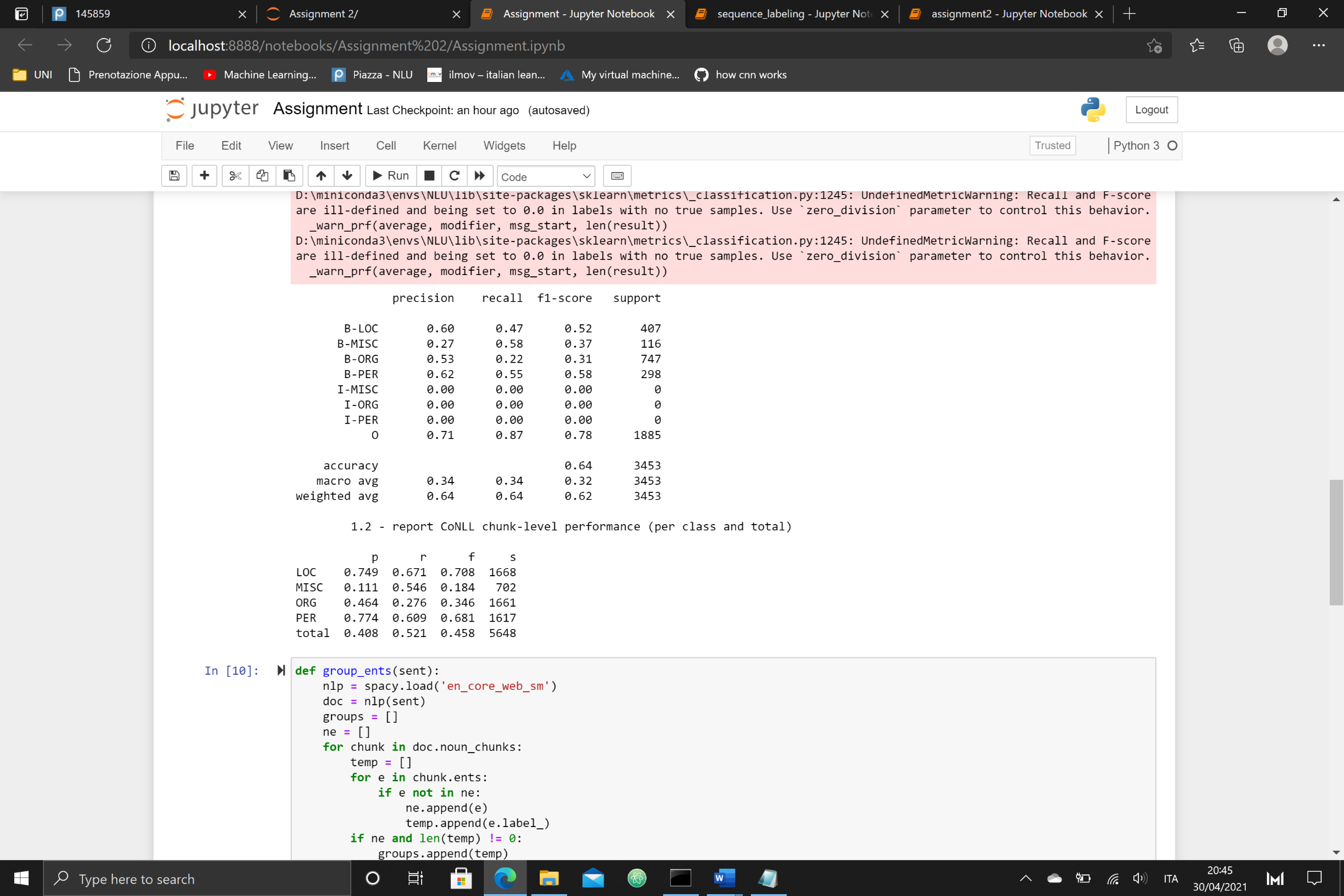
**Assignment NLU course**

* **Evaluate spaCy NER on CoNLL 2003 data**: to evaluate the performance of the spaCy NER model on the CoNLL dataset I defined three main functions:
  + **changeRep(token):** function that takes as input a token and gives as an output the converted label of that token. To be more precise this function is defined to change the model used to encode the NE label from CoNLL dataset model to the spacy one. For the input then it extracts the iob label to understand if it corresponds to ‘O’ or ‘B’ or ‘I’. In the first case it returns only the iob, so ‘O’ without any modification. In the two other cases, the function extracts the type of the label, and based on the value of it, it maps this value to a corresponding one of the spaCy models. Finally, it concatenates the iob and type to obtain the final label.
  + **get\_sent(sent):** function that takes as input a tokenized sentence and outputs a tuple containing in (token, label) format, where the label is transformed using the changeRep function. This function uses the whitespace\_ when recreating the sentence. This allows us to avoid issues in the following evaluation (for example ‘AL-AIN’ is processed as a whole instead of ‘AL’, ’-’, ‘AIN’)
  + **evaluate(path):** main function that takes as an input the path of the CoNLL dataset and outputs a report of the evaluation of the model. First it loads the dataset, and it removes the ‘-DOCSTART-’. After that, it extracts the ground truth labels from the dataset, by using the function written in the sequence labelling notebook. Then, for each ‘sentence’ stored in the dataset, it takes the token stored at the position [0] and it concatenates it until the end of the sentence in a string. This sentence is then passed in the spaCy parser creating the doc. After that the function calls the the get\_sent(doc) function and the output is concatenated in the appropriate list. At this point we have both the true labels and the ones to predict, so we feed the labels to the function that handles the classification, which uses the classification\_report of sklearn and outputs the evaluation details at token-level. Finally, the function calls the evaluation function of the conll.py file. Again, this outputs the report evaluation at chunk level. The model is not able to identify the intermediate labels, that is why the values are automatically set to 0 by the classification report.
* **Grouping of Entities:** for the testing part of this point I used the sentence “Apple's Steve Jobs died in 2011 in Palo Alto, California.”. The function takes a long time to process all the entries of the dataset, to avoid it is possible to reduce the number of entries computed. To obtain groups of named entities I defined two functions:
  + **group\_ents(sent):** this function takes as input a sentence and returns a list containing the grouped NE. To obtain this result, first the sentence is reduced to tokens, then each chunk is processed, by using the noun\_chunks function. For each chunk, it checks the entities contained in it. At this point we append the entities to two lists (if it was not already part of the ne list), one that is used to store the entity (used later) and one for the entity labels. Then if we actually appended entities in the previous step, after having processed each entity of the chunk, we append list with the labels to a general list. Finally, the function checks whether all the entities of the document are part of the general list, and if this is not the case, then it adds the ones that are not part of it. This causes the missing groups to be appended at the end of the list. For example, the result of the test sentence should be *[['ORG', 'PERSON'], ['DATE'], ['GPE'], ['GPE']],* while actually in my solution is *['ORG', 'PERSON'], ['GPE'], ['GPE'], ['DATE']]*
  + **get\_frequencies(path):** main function of the second step that takes as input the path to the dataset and outputs a dictionary containing the frequency in which each group of NE was found in the dataset. For each entry, the function creates a string which is fed to the group\_ents function, to obtain the groupings of that sentence. At this point, the function iterates through the output of the previous one and it concatenates the tags that are contained in the same list (so the ones grouped together), and this concatenated string is then used as the index of the dictionary. In this way we can increment the frequency value of each group.

FREQUENCIES

* CARDINAL: 1624
* GPE: 1255
* PERSON: 1074
* DATE: 997
* ORG: 873
* NORP: 293
* MONEY: 147
* ORDINAL: 111
* TIME: 83
* PERCENT: 81
* EVENT: 58
* LOC: 54
* CARDINAL PERSON: 51
* QUANTITY: 51
* NORP PERSON: 43
* GPE PERSON: 34
* GPE GPE: 26
* **Fix segmentation errors:** for the task of extending the entity span to cover the full noun-compounds I defined two functions:
  + **compound\_seg(sent):** function that takes as input a tokenized sentence and returns a list of lists containing the extended entities. The function iterates over the tokens of the sentence, and gets all the tokens which are part of a compound phrase, by using the get\_head function and scoring the values in a list. These values are then converted to indexes and stored again. The same indexes are stored separately to use them as a reference for the indexes that will not be added. Then, the other indexes are appended to the index list of lists, that is then sorted to obtain the original structure. Finally, we append the span of tokens that constitute a compound, by using the stored indexes.
  + **get\_head([], token):** recursive function that takes as input a list (containing the dependencies) and a token, and that gives as output the chain of compound dependencies. The function appends the compound token and iterates itself, until it reaches the head. In this way we are sure to obtain the right span

The test function gives as result:

“Apple”: B-ORG ; “'s” : O ; “Steve” : B-PERSON ; “Jobs” : I-PERSON ; “died” : O

“in” : O ; “2011” : B-DATE ; “in” : O ; “Palo” : B-GPE ; “Alto” : I-GPE ; “,” : O ;

“California” : B-GPE ; “.” : O