NLP homework 2 - Lapassat Louis

The aim of this project is to build simple French probabilistic parser using a Probabilistic Context-Free Grammar (PCFG), fitted on the first 80% data from SEQUOIA treebank v6.0, in addition to a Out Of Vocabulary (OOV) words module to deal with unseen words during the training phase. The parsing method is the Cocke–Younger–Kasami algorithm (CYK).

1 Modules descriptions

Here we briefly describe the main modules that we are using for the project. Even if we didn't implemented everything by ourselves (for instance the grammar is extracted using NLTK library), the methods/algorithms are fitted for our case in order to optimize our running time.

1.1 PCFG

To extract the grammar from the sentences (dataset *sequoia-corpus*) we choose the library NLTK. Here we simply add to transform the string sentence in tree (calling the *Tree* function from NLTK) and infer the grammar with the productions. Of course we already treated the dataset, i.e. we removed the functional labels as mentioned in the GitHub (we used regular expression for both rapidity and simplicity). Finally to fit the grammar for our case we choose to collapse the collapse unary productions and transform the trees in a Chomsky normal form.

1.2 OOV

For this part we had to use two scores, the cosine similarity and the Damereau Levenshtein distance, to assign for a given unseen word (with respect to the training dataset) a part of speech. For the cosine similarity we use the following formula:

$$\text{similarity} = \frac{xy^T}{||x||_2||y||_2},$$

where x and y are respectively the embedding for the first word and for the second one. To implement the Damereau Levenshtein (DL) distance we simply used the slides showed during the class and the slides. We also added the Levenshtein distance to do some benchmark between these two distances. As the cosine similarity spans in $[-1, 1]$ and the DL distance in $[0, +\infty)$ we choose to map the DL distance in $[-1, 1]$ in order to use the two scores together. To do so we used the following scaling:

1. DL distance = - DL distance: simply because if the DL distance is big it means that the two words are not really similar, for this metric.
2. $2 \times \frac{\text{DL distance} - \min(\text{DL distance})}{\max(\text{DL distance}) - \min(\text{DL distance})} - 1$: map the DL_distance in $[-1, 1]$.

Finally we just compute an average score like so:

$$\text{final score} = \frac{0.5 \times \text{DL distance} + 0.5 \times \text{cosine similarity}}{2},$$

and the top k words (so the ones with the highest final score) are selected. At the end we just loop over the selected words until we found one that is present in our training set and choose the most probable part of speech.

1.3 CYK

To implement the parser we mainly used the chapter 12 and 13 from *Speech and Language Processing, Daniel Jurafsky and James H. Martin.* We simply fill the matrices (we used dictionaries because they are easy to handle and access) with both probabilities and the rules. At the end we reconstruct the most probable tree (we parse the sentence).

2 Results and analysis

As expected the grammar and lexicon are really fast to create, simply because the libraries we used are well optimized. Interestingly playing with *horzMarkov* (Markov order-N smoothing) does not change the precision (score, explanation further below) too much, we tried with 1 and 2 without strong impact on the final precision score.

```
There is 12706 productions in your grammar. Below is a quick look at some of them:

[SENT -> NP+NPP [0.00242033],
 NP+NPP -> 'Gutenberg' [0.00247525],
 SENT -> NP SENT|<VN> [0.137959],
 NP -> DET NC [0.220609],
 DET -> 'Cette' [0.00236469],
 NC -> 'exposition' [0.00120797],
 SENT|<VN> -> VN SENT|<Ssub> [0.0439952],
 VN -> CLO V [0.0189378],
 CLO -> 'nous' [0.06],
 V -> 'apprend' [0.000483325]]
```

```
There is 8958 keys in your lexicon. Here is a quick look at him:

{'prouvé': {VPP: 1.0},
 'affirmait': {VN+V: 1.0},
 'daté': {VPP: 1.0},
 'valider': {VN+VINF: 1.0},
 'filières': {NC: 1.0},
 'complémentarité': {NC: 1.0},
 'Soulignant': {VN+VPR: 1.0},
 'Tiananmen': {NP+NPP: 1.0},
 'rencontrés': {VPP: 1.0},
 'au_cours_du': {P+D: 1.0}}
```

Figure 1: Production of our grammar (left) and the lexicon (right)

The OOV module does pretty well at selecting the most similar words given an unseen word (and assigning a part of speech). That was also expected since we load a large set of words (in fact 100004 words are loaded) and prelearned embeddings. The only big problem we faced with this module was the time needed to compute the part of speech (and so the k-most similar words). In order to speed up the process we choose only to compute the DL distance for the words in the top 20% with respect to the cosine similarity. In practice we dropped from 2.7 s to 0.7 s which is a huge improvement (time equivalent to one module call). But the main drawback was that even if the module is doing create for misspelling, we can see some little changes in the selected words:

```
: oov.most_similar_words('tres')
['trés', 'trop', 'très', 'assez', 'very']

: oov.most_similar_words('etre')
['être', "m'être", 'estre', 'ete', 's’être']
```

speed = False

```
: oov.most_similar_words('tres')
['trés', 'trop', 'très', 'assez', 'infiniment']

: oov.most_similar_words('etre')
['être', "m'être", 'estre', 'ete', "qu'être"]
```

speed = True

Figure 2: Most similar words without speed option on left and with on right ($k = 5$)

It was expected as by doing so we increase our variance but the results are not changing that much. Finally after testing both method on the evaluation dataset we choose to stick we the "speed up" version mainly for time reason even if we drop a little bit of our final precision (we still think it is a good trade-off between score and time).

Parsing is achieved for most of the evaluation sentences but we have some "outliers" (some sentences can not be parsed). If we take a look at these sentences it seems like the combination of OOV words and rare words (i.e. not common, thus hard to find/assign a correct part of speech) that leads to this situation: "Not correctly parsed." (the parser did not achieved its job). But on the other hand the parser seems to handle misspellings correctly. We think that by rethinking our OOV module we could have achieved a better result, with respect to the number of sentences that the parser can actually parse. For instance our idea of averaging the two scores is maybe time efficient but it is maybe to simple and maybe using a method based on K-nearest neighbors (for example K-d tree) is the way to go. Also it could have been interesting to perform a grid search on the weights (in the averaging of the two scores, $0.5, 0.5$), but because it was really long to run we did not done it (we still think that it is a first step before changing completely to method). Below we sum up our main results (using evalb):

	sentences parsed	overall precision in $[0, 1]$	parsed precision in $[0, 1]$
score			