Natural language processing and speech algorithms : Homework 2

Matthieu Plaszczynski

March 9, 2020

1 How things work

I have chosen to code everything by myself. I could have used NLTK, but it seemed not so much adapted, because the required work was very precise. It looked like using NLTK would not only have consisted in applying some precoded method, but also imposing a whole architecture, that could have been irrelevant to the problem. Thus, recoding everything was a solution because it let me implement just the things that I needed and know exactly what my algorithm is doing.

Step 1 : computing the PCFG I have implemented two useful classes : one is a Tree structure, the other one is the PCFG one. I begin by computing a tree corresponding to the input sentence : the leafs are the PoS (with an additional attribute being their token), and the internal nodes are made of non-terminal symbols extracted from the sentence. After that, I can iterate on the nodes in order to add all the new production rules to the PCFG grammar. I also update the PCFG lexicon according to the tokens that have just been treated. The PCFG class is thus composed by a lexicon and the production rules. Both are implemented as dictionnaries of dictionnaries. I have also added the classical grammatical information to the PCFG attributes : the S symbol, the non-terminal terms and the terminal terms (PoS then).

Step 2 : converting it into Chomsky Normal Form I have implemented it in 3 steps, as written in the Wikipedia article about CNF. I begin by doing the TERM phase, then the BIN phase, and finally the UNIT one. All of these methods consist in changing the keys in the dictionnaries of the existing PCFG.

Step 3 : the OOV module First of all, I check if the input word is in the lexicon. If it is not, I check if there is words in the lexicon that are at a Levenshtein distance of 1 to the input word : if so, I take the most likely PoS tag in all the possibilities. If not, I start using the Polyglot embeddings : I compute the 8 nearest neighbours to the input word (according to the embedding) that are already in the lexicon. I then chose the most likely PoS tag in all the possibilities. If the input word does not belong to the Polyglot dictionnary, I

find the closest word in the Polyglot dictionnary (according to the Levenshtein distance), and I treat it as before.

Step 4 : the CYK algorithm I begin by creating a tree for every possible PoS tag of every token composing the input sentence. After that, I have implemented a CYK algorithm that, for every step k computes every possible tree for $S_{i,i+k}$ (for all i) and then only keeps the 40 most likely trees (for each i) before doing $k \to k+1$. When it reaches $k = n$ the number of input tokens, it looks for every tree in $S_{0,n}$ with SENT as root symbol, and outputs the most likely tree (which corresponds to a parsing).

2 Results

The results seem very good both for the quantity of parsed sentence and the accuracy of the choice of the PoS. The main drawback is the time of computation. The choice of keeping the 40 most likely trees at each iteration of the CYK algorithm is indeed very costful. If we change this number to 10 for example, the algorithm is really faster, but it fails a lot to parse sentences that the previous version was successfully parsing (the proportion of parsed sentences with this version is around 15%.