TD2: Part of speech tagging and parsing. Rémy SUN.

1 Implementing an NLP parser with python3

We learned a grammar from a treebank, then constructed a parser using a modified probabilized version of the CYK algorithm on a CNF grammar. The CNF tree is then printed in bracket form. Finally, we added a module to deal with words that did not appear at training time. Because execution times were very long, the output file `evaluation_data.parser_output` was not directly generated from standard input streaming lines but a file to make a corpus and *parallelize* computations over multiple cores using the `multiprocessing` library.

1.1 Extracting a Probabilistic Context Free Grammar from a Treebank

We used `nltk` to parse the annotated trees provided in the `SEQUOIA treebank v6.0` into `Tree` objects that were easier to manipulate. From the parsed tree structure, we *manually* filled a *dictionary* with frequencies of grammar rules. More precisely, we represented the grammar as **two nested dictionaries** where the first dictionary maps the *left side* of grammar rules to a second dictionary, this time indexed by the *right hand side* of a particular rule, and returning the number of times the rule "left side" $\rightarrow$ "right hand side " has appeared. Therefore, **access time is constant** and **iterating through possible right hand sides is efficient**. Tree leaves were separately indexed in a probabilistic lexicon (where Part-of-speech tags are mapped to words by a *dictionary*). In accordance with the indications, an inverse mapping of POS tags with respect to words was also kept to form a *lexicon.* Those count dictionaries were **cleaned** (eliminating useless entries) and, most importantly, **normalized** so that we do have probability distributions. It was *indicated* to consider a context free grammar with part of speech tags as terminals, adding a unitary rule from POS to words. This *became useful* when evaluating parsing accuracy from POS tagging as conversion to CNF might otherwise "eliminate" POS tags from rules.

The *CYK* algorithm we used to find the likeliest parse tree for sentences only works with Chomsky Normal forms. This was implemented explicitly *without*the use of any external NLP library. Standard procedure to go from a context free grammar to Chomsky Normal Forms is to apply five routines: **START** (Create a new start symbol S_0), **TERM** (Replace non isolated terminals x in right hand sides by new terminal NT parents), **UNIT** (iteratively eliminate rules $A \rightarrow B$ by replacing B by A in left hand sides, distributing the probability mass), **DEL** (Eliminate null productions) and **BIN** (Add new non terminals to binarize rules). In fact, we only have worry about three out of the five procedures in our case: **TERM**, **UNIT** and **BIN**: no empty word and the start symbol is only in left rule sides.

The **order** of transformation is important: some do not preserve the properties of the others ! Moreover, depending on the application order, **the new grammar can be much bigger than the original grammar** (ranges from quadratic ratio to cubical or worse). In the end, we applied the sequence **TERM**,**BIN** and **UNIT** which leads to a *quadratic* blowup of the grammar size.

1.2 Parsing a context tree

The variant of the CYK algorithm used to parse a sentence can be summed up as, for every connected substring of the sentence, and for every non terminal symbol consider the likelihood of a parse tree rooted in the non-terminal deriving this substring. This is initialized on substrings of size 1 (for which we derive Part of speech probability using the probabilistic lexicon) using unitary rules then using binary rules to combine possible derivations on substrings of increasing sizes, *keeping track of the most probable derivation.* When we know likeliest parse tree rooted in the **starting** variable over the whole sentence, we can recursively backup. While this is done through *dynamic programming*, it is still very expensive to compute ($O(|sentence|^3|grammar|)$). To alleviate some of the complexity, the grammar was separated into *unary* and *binary*. To avoid *numerical underflow* issues multiplying probabilities, we looked at log likelihoods.

1.3 Dealing with Out Of Vocabulary forms

We designed an approach that matches each unknown words w to *one* part of speech tag p (as per the instructions), scanning new input for unknowns at the start (to gain time). We then add the rule $p \rightarrow w$ with probability 1 to the probabilistic lexicon. In fact, **regardless of the probability**, the parse trees derived by CYK **would be the same**: there is only one possible POS tag for the word.

We match words to tags through a mixture of spelling similarity and embedding similarity to known words. The idea is that similar words have similar tags. Spelling similarity is a function of the levenshtein distance d_{spell}, $e^{-d_{spell}}$. But since computations are quadratic, we compute a k-bounded version of the distance (k=3) which is faster as more distant words are very different anyway. The embedding similarity is determined as the cosine similarity s_{emb} to polyglot word embedding when it exists. This leads to similarity score $s = s_{emb} + \lambda e^{-d_{spell}}$, set to 0 if there is no similarity. In the end, for each tag, the score is sum weighted by the computed similarity of the probability of each tag corresponding to the reference words. The tag that scores "the highest" in the end is chosen. The influence of λ was studied on the dev set, but did not seem too important (the final accuracy always hovers around 92% accuracy). In the end we fixed $\lambda = 0.5$ as we thought spelling error as a bit less important as embedding error.

When no decision could be made (as there were no close words spelling wise) and the word was not embedded in polyglot, we *defaulted* to recognizing the word as a *proper noun.*

2 Discussion

In the end, for the chosen parameter, the part of speech tagging accuracy on the dev set was 92% and 89% on the test set. Excluding Out of Vocabulary words, we see a significant increase to 96% and 95%. For the record, Accuracy on subsets of the training set hovered between 98% and 99%. Interestingly, the vocabulary seems significantly different at the end and beginning of the treebank. As such, experiments with shuffling before splitting led to less OOV forms and slightly better accuracy.

The biggest issue is Out of Vocabulary forms, but not to the point of causing too much of a drift in the accuracy of known words as can be seen from the accuracy scores. This shows there is a fairly strong **robustness** of the parsing to small, occasional mistakes in the leaves as the tree needs to fit the sentence "as a whole". For instance, "'Il demande toutefois le renvoi devant le tribunal correctionnel de la majorité de la cinquantaine de personnes mises en examen .' " is mostly correctly parsed even though "cinquantaine" is wrongly classified as an adjective instead of a common noun.

Therefore, the Out of Vocabulary module is probably where the most improvement can be made. One possibility would be to use smoothing when constructing the lexicon and grammar so that every tag/word combination is possible and considered by cyk. Another would be to extend the bound k on the maximum spelling distance to include very distant words which might be marginally better but might also backfire. The accuracy seemed to marginally better with a smaller λ, so we could put more emphasis on embedding similarity and try to find bigger embedding spaces so that *every* OOV form can be compared through their embedding.

That is not to say however that we had no successful handling of OOV forms. For instance "susciter" was correctly recognized as an infinitive verb. On the other hand, "En_ce_sens" was recognized as a proper noun (by default) and "cinquantaine" as and adjective.

The other big issue is running time. This was mitigated through multiprocessing. A number of rules could be eliminated from the CYK grammar, which would lead to faster CYK run times (although this would be a linear improvement, the slope would be very high for long sentences as it is cubical in that length).