INGI2263 – Computational Linguistics

Assignment 3

Probabilistic parsing

1 Introduction

In this assignment, you will be asked to train a Probabilistic Context-Free Grammar (PCFG) from a treebank corpus and to use it with your own implementation of the CYK algorithm in order to parse a new set of sentences.

The corpus provided is a modified version of the QuestionBank corpus [1] which encompasses the parsing annotations for 4,000 questions. For instance, the annotated sentence

(SBARQ (WHNP what)(SBARQ (SQ (VBD caused)(NP (DT the)(NP (NNP lynmouth)(<NN> floods))))(<?> ?)))

corresponds to the parse tree depicted by Figure 1.

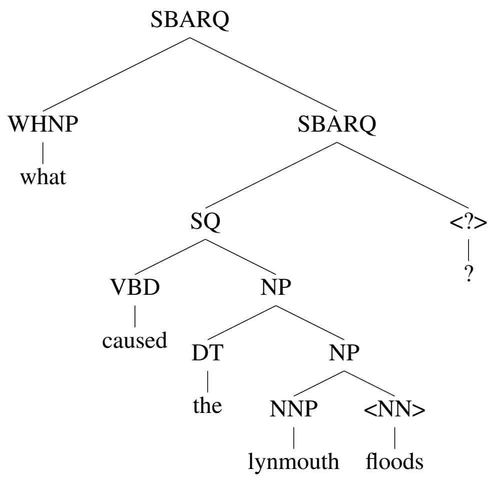

Figure 1: Sample parsing tree.

The labels of each internal node consist of non-terminal symbols while the labels of leaf nodes are terminal symbols, that are word types of the vocabulary.

This corpus is provided as five files:

- `train.txt` contains the annotated sentences **with the <unknown> token** (see Section 2) of the training corpus;
- `train_with_annotations.txt` contains the original training corpus;
- `train_without_annotations.txt`;

- `test_with_annotations.txt`;
- `test_without_annotations.txt`.

The first file must be used to train the grammar, while the last 4 corpora are provided to assess the performances of your parser. These files are readily available from the course website (see `QuestionBank.zip`).

> In practice, you are asked to turn in a report (as a `PDF` file) addressing all the points mentioned in a frame in this document.

2 Grammar training

The first step of the assignment consists in training the PCFG from `train.txt`. A PCFG grammar can be defined as a tuple

$$G = (N, \Sigma, R, S)$$

where N is the set of non-terminal symbols, Σ is the set of terminal symbols, R is the set of rules (each rule has an associated probability) and S is the start symbol. In our case, the start symbol S is `SBARQ`.

As presented in class, the CYK parsing algorithm requires the grammar to be in Chomsky Normal Form (CNF). Thankfully, the corpus has already been processed for you to make sure that all the derivations that it contains are CNF compliant. In addition to that processing, the corpus has been modified in two ways. Firstly, the less frequent words (less than three occurrences) have been replaced by a new terminal symbol: `<unknown>`. This symbol will be used to estimate the emission probabilities of out-of-vocabulary words (words in the test data that are not present in the training data). Secondly, the naming conflicts between terminals and non-terminals has been eliminated for the sake of simplicity.

The set of rules is computed by extracting all the derivations present in the training data. The rule probabilities are estimated from the counts using the maximum-likelihood estimators:

$$\hat{P}(A \to BC|A) = \frac{C(A \to BC)}{C(A)} \qquad \text{and} \qquad \hat{P}(A \to w|A) = \frac{C(A \to w)}{C(A)}.$$

Finally, don't forget to check the consistency of the grammar: the sum of probabilities of all rules with a given non-terminal as left-hand side must be 1[1].

[1] If such a condition is satisfied the PCFG is said to be *proper*. The true notion of consistency is stronger and means that the sum of the probabilities of all sentences that could be generated from such a model is equal to 1. In other words, a consistent PCFG defines a probability distribution on the set of sentences it generates. Properness is a necessary condition for consistency of a PCFG but further technical conditions must also hold. We will ignore those technical considerations here and only check properness.

Train a PCFG from the training data *train.txt* and report:

- The size of the vocabulary (= the number of terminals), the number of non-terminals, and the total number of rules.
- The list of rules with `WHADJP` as left-hand side and their estimated probabilities. Are those probabilities consistent?

The introduction of the terminal `<unknown>` is useful to deal with unseen words when parsing new sentences. Let's now imagine that there is no word with a number of occurrences smaller than 3. In that case, can you come up with another way to deal with new words (other than increasing the threshold on the word count)?

3 Probabilistic CYK parsing

The second part of the assignment consists in implementing the probabilistic CYK algorithm as introduced in class. However, for practical reasons, we introduce a small modification. Instead of maximizing probabilities, the algorithm will play it safe and maximize log-probabilities. In order to make your results comparable with ours, let's agree on the natural logarithm (base e). This transformation does not affect the parser output. Indeed, since the logarithm is a strictly increasing monotonic function, we have

$$\hat{T} = \underset{T}{\operatorname{argmax}} \ln\left[\hat{P}(s, T|G)\right] = \underset{T}{\operatorname{argmax}} \hat{P}(s, T|G)$$

where G stands for the grammar, s is a sentence and $\hat{T}$ is a most likely parse tree of such sentence.

To ease your implementation of CYK, your trained grammar should provide a convenient way to reverse the derivation rules. In other words, a simple way to find the possible left-hand sides from the right-hand side of a rule. Finally, recall that we are only interested in parse trees rooted with the `SBARQ` symbol.

Report the parse tree[2] and the log-probability of one sentence of your choice in `test_without_annotations.txt` of 10 words at least. Report as well the parse trees and log-probabilities for the two following sentences:

- *what sport features snatches and clean jerks ?*
- *who is the prime minister of australia ?*

This second sentence is part of the training set. How does the result of your parser compare to the annotation in `train_with_annotations.txt`? Comment.

From a computational point of view, what is the most tractable approach between the probabilistic or the non-probabilistic CYK algorithm ? Why ?

Using your implementation, can you produce the 5 most likely parse trees for a given sentence ? If not, how would you proceed ?

[2] If you are latex fluent, you might find the *qtree* package helpful.

4 Performance assessment

In the last step of this assignment, you are asked to measure the performance of your parser. This can be done by comparing the parsing produced by your implementation to a reference parse tree.

To perform this comparison, the trees are first decomposed into constituents that indicate for each non-terminal in the tree which part of the sentence it produced. In practice, the constituents have the form $(nt, start, end)$ where $start$ and end are the starting and ending indices of the phrase spanned by the non-terminal nt. In addition, only the non-terminals that span at least two words are considered. For instance, the decomposition of the trees depicted in Figure 2 is reported in Table 1.

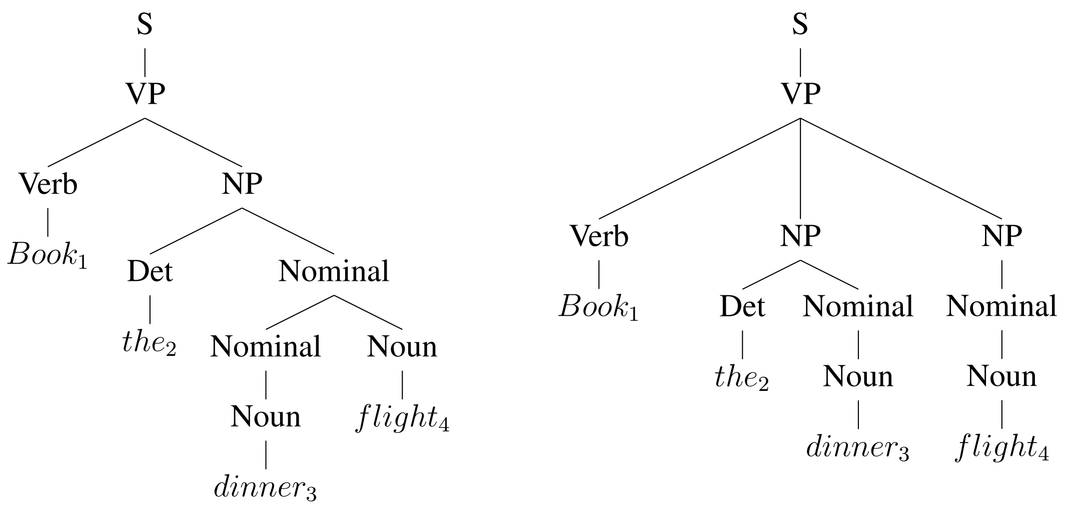

Figure 2: Two alternative parse trees of the same sentence.

Tree 1	Tree 2
(S 1 4)	(S 1 4)
(VP 1 4)	(VP 1 4)
(NP 2 4)	(NP 2 3)
(Nominal 3 4)	

Table 1: Constituents of trees depicted in Figure 2.

The performance assessment is then carried out by comparing both sets of constituents with two complementary measures: the recall and the precision. If R and P are respectively the number of constituents in the reference and in the parser output, and C is the number of correct constituents, precision and recall are defined as $precision = C/P$ and $recall = C/R$. Finally, those measures can be summarized into a single one called the F1-score using the harmonic mean:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

In our example, if Tree 1 is the gold standard and we evaluate Tree 2 , $recall = 2/4 = 0.5$, $precision = 2/3 \simeq 0.667$, and $F_1 \simeq 0.571$.

In this case, the measures compare only two trees, but they can be readily applied to a corpus of trees by aggregating counts. It is likely that some sentences in the corpus cannot be generated by the grammar. Still, these sentences must be taken into account. In that case, the number of constituents C of the parser output is considered to be zero. That leads to a recall of 0 ($0/R$) and to an undefined precision ($0/0$). But thankfully, aggregating counts over the whole corpus effectively circumvent the issue with precision.

You are asked to compare the parsing results of `test_without_annotations.txt` with the reference annotations in `test_with_annotations.txt`, and to report the precision, the recall, and the F1-score. Report these same performance measures when parsing the train sentences in `train_without_annotations.txt` and comparing the result to `train_with_annotations.txt`.

5 Practical information

For any question related to this assignment, you are invited to contact the teaching assistant: `charlotte.hansart@uclouvain.be`

In order to submit your work (in the form of a PDF report), you must be member of a group on the Moodle website for this course `http://moodleucl.uclouvain.be/course/view.php?id=7865`

References

[1] John Judge, Aoife Cahill, and Josef Van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics, 2006.