## Thesis PIO element extraction on RCT studies

### Repository content

In this repository you can find the project described in the thesis (*Thesis\_v1.pdf*). We provide the following information: - Replication instructions for the model described in the thesis (see replication instructions below) - Code with which the scores for the naive baseline was calculated (*naivebaseline.py*) - Brief EDA on the dataset and the counts of each element in the training set (*eda.py* and *picocounts.xlsx*) - Results of round 1 and round 2 of the error analysis as described in the thesis (read README.md in *error\_analysis* folder for more information) - Results of exploring the data saturation (*saturation.xlsx*)

To replicate the data saturation numbers simply take the first 20%, 40%, 60%, 80% and 100% of the dataset and run the model with this data on the same config file with just one modification: change the eval\_frequency hyperparameter inside the config.cfg file with respect to the percentage of the training set to compensate for the decreasing size. So, eval\_frequency is 200 for 100% of the training set, 160 for 80% etc.

### Dependencies:

python 3.8. spaCy version 3.1.3 (pip install spacy==3.1.3) + en\_core\_web\_lg pipeline (python -m spacy download en\_core\_web\_lg).  
numpy version 1.18.5 (pip install numpy==1.18.5).  
pandas version 1.0.5 (pip install pandas==1.0.5).  
matplotlib version 3.2.2 (pip install matplotlib==3.2.2).  
seaborn version 0.10.1 (pip install seaborn==0.10.1).  
scikitlearn version 0.23.1 (pip install scikit-learn==0.23.1).

### Table 1: All entities with their label inside the transformed dataset.  
P = Population, I = Intervention, O = Outcome.

|  |  |
| --- | --- |
| Label | Entity |
| 0 | None |
| 1 | P: Age |
| 2 | P: Sex |
| 3 | P: Sample Size |
| 4 | P: Condition |
| 5 | I: Surgical |
| 6 | I: Physical |
| 7 | I: Drug |
| 8 | I: Educational |
| 9 | I: Psychological |
| 10 | I: Other |
| 11 | I: Control |
| 12 | O: Physical |
| 13 | O: Pain |
| 14 | O: Mortality |
| 15 | O: Adverse Effects |
| 16 | O: Mental |
| 17 | O: Other |

## Replication instructions main model:

### 1. Clone the repository

### 2. Unzip the ‘ebm\_nlp\_2\_00.tar.gz’ file so you get the ebm\_nlp\_2\_00 folder in the root directory.

### 3. Execute the prepare.sh script which prepares the data by running the following files (could take up to a couple minutes depending on your hardware):

#### loaddata.py:  
This file first loads all the .tokens files which hold the tokenized texts and all the .txt files which hold the full texts. The tokens and documents are stored in seperate arrays together with the document ID for each document. Now all the individual annotations are read into arrays. The dataset provides individual annotations for every PIO element but since we want to have a dataset where all entities are combined, we merge the annotations. We do this by first making all the assigned labels distinct (see table 1) so that we can uniquely identify each of the 17 elements (18 if you count the ‘None’ class).

#### reformdata.py:  
Here we use the loaded data from the last file to apply some transformations. Firstly, the model requires the annotations to provide the character indices of the beginning and the end of all the annotated tokens instead of the text of the tokens. Therefore, from the token-level annotations we generate character-level annotations instead. Next, we match the annotations with the corresponding texts and tokens using the document ID’s we stored. Lastly, we map the numerical labels to the entity names enumerated in table 1 so that the labels are more intuitive.

#### ner\_pico.py: After the previous steps the data is in the following format:  
[(‘None’, ‘Effect’), (‘None’, ‘of’), (‘I: Drug’, ‘aspirin’), (‘None’, ‘for’), (‘O: Outcome’, ‘headaches’)]  
Where the left element of each tuple is the entity and the right element the token. We want to transform it into the following format:  
[‘Effect of aspirin for headaches’, {entities: [(8, 14, ‘I: Drug’), (18, 26, ‘O: Outcome’)]}] Here we have the full text on the left and all the entities (excluding the ‘None’ class) with the character indices in a dictionary. This was the original format for spacy version 2.x and can easily be converted to the .spacy format used in spaCy 3.x.  
To get this result we first must change the tokens by the character indices we generated in reformdata.py and filter out the ‘None’ instances. Now we make a tuple of the full text and the dictionary containing the entities. Out of this format we can make instances of spaCy’s ‘Doc’ objects. All these ‘Doc’ objects are stored in a DocBin object. This DocBin gets stored in the .spacy format and now we are ready to train our model with this .spacy file.

### 4. Execute the *model.sh* script inside the *spacy\_acc* folder which trains and evaluates the model and saves the model in the *output* folder.

For training the model, the hyperparameters described in the config.cfg file are used. The output can be seen in the file scores.txt after running the script.  
Running this script could take up to a couple hours. If you have a GPU available, you can add *–gpu-id = 0* at the end of the spacy train command to speed it up considerably.