Goal & Overview

The purpose of this TD is to develop a basic probabilistic parser for French that is based on the CYK algorithm and the PCFG model that is robust to unknown words. To do so, we proceed in three parts: (i) Implement an OOV module (ii) estimate the probabilistic CFG model from parsed sentences (iii) implement CYK algorithm .

1 Modelling choices

1.1 OOV

Here we attempt to build a module that assigns a (unique) part-of-speech to any token not included in the lexicon extracted from the training corpus. Therefore, the underlying problem is to build an underlying proximity measure between tokens. But this metrics combines Levenshtein formal distance d_1 to handle spelling errors and embedding similarity s_2 to handle contextual proximity. Therefore, given two tokens x, and y, we use the following similarity metrics:

$$s(x,y) = \lambda \exp\left(-\Gamma d_1(x,y)\right) + (1-\lambda)\left(\frac{1+s_2(x,y)}{2}\right)$$

with the weight $\lambda \in (0,1)$ and $\Gamma > 0$. We see that the exponential allows us to put the two terms of our convex combination in the the same range $[0,1]$[1]. The hyperparameters Γ and λ have been chosen based on what we called the *greedy parser*. Imagine that we a sentence of token $w_1 w_2 ... w_n$, the greedy parser outputs $t_1 t_2 ... t_n$ where t_i is the most likely POS tag of w_i if it is the lexicon or its closest word $\tilde{w}$ present in the lexicon.

1.2 PCFG model inference

Before estimating the rules probabilities we should extract and count all the rules encountered in the corpus of parsed trained sentences, the anchors and the POS tags as well. Now we could have used python `NLTK`,but we finally decided to do it from scratch since it gives us more control on the whole chain. Indeed to run CYK algorithm, we need to convert CFG into Chomsky normal form. In our case we must (i) eliminate right-hand sides more with more than 2 non-terminals (ii) eliminate unit rules. Extracting the rules and other related features from scratch gives us the advantage to deal (eliminate) with unit rules. To extract the rules, we detect the non-terminal symbols and their level (height) using a simple graph traversal algorithm based on symbols `'('` and `')'`. By doing so we can extract all the rules of form $A_0 \to A_1 A_2 ... A_k$ sentence by sentence. During the reading of a sentence, we also extract the anchor words with their related non-terminal symbols. Therefore for given symbols A, B and anchor word w, we could estimate probabilities in a (naive) Maxiimum Likelihod fashion by:

$$\hat{\mathbb{P}}(A \to B) = \frac{\text{count}(A \to B)}{\sum_C \text{count}(A \to C)}, \quad \hat{\mathbb{P}}(w \to B) = \frac{\text{count}(w \to B)}{\sum_C \text{count}(w \to C)}$$

Now coming back to the way, we eliminated right-hand sides with more than two non-terminals. For instance, let's examplify with the rule $A_0 \to X_1 X_2 .. ; X_n$ with probability $\hat{p}$. Drawing inspiration from [2] (p. 36), this rule is then replaced by $n-1$ following rules: $A_{n-2} \to X_{n-1} X_n$ with probability 1, $A_i \to X_{i+1} A_{i+1}$ for $i \in [1, n-3]$ with probability 1, and $A_0 \to X_1 A_1$ with probability $\hat{p}$.

1.3 Probabilistic CYK

PCYK outputs the most probable parse tree for a given sequence using dynamic programming. We implement with a class called `PCYK` with a method called `MakeTable` which computes the probabilities table `table` and the backpointers `back` required to compute the best parse as described in [1] (p. 243). Once we have `back` we use backward induction to effectively recover the best parse tree. The ultimate methode `parse` produces the sparse in a format similar to file SEQUOIA. Sometimes CYK algorithms *failed*, i.e. it groups all the tokens under one

[1] If the tokens x and y do have embeddings w_x and w_y in $\mathbb{R}^k$, then $s_2(x,y) = \frac{w_x^T w_y}{\sqrt{w_y^T w_y}\sqrt{w_x^T w_x}}$

symbol SENT. In this case, we rather split the tokens by the greedy parser described above. The greedy parser is implemented like a method called `OOPParse`.

2 Results and Comments

2.1 Hyperparameters optimization

A question that might be asked is why did we optimize the hyperparameters with the greedy parser. We did this because, the CYK parser is very time consuming given its complexity and many runs were not feasible. In this TP, we only rely on part-of-speech accuracy to evaluate our modelling choices. Therefore for integers i, j in $[0, 10]$, we keep the average accuracy s_{ij} over the first 10% of the dataset for $\Gamma_i = \frac{i}{10}$ and $\lambda_j = \frac{j}{10}$. The best performing parameters were $\Gamma^* = \lambda^* = 0.3$.

2.2 Evaluation and comments

The accuracy on the evaluation dataset is 72.81% (75.62% for the dev dataset). Like we said above we encountered some failures with a rate $\approx 30\%$, behind the hood, we proceeded to two random splittings on the datasets and recorder failure rate of 29.9% and 32%.. Notwithstanding, our tool is able to learn diverse grammatical structures from simple to more complex ones. Among the complex structures, it is successful to identify simple sentence with various complements and pooled simple sentences by complements *et*, *ou*, etc., but does struggle with relative subordinate clauses and subordinate conjunctions in general. below are some examples:

- **input 1**: Lors_de ces échanges , Moggi donnait ses instructions pour la désignation des arbitres dans le matches de son équipe .
- **parsing 1**: ((SENT (P Lors_de) (DET ces) (NC échanges) (PONCT ,) (NPP Moggi) (V donnait) (DET ses) (NC instructions) (P pour) (DET la) (NC désignation) (P+D des) (NC arbitres) (P dans) (DET le) (NC matches) (P de) (DET son) (NC équipe) (PONCT .)))
- **input 2**: Le préfet Bernard Bonnet est placé en garde à vue et est suspendu de ses fonctions .
- **parsing 2**: ((SENT (NP (DET Le) (NP (NC préfet) (NP (NPP Bernard) (Srel (NP Bonnet) (VN (V est) (COORD (VPP placé) (PP (P en) (NP (NC garde) (PP (P à) (NP (NC vue) (COORD (CC et) (VPinf (VN (V est) (VPP suspendu)) (PP (P de) (NP (DET ses) (NC fonctions))))))))))))))) (PONCT .)))
- **input 3**: En_ce_sens , les prévenus ont été condamnés solidairement à payer environ 17_600 euros de dommages-intérêts aux parties civiles pour les seuls meubles entreposés dans les immeubles qui , juridiquement , ne leur appartenaient pas .
- **parsing 3**: ((SENT (ADV En_ce_sens) (PONCT ,) (DET les) (NC prévenus) (VN ont) (VPP été) (VPP condamnés) (ADV solidairement) (P à) (VINF payer) (ADV environ) (DET 17_600) (NC euros) (P de) (NC dommages-intérêts) (P+D aux) (NC parties) (AP civiles) (P pour) (DET les) (ADJ seuls) (NC meubles) (VPP entreposés) (P dans) (DET les) (NC immeubles) (PROREL qui) (PONCT ,) (ADV juridiquement) (PONCT ,) (ADV ne) (DET leur) (V appartenaient) (ADV pas) (PONCT .)))

2.3 Improvement proposals

Aside the shortcomings of PCFG pointed in [1], i.e *poor independence assumptions* and *Lack of lexical conditioning*, we have two proposals to better our tool. To improve the present work, we can use a larger training dataset to enrich our grammar. We can also try to use a better OOV module, using for instance a language model or other string distances (kernels).

References

[1] D. Jurafsky and J. H. Martin. *Speech and language processing*, volume 3. Pearson London , 2018.

[2] B. Sagot, *Practical Assignment 2*, ENS-CACHAN , 2019.