Name: Danil Meshcherekov

E-mail: d.meshcherekov@innopolis.university

Codalab nickname: SSBsb3ZlIE5FUiA8Mw

Link to github repo.

Solution 1 - GLiNER model

The GLiNER model is a NER model that uses a BERT-like bidirectional transformer encoder. It has been pre-trained on Pile-NER and NuNER datasets and has shown good results in many languages and on various datasets [1].

The major advantage of this model is that it is generalist and can be used out-of-the-box for custom datasets and labels without any fine-tuning. Additionally, the model provides a numerical value of how "certain" it is about each extracted entity.

The major disadvantage is that it does not support nested entities. I have tried to solve this problem by calling the model on the full sentence to find all "top-level" entities, and then calling this model again on all found entities to extract all sub-entities recursively.

Therefore, the my solution with GLiNER model goes like this:

1. Initialize the model, set the "certainty" threshold.
2. Call the model on the entire sentence and extract all entities that the model has found with the given threshold.
3. Iterate over all extracted entities, treat them as "entire sentences" as in step 2 recursively, until some maximum recursion depth is reached.
4. Collect all entities from all sub-calls and filter out duplicates.

This approach solves the problem of label imbalance in the training set (because the model is trained to make predictions based on label name and not fine-tuning) and the problem of nested entities.

However, this approach has several limitations: it is slow because it uses an LLM to make inferences, and this solution to nested NERs is quite poor: in the sentence “Moscow Drama Theater named after M. N. Yermolova” it will only find two entities (“Moscow Drama Theater” and “M. N. Yermolova”) instead of all three.

The model has been fine-tuned on two parameters: maximum recursion depth and the threshold. These two parameters are crucial in my algorithm, because small depth and high threshold will ignore correct entities, and big depth and low threshold may incorrectly detect a lot of entities.

The F1-scores of fine-tuning are as follows:

Threshold	depth=0	depth=1	depth=2	depth=3
Thr = 0.3	39.66%	39.66%	39.66%	39.66%
Thr = 0.35	39.66%	39.66%	39.66%	39.66%
Thr = 0.4	39.66%	39.66%	39.66%	39.66%
Thr = 0.45	39.66%	39.66%	39.66%	39.66%
Thr = 0.5	39.66%	39.66%	39.66%	39.66%
Thr = 0.55	38.31%	38.31%	38.31%	38.31%
Thr = 0.6	37.23%	37.23%	37.23%	37.23%
Thr = 0.7	33.38%	33.38%	33.38%	33.38%
Thr = 0.8	26.37%	26.37%	26.37%	26.37%

As we can see, the model performed quite poorly due to the lack of fine-tuning.

Additionally, this model is uncertain about many words, so smaller threshold values provide in general better results. Also, it does not handle nested entities at all despite my recursive implementation of the entities extractor, so the results are the same for any depth.

Solution 2 - spaCy pipeline fine-tuning

In this approach I decided to use the classic spaCy Python library with the pre-trained Russian language model with some fine-tuning.

The advantages of this solution is that it works well with Russian, it is fast, and it is customizable for any specific task – even though it does not detect entities with custom labels out-of-the-box, it is possible to train it on a custom dataset, so that it can predict any label.

The major disadvantage for this assignment is that it also does not support nested entities. However, it is possible to address this problem by calling the spaCy model recursively on "top-level" results. Additionally, unlike in the previous approach, it is possible to train it on this scenario, so that it learns to select top-level entities first.

To implement this solution, I have additionally preprocessed the provided dataset so that all nested entities are in separate data points. For example, a sentence:

`"Ilya works in Moscow University"`

Would have 4 entities: `["Ilya", "Moscow University", "Moscow", "University"]`

But after preprocessing it is split into two separate sentences without nested entities:

`"Ilya works in Moscow University" -> ["Ilya", "Moscow University"]`

with top-level entities only, and

`"Moscow University" -> ["Moscow", "University"]`

With sub-entities only.

At inference time, the process is similar to that in Solution 1. Here's an overview of the entire process:

1. Preprocess the training dataset as described above
2. Initialize the model and train it on the preprocessed dataset
3. When making predictions, call the spaCy model on the entire sentence and extract all top-level entities

4. Treat each extracted entity as an “entire sentence” as in step 3 and call the model on them, until some maximum recursion depth is reached
5. Collect entities from all calls and filter the duplicates out.

This approach combines the power of spaCy NER models, allows us to make predictions with custom class labels, and also solves the problem of nested entities. However, due to label imbalance in the training set, this model may provide incorrect labels for the test data.

There is only one tunable parameter in my implementation – the maximum recursion depth. My motivation for fine-tuning the recursion depth is similar to that in the first solution - a small depth may overlook correct entities, and a big depth may mark too many entities incorrectly.

Here is the table of F1-score values for different depths:

	depth=0	depth=1	depth=2	depth=3	depth=4	depth=5	depth=6
F1-score	66.36%	70.73%	71.14%	71.21%	71.19%	71.19%	71.19%

The spaCy model performs much better compared to the GLiNER because it handles the Russian language better and it was fine-tuned to this particular type of task.

Additionally, this time different values of depth matter and we can see that too small or too big values of depth provide less correct results, and the ideal depth value is 3.

Submission results comparison

Since this competition allows multiple submissions and the scores are shown for each submission, I decided to submit the results of both models and different hyper-parameters and compare them against each other.

Here is the comparison table:

Model name	Threshold	Depth	F1-score of submission
GLiNER	50%	0	29%
GLiNER	30%	0	29%
GLiNER	70%	2	22%
GLiNER	60%	4	26%
spaCy (fine-tuned)	-	1	58%
spaCy (fine-tuned)	-	2	59%
spaCy (fine-tuned)	-	3	59%
spaCy (fine-tuned)	-	4	59%
spaCy (fine-tuned)	-	5	59%

Although the results on the test set are different from the validation set results above, they follow a similar distribution: GLiNER has the same results independently of the depth, lower thresholds produce better results, and it in general performs way worse than the spaCy model, having the F1-score at 29% at most. The results of spaCy are much better, but the depth does not make a lot of difference as well, although the best hyperparameter of depth=3 still holds.

The best solution out of these two that are used for the final submission on Codalab is the fine-tuned spaCy model with depth=3. I decided to choose this one because it has shown the best results (in both evaluation and test scores), and it takes much less time to run, as compared to the GLiNER model.

References

[1] - Zaratiana U. *et al.*, Generalist Model for Named Entity Recognition using Bidirectional Transformer, 2023, arXiv 2311.08526