Dependency Parsing exercises: Implementation of a transition-based parser (part 3)

Deadline: 21.06.2021

Please send completed solutions to evang@hhu.de with subject "dependency homework" and attachment `projectivize.py`, as well as your parser output.

In this exercise, we will apply our parser to a new treebank from the Universal Dependencies treebank collection. Because these treebanks are in general non-projective and our parser unfortunately does not support non-projective trees, we have to projectivize them first (cf. Lecture 6). We will use the projectivization algorithm of Nivre and Nilsson (2005), but not attempt to reconstruct non-projectivities in the parser output.

1. Download the starter code in `projectivize.py`. Implement Nivre and Nilsson's projectivization algorithm by filling in the blanks (where it says `raise NotImplementedError()`). Check the correctness of your implementation by running it on the provided `example-nonprojective.conllu` and making sure the output is identical to the provided `example-projective.conllu`.

   - Hint: make sure to translate correctly between the CoNLL format's 1-based indexing and Python's 0-based indexing. For example, to look up the head of word 7, use `heads[6]`. More generally, to look up the head of word i, use `heads[i - 1]`.

2. Download a Universal Dependencies treebank of your choice that is not in English. It should not use multiword tokens[1] and it should be divided into a train, dev, and test set. To do so, go to `https://universaldependencies.org`. Under "Current UD Languages", click on a language, then click on one of the treebanks for that language. Click on "master" to open the treebank on GitHub. There should be `.conllu` train, dev, and test files. If there is only a test file, select another treebank. Download the train and dev (validation) files.

3. Projectivize the downloaded files. If you get an assertion error from the `read_conll` function, the treebank probably uses multiword tokens. Select a different treebank in this case.

4. Adapt the parser to the Universal Dependencies set of labels. (If you did the optional sub-exercise 4b last week, you should not have to do anything here.) (You can also use the example solution for exercise 9 as a starting point.)

5. Train the parser for at least one iteration and observe its accuracy on the development data. Submit the output of the parser (where it reports UAS and LAS) as a text file.

6. Run the parser in parse mode, on an example sentence, either from the development data or any sentence that you choose. Submit the output in CoNLL format.

References

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics. https://www.aclweb.org/anthology/P05-1013.

[1] Cf. https://universaldependencies.org/format.html