Second Assignment Natural Language Understanding

Bertelli Davide
mat. 223718

1 Abstract

The following report illustrates the choices and intentions behind the second assignment provided during the course of Natural Language Understanding. The requirement were to implement custom functions and perform tasks regarding the **spacy** python module. Specifically:

1. Evaluate *spacy* Name Entity Recognizer on conll2003 english dataset.
   (a) Accuracy of model.
   (b) Precision, Recall, F-measure for categories and in total.
2. Create a function to group named entities and report their frequencies.
3. Create a post-processing function which extends the entity spans to cover full-noun compounds.

2 Code requirements

In order for the code to have all the necessary modules to work, **spacy** and its english pipeline *'en_core_web_sm'*, **pandas** and **sklearn** are needed. Moreover, the conll.py file and even the conll 2003 english dataset are required. Further informations about installation of packages and download of resources are reported into the README file containted into the same repository of this document.

3 Code Implementation

3.1 Extra functions

In the code contained inside the *2nd_Assignment.py* have been added some extra functions to reduce the amount of code written or transform the data into a more manageable shape. These extra functions are:

3.1.1 loadConll

Given in input the name of the dataset in conll format, searches it into the default directory *../data/conll2003/* and loads it. In the loading phase, after removing the '-DOCSTART-' lines, it initiates a dictionary having a key for each class of data contained in the conll dataset. Specifically, the keys are *text*, *POS tag*, *Syntactic Chunk tag* and the *Name Entity tag*. After being populated the dictionary is returned to the caller.

3.1.2 converter

Is a function that is used to translate the spacy named entities into conll ones. This direction of conversion has been chosen for minimizing the error of translation. The rules of translation are:

Spacy tag	Conll tag
PERSON	PER
GPE	LOC
ORG	ORG
O	O
others	MISC

3.1.3 computeConllFreqs

Is a function which computes the frequencies of entites groups that spacy creates after processing the conll 2003 dataset through its pipeline. This function exploits 'loadConll' extra function, and a custom function, that will be discussed in section 3.3.

3.2 First Function

The first function aims at evaluating the spacy named entity recogniser exploiting the data contained into the conll dataset. It takes as input the name of the conll dataset to use and an optional parameter allowing the user to reduce the amount of sentences to process. To fulfill its purpose it loads spacy's english pipeline *'en_core_web_sm'* creating the nlp object, then the *conll 2003* dataset is also loaded exploiting the extra function 'loadConll'. After a check, that if satisfied, creates the test Doc items with a reduced dimensionality, an operation of re-tokenization takes place. The re-tokenization is needed to align the conll data with the spacy processed ones, this because spacy in case of dates or words connected by the token '-' reads each element in the connection as a single word, while in the conll dataset they are a unique token. To perform this re-tokenization it has been used a for-cycle iterating among all the Doc items existing into a list created after the processing of the conll dataset. For each element in the list, a retokenizer object is built by calling the *retokenize* method on the Doc item. During the parsing of the tokens inside the Doc object a check is performed on the presence of whitespaces, in which case the merging indexes are set and used to link together the tokens that do not have any whitespace. This kind of choice had been followed due to the fact that the tokens splitted by spacy into multiple words never end with a whitespace, while the conll data does. Then the lists of spacy's predicted tokens and conll's test tokens are created and populated by two for-cycles. Specifically, the first iterates among the tokens inside each Doc items stored in a list and for each tokens it checks whether its entity type is empty or not. If it is, the named entity pushed into the list is just the 'O' term coming from the *IOB* notation, otherwise the IOB notation plus the named entity type are converted by calling the *coverter* function and then pushed into the list. This is done to have agreement between the types of conll dataset and spacy's ones. To fill in the conll named

entities' dataset a for-cycle iterates among the lists associated to the *NE_tag* key in the conll data dictionary and pushes each element of them in a list. Once the lists are full, the accuracy of the spacy model is computed by using them as test and hypothesis examples passed to the *classification_report* class of sklearn module. The computed accuracy of spacy's model, using conll 2003 as corpus is 0.7089.
In a similar way the precision, recall and f-measure are computed. Starting with two emptiy list, one for the reference token and the hypotesis list for spacy's tags. A for-cycle iterates among the lists inside the value indexed at *NE_tag* in conll's data dictionary, and each token inside the sentences in the list is processed by a for-cycle in which it is pushed in a previously empty defined list, which will be used to keep the sentence structure, and then also the corresponding spacy's token is pushed in another empty list. After the cycle, we obtain two lists of lists containing the named entities of the processed tokens. To compute the requested measurements the *evaluate* method in the conll.py module is used and the scores are printed in an user-friedly way thanks to the pandas module. The computed results are as follows:

NE	precision	recall	f-score	support
LOC	0.756	0.665	0708	1668
PER	0.774	0.609	0.681	1617
Misc	0.100	0.547	0.169	702
ORG	0.464	0.276	0.346	1661
total	0.388	0.520	0.444	5684

3.3 Second Function

The second function aims at grouping the entities of a sentence or a Doc item and process the frequencies of the groups of entities composing it. This task is achieved by splitting the function logic in two pieces.
groups_NE() deals with the task of grouping the named entities adding the ones that aren't present in the object returned by *doc.noun_chunks*. To perform its task, after creating the Doc object of the input text, it allocates two lists, one containing the entities given by *doc.ents* method and the other with the entities coming from *doc.noun_chunks*. After this it cycles among the entities inside the latter list and unifies the duplicate entities pushing the result into another list. Then the list coming from the *ents* method is copied into a temp one and whenever its tokens are common to the one having the unified entities, they will be removed. Finally another cycle is performed on the remaining items of the temp list and they are pushed in the right position into the unified entities list which will be returned to the caller ending the execution.
frequencies_comp() is the one in charge to compute the frequencies of the groups found by the previous function. It takes as arguments a list of tuples and other parameters governing the final output. Immediately after being called it creates an empty dictionary to store the groups and their occurencies. Then for each element inside the input list it creates an empty string that will be updated and used as key for the dictionary. The updated key will be the text of each named entity in the group divided by a whitespace. After ensuring that the key is not empty, case in which the list itself is, a new entry in the dictionary is made, if not already present, and its value is setted to 1, otherwise updated adding 1. Finally, after sorting the keys by values in descending order and having checked the output parameter and flag, the results are printed and the sorted dictionary is returned to the caller. It is important to point out that all the entities are labelled keeping spacy's notation and not using conll's one. In case of the conll 2003 dataset, the computation has been performed by the extra function *computeConllFreqs* which returned that the entries in the dictionary were 96. Following are listed the five most frequent groups of entities, with their respective counts.

1. 'CARDINAL PERSON': 57
2. 'NORP PERSON': 43
3. 'GPE PERSON': 34
4. 'ORG PERSON': 22
5. 'CARDINAL ORG':22

3.4 Third Function

The third function aims at fixing segmentation errors in which some tokens having a compound dependency relation are not in the same entity. It takes as input a sentence, that will be converted by the spacy pipeline, or a Doc item. In both cases it cycles among the tokens contained into the Doc object and for each of them checks whether their language dependencies are 'compound' or not. If they are, the head of the token is extracted, and used as a key in a dictionary containing a list of the compound tokens as value. Next a for-cycle on the values of the dictionary pushes the head token in the right position of the list to have it ordered. Then two lists are initiated: one for the grouped entities and one for just the entities. A for-cycle iterates among the keys in the dictionary and for each token in the associated list, checks whether the token is already inside the list of entities, if it isn't it get pushed inside, otherwise it keeps going with a new token. This cycle, among with its checks, allows to follow the head-token compound dependency in order to group toghether the entities in this relation. Next, another for-cycle iterates among the entities returned by *doc.ents* and for each token in them cheks whether corresponds to another token in the dictionary of compound entities. If it does, a flag is set to True, and the indexes of the position of the token in the two lists are saved. After that, whenever the flag is True the list of ents that raised it is removed from the list of doc.ents. Then the left entities are copied in another list and a cycle is performed over the saved indexes in order to insert the grouped entities in the right location inside this new list and removed by their original list. At the end of the process the only entities left in the grouped entities' list are the ones with index higher than the one already present in the doc object, and so they are appended to the list. Finally the elements in this new list are converted into spans of the original doc and the entities of the doc itself are overwritten. After printing the difference between the initial entities of the doc item and the new ones, the function returns the new doc object to the caller.