Integrating a POS Tagger and a Chunker Implemented as Weighted Finite State Machines

Alexis Nasr, Alexandra Volanschi*

LATTICE-CNRS (UMR 8094)4, Université Paris 7
alexis.nasr, alexandra.volanschi}@linguist.jussieu.fr

Abstract. This paper presents a method of integrating a part-of-speech tagger and a chunker. This integration lead to the correction of a number of errors made by the tagger when used alone. Both tagger and chunker are implemented as weighted finite state machines. Experiments on a French corpus showed a decrease of the word error rate of about 12%.
Keywords : Part-of-speech tagging, chunking, weighted finite state machines...

1 Introduction

POS Tagging is often a prerequisite for more elaborate linguistic processing such as full or partial parsing. Probabilistic taggers implementing Hidden Markov Models (HMMs) are based on the hypothesis that the tag associated to a word depends on a local context, usually limited to the category(ies) of the preceding word(s). This hypothesis is generally verified and taggers implementing this model are known to be efficient and accurate (about 95% precision). The approximation is nevertheless responsible for the majority of tagging errors, which in turn lead to errors in subsequent processing stages. This situation is particularly frustrating since, at subsequent syntactic processing stages, the knowledge which could prevent the errors might be available. The present work is an attempt to deal with this problem by coupling the tagging and partial parsing stages. In this configuration, the choice of the tag is dictated by knowledge associated both to the tagger and shallow parser. The model constitutes an alternative to the classical sequential model, in which the partial parser input is the most probable solution of the tagger. In this configuration, choices made by the tagger can no longer be altered.

The type of error the present work tries to deal with is illustrated by the French sentence: *La démission n'est pas indispensable* (*the resignation is not indispensable*). Tagging the adjective *indispensable* might prove difficult as it may be masculine or feminine and the noun it agrees with (*démission*) is too remote from it (at least for an HMM tagger). In exchange, a partial parser would group the sequences *la démission*, *n'est pas*, and *indispensable* into larger units called *chunks*. This would bring the two units agreeing in number and gender,

* This work was partly funded by the Project WATSON of the French government Technolangue programme.

namely (*la démission* and *indispensable*) closer to each other, thus making it possible for an HMM to capture agreement.

This article also aims to point out the advantages of using weighted finite state machines and operations defined on them for the whole processing. In this framework, all data (sentences to be analyzed, lexicons, grammars, n-grams) are represented as finite state automata and (almost) all treatments were implemented as standard operations on automata. This homogeneity has several advantages, the most significant of which being the possibility to easily combine different modules using automata combining operations, combinations which would be more difficult to achieve between modules based on different formal models. Another advantage of this homogeneity is the simplicity of the implementation: one no longer has to define specific formats for different types of data, to implement, adapt or optimize existing algorithms. Software libraries for automata manipulation are essential for such treatments; in the present work we have used the utilities FSM and GRM developed by ATT [1]. Our work pertains to probabilistic language processing using weighted finite state machines, an overview of which can be found in [2]. It is distinct from approaches based non-probabilistic finite state automata such as INTEX [3], in which rules are manually built to be later used in automatic processing.

The paper is organized as follows: section 2 is a brief reminder of definitions concerning weighted finite state automata, introducing a number of notations used in the remainder of the article. Sections 3 and 4 describe respectively the principles of a probabilistic POS tagger and of a partial parser and their implementation using weighted finite state machines. The method of integrating the two modules is presented in the section 4.3.

2 Definitions and notations

In this article two types of finite state machines are manipulated: on the one hand automata, which recognize words u built on an alphabet Σ ($u \in \Sigma^*$), and on the other transducers, which recognize word pairs built on two alphabets Σ_1 and Σ_2 ($(u, v) \in \Sigma_1^* \times \Sigma_2^*$). In addition to standard regular operations (union, concatenation and iteration) defined on both types of machines, certain operations are specific for transducers, in particular *composition*, which plays an essential role in the present work. Given two transducers, A and B which recognize respectively the word pairs (u, v) and (v, w), the composition of A and B ($A \circ B$) is a transducer which recognizes the couple (u, w).

In addition, we use the notion of semiring, which is defined as a 5-tuple $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ such that $\mathbb{K}$ is a set equipped with two operations defined on it, generally called addition (denoted by $\oplus$) and multiplication ($\otimes$), each having its neutral element denoted respectively by $\bar{0}$ and $\bar{1}$. By associating to every transition in an automaton a weight from the set $\mathbb{K}$, we obtain a weighted automaton built over the semiring $\mathbb{K}$. A weighted automaton together with a semiring $\mathbb{K}$ generates a partial function which associates values from $\mathbb{K}$ to every word recognized. Given an automaton R and a word u, the value associated to u by R, denoted by $[\![R]\!](u)$, is the multiplication ($\otimes$) of weights on transitions along the path in R corresponding to u. If several paths in R recognize u, then $[\![R]\!](u)$ equals the Addition ($\oplus$) of weights of the different paths corresponding to u.

In the experiments described in the present work we used the ***tropical*** and the ***log*** semirings on $\mathbb{R}^+$: the weights used on transitions are opposites of the logarithms of probabilities[1]. With the tropical semiring the operation $\otimes$ corresponds to arithmetic addition (to compute the weight of a path in the automaton, weights on individual transitions constituting a path are added), while the $\oplus$ operation is the minimum (the weight associated by an automaton to a word recognized is the minimum weight of all paths corresponding to the word, i.e. the most probable path). Given a weighted automaton R over the tropical semiring, one can define the ***n-best paths*** operator, denoted by $bp(R, n)$, which yields the automaton made of the union of the n most likely paths in R. This semiring allows us to compute products of probabilities by composing automata, and to select the most probable tag sequence associated to a sentence. With the log semiring, the operation $x \oplus y$ corresponds to $-log(e^{-x} + e^{-y})$ while the operation $\otimes$ corresponds to mathematic addition. When weights correspond to opposites of probability logarithms, the $\oplus$ operation amounts to adding probabilities.

3 Standard POS Tagging

In the present work, part of speech tagging follows the principles of Hidden Markov Models, introduced by [4]. The states in the HMM correspond to parts of speech and the observable symbols to words in the lexicon. The latter constitute the alphabet Σ_L and categories in the tagset constitute the alphabet Σ_C. POS tagging in such a framework consists in finding the most probable sequence of states through which the HMM passes, given a sequence of observable symbols produced by the model (the sentence).

The parameters of a HMM may be divided into emission probabilities and transition probabilities. An emission probability is the probability of a word given a category ($P(m|c)$) while the transition probability is the probability that a given category x immediately follow a category y ($P(x|y)$). The joint probability of a sequence of categories $c_{1,n}$ (a sequence of states the model goes through) and of a sequence of words $m_{1,n}$ (a sequence of observable symbols) is computed on the basis of the emission and transition probabilities:

$$P(c_{1,n}, m_{1,n}) = P(c_1)P(m_1|c_1)\prod_{i=2}^{n} P(m_i|c_i)P(c_i|c_{i-1})$$

Such a model, called a ***bigram*** model, relies on the Markov hypothesis, according to which a category only depends on the the preceding category. This restraining hypothesis may be rendered more supple without changing the theoretical

[1] Logarithms of probabilities are preferred to probabilities for reasons of numerical stability (as probabilities may be very small real numbers, by multiplying probabilities one may be not able to represent these values on computer).

framework by making the hypothesis that a category depends on the two preceding categories; this type of model (*trigram*) is the one most commonly used for the POS assignment task. In a trigram model, a state no longer corresponds to a category, but to a pair of categories.

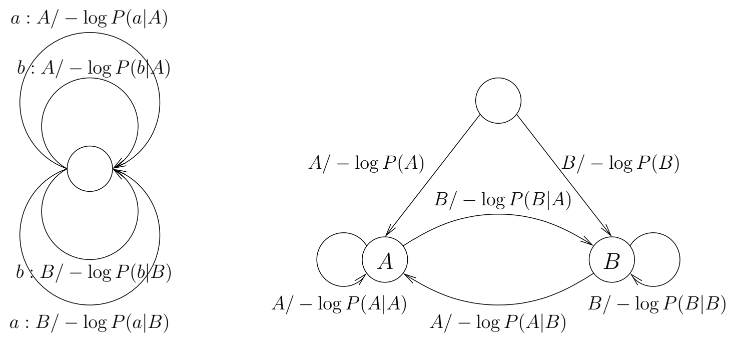

Fig. 1. The transducers E and T

Such a HMM may be represented by two weighted transducers. The first one, E, represented on the left-hand side of figure 1 (in this example $\Sigma_L = \{a, b\}$ and $\Sigma_C = \{A, B\}$), encodes the emission probabilities. Its input alphabet is Σ_L and its output alphabet Σ_C. The transducer has a single state and as many transitions (from this state to itself) as there are pairs (m, c) where m is a word from the lexicon and c a category ($c \in \Sigma_C$) such that the emission probability $P(m|c)$ is non-null. The opposite of the logarithm of this probability ($-\log P(m|c)$) constitutes the weight of the transition tagged (m, c). In figure 1 such a transition is labeled $m : c/ -\log P(m|c)$.

The second transducer, T, the input and output alphabet of which is constituted by Σ_C (on the right-hand side of figure 1), encodes transition probabilities. Its structure is isomorphic to that of the HMM: as many states as there are state pairs (x, y) (oriented from x to y) such that $P(y|x)$ is non null. Weights on transitions equal $-\log P(y|x)$[2]. With a trigram model the structure of the automaton T is more complex: a state corresponds to a sequence of two categories and the transition weights equal $-\log P(z|xy)$.

The composition of E and T ($E \circ T$) allows to combine the emission and transition probabilities; the input alphabet of the transducer obtained is Σ_L and the output alphabet is Σ_C. Such a transducer associates to every couple $(m_{1,n}, c_{1,n})$ the weight $[\![E \circ T]\!](c_{1,n}, m_{1,n}) = -\sum_{i=1}^{n} \log P(m_i|c_i) - \log P(c_1) -$

[2] Strictly speaking, the machine described is an automaton, but it can be viewed as a transducer whose transitions output the same symbol as the one they read in the input. Such a transducer represents the identity relation restricted to the language recognized by the automaton.

$\sum_{i=2}^{n} \log P(c_i|c_{i-1})$ which is in fact the opposite of the logarithm of the probability $P(c_{1,n}, m_{1,n})$, such as defined above.

Tagging a given sequence of words M is achieved by representing M as a linear automaton also called M (with one transition for every word in M) and subsequently composing M with $E \circ T$. The most probable sequence of categories given M is identified by looking for the best path of the transducer $M \circ E \circ T$. The POS tagger could then be formally defined as : $bp(M \circ E \circ T, 1)$

The trigram probabilities encoded in the automaton T are not, as a rule, estimated by maximum likelihood on a training corpus, as trigrams appearing in the texts to be tagged may never have occurred in the training corpus. It is therefore essential to use a probability smoothing technique, such as back-off [5]; the method consists in *backing-off* on the probability of bigram `b c` when trigram `a b c` is not observed in the training corpus, and on to the probability of the unigram `c` when bigram `b c` is never encountered. A back-off model may be directly represented by using *failure transitions* as described in [6]. Given a symbol α, a failure transition coming out of a state q is taken when there is no transition labeled by α. In the case of a back-off model, a failure or default transition is taken when a trigram or bigram was never observed. For more details, the reader is referred to the article cited above.

Several approaches in the literature [7–9] use finite state machines in order to simulate the functioning of HMMs. In all approaches, n-grams are represented by finite-state automata in a manner similar to ours. They are nevertheless different from our work in that they don't directly manipulate emission probabilities ($P(m|c)$) estimated on a training corpus, but resort to ambiguity classes, which are sets of categories associated to a word.

Preliminary Results All the experiments described in the present work are conducted on the tagged and hand-validated corpus produced at the University of Paris 7[10]. The corpus consists of $900K$ words tagged with 222 tags indicating the category and morphological features. $760K$ words were used for training (*Train*). Tests are done on a $66K$ words fragment (*Test*). All experiments are achieved using the libraries FSM and GRM made available by AT&T. The error rate of the trigram model (denoted by $\mathcal{M}_1$) such as described above on *Test* is of $2,18\%$ This figure is our reference.

4 Language Models Derived from Probabilistic Partial Parsing

The models of finite-state-based POS Tagging mentioned in section 3, agree on the necessity of integrating syntactic constraints. Kempe [8] anticipates the possibility of composing the tagger output with transducers encoding correction rules for the most frequent errors, Tzoukerman [7] uses negative constraints in order to drastically diminish the weight of paths containing unlikely sequences of tags (such as a determiner immediately followed by a verb). Basically, our work is different from the others in that it integrates two distinct, complete modules (a tagger and a chunker) within a single module which accomplishes both tasks at the same time. The approach does not consist in integrating a number of local grammars to the tagger, but in combining statistical information with the linguistic knowledge encoded by the chunker with a view to improving the quality of the tagger and, additionally producing a likely sequence of chunks. Before going into the details of the various integration models we conceived, let us briefly remind the principles of partial parsing, present a way of implementing a partial parser as a finite state machine and explain the necessity of converting it into a probabilistic partial parser which is also represented as a finite state machine.

4.1 Partial Parsing as Finite State Machines Manipulation

Partial parsing designates a series of techniques whose purpose is to uncover the syntactic structure of a sentence, or more precisely the structure associated to the fragments which may only have one analysis. For instance, even if for a traditional grammar a sequence like *maison des sciences de l'homme* (*house of the science of the man* (*center for human sciences*)) constitutes a noun phrase, having a complex structure with several intermediate levels, a partial parser would split it into 3 units called *chunks* : $[\text{maison}]_{NC}$ $[\text{des sciences}]_{PC}$ $[\text{de l'homme}]_{PC}$ as the prepositional attachment is potentially ambiguous. The technique, also known as *chunking*, was introduced by [11] in answer to the difficulties that robust analysis of raw text encountered.

Several approaches among which [12] have implemented partial parsing by finite state machines, or more precisely as cascaded finite state transducers. A cascade of finite state transducers is a sequence of transducers, each recognizing a type of chunk. The input of every transducer is constituted by the output of the previous one. Our solution consists in the simultaneous, rather than sequential, application of all the chunk automata which are integrated into a single transducer.

A chunk is the non-recursive core of a phrase, irrespective of its category. As opposed to Abney[13], chunking embedding is not allowed in the present approach; in exchange, the longest string matching the definition of a chunk is selected: we prefer an analysis like $[\textit{le nouveau batiment}]_{NC}$ (*the new building*) rather than $[\textit{le}\ [\textit{nouveau}]_{AC}\,\textit{batiment}_{NC}]$.

Given their non-recursive character, chunks may be represented by finite state automata built on the alphabet Σ_C. 28 chunk grammars corresponding to nominal, adjectival, adverbial and prepositional chunks are constructed manually. These grammars belong to a class of context-free grammars ([14]) which represent regular languages and which may, consequently, be compiled as finite-state machines and integrated to the chunker. For every type of chunk K, an automaton also called K is built to recognize well-formed chunks of type K. Moreover, two symbols marking the chunk beginning (`<K>`), and the end (`</K>`) are associated to every chunk of type K. The whole set of chunk beginning and chunk end marks constitute a new alphabet called Σ_K. Chunk automata are grouped within a single transducer called A, i.e. the shallow parser, whose structure is represented in figure 2.

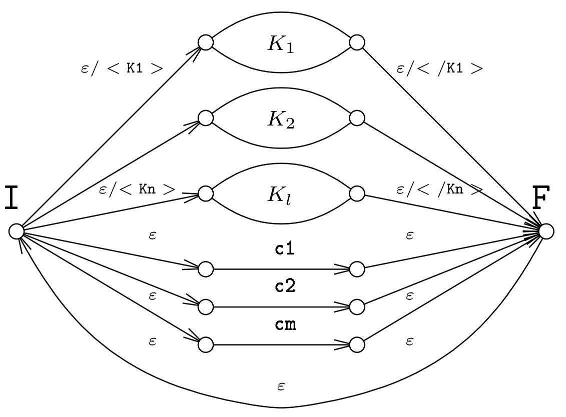

Fig. 2. Partial parser structure

A's input alphabet is Σ_C and its output alphabet is $\Sigma_C \cup \Sigma_K$. It accepts sequences of categories and outputs sequences of categories and chunk beginning or end marks. Given a sequence of categories C, A outputs the same sequence in which every occurrence of a K chunk is delimited by the two symbols `<K>` and `</K>`. As represented in figure 2, A is composed of two parts: the upper side is the union of the different chunk automata, K_i, while the lower side is made of as many transitions as there are POS categories in the input alphabet. Transitions linking A's initial state to the initial states of the chunk automata K_i introduce the chunk beginning marks, while transitions linking chunk automata K_i acceptance state to the state `F` of automaton A insert chunk ending symbols. Finally, an ε transition linking `F` to `I` builds a loop, thus making it possible to recognize several chunk occurrences in the same sequence of categories.

The automaton A recognizes any word C built on Σ_C. The analysis of C is achieved by representing it as a linear automaton, C, and then making the composition $C \circ A$. The product of this composition is most likely ambiguous, since for every sub-string s of C corresponding to a chunk K_i, two results are produced: one recognizes s as a chunk (passage through the automaton K_i), the other doesn't (passage through the lower part of A). Of these results the only one which is of interest to us is the one in which all chunk occurrences are marked by beginning and end tags. It is easy to limit the composition product to this result alone by associating a weight of 0 to intra-chunk transitions and a weight of 1 to extra-chunk transitions and selecting from the resulting transducer the minimal weight path. An additional penalty score is associated to transitions introducing chunk boundaries marks to ensure that the longest match would be chosen (i.e. the preferred analysis of a sentence would be one containing chunks, but as few boundaries marks as possible). This ensures that an analysis like [*le livre rouge*]$_{NC}$ (*the red book*) would be preferred to [*le livre*]$_{NC}$ [*rouge*]$_{AC}$ The analysis may thus be expressed by: $bp(C \circ A, 1)$.

The tagger and chunker integration could now be accomplished by a simple composition of the two models previously described, which may be expressed by: $bp(bp(M \circ E \circ T, 1) \circ A, 1)$. However, this model is an instance of the sequential architecture which was introduced and criticized in section 1 : the selection of a particular POS tag is done independently of the partial parsing stage (in the present work, this stage being a mere chunk segmentation), and may not be altered to take into account the information available to the chunker. It is possible to provide the chunker not only with the best tagging solution but with the set of all tagging solutions represented by the automaton: $bp(M \circ E \circ T \circ A, 1)$. This goes to prove the flexibility of finite state processing. Nevertheless, such a model is not very interesting either since the chunker has no discriminating power: it cannot favor any of the tagger solutions. Indeed, unlike a CFG parser, which only associates structure to sequences belonging to the grammar language, our parser accepts any sequence of categories in the tagger output, its role being limited to identifying certain sub-strings as chunks.

This is the reason which led us to build a probabilistic version of the chunker, that not only recognizes chunks, but associates them a probability according to a Markov model, the parameters of which are estimated on a corpus. Before going into the details of the ways in which such a probabilistic chunking model could be integrated with a part-of-speech tagger (section 4.3), we describe the construction of the Probabilistic Parser.

4.2 The Construction of a Probabilistic Chunker

The purpose of this model is not to favor a particular segmentation of a sequence of categories into chunks, but to provide a way to rank the different possible sequences of categories corresponding to the same sentence. To this effect, the chunker associates every sequence of categories a probability which increases function of the following factors:
- the sequence of categories corresponds to a sequence of well formed chunks
- appearing in a linear order that has frequently been observed in a training corpus. In this respect, the chunker is (functionally) very similar to an ngram of categories.

This approach has a lot in common with [15] which also employs weighted finite state transducers in order to build a probabilistic partial parser. Nevertheless, in their work there are several ways of segmenting a sentence into chunks and their parser is meant to find the most probable one among them. Moreover, the input of their parser is a linear sequence of part-of-speech tags.

The probability of a sequence of categories segmented into chunks is computed function of two types of probabilities: intra- and extra-chunk probabilities. An intra-chunk probability is the probability of a sequence of categories $c_{1,k}$ making up a chunk of type K_i, probability denoted by $P_I(c_{1,k}|K_i)$. An inter-chunk probability is the conditional probability of the occurrence of a particular type of chunk given the $n-1$ preceding chunks or categories (as this is a partial parse, certain categories in the sequence being analyzed are not integrated into chunks). The probability associated by the chunker to a sequence of categories is the product of internal probabilities of the chunks composing it and of the external probabilities of the sequence of chunks recognized.

Given the sequence of categories `<s> D N V D N P D A N </s>`[3], the segmentation proposed by the parser is:

```
C = <s> <NC> D N </NC> V <NC> D N </NC> <PC> P D A N </PC> </s>
```

The probability associated to this sequence is the product of the external probabilities of the chunks recognized ($P_E(\cdot)$), and of internal probabilities of all chunks in the sequence:

$$P(\texttt{C}) = P_E(\texttt{<s> <NC> V <NC> <PC> </s>}) \times P_I(\texttt{D N|<NC>})^2 \times P_I(\texttt{P D A N| <PC>})$$

Internal probabilities are estimated by maximum likelihood on a training corpus, as will be shown in section 4.2. The probability of a sequence of chunks and categories is estimated using an n-gram model learned on a training corpus, called external model, which encodes the probability of a chunk given the $n-1$ preceding chunks. In the case of a bigram model the external, inter-chunk probability of C is computed as follows:

$$P_E(\texttt{C}) = P_E(\texttt{<NC>|<s>}) \times P_E(\texttt{V|<NC>}) \times P_E(\texttt{<NC>|V}) \times P_E(\texttt{<PC>|<NC>}) \times P_E(\texttt{</s>|<PC>})$$

Model Construction and Parameter Estimation The parameters of the external model and of the internal chunk models are estimated in two stages on a tagged corpus as illustrated in figure 3. First the corpus is analyzed using the non-probabilistic version of the chunker, `A`. The result of this analysis is a new corpus in which chunk beginning and end symbols are inserted. Two objects are derived from this corpus: on the one hand, all sequences of categories corresponding to a type of chunk K_i (which we call patterns) and on the other hand a *hybrid* corpus in which every chunk occurrence is replaced by a symbol representing the chunk. This corpus is made of sequences of categories and chunk symbols replacing sequences of categories grouped into chunks. The first object is used to compute intra-chunk probabilities (by converting the chunk grammars written for the non-probabilistic version of the chunker described in section 4.1 into probabilistic context-free grammars) while the second one is used for estimating inter-chunks probabilities.

The estimation of inter-chunk probabilities starting from the hybrid corpus is identical to the estimation of n-gram probabilities described in section 3.

[3] Where `D`, `N`, `V`, `P` and `A` are the tags corresponding to the categories *determiner*, *noun*, *verb*, *preposition* and *adjective* respectively.

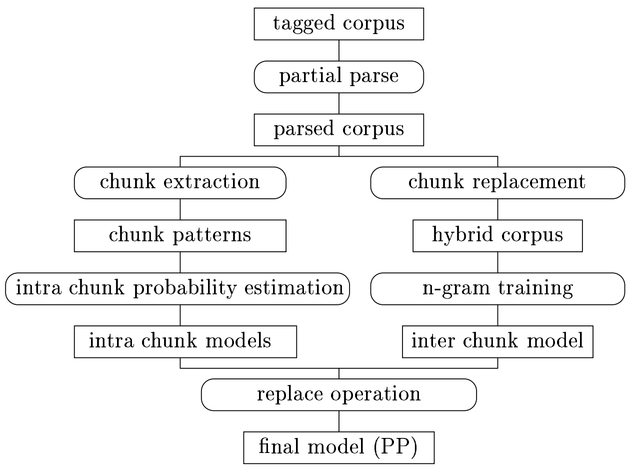

Fig. 3. The model construction stages

These probabilities are represented by a transducer (the external model) whose structure is similar to that of T in figure 1 and whose transitions are labeled with categories or chunk symbols. The estimation of intra-chunk probabilities is done by maximum-likelihood. Given a chunk C_k and n different sequences of categories $(s_1,\ s_2,\ \ldots s_n)$ representing all possible patterns for this type of chunk, observed in a training corpus, n_i denotes the number of occurrences of the sequence c_i. The probability of the sequence s_i equals its relative frequency: $P(s_i) = \frac{n_i}{\sum_{k=1}^{n} n_k}$. This probability is that of the path corresponding to s_i in the automaton K_i.

Intra-chunk models and the external model are combined within a single transducer using the *replacement* operation described in [16]. This substitutes the automaton K_i for each transition `<Ki>` in the external model. The resulting transducer is called PP (for probabilistic parser) and has about 10 times more states and transitions than T, the transducer encoding trigram probabilities, however, we assumed that it encodes longer-distance dependencies for the estimation of which our training data would have been sparse. Therefore we conducted a series of experiments in which we replaced the transducer T with PP, using the tagging model $bp(M \circ E \circ PP, 1)$.

Probabilistic Parser Performance Intra- and inter-chunk probabilities are estimated on *Train* and the new tagging model $bp(M \circ E \circ PP, 1)$ $(\mathcal{M}_2)$ was applied to *Test*. Disappointingly, we found that $(\mathcal{M}_2)$ performs only slightly better than $\mathcal{M}_1(2.05\%)$. Nevertheless, the model is interesting in that it functions as a tagger and a chunker at the same time.

More interestingly, the two models don't make the same mistakes. $\mathcal{M}_2$ corrects 30% of the errors made by $\mathcal{M}_1$ but makes almost as many of its own.

The new type of errors have various causes, most of which are related to the hypothesis we made that the form of a chunk (the sequence of categories which constitute a chunk) is independent of the context in which the given chunk occurs. This hypothesis is an approximation just like the Markov hypothesis. The sentence *'la discussion a été ouverte par l'article ...'* (*the discussion was initiated by the paper ...*) is a good case in point. In this case, *ouverte* is rightly tagged past participle by model $\mathcal{M}_1$, while $\mathcal{M}_2$ tags it adjective. The reason is that $\mathcal{M}_2$ grouped *a été ouverte* as a verb chunk, but chose the category of *ouverte* without taking into account the context where the chunk occurred. On the other hand the model $\mathcal{M}_1$ takes advantage of the fact that *ouverte* is followed by a preposition and attributes it the right category.

4.3 Integration of the POS Tagger and the Probabilistic Chunker

In order to deal with the limitations of models $\mathcal{M}_1$ and $\mathcal{M}_2$ we have combined the two within a single complex model: $bp(bp((M \circ E \circ PP), n) \cup (bp(M \circ E \circ T), n), 1)$, denoted by $\mathcal{M}_3$. This automaton is the union of the n-best solutions common to $\mathcal{M}_1$ et $\mathcal{M}_2$ to which it associates the sum of weights attributed by $\mathcal{M}_1$ and $\mathcal{M}_2$ ($[\![\mathcal{M}_3]\!](x) = [\![\mathcal{M}_1]\!](x) \oplus [\![\mathcal{M}_2]\!](x)$).

Technically, this is achieved in several stages : first the n-best solutions of $\mathcal{M}_1$ and $\mathcal{M}_2$ are computed (using the tropical semiring) then the two resulting automata are normalized and converted to the log semiring. Then the union of the two resulting automata is performed, during which an averaged sum of the common solutions is done. Finally the reconversion to the tropical semiring allows us to select the best path.

This combination is a way of partially attenuating the effect of the independence hypothesis mentioned above. The dependency between the form of a chunk and the context where it occurs is partially modelled by $\mathcal{M}_1$. The error rate obtained by the model $\mathcal{M}_3$ on *Test* is of $1,92\%$, which is $11,9\%$ less than our reference model $\mathcal{M}_1$. Error analysis showed that $\mathcal{M}_3$ corrects $15,5\%$ of the errors $\mathcal{M}_1$ makes but makes $7,9\%$ new errors. The errors specific to $\mathcal{M}_3$ may be explained by a number of causes: some are still due to the independence hypothesis mentioned above, others (about 10%) are due to the method of estimating intra-chunk probabilities (the probability that a given sequence of categories constitutes a chunk). These probabilities are estimated by maximum likelihood and therefore attribute a null probability chunk patterns which, although envisaged by the chunk grammar, have never been seen in *Train*. A technique for smoothing these probabilities is necessary for the method to be more robust. Other errors are due to the theoretical limitations of the model and would probably only be corrected by using full syntactic analysis.

5 Conclusion

The work presented in this paper has shown that taking into account the syntactic knowledge encoded in a partial parser may improve the result of a part-of-speech tagger. It also proved that the different processes involved in this treatment may be implemented as weighted finite-state machines. The model described could be improved in a number of ways, such as by a better method of estimating intra-chunk probabilities as well as a better modeling of the influence the context has on the form of a chunk. An evaluation of the chunker performances could also be performed.

References

1. (http://www.research.att.com/sw/tools/{fsm,grm})
2. Mohri, M.: Finite-state transducers in language and speech processing. Computational Linguistics **23** (1997)
3. (http://www.nyu.edu/pages/linguistics/intex/)
4. Bahl, L.R., Mercer, R.L.: Part of speech assignment by a statistical decision algorithm. In: Proceedings IEEE International Symposium on Information Theory. (1976) 88–89
5. Katz, S.M.: Estimation of probabilities from sparse data for the language model component of a speech recogniser. IEEE Transactions on Acoustics, Speech, and Signal Processing **35** (1987) 400–401
6. Allauzen, C., Mohri, M., Roark, B.: Generalized algorithms for constructing statistical language models. In: 41st Meeting of the Association for Computational Linguistics, Sapporo, Japon (2003) 40–47
7. Tzoukermann, E., Radev, D.R.: Use of weighted finite state trasducers in part of speech tagging. Natural Language Engineering (1997)
8. Kempe, A.: Finite state transducers approximating hidden markov models. In: 35th Meeting of the Association for Computational Linguistics (ACL'97), Madrid, Spain (1997) 460–467
9. Jurish, B.: A hybrid approach to part-of-speech tagging. Technical report, Berlin-Brandenburgishe Akademie der Wissenschaften (2003)
10. Abeillé, A., Clément, L., Toussenel, F.: Building a treebank for french. In Abeillé, A., ed.: Treebanks. Kluwer, Dordrecht (2003)
11. Abney, S.P.: Parsing by chunks. In Berwick, R.C., Abney, S.P., Tenny, C., eds.: Principle-Based Parsing: Computation and Psycholinguistics. Kluwer, Dordrecht (1991) 257–278
12. Abney, S.: Partial parsing via finite-state cascades. In Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information, Prague, Czech Republic, pages 8–15. (1996)
13. Abney, S.: Chunk stylebook. `http://www.vinartus.com/spa/publications.html` (1996)
14. Mohri, M., Pereira, F.C.N.: Dynamic compilation of weighted context-free grammars. In: 36th Meeting of the Association for Computational Linguistics (ACL'98). (1998)
15. Chen, K.H., Chen, H.H.: Extracting noun phrases from large-scale texts: A hybrid approach and its automatic evaluation. In: Meeting of the Association for Computational Linguistics. (1994) 234–241
16. Mohri, M.: Weighted Grammars Tools: the GRM Library. In: Robustness in Language and Speech Technology. Jean-Claude Junqua and Gertjan Van Noord (eds) Kluwer Academic Publishers (2000) 19–40