

# Recognai's Working Notes for CANTEMIST-NER Track

David Carreto Fidalgo, Daniel Vila-Suero and Francisco Aranda Montes

## Abstract

These working notes describe the two Named Entity Recognition (NER) systems designed by Team Recognai for the CANTEMIST (CANcer Text Mining Shared Task – tumor named entity recognition) NER track.

While the first system tries to maximise the performance with respect to the F1-score, the second system tries to maximise its efficiency with respect to model size and speed while maintaining acceptable performance.

## 1. Introduction

To better understand diseases and to improve clinical decision-making, Natural Language Processing (NLP) can help to use the information from literature and digital health records. The main objective of the CANTEMIST-NER track was to extract certain key entities like diseases, treatments or symptoms from clinical documents.

Our contribution consists of two Named Entity Recognition (NER) systems that are based on Deep Neural Network architectures.

While both systems display a big difference in model size and speed, conceptually they are very similar:

- Both have an embedding layer that returns an encoded representation of each word;
- Both systems contextualize their embeddings by means of a Long Short-Term Memory (LSTM) layer;
- Both systems pass on the hidden states of their LSTM layer to the token classification head that decides if the word forms part of an entity.

The biggest difference between both systems lies in the respective embedding layer. Both systems were designed and trained using our open source *biome.text* library<sup>1</sup>.

---


*Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2020)*

EMAIL: david@recogn.ai; daniel@recogn.ai; francisco@recogn.ai

URL: <https://www.recogn.ai>



© 2020 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

## 2. Data preprocessing

Our data preprocessing was minimal and consisted of two major steps:

1. As a first step we transformed the given *brat* annotations<sup>2</sup> of the *train*, *dev1* and *dev2* data sets to commonly used *BLOU* tags[1]. For this we used *spaCy*<sup>3</sup> and a customised tokenizer from *spaCy*'s "es" language model.
2. After the tokenization and the transformation to *BLOU* tags, we used *spaCy*'s sentence splitter to divide the *train*, *dev1* and *dev2* data into sentences. Our two systems were trained and evaluated based on sentences, hence the maximum context our models can take into account is on a sentence level.

We used the same *spaCy* sentence splitter for splitting the test and background data into sentences and feed them to our models to obtain the submitted predictions.

No data augmentation or external data was used for the training of our systems.

## 3. XLM-R System

The goal of this system was to maximise its performance with respect to the F1-score.

### 3.1. Architecture

To achieve the goal of maximal performance, we use a pretrained transformer-based masked language model provided by the Huggingface Transformers library[2].

The *XLM-RoBERTa*[3] (XLM-R) model, trained on one hundred languages and outperforming multilingual BERT on NER, seemed to be the most appropriate choice for our task. We opted for its *xlm-roberta-base* implementation by Huggingface, after verifying that the bigger *xlm-roberta-large* variant yielded no significant improvement.

We introduce this model in our embedding layer to obtain contextualized word embeddings. Since XLM-R applies subword tokenization, we simply sum up the subword vectors when necessary to end up with embeddings at word level.

To facilitate the model's identification of words that are likely entities regardless their context (like technical names of cancer types), we extend our embeddings with character features. Another reason to add character features in general, is to make the model more robust against typographical errors.

The character feature consists of the last hidden outputs of a bidirectional Gated Recurrent Unit (GRU) that is fed with the characters of the respective word.

The stacked embeddings are then passed on to the bidirectional LSTM layer in which we seek after a task specific contextualization of our embeddings.

The hidden states of the LSTM layer are finally fed into a token classification head. This head consists of a linear transformation of the hidden dimension of the LSTM layer to the number of

---

<sup>1</sup><https://www.recogn.ai/biome-text>

<sup>2</sup><http://brat.nlplab.org>

<sup>3</sup><https://spacy.io>

**Table 1**

Summary of our XLM-R system. The size refers to the number of parameters of the component.

Component	Size
<b>Embedding layer</b>	
XLM-R feature (pretrained transformer, output size 768)	280M
Char feature (bidirectional GRU, output size 256)	160k
<b>Encoder</b>	
Bidirectional LSTM (1 layer, hidden size $2 \times 128$ )	1M
<b>Token classification head</b>	
Linear	1k
CRF	100

possible BIOES tags, and a subsequent Conditional Random Field (CRF) model that predicts the sequence of BIOES tags for the input.

The single components of the system and their approximate sizes in terms of number of parameters are summarised in Table 1.

### 3.2. Training

Experimenting and optimisation of some hyperparameters were done with the train and dev1 data set. We used the *AdamW* algorithm implemented in Pytorch for optimising all parameters of our model with a learning rate of  $10^{-5}$ .

For an estimation of the model performance we used the train and dev1 data set as training data and the dev2 set as validation data. For the final submitted predictions we trained our model on the combined set of train, dev1 and dev2. We stopped the final training after 10 epochs, a number estimated from previous training runs, to prevent overfitting.

## 4. FastText System

The goal of this system was to maximise the model’s efficiency with respect to the model size and speed while maintaining acceptable performance.

### 4.1. Architecture

In contrast to our XLM-R system, we opted for a more light-weight solution regarding the pretrained component. For this reason we chose the pretrained Spanish word vectors provided by FastText[4]. These vectors encompass 2 million words that were trained on Common Crawl<sup>4</sup>

<sup>4</sup><https://commoncrawl.org/>

**Table 2**

Summary of our Fasttext system. The size refers to the number of parameters of the component.

Component	Size
<b>Embedding layer</b>	
Word feature (pretrained vectors from FastText, output size 300)	4M
Char feature (bidirectional GRU, output size 256)	160k
<b>Encoder</b>	
Bidirectional LSTM (1 layer, hidden size $2 \times 512$ )	4M
<b>Token classification head</b>	
Linear	5k
CRF	100

and Wikipedia with an embedding dimension of 300.

To assure a well generalised word vocabulary, we only add words to it that appear at least 2 times in our training data set.

Furthermore we add character features to our embeddings to make our model more robust against typos. The character feature consists of the last hidden outputs of a bidirectional GRU that is fed with the characters of the respective word.

The stacked embeddings are then passed on to the bidirectional LSTM layer in which we seek after the contextualization of our embeddings.

The hidden states of the LSTM layer are finally fed into a token classification head. This head consists of a linear transformation of the hidden dimension of the LSTM layer to the number of possible BILOU tags, and a subsequent CRF model that predicts the sequence of BILOU tags for the input.

The single components of the system and there approximate sizes in terms of number of parameters are summarised in Table 2.

## 4.2. Training

Experimenting and optimisation of hyperparameters were done with the train and dev1 data set and performing Hyperparameter Optimization of both architecture parameters (e.g., encoder hidden sizes) and training hyperparameters (e.g., learning rate). For hyperparameter optimization, we used the integration of *biome.text* with the *Ray Tune* library[5] to perform random hyperparameter search with the *ASHA* trial scheduler[6]. We used the *AdamW*[7] algorithm implemented in Pytorch for optimising all parameters of our model with a learning rate of  $3.9 \times 10^{-3}$ .

For an estimation of the model performance we used the train and dev1 data set as training data and the dev2 set as validation data. For the final submitted predictions we trained our

**Table 3**

Evaluation results by system for the CANTEMIST-NER track. The prediction time refers to the real time spent on predicting the entire test data set with an i7-9750H CPU with 6 cores.

	Precision	Recall	F-Measure	Prediction time
XLM-R System	0,85	0,84	0,845	3 h
FastText System	0,846	0,844	0,845	0.3 h

model on the combined set of train, dev1 and dev2. We stopped the final training after 4 epochs, a number estimated from previous training runs, to prevent overfitting.

## 5. Results

Table 3 presents the results provided by the CANTEMIST organizers for both systems on the test set. It seems the XLMR model was not able to take advantage of the pretrained language model together with its additional parameters compared to the FastText model. We suspect that the unusual language used in medical reports makes the pretrained language model unsuitable for this data set. A possible solution, which was not pursued due to time and resource constraints, could be to fine tune the language model first on the entire train and test set, and then to fine tune the NER system. Finally, another potential improvement to the current approach would be including other features such as gazetteers or linguistic features (e.g., part of speech tags) into the end-to-end neural network model.

## Acknowledgments

This work was supported by the Spanish *Ministerio de Ciencia, Innovación y Universidades* through its *Ayuda para contratos Torres Quevedo 2018* program with the reference number *PTQ2018-009909*.

## References

- [1] L. Ratinov, D. Roth, Design challenges and misconceptions in named entity recognition, in: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009), Association for Computational Linguistics, Boulder, Colorado, 2009, pp. 147–155. URL: <https://www.aclweb.org/anthology/W09-1119>.
- [2] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, HuggingFace’s Transformers: State-of-the-art Natural Language Processing, arXiv e-prints (2019) arXiv:1910.03771. arXiv: 1910.03771.
- [3] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, V. Stoyanov, Unsupervised Cross-lingual Representation Learning at Scale, arXiv e-prints (2019) arXiv:1911.02116. arXiv: 1911.02116.

- [4] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, T. Mikolov, Learning Word Vectors for 157 Languages, arXiv e-prints (2018) arXiv:1802.06893. arXiv:1802.06893.
- [5] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, I. Stoica, Tune: A Research Platform for Distributed Model Selection and Training, arXiv e-prints (2018) arXiv:1807.05118. arXiv:1807.05118.
- [6] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, A. Talwalkar, A System for Massively Parallel Hyperparameter Tuning, arXiv e-prints (2018) arXiv:1810.05934. arXiv:1810.05934.
- [7] I. Loshchilov, F. Hutter, Decoupled Weight Decay Regularization, arXiv e-prints (2017) arXiv:1711.05101. arXiv:1711.05101.

## A. Online Resources

A GitHub repository was created at <https://github.com/recognai/cantemist-ner> that contains the data sets as well as the data preparation, training and evaluation notebooks.