Second NLU assignment

Giorgio Scarton 225606

April 27, 2021

Evaluation of Spacy NER in CoNLL 2003

The first step of the evaluation consist of the reconstruction of the original corpus from CoNLL 2003, to do so has been created a function called `conll_to_string(path)` which takes in input the absolute or relative path of the source file and returns a string made of all tokens from CoNLL separated by a white space (for more details see code comments)

Then SpaCy Tokenizer rules has to be change to adapt to the CoNLL tokens, although this can lead to suboptimal performances, from several tests on CoNLL provided files it came out that editing the tokenizer is more computationally efficient than post-process the tokenized document and performances of the identification are similar.
A *WhitespaceTokenizer* class, which only splits token at spaces, has been defined to replace the built-in tokenizer in `evaluate_NER_on_CoNLL(path)`, in such a way it is not required to post-process tokenization becasuse the input has been built from `conll_to_string(path)`. Anyway another post-processing is required to adapt SpaCy *entity types* to CoNLL ones, the mapping has been made in an arbitrary way considering both SpaCy entity definitions accessible via token property and CoNLL ones[1]. Follows the map:

SpaCy	CoNLL
PERSON	PER
FAC	LOC
GPE	LOC
EVENT	MISC
LAW	MISC
NORP	MISC
LANGUAGE	MISC
WORK_OF_ART	MISC
Others	O

To apply the map to the SpaCy document has been defined a function `remap(doc)` which takes as input the document and returns the same document with updated token's *entity types.*

Token level performance

To evaluate token level performance a modified version of the provided `conlleval(data,otag='O')` function has been exploited. The input to the evaluation function `evaluate_token(refs,hyps)` is a list of list of tuple, to provide such kind of input two functions has been created: `get_list_from_doc(doc)` and `get_list_from_conll(path)`. Both of them return a list of list compatible with `conlleval(data,otag='O')` function given in input a SpaCy document or a CoNLL file for each token they generate the tuple *(text,NER)*, add this tuple to a list and then collects these lists in an outer list. Once the correct input format il available it's passed to `evaluate_token(refs,hyps)` which fist uses the provided

[1]Introduction to the CoNLL-2003 Shared Task:Language-Independent Named Entity Recognition - Erik F. Tjong Kim Sang, Fien De Meulder - 2003

`align_hyp(ref, hyp)` function to align hyps and refs then, iterating over each token, if tag is not None generate classes if they don't already exist and check correspondence between hypothesis and references. In case of None tag it is replaced by O and the process goes on as in the previous case. Accuracy is calculated as:

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} = \frac{Number\,of\,correct\,assessment}{Number\,of\,all\,assessment}$$

Chunk level performance

For the chunk level performance the `conlleval` function provided in *conll.py* has been exploited, the process is like the one described for the token level one except for the parameters extrapolated which comes from *conll.py*'s `score(cor_cnt,hyp_cnt,ref_cnt)`.

Grouping of entities

Group entities

The function `group_entities(text)` takes as input a text as string and returns a tuple of grouped entities and frequencies of each chunk. To do so it iterates over tokens in the document checking if they are part of a chunk, if so a sublist of the entities in the chunk is generated and added to the output list changing the current index to $index + lenght(chuck)$ so that no more checks on the elements ore the chunk are required, in the opposite case the token label is directly added to the ouput list and the index is incremented by 1.

Compute frequencies

To compute frequencies the function `compute_frequency(lista)` takes as input the previously generated list and for each sublist creates a key in a dictionary which has as value another dictionary with relative and absolute frequencies of each chunk. The order of the elements in the sublist is taken into account because a different order may lead to a different structure of the sentence and consequently to a different interpretation of the sentence itself. The absolute frequency is the number of instances of the tuple in the text, the relative one is given by this number divided by the total number of sublist in the input list which corresponds to the total number of chunks.

Fixing segmentation errors

The function `fix_segmentation(text)` takes as input a text and fixes chunk segmentation errors if present, the segmentation is due to entities that are detached from the span to which they refer to and so that are not recognized as part of the chunk. To fix that the function checks if there are token with *compound* dependency label and, if the head of this token is labeled, recompute the chunk setting the *entity type* of the token equal to the one of his head and changing the *entity IOB* according to the position of the token with respect to his head, this is done because otherwise ti would not be possible to recognize the token as part of the chuck to which ti belongs to.