Algorithms for Speech and NLP — TD 2 (NLP)

Hugo Cisneros

March 18th 2019

1 System description

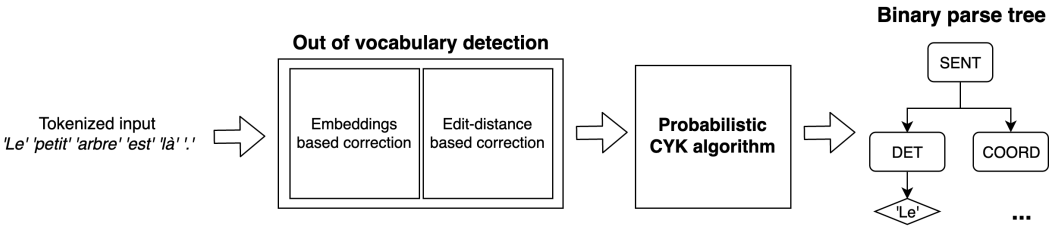

Figure 1: System overview

The work I carried out for this project can be divided into two parts:

- Build a tree parsing system that can construct and save a PCFG from a given treebank.
- Build the parser that uses the PCFG to parse input sentences. This system is illustrated on Figure 1 and is composed of an a first part handling out-of-vocabulary (OOV) words by replacing them with appropriate other words, and an implementation of the probabilistic CYK algorithm as described by Jurafsky and Martin in the draft of the book *Speech and Language Processing* [1]

1.1 PCFG building

The custom tree parsing system is based on detecting regular expressions and progressively substituting them to reduce the tree from bottom to top.

Example: `(SENT (NP (NPP Paul)) (V mange))` is first replaced by `(SENT (NP (NPP)) (V))` and then `(SENT (NP) (V))` and finally `(SENT)`. At each step all elements substituted are added to the grammar as rules.

After, the grammar is transposed into Chomsky's normal form to be ready to use for the CYK algorithm.

1.2 OOV substitution

OOV word substitution is done in two steps. If an unknown word exists in the embedding dictionary (we used the French polyglot embeddings[2]), the closest (cosine distance wise) word from the lexicon is used as substitute. This allows to replace words by others that have similar meaning and function in a sentence.

[1] `https://web.stanford.edu/~jurafsky/slp3/`

[2] `https://sites.google.com/site/rmyeid/projects/polyglot`

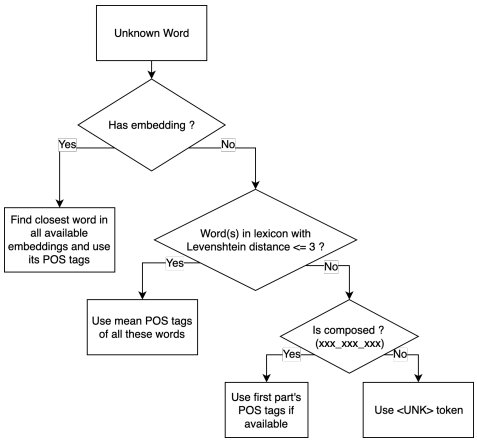

Figure 2: OOV module overview

In the cases where the word isn't in the embedding dictionary, its POS tags are found by averaging all possible POS tags and probabilities from words within a levenshtein distance of 3 form the original word. This allows to deal with small typos that are common in user input, while avoiding some mistakes that could arise when replacing a word by its closest neighbor.

For unknown group tokens of the form `xxxx_xx_xxx` that failed the two first tests, the first part of the token `xxxx` is selected.

For other words, an `<UNK>` token mapping to all possible POS tags with equal probability is used to make sure that completely unknown words can still lead to a grammatically valid sentence. Figure 2 summarizes the whole system.

2 Error analysis

The resulting system sometimes cannot parse some sentences it consider grammatically incorrect. The CYK algorithm is unable to correctly build the tree up to the start symbol. For example, with the following sentence `Le juge d' Amiens se dessaisit du dossier.`, the token `dessaisit` is replaced by the word `dessaisi` by the OOV module, because it is only at a levenshtein distance of 1 from the original word. However, the function and POS tags associated with this new word are radically different from the original ones and the sentence `Le juge d' Amiens se dessaisi du dossier.` cannot lead to a grammatically correct sentence when parsed with the CYK algorithm.

Similarly, the sentence `Dans la maladie de Paget , le remodelage osseux est trop rapide et l' os nouveau se forme de façon désordonnée , ce qui le rend plus faible que l' os normal .` which has particularly intricate structure, cannot be built from the learned grammar.

These two example are caused by sparsity issues with the model, and a bigger training set of both words and sentences would help cope with these problems.

The model could also be further improved by adding some refinements to the OOV module, since it cannot handle edge cases. Word embeddings have a very high accuracy at finding the closest word in terms of function in the sentence whereas levenshtein distance is good at correcting spelling errors but can also lead to problematic situations.