KILANI Al Houceine
houcinekila@gmail.com

Assignement 1

Algorithms for Speech recognition and NLP
2019-2020

03/05/20

1 The approach :

1.1 Reading the data :

I begin with reading the data in lowercases (with induces the tags to be also in lowercase) and splitting it accordingly (using sklearn famous train_test_split). It is at this stage that I perform the **cleaning from the functionnal tags and replace every word that contains digits by #**.

1.2 PCFG :

For each sentence in the training split

- I transform the sentence into CNF using the Natural language processing ToolKit library. (Here we pay attention to get rid of the empty token that is introduced and forcing the collapsing of all the unary rules)
- Append each set of rules (an nltk grammar instance) to a list
- It is at this moment too that we extract sentences without tags for the training of the Language model needed in the OOV module.

The CNF binarisation induces some new characters ("—", "+") and therefore new tags (sent+np+nc).
I have tried to keep them and infer the grammar on the dev set but it was always giving me bad results even on short sentences. This is why i get rid of them while building the dictionary of the grammar.
I also split into two different dictionaries the terminal tokens and non terminal tokens at this stage.
While adding the rules I also keep the count to normalise them into probabilities in the end :
$P(A->B) = \frac{Count(A->B)}{\sum_T Count(A->T)}$

1.3 OOV

As we are going to see in the CYK algorithm, the first step is to retrieve what are the rules (parent TAGs) that generate each token of the sentence to infer. The problem is that that token may not have been encountered before in the training set. To solve this problem this is the approach I have chosen :

- First train a language model based on the bigrams seen in the training corpus
- if the word is among our learned vocabulary ==¿ Proceed normally
- else :
  - If there exists words within a a Levenstein distance of 4 that exist in our vocabulary
    - If the word being processed is the first word of the sentence ==¿ Pick one randomly
    - else : pick the most probable word given the previous token
  - else : look for the most similar word in the polyglot dataset that exists in the vocabulary (by taking the nearest neighbour in term of cosine distance in the 64 dimensional. embedding space).

If everything fails, we do not parse the tree.

1.4 CYK

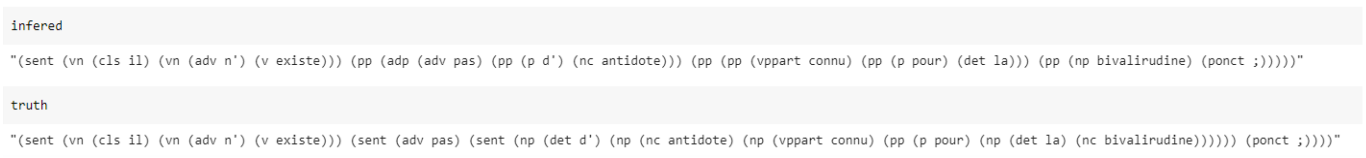
Given our extracted PCFG, I can apply a probabilistic version of the CYK algorithm to parse the sentence (https://www.youtube.com/watch?v=CFEGKVjEH1Qt=1194s). In the end, I obtain the **back** dictionary that has as values the necessary information (the most probable generating tokens of the two splits of the sentence and the index that splits the sentence) and go on recursively to reconstitutee the parsing tree.

2 Results :

2.1 Observations :

I obtain 40 PoS tags and 8158 unique vocabulary tokens.

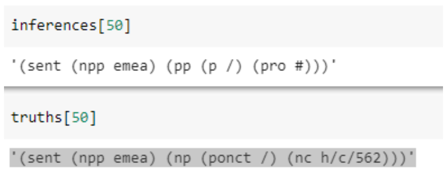
A majority of errors come for the fact that the CNF normalization introduces some weird POS (as shown in the figure below), but i could not explain the origin of the mistake... and did not have enough time to code to CNF from scratch. It happens mainly on the long sentences (maybe a parameter of NLTK that I do not set properly)



```
infered
"(sent (vn (cls il) (vn (adv n') (v existe))) (pp (adp (adv pas) (pp (p d') (nc antidote))) (pp (pp (vppart connu) (pp (p pour) (det la))) (pp (np bivalirudine) (ponct ;)))))"
truth
"(sent (vn (cls il) (vn (adv n') (v existe))) (sent (adv pas) (sent (np (det d') (np (nc antidote) (np (vppart connu) (pp (p pour) (np (det la) (nc bivalirudine)))))) (ponct ;))))"
```

Figure 1: CNF Parsing error — We see that the parsing is correctly performed if we discard the tokens wrongly induced by the CNF

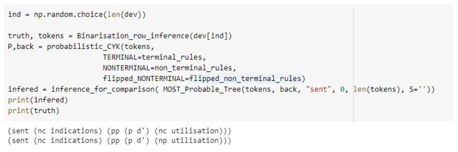
Sometimes, I do not manage to parse the tree. Indeed the OOV module I developed is too simplistic and I should have considered a more clever way to merge the : Levenstein distance, with the semantic meaning given by the polyglot and the language model. And sometimes it is the preprocessing (replacing digits) that falses the result.



```
inferences[50]
'(sent (npp emea) (pp (p /) (pro #)))'
truths[50]
'(sent (npp emea) (np (ponct /) (nc h/c/562)))'
```

Figure 2: Caption

Another issue concerns the proper nouns when they are in the start of the sentence. The way we handled the OOV leads to then being replaced randomly by a closer words in regards to the Levenstein distance which might be a noun or a verb. All these issues lead me to get a 35% accuracy on the dev set which is very low... but I believe that solving that problem in the binarisation will increase considerably the performance.



```
ind = np.random.choice(len(dev))

truth, tokens = Binarisation_row_inference(dev[ind])
P,back = probabilistic_CYK(tokens,
                           TERMINAL=terminal_rules,
                           NONTERMINAL=non_terminal_rules,
                           flipped_NONTERMINAL=flipped_non_terminal_rules)
infered = inference_for_comparison( MOST_Probable_Tree(tokens, back, "sent", 0, len(tokens), S=''))
print(infered)
print(truth)

(sent (nc indications) (pp (p d') (nc utilisation)))
(sent (nc indications) (pp (p d') (np utilisation)))
```

Figure 3: Correct prediction

2.2 Paths for improvements

:

- Speed up things by avoiding to leep over disctionnaries (especially in the OOV module), maybe leep the first time on the dev/test set to spot the OOV words and then replace them adequately before going into the CYK.
- Combine the Levenstein distance and the unigrams/bigrams probabilities of the language model in one unique score and retain the terms in the vocabulary that maximize this score of replacement (instead of just computing if/else statements and handmade rules like we did)
- Try the Damereau Levenstein distance that allows also the inversion of letters in the tokens.
- A less harsh preprocessing (brutely replacing some tokens by others is not always a goog solution Cf Figure 1)
- Use NLTK data structure to induce the pcfg in CNF (I realised it too late that it was possible to do that but i had to recompute all the code I have already written up to that moment)