ASSIGNEMENT

1

WORKING WITH DEPENDENCY GRAPHS

.PY

FRANCESCO MINCHIO

NLU PROJECT

2021

PROJECT’S PURPOSE

The objective of the assignment is to learn how to work with dependency graphs by defining functions.

OPERATIONS

Define functions to:

1. Extract a path of dependency relations from the ROOT to a token
2. Extract subtree of a dependents given a token
3. Check if a given list of tokens (segment of a sentence) forms a subtree
4. Dentify head of a span, given its tokens
5. Extract sentence subject, direct object and indirect object spans

EX_1

FUNCTION
getDependencyPath

INPUT
input_sentence (string)

OUTPUT
List of lists, each from the ROOT to the token. Each item of these lists is a dependency relation between two tokens.

You need to process the phrase to get a “doc” object by spacy_nlp(sentence).
In this way a container (dictionary) is created to store all dependent relationships on the single ROOT for each token. These relationships are extracted from ROOT by exploiting the properties of the token ancestors.
It is clear from this that it wil result in a list of relationships which will be inverted by recting the first element of the token path from the ROOT.
The last one will be the dependency relationship that connects that particular token.

RESULTS

```
Printing EX1 results...

i : [‘ROOT : saw’, ‘nsubj : i’]
saw : [‘ROOT : saw’]
the : [‘ROOT : saw’, ‘dobj : man’, ‘det : the’]
man : [‘ROOT : saw’, ‘dobj : man’]
with : [‘ROOT : saw’, ‘dobj : man’, ‘prep : with’]
a : [‘ROOT : saw’, ‘dobj : man’, ‘prep : with’, ‘pobj : telescope’, ‘det : a’]
telescope : [‘ROOT : saw’, ‘dobj : man’, ‘prep : with’, ‘pobj : telescope’]
```

EX_2

FUNCTION
getDependencySubtree

INPUT
input_sentence (string)

OUTPUT
List of lists. Each list at index “i”contains the subtree of token “i” in the sentence.

A dependency tree is a grammatrical structure added to a sentence which delineates the dependency between a word and the phrases it builds upon (verb-phrase relation).
The function used analyzes a sentence by extracting the subtrees of each token by using the subtree property of the token object (transformation into string).

RESULTS

```
Printing EX2 results...

[“I -> [‘I’]”, “saw -> [‘I’, ‘saw’, ‘the’, ‘man’, ‘with’, ‘a’, ‘telescope’]”, “the -> [‘the’]”, “man
-> [‘the’, ‘man’, ‘with’, ‘a’, ‘telescope’]”, “with -> [‘with’, ‘a’, ‘telescope’]”, “a -> [‘a’]”, “te-
lescope -> [‘a’, ‘telescope’]”]
```

EX_3

FUNCTION
is_subtree

INPUT
input_sentence (string)
subtree (list of strings)
string = token

OUTPUT
True o False.

This function is used to find for a match between the list of subtrees and the subtrees of each token in the sentence. In this way it is understood if the input subtree is a valid subtree present in the analyzed sentence, as a kind of verification.
In this case the result will be true or false depending on the correspondence.

RESULTS

```
Printing EX3 results...

Testing: ['a', 'telescope']
Found subtree: ['a', 'telescope']
True
Testing: ['saw', 'the']
Not a subtree.
False
Testing: ['the', 'man']
Not a subtree.
False
```

EX_4

FUNCTION
HeadOfSpan

INPUT
Single span (string).

OUTPUT
The token object which rapresent the head of the span.

Span is a portion of a doc object formed by one or more tokens (sequence of token objects).
In order to use the property you need to “convert” the input phrase into a span (span = doc[start : end]).
The cells below show the function code described and its application.

RESULTS

```
Printing EX4 results...

span:  I saw the man with a telescope
head of span:  saw
```

EX_5

FUNCTION: get_spans

INPUT
input_sentence (string).

OUTPUT
Dict of lists.
Dict is composed of 3 keys “nsubj” , “dobj” , “iobj” with one list each.
The items of the lists are the extracted corresponding spans (strings).

We check if the selected dependency matches the one of interest.
Dependency:

- ‘nsubj’ for the subject
- ‘dobj’ for the direct object
- ‘dative’ for the indirect object

Once the match is found, the function creates a subtree Span object with the token as root (token.left_edge.i and token.right_edge.i), respectively the first and last index of the subtree token (token list). Finally, a tuple is added to the dictionary as a key and the list of the span as a value.

RESULTS:

```
Printing EX5 results...

{(I, ‘nsubj’): [I], (man, ‘dobj’): [the, man, with, a, telescope]}
```

Production

Colab
GitHub
Adobe InDesign

May 2021