NLP - TD 2

ACHER Clément

18/03/2019

1 Implementation

The goal of this homework is to implement a probabilistic parser for French that is based on the CYK algorithm and a PCFG model. The parser should be robust to unknown words.

1.1 PCFG

The first step to tackle this problem is to build a PCFG. In other words, we need to code :
— A parser to collect, count and clean all the rules from a treebank
— A way to normalize the collected rules into the Chomsky Normal Form

My first take was to start from scratch and code all this by myself. I went pretty far until I gave up because of the removal of the unary rules during CNF conversion. Parsing the tree can be done pretty efficiently using simple data structures in a linear time (a stack for instance, `push` until you encountering a closing bracket, then `pop` until encountering an opening bracket. Everything in between is a rule) or using regex. However things get tricky when normalisation comes.

At that point, I had rules stored with their associated probabilities of appearance. A grammar can be converted to CNF in 4 step [1]. In this case, only 2 steps are necessary : the BIN step (eliminate right-hand sides with more than 2 nonterminals) and the UNIT step (eliminate unit rules). As we are dealing with a PCFG, we also need to think about the probabilities we will give to the artificial rules crafted in the process. For the BIN step, this is pretty straightforward as, except for the first rule, the added rules should have a probability of 1. UNIT step is way more tough. Unary rules are collapsed, and probabilities should be multiplied. However, I had at this step different data structure for the terminal rules and the non-terminal one, things get messy as there are for instance some trees like $SENT \rightarrow NP \rightarrow NPP \rightarrow Gutenberg$. Dealing with this is pretty tough, and should probably be done directly while the rules are still in a tree, as `nltk` does. denormalisation would have been quite challenging too.

I then decided to start over and use `nltk` to the parsing and normalisation. The removing of the functional tags is done by traversing the tree and with some simple regex.

1.2 Probabilistic CYK algorithm

I have mostly reimplemented the algorithm described in Jurafsky and Martin's chapter on Syntactic Parsing. I have added a couple tricks :
— As the final probability can be extremely low, we are very likely to reach the limit of 32 bits to code floats. The log of the probabilities are used instead to avoid to small values.
— Use adapted data structure for faster lookup when looking for a rule $A \rightarrow BC$ knowing B and C

While the algorithm seems to work, it is terribly slow. The complexity of the CYK algorithm is $\mathcal{O}(n^3 \cdot |G|)$, where n is the length of the parsed string and $|G|$ is the size of the CNF. Using better data structures has made the algorithm a couple times faster, but it is still very long to parse a bunch of strings.

1. `https://en.wikipedia.org/wiki/Chomsky_normal_form`

1.3 Handling out-of-vocabulary words

As it is very likely that we will encounter unknown words or typos during inference time, this case must somehow be handled. The following strategy is used :

1. If the word is in the vocabulary, i.e. was seen in the training set we simply return the word.
2. Check if the word or a slightly modified version of it (`title`, `upper`, `lower`...) is in the polyglot vocabulary. If it turns out to be the case, the word should not have a typo in it. We then use the embeddings to find the closest word in the available word in the vocabulary. The POS tags for this proxy word will be used.
3. If the word was not found in any of the available vocabularies, we look for close words using the editing distance. If we find a word close enough, we use the POS tags of this word.
4. If none of the above was successful, default tokens are used. There are two of them : `<NPP>` and `<UNK>`. If the first letter of the word is a capital letter, it is likely that it is a *nom propre* as it was not found in the other vocabularies. We then return the token `<NPP>` that has the POS corresponding to *nom propre* with the frequencies seen in the dataset. The `<UNK>` is about the same, if nothing has worked, we give the `<UNK>` token that has most of the POS associated with it with their probability of appearance in the train dataset.

2 Error analysis

Before mentioning the overall performances of the parser, we can discuss a bit about the OOV handling strategy as a failure at this step will introduce errors in the final results.

I have written an extra module, `oov_evaluation.py` to evaluate this strategy. It takes the words from a dataset different from the training set and checks if the ground truth POS tag is in the list of potential POS tags that are given by the lexicon and the OOV strategy for this word. Using the dev test, this is the case for 98% of the words. There are however three main kind of errors there :

— Words that have different possible POS tags for which only a part of it has been seen in the training dataset. For instance, the word *ancien* can be both an adjective or a noun. In the training dataset, it is only labeled as an adjective, but is a noun in the dev dataset. Some other examples : *fournisseur* (only noun in training set), *personnel* (same)...
— Numbers sometime. This could be fixed
— The strategy for *nom propre* is not perfect. Maybe checking if the word is the first word of the sentence could help.

Regarding the parsing itself, it works pretty well on the training set. As expected, all sentences get parsed and the accuracy for POS tag is about 98%. However, about 20% of the sentences are not successfully parsed in the other datasets. I can see two different explanations : either the rules forming the sentence are not in training set, or a wrong inference by the lexicon at the word level (out of vocabulary word) has made the parsing impossible to finish. In the first case, I don't think that much can be done about it. For the parsed sentence, the tagging accuracy is 91%.

2.1 Possible improvements

Here's a couple things that could be done :

— Gather all the numbers together into a single word in the lexicon
— Get more rules for the OOV part (even tough a miss can lead to a sentence not parsed)
— A larger dataset could be quite nice, but this is pretty hard to gather a treebank like this...