Evaluation of Named Entity Recognition in Latin using an Unsupervised Model

Gabriel Cristian Circiu
IT University of Copenhagen
Copenhagen, Denmark
gaci@itu.dk

Mykyta Taranov
IT University of Copenhagen
Copenhagen, Denmark
myta@itu.dk

Wenzel Keil
IT University of Copenhagen
Copenhagen, Denmark
weke@itu.dk

Abstract

Tackling the challenges of Natural Language Processing (NLP) in Latin using an unsupervied model has lead to the observation that certain tokenizers such as BERT are better suited for such tasks and that the choice of the model itself plays a significant role in the final results. Clustering has proved to be the most straightforward method to group entities together but a more intricate pipeline lead to a significantly better result, achieving an averaged macro F1 score of 0,51.

1 Introduction

As international students, the probability of sharing a common language outside of English is not high, however, during our studies we came to the realization that some of usshare knowledge in one such language, Latin. Albeit we are nowhere near fluent, it has sparked curiosity to work with it. We chose to head in the direction of our shared curiosity of Latin, and to poke at it within the Named Entity Recognition (NER) sphere.

As such we chose to build upon a past research, titled *Challenges and Solutions for Latin Named Entity Recognition*[1] (Erdmann et al., 2016), which tackles the problem of NER in Latin using a supervised and semi-supervised model. Our dataset is based on the one used in the original paper, following the same structure, with some additional specifications for clarity.

2 Dataset

The dataset from the original paper named 3 works of literature, Caesar's *De Bello Gallico (BG)*, Pliny the Younger's *Epistulae (EP)*, and Ovid's *Ars Amatoria (AA)*. We have had the great fortune of receiving the full dataset in its entirety, as large text files, in IOB format. While all the literature was scrambled in each file, having it in the right format has made it all the more easier to process, and do some Exploratory Data Analysis (EDA). After thorough investigation, we have found that there is a slight inconsistency in the original paper regarding the dataset, and that of which we have received. As mentioned, the entirety of *BG*, and parts of *EP* and *AA* were annotated, however, parts of Caesar's *De Bello Civili (BC)* were also annotated within the dataset, but left unmentioned in the original paper.

To have an initial understanding of the expectations of our models, we must understand the distribution of the dataset, namely the amount of *Named entities (NE)* and *Non-Named entities (Non NE)* in the dataset. As shown in Figure 1 and Figure 2, we can see that there is a significant imbalance in the dataset, with only about 5.5% and 3.3% of the tokens being NE.

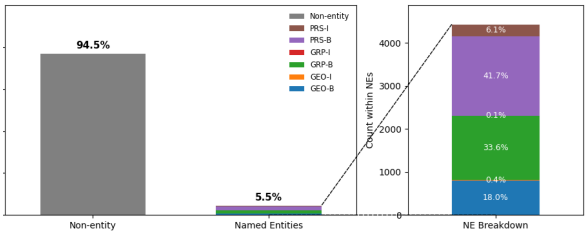

Figure 1: Distribution of data for *BG*, *BC*, and *AA* combined, used for one of the training folds.

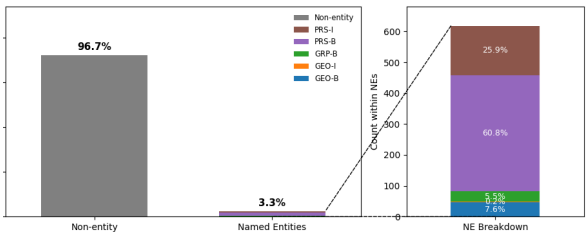

Figure 2: Distribution of data for *EP*, used for one of the testing folds.

[1]Original paper: https://aclanthology.org/W16-4012/

3 Methodology

Once we had the dataset split into training and testing files, we started with tokenizing the words, and finding a solutions on how to handle the words, embeddings, and labels, without having supervision over the original labels that we were given. Afterwards we have decided on multiple approaches in training a model, namely Clustering, and a Neural Network.

3.1 Embeddings

Our initial approach was to vectorize tokens using a basic `Word2Vec` method from the `Gensim` library. However, this produced unacceptably low results with clustering — the model wasn't able to separate entities from non-entities in a meaningful way. So we moved on to a more powerful method: using contextual word embeddings from a `pretrained Bidirectional encoder representations from transformers (BERT)` model, specifically `bert-base-multilingual-cased`. This model was trained on a large corpus of multilingual text, including Latin, and is capable of generating high-quality embeddings for words in context.

We used the `Hugging Face Transformers` library to load the model and generate embeddings for our tokens. We extacted the embeddings from the last hidden layer of the model. Since `BERT` works on subword level, we combined the embeddings of the subwords that make up a token by averaging subwords embeddings. This approach has been shown to work well in practice and allows us to obtain a single embedding for each token. We discusse the options of using subwords embeddings for clustering and switching to the word level later on, but this led to unnecessary complications of the pipeline, for example, by introducing the necessaty of handling scenarios when embeddings of one word end up in different clusters.

3.2 Clustering

Once we had the word level embeddings, we applied `KMeans Clustering` to automatically group the tokens into clusters. `Kmeans` was the most straightforward choice for the task. One of the challenges we faced was the choice of `K` - number of clusters. We tried several methods to determine the optimal number of clusters, including the *Elbow Method* and *Silhouette Score*. The elbow method suggested that big values of `K` are better, up to the point that number of clusters was close to the number of tokens. Silhouette score also was not very helpful, since bigger score didn't correspond to better evaluation scores afterwards. So we boiled down the problem to the simplest approach: trial and error. First we started with a binary classification, where we had two clusters: one for NE and one for Non NE. For this we settled on a value of `K` = 7.500 where we then passed the clusters classified as NE to another kmeans clustering, where we had 207 clusters for the different types of NE.

We used the KMeans implementation from the `scikit-learn` library, which provides a simple and efficient way to perform KMeans clustering. After clustering we named each cluster by taking the most frequent token in the cluster. We had a short discussion on whether this makes the model semi-supervised, but concluded that it doesn't: the label data is only used after clustering, and doesn't influence how the clusters are formed. Then we evaluated the clusters as a baseline for our model. We used the macro F1 score to measure the perfomance, and got a score of 0,43.

3.3 Neural Network

Next we reconstructed the model using a `Neural Network` (NN). We used a lightweight NN consisting of two linear layers with `GELU Activation` and `LayerNorm`, mapping contextual BERT embeddings to entity class logits. The network is trained using pseudo-labels derived from the initial clustering, which makes the inherently supervised neural network **unsupervised**.

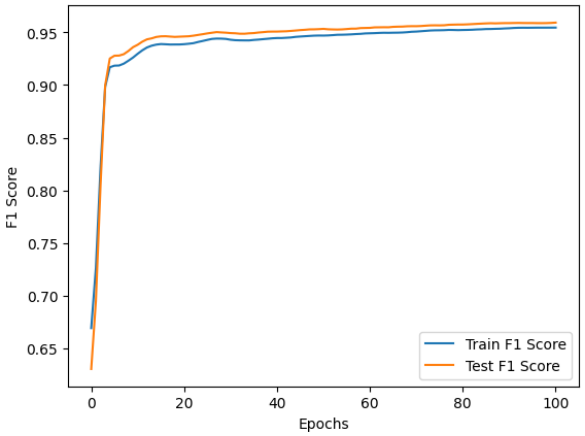

Figure 3: Rate of change of weighted average F1 score with increasing number of epochs.

We used `Pytorch` to implement and train the model. The choice of the number of epochs was based on the rate of change of the F1 score, which

we observed in most cases to not significantly increase after 10 epochs, and plateau after 30 epochs. As shown in Figure 3, which is what we chose.

4 Results

We have stayed true to the original paper and used the same train and test sets, indicated in Table 1, and have used the same evaluation metrics, indicated in Table 2. For the sake of consistency, we have included *BC* in the train set, and all 8 books of *BG* in the training set, even though it was not part of the original report.

	Test Set	**In or Out-of-domain**
Fold 1	Piny	Out
Fold 2	Ovid	Out
Fold 3	Caesar	In

Table 1: Train and Test Sets for each fold

To have a detailed overview of the results, we have included a classification report for each fold and each model in Table 2, which includes the precision, recall, F1 score, and weighted average F1 score for each fold and each model, namely Binary Clustering, Multi-Class Clustering, and Neural Network. While we understand that the standard score for NER is the Span F1 score, we have used the normal F1 score for the sake of consistency with the original paper, as there is no mention of specificity of the type of F1 score used, and we also understand that on a technical level we could not use the Span F1 score, as our model by design provides lower performance than supervised models, so it was not in our final goal to achieve high scores measured by a more strict metric like Span F1.

	Prec.	**Rec.**	**M. F1**	**W. F1**
Binary Clustering				
Fold 1	0.90	0.91	0.91	0.99
Fold 2	0.86	0.78	0.82	0.98
Fold 3	0.98	0.98	0.98	1.00
Multi-Class Clustering				
Fold 1	0.41	0.43	0.41	0.98
Fold 2	0.38	0.34	0.35	0.97
Fold 3	0.56	0.53	0.54	0.99
Neural Network				
Fold 1	0.33	0.52	0.39	0.96
Fold 2	0.29	0.42	0.32	0.95
Fold 3	0.48	0.60	0.53	0.97

Table 2: Classification report

These results suggest that binary clustering performs extraordinarily well, highly exceeding our expectations, however once we dive deeper into being able to distinguish between different named entitities, the results also dive quite deep into the lower end of resulting scores. Additionally, the Neural Network does not seem to provide substainal, if any at all, improvement over the multi-class clustering, however, we must note that the purpose of the Neural Network is to generalise on unseen data, and not to specifically outperform the multi-class clustering.

There are a few key points worthy of mention, that may not be initially obvious. Fold 3 represents the In Domain datasets, which have higher chance of performing better, as per our observation, the training and testing datasets mention the same entities more often, resulting in a better fit. Finally, some entities are better detected, which the results table also does not show. The entities that we found to be better detected, were GEO-B, GRP-B, PRS-B, with F1 scores on average of around 0.6 to 0.8 for each entity.

5 Future Work

As our results are not very promising, it gives us a glimpse into the mysteries of the Latin language, and the challenges of NER within it. We plan to explore some aspects of this field as there is a lot of room for improvement, and a lot more to discover.

To start, we would like to explore having a much larger dataset to work with, and observe how results improve with the proportion of data. Next, we would like to explore different capitalizations of the language, and observe how results change, as we suspect that the capitalization played a significant role in the performance of the model. Finally, considering an ensemble method of different models may prove to be the most effective approach, as it has been observed so far, and we would like to explore the limits of this approach and benefits of it.

Limitations

Due to the size of the dataset and the complexity of the task, we were not able to train a more capable model. A similarly comparable evaluation of NER using an unsupervised model within the English language has been shown to achieve much better results, indicating that the size and distribution of the dataset is a significant factor in the performance of the model, even more so than the choice of model.

Having a discrepancy between the original dataset wordcount and what we have received, based ont the original splits, having to manually split the dataset has made our work substantially longer than expected, as we had to manually separate the dataset up into new files. Using the help of the original source of the dataset, we managed to find and split it appropriately, at a significant cost of time.

Due to the limitations of time, we were not able to explore more methods of approach and tinker extensively with the hyperparameters of the model.

A Appendix

Group contributions are as follows:

- Mykyta: Initial EDA to identify distribution of NE's and non NE's, observations on inconsistencies in the dataset, implementation of binary pipeline and of the Neural Network.
- Wenzel: Implementation of the Multi-Class Clustering method, multi-class classification, training the models, and reporting the results.
- Gabriel: Collaborating with previous papers authors, obtaining the dataset, EDA, splitting up dataset into train and test sets, implementing initial Word2Vec and Tokenization methods.

We feel these contributions are fair and equal, and reflect the work done by each of us.

We have used chatbots, in the form of spellcheckers after writing our paper to make sure we are formulating our sentences in an appropriate manner, and autocompletion in our code to speed up the development process.

All of our code is available on GitHub[2]. As the dataset shared was not disclosed to be public, it is left out of the repository.

[2]Github Repository: https://github.com/GabrielCirciu/Latin-NER-NLP

References

Alexander Erdmann, Christopher Brown, Brian Joseph, Mark Janse, Petra Ajaka, Micha Elsner, and Marie-Catherine de Marneffe. 2016. Challenges and solutions for Latin named entity recognition. In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pages 85–93, Osaka, Japan. The COLING 2016 Organizing Committee.