Assignment 4: CYK algorithm and the PCFG model

Adil Rhoulam

arhoulam@ens-paris-saclay.fr

1 The principales of the system

1.1 The functions and approach used:

- The file parser contains 3 python files (.py) *Code_PCFG.py, CYK_parser.py* and *Evaluation.py*. All the codes are adapted to python3 and will, maybe, give errors if compiled with python2.

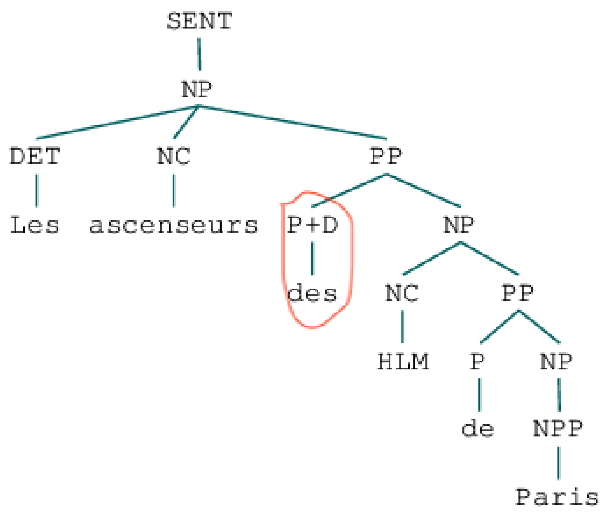
- The file *Code_PCFG.py* contains the class PCFG taking in argument the path to the SEQUOIA corpus and contains 4 methods: *pre_process(self)*, *extract_CFG_rules(self)*, *extract_CNF_rules(self)* and *learning_CNF_probabilities(self)*. The first one removes the functional labels and splits the corpus into 3 sets (train/ dev and eval). The second one extract the CFG rules by using the *nltk* library (*nltk.Tree.fromstring().productions()*). The method *extract_CNF_rules* extract the CNF rules by transforming the trees to chomsky_normal_form before extracting the rules by using *nltk.Tree.chomsky_normal_form(tree)*. The last method learn the probabilities by using the *ntlk* library also (*nltk.grammar.induce_pcfg*) and other *nltk.grammar*methods are used *Nonterminal* which define a nonterminal node and *is_nonterminal* to test if the node is a nonterminal or not. The outputs are saved respectively in: *train_set.txt*, *dev_set.txt*, *eval_set.txt* and *grammar_learnt.txt*.

- The file *CYK_parser.py* contains the class CKY taking in argument the *grammar* and contains *nonTerminal(self)*, *leavesRules(self)*, *unaryRules(self)*, *binaryRules(self)*, *backtrack(self,sentence,nodes,begin,end,label)*, *cky(self,sentence)* methods. I implemented this class inspired from the website: https://www.datasciencecentral.com/profiles/blogs/some-nlp-probabilistic-context-free-grammar-pcfg-and-cky-parsing. The method *nonTerminal(self)* adds to a dictionary all the non terminal nodes in the grammar. *leavesRules(self)* adds all the words (leaves nodes) to a dictionary. *unaryRules(self)* adds all the unary rules A → B from the grammar to a dictionary. *binaryRules(self)* adds all the binary rules A → B,C from the grammar to a dictionary. The method *cky(self,sentence)* takes in argument a sentence and predict its grammatical tree (the tokens in the sentence should all be in the learnt grammar, otherwise the output is None), it builds two tables of nodes and scores (probabilities) using the CKY algorithm and then backtrack the most probable grammar for the sentence by using the *backtrack(self,sentence,nodes,begin,end,label)* method which return the predicted tree after debinarizing it using the *nltk.Tree.un_chomsky_normal_form(tree)* method, the predicted trees are then stored in the same format of the corpus in the *eval_predicted.txt*. This method is tested over the sentences from the evaluation set by first extracting the sentences from *the eval_grd_truth.txt* and storing them in the *eval_set.txt*.

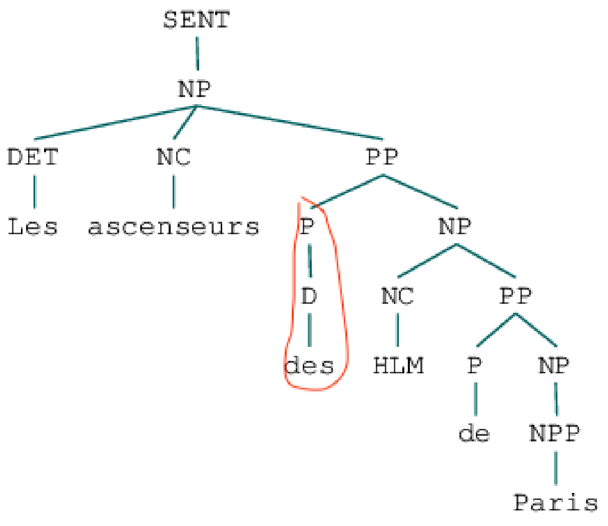
- The file *Evaluation.py* contains the class Evaluation that takes into arguments two paths the one for the ground truth of the sentence's grammar that are predicted (*the eval_grd_truth.txt*) and the ones predicted (*eval_predicted.txt*). It contains 6 methods, *preprocess_eval(self)* that opens the two text files and extract from each grammar sentence its productions. *Precision(self)* that calculate the precisions by comparing the productions from the ones predicted and the ground truth in each sentence and add one to a counter when there is a match, nothing otherwise and divide on the number of the productions of all the sentence in the ground truth. *Recall(self)* does the same as the previous method, however instead of dividing on the total number of productions in the ground truth, it divides them on the total number of productions in the predicted grammar sentences structure, the method *F1_score(self)* that calculates the F1 score, the method *true_predicted(self)* returns the number of well predicted sentences when comparing them to their ground truths and the accuracy which is the number of well predicted divided by the number of predicted sentences, the method *false_predictions(self)* returns the false predicted sentences. All the outputs of the Evaluation class are stored in the text file: *evaluation.txt*.

2 System evaluation

By evaluating over 310 sentences with their ground truth, we get as can be seen in the *evaluation.txt* a Precision of 0.9148 and a Recall of 0.8998 and a F1 score of 0.9072. Since the grammar is very large: 16565 rules where 42 of them are non terminal unary rules, 8420 non terminal binary rules and 10926 terminal rules (leaves rules), and since the complexity to parse one sentence is $O(|G|.n^3)$ where —G— is the length of the grammar, the time elapsed to evaluate all those sentence is about 4 hours. The number of well predicted sentences is : 99 which give a rate of accuracy of 99/310 = 0.32. By inspecting the false predicted sentences, we can see that the most false predictions are not duo to rules (productions) predictions but to structure retrieval. We can notice 3 types of error prediction: rule false prediction as can be seen in figure 1 where the (P+D des) rule is predicted in the form of two equivalent rules (P(D des)), the most of those error are duo to this error either as seen in the exemple or the inverse (i.e (P(D des)) is the true rule and (P+D des) is predicted). The second and the most noticeable type of errors happens when there is false structure retrieval with no false rule as can be seen in figure 2. The third and the last type is when both the structure and rule are not well predicted as can be seen in figure 3. To avoid the first type of error we have to use only one structure and not equivalent ones (P+D or P(D) or consider that the prediction is correct when this equivalence occur between ground truth and predicted trees. The second and third type of errors are seen when we have to transform the tree to a binary tree in order to apply the CYK algorithm and then retransform it back to its initial form, in fact when we are predicting the binary tree confusions occur when the tree contains redundancy of the same rules in different parts (such as PP (NP) in the figure 3, this type of errors could not be avoided by the CYK algorithm however it could be avoided, maybe, if we learn in the grammar different simple sentences (with mono or binary trees) with this types of redundancy.

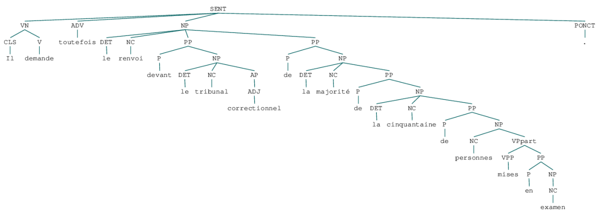


(a) Ground truth

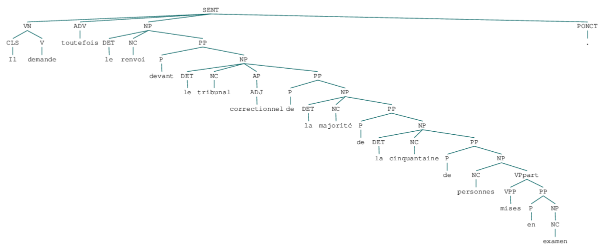


(b) Prediction

Figure 1: Tree false rule prediction between ground truth and prediction

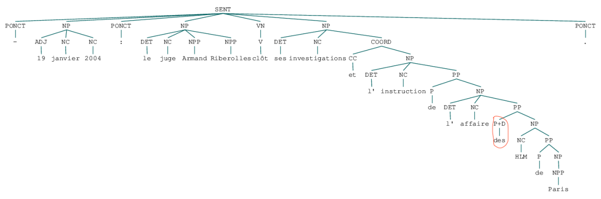

SENT
VN ADV NP PONCT
CLS V DET NC PP
Il demande toutefois le renvoi devant le tribunal correctionnel de la majorité de la cinquantaine de personnes mises en examen .

(a) Ground truth

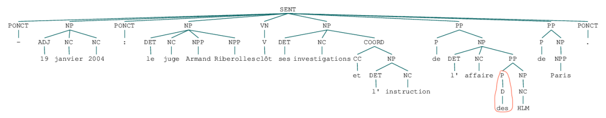

SENT
VN ADV NP PONCT
Il demande toutefois le renvoi devant le tribunal correctionnel de la majorité de la cinquantaine de personnes mises en examen .

(b) Prediction

Figure 2: Tree false structure prediction between ground truth and prediction


SENT
PONCT NP PONCT NP VN NP PONCT
- 19 janvier 2004 : le juge Armand Riberolles clôt ses investigations et l' instruction de l' affaire des HLM de Paris .
P+D

(a) Ground truth


SENT
PONCT NP PONCT NP VN NP PP PP PONCT
- 19 janvier 2004 : le juge Armand Riberolles clôt ses investigations et l' instruction de l' affaire des HLM de Paris .
P D

(b) Prediction

Figure 3: Tree false structure + rule prediction between ground truth and prediction