NLU 2021 First assignment - Report

Lucie Nass - 219524

Task 1: Extract a path of dependency relations from the ROOT to a token

As stated in the documentation, it is far easier to find the relation from the lowest token than from the top, as each token can have multiple children but only one “parent” (head).

Knowing this, to get the path from any token to the root, I use the `head` property. The head and dependency label of the token is added to a list, then the head and dependency label of that head, etc. until the root is reached. The list is then reversed so it is in the asked order, and stored in a dictionary where the key is the original token.

Task 2: Extract subtree of a dependent given a token

This function uses the `Token.subtree` attribute on each token of the sentence, storing the subtrees in a dictionary, with the key being the token.

Task 3: Check if a given list of tokens (segment of a sentence) forms a subtree

In this function, `True` is returned if there is a subtree in the sentence that:

- contains every word given in the list once and only once
- doesn’t contain any other word

The order in which the words are given does not matter here, so giving the lists [“the”, “man”] and [“man”, “the”] will return the same value.

To avoid uselessly going over all the subtrees, for each token, the function first tests if the token is in the list of words given, and if the number of tokens in the subtree is equal to the number of words in the list. If any of these conditions is false, then we go to the next token (as it means that either the subtree does not contain all the words of the list or contains at least one other word).

If the token and its subtree satisfy these conditions, we look at each element of the subtree and make sure it both is part of the list of words, and hasn’t been seen in this subtree already. If any element doesn’t satisfy these conditions, the current token and its subtree are abandoned, and we go on to the next token and subtree.

Task 4: Identify head of a span, given its tokens

This function takes as input a list of strings. After collapsing the list into a string (adding a space between each word), it is turned into a `Doc`. The `Span` type in spaCy allows us to very simply retrieve the head of a span by using the attribute `root`. To be able to use it, the entire `Doc` is turned into a `Span` using the `doc[:]` operator.

Task 5: Extract sentence subject, direct object and indirect object spans

This function returns a dictionary containing three lists: the subject(s), the direct object(s), and the indirect object(s) of the sentence. For each token in the sentence, the dependency relation is checked. If its value indicates a subject, direct object, or indirect object, the span corresponding to its subtree is appended to the right list. The inferior limit of the span is given by subtracting the `n_lefts` attribute value from the index of the token, while the superior limit is given by adding the `n_rights` attribute value plus one to the index of the token.

The dependency labels that are taken into account for each category are:

- Subject: clausal and nominal subjects (CSUBJ and NSUBJ), including passive ones (CSUBJPASS and NSUBJPASS)
- Direct object: DOBJ
- Indirect object: although there is isn't an IOBJ dependency label, the DATIVE dependency label seems to be the closest, so this is what I used for indirect objects