Università di Pisa, Master Degree in Computer Science
Course of ”Human Language Technologies” (HLT)

Named entity recognition for Clinical records

Report of the Final Project for the A.Y. 2022/2023

Andrea Iommi

January 16, 2023

Abstract

As final project for Human Language Technologies (HLT) I developed a project that extracts knowledge from Italian medical records written by physicians and provides a simple web interface to make prediction on sentences. I also compared the quality of the project's result with the result of MultiCoNER competition.

1 Introduction

Named entity recognition (NER) is a form of natural language processing (NLP), which is a subfield of artificial intelligence. NER is the task of detecting key information in texts, a specific entity can be only a word or a group of words that refer to the same concept. This task can be adopted in a plenty of fields (Human resources, Customer support, Content classification, Healthcare and so forth). In the paper, I focused on the Healthcare field, in particular I had to deal with the clinical records. Clinical records are very powerful tools from which useful information can be extracted, for example, to categorize the sentences based on the entities detected. In addiction, several Data mining techniques can be applied (e.g. Association Rule Mining [1]). Extracting relevant and useful entities is important but, unfortunately, it is not straightforward because the same concept in a text can be written in a different way among the physicians, such as abbreviations or synonyms. Therefore, we need a sophisticated technique of language analysis in order to recognize these expressions.

2 Related Work

In the early state of NLP several tasks like NER are solved by Symbolic Artificial Intelligence (SAI). It uses human-readable symbols that represent real-world entities or concepts in order to create "rules" for the manipulation of those symbols, leading to a rule-based system. As AI has progressed, we have moved on techniques such as Conditional Random Fields (CRF) or Support Vector Machines (SVM) models provided by Machine Learning field. It is worth noting that in 2015 [2] a task similar to ours has been solved with the CRF, which was, at that time, the state of the art. In these last years, a new Machine Learning framework has earned the supremacy in the NLP: it is called BERT (Bidirectional Encoder Representations from Transformers). BERT is a model which is composed of several transformer layers with a Multi-Head attention module that allows to understand better (respect to previous models) the context of the sentences. BERT is a general pre-trained framework, so this allows us to use it as a baseline and to extend it in order to solve a specific task like NER.

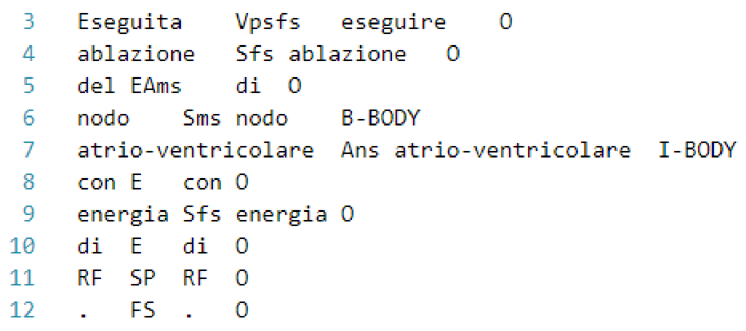
2.1 BERT model

The corpus used in this project is in italian, thus I had to choose a model that it was pre-trained with an Italian dataset. The model chosen for this task was provided by Huggingface [5]. In particular, I utilized *dbmdz/bert-base-italian-xxl-cased*, where the word *cased* means that there is a distinction between the lower and the upper case. For the NER, I need a model that classify each tokens individually into the sentence received as input, so I adopted *BertForTokenClassification*. BertForTokenClassification is a model that implements the BERT and an additional linear layer at the end used for the classification. Actually, a complex solutions such as MultiCoNER (which will be introduced later) has adopted another point of view. It adopts a further CRF layer as sequence tagging instead of single token classification, because usually a tag for one specific token is affected by the previous tags. However in my project I adopted a simplistic solution and I will show that the results are not so much different, even if the MultiCoNER provided better results in some metrics (respect to our task).

3 Medical records corpus

3.1 CoNLL and IOB Format

The italian medical records I have used have been formatted by CoNLL [3], this means that we have one token (we can consider a token as a single word) for each line. Additional informations such as POS tag, labels or general annotations are separated by a single tab, instead different sentences are separated by an empty line. In the project, I need a label for each token that identify if the token is associated to a specific tag (Person, Location, Facilities, and so on). Sometimes one tag could be associated to more than one single word, for example: *"Mario Rossi is a policeman"*, *"Mario rossi"* are two tokens but together the tag is "Person". To solve this, the annotations are coded by IOB format[6] (Inside–outside–beginning tagging), where the I- prefix before a tag (e.g. "I-PER") indicates that the tag is inside a group. An *O* tag indicates that a token belongs to no group. The B- prefix before a tag indicates that the tag is the beginning of a group that immediately follows another group without *O* tags between them. It is oblivious that in order to obtain a well-formatted IOB, we have to follow a precisely order to assign the tag (first B then I).



```
Eseguita    Vpsfs   eseguire    O
ablazione   Sfs ablazione   O
del EAms    di  O
nodo    Sms nodo    B-BODY
atrio-ventricolare  Ans atrio-ventricolare  I-BODY
con E   con O
energia Sfs energia O
di  E   di  O
RF  SP  RF  O
.   FS  .   O
```

Figure 1: Example of CoNLL format and IOB annotation

In figure 1, we can see that we have a token for each line, and further information (POS tag, Lemmatized token and entity). For our purpose the POS tag and lemmatized token are not utilized. The figure also shows how IOB annotation works.

3.2 Entities

The dataset I used was composed in two groups, each group shares the same sentences but it was annotated by different tags. To identify entities of interest in text I used two classifiers: NER A, for Disease or Syndrome, Drug, Active Ingredient and Sign or Symptom; NER B for Body Part and Treatment; NER A and NER B are used on sets of disjoint categories, i.e., each mention belongs to a single category.

Group A	**Group B**
Disease or Syndrome	Body part
Drug	Treatment
Active Ingredient	
Sign or Symptom	

Table 1: Entities divided in groups

3.3 Sentences

In order to train the model, I have pre-processed the datasets. The original datasets were composed for the group A: *ananmnesi.a.iob* and *esami.a.iob*; and for the group B: *ananmensi.b.iob* and *esami.b.iob*. The first phase was unify all documents in only two datasets: *dataset.a.conll* and *dataset.b.conll*. The second phase, instead, takes the dataset and it parses it into a set of sentences. Finally, both datasets A and B reach *265790* sentences and, after the removal of duplicates, the the number was reduced to *107950*[1]. The third (and last) phase consists into splitting the entire datasets by holdout

[1]The huge number of duplicate is provided from the fact that inside the datasets there are several sentences, repeated many times, that are composed by few words (e.g. 3 - 4 tokens) and in the most cases they are not so much informative.

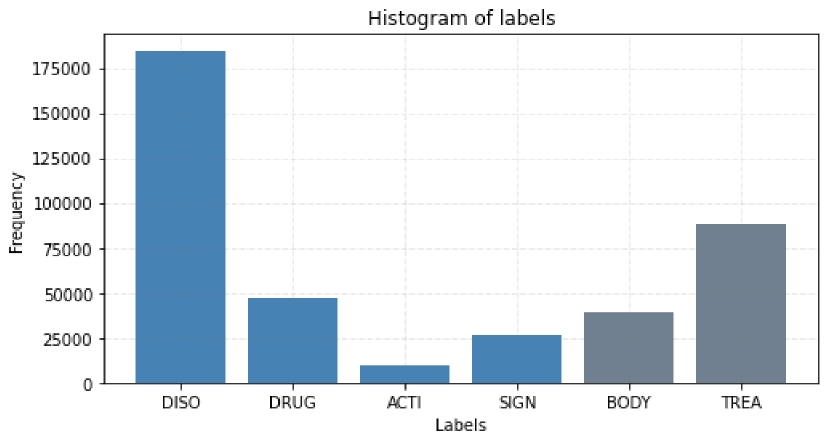


Figure 2: Distribution of entities used

technique (80%TR - 10%VL - 10%TS), obtaining *86360* sentences for training, *10795* for validation and *10795* for test.

4 Experiments

4.1 Hyperparameters

As preliminary phase, I selected the parameters which were to consider "static" (fixed as constant), and parameters to try in different configurations. I chose the following parameters as static parameters: *Optimizer, Momentum, Batch size, Weight decay, Max epochs, Early Stopping*. Concerning the batch size, it was done various attempts and the value assigned was very influenced by the GPU memory and the computational power available. Regarding to Max epochs, also in this way since the resource and time was limited, we chose to apply a barrier to a maximum number of epochs per model. However we discovered that, even if the threshold was set with a low number, the model reaches a very good performance after few steps. This is due to the fact that the model is already trained, so a few iterations is enough to specialize in our task. Sometimes, I noticed that the model diverge to overfitting, nevertheless given a Early Stopping mechanism I had the opportunity to stop the train and avoid unnecessarily additional iterations. Talking about the free parameters, there is the Learning Rate, they were tried exactly 7 distinct values. The table 2 shows all configurations tried for each NER separately (14 in total).

Parameters	Values
Learning rate	0.0005, 0.001, 0.002, 0.004, 0.006, 0.008, 0.010
Momentum	0.9
Weight decay	0.0002
Batch size	16
Max epochs	12
Early Stop.(Patience)	3
Optimizer	Stochastic gradient descent
Nesterov mom.	True

Table 2: Hyper-parameters

4.2 Grid search's results

Position	Learning rate	Optimal Epoch	Stopping Epoch	Tr loss	Vl loss	Vl F1-score
1	**0.004**	**8**	**12**	**0.0061**	**0.0164**	**0.9627**
2	0.006	3	6	0.0095	0.0165	0.9595
3	0.002	5	8	0.0048	0.0173	0.9543
4	0.008	2	5	0.0141	0.0179	0.9550
5	0.010	2	5	0.0152	0.0191	0.9529
6	0.001	8	11	0.0035	0.0194	0.9600
7	0.0005	12	12	0.0149	0.0258	0.9486

Table 3: Grid search for NER A

Position	Learning rate	Optimal Epoch	Stopping Epoch	Tr loss	Vl loss	Vl F1-score
1	**0.008**	**4**	**7**	**0.0037**	**0.0045**	**0.9870**
2	0.004	9	12	0.0009	0.0047	0.9878
3	0.006	8	12	0.0014	0.0048	0.9877
4	0.002	9	12	0.0005	0.0055	0.9867
5	0.001	10	12	0.0006	0.0063	0.9833
6	0.010	3	6	0.0043	0.0070	0.9727
7	0.0005	12	12	0.0042	0.0085	0.9751

Table 4: Grid search for NER B

The table 3 and 4 are composed by different columns: the learning rate, the epoch where the model achieves the best score (in terms of minimum validation loss) called *"Optimal Epoch"*, the epoch where the execution is terminated by Early stopping called *"Stopping Epoch"*, training loss *"Tr loss"*, validation loss *"Vl loss"* and the *"F1-score"*, which representing the mean of all F1-scores, calculated one for each entity. The results of table 3 is associated to the NER with the group of entities A, the table 4 to the group B. Analizing the results, I can point out basically two things:

- The F1-scores in table 4 are higher respect to the ones in table 3, and this phenomenon could be explained by the fact that the group A contains more entities respect to the group B. More precisely, the group A contains 4 labels, the group B only 2, thus the NER A has more etiquettes to assign and hence the error rate is higher. Another explanation could derive from the sentences built: in fact, inside the datasets, there are some phrases where it is clear that a token is assigned to a certain label, conversely sometime it is ambiguous (if you do not know the solution, of course);
- The configurations with high learning rate have the particularity to achieve the optimal epoch with a very few steps, in addiction the execution ends after few iterations because the model exhibits overfitting symptoms.

4.3 MultiCoNER and related problems

The *Multilingual Complex Named Entity Recognition* (MultiCoNER) [7] is a project developed by Amazon that represents the state of the art for NER task. I adopted the MultiCoNER as reference to check the quality of project (for simplicity I call *HLTProject*). In order to make possible this comparison I utilized the same BERT pretrained model (*dbmdz/bert-base-italian-xxl-cased*) and dataset. There are multiple differences between this two project both in the framework exploited and in model implementation. Regarding the framework the mainly difference is that the MultiCoNER use Pytorch-Lightning for training and testing. Concerning the model implementation, as previously said, the MultiCoNER deploys a further CRF layer over the BERT. There are others small differences regarding the implementation but they do not influence the final results and thus they will not be mentioned. In order to run the MultiCoNER, I had to fix some errors inside the code, and I also encountered several problems regarding the python environment, however creating an ad hoc environment I were able to use the project. Table 5 shows the parameters for MultiCoNER executions.

Parameters	**Values**
Learning rate	0.0001
Dropout	0.1
Batch size	64
Max epochs	5
Optimizer	Adam

Table 5: MultiCoNER hyperparameters

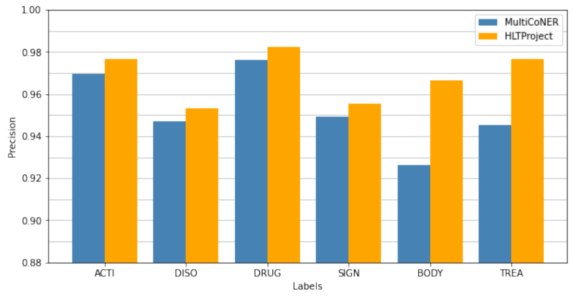
The MultiCoNER requires that the training, validation and test datasets are in separated files. Since the CoNLL reader was already implemented, I just split manually the *dataset.a.conll* and *dataset.b.conll*, that I already mentioned in order to obtain *train.a.conll, train.b.conll, val.a.conll, val.b.conll, test.a.conll and test.b.conll*. It is fundamental to point out that, for example, the *train.a.conll* shares exactly the same sentences that are utilized for training dataset of HLTProject, and this holds both for validation and test. Also in this way I built two model MultiCoNER A and MultiCoNER B.

4.4 Results

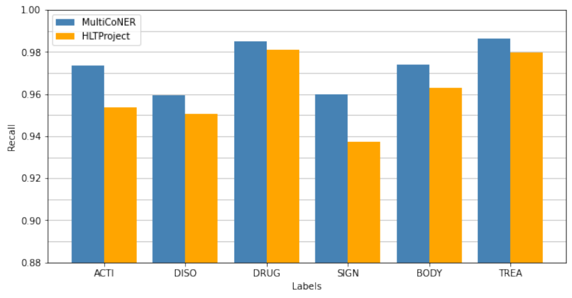
Table 6 and figure 3 illustrate the results provided by MultiCoNER and HLTProject on test datasets. From the results we can say that I have obtained very good results. Concerning the precision, we obtain better results by HLTProject from all entities, conversely, regarding the Recall Metric, MultiCoNER performs better. It is a different situation for F1 score where there are some entities that are classified better by HLTProject and others with MultiCoNER.

	ACTI	**DISO**	**DRUG**	**SIGN**	**BODY**	**TREA**
HLTProject - Precision	**0.9764**	**0.9530**	**0.9824**	**0.9552**	**0.9663**	**0.9767**
MultiCoNER - Precision	0.9696	0.9471	0.9761	0.9493	0.9264	0.9452
HLTProject - Recall	0.9538	0.9504	0.9808	0.9373	0.9630	0.9799
MultiCoNER - Recall	**0.9733**	**0.9593**	**0.9850**	**0.9597**	**0.9739**	**0.9864**
HLTProject - F1	0.9650	0.9517	**0.9816**	0.9461	**0.9646**	**0.9783**
MultiCoNER - F1	**0.9715**	**0.9532**	0.9805	**0.9545**	0.9495	0.9654

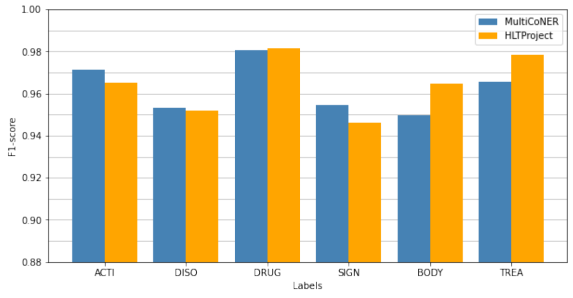
Table 6: Result



(a) Precision



(b) Recall



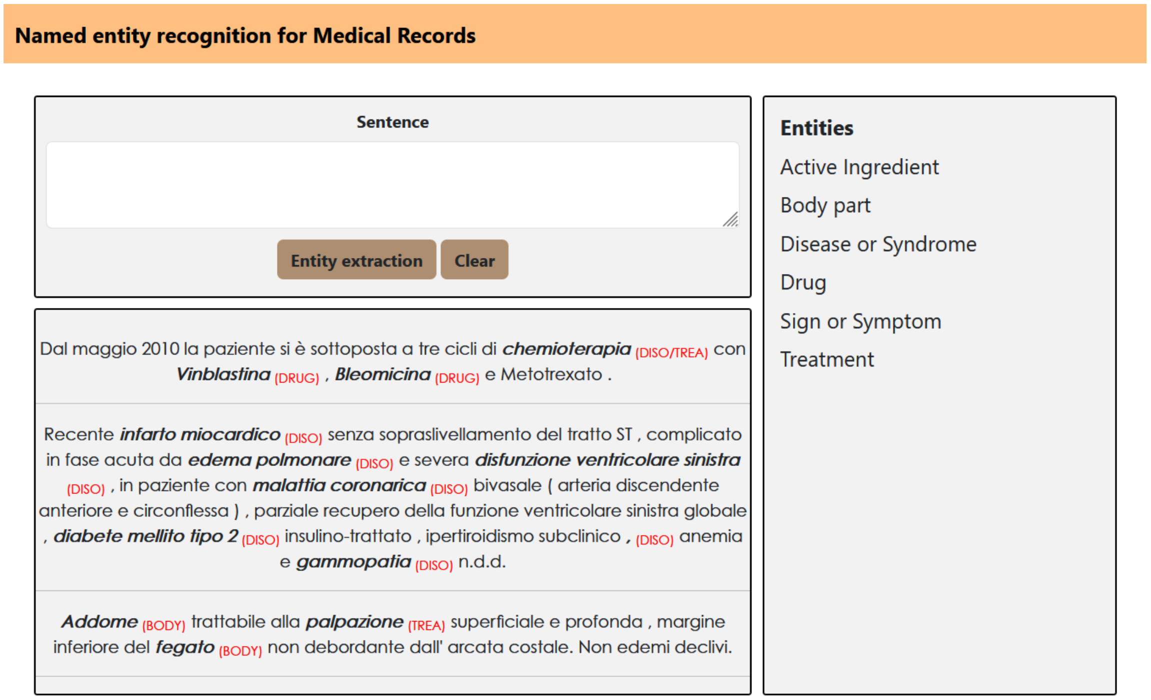
(c) F1 - score

Figure 3: Results compared

5 Web-interface and Execution

5.1 Web-interface

Figure 4 reveals a naive web interface where an user can interact with the project. On the left there is a list of all entities that the project handles, and we can notice that there are not distinctions between group A and group B, in fact it uses both. As soon as the user inserts the sentence, the latter is submitted at the same time to *model_A* and *model_B* (the best model for each group), then the program unifies the response in a unique result, which involves, potentially, all entities together. On the right, under the input form, there is a history of sentences processed. For example, in the first sentence there is the word *chemioterapia* that is labeled simultaneously as *Disease or Syndrome* and *Treatment*, which are entities that belongs to different groups. Of course both models are not perfect, and sometimes can occur some erroneous values, that I will discuss in the conclusion paragraph.



Dal maggio 2010 la paziente si è sottoposta a tre cicli di ***chemioterapia*** (DISO/TREA) con ***Vinblastina*** (DRUG) , ***Bleomicina*** (DRUG) e Metotrexato .

Recente ***infarto miocardico*** (DISO) senza sopraslivellamento del tratto ST , complicato in fase acuta da ***edema polmonare*** (DISO) e severa ***disfunzione ventricolare sinistra*** (DISO) , in paziente con ***malattia coronarica*** (DISO) bivasale (arteria discendente anteriore e circonflessa) , parziale recupero della funzione ventricolare sinistra globale , ***diabete mellito tipo 2*** (DISO) insulino-trattato , ipertiroidismo subclinico , (DISO) anemia e ***gammopatia*** (DISO) n.d.d.

Addome (BODY) trattabile alla ***palpazione*** (TREA) superficiale e profonda , margine inferiore del ***fegato*** (BODY) non debordante dall' arcata costale. Non edemi declivi.

Entities

Active Ingredient

Body part

Disease or Syndrome

Drug

Sign or Symptom

Treatment

Figure 4: Example of web interface with some example of use

5.2 How to run

To develop this project from scratch they were adopted different frameworks, basically: *PyTorch, Pandas, NumPy and Flask*. Flask was used to create a very straightforward web-interface that interact with the trained model. The code of this project is entirely available on Github [8].

train_model.py

train_model.py handles the training phase. The mandatory parameters are: the source of dataset and the name of model; all the rest is optional (I suggest to read the documentation on github). I supply to the project as example *dataset.a.conll*, so the script trains the model with the group A of entity. Inside the implementation the dataset will be split into training, validation and test, but only the training and validation will be used. A static seed (for the random splitting) was designed to obtain the determinism in multiple executions.

```
#!/bin/bash
python train_model.py --datasets dataset.a.conll --model_name modelA
```

evals_models.py

evals_model.py manages the evaluation phase. Contrary to train_model.py, this time only the test dataset is used (thanks to determinism we are sure that we are using always the same test dataset even with multiple executions). The mandatory parameters are: the source of datasets and the name of models. In this case it requires to load both datasets and models trained. We can perform two kinds of evaluation: the evaluation provided by *Conlleval* [4] or the hand-made implementation.

```
#!/bin/bash
python eval_models.py --models modelA.pt modelB.pt
    --datasets dataset.a.conll dataset.b.conll  --type_eval conlleval
```

6 Limitations and Conclusion

I would like to conclude this report with some considerations. First of all, the comparison with MultiCoNER gives us insight that CRF (used for sequence tagging) is very advantageous as an additional layer over the BERT. In fact, with only 5 epochs the MultiCoNER gave extraordinarily good results and sometimes even better results then HLTProject. However, the aim of this project was not to go into details concerning the MultiCoNER and for this reason I did not explore more suitable parameters. This means that the results displayed could be worst respect the potential of code. Secondly, even if HLTProject achieves good results, as we can see in figure 4, it is not perfect and sometimes trivial errors occur (e.g. the comma labeled as DISO). However, since the time spent for this project was limited, some advanced solutions have not been adopted. It is worth saying that extending the HLTProject with a sequence tagger such as CRF gives it the opportunity to perform better.

At the end of this project, I have learned how to deal with BERT framework or, more in general, with NLP. I also improved my personal skills with Python languages and frameworks like *pytorch, pandas and flask*. I would like to thank Daniele Sartiano for helping me with MultiCoNER project.

References

[1] *Association Rule Mining*. URL: https://en.wikipedia.org/wiki/Association_rule_learning.

[2] Giuseppe Attardi, Vittoria Cozza, and Daniele Sartiano. "Annotation and Extraction of Relations from Italian Medical Records." In: *IIR*. 2015.

[3] *CoNLL*. URL: https://universaldependencies.org/format.html.

[4] *CoNLL script evaluation*. URL: https://github.com/sighsmile/conlleval.

[5] *Huggingface*. URL: https://huggingface.co/.

[6] *Inside–outside–beginning (tagging)*. URL: https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging).

[7] *Multiconer*. URL: https://github.com/amzn/multiconer-baseline.

[8] *NER for Medical Records*. URL: https://github.com/jacons/NERMedicalRecords.