Bartłomiej Kowalczuk - Nanodegree Capstone Project

Named entity recognition

1. Definition

1.1. Project overview

Named entity recognition (NER) is a task of tagging entities in text data, that focuses on classifying real-world entities into predefined categories such as, among others, organizations, locations or person identifications. In simple terms, we want to know if a particular word represents some contextual meaning that we could use for further inference.

In Natural Language Processing NER can highly enhance systems that perform text search and text inference. Knowing the named entities, we can simply extract part of texts that contain them, when initially we didn’t know the exact entities. We can improve our and algorithm understanding of everything we know about a person, an organization or a place mentioned in a document.

A great example of usage of an algorithm that can perform NER with high precision would be resume scoring model. Such a model would obtain a resume in unstructured format, process and tag entities which could be followed by some scoring model that would rate it based on features (entities). Another way it could be useful is simply for analytical purposes during the recruitment processes. Extracted and highlighted entities would speed up the process of resume analysis.

Currently the best algorithms used for named entity recognition can be found here Named Entity Recognition - paperswithcode and here Named entity recognition - NLP-progress. These are mostly transformer based, however for some specific tasks solutions utilizing Bidirectional LSTM (with multiple modifications) are still performing better. On this basis, I chose BiLSTM as a challenger to RoBERTa, which recently achieved state-of-the-art results in NER domain[1]. If you would like to get to know those models better, I would strongly encourage you to start with articles published by Jay Allamar: 1, 2, 3.

[1] https://arxiv.org/abs/2002.08902

1.2. Dataset and Input

Data source: Annotated Corpus for Named Entity Recognition

This is the extract from Groningen Meaning Bank (GMB) corpus which is tagged, annotated and built specifically to train the classifier to predict named entities such as name, location, etc.

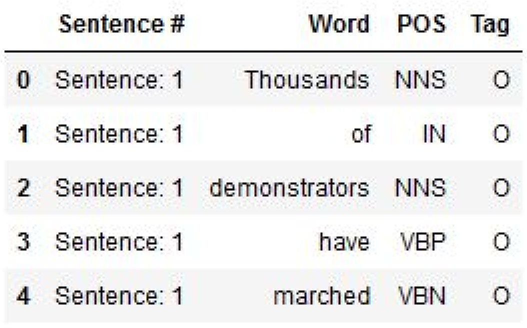
In this project I use BIO notation, which differentiates the beginning (B) and the inside (I) of entities. O is used for non-entity tokens.

Dataset consists of:

- 1354149 words
- 47959 sentences
- 17 distinct entity tags (names)

First five rows of the dataset are presented in Figure 1 below.

Figure 1. Dataset



	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	Sentence: 1	of	IN	O
2	Sentence: 1	demonstrators	NNS	O
3	Sentence: 1	have	VBP	O
4	Sentence: 1	marched	VBN	O

Source: Own preparation.

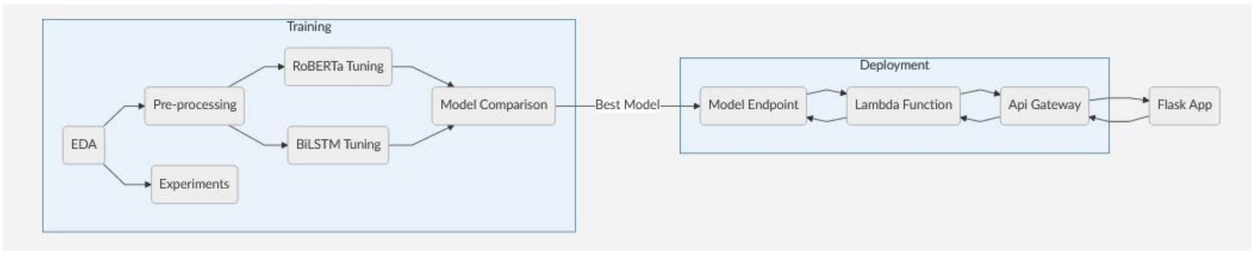
Each token has exactly one label (tag) and sentence number. *Word* variable (aggregated by sentence number) will be used as an input (predictor) and *Tag* as an output (explained variable).

1.3. Problem statement

Problem can be defined as: classification of tokens (words) in a sequence into multiple classes - named entity recognition. For classification BiLSTM and RoBERTa were utilized and compared.

Detailed strategy is presented below in Figure 2.

Figure 2. Project flow



Source: Own preparation.

Project is separated into three parts: training, deployment and web app. Training part consists of initial exploratory data analysis (EDA), experiments, preprocessing, tuning and finally model comparison. Best model is then deployed in the second part, where the endpoint is reachable by the API via Lambda function. Lastly, I enabled users to request the API directly from Flask web app.

1.4. Metrics

For evaluation purposes and comparison of models, F1 score was used. High quality classifiers require high values of precision and recall, by utilizing F1 score we summarize a model's performance into a single metric making it a perfect candidate for NER task evaluation, defined as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F1\ Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

In the multi-class case, the final metric is the average of the F1 score of each class. I used entity-level evaluation instead of token-level to correctly assess the quality of the model[2].

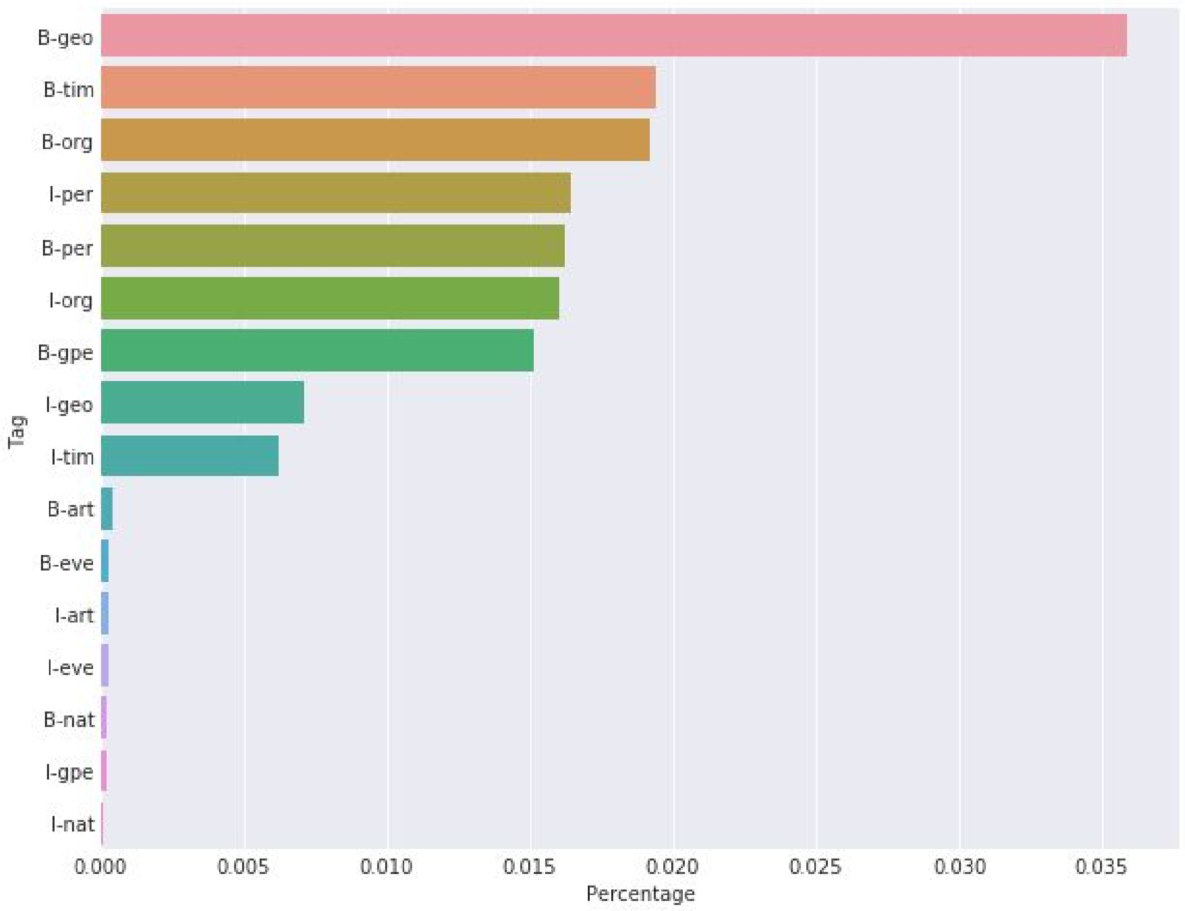
[2] https://towardsdatascience.com/entity-level-evaluation-for-ner-task-c21fb3a8edf

2. Analysis

2.1. Data exploration and visualization

As we could expect the most frequent tag is the 'O' that basically says that the word is not an entity or it is not categorized. The next most frequent tags are location names (geo), time (tim), organisation (org) and person (per) - as it is presented in Figure 3, excluding the ‘O’ tag.

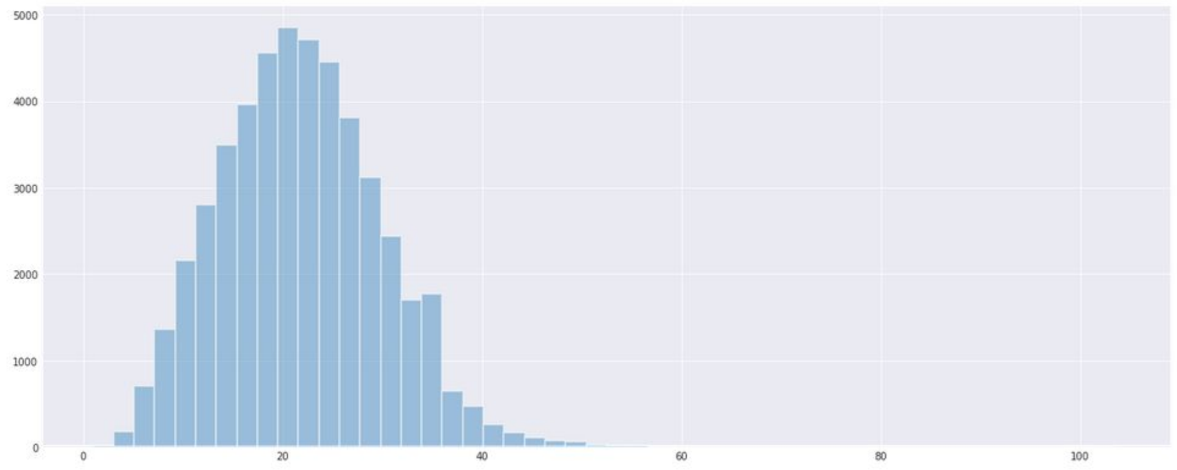
Figure 3. Percentage of tags in the dataset



Source: Own preparation.

I could try undersampling sentences with only the 'O' tag, however that might affect negatively bidirectional capabilities of the model. Thankfully, F1 score helps to overcome this problem, focusing on entities. To make the models work, I had to choose sequence length precisely to standardize the input shape. For this purpose I checked the distribution of sequence lengths, it is presented in Figure 4.

Figure 4. Sequence length histogram



Source: Own preparation.

It seemed that the most reasonable sequence length would be around 45 - we would cover most cases and won't sacrifice training and inference time (by going for the maximum sequence length). I decided to choose 45 which resulted in cutting 0.0059% of sentences.

2.2. Algorithms and techniques

Initially, based on current state-of-the-art models, I experimented locally with a simple BiLSTM model, BiLSTM-CRF and fine-tuned BERT in order to prepare proofs of concepts. It turned out that CRF layer is implemented only for Tensorflow 2.2 (or higher) and that is not currently supported in TensorflowEstimator in AWS SageMaker, so I decided to tune simple BiLSTM - it played a role of challenger for the main model. BERT model achieved great results, however after analyzing articles that recently examine current state-of-the-art models in NER task I went for RoBERTa[3]. Brief description of both architectures:

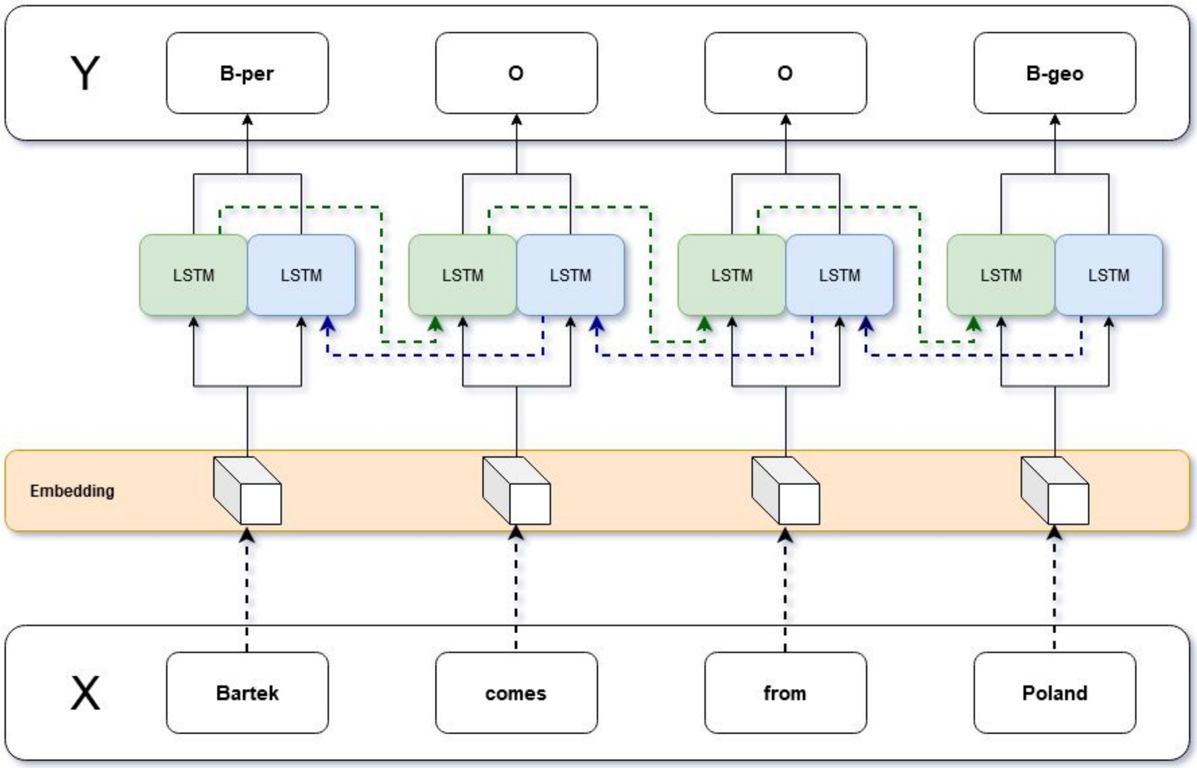
- BiLSTM

This model consists of an embedding layer which is created during training and is used to embed tokens (words) in high dimensional space - in simple words it tries to map each word to some sense that is understood by the model, similar words have close representation. Then such embedded tokens go through a Bidirectional LSTM layer, which you can understand in a high level as a set of two sequences of cells that collect knowledge and relations among the input sequence - one going from the beginning of

[3] https://arxiv.org/abs/2002.08902

that sequence and one going from the end. This knowledge flow is presented on Figure 5 below.

Figure 5. BiLSTM diagram



Source: Own preparation.

Finally, the knowledge from LSTM cells is combined and transformed by softmax function into a target distribution, which basically is a set of probabilities for each entity tag.

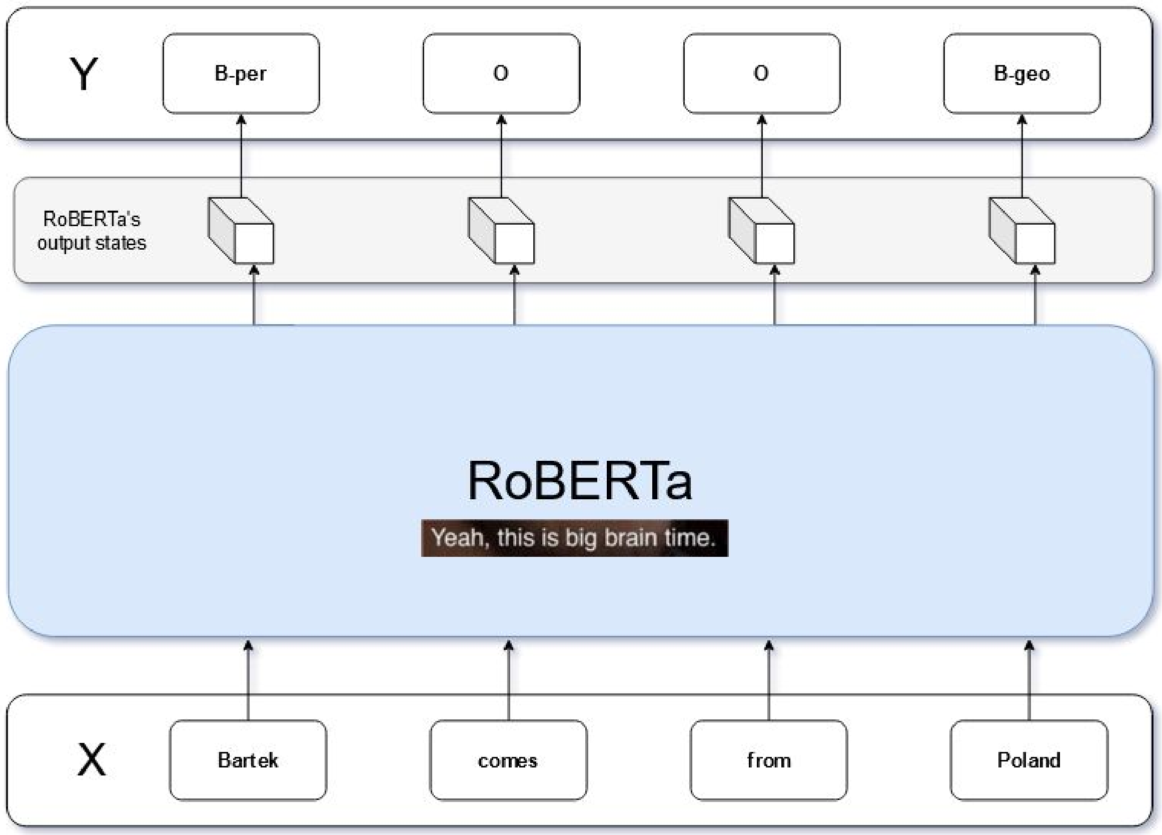
- RoBERTa

As you could see, BiLSTM combines knowledge from both directions in a way that does not adapt the knowledge from both sides simultaneously. It only combines the data in the final step. If we really want to extract the internal knowledge that is hidden in a sequence and around each word we have to gather it all at the same time. Thankfully, algorithms from transformer family comes to the rescue[4]. In such algorithms we are utilizing a specific mechanism called self-attention. It is a technique that allows us to look at other words while we encode it. What is beautiful about this solution is that we

[4] https://arxiv.org/abs/1706.03762

not only look at other words but also choose the words that bring more value to interpretation of a given word - all thanks to attention. In a high level you simple need to know that models utilizing self-attention, like BERT[5] or GPT-2[6], are basically enormous embedding blocks that have a high understanding of a certain language, its grammar and are capable of assigning values (or weights) to other words and use them while trying to encode a given word. They obtained their knowledge by extensive training on gigantic datasets, thankfully we can use their pre-trained versions, utilizing libraries like e.g. Hugging Face Transformers[7] to fine-tune those models on our datasets. In the NLP community there were already a lot of different ways to the training phase, result of one of the researches was Facebook's RoBERTa[8]. There is a lot more to the complexity of this model but for the sake of simplicity, as I stated before, let's imagine it as a huge, pre-trained embedding with a linear layer on top of it which utilize RoBERTa's hidden states to produce the target distribution, you could visualize it like in Figure 6 below.

Figure 6. RoBERTa diagram



Source: Own preparation.

[5] https://arxiv.org/abs/1810.04805
[6] https://paperswithcode.com/paper/language-models-are-unsupervised-multitask
[7] https://huggingface.co/transformers/
[8] https://arxiv.org/abs/1907.11692

2.3. Benchmark

Solutions published on kaggle kernels score around 80% F1-Score on their validation set and I would aim at achieving a higher score on a similar test set. I assume that my first proposed model, BiLSTM, would play a role of challenger for RoBERTa and I will be able to properly evaluate them on the same test set.

3. Methodology

3.1. Data Preprocessing

Data processing has to be splitted into two separate processes for each model. First of them prepared data for BiLSTM and the second for RoBERTa. This distinction is caused by the specific format of data that RoBERTa requires.

Both processes started with the same sentence aggregation - I had to combine words into sequences that models consumed. Then for the BiLSTM I tokenized each word by ids for the 5000 most common words (it turned out that such small quantities played a huge role in model generalization). In the case of RoBERTa, I imported tokenizer from Hugging Face library[9] and utilize it to tokenize all words. In comparison to the challenger model this tokenizer has a vocabulary of 50000 word pieces. However, "internal knowledge" that lies inside RoBERTa pre-trained layers ensures generalization.

After tokenizing inputs for both models had to be padded to the maximum length to standardize input sequence lengths. However, RoBERTa's requires additional tokens on beginning and end of sequences (*[CLS]* and *[SEP]*) as well as attention masks that allow it to precisely track only true values of tokens during training (excluding padding). I also provided RoBERTa with segments that signify sentences in the input - but in this case we've got single sentence input, so the segments were basically arrays of zeros.

Tags (entity names) were also tokenized and dictionaries of those mapped indexes were saved for future inference in both cases.

Finally, both input sets were splitted into training and testing in the same proportions (90% training and 10% testing) and order - making the exact same sets of sentences differently transformed.

[9] https://huggingface.co/transformers/model_doc/roberta.html#robertatokenizer

3.2. Implementation

BiLSTM model was implemented in Keras[10] and SageMaker Python SDK[11], while for fine-tuning of RoBERTa I utilized Hugging Face pre-trained model[12], PyTorch[13] and SageMaker Python SDK[14].

The BiLSTM model consists of embedding, Bidirectional LSTM layer and time distributed dense layer which allows multi-class classification on token level. For RoBERTa implementation I utilized RobertaForTokenClassification class[15], which basically is a wrapper around RobertaModel[16] that adds a linear layer on top of the hidden-states output. Precise implementation can be explored in this repository.

BiLSTM was optimized using Adam optimizer, as it proved to achieve great results in many researches and the sparse categorical cross entropy loss function. RoBERTa on the other hand was also optimized with Adam optimizer, but with weight decay[17] that is recommended in fine-tuning scripts from Hugging Face.

Both models were trained using source scripts that made use of Tensorflow and PyTorch estimators from SageMaker Python SDK. Training and tuning was performed on ml.p2.xlarge instance that utilizes Nvidia's K80 graphic card[18].

3.3. Refinement

In the case of the BiLSTM model I tuned embedding dimensions, hidden units in LSTM layer and the batch size. I also used early stopping technique and my own callbacks that prevented the model from overfitting to the training set and stopped the training if the validation loss stopped improving or achieved a defined state. During the training process I also specified a 10% validation set. The best BiLSTM model consists of:

- 128 dimension embedding (from [64, 128, 256])
- One dimension spatial dropout with dropout rate of 0.1
- Bidirectional LSTM with 256 hidden units (from [128, 256, 512])

[10] https://www.tensorflow.org/guide/keras/overview
[11] https://sagemaker.readthedocs.io/en/stable/frameworks/tensorflow/index.html
[12] https://huggingface.co/transformers/model_doc/roberta.html#roberta
[13] https://pytorch.org/
[14] https://sagemaker.readthedocs.io/en/stable/frameworks/pytorch/index.html
[15]
https://huggingface.co/transformers/model_doc/roberta.html#transformers.RobertaForTokenClassification
[16] https://huggingface.co/transformers/model_doc/roberta.html#transformers.RobertaModel
[17] https://huggingface.co/transformers/main_classes/optimizer_schedules.html#transformers.AdamW
[18] https://aws.amazon.com/sagemaker/pricing/instance-types/

- Time distributed dense layer

BiLSTM achieves best results on batch size of 32 (from [16, 32, 64]).

Size of RoBERTa model has been an obstruction in the tuning process, due to extremely long training time. However, I still managed to try some combination of batch size and number of epochs. From the recommended[19] three epochs I extended it to four which turned out to have an impact on the result, lowering the training loss by 0.01. Initially, I set the recommended 32 batch size and it proved to be the best configuration compared to smaller 16 (64 trained to long and I had to terminate the process).

4. Results

4.1. Model evaluation and validation

Models achieved great results on the test set, nonetheless RoBERTa proved to be superior. Results of both models are presented in Table 1.

Table 1. Model results

	F1 Score
BiLSTM	0.788
RoBERTa	**0.838**

Source: Own preparation.

Fine-tuned RoBERTa achieves the highest result from all that are posted in kaggle kernels for this dataset.

4.2. Justification

Due to not only the generalization and regularization that the internal knowledge of RoBERTa pre-trained layers has but also the self-attention mechanism, it was capable of overcoming all other solutions. Probably making it the state-of-the-art model for NER task (especially on this dataset).

[19] https://github.com/huggingface/transformers/tree/master/examples/token-classification

4.3. Deployment

RoBERTa model has been successfully deployed using SageMaker Python SDK and could be used thanks to Lambda function and API Gateway. For this purpose, a Web App was developed in Flask framework.

4.4. Further research

In further development I plan to compare the results from this project with architectures utilizing CRF layers and embeddings like ELMo.