Probabilistic parsing TD2

w.mossaab Wiklandour

1 Algorithms and implementation

1.1 Parser

The aim of our parser is to first extract a pcfg from the sequoia tree bank. For a rule $lhs \to rhs$ we estimate the probability using a frequency approach with :

$$P(rhs|lhs) = \frac{count(lhs \to rhs)}{count(lhs)}$$

The CYK algorithm uses dynamic programming to solve the word problem, the main idea to consider the different splits of the sentence. Noting $k = \arg\max(size(rhs))$, the algorithm is of complexity $\mathcal{O}(n^{k+1}|G|)$ where $|G|$ is the size of the grammar (i.e number of rules). In fact, we need to fill the $\mathcal{O}(n^2)$ cells of the cyk table, to fill each one we need to consider all the splits of parts less than k in $\mathcal{O}(n^{k-1})$ and check if the split is a valid right hand side of a rule in $\mathcal{O}(|G|)$, this last check can be optimised using a reverse hash map that we implemented. This theoretically results in a complexity of $\mathcal{O}(n^{k+1})$.

Using Chomsky normal form creates an equivalent aims to samplify the implementation and limit the complexity of the algorithm by creating a bigger set of rules and limiting the size of the right hand side to 2.

This trade-off makes the implementation easier and the algorithm reaches a complexity of $\mathcal{O}(n^3|G|)$, but the grammar size jumps from 19267 to 22690. We can see below that the worst time complexity is clearly polynomial.

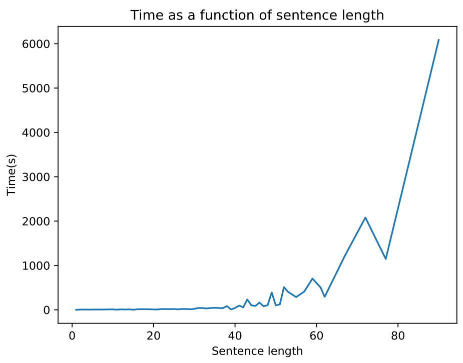

Figure 1: Worst time for sentence length

We adapt the CYK algorithm to save the combinations used to save at each time the most probable expansion for a certain grammatical symbol. This is clearly translated in the following code piece:

```
for rule, log_proba in self.pcfg.reverse_table.get((c1, c2), {}):
    # Add combination or replace if probability is greater
    if rule.gsymb not in cyk_table[length][k].keys() or \
        likelihood + log_proba > cyk_table[length][k][rule.gsymb][1]:

        cyk_table[length][k][rule.gsymb] = (rule, likelihood +
        log_proba, i, k, length - i - 1, k + i + 1)
```

This allows also to save the decomposition used in the CYK table and only keep the best one. A sentence is parsable using our grammar if the root_symbol (sent) is at the top of the CYK table. We construct then the predicted parse recursively from the top of the table while making sure not to print the dummy rules created to form the Chomsky normalisation.

1.2 Out of vocabulary handler

To handle out of vocabulary words we chose a logic we chose to do the following:

- detect the words in our terminal set that do have embedding in polyglot.
- for an OOV word find the closer word in polyglot using Levenshtein distance.
- find the closer word in the terminals using cosine distance.

2 Result Analysis

It is worth noting that because the python implementation is still very slow we couldn't run the parser multiple times so we tried to implement it in c++ the following link gives acces to out github repo `https://github.com/Bronzekorean/CYK_Parser_cpp`.

The first thing noticable is that the tagging precision is not optimal for two reasons:

- First, The tags from the test_file have changed because we are using are different because we deleted the hyphens as suggested in the assignment. For example, NP-SUJ is read as NP.
- Second is that the NT to NT rules are not seen in the output as they are deleted when transforming to Chomsky normal form. For example, ((SENT (NP (NPP Gutenberg)))) is read as ((SENT Gutenberg)).

The oov handler is sometimes good and sometimes completely misses the point because the logic we are using have a lot of weak points, here are some of its replacements: maire to député, indifférence to inquiétude, implication to intervention and interdiction to impossibilité.

Also the CYK algorithm could not parse 22 sentence from 319 because they are not solutions to the word problem. We think this is acceptable as we have trained on a corpus of 2479 sentences. To improve we can first use a bigger tree bank, we can also use a spelling checker and also try to solve the context problem using parents labeling as suggested in the reading material of Jurafsky and Martin's chapter on Syntactic Parsing.

The evalb result is saved in the result.txt file.The summary is as follows:

```
-- All --                                      -- len<=40 --
Number of sentence        =   310     Number of sentence        =   267
Number of Error sentence  =     0     Number of Error sentence  =     0
Number of Skip  sentence  =    25     Number of Skip  sentence  =    21
Number of Valid sentence  =   285     Number of Valid sentence  =   246
Bracketing Recall         =  44.54    Bracketing Recall         =  49.27
Bracketing Precision      =  56.25    Bracketing Precision      =  62.51
Bracketing FMeasure       =  49.71    Bracketing FMeasure       =  55.11
Complete match            =   3.51    Complete match            =   4.07
Average crossing          =   3.45    Average crossing          =   2.15
No crossing               =  38.60    No crossing               =  44.72
2 or less crossing        =  57.89    2 or less crossing        =  67.07
Tagging accuracy          =  76.03    Tagging accuracy          =  75.44
```