# MVA - Algorithms for Speech and NLP TD 2 (NLP)

Zhexuan HUANG
zhexuan.huang@polytechnique.edu

## Implementation of PCFG parser

In our experiment, we use SEQUOIA treebank v6.0 dataset and split it into 3 parts, 80% for training, 10% for validation and final 10% for testing.

### 1. Extract PCFG from the training corpus (PCFG module)

We use training set to extract our PCFG features. This is carried out by using nltk package:

- Firstly, given a sentence, we convert it into nltk Tree by using Tree.fromstring() method provided by nltk.
- Then we further convert nltk Tree into Chomsky normal form by using Tree.chomsky_normal_form()
- Finally, given a nltk Tree with Chomsky normal form, we can use Tree.productions() to extract rules followed by the tree. We go through the entire training set, extract all rules in training and calculate the probability of each rules. More precisely, the probability of a specific rule $A \to B$ is calculated by:

$$P(A \to B|A) = \frac{\text{The number of } A \to B \text{ in training set}}{\text{The number of } A \text{ in training set}}$$

### 2. Deal with Out-of-vocabulary (OOV module)

OOV module aims to assign a part-of-speech to any token which is not presented in the lexicon extracted from the training set.
This is carried with the help of Polyglot embedding for French [1]. At each time we meet a part-of-speech which has not occurred in lexicon:

- We firstly try to embed it Euclidean space by using Polyglot embedding data. We also embed all of the lexicons into the same Euclidean space. Then we try to find the k-nearest neighbors of the part-of-speech from the lexicons by calculating their Euclidean distances.
- If embed the part-of-speech $w$ by Polyglot embedding is not possible, we can calculate a set of words $S(\mathrm{n})$ whose elements have Levenshtein distance from $w$ smaller then $n$. Then we try to embed elements in $S(n)$ to Euclidean space and return k-nearest neighbors of embedded elements.
- Finally, if all of the elements in $S(n)$ can not been successfully embedded into Euclidean space, we return an unknown word symbol <UNK>.

We use our validation set to deal with <UNK>. We extract a set $S$ which contains all of the terminal rules $A \to w$ in validation set where $w$ is a terminal word and $w$ cannot be dealt with OOV module (i.e. $w$ is <UNK>). Then we calculate the probability of <UNK> by:

$$P(A \to < UNK > |A) = \frac{\text{The number of } A \to < UNK > \text{ in } S}{\text{The number of } A \text{ in } S}$$

### 3. CYK algorithm

Once the PCFG probabilities as well as <UNK> probabilities are extracted, then for every given tokenized sentence $s$ as input, probabilistic CYK algorithm aims to solve the problem:

$$\mathrm{T}(s) = arg \max_{T \text{ yield } s} P\,(T)$$

This can be done by using dynamic programming and the probabilities learned on training data during the PCFG phrase.

## Result and Conclusion

As described before, we use SEQUOIA treebank v6.0 dataset and split it into (80%, 10%, 10%) for training, validation and testing. We use pyevalb package to evaluate our result and following is the result obtained over test set:

```
Number of sentence:     310.00
Number of Error sentence:   39.00
Number of Skip sentence:    0.00
Number of Valid sentence:   271.00
Bracketing Recall: 63.15
Bracketing Precision:   61.27
Bracketing FMeasure:  62.20
Complete match:   12.55
Average crossing:  3.06
No crossing:   45.76
Tagging accuracy: 93.37
```

The F1 score of our algorithm over test set is 62.2%. To improve the performances, we believe the simplest way is to use a larger dataset to include more lexicons, since a lot of out-of-vocabulary are observed in test set. Another possible solution is to include plural rules into our rule bank.