Assignment 2

Luigi Riz - Mat. 223823

To complete the second assignment, I decided to work on a Jupyter Notebook, that helped me divide my code into chunks and keep track of the work already done, through the markdown blocks it provides. The work is subdivided into three main parts, representing the three different tasks we were asked to solve. Each of them is divided into several little chunks, that build up the solution step by step. The whole work is about *CoNLL 2003* database and the performances reached by the *spaCy*'s English model, while processing it.

The main libraries used are **spaCy**, to process the corpus, to extract statistics from it and to develop heuristics that could improve performances; **pandas**, to display user-friendly tables representing the results achieved; **scikit-learn**, to evaluate performances of the classifier; **conll**, a library we were provided with, to evaluate chunk-level performances of the corpus; and other minor libraries for pure data processing, including **itertools** and **zipfile**.

Note that the first block in the Notebook does not perform any of the three requested tasks, but it simply loads the `test.txt` file of the corpus into memory and stores it in a convenient format, accessible from the successive blocks.

I. Evaluation of spaCy NER model on CoNLL 2003 corpus

The first task required to evaluate the performances of the pre-trained *spaCy* English model, when applied to the provided *CoNLL 2003* data. The first problem we had to face regarded tokenization: the corpus we were asked to manipulate provided sentences that were already tokenized, whereas *spaCy* usually is fed with plain text. Looking more deeply at the documentation, I have found that the library provides the possibility to enforce tokenization, by excluding the `tokenizer` component from the *spaCy* processing `pipeline`, but, according to the maintainers, it could lead to sub-optimal performances. So I decided to explore both the possibilities, and to evaluate the classification performances in both cases.

A. Evaluate model enforcing tokenization

1) Prediction of the Named Entities: The procedure to apply *spaCy* language model while avoiding tokenization, is rather simple, and it is performed iteratively on the sentence level. The last part of the loop? contains a crucial step, that consists in substituting the `NER` labels generated by the *spaCy* model (derived from the *Ontonotes15* specification) with the ones used in the *CoNLL 2003* dataset. This mapping is non-trivial, since the fifteen specific *SpaCy* labels must be mapped to the four more general *CoNLL* labels. To solve this problem, I have searched for the description of each *spaCy* label and mapped it to the *CoNLL* one whose definition was the nearest among them. The table of mappings is reported inside the Notebook, and it is worth noting that in the case of labels related to numbers, I preferred to lose such kind of information setting the related tokens to "`non-NER`", rather than mapping them to a *CoNLL* label which was not really related to them.

2) Token-level accuracy evaluation: Once the `NER` labels were defined, I had to evaluate the accuracy of the classification at the token level. Initially, I decided to develop myself some code to derive the statistical measures needed, such as precision, recall and f1-measure for each class, but secondly I also used the `scikit-learn classification-report` class, to speed up and automatize the process. The results are presented in the Notebook, but it is worth highlight here some facts:

- The overall statistics are fulfilling. For example the accuracy and the weighted statistical values are over 90%.
- Only five classes out of nine have precision, recall and f1-value above 70%. Moreover, the only class performing optimally is the `O` class, which has statistical values above 95% and, having a support that is 30 times the one of the other classes, it raises the overall performances at a good level.
- For some classes the statistical measures are below the threshold of chance, probably due to the suboptimal mapping that is performed in the previous post-processing phase.

3) Chunk-level performance evaluation: To perform this phase of the evaluation, I simply used the provided `conll evaluate` function. Also this phase presented the problems defined above: some classes provide an acceptable level of performance, whereas some others underperform, probably because of the poor mapping used in the post-processing phase.

B. Evaluate full pipelineof the model

1) Prediction of the Named Entities: To evaluate the performances of the full pipeline of the *spaCy* model, the sentences were recreated by simply joining all the tokens together with a space character. Then the hypothesis were generated using the classical procedure, involving the `Language` object. At this point, the original tokenization had to be recreated. To do so, I used the `whitespace_` information contained in each *spaCy* `Token`. Also in this case, the same label mapping previously defined was used, in order not to alter the final results.

2) Token-level accuracy evaluation: As in the previous case, the statistics of the `NER` labelling were evaluated. Despite the usage of a different approach about tokenization, the results were very similar to the ones explained in the previous phase: the overall weighted statistical measures were around 90%, but looking at some classes, the performances were really poor.

3) Chunk-level performance evaluation: Furthermore, when considering the performances on a chunk-level, the results are similar to the ones obtained when enforcing tokenization, and thus they are not satisfactory. Probably this is due to the fact that the *spaCy* model has been trained on a corpus which is too different from the *CoNLL 2003* dataset and, for sure, the fact of mapping *spaCy* labels to *CoNLL* ones by hand did not boost the performances.

II. Grouping of entities

The second task we were asked to solve was about grouping of entities: we had to pass through the list of the `entities` and group together all the ones that generated a `noun_chunk`. As last thing, we had to extrapolate some statistics regarding the groupings in the *CoNLL 2003* dataset.

A. Group recognized named entities

As first thing I developed a function, group_ners(doc), that takes as input a `doc`, and extracts from it both the list of `entities` and the list of `spans` constituting a `noun chunk`. Then the algorithm passes through the list of `noun chunks` and decides whether to insert the whole `chunk` in a new list or only the currently pointed `entity` in the other list. At the end of the processing, the function returns a list of lists, representing the entities in the sentence grouped according to the information contained in the `noun_chunks` field of the `doc` object.

B. Analyze frequencies

As last thing, I applied the above function to the provided dataset, obtaining some statistics about the groupings present in the dataset. More precisely, for each of the fifteen labels provided by *spaCy* and looking at groups of cardinality between 1 and 4, I presented both the counts for each group and the most frequent combination of labels. Moreover, I also provided the counts and the statistics in the general case. It is worth noting two facts: in approximately 90% of the times, the groups are composed by just one label, meaning that we have in most of the cases the simplest possible configuration; moreover the labels tend to form stable couples/triplets (e.g. `(DATE, EVENT)`), and so they appear very frequently together.

III. Fix segmentation errors

The last task asked to use the `compound` dependency relation to fix segmentation errors in the parsing of the document. As first thing I developed some code to manipulate the preprocessed corpus, then I evaluated the results, using the techniques presented in the previous sections.

A. Fix errors

The approach I used to post-process the outputs of *spaCy*'s parsing is rather simple and is performed on the sentence level. I simply scanned the `tokens`, looking for the ones having `dep_` label equal to `compound` and having `ent_iob_` label setted to `O`. Once such a token was found, I checked if the following word was the `head` of the currently analysed token and if it had some `NER` information I could propagate back to its children. In case of positive check, the `NER` information was propagated to the current children, otherwise the analysed token was left blank. The whole procedure has been performed until convergence, meaning that it was repeated until no change was detected, in a way to be able to propagate `NER` information across chains of `compound` relations. The same procedure was performed also analysing the token and the previous word, in a way to propagate entity information in both directions.

B. Performance evaluation

The evaluation of performances has been carried out both at token level and at chunk level, as in previous case. The obtained data highlights that, unfortunately, the post-processing did not have a positive impact on performances. For sure, it did not lead to poorer classification, but it did not lead to expected improvements in accuracy.