Algorithms for Speech and NLP
TD4

Michaël Ramamonjisoa
mramamon@ens-paris-saclay.fr

In this homework, we have designed a sentence syntactic parser, which is learned on a corpus and tested on sentences made of words of the same corpus. Its output is a tree, representing the predicted parsing.

A complete description of our code is contained in the IPython notebook **TP4.ipynb**. This document summarizes our approach and the main results obtained with the system we built.

1 Our approach

1.1 Grammar rules processing : computing the PCFG

1.1.1 Converting the grammar to Chomsky Normal Form (CNF)

After splitting the corpus into sentences, and extracting the rules in each sentence, we must convert each of them to CNF format. This is done using NLTK package :

- Tree.fromstring(str) : converts string sentence into a tree :
  `( (SENT (NP (DET Le) (NC Procès) (PP (P en) (NP (ADJ première) (NC instance))))))`
- Tree.chomsky_form() and Tree.productions() : returns
  `[ -> SENT, SENT -> NP, NP -> DET NP|<NC-PP>, DET -> 'Le', NP|<NC-PP> -> NC PP, NC -> 'Procès', PP -> P NP, P -> 'en', NP -> ADJ NC, ADJ -> 'première', NC -> 'instance']`

After converting the corpus' sentences to Chomsky Normal Format using nltk package's **chomsky_normal_form()** function, some rules are of the form :

```
NP -> NP SENT|<PP-PONCT-VP>
```

We processed those rules by removing everything after the | sign. The rules between brackets are those who are derived from the (NP $\rightarrow$ SENT) rule.

1.1.2 Building the PCFG

We then build the Probabilistic CFG by adding to a grammar each rule that has been processed. Each time a rule is encountered in the corpus, we add one occurence of this rule to our grammar, and hence update its probability which is the empirical frequency.

Then, to make sure every word is in the corpus (including words in the testing and development parts) are understood by our algorithm, we need to add all terminal rules such as (A $\rightarrow$ word).

1.2 CKY algorithm

Our system is a CKY parser, which uses probabilities learned on a corpus through contained in the PCFG.
Our system must take as input a French tokenized sentence with exactly one blank space between each word :

```
La terre est ronde .
```

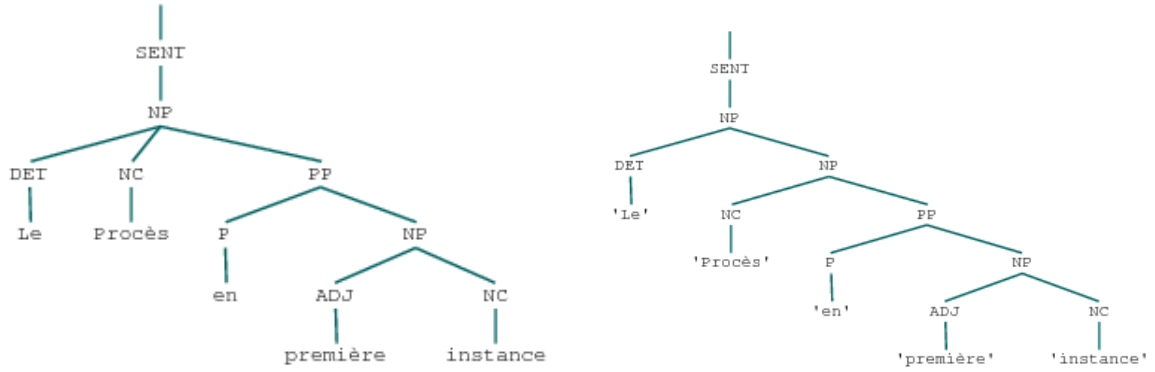
In the writing of our code one main difficulty was the handling of unary rules (or unaries) such as (A $\mapsto$ B) with A and B non terminal. Indeed, in the Jurafsky and Martin textbook, the algorithm PROBABILISTIC-CKY does not handle unaries. We used an implementation from an MOOC on NLPs on Coursera. Details can be found in our code and notebook.

Our BUILD_TREE function uses recursive backpointing to recover the most probable parsing from the PROBABILISTIC-CKY algorithm's output.

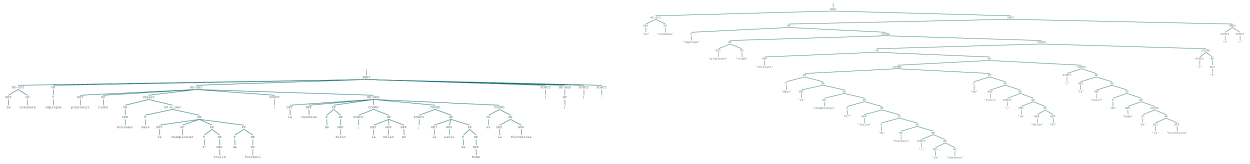
2 Results and conclusion

As shown in the figure below, the algorithm produces good results on some short and simple sentences. Indeed in this example, the only difference between the two trees is that the first tree (the ground truth) is not in CNF format.

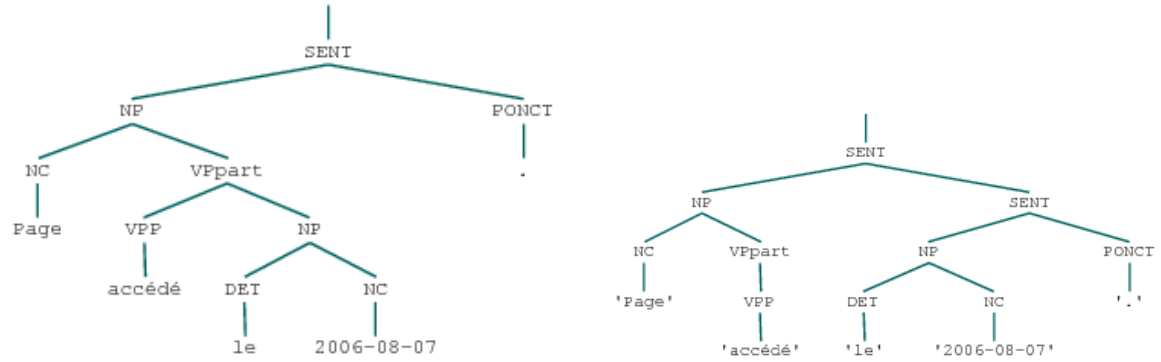
We also notice that the unary (SENT $\mapsto$ NP) is correctly handled.



However, long sentences are sometimes not handled correctly, as can be seen in the figure below (left : ground truth / right : prediction) :



Another issue is that some sentences do not contain any verb, as seen in the figure below (left : ground truth / right : prediction). Our algorithm does not parse sentences with such minimal structure correctly.



We could also improve our system by automatically get the POS tag of an unknown word (i.e. a word which does not appear in the corpus), so a sentence parsing can still be infered. One could for instance look at the Stanford French POS tagging : (https://nlp.stanford.edu/software/tagger.shtml).