# Algorithms for speech and natural language processing
## HOMEWORK 2: PROBABILISTIC PARSER

**March 9th 2020**

Emilia Siviero
M2 MVA, ENS Paris-Saclay
emilia.siviero@ens-paris-saclay.fr

## Introduction

The goal of this project is to build a French probabilistic parser based on the PCFG model and on a probabilistic version of the CYK algorithm. The probabilistic parser will use an Out Of Vocabulary model in such a way to become more robust to unknown words (out of vocabulary words).

The dataset is the SEQUOIA treebank that is splited into 3 parts: the first $80\%$ goes for the training part, the next $10\%$ for the evaluation part and the remaining $10\%$ part for the testing task of our model.

First, we will introduce the Probabilistic Context Free Grammar module that will extract the tools to build a PCFG from the training corpus provided. Then, we will present the Out Of Vocabulary module that, given a token that is not in the lexicon of the PCFG object, will return a part-of-speech computed thanks to both edit distance and word embedding similarity, and show how it is robust to unknown words. Finally, we will present the probabilistic version of the CYK Algorithm and apply it on our test dataset.

## 1. Architecture

### 1.1. Preprocessing the data

Before going to the deep of the project, we have to preprocess the data.

We have to ignore the functional labels in the dataset, which means that we have to ignore the hyphen and all what follows in every non-terminal tag.

This work is achieved in the `preprocessing_data` function that is contained in the `functions_nlp` file. It makes use of the `re` package of `Python` to find all the tags with an hyphen in it, but that does not end with a ')' (in this case, the token is simply a word with an hyphen, which we want to keep as a part of the sentence). Then, thanks to another function of the same package, it cancels the hyphen and what follows.

In the same file of functions, one can also find two other functions that are:

- get_unparsed: given a parsed line, it returns the sentence without all the brackets and the part-of-speech tags. This function will be used to build the test dataset in the non bracketed format to verify on it our CYK algorithm.
- get_parsed: given a sentence, the table of backpointing triples and the best rule given by the CYK algorithm, it returns the sentence in the bracketed format.

### 1.2. Probabilistic Context Free Grammar

For the PCFG module, I choosed to use the `nltk` package [1] and use the associated functions [1] to build the probabilistic context-free grammar and the probabilistic lexicon (triples of the form (token, part-of-speech tag, probability)) based on the training set.

In the same module, we built four others dictionnaries (all built based on the context-free grammar productions) that are:

- an inverse lexicon: it is composed of triples of the form (part-of-speech tag, token, probability). It will be useful in the CYK algorithm
- a tokens, a tags and a probs dictionnaries that will store all the tokens, all the tags and all the probabilities of the lexicon.

As we wanted the sum of the probabilities for all triples for a given token to be equal to 1, we normalize the lexicon, the inverse lexicon and the probs dictionnary.

[1] https://www.cs.bgu.ac.il/ elhadad/nlp16/NLTK-PCFG.html

### 1.3. Out Of Vocabulary

The OOV module is used to find the most similar word in the PCFG lexicon, given an unknown word in the sentence.

The function `most_similar_words` gives the K Nearest Neighbours of an input word. The function calls two other functions that measure the distance between two words:

- the Levenshtein distance is computed between two words by the Damerau-Levenshtein Algorithm [2]
- the edit distance is computed by the cosine similarity between the embeddings of two words that must be in the vocabulary. The vocabulary is obtained by the `polyglot-fr.pkl` file.

As the Damerau-Levenshtein distance is a number between 0 and the maximal length of the two words and as the cosine similarity gives a number between -1 and 1, we have to normalize the Damerau-Levenshtein distance.
Then, to obtain the score between the two words, we take the mean between the two distances (after normalization) if the word is in the vocabulary, and otherwise we only take the Damerau-Levenshtein distance.

Finally, to get the K most similar words of a given word, we compute the scores between the input word and all the words in the vocabulary and we take the K words with the highest score. As an example, we have the input word "universite" (note that it is the French word without the accent) and we get back the results in the FIGURE 1: the algorithm returns the same word, first in Italian, then in French and then in English (and also in the plural way for the last two languages). As an other example, that can be seen in the same figure, we study the results for the mistyped word "acress" and we observe as the most similar word the correct way to write "actress".

```
K Most Similar Words of: accress
['Actress', 'access', 'décrets', 'sacres', 'ocres']
####################################
K Most Similar Words of: universite
['Università', 'université', 'University', 'universités', 'Universities']
####################################
```

Figure 1: Results of the K most similar words of the OOV module

This way, the OOV module we built is robust to the unknown words that we can find in a sentence.

### 1.4. CYK

The module CYK will make use of the lexicon, the inverse lexico, the context-free grammar, the dictionnary of tokens and the dictionnary of probabilities computed on the training dataset thanks to the PCFG module.

I implemented a `most_similar_tags` function that works as the `most_similar_words` (it makes use of it) but instead of returning the K most similar words it returns the tags of these found words (if any). We can observe the results when this method is applied on the same unknown word ("universite") in FIGURE 2. This method will be used in the CYK algorithm when the word in the sentence that is being studied is not in the lexicon.

The CYK algorithm [3] we built is based on the algorithm that is detailed in FIGURE 3.

## 2. Results

As precised in the above sections, the OOV module and it's associated functions works well and is robust to the out of vocabulary words. Furthermore, excepted the computation of all the word embeddings of the vocabulary obtained thanks to the `polyglot-fr.pkl` file, the other functions are quite fast. We can also notice that the attributes of the PCFG class (lexicons, grammar and dictionnaries) work well too and are computed quite fast.

Due to lack of time, I couldn't test the CYK algorithm on all the test dataset.

Furthermore, the CYK algorithm is quite slow to run and does not always gives a correct result.

Indeed, is it possible to study the result on the first test sentence in the FIGURE 4. The results precall and precision are obtained thanks to the `EVALB` package. As we may see, the precision is really low.
To obtain better results, we may improve the accuracy of the CYK algorithm. An other source of the problem we encountered may be the OOV module. We could think about other ways to compute the final distance given the Damerau-Levenshtein distance and the cosine similarity. Maybe a way to improve the efficiency of the algorithm is to give more importance to one of the two distances, depending on the unknown word we are studying.

[2] https://en.wikipedia.org/wiki/Damerau%E%80%93Levenshtein_distance

[3] https://en.wikipedia.org/wiki/CYK$_{algorithm}$

## References

[1] E. L. Bird, Steven and E. Klein. Natural language processing with python. *O'Reilly Me-dia Inc*, 2009.

## Annexe

```
K most similar tags of: universite
[NC, NC]
```

Figure 2: Results of the K most similar tags of the OOV module

```
let the input be a string I consisting of n characters: a_1 ... a_n.
let the grammar contain r nonterminal symbols R_1 ... R_r, with start symbol R_1.
let P[n,n,r] be an array of real numbers. Initialize all elements of P to zero.
let back[n,n,r] be an array of backpointing triples.
for each s = 1 to n
  for each unit production R_v →a_s
    set P[1,s,v] = Pr(R_v →a_s)
for each l = 2 to n -- Length of span
  for each s = 1 to n-l+1 -- Start of span
    for each p = 1 to l-1 -- Partition of span
      for each production R_a → R_b R_c
        prob_splitting = Pr(R_a →R_b R_c) * P[p,s,b] * P[l-p,s+p,c]
        if P[p,s,b] > 0 and P[l-p,s+p,c] > 0 and P[l,s,a] <  prob_splitting then
          set P[l,s,a] = prob_splitting
          set back[l,s,a] = <p,b,c>
```

Figure 3: Probabilistic CYK Algorithm

```
Input sentence:
- Février 2005 : le parquet de Paris requiert un non- lieu en_faveur_de Jean Tiberi , accordé par
le juge Armand Riberolles .


Result sentence:
(SENT (AP|<PONCT-NP> (PONCT -) (AP|<NP-PONCT> (NP (NC Février) (NC 2005)) (AP|<PONCT-NP> (PONCT :)
(AP|<NP-PONCT> (NP (DET le) (NP|<NC-PP> (NC parquet) (PP (P de) (NP (NPP Paris) (VPpart (VN (V
requiert) (VN|<NP-VPP> (NP (PRO un) (NP (DET non-) (NP|<NC-PP> (NC lieu) (PP (P en_faveur_de) (NP
(NPP Jean) (NP|<NPP-PONCT> (NPP Tiberi) (PONCT ,)))))))) (VPP accordé))) (VPpart|<PP-NP> (PP (P
par) (NP (DET le) (NC juge))) (NP (NPP Armand) (ADJ Riberolles)))))))) (PONCT .))))))


The recall is: 0.16666666666666666
The precision is: 0.08695652173913043
```

Figure 4: Result of the Probabilistic CYK Algorithm