Web Mining IE684

Elina Tugaeva, Eliana Ruslanova, Maida Nazifi, Nikolaj Kolbasko, Samail Guliev

Faculty of Business Informatics and Mathematics
University of Mannheim
May, 2020

Abstract

This document contains the description of the project implementation for the course Web Mining IE684 FSS2020. We introduce various algorithms for Web Content Mining task - namely, Named Entity Recognition. We start by justifying the importance of this task in the current situation of growing requirements for businesses according to GDPR, proceed with short introduction and analysis of data set acquired and then discuss in detail all implemented algorithms. The ultimate goal of this project is to obtain practical and domain-independent techniques in order to detect named entities with high accuracy.

In the course of reading one can regard self-tuned BERT and FNNs with combination of LSTM layers with different common algorithms for Named Entity Recognition, such as CRF, CNN and RNN, to be most suitable for obtaining high performance in the task.

Our experiments and analyses raised the idea that also in Named Entity Recognition tasks there may exist an unavoidable trade-off between the running time of the algorithm (speed) and performance level this algorithm exhibits.

The accompanying Jupiter Notebooks can be used as supplementary material to assess and reproduce any results mentioned in the text.

1 Introduction

NER, which stands for Named-Entity-Recognition, is a process in natural language processing (NLP) to split a sentence into meaningful categories/pieces like names or organizations. It allows receiving information on which words represented an entity and which kind of entity.

A system which enables the accurate prediction of entities has different capabilities in science and practice. In our case, we got acquainted with a relatively new problem in practice: Data anonymization due to The European Union's General Data Protection Regulation (GDPR).

If companies violate GDPR, for example, by transferring data for machine learning without anonymization, the penalty fine can be up to 20 million or 4% of the company's annual revenue. Companies are facing a conflict: on the one hand, they are open to the usage of technologies of the industry 4.0, like machine learning, on the other hand, they fear that the incorrect treatment of confidential data might cause irreparable financial and reputational damage.

This leads to our decision that we want to work on a practical problem to build a stable named-entity-recognition using different methods to be able to anonymize data. The performance of our algorithms will be compared with the ready-to-use NER of spaCy.

2 Data Description

The dataset we proceeded with is obtained from Kaggle (Walia). It is of a B-I-O-annotation scheme (Beginning-Inside-Outside Encoding) is the extract from a wider Groningen Meaning Bank corpus which is tagged, annotated and built specifically to train the classifier to predict named entities, namely Geographical Entities, Organizations, Persons, Geopolitical Entities, Time indicators, Artifacts, Events, and Natural Phenomenon. Overall, this data set includes 47,959 sentences, consisting of 1 048 575 English words, which allows us to train the models with additionally engineered features and test the model appropriately.

In our case, the specific data set turns up being considerably neat and cleaned up, without any missing values, so we are ready to jump into the brief analysis of the data distribution. As for the shape of that, one can note that data is heavily skewed: 85 percent of examples belong to the negative class. It is crucial, for example, for the evaluation of results in a sense that usual accuracy metric is not enough.

We will closely look at that in the relevant section. Tokens[1] per sentence seems like a bell-shaped distribution, which was expected due to central limit theorem. The mean seems to be around 20 words per sentence. Another interesting thing to be noted is that among word classified as a named entity the leader was "B-per", which stands for the beginning of the mention of the person. Quite naturally among those were titles ("Mr" and "Ms") and positions ("President", "Prime (minister)", etc).

3 Methodology

In this paper, we provide the implementation of different supervised approaches to NER. Namely, we experiment with Conditional Random Fields (CRF), Recurrent Neural Networks (RNN), BERT and spaCy libraries.

It is known that CRFs require manual feature engineering that may be time-consuming. However, Annotated Corpus for Named Entity Recognition that we use has an additional dataset of 19 linguistic features such as word, lemma, POS-tag and capitalization. This allowed us to see which of those features have a greater impact on NER.

The advantage of RNN over the CRF is that there is no need in feature engineering. Moreover, RNNs take as their input, not just the current input example they see, but also what they have perceived previously in time. That makes them a good approach for the sequence labelling, in general, and for NER, in particular. There are different variations of RNNs but we will focus mainly on Long short-term Memory (LSTM) approach. We are going to use bi-directional LSTMs because using a standard LSTM to make predictions will only take the "past" information in a sequence of the text into account. For NER, since the context covers past and future labels in a sequence, we need to take both the past and the future information into account.

Next thing we tried out and analysed is a quite popular tool nowadays - BERT. Bidirectional Encoder Representations from Transformers allows the selection of an off-the-shelf model that has been trained on the task of "language modelling" (predicting which words belong in a sentence), then "fine-tuning" the model with data from your specific task rather than the model from scratch. BERT was pre-trained by Google AI Language using only a plain text corpus applying the bidirectional training similar to the LSTM approach introduced above.

Last but not least, we are going to use the spaCy library on our dataset. SpaCy features an extremely fast statistical entity recognition system, that assigns labels to contiguous spans of tokens. It was trained on the OntoNotes 5 corpus which supports a different annotation from the one that supports our dataset. However, spaCy's models are flexible and allow to add new entities and new examples for training.

3.1 CRF

Let us start with the classical and fairly straightforward algorithm used for NER - Conditional Random Fields (CRF). It is a class of discriminative models best suited to prediction tasks where contextual information or state of the neighbours affect the current prediction.

In order to implement prediction, we use Feature Functions - the set of rules engineered by the researcher, based on which labels are to be assigned. Since these Feature Functions are not explicitly stated, the researcher has enough freedom to include area-specific rules to increase the performance of the model when using in highly expert domains. Let us include some notations: we denote by $x = (x_1, .., x_m)$ the input sequence and by $s = (s_1, .., s_m)$ the sequence of resulting states. By feature functions we model the joint distribution $\Phi = \Phi(x_1, .., x_m, s_1, .., s_m)$. For the multi-classification task we will use general log-linear model, under the equation 1:

$$p(s_1, .., s_m | x_1, .., x_m, w) = \frac{exp(w\Phi(x, s))}{\sum exp(w\Phi(x, s))} \quad (1)$$

In order to obtain the weights as a learning process we need to optimize log-likelihood function 2:

$$L = \sum_{i=1}^{n} p(s^i | x^i, w) \quad (2)$$

We are going to proceed with features that empirically work well on the NER tasks. Namely, regarding each separate word and preceding word in the sentence (if exists, otherwise "beginning of the sentence" feature) there are binary features:

1. all case-based characters are lowercased
2. ending of a word

[1] Throughout following code implementations we use SentenceGetter function(Tobias), which allows gathering all the words in the triples (word, POS tag, NER tag)

3. word is in caps
4. word starts with an upper case letter
5. all the characters in the word are digits
6. postag
7. end of the sentence

Next step is to use these features and initialize the algorithm. We use a fairly straightforward CRF implementation provided by sklearn-crfsuite. The model evaluation will be discussed in the corresponding section.

3.2 RNN

Deep Learning models have been known for yielding good results, especially in Natural Language Processing (NLP) problems so that is why we decided also to use Recurrent Neural Networks and gather the results. Recurrent Neural networks are essentially a class of artificial neural networks whose connections form directed graphs. Another advantage of RNN is that there are tools that aid in avoiding the vanishing gradient problem, which is the direct result of the feature that makes RNNs advantageous, the interdependence of the inputs of former layers and outputs of the layers further away in time. Most used types of Recurrent Neural Networks with such type of memory are GRU (Gated Recurrent Units) and LSTM (Long Short-Term Memory). We will focus on the latter.

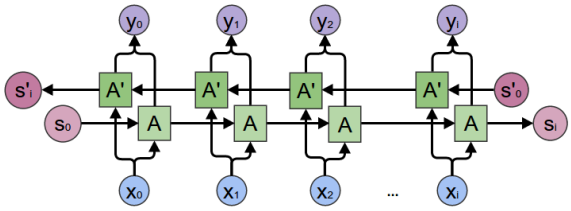

Figure 1: Bidirectional RNN (Lee)

LSTM approach is mainly used to filter the unwanted information and to keep only the important features or necessary information. To be more specific, in doing so, we can efficiently make use of past features (via forward states) and future features (via backward states) for a specific time frame.

LSTM cells will be used in a Bidirectional RNN model. Simply put, a bidirectional RNN is putting 2 separate RNNs together, one serving as the feed-forward and the other as the feed-backward layer, through which information is collected from the past and the future simultaneously. The advantage of this method is that the model has both backward and forward information about sequences at every timestep, meaning that the model remembers the past to be able to make better predictions.

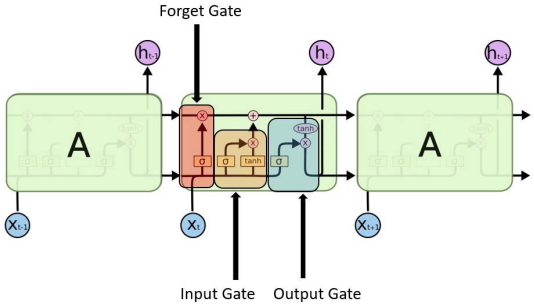

Figure 2: Visual Representation of LSTM (Mittal)

In our model we use *Sequential* from `keras` library, which allows us to group a linear stack of layers. The layers used are almost the same in both SimpleRNN and LSTM models. The Embedding layer, the Dense layer where the activation function is defined which is 'tanh' in these models. And the Dropout Layer which randomly sets input units to 0 with a frequency of 0.5 at each step during training time, which helps prevent overfitting.

3.3 LSTM-CRF

Now let us allow our simple CRF model to use some memorization through the usage of bidirectional Long Short-Term Memory algorithm.

The architecture is as the following: before the CRF layer discussed above, we add word-based BI-LSTM layer to produce vector representation for words in the corpus in both directions (i.e; forward and backward). This is the score in the nature that is defined as the sum of transitions and emissions from the bidirectional LSTM (BiLSTM):

$$score(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{n} (A_{y_i, y_i+1} + F_{\mathbf{x}, y_i}) \qquad (3)$$

where A is transition matrix between output states, where A_{y_i,y_i+1} is transition parameter between states y_i and y_{i+1} and F is emission matrix representing label y_i at i-th position. These scores are provided by the parameterized LSTM networks(Hochreiter and Schmidhuber, 1997). During training, we minimize the negative log-likelihood to obtain the model parameters including both LSTM and transition parameters.

A simple case of CRF built on top of a BI-LSTM effectively models several hard constraints on the

output to ensure they are valid by incorporating dependencies across the output labels. The very common example of such constraint would be -"The label of the first word in a sentence should start with "B-" or "O", not "I-"", or "The first label of one named entity should start with "B-" not "I-".

The resulting network can efficiently use past input features via a LSTM layer and sentence level tag information via a CRF layer, meaning that CRF layer is basically an optimisation on top of BI-LSTM layer. It can be used to efficiently predict the current tag based on the past attributed tags.

3.4 LSTM and character embeddings

We have discussed the methods that are trained on word embeddings, however, the NLP systems can be trained on additional information gained from inputs (words). Santos and Guimaraes (2015) employed not only word-level representations in their NER system but also character-level representations which can extract morphological information about the word (e.g. suffix, prefix, etc). Moreover, these character embeddings were proved to have a great influence on POS-tagging systems performance (Santos and Zadrozny, 2014). In this section, we are going to discuss two models with char embeddings that were implemented in our work.

For both models, we map each character in the word to unique numerical ID's so that each unique character in the vocabulary is represented by a particular integer ID. Then, words are padded with a number of zeros depending on max size of word vector. After these steps, we create character embeddings and fed them into a chosen layer.

First, we employ a model with a simple LSTM layer for character embeddings, then, the output of this layer is concatenated with the word embeddings and the second bi-LSTM layer is trained on the combination of character and word embeddings.

Second, we implement another deep learning method for character embeddings, namely, Convolutional Neural Networks (CNNs) that have shown state-of-the-art performance for extracting character-level features in POS-tagging (Labeau et al., 2015) and NER (Santos and Guimaraes, 2015).

The architecture of the CNN employed in our project can be seen in Figure 3. After a basic preprocessing of characters, we apply a convolution layer that generates a feature for each character and

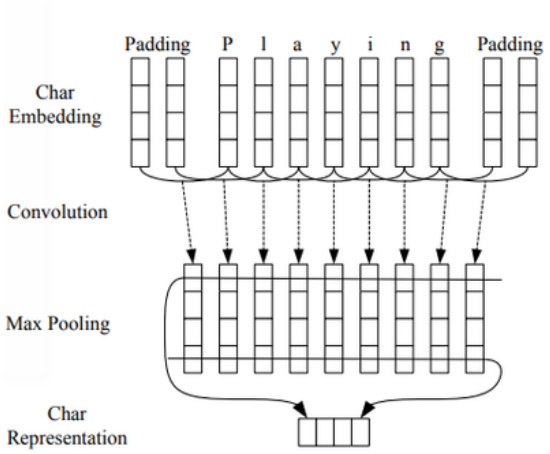

Figure 3: The CNN for extracting character-level representations. Source: (Ma and Hovy, 2016)

use a max pooling layer to extract only meaningful features out of our convolution layer. These character features are then concatenated with the word representations and are fed into the same LSTM layer that was already discussed above. Finally, we use a softmax layer which maps the non-normalized output of our network to a probability distribution over predicted output classes. We will further recall this model as LSTM-CNN model.

For our model training, we use a sparse categorical crossentropy for the loss function since we have multi-labelling task and each input belongs to exactly one class. This loss function saves computational time because it does not perform the summation over all classes as a simple categorical crossentropy. For optimizer, we choose Nadam optimizer since it shows good results on large datasets, consumes less computational power and, more importantly, is fast.

3.5 LSTM-CNN-CRF

In papers (Labeau et al., 2015), (Santos and Guimaraes, 2015) LSTM-CNN models showed good results in NER, however, the word's label in NER task is heavily dependent on the labels of its neighbors. We already discussed that in this case CRF models can be a great help. For example, it can learn that "I-ORG" cannot directly follow "I-PER". Therefore, in addition to our simple CRF and LSTM-CRF models, we are going to employ two LSTM-CNN-CRF models. One with CNN layer for character embeddings, another - with LSTM layer. *Ma and Hovy (2016)* showed that this model can outperform mostly all deep

learning algorithms and learn the dependencies between neighbouring labels really well (Ma and Hovy, 2016).

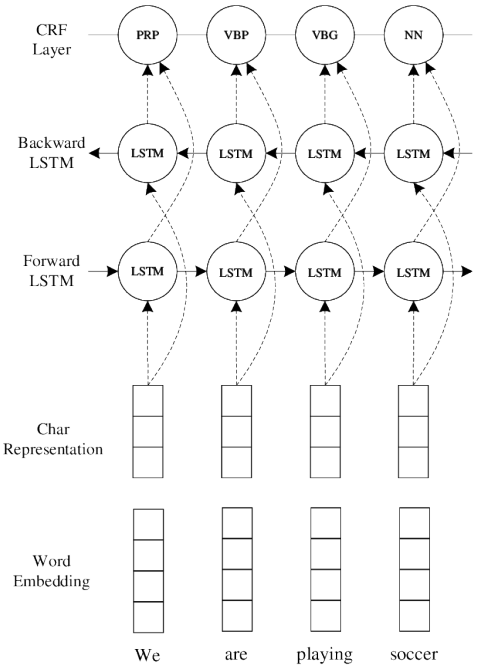

Figure 4: LSTM-CNN-CRF model architecture. Source: (Ma and Hovy, 2016)

These models have a general architecture which is illustrated in Figure 4 and is much alike the LSTM-CNN model described above. It learns a char representations with the use of CNN/LSTM layers (the model with the LSTM layer will be denoted as LSTM-LSTM-CNN in our work), feed them with word embeddings into a bi-LSTM layer and, on top of that, apply a CRF layer.

One important difference between LSTM-CNN and LSTM-CNN-CRF models is that now we cannot use a softmax layer because, in this case, we assume that the likelihood of the labels is conditionally independent given the RNN states. We do not require this assumption anymore since we are going to learn the model the dependencies between labels. Lastly, as we already mentioned before, we are using the log-likelihood function as the loss function while using a CRF.

3.6 BERT

In the BERT paper (Devlin et al., 2019) it was proven that BERT changed the landscape for tasks related to natural language processing. One of these tasks was named entity recognition where an overall F1 score of 92.4 was achieved by using the base BERT model. The main libraries we used were keras (Chollet et al., 2015), torch (Paszke et al., 2019) and transformers (Wolf et al., 2019).

The very first step to start working with the dataset is the preprocessing step. We need to extract the most important information from the dataset which are the words and the tags but not the POS as we do not work with it. Therefore, we create two lists, the first one contains the sentences which are split so that you can access every word, the other one contains the tags.

In the next step, we set up our BERT tokenizer. Therefore, we use the BertTokenizer from transformers to load the tokenizer for the basic BERT model, which was trained on cased texts. This model is used to tokenize our input sentences because we need to prepare a structure which is typical for BERT, which is also sometimes called ”Word Piece Embedding”.

Those prepared sentence tokens and the corresponding labels are fed into ”pad_sequences”, a method from keras to ensure that the length of those token-label tupels does not exceed the maximal length of 72, which is proposed by the author.

Afterwards, we split our dataset to get a training and a validation set. For the data loader of our sets we use the proposed batch size of 32.

Before we can train our model, we have to load it, using Huggingface’s BertForTokenClassification. Again, we load the cased basic BERT model. Additionally, we use the optimizer AdamW and try different values for the learning rate, parameters and epsilon. During the training/validation phase, we work with 3-4 epochs.

Generally, you should ensure to have outstanding hardware or to you use Google’s collab and to train the model on a GPU.

4 Performance measures

As discussed in the section of data investigation the data is highly skewed as a performance measure we use Precision, Recall and F-score, which are commonly used in language processing tasks. Precision is the fraction of relevant instances among the retrieved instances, while recall is the fraction of the total amount of relevant instances that were actually retrieved. F-score is the geometric average of precision and recall. Since we have a multi-label case, we are considering the average of the F1 score of each class with weighted by support (the number of true instances for each label). All three of these metrics are therefore based on an understanding

and measure of relevance.

4.1 spaCy benchmark

spaCy is the advanced library for Natural Language Processing (NLP). The spaCy models perform competitive industry level results especially beneficial for comparison with more sophisticated methods. The straightforward end-to-end raw text pipelines allow to achieve a solid prediction baseline with significantly less resource and in a time-consuming manner.

Used pre-trained linguistic model provides the abstraction leaving preprocessing steps behind the scenes. Different tokenisation algorithms are available, however, our proceeding was based on the default tokenizing settings since it is the most commonly used approach.

The model receives an input in the form of single text corpora and outputs resulting word-entity pairs. The size of text input is limited, thus our implementation based on 100.000 examples.

This SpaCy pre-trained model's performance for Named Entity Recognition task is hurdled by the different annotation schemes used in pre-trained model implementation and the at-hand dataset labels.

Generally, F1-score averages to 42% as it is visible from Figure 14 in the Appendix, that per label F1-score reaches maximum of (57 %) for easily deffirentiable label ("tim") and performs worse, at approximately 10 percent, for "gpe" and "geo", which are harder to distinguish contextually. However, it is remains being a strong baseline approach.

4.2 Pre-trained BERT-NER benchmark

Pre-trained BERT-NER model for Named Entity Recognition available in "Transformers" library is considered even stronger baseline. Similar to spaCy model in use, BERT-NER has usually higher performance. Figure 15 shows higher average F1 score (56 %). The maximum F1 score of 75 % is achieved for "loc" tag. This is yet not very precise because of the tag number reduction.

5 Performance evaluation

5.1 CRF

In this section, we proceed with the performance evaluation of CRF with the features generated by the word parts, simplified POS tags, lower/title/upper flags and features of nearby words. With 5-fold cross-validation the model's F-score is 65%. More detailed information regarding the model one can observe in the table 6 in the Appendix. Using Eli5 package we can also explain model predictions. For example, one can note that a lot of features rely on the words themselves, such as "events" and "ramadan" for the class event. Detailed report on the features and their weights can be found in the Jupiter Notebook with space considerations.

The above observation reveals highly overfitted model. For example, for the tag 'B-per', force the algorithm not to remember the words 'president' and 'obama'. So let us use regularized likelihood 4 instead of the usual one to account for possible overfitting:

$$L = \sum_{i=1}^{n} p(s^i|x^i, w) - \frac{\lambda_2}{2} \times |w|^2 - \lambda_1 \times |w| \quad (4)$$

The F-score of the resulting model is lower than it was without regularization - 56% 9. As expected, we see, that the model stops to rely on words and uses the context more, as it generalizes better is more useful over multiple training instances.

5.2 RNN

For the evaluation of our SimpleRNN and LSTM model we take into account, as mentioned before, Recall, Precision and F1-score which can be found detailed in Figure 5 in the appendix. SimpleRNN performed better with 10 epochs and a `validation_split`of 0.2 giving us an F1-score of 41.3% which is a relatively low one and certainly lower than our expectations. This can be explained by the disability of the model to deal with long term dependencies. To set the Dropout we started with a value of 0.1 and gradually increased it to 0.5, value which the model performed best. Dropout values of 0.6 and 0.7 were also tested which resulted in a significant drop in performance measures especially in F1-score.

After not getting very good results with SimpleRNN, better prediction accuracy and F1-score was expected with Bi-LSTM, but it failed our expectations with a lower F1-score of 39.5%.

What can clearly be seen is that even though the overall accuracy measures are relatively low, if we have a look at the Name Entity Tags like "geo" (Geographical Entity) and "per" (Person) as can be seen in Figures 5 and 6 could be considered not particularly bad which are the main goal for the Anonymization problem our model wants

to solve, which can lead to better results with further training, as it is very hard to train an RNN properly. However, other algorithms like combining Bi-LSTM with CRF and CNN yielded better results.

5.3 LSTM-CRF

Now we turn to the evaluation of LTSM-CRF model, whose architecture with 4 hidden layers accounts for 738,240 trainable parameters. With 90%-10% train-test split (training on 38846 samples on 10 epochs, validating on 4317 samples) F-score is equal to 82.7%. Compared to traditional CRF model, Bidirectional LSTM-CRF not only takes advantage of CRF but also gets the benefits of Bidirectional LSTM, which can generate long-distance context representations from the past and future input features. Moreover, BiLSTM-CRF model has a remarkable advantage in taking little effort on feature engineering as compared to the simple CRF, where the researcher has to define relevant set of features.

5.4 LSTM and character embeddings

Earlier we discussed two possible deep learning methods for character embeddings. In this section, we are going to present its performance results and discuss the differences between them. Both models were trained on 10 epochs and with a `validation_split` of 0.2 in order to gain consistency between results of our implemented models and RNN models implemented before. The detailed performance report for the models can be found in Figures 10, 11 in the Appendix.

Comparing two approaches that perform feature generation for character embeddings, a difference in the performance can be seen. Surprisingly, the model with LSTM layer showed better result (F1 = 82.3%) than the model with CNN (F1 = 79.8%). However, the LSTM-CNN model used less computational time. One epoch of learning was running in average for 85 seconds for LSTM-CNN model, whereas, it took around 120 seconds for the model with LSTM layer to run one epoch. Here, we can observe a trade-off between performance and time efficiency - better results require more time.

One more dissimilarity can be seen between the predicted results. For example, in Figure 5 we displayed the result of predicting the labels using two models discussed in this section. Although the model with LSTM layer exhibited better numerical results, the LSTM-CNN model predicted

```
Word            ||True ||Pred
==============================
On              : O     O
Monday          : B-tim B-tim
,               : O     O
British         : B-org B-gpe
Foreign         : I-org O
Secretary       : B-per B-per
Jack            : I-per I-per
Straw           : I-per I-per
```

(a) LSTM

```
Word            ||True ||Pred
==============================
On              : O     O
Monday          : B-tim B-tim
,               : O     O
British         : B-org B-org
Foreign         : I-org I-org
Secretary       : B-per B-per
Jack            : I-per I-per
Straw           : I-per I-per
```

(b) CNN

Figure 5: True and predicted labels for a part of test sentence for the models that employ either LSTM or CNN layer for character embeddings

labels better in this given sentence. LSTM model did not identify entity "British Foreign" correctly. First, it mislabeled the part of organizational entity "British" by assigning the geopolitical tag to it. This can be explained by the fact that "British" may correspond to different type of entities (e.g., geopolitical, geographic, organizational). And, second, it did not identify "Foreign" and its connection to the "British" entity. I the meantime, the LSTM-CNN model managed to label all words in this example correctly.

Since we are focusing on the NER approach that can anonymize the data, some labels are more interesting for us than others. For example, "per" ("Person"), "org" ("Organization") or "geo" ("geopolitical entity"). Clearly, the model with the LSTM layer for character embeddings had bigger F1-score at each class than the model with CNN layer since it has bigger overall average F1-score.

5.5 LSTM-CNN-CRF

As we have seen in the previous section, the models with character embeddings already showed state-of-the-art results. However, the LSTM-CNN-CRF and LSTM-LSTM-CRF models displayed even better performance. The weighted F1-score equals 83.9% and 84.1% respectively (detailed report can be seen in fig. 12, 13 in Appendix). However, due to the additional CRF layer, the computational time increased: the run time of one epoch for a model

with CNN layer was approximately 130 seconds while for the model with LSTM layer it took more than 300 seconds to run one epoch. Once again, we observe the dependence of performance metrics on computational time.

5.6 spaCy

Generally, one can see from Figure 14 in the Appendix, that per label F1-score reaches maximum of (57 %) for easily deffirentiable label ("tim") and performs much worse (F1-score equals approximately 10%) for "gpe" and "geo", which are harder to distinguish contextually. However, it is remains being a strong baseline approach.

5.7 BERT

To measure the performance of BERT, we used 10k, 100k and all words from the dataset, where 10% was used for validation. Although we worked with different parameters, we could not even get close to an F1 score of 92%, which was achieved in the original paper. The maximal F1 score we could get was 85.44% using full fine tuning, a learning rate of 3e-5, an epsilon of 1e -8 and 4 epochs. The table you see below describes the accuracy and F1 score if we used 3 epochs, a learning rate of 3e-5, an epsilon of 1e -8. Full fine tuning means the tuning of the parameters in Adam.

#Words	Accuracy	F1
10,000	0.87167	0.35335
100,000	0.94249	0.7303
ALL	0.96268	0.83785

Table 1: Accuracy and F1 using BERT with fine tuning Adam

#Words	Accuracy	F1
10,000	0.08524	0.0254
100,000	0.80744	0.00514
ALL	0.84254	0.27026

Table 2: Accuracy and F1 using BERT without fine tuning Adam

6 Conclusion

Our study investigated the different tools and algorithms for Named Entity Recognition aiming to improve existing procedures to meet newly imposed GDPR regulations. We started with arguably most common and straightforward CRF algorithm, which exhibited rather low performance of 65% F-score without regularization and 56% with that, which makes it unreliable as stand-alone tool. Next we introduced template RNN model for NER task with "memory" - LSTM - that is needed to utilize existent dependencies between terms in the sequence. Simple RNN performed even worse than CRF (41.3%), whereas LSTM dropped even further to 39.5%. However, the experiment with allowing simple CRF to retain some "memory" in the form of LSTM boosted performance to 82.7%.

We proceeded to combine different algorithms in order to find out, if it is possible to upgrade performance even further by exploiting strengths of single algorithms. So LSTM with the layer of not only word-level but also character level embeddings was combined with CNN also resulting in a high level of F-score - 79.8%, which can be directly compared to the character embeddings with LSTM level - 82.3%. Another combination LSTM-LSTM-CRF, where there were two consecutive LSTM layers with character level embeddings performed also well with F-score being 84%. Last combination was LSTM (again with character level embeddings), CNN and CRF was used resulting in F-score of 83.9%.

Last but not least we implemented trending nowadays models - self-fine-tuned BERT and spaCy. The former exhibited the highest performance of F-score being equal to 85.44% and the latter in only 57%.

Future work can be done by assessing performance not only in terms of performance measures typical for NER tasks, but also speed of the execution. This can be vital, when applying models to the real GDPR settings, when time is major factor for vast majority of businesses.

References

François Chollet et al. 2015. Keras. https://keras.io.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. *Long short-term memory. Neural computation*.

Matthieu Labeau, Kevin Löser, and Alexandre Allauzen. 2015. Non-lexical neural architecture for fine-grained POS tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237, Lisbon, Portugal. Association for Computational Linguistics.

Ceshine Lee. Understanding bidirectional rnn in pytorch.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Aditi Mittal. Understanding rnn and lstm.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Cicero D Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st international conference on machine learning (ICML-14)*, pages 1818–1826.

Cicero Nogueira dos Santos and Victor Guimaraes. 2015. Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*.

Tobias. Depends on the definition.

Abhinav Walia. Annotated corpus for named entity recognition. https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus/activity.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

A Accompanying Tables and Graphs

	precision	recall	f1-score	support
B-art	0.37	0.11	0.17	402
B-eve	0.52	0.35	0.42	308
B-geo	0.85	0.90	0.88	37644
B-gpe	0.97	0.94	0.95	15870
B-nat	0.66	0.37	0.47	201
B-org	0.78	0.72	0.75	20143
B-per	0.84	0.81	0.82	16990
B-tim	0.93	0.88	0.90	20333
I-art	0.11	0.03	0.04	297
I-eve	0.34	0.21	0.26	253
I-geo	0.82	0.79	0.80	7414
I-gpe	0.92	0.55	0.69	198
I-nat	0.61	0.27	0.38	51
I-org	0.81	0.79	0.80	16784
I-per	0.84	0.89	0.87	17251
I-tim	0.83	0.76	0.80	6528
O	0.99	0.99	0.99	887908
accuracy			0.97	1048575
macro avg	0.72	0.61	0.65	1048575
weighted avg	0.97	0.97	0.97	1048575

Figure 6: Performance of CRF without regularization

	precision	recall	f1-score	support
B-art	0.00	0.00	0.00	402
B-eve	0.80	0.27	0.40	308
B-geo	0.82	0.90	0.86	37644
B-gpe	0.95	0.92	0.94	15870
B-nat	0.69	0.09	0.16	201
B-org	0.78	0.67	0.72	20143
B-per	0.80	0.76	0.78	16990
B-tim	0.93	0.83	0.88	20333
I-art	0.00	0.00	0.00	297
I-eve	0.64	0.12	0.20	253
I-geo	0.81	0.73	0.77	7414
I-gpe	0.93	0.37	0.53	198
I-nat	0.00	0.00	0.00	51
I-org	0.75	0.76	0.75	16784
I-per	0.80	0.90	0.85	17251
I-tim	0.84	0.67	0.74	6528
O	0.99	0.99	0.99	887908
accuracy			0.97	1048575
macro avg	0.68	0.53	0.56	1048575
weighted avg	0.96	0.97	0.96	1048575

Figure 7: Performance of CRF with regularization

F1-score: 48.6%

	precision	recall	f1-score	support
geo	0.52	0.78	0.63	3816
gpe	0.42	0.88	0.57	1565
org	0.00	0.00	0.00	1939
tim	0.00	0.00	0.00	2076
per	0.44	0.72	0.55	1692
eve	0.00	0.00	0.00	41
art	0.00	0.00	0.00	31
nat	0.00	0.00	0.00	16
micro avg	0.47	0.50	0.49	11176
macro avg	0.30	0.50	0.38	11176

Figure 8: Performance of SimpleRNN

F1-score: 39.5%

	precision	recall	f1-score	support
geo	0.37	0.78	0.50	3816
gpe	0.00	0.00	0.00	1565
org	0.00	0.00	0.00	1939
tim	0.00	0.00	0.00	2076
per	0.56	0.74	0.64	1692
eve	0.00	0.00	0.00	41
art	0.00	0.00	0.00	31
nat	0.00	0.00	0.00	16
micro avg	0.41	0.38	0.40	11176
macro avg	0.21	0.38	0.27	11176

Figure 9: Performance of LSTM

	precision	recall	f1-score	support
B-art	0.42	0.10	0.16	49
B-eve	0.75	0.27	0.40	33
B-geo	0.85	0.91	0.88	3735
B-gpe	0.95	0.94	0.95	1596
B-nat	0.83	0.22	0.34	23
B-org	0.81	0.70	0.75	2071
B-per	0.86	0.83	0.84	1694
B-tim	0.92	0.87	0.90	2158
I-art	0.00	0.00	0.00	42
I-eve	0.33	0.12	0.18	33
I-geo	0.81	0.81	0.81	707
I-gpe	1.00	0.44	0.61	16
I-nat	0.75	0.60	0.67	5
I-org	0.81	0.80	0.80	1786
I-per	0.86	0.89	0.88	1739
I-tim	0.86	0.67	0.75	726
O	0.99	0.99	0.99	89144
accuracy			0.97	105557
macro avg	0.75	0.60	0.64	105557
weighted avg	0.97	0.97	0.97	105557

Figure 10: Performance of LSTM layer for character embeddings

	precision	recall	f1-score	support
B-art	0.27	0.06	0.10	49
B-eve	0.48	0.36	0.41	33
B-geo	0.85	0.87	0.86	3735
B-gpe	0.96	0.92	0.94	1596
B-nat	0.38	0.52	0.44	23
B-org	0.74	0.73	0.73	2071
B-per	0.84	0.81	0.83	1694
B-tim	0.92	0.87	0.89	2158
I-art	0.25	0.02	0.04	42
I-eve	0.24	0.18	0.21	33
I-geo	0.80	0.77	0.78	707
I-gpe	0.70	0.44	0.54	16
I-nat	1.00	0.20	0.33	5
I-org	0.78	0.77	0.77	1786
I-per	0.86	0.86	0.86	1739
I-tim	0.82	0.70	0.75	726
O	0.99	0.99	0.99	89144
accuracy			0.97	105557
macro avg	0.70	0.59	0.62	105557
weighted avg	0.96	0.97	0.96	105557

Figure 11: Performance of LSTM-CNN

	precision	recall	f1-score	support
B-art	0.00	0.00	0.00	49
B-eve	0.59	0.30	0.40	33
B-geo	0.86	0.90	0.88	3735
B-gpe	0.96	0.94	0.95	1596
B-nat	0.46	0.26	0.33	23
B-org	0.81	0.71	0.75	2071
B-per	0.84	0.84	0.84	1694
B-tim	0.93	0.85	0.89	2158
I-art	0.00	0.00	0.00	42
I-eve	0.43	0.18	0.26	33
I-geo	0.81	0.81	0.81	707
I-gpe	0.91	0.62	0.74	16
I-nat	0.00	0.00	0.00	5
I-org	0.82	0.77	0.79	1786
I-per	0.84	0.93	0.88	1739
I-tim	0.89	0.66	0.75	726
O	1.00	1.00	1.00	343287
accuracy			0.99	359700
macro avg	0.66	0.57	0.60	359700
weighted avg	0.99	0.99	0.99	359700

Figure 12: Performance of LSTM-CNN-CRF

	precision	recall	f1-score	support
B-art	0.75	0.06	0.11	49
B-eve	0.29	0.30	0.30	33
B-geo	0.84	0.93	0.88	3735
B-gpe	0.98	0.92	0.95	1596
B-nat	0.50	0.35	0.41	23
B-org	0.80	0.73	0.76	2071
B-per	0.88	0.82	0.85	1694
B-tim	0.90	0.89	0.89	2158
I-art	0.00	0.00	0.00	42
I-eve	0.21	0.27	0.24	33
I-geo	0.74	0.86	0.80	707
I-gpe	1.00	0.44	0.61	16
I-nat	0.50	0.20	0.29	5
I-org	0.77	0.84	0.80	1786
I-per	0.86	0.90	0.88	1739
I-tim	0.74	0.77	0.75	726
O	1.00	1.00	1.00	343287
accuracy			0.99	359700
macro avg	0.69	0.60	0.62	359700
weighted avg	0.99	0.99	0.99	359700

Figure 13: Performance of LSTM-LSTM-CRF

0.8826057133634603
0.4268780634190257

	precision	recall	f1-score	support
o	0.54	0.53	0.53	9868
org	0.46	0.44	0.45	1817
geo	0.64	0.05	0.09	3283
gpe	0.09	0.18	0.12	1738
per	0.44	0.47	0.45	1658
tim	0.48	0.69	0.57	1807
eve	0.17	0.33	0.23	52
micro avg	0.43	0.42	0.43	20223
macro avg	0.50	0.42	0.41	20223

Figure 14: Performance of spaCy model

0.917380804821234
0.5610241273100617

	precision	recall	f1-score	support
o	0.61	0.52	0.56	8154
per	0.59	0.54	0.56	1196
loc	0.69	0.81	0.75	2543
org	0.38	0.40	0.39	1370
misc	0.65	0.36	0.47	3481
micro avg	0.61	0.52	0.56	16744
macro avg	0.61	0.52	0.55	16744

Figure 15: Performance of Pre-trained BERT-NER