Second Assignment Natural Language Understanding

Bertelli Davide
mat. 223718

1 Abstract

The following report illustrate 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 correctly recognised named entities.
2. Create a function to group named entities and report their frequencies.
3. Create a post-processing function which extend the entity span 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 which reduce the amount of code written and transforms the data into a more manageable shape. This extra functions are shown below:

3.1.1 loadConll

loadConll(): is a function that, 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 is initiated a dictionary having a key for each class of data contained into any single line of conll dataset. Specifically the *text*, *POS tag*, *Syntactic Chunk tag* and the *Name Entity tag*. After populating this dictionary with the entries of conll, it is returned to the caller.

3.1.2 converter

converter(): is a function that is used to translate the spacy name entities into conll ones. This direction of conversion has been chosen due to the fact that the entities of spacy were easier to transalate into the conll ones than viceversa. 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 exploit the loadConll extra function, and a custom function to compute the occurencies that will be discussed further at section 3.3.

3.2 First Function

The first function aims to evaluate 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 was 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 entity. To perform this re-tokenization it has been used a for-cycle iterating among all the Doc items existing into the list of Doc 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 that spacy splits into multiple words never end with a whitespace, while the conll data does. Then the lists of spacy predicted tokens and conll test tokens are created and filled trough two separates for-cycles. Specifically, the

first iterates among the tokens inside each Doc item in the list of doc items and for each tokens it checks wether 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 the conll named entities' dataset the for-cycle iterates among the lists associated to the *NE_tag* key in the conll data dictionary and pushes each element of it in the 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 a corpus of 3683 entries, is 0.7089.
In a similar way the precision, recall and f-measure are computed. It starts with two emptiy list, one for the reference token, assumed to be true, 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 for each token inside the sentences in the list is processed by a for-cycle in which 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 have 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 are printed in an user-friedly way thanks to the pandas module. The results for the measurements 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 not 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 will be removed. Finally another cycle is done 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 a string with each named entity text 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 set, if not already present, and its value is setted to 1, or updated adding 1 otherwise. Finally after sorting the keys by values in descending order and 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 by spacy and not translated into conll format. 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. As follows are listed the five most frequent groups of entities.

1. 'GPE': 1296
2. 'PERSON': 1071
3. 'CARDINAL': 1018
4. 'ORG': 916
5. 'DATE': 784

Showing how the groups composed by just one entity are the most frequent ones across all dataset processed by spacy.

3.4 Third Function

The third functions aims at fixing segmentation errors in which some tokens into 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 wether their language dependencies are 'compound' or not. If they are, the head of the token is extracted, the entities of the document are stored in a list, and a boolean variable, stating if exists a span in which both the head and the token are, is set to False. Then a for-cycle iterates among the span contained into the list of entities and if it finds a span in which both head and tokens are it sets the boolean variable to True and keeps with the other tokens. If a span containing only the head is found, then the *spacy.tokens.Doc.retokenize* method is used to take a retokenizer and retokenize the span in order to include both the head and the token having the compound relation, also the boolean variable is set to True. Lastly, if no span containg the head of the token is found, then the retokenizer is called to make a span containing just the head and the token. Finally the retokenized Doc object is returned to the caller.