TP2 - REPORT

By : Thibault Lahire (thibault.lahire@student.isae-supaero.fr)

Abstract
In this practical assignment, a basic probabilistic parser for French is developed. Based on the CYK algorithm and the PCFG model, it is robust to unknown words. Libraries such as NLTK were used. In this report, we reuse the notations of the book Speech and Language Processing (Jurasky and Martin). The CYK algorithm has been implemented thanks to the pseudo-code in `http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf`

1 Developed system

1.1 Building a PCFG

The rules $\alpha \longrightarrow \beta$ and the non-terminals α encountered in the training corpus are counted, in order to compute $P(\alpha \longrightarrow \beta) = \text{Count}(\alpha \longrightarrow \beta)/\text{Count}(\alpha)$. We can then build the PCFG, i.e. we are able to assign a probability of a parse tree T to a sentence S. Indeed, denoting the i-th rule $LHS_i \longrightarrow RHS_i$, $1 \leq i \leq n$, we have: $P(T, S) = \prod_{i=1}^{n} P(RHS_i|LHS_i)$.

1.2 CYK Algorithm

The CYK algorithm looks for the most likely parse tree $\hat{T}(S)$ for the sentence S, given the PCFG we built on the training set. Following Jurafsky's book, the CYK algorithm solves :

$$\hat{T}(S) = \arg \max_{T \text{ s.t. } S \text{ yields } T} P(T, S) = \arg \max_{T \text{ s.t. } S \text{ yields } T} P(T)$$

The algorithm processes CNF (Chomsky Normal Form) trees, that's why we only considered the probabilities of unary and binary rules when building the PCFG. As required in the TP handout, functional labels were ignored. The algorithms takes a sentence (a list of tokens) as input and outputs the bracketed format of the training data.

1.3 OOV module

The aim of the Out-Of-Vocabulary module is to find the best correction of a word which is not in the vocabulary. To do this, we first reuse the code provided in the TP handout, enabling our system to deal with numbers and errors such that capital letters thanks to the computation of a distance in the embedding space provided with polyglot. This provided code returns an ordered list of potential candidates, called embedding candidates.

Then, we use the Levenshtein distance. Indeed, all possible sequences of strings under a certain distance to the unknown word are computed. In practice, this means we consider all possible substitutions, insertions and deletions. We choose a distance of 1 for faster computations and we did not implement the case of a distance larger than 1.

Finally, we combine these two set of canditates (embedding and Levenshtein) to extract the best correction of the unknown word. We use the fact that embedding candidates are ordered to retrieve the first which is also in the set of Levenshtein candidates. If it is not possible, we return a random Levenshtein candidate, and if there is no Levenshtein candidates we return the first embedding candidate.

2 Results

2.1 Performance

We use the provided SEQUOIA treebank that we split into 3 parts (80% training, 10% validation and 10% testing) as required. We did not use the validation set but we propose in the last section a possible use. Using pyevalb, we obtain on the test set an accuracy of 61.5%, a precision of 78.71%, a F1 score of 68.09%, and a recall of 61.5%. The number of mistakes with respect to the length of the sentence is shown in Fig. [1], and one can execute the oov.py file independently of the rest of the system to see the results of the OOV module reported on Fig. [2].

We see on Fig. [1] that the number of mistakes does not necessarily increase with the length of the sentence. We have a robust system, even if we expected that the number of mistakes increases with the length of the sentence, since the number of possible errors is higher for long sentences.

We see on Fig. [2] that the OOV module perfectly works when the Levenshtein distance between the unknown word and a known word is 1. However, for a larger distance, the system fails, since it has not been implemented for.

2.2 Evaluation on a small set

We evaluate the parser on some sentences :

- For the sentence "Monsieur Macron a pris cette décision.", the parser does not know Macron and replaces the word.
- "Le parquet a requis des peines de prison." is correctly parsed, all the words are known by the system and there is no error.
- The parser recognizes the mistake in "Le procureur a obtenu la protecton." and transforms "protecton" in "protection".

2.3 Possible improvements

- The training could be done on a larger dataset to have a larger lexicon, yielding in a more robust parsing.
- The data structures used for the implementation of the CYK algorithm can be optimized: the parser is not very fast currently.
- In the OOV module, instead of what we did, a bigram language model could have been used. Such a model would assign a probability of a correction given the previous word in the sentence. Here the validation set would have been used to tune the smoothing parameters of the model.

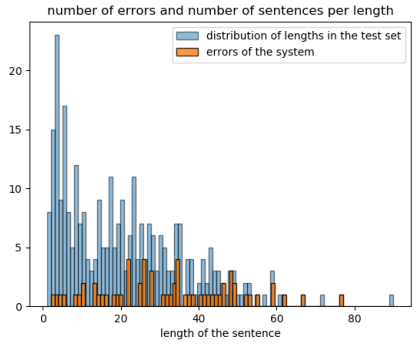

Figure 1: Parser results

```
The unknown word "constituTion" has been
transformed into "constitution"

The unknown word "anarchite" has been
transformed into "anarchie"

The unknown word "voitufre" has been
transformed into "voiture"

The unknown word "voistufre" has been
transformed into "pointent"
```

Figure 2: OOV module results