Terminology project
TERM TAGGER FOR THE NLP DOMAIN

AGUIAR Mathilde NIAOURI Dimitra

16 December 2022

Contents

1 Introduction

The aim of this project is to create and evaluate a term identification system adapted to the domain of Natural Language Processing (NLP) using a supervised approach. A corpus of 120 abstracts of scientific papers on NLP were collected and annotated manually using the IOB tagset. 4 models were implemented: a Random Forest Classifier, a Conditional Random Fields, a Spacy, and a BERT model.

2 Dataset

For easier use of our models, we release the dataset of `final` files, manually cleaned. We split the dataset into 3 folders: `train` `dev` `test`. The dataset has been collaboratively collected and annotated manually by different groups of the M2 Terminology class and now consists of gold-standard annotations of abstracts from NLP scientific papers. It needs to be noted that annotated terms are words related to NLP terminology.

Annotation process In groups of two, each student produced an annotated file following the `IOB` format and the guidelines. A 3rd student from another group reviewed these 2 files and produced a final annotation. The `IOB` format that was followed is presented below:

- `B` – indicating the beginning of the term
- `I` – indicating that the token is inside the term
- `O` – indicating that the token is outside of the term

Dataset characteristics In the following tables, the dataset characteristics are presented. Specifically, the number of files of each dataset folder is presented along with the size of each folder. Additional information is given about the number of tokens and sentences included in the dataset.

Dataset	Number of files	Size
Train	95	156.9 kB
Dev	12	17.7 kB
Test	13	22.6 kB
Total	120	197.2 kB

Table 1: Dataset size

Dataset	Number of Tokens	Number of Sentences
Train	19356	692
Dev	2222	82
Test	2772	105
Total	24350	879

Table 2: Number of tokens and sentences

Qualitative analysis of the annotation For the qualitative analysis of the annotation, we calculated the similarity score between the different annotated versions of files from the dataset. To do so we used the Python `SequenceMatcher`. The results are summed up in Table 3.

Annotators	Similarity
ann1 and *ann2*	43.66%
ann1 and *final*	60.56%
ann2 and *final*	54.27%

Table 3: Similarities between different annotations

Remarks At first glance, the annotation task could seem simple but the goal is to have a unified dataset where the files follow the same format and can be used by anyone for their experiments without a huge effort on data pre-processing. However, after a closer look at the files, we notice that there are a lot of inconsistencies (encoding problems, missing space, etc.) that make the similarity score drop quite fast. An interesting approach would be to provide the code of the tokenization of the original abstracts so the result format would be homogeneous. From our observations presented in Table 3, the overall results are low. It seems that the final annotation mostly agrees with Annotator 1, but this is again subject to formatting problems.

3 Baseline

As the baseline of our term tagger, we first tested a simple Random Forest Classifier along with a Conditional Random Fields model. We decided to implement a classifier-based approach and a sequence tagging approach as our baselines as those are well-suited for tasks of named entity recognition (NER), resembling the task of term identification that our project focuses on. As our main goal, we set the identification of all textual mentions of the NLP terms with a particular aim to identify the boundaries of the NLP terms included in our dataset. In the following sections, the two models tested will be presented in detail.

3.1 Random Forest Classifier

Our initial approach towards the creation of our term tagger was the implementation of a simple Random Forest Classifier, a tree-based model, using a simple feature map. We included this model as a part of our baseline due to the advantages that it offers with respect to efficacy, computational power, and time.

3.1.1 Methodology

In this section, the methodology used for the implementation of the Random Forest Classifier is presented. Detailed information about the data pre-processing and the model design is given.

Data pre-processing Despite our releasing a cleaner version of the manual annotation of our dataset, we had to make sure that no inconsistencies are left in the final version of our annotated data. For this reason, we pre-processed our data, cleaning them from empty values, punctuation (except for "-"), and some additional mistakes in the annotation not so easily detectable from a human annotator (e.g., the use of "space" instead of "tab" for the separation of tokens from tags). Additionally, in order to facilitate the implementation of our model we structured our data in separate columns using `pandas` dataframes as follows:

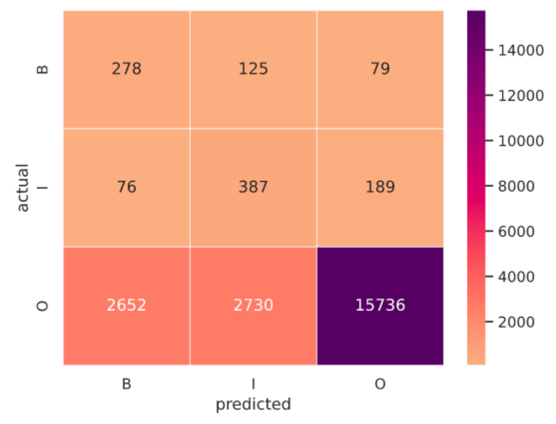
- `Index` - Index number of each token
- `Sentence_id` - The number of sentences in the dataset
- `Token` - Separate tokens
- `POS` - Parts Of Speech tags, added using spaCy following thus the Penn TreeBank Tagset
- `Tag` -The tags given to each token following the IOB tagset.

Model and training procedure For the implementation of our model, we used a simple feature map. Among the features included are the following: `.istitle()`, `.islower()`, `.isupper()`, `len()`, `.isdigit()` and `.isalpha()`

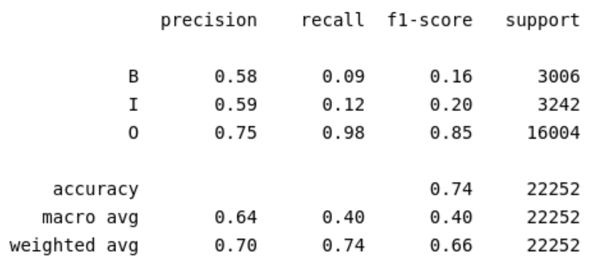
We applied a `5-fold cross-validation` as an input parameter to the classifier dividing and shuffling our dataset into 5 subsets and training-testing them to compensate for the small dataset and make sure that our inputs are not biased in any way. In order to avoid overfitting and get a good performance, we trained our model on one subset and tested it on the other, and repeated the process.

3.1.2 Results

The results of the Random Forest Classifier are presented in the following figure.



(a) RF classifier - confusion matrix



```
              precision    recall  f1-score   support

           B       0.58      0.09      0.16      3006
           I       0.59      0.12      0.20      3242
           O       0.75      0.98      0.85     16004

    accuracy                           0.74     22252
   macro avg       0.64      0.40      0.40     22252
weighted avg       0.70      0.74      0.66     22252
```

(b) RF classifier - report

Figure 1: Results of the Random Forest classifier

The results suggest that the model had a 74% overall accuracy. Further examining the results we notice that the F1-macro score is only 40% suggesting that the labels `B` and `I` were not learned properly during the training time. Even though the metrics for the `O` tag could be sufficient, the scores of the other tags are very low suggesting that features that could facilitate the predictions of the model are missing. In this way, the model seems to be memorizing words and tags which is not sufficient for an accurate prediction of the NLP terms. This is easier shown by the low F1-score in each of the underrepresented labels: 16% for the tag `B` and 20% for `I`.

3.2 Conditional Random Fields (CRF)

As a continuation of our baseline approaches, we implemented Conditional Random Fields, a model that is usually applied for labeling or parsing sequential data. This model, proven to have better performance in NE recognition than the tree-based ones, predicts the most likely sequence of labels corresponding to a sequence of inputs. The advantage of this method is that it takes context into consideration, contrary to tree-based models that do not.

3.2.1 Methodology

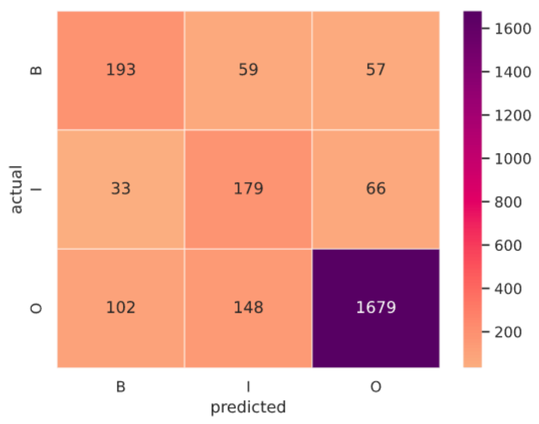
In this section, the methodology used for the implementation of the Conditional Random Fields is presented.

Data pre-processing The same data-preprocessing procedure as the one in section 3.1.1 was used.

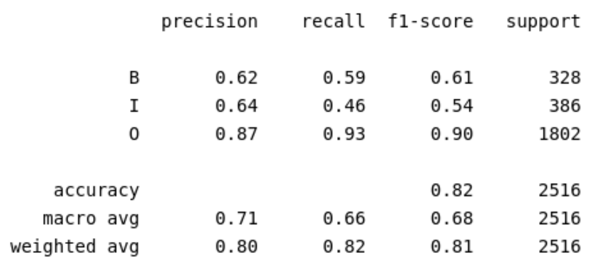
Model and training procedure For the implementation of our model, we used `sklearn-crfsuite` on our data set. It needs to be noted that our dataset comprised a training set (the data of the dev set were added here) and a test set. Additionally, we enhanced the feature set by creating more features to be taken into consideration. Specifically, POS tagging was added along with other features such as word parts, and lower/upper/title/digit flags.

3.2.2 Results

In the following figure our results are presented:



(a) Conditional Random Fields - confusion matrix



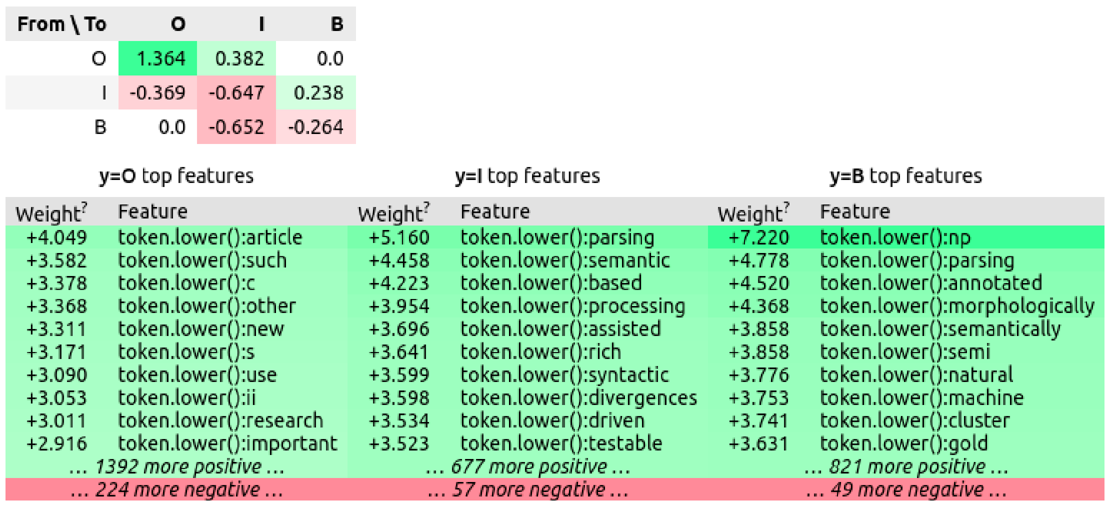
```
              precision    recall  f1-score   support

           B       0.62      0.59      0.61       328
           I       0.64      0.46      0.54       386
           O       0.87      0.93      0.90      1802

    accuracy                           0.82      2516
   macro avg       0.71      0.66      0.68      2516
weighted avg       0.80      0.82      0.81      2516
```

(b) Conditional Random Fields - report

Figure 2: Results of the Conditional Random Fields

The results of this experiment suggest a better performance for the conditional random fields tagger than the Random Forest Classifier. The overall accuracy is 82%, significantly better than the one of the Random Forest. An improvement in the F1-macro score is also attested (68% from 40%) suggesting that the underrepresented labels `I` and `B` are better predicted by this model. Specifically, the F1-score of the tag `B` went from 16% to 61% and the tag `I` went from 20% to 54%.

For a better understanding of the results, we plotted the weights of the top features of the model:



From \ To	O	I	B
O	1.364	0.382	0.0
I	-0.369	-0.647	0.238
B	0.0	-0.652	-0.264

y=O top features		y=I top features		y=B top features	
Weight?	Feature	Weight?	Feature	Weight?	Feature
+4.049	token.lower():article	+5.160	token.lower():parsing	+7.220	token.lower():np
+3.582	token.lower():such	+4.458	token.lower():semantic	+4.778	token.lower():parsing
+3.378	token.lower():c	+4.223	token.lower():based	+4.520	token.lower():annotated
+3.368	token.lower():other	+3.954	token.lower():processing	+4.368	token.lower():morphologically
+3.311	token.lower():new	+3.696	token.lower():assisted	+3.858	token.lower():semantically
+3.171	token.lower():s	+3.641	token.lower():rich	+3.858	token.lower():semi
+3.090	token.lower():use	+3.599	token.lower():syntactic	+3.776	token.lower():natural
+3.053	token.lower():ii	+3.598	token.lower():divergences	+3.753	token.lower():machine
+3.011	token.lower():research	+3.534	token.lower():driven	+3.741	token.lower():cluster
+2.916	token.lower():important	+3.523	token.lower():testable	+3.631	token.lower():gold
… 1392 more positive …		*… 677 more positive …*		*… 821 more positive …*	
… 224 more negative …		*… 57 more negative …*		*… 49 more negative …*	

Figure 3: Inspection of model weights

The results of the weight inspection indicate that `I` is less commonly followed by `I` or `O`, `O` by `I`, and `B` by `B`. On the other hand, `B` is more likely to be followed by `I` and `O` by `O`. We can additionally notice the most frequent tokens in each `IOB` category. For example, "semantic", "processing" and "parsing", are one of the most frequent tokens appearing under the tag `I`. Similarly, "morphologically", "semantically" and "annotated" are some of the most frequent tokens under the category `B`.

4 Spacy Model

The second approach we studied is based on the Spacy library. We used the Spacy classic framework to tokenize, train and test our model. The overall goal was to train the NER Pipeline Spacy component so it would be able to tackle NLP terms.

4.1 Methodology

Data pre-processing To train the Spacy Pipeline we needed to format the data in a supported format. The first part included reformatting `.final` files into a correct format (suppressing tags non-linked to a token, doubled tags (e.g `II` instead of `I`), wrongly encoded tokens, etc. as presented in section 3.1.1.

Adapting the IOB tags The supported IOB tag format for Spacy NER is the following: `B-CATEGORY` `I-CATEGORY` and `O`. For this reason, we transformed our simple `B` and `I` tags into `B-NLP` and `I-NLP`. In this way, the NER Pipeline only considered one category, the `NLP` one and the new files were stored into `.iob` files.

From IOB files to Spacy DocBin In order to train the Pipeline with the `IOB` tagset, Spacy requires to convert the `.iob` files into `.spacy` *DocBin* binary files. To do so, we used the `convert` CLI command and specified which model (*en_core_web_sm, en_core_web_md*, etc.) we were using to do the conversion.

Model and training procedure Fine-tuning the Spacy Tagger Pipeline was an easy procedure. We simply had to run the command line including the `.spacy` files generated from the train and dev datasets. This generated different models: `model-best` and `model-last`. We kept `model-best` for the rest of the experiments.

Configuration file The `.cfg` configuration file was fed to the Spacy Pipeline to define the different training hyper-parameters and other configuration options. The following table sums up the different configurations tried:

Training parameters				
Pipeline name	en_core_web_sm	en_core_web_md	en_core_web_lg	en_core_web_trf
Components	Tok2Vec and NER	Tok2Vec and NER	Tok2Vec and NER	Transformers (RoBERTa-base) and NER
Tokenizer	Spacy Tokenizer V1	Spacy Tokenizer V1	Spacy Tokenizer V1	Spacy Tokenizer V1
Batch size	1000	1000	248	128
Learning Steps	1800	1800	2000	2200
Learning rate	0.001	0.001	0.001	5.10^{-5}
Optimizer	Adam	Adam	Adam	Adam
Size	13 MB	43 MB	741 MB	438 MB

Table 4: Configuration of the different Spacy Pipelines

Training Metrics During the training we analyzed several metrics, here are the results for the last training step:

Metric	Value
LOSS TOK2VEC	563.58
LOSS NER	604.32
ENTS_F	54.62
ENTS_P	58.62
ENTS_R	51.13
SCORE	0.77

(a) en_core_web_sm

Metric	Value
LOSS TOK2VEC	174.26
LOSS NER	501.05
ENTS_F	56.32
ENTS_P	57.42
ENTS_R	55.26
SCORE	0.56

(b) en_core_web_md

Table 5: Training metrics for *en_core_web_sm* and *en_core_web_md*

Metric	Value
LOSS TOK2VEC	277.14
LOSS NER	692.76
ENTS_F	54.01
ENTS_P	56.33
ENTS_R	51.88
SCORE	0.54

(a) en_core_web_lg

Metric	Value
LOSS TRANSFORMER	189.17
LOSS NER	877.07
ENTS_F	66.42
ENTS_P	64.87
ENTS_R	68.05
SCORE	0.66

(b) en_core_web_trf

Table 6: Training metrics for *en_core_web_lg* and *en_core_web_trf*

4.2 Results

To test the trained Pipeline, we used the `test_data` dataset and hid the IOB tags from the model. We run the corresponding command line `spacy evaluate` to get the resulting metrics.

Metric	Value
NER Precision	57.05
NER Recall	53.61
NER F1	55.28
Speed	24486

(a) en_core_web_sm

Metric	Value
NER Precision	57.23
NER Recall	54.82
NER F1	56.00
Speed	4611

(b) en_core_web_md

Figure 4: Testing Metrics for *en_core_web_sm* and *en_core_web_md*

Metric	Value
NER Precision	65.12
NER Recall	61.95
NER F1	63.49
Speed	4501

(a) en_core_web_lg

Metric	Value
NER Precision	63.40
NER Recall	66.27
NER F1	64.80
Speed	2116

(b) en_core_web_trf

Figure 5: Testing Metrics for *en_core_web_lg* and *en_core_web_trf*

Comparison of the different models After comparing the different models, we see that the fine-tuning of the Spacy Pipeline did not achieve good results. The F1 score is still low and does not outperform our baseline. The large model *en_core_web_lg* and the transformer version achieved some similar performances but the transformer version achieved faster results (using a GPU) than the simple large model.

5 BERT Model

In this section, the implementation of the BERT model is presented. We chose to implement this model in order to test the performance of a Deep Learning model in a `IOB` tagging task and compare it with models of a simpler architecture. The methodology used and the results acquired are analyzed below.

5.1 Methodology

Data pre-processing BERT works with full sentences, so we had to reconstruct them from the tokens of the dataset. We then obtained pairs such as `(sentence, seq_labels)`.

BERT Tokenizer and its sub-words We used the BERT Tokenizer, which tokenizes sentences into sub-words and not whole words. To face this problem we had to extend the annotation to each sub-word: eg. `phone`, `##me` has the corresponding IOB tags `B`, `B`.
We also needed to add special tokens for the BERT model: `CLS`, `SEP`. We additionally added padding with the `PAD` token to make the sentences have the same size.

Model and training procedure We used the pre-trained `BertForTokenClassification` base uncased model and instantiated it with our custom labels. The optimizer that we selected is the Adam optimizer. The hyper-parameters for the training are the following:

Parameter	Value
MAX_LEN	64
TRAIN_BATCH_SIZE	4
VALID_BATCH_SIZE	2
EPOCHS	1
LEARNING_RATE	1e-05
MAX_GRAD_NORM	10

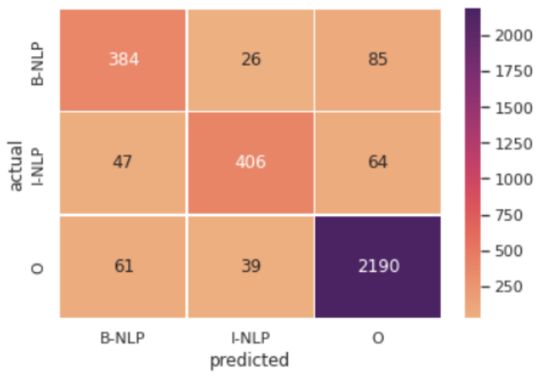
Table 7: Hyper-parameters

Metric	Value
Training accuracy	0.807
Validation accuracy	0.902

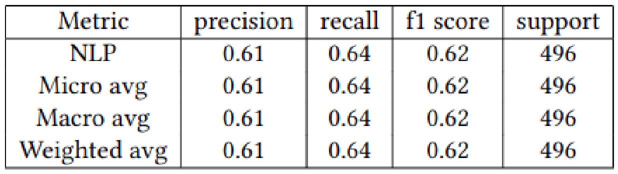
Table 8: Training Metrics

5.2 Results

The results of the best model are presented in the following table:



(a) BERT - confusion matrix



Metric	precision	recall	f1 score	support
NLP	0.61	0.64	0.62	496
Micro avg	0.61	0.64	0.62	496
Macro avg	0.61	0.64	0.62	496
Weighted avg	0.61	0.64	0.62	496

(b) BERT - report

Figure 6: Results of BERT

The overall F1 score for the NLP label is **0.62** for our best model. From the confusion matrix and the predicted/actual ratio, we see that the model performed better on detecting `O`, then on detecting `B`, and finally `I`.

5.3 Discussion

To train a deep-learning model like BERT we would need more data to reach a better accuracy. Here we only got a few data (see Table 1) which can explain the low accuracy.
Our approach to tag the sub-words can also be changed and we might achieve different results in a future implementation of the model. One different approach that could provide better results is tagging only the first word-piece of the sub-word instead of tagging all its word-pieces.

6 Conclusion

After testing a range of models, from simple CRF to deep learning models like BERT, we reached low scores on this IOB tagging task. Comparing the F1 scores of the 4 models that we implemented, we can conclude that the best performance attested is the one of the Conditional Random Fields. The aforementioned model performed significantly better than the Random Forest Classifier (68% contrary to 40% F1-macro score). In the first case, the `IOB` tags were better recognized by the tagger proving that this model is more efficient in predicting the most likely sequence of labels corresponding to a sequence of inputs. It also needs to be noted that the model is easy to train and doesn't require high computational power and time.

The next highest performance is the one of the spaCy model under the transformer configuration (64.80% F1 score). The other configurations led to a bit lower performances with the best among the rest three configurations tested being the large model en_core_web_lg. Spacy models are easy to train thanks to the CLI commands but their training time is usually longer compared to the other approaches and the Tranformers version needs a GPU in order to train it in a decent time.

The results achieved by the BERT model were below our expectations (62% F1 score). Those results are expected to get better if the dataset used is larger. We also need to note that BERT needs specific data pre-processing which makes its implementation more demanding in the case of handling inconsistent data.