TP2 : Probabilistic parsing system for French

John LEVY
Ecole Normale Supérieure - Master MVA
Monday 9 March 2020
johnlevy125@gmail.com

ABSTRACT

The goal of this assignment is to build a working statistical constituency parsing architecture based on the CYK algorithm, with a module for handling out-of-vocabulary words (OOVs) based on edit distance and word embedding similarity.

1 PCFG MODEL

The PCFG module enables to extract the grammar from the training corpus. I have implemented myself the PCFG grammar, except the function `chomsky_normal_form` that allows me to have the chomsky form for each sentence in the training corpus and so to move more easily. First, I have created the **`gramar`** dictionary which stores all rules in the following format : the key are tuple (X,Y) with X any Part of speech tag (POS), and Y denotes any right-hand term, and the value is the probability to have this tuple (X,Y).
I have also created additional dictionary and list :

- **`lhs_terminal_proba` :** dictionary which is a probabilistic lexicon, i.e. triples of the form (token, part-of-speech tag, probability) such that the sum of the probabilities for all triples for a given token sums to 1.

- **`lhs_non_terminal_proba`:** dictionary which stores the probability of $P(\alpha \rightarrow \beta|\alpha)$ where α and β are two grammar rules. We compute this probability by using the formula :
$P(\alpha \rightarrow \beta|\alpha) = \frac{Count(\alpha\rightarrow\beta)}{\sum_\tau Count(\alpha\rightarrow\tau)} = \frac{Count(\alpha\rightarrow\beta)}{Count(\alpha)}$

- **`lexique` :**which stores all word present in the training corpus. This list will be useful for the OOV module.

All other dictionary that are implemented in my PCFG module are just useful for computing probability and keep a track of the occurrence for a (POS,token) or a (α, β).

2 OOV MODEL

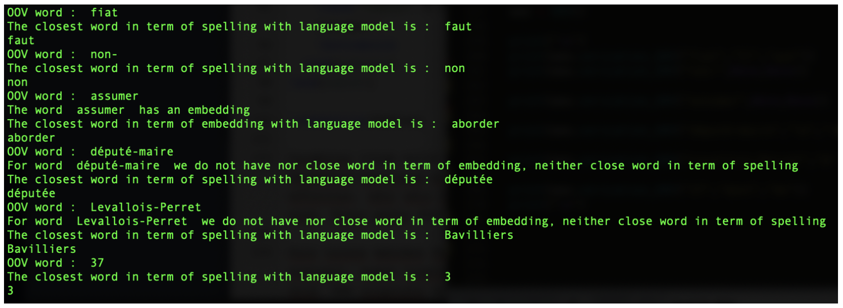
We need to handle Out-Of-Vocabulary word, which means that when we need to assign a POS tag for any unseen token. All the token that we have are stored in the lexique list, these tokens are all the the different words that we have seen in the training corpus. The underlying idea is to assign to an OOV the part-of-speech of a "similar" word. First, with the embeddings given by the polyglot, I store all word in the training corpus that have an embeddings, this will be useful when we will compute the cosine similarity. I have also implemented the Levenshtein distance function that give the distance of 2 words in terms of spelling errors.
I also chose to compute a language model by taking into consideration bigram in order to make my model more robust. All the bigram are stored into the bigram matrix. To find this similar word from an unseen token, I have implemented the following strategy :

- **Normalization :** Sometimes, the OOV word is the same as a word that we have in the training corpus, but with a capital letter, or written in small caps, or other. For instance, in my vocabulary (i.e all the words from the training corpus), the word february is present, but February is an OOV word. To hande this problem, the function normalize change the oov word into the same word but in small caps, or big caps, or in title format, and see if this word is in our vocabulary. In that way, all the small mistake as small caps to capital letter are supported.

- **close_word_emb :** When we have an oov word that has an embedding, thanks to the polyglot embeddings, we are computing the cosine similarity for this word with any word in the corpus training that have an embedding. This techniques gives me quite good result but I chose to go further and take into consideration any word that has a Levenshtein distance smaller than 4 into consideration with our oov word. The main motivation is that sometimes, for instance, the embedding of the word "enchanté" can be different from the word "enchantée", so I rather take into consideration words that have a small Levenshtein distance to make my oov module more robust. All this word are stored in a list of `candidates`. From this candidates, I remove all words that don't have an embeddings, and this time again for each remaining word, compute the cosine similarity with the training corpus embedding. At the end, I have all the candidates with the associated cosine similarity, and I now add the probability transition $P(w_i|w_{i-1}) * P(w_{i+1}|w_i)$ given by the bigram matrix, here w_i is the candidate, w_{i-1} is the word that comes before the oov word in our sentence, and w_{i+1} the word that comes after. By adding this probability, we try to adding a context that will allow our model to understand the "meaning" of a sentence, and so to assign to any OOV word, close word in term of meaning for the context of the sentence.

- **close_word_spe :** When the oov word doesn't have any embedding. I compute the Levenshtein distance for this word with all the words in my training corpus, and keep all words that have a Levenshtein distance smaller or equal to 4. I have my list of `candidates`, and with this list I compute the transition given the previous word and the next word thanks to the embedding matrix. I multiply this quantity by a ponderation of the Levenshtein distance, in order to put a preference on words that have a small Levenshtein distance. Another useful trick is at the end, divide the above quantity by the frequence of each word, by doing so, we are sure that we will not take anytime the words that are present to often like punctuation.

I test my OOV module on some words that are not present in our training corpus.



```
OOV word :  fiat
The closest word in term of spelling with language model is :  faut
faut
OOV word :  non-
The closest word in term of spelling with language model is :  non
non
OOV word :  assumer
The word  assumer  has an embedding
The closest word in term of embedding with language model is :  aborder
aborder
OOV word :  député-maire
For word  député-maire  we do not have nor close word in term of embedding, neither close word in term of spelling
The closest word in term of spelling with language model is :  députée
députée
OOV word :  Levallois-Perret
For word  Levallois-Perret  we do not have nor close word in term of embedding, neither close word in term of spelling
The closest word in term of spelling with language model is :  Bavilliers
Bavilliers
OOV word :  37
The closest word in term of spelling with language model is :  3
3
```

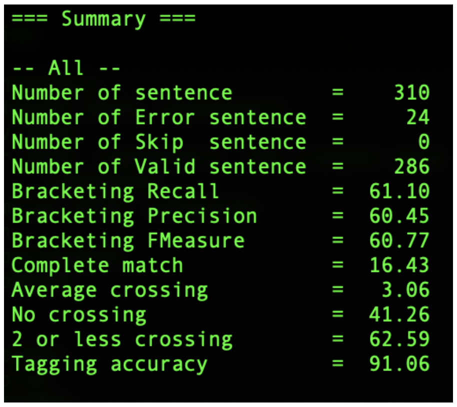
Figure 1: OOV examples.

3 CYK MODEL

The goal the of CYK model is to find the grammar tree that suits the best with our sentence. The algorithms works as follow : given a set of grammar rules, the input sentence, and the oov module for handle unseen token, all possible parses of the sentence are computed. The algorithm works in a dynamic programming way, indeed, we are parsing the subtree of lenght 1, then 2 and so on until the entire sentence has been parsed.

4 RESULTS

First, I prepossessed my sequoia set by removing all the "-" of grammar rules to avoid sparsity problem. Then I split this data set into a training set on which I trained my model, a development set on which I fine tuned some parameter present in the OOV module, and the test set on which I test my model.
I have evaluated my parsing with the open-source package EVALB, the results are presented in the following figure. We have 91% of accuracy and 16 complete match, which shows that our model is quite accurate.

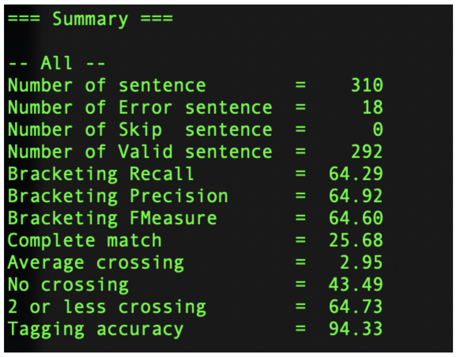


```
=== Summary ===

-- All --
Number of sentence          =    310
Number of Error sentence    =     24
Number of Skip  sentence    =      0
Number of Valid sentence    =    286
Bracketing Recall           =  61.10
Bracketing Precision        =  60.45
Bracketing FMeasure         =  60.77
Complete match              =  16.43
Average crossing            =   3.06
No crossing                 =  41.26
2 or less crossing          =  62.59
Tagging accuracy            =  91.06
```

Figure 2: Results of the parsing without shuffle.

I also found out that by shuffling the sequoia data set, so we take randomly 80% of the data set rather than the first 80% lines, our result improve as we take words that usually are just at the end of this data set. The results are in the following figure. We can see that now the accuracy is close to 95% and we have 26 complete match.



```
=== Summary ===

-- All --
Number of sentence          =    310
Number of Error sentence    =     18
Number of Skip  sentence    =      0
Number of Valid sentence    =    292
Bracketing Recall           =  64.29
Bracketing Precision        =  64.92
Bracketing FMeasure         =  64.60
Complete match              =  25.68
Average crossing            =   2.95
No crossing                 =  43.49
2 or less crossing          =  64.73
Tagging accuracy            =  94.33
```

Figure 3: Results of the parsing with shuffle.

5 ERROR ANALYSIS AND POSSIBLE IMPROVEMENTS

The model that we have implemented gives us a parsing that relies on the context free grammar assumptions. The problem with CFG is we do not take into consideration the dependency of each word in the sentence. So we could use another way to represent phrase structure : The Dependency structure. This method enables to describe the structure of the sentence by taking each word and saying what it is depend on.
We can also improve our model by training on a larger corpus : with a larger training set, we will see more word and our lexicon will be reacher. The OOV module can be also improve by moving from the bigram language model to a n-gram model. Indeed, the n-gram will allow our model to understand the whole meaning of the sentence. About the error that I have encountered, about 20 sentence can't be parsed with my model, the main explanation are that I have never encountered this grammatical structure before, or because the oov word that has been associated to a POS tag is not good and leads to an error.