta^A(q_i, x) &= q_j \in Q^A : \ \xi^A(q_i, x, q_j)\text{exists} \\ \gamma^A(q_i, x) &= p : \exists q_j : \ \xi^A(q_i, x, q_j) = p \end{aligned} \tag{1}$$

Let I_+ denote a *positive sample*, *i.e.* a set of strings belonging to the 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 related to the number of times it is used while generating, or equivalently parsing, the positive sample. 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 $C(q, \#)$ denote the number of times a string of I_+ ended on q. 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 $\hat{\gamma}(q, a) = \frac{C(q,a)}{C(q)}$, $a \in \Sigma \cup \{\#\}$.

Fig. 1. PPTA built with $I+ = \{$aac,λ, aac, abd, aac, aac, abd, abd, a, ab, $\lambda\}$

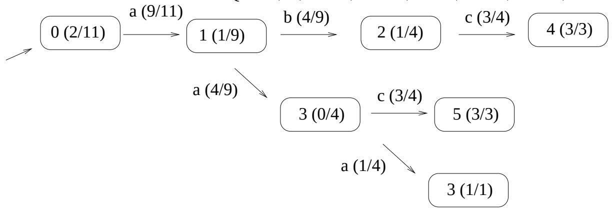
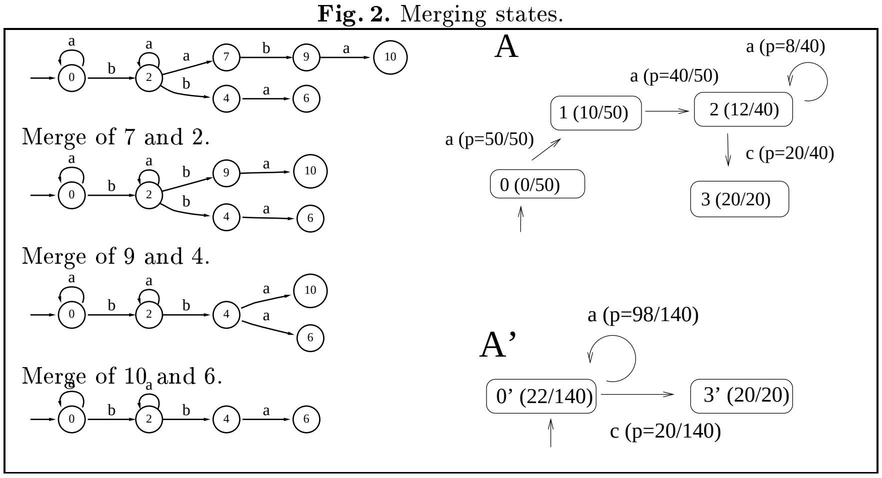


Figure 1 exhibits a $PPTA(I+)$.

We now present the second tool used by the generic algorithm: the state merging operation. This operation provides two modifications to the automaton: (i) it modifies the structure (figure 2, left part) and (ii) the probability distribution (figure 2, right part). It applies to two states. Merging two states can lead to non-determinism. The states that create non determinism are then recursively merged. When state q results from the merging of the states q' and q'', the following equality must hold in order to keep an overall consistent model:

$$\gamma(q,a) = \frac{C(q',a)+C(q'',a)}{C(q')+C(q'')} \ , \forall a \in \Sigma \cup \{\#\}$$

Fig. 2. Merging states.



One can note two properties of the update of the probabilities: first, $\frac{C(q',a)+C(q'',a)}{C(q')+C(q'')}$ is included in $[\frac{C(q',a)}{C(q')}, \frac{C(q'',a)}{C(q'')}]$ which means that, the probability of an after merge transition has a value bounded by the two values of the transitions it comes from. Secondly, $\frac{20+100}{1000+105} = \frac{120}{1105}$ is closer to $\frac{20}{1000}$ than to $\frac{100}{105}$. The merge naturally weights more the probability of the transition that holds the more information.

These remarks hold for each pair of transitions that takes part in the merge. Let us merge states q_i and q_j and define P_{q_i} (resp. P_{q_j}) as the probability distribution defined by considering state q_i (resp. q_j) as the initial state. Since the merge is recursively applied (see figure 2), the probability distribution after merging states q_i and q_j will be a weighted mean between the distributions P_{q_i} and P_{q_j}.

The next section address now the description of the DDSM algorithm itself.

3.1 The DDSM algorithm

The DDSM [22] algorithm (algorithm 1) takes two arguments: the learning set I_+ and a tuning parameter α. It looks for an automaton tradeoff between a small size and a small distance to the data. The distance measure used is the Kullback-Leibler divergence. The data are represented by the PPTA as it is the maximum likelihood estimate of the data. While merging two states, the distance between the automaton and the data increases and, at the same time, the number of states and the number of transitions, in general, decreases. Two states will be set compatible if the impact in terms of divergence of their merge divided by the gain of size is smaller than the parameter α.

The DDSM algorithm builds as a first step the $PPTA(I_+)$ and then repeats the following operations: choose two states (states q_i and q_j in the code), and computes the compatibility value (Relative_Divergence in the code) induced by their merge. It then performs the best merge if it is compatible. The ordering in which states are chosen is adapted from the EDSM algorithm, which was the winning algorithm of a non probabilistic grammatical inference competition [13].

Algorithm 1: DDSM $(I+, \alpha)$.

```
A ← Numbering_in_Breadth_First_Order(PPTA);
for q_i = 1 to Nb_State (A) do
    best_div = ∞ ;
    for q_j = 0 to i − 1 do
        if Compatible (A, q_i, q_j) < best_div then
            best_pred = q_j ;
            best_div = Compatible (A, q_i, q_j);
    if best_div < α then Merge (A,q_i,best_pred) ;
Return A ;
```

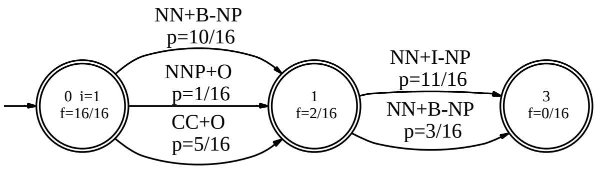
The DDSM algorithm aims at inferring a small automaton that is close to the data. In fact, the bigger α, the more general (and small with respect to the number of states) the resulting automaton should be.

4 Automata as transducers

In this section, we describe how the automaton can be interpreted as an aligned transducer. Let us suppose that the automaton of figure 3 was inferred by the

DDSM algorithm. It can then be considered as a transducer with the same structure which emits symbols on transitions. The transducer transitions are built from the ones of the automaton when taking the POS as the input symbol and the *Chunks* as the output. The transducer is then non-sub-sequential, that is non-deterministic with respect to the input symbols. The automaton of figure 3 can be considered as a fully aligned non sub-sequential transducer.

Fig. 3. probabilistic automaton/probabilistic transducer



We can then use the Viterbi algorithm [9] to find out the most probable path given the input POS tags.

5 Evaluation

The quality measure used is the F-score based on precision and recall [23]. A given *Chunk* is correct if the starting *Chunk* is correct (*e.g.* B-NP for the NP *Chunk)* and all the inside *Chunks* (I-NP for *Chunk* NP) are proposed correctly.

For a *Chunk* C, the recall is the number of times C is correctly proposed divided by the number of times it should be. Raising the recall can be done easily by proposing a given *Chunk* very often. Conversely, the precision computes the number of times a *Chunk* C is truly proposed divided by the number of times it is proposed. The two criteria are usually combined with the function F_β [23] : $F_\beta = \frac{(\beta^2+1).Recall.Precision}{\beta^2.Recall+Precision}$ In the following, we will set $\beta = 1$, as is standard, in order to weight equally the precision and recall.

The original corpus is divided in two sets: the training set (AppTot) (with 8936 sentences and 211727 symbols), and a test set (Test) with 2012 sentences and 47377 symbols. We randomly divided the sentences of the training set into two sets: a training set for the validation step (AppV, 8092 sentences, 191683 symbols) and a validation set (Valid, 844 sentences, 20044 symbols) . AppV contains roughly 90% of the sentences of AppTot. The average number of words per sentence is around 24. The size of the vocabulary (in this case pairs of POS tags and *Chunk* tags) is around 300 (316 for AppV and 319 for AppTot). The PPTA contains roughly 160,000 states when built on the AppV set and 177,000 with AppTot. Our C++ implementation of the DDSM algorithm needs around 30 Megabytes to handle this data. It took from 45 mn (for $\alpha = 0.001$) to 13 hours (for $\alpha = 0.0004$) on a `sun ultra 10` with a 333 MHz processor.

The free parameters were estimated on the corpus (AppV, Valid). A final inference was made on the AppTot learning set with the value of α defined during the validation step. The parsing of the test set was then done using the Viterbi algorithm using the parameter values defined on the AppV set.

We now describe the behavior of the machine on the validation set.

6 Validation set analysis

This section will deal with the analysis of the different techniques we applied, evaluated on the validation set. We first briefly describe the smoothing technique choosen. We then discuss an adaptation of the bagging technique [3]. Finally we describe an efficient method for using lexical information in the learning process.

The smoothing technique The smoothing technique used is inspired from Katz smoothing [10]: a small quantity is discounted from each probability available in the automaton. The probability mass available is then redistributed on the events the automaton cannot handle (*e.g.* parsing NN VP with the transducer of figure 3). The redistribution of the probability mass is made *via* a unigram model to obtain an overall consistent model. The unigram model is built using the same learning set as for the automata inference, extended by an end-of-string symbol. The parsing of the input by a unigram is made using the same technique as for an automaton. The unigram model can also be considered as a one state automaton that contains a loop for each element of the vocabulary.

The discounting value was estimated on the validation set. The discounting is done by subtracting a value to the numerator of the count of each estimated probability. In such a way, the discounted mass gets smaller as the frequency of the transition - and hence the statistical relevance of the probability estimated - increases. The value of this discounting parameter was estimated on the validation set as 0.5. Unlike in other domains such as language modeling, it seems that the tuning of this parameter is not crucial.

The strategy used here smoothes *each* path of the transducer. Even if the best automaton is usually able to parse the input sentence totally, this strategy leads to better results. An interpretation of this phenomenon can be that it is better to trust the choices made at the beginning of the parsing.

Bagging the density estimator The bagging [3] technique has been successfully applied in many domains but was reported not to work on this particular task [18]. On the other hand, it was shown to improve the Collins statistical full parser [7]. Moreover, interpolating PDFA has been reported to improve results significantly [22]. Finally, it has been shown that boosting does not perform better than bagging on the full task. We hence use the bagging method here.

The bagging strategy samples the training set in samples (B-sets) of the same size. A model is then inferred from each B-set and a majority vote or an interpolation is done using all these models. Applying this raw technique does not work well since many strings have a very small probability in the original training set. They hence do not appear in the B-sets. In order to get good B-sets, we increse their size up to 50,000 sentences because, with this size, the size of the B-sets PPTA matches roughly the size of the PPTA built from the original training set. Following this strategy, we know that on the one hand, nearly all the input sentences of the original training sample will be present in the B-sets and, on the other hand, the statistical relevance of the main sentences increases. While bagging the lexicalized training set, we needed to increase again the size of the B-set as going up to 130,000 sentences continue to improve the dev set performances.

Including the lexical information Lexical information is known to be quite powerful. For example, Pla et *al.* [15], who use a statistical finite state machine, improve their results by 4%, raising the $F_{\beta=1}$ from 86 to 90. They choose some POS tags and split them with respect to the word they were matching with. A POS tag T_i is then replaced by the symbol W_iT_i for the word W_i they are tagging. The idea behind that is to resolve the ambiguity of a tag T_i when the word can help. As an example, the word *against* always start an B-PP but its POS tag (namely IN) is ambiguous because it can predict a beginning of preposition (B-PP *Chunk* tag) or a beginning of clause with a complementizer (B-SBAR *Chunk* tag). By adding a new POS tag *against-IN* we will be able to save some problem when the POS tag is IN and when the matching word is *against*. The problem then is to choose which POS tag to split and in which way to do it. One obvious idea is to create a new POS tag for each word/POS pair but this will (i) be intractable and (ii) lead to data sparseness. We thus try to determine automatically which POS tag to split and in which way. We will use a criterion based on mutual information.

Choosing the POS tag We wish to identify which POS tags will be helped by the addition of lexical information. For these POS tags, we would expect there to be a high dependency between the word and the *Chunk* tag. The mutual information between two random variables X and Y is defined as

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{2}$$

We therefore calculate for each POS tag t the mutual information between the conditional *Chunk* distribution and the conditional Word distribution.

$$I(C|T=t; W|T=t) = H(C|T=t) - H(C|W,T=t) \tag{3}$$

We then select the POS for which this is highest. This is the POS which will give us the greatest reduction in the uncertainty of the *Chunk* tag if we know the word as well as the POS tag.

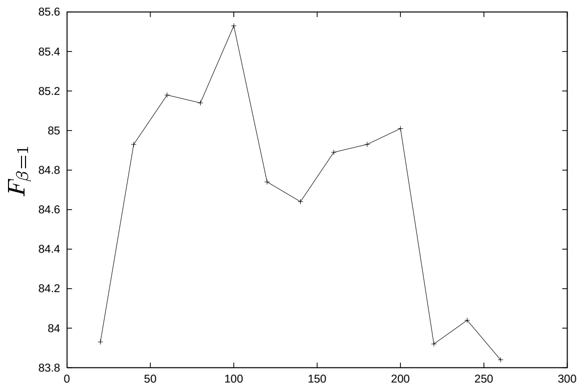
All the probabilities are estimated using the maximum likelihood. To avoid problems in the estimation of this quantity due to data sparseness, we did not take into account rare combinations of word/*Chunk*, *i.e.* the combinations that appears less than six times.

Splitting the POS tag The POS being chosen, we need to create a new item (a combination of word and POS tag) according to the ability of the word to predict the *Chunk*. We here again aim at using an information theoretic criterion. Let W_t be the set of words that have t as a POS. Let Π^t be the set of partitions of W^t. We now aim at finding the partition $\pi^t \in \Pi^t$ that leads to the greatest reduction of the uncertainty of the *Chunk*[2]. Obviously, having an exhaustive search of the set of partitions is not computationally possible. Let us define $\pi_i^t = \{\{w_i\}, \{w_1, ..., w_{i-1}, w_{i+1}, ..., w_n\}\}$. We will choose the partition π_i^t that leads to the minimal conditional entropy of the *Chunk* given the partition:

$$\pi_i^t = \mathrm{Argmin}_{\pi_j^t}\ H(C|\pi_j^t) = \mathrm{Argmin}_{\pi_j^t}\ -\sum_{s \in \pi_j^t} \sum_{c_i \in C} p(c_i, s) \log p(c_i|s)$$

We then repeat the procedure until the right number of splits is reached. Figure 4 shows the relationship between the number of splits and the $F_{\beta=1}$ dev set performance. The optimal number of splits as measured on the dev set (see figure 4, seems to be close to 100 which means that only a small amount of POS tag need to be split in order to perform well. This is helpful because it doesn't change significantly the time response of the learning algorithm.

Fig. 4. Number of splits vs F-score



We also experimented with a more global method of optimisation where we searched for an optimal partition of all <word,POS> pairs, but preliminary experiments indicated that this did not perform well.

7 Results on the test set

This section describes the results obtained on the test set. The task baseline is performed by a unigram model or equivalently a one state automaton. It provides

[2] We actually aim at creating a new item <word,POS>. Since the POS is fixed here, we just need to partition the word set.

an overall $F_{\beta=1}$ of 77. Our results are summarized in table 1. They were obtained on the test set after an inference (on the whole training set) using the value of the parameter α that was optimal on the validation set.

The first column shows the *Chunk*; the remaining columns show the $F_{\beta=1}$ score under various settings: the initial Viterbi technique without lexical information (first column with label I), the results obtained when adding lexical information by splitting the POS tags 100 times, (column L), the bagging technique without lexical information (column B), and finally the results obtained combining the bagging and the lexical information (column LB). The last two columns give some comparable results from other techniqyes applied to this task, *i.e.* a n-gram model without lexical information [8] and a lexicalized model using Hidden Markov Models [15].

Table 1. Comparison of results on the test set $F_{\beta=1}$

Chunk	I	L	B	LB	ngrm	LMM
ADJP	48	49	58	63	55	70
ADVP	63	69	69	74	70	77
INTJ	33	22	40	67	40	100
NP	85	86	89	89	89	90
PP	87	95	90	95	92	96
PRT	9	55	20	67	32	67
SBAR	15	77	16	81	41	79
VP	86	86	89	88	90	92
ALL	83	86	87	89	87	90

The n-gram approach (ngrm column) extends the baseline algorithm by using a context sensitive approach instead of using a unigram. A given *Chunk* is chosen with respect to its POS tag context. The context was defined as a centered window around the current POS tag. In the results outlined here a window of 5 POS tags was taken. The author mentioned that the results do not improve significantly while using a bigger window size. The differences between our approach and the n-gram one are that (i) it does not generalize the data and (ii) given the smoothing strategy applied, the method does not provide a true probability.

With regard to (i), it seems that, even if we generalize, we get roughly the same results. However, it is important to note that the generalization provides rather small automata (roughly 200 states and 8,000 transitions), that are significantly smaller than the n-gram models. The item (ii) is not very serious since the results do not rely on the fact that the model define a true probability distribution. Nevertheless, we note that having a consistent probability distribution allows us to combine our model with other probabilistic approaches cleanly.

The Lexicalized Markov Model (LMM in table 1) takes the 470 most frequent words of the training data (excluding the punctuation, symbols,proper nouns and numbers) and split their respective POS tag [15]. They got an overall improvement of 4% of the $F_{\beta=1}$. The main difference with our approach comes from our automatic detection of the POS tags to split. Our method has the advantage of splitting fewer POS tags (100 comparing to 470). Moreover, since identifying proper nouns in large corpora is a research area in itself, having a automatic procedure is preferable. Finally, the compactness of our model remains a strong point in favour of our method. To be complete, we need to mention that the best results reported came with a support vector machine approach [12] which perform an overall $F_{\beta=1}$ of 93.

8 Conclusion and further work

We have presented an algorithm for shallow parsing based on a grammatical inference algorithm; first we build an automaton to model the data sequence, and then we use the automaton as a transducer to recover the optimal sequence of labels. Advantages of using the grammatical inference technique are that the resulting models are very compact and efficient, and can model dependencies of unbounded length. It appears from our experiments, that the smoothing technique is less important here than in other domains such as language modeling. In addition, we note that it appears possible to get substantial improvements in the performance by using very limited lexical information – in our experiments, it was only necessary to model the effect of 100 words. It seems that bagging can be successfully applied to improve probability density estimators. From our knowledge this is the first application of the bagging technique to probability density estimation.

In future work we will continue our investigations of bagging, particularly examining the use of more bags, and the size of the bags. We will also experiment with some other techniques such as boosting, that we feel could be useful. There are two other directions we will also explore: first, since the automaton we infer is deterministic, it cannot use information about the future context, though the use of the Viterbi algorithm does compensate a bit. We will experiment with inferring an automaton that models the reversed sequences and combining the predictions of the "forward" and "backward" automata. Secondly, we will experiment with combining our model with other statistical models, *e.g.* to combine the probabilities provided by our model and the one provided by some statistical POS taggers [20, 2].

References

1. S. Ait-Mokhtar and J.P. Chanod. Incremental finite state parsing. In *Proc. of Applied Natural Language Processing*, Washington, DC, April 1997.
2. Thorsten Brants. TnT – a statistical part-of-speech tagger. In *Proc. of the 6th Conference on Applied Natural Language Processing*, Seattle, WA, April 2000.
3. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
4. Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

5. R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Second Intl. Collo. on Grammatical Inference and Applications*, pages 139–152, 1994.
6. Joshua Goodman. A bit of progress in language modeling. Technical report, Microsoft Reserach, 2001.
7. C. John Hederson and Eric Brill. Bagging and boosting a treebank parser. In *NAACL*, pages 34–41, Seattle, Washington, USA, 2000.
8. Christer Johansson. A context sensitive maximum likelihood approach to chunking. In *CoNLLL-2000 and LLL-2000*, pages 136–138, Lisbon, Portugal, 2000.
9. Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall, Englewood Cliffs, New Jersey, 2000.
10. 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*, vol. 35(num 3):400–401, 1987.
11. 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.
12. Taku Kudoh and Yuji Matsumoto. Use of support vector learning for chunk identification. In *CoNLLL-2000 and LLL-2000*, pages 142–144, Lisbon, Portugal, 2000.
13. 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 *Proc. 4th Intl. Coll. on Grammatical Inference - ICGI '98*, volume 1433, pages 1–12. Springer-Verlag, 1998. citeseer.nj.nec.com/lang98results.html.
14. M. Marcus, S. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.
15. F. Pla, A. Molina, and N. Prieto. An Integrated Statistical Model for Tagging and Chunking Unrestricted Text. In Petr Sojka, Ivan Kopeček, and Karel Pala, editors, *Proc. of the Third Intl. Workshop on Text, Speech and Dialogue—TSD 2000*, Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, pages 15–20, Brno, Czech Republic, September 2000. Springer-Verlag.
16. D. Ron, Y. Singer, and N. Tishby. Learning probabilistic automata with variable memory length. In *Seventh Conf. on Computational Learning Theory*, pages 35–46, New Brunswick, 12–15 July 1994. ACM Press.
17. D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *ACM*, pages 31–40, Santa Cruz CA USA, 1995. COLT'95.
18. Erik Tjong Kim Sang. Noun phrase recognition by system combination. In *Proceedings of BNAIC'00*, pages 335–336. Tilburg, The Netherlands, 2000. (extended abstract of ANLP-NAACL 2000 paper).
19. Erik Tjong Kim Sang and Sabine Buchholz. Introduction to the conlll-2000 shared task: Chunking. In *CoNLLL-2000 and LLL-2000,*, pages 127–132, Lisbon, Portugal, 2000. `http://lcg-www.uia.ac.be/lcg/`.
20. H. Schütze and Y. Singer. Part-of-speech tagging using a variable memory markov model. In *Meeting of the Assoc. for Computational Linguistics*, pages 181–187, 1994.
21. A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* Ph. D. dissertation, University of California, 1994.
22. F. Thollard. Improving probabilistic grammatical inference core algorithms with post-processing techniques. In *Eighth Intl. Conf. on Machine Learning*, pages 561–568, Williams, July 2001. Morgan Kauffman.

23. C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, United Kingdom, 1975.