First NLU assignment

Giorgio Scarton 225606

April 18, 2021

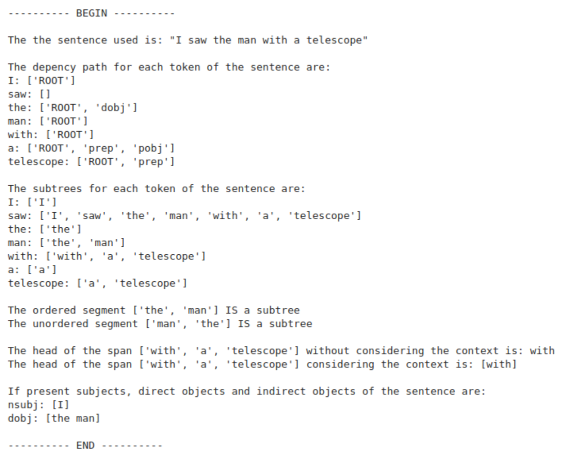
Functions

- `extract_path(sentence)`
  - Input: a sentence as string
  - Output: a dictionary of the form *token:[path]*
  - Description: executes nlp on the sentence and for each of the extracted tokens get the attribute *dep_* of its ancestors and store them in a list, then it reverts the list to get the path from ROOT to the token
- `extract_subtree(sentence)`
  - Input: a sentence as string
  - Output: a dictionary of the form *token:[subtree]*
  - Description: executes nlp on the sentence and for each of the extracted tokens get the attribute *text* of its subtree and store them in a list
- `check_if_subtree_ordered(sentence,segment)`
  - Input: a sentence in form of a string and a segment in form of a list of strings
  - Output: True if the segment is a subtree else False
  - Description: calls `extract_subtree(sentence)` and for each item of the result it checks if it is equal to the passed segment returning the corresponding boolean value
- `check_if_subtree_unordered(sentence,segment)`
  - Input & output: same of `check_if_subtree_ordered(sentence,segment)`
  - Description: alphabetically sorts the passed segment then calls `extract_subtree(sentence)` and for each item of the result it sorts it and checks if it is equal to the sorted segment returning the corresponding boolean value

- `span_head(list)`
  - Input: a list of string forming a span
  - Output: the root of the span token
  - Description: joins the elements of the list into a string, processes it using nlp, convert the doc to span and gets the ROOT property
- `span_head_with_context(list,context)`
  - Input: a list of string forming a span and a sentence as string
  - Output: the root of the span token in a list
  - Description: using SpaCy *PhraseMatcher* finds the span into the sentence and gets it's root, in the case it is not part of the sentence calls `span_head(list)` and returns its output
- `subj_dobj_iobj(sentence)`
  - Input: a sentence as string
  - Output: a dict of the form *dependency:[token]*
  - Description: using SpaCy *Matcher* finds the span with the right dependency, gets its *noun_chunk* and stores it in a dictionary

Test script

The test script takes as input a sentence as string, a segment as list of strings and a span as list of string, then using that inputs executes all the functions and prints their output in a pretty way. To do that a `print_dict(dictionary)` function has been created.



```
---------- BEGIN ----------

The the sentence used is: "I saw the man with a telescope"

The depency path for each token of the sentence are:
I: ['ROOT']
saw: []
the: ['ROOT', 'dobj']
man: ['ROOT']
with: ['ROOT']
a: ['ROOT', 'prep', 'pobj']
telescope: ['ROOT', 'prep']

The subtrees for each token of the sentence are:
I: ['I']
saw: ['I', 'saw', 'the', 'man', 'with', 'a', 'telescope']
the: ['the']
man: ['the', 'man']
with: ['with', 'a', 'telescope']
a: ['a']
telescope: ['a', 'telescope']

The ordered segment ['the', 'man'] IS a subtree
The unordered segment ['man', 'the'] IS a subtree

The head of the span ['with', 'a', 'telescope'] without considering the context is: with
The head of the span ['with', 'a', 'telescope'] considering the context is: [with]

If present subjects, direct objects and indirect objects of the sentence are:
nsubj: [I]
dobj: [the man]

---------- END ----------
```

Figure 1: Output sample