Audio speech recognition and NLP TP2

Guillo Clément

March 8th 2020

This report aims at explain the choices I made in order to create a parser based on a probabilistic grammar and which is robust to unknown words.

1 Description of the system:

1.1 Construction of PCFG

The construction of the probabilistic context free grammar from the train corpus is performed in the CorpusManager class of my code and is only based on the first 80 % of the treebank provided. I first transform the sentences with brackets, one by one into a tree in a recursive way. From these trees I first extract the lexicon which is a list of triplets (word|part-of-speech(POS) tag|probability) and I then extract a grammar which terminals are POS tags. In order to do so. I increase the count of the rules of my grammar according to the transitions I find in each tree before normalizing the obtained grammar in order to have for each left-hand side L a distribution of probabilities over the right-hand sides R.

I then put my grammar in the Chomsky Normal Form (CNF): In order to do this, I first handle the "non-solitary terminal" rules with creating new non-terminal nodes, then I handle the non-binary rules by splitting the initial right-hand side until they are only two POS tags as result. Finally I finish with correcting the unary-rules which give a non-terminal part of speech (POS).

At each step I try to keep my grammar probabilities relevant. As an example: in the configuration, $A \rightarrow B$ (unary rules with non-terminal right-hand side) and $B \rightarrow (C, D)$, I erase the unary rule and I compute the probability $P(A \rightarrow (C, D))$ with:

$$P(A \rightarrow (C, D)) = P(A \rightarrow B) \times P(B \rightarrow (C, D))$$

If the rule $A \rightarrow (C, D)$ already exists, I sum the former probability and the new calculated probability. It sometimes happens that after the unary rules' correction, some unary rules with non terminals still remain on the grammar. In order to avoid problems later on, I decided to erase these rules since it happens only when a POS on the left gives itself on the right or when two POS lead to each other by an unary rule, which is rare.

1.2 Probabilistic CYK algorithm

The Cocke-Younger-Kasami algorithm is usually used to find out whether a given sentence is possibly generated by a grammar in Chomsky Normal Form. Here the previously constructed grammar is probabilistic so it is possible to select the most likely generator tree for each sentence and thus to use CYK as a parsing algorithm.

However, since the terminals of the grammar are POS tags, a first step is necessary to plug the POS tags to the sentence's words using the built lexicon. Using a back table it is then possible to reconstruct the most likely bracket sentence for a given input sentence. However, in order to have comparable results, I modify the obtained bracket sentence in order to make it contain only POS tags which had not been created during the Chomsky Normal Form transformation process.

The time complexity of the CYK algorithm is $\mathcal{O}(n^3)$ with n the sentence length, it can take a lot of time on big sentences. I try to handle this problem using dictionary research instead of list research but the whole process remains slow for big sentences. The CYK algorithm takes as input a sentence which has all its words belonging to the lexicon built from the train. That is why the "Out of vocabulary" module described below permit to replace unknown words of a sentence by words on the train set.

1.3 OOV module

The OOV module combines the Damerau-Levenshtein distance and the cosine similarity in order to find candidate words in the train set which will be able to replace an unknown word in a given sentence we want to parse.

For each word that does not appear in the train set, my OOV module finds on the train set n_{cos} similar words obtained with the cosine similarity (using embeddings) and n_{DL} nearest neighbors obtained with the Damerau-Levenshtein distance. In order to compute the cosine similarity, the unknown word has to belong to the words contained on the embedding file. I can then compute the cosine similarity between the unknown word and the words belonging to both the train set and the embeddings file. Since a lot of words can give the same Damerau-Levenshtein distance, the candidates coming from this distance are selected according to their position in the lexicon.

The candidates from the two distance are then evaluated locally in the sentence, using transition probability computed using a bigram model and a unigram calculated from the transitions and words encountered in the train set.

$$P(w_k|w_{k-1}) = \lambda P_{uni}(w_k) + (1-\lambda)\left(\frac{\#(w_k|w_{k-1})}{\#w_k}\right)$$

The OOV parameters n_{cos}, n_{DL} and λ can be tuned. In order to do so, and because of the time complexity of the CYK algorithm, I run the whole process on a subset of 90 sentences randomly selected from the 10 % validation part of the provided tree-bank and among the sentences with less than 20 words to be able to compute a grid search.

The hyper parameters which give the best results are, $n_{cos} = 30$, $n_{DL} = 4$ and $\lambda = 0.05$. This choice of parameters is then used to parse the last 10 % sentences of the provided treebank. At the end, the OOV module produces another sentence which will be the input of the probabilistic CYK algorithm.

2 Performance and Error analysis:

We obtain a score of 90.5 % on the test set with a bracketing recall of 60.1 % and a Bracketing precision of 58.2 %. Out of the 310 input sentences, 20 are not parsed because they do not seem to be generated by the previously constructed grammar. A part of the errors made by the parser can be explained by the poor performance of the OOV module.

Indeed, since our module does not modify words belonging to the train set, true-word errors are not handled by the module. The nearest neighbors from the Levenshtein distance are computed among the words of the train set but it might be more judicious to extract the nearest neighbors from the words present in the embeddings file and then find the nearest neighbors in the train set but it is very expensive in computational time.

The candidates found with the cosine similarity computed from embeddings can have very close meanings but their associated POS tag is different. As an example, the common noun "alimentation" can be closed to the verb "eat" without having the same POS. Moreover if the transition between two words is not frequent, the selection among candidates with language model is not relevant.

In order to improve the OOV module, instead of using directly the Damerau-Levenshtein distance, one could generate words at distance 1 of the unknown words and see if it does belong to the embeddings words which may be more relevant than just computing the distance with the training words. Here, I do not accept that sentences are generated by any other start symbol than "SENT" which can be restrictive when new sentences occur. Allowing more initial POS could allow parsing of these sentences.