ASSIGNEMENT 2 NLU

Francesco Ferrini

April 30, 2021

Exercise 1

1) Evaluate spaCy NER on CoNLL 2003 data (provided): Report token-level performance (per class and total)
The first function used for this exercise is **reconstructSentences**. It takes as input the CoNLL file and recreates the sentences using the read_corpus_conll function from conll.py. At the end reconstructSentences returns a list with all the sentences.
Then the sentences_list is passed to the function **retokenizationOfSentences** which has the goal of retokenize the sentences using spacy and assign an IOB tag to each token. The retokenization part is done using retokenize() from spacy. It retokenizes the tokens that have no whitespace. After this operation for each doc the function creates 2 lists, the first one is called iob_tag_list which contains all the iob tags converted into conll format (and mantaining the ent_iob in order to distinguish as two different classes I_ORG and B_ORG) and the second is ent_list that contains lists (one for each doc) of tuples (each containing the token.text and token iob tag). The second list will be used in the ex 1.2.
Then the function groundTruthList is called. It takes as input a corpus file and gives in output 2 lists that corresponds to the ground truth lists. The first one (gt_list1) returns a list of iob tags taken from the corpus file from CoNLL; the second one (gt_list2) instead is a list of tuples that contains the first element of each row in the file (word) and, as second element, the iob tag. Then in order to compute accuracy for class, in the main is called the classification report function from sklearn.

2) Report CoNLL chunk-level performance (per class and total): precision, recall, f-measure of correctly recognizing all the named entities in a chunk per class and total
In the exercise 1.2 in order to evaluate the chunk level performances, is used the evaluate function from conll.py. In this case to the function are passed the second lists returned from the functions in the previous point (ent_list and gt_list2).

Exercise 2

Grouping of Entities. Write a function to group recognized named entities using noun_chunks method of spaCy. Analyze the groups in terms of most frequent combinations (i.e. NER types that go together)
In this exercise is asked to group together the IOB tags of tokens that belongs to the same chunk. Firstly again I used the **reconstructSentences** function to reconstruct the sentences

from the corpus file.
Then in a for loop the sentences are passed to the **entityGroup** function. This function parses the sentence getting a Doc object of Spacy. Then I created a list of tuples (where each tuple is formed by the text of the entity and the corresponding label) and dictionary of chunks where the keys are the chunk.text in noun_chunk. After this I iterated over the first list of the tuples and checked if the first element of the tuple (entity text) is in one of the chunk using the function **isInChunk** that returns the chunk if the entity belongs to one of them, otherwise an empty string. In the second case I added to the chunk dictionary a new key that is the entity text. Then all the entity labels are added in the list corresponding to the appropriate chunk.
Finally with the **mostFrequent** function I created a dictionary putting only the groups of entity labels that correspond to the same chunk. Then I ordered the dictionary in order to print the most frequents. For counting them I decided that if the function finds the groups ['ORG', 'GPE'] and ['GPE', 'ORG] then it considers them as different groups since also the position is important.

Exercise 3

One of the possible post-processing steps is to fix segmentation errors. Write a function that extends the entity span to cover the full noun-compounds. Make use of compound dependency relation.
In this exercise the idea is that if a token is in a compound relation and the head of this token is not labeled with 'O' (so it means that is not out of a chunk), the token takes the same iob entity tag of his head.
For this purpose I created the **fixSegmentation** function that takes as input a sentence and parses it returning a doc object. Then I created 4 lists, one for the iobs, one for the entity types, one for the entity texts and one for the tokens. Then I iterated over the tokens in the doc and the function finds one that has a compound relation and his head has a not empty ent_type_ then I changed the type of the token with the same of the head. To decide which iob to give to the token I controlled if it was before or after the head. In the first case, the ent_iob_ will be B, otherwise I.