**Assignment NLU course**

For this first assignment we were asked to solve 5 points regarding the usage of the spaCy library

* To extract a path of dependency relations from the ROOT to a token I used the classes *find\_root, get\_head* and *extract\_path.*
  + **get\_head([], token, root):** this function is recursive, and it follows the dependency chain up until the root. So, given a list, the start token, and the root token, it recursively builds the dependency chain, and in this way the complete list with the dependency is created and returned.
  + **extract\_path(sentence):** is a function that, given in input a sentence, returns a subtree of dependants for each token, more precisely a list of lists each containing the token and the dependency relations. First the sentence is parsed, after that the root of the sentence is extracted by using the function root, provided by spaCy, on the doc defined as a span. Having the root, now for each token the function get\_head is called. The output of the last function then is reversed to obtain a list of dependencies going from the root to the token and finally appended to the list of lists.
* The second request is about extracting a subtree of dependents given a token. To do so I defined the class *extract\_deptree*.
  + **extract\_deptree(sentence):** it is a function that, given a sentence as input, it returns a list of lists, containing the subtree of dependents for each token composing the sentence, where each instance is composed of the token and its list of descendents (token included). The input sentence is first parsed, and after that, for each token composing the sentence, we extract the subtree of dependents using the spaCy function *token.subtree.* Finally, we append each token composing the subtree to the output list and return the result.
* To check if a given list of tokens forms a subtree, I defined the function *check\_valid\_subtree.*
  + **check\_valid\_subtree(test\_subtree, sentence):** function that takes in input a list of words (in my case I tested the lists [‘hill’, ‘on’] and [‘a’, ‘telescope’, ‘with’]) and a sentence, and returns a Boolean value describing if the provided list of words corresponds to a valid tree. To check this, first the list of subtrees is defined by using the function extract\_deptree used in point 2. For each subtree defined in the list of lists, first the length of the test and actual subtree is checked. It is checked if all the items composing the test subtree are present in the actual subtree. If that is the case, then it means that the test subtree is an actual valid subtree. The Boolean value is finally return based on the validity of the subtree.
* To identify the head of a span given its tokens I defined the function *spanhead*.
  + **spanhead(sentence, start\_point, end\_point):** it is a function that, given a sentence and a starting and ending points as inputs, it returns the head of the span created from the sentence and the two points provided (so I defined the span required directly from the sentence used as a test for the whole assignment).To find the head of the span the function simply relies on the function span.root provided by spaCy, which returns the token with the shortest path to the root. The function finally returns the head token of the span.
* To extract sentence subject, direct object, and indirect object as spans of the sentence I defined the function *extract\_sdi.*
  + **extract\_sdi(sentence):** it is a function that, given a sentence, extracts the span of subject, direct and indirect object of that sentence. First, the sentence is parsed, and the root is extracted from the document. Then, for each token, its dependency label is checked to see if it is part of the three categories mentioned above. To be part of the subject the token must have a dependency label equals to ‘nsubj’ for nominal subject, ‘nsubjpass’ for passive nominal subject, ‘csubj’ for clausal subjects, ‘csubjpass’ for passive clausal objects and ‘expl’ for the expletive subject (when there is no subject ‘there’ is taken as subject). For the direct object then it is used the tag ‘dobj’ and for the indirect object the tag ‘dative’. Then, if a token has one of these labels, first its subtree is extracted, then we obtain from the document the span that starts from the index of the first token of the subtree and ends in the position of the last token’s index in the subtree +1. The span is then appended to the dictionary and this is finally returned.