Named Entity Recognition

NLP project

MDS @ Universitat de Barcelona

Alejandro Castrelo - Gerard Marrugat - Toni Miranda -

Eduard Ribas - Pilar Santolaria

16 / Juny / 2019

Table of contents

1. Project description

The goal of this project is to build a Named Entity Recognition (NER) system. To do so, we will approach the problem from two different angles. First, we will train a Structured Perceptron, and second we will approach the problem with a Neural Network architecture based on Keras.
We are requested to split the data provided (ner_dataset.csv from Kaggle's Annotated Corpus for Named Entity Recognition) by:

- Train set: From "Sentence: 1" to "Sentence: 35970"
- Test set: From "Sentence: 35971" to "Sentence: 47959"

We have `47959` sentences that contain `35178` unique words and tagged by `17` tags. The total of `47959` sentences are provided and labeled as shown in Figure 1.

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O
5	NaN	through	IN	O
6	NaN	London	NNP	B-geo

Figure 1. Sample of data provided

Note that the **Tag** column has the NER entities, which include:

- geo = Geographical Entity
- org = Organization
- per = Person
- gpe = Geopolitical Entity
- tim = Time indicator
- art = Artifact
- eve = Event
- nat = Natural Phenomenon

Each of these entities may be given as the Beginning (B) or Intermediate (I) tag of the Named Entity like, for instance, B-geo at the beginning or i.e. I-geo as an intermediate position. With all these considerations, and including the tag 'O' (not a Named Entity), there is a total of 17 entities. See below the overall count for the different *entities* in the dataset:

	Tag	counts
0	B-art	402
1	B-eve	308
2	B-geo	37644
3	B-gpe	15870
4	B-nat	201
5	B-org	20143
6	B-per	16990
7	B-tim	20333
8	I-art	297
9	I-eve	253
10	I-geo	7414
11	I-gpe	198
12	I-nat	51
13	I-org	16784
14	I-per	17251
15	I-tim	6528
16	O	887908

	Tag	counts
16	O	887908
2	B-geo	37644
7	B-tim	20333
5	B-org	20143
14	I-per	17251
6	B-per	16990
13	I-org	16784
3	B-gpe	15870
10	I-geo	7414
15	I-tim	6528
0	B-art	402
1	B-eve	308
8	I-art	297
9	I-eve	253
4	B-nat	201
11	I-gpe	198
12	I-nat	51

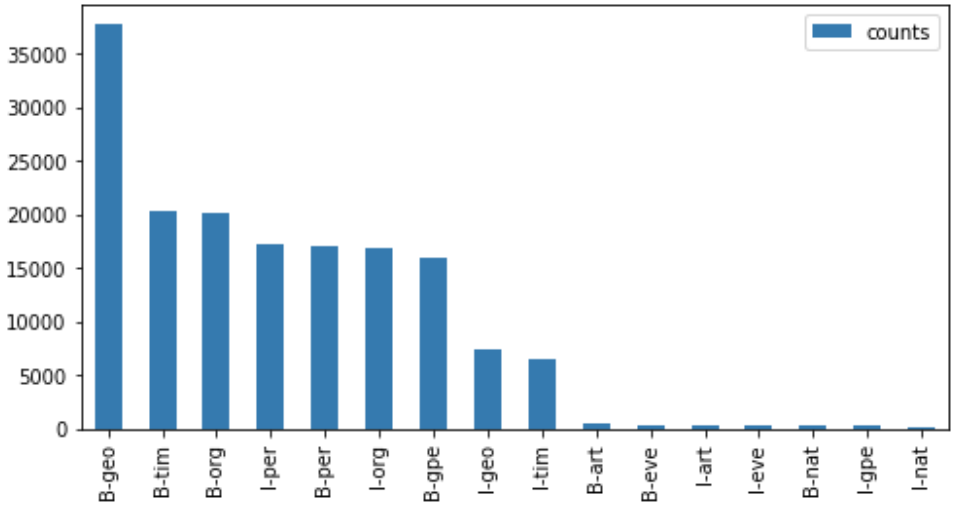

Figure 2. *Top*: Tag counts (unsorted and descendently sorted) for the overall dataset. *Bottom:* Histogram of Tags (without 'O' to facilitate readability)

We can see in Fig 2 (bottom) that the Tag appearance is quite unbalanced in the dataset. Notably, the ones appearing in the several thousands are B-geo, B-tim, B-org, I-per, B-per, I-org, B-gpe, I-geo, and I-tim (despite of course the 'O' which is reserved for the words that are not a named entity).

We are requested to present the following metrics of our predictions:

- Accuracy on the train set and test set.

- Confusion matrix on the train and test set.
- Number of sentences without any label error in both train and test sets (for each sequence we score a *1* if all the words have the correct label and a *0* otherwise).

We are also requested to test with our approaches the following sentences:

- *"The programmers from Barcelona might write a sentence without a spell checker." "The programmers from Barchelona cannot write a sentence without a spell checker." "Jack London went to Parris."*
- *"Jack London went to Paris."*
- *"We never though Microsoft would become such a big company."*
- *"We never though Microsof would become such a big company."*
- *"The president of U.S.A though they could win the war"*
- *"The president of the United States of America though they could win the war" "The king of Saudi Arabia wanted total control."*
- *"Robin does not want to go to Saudi Arabia."*

Ideally, our system should give an outcome for those sentences as the following (tested on the Stanford NLP online NER system, arguably the state-of-the art system for NER):

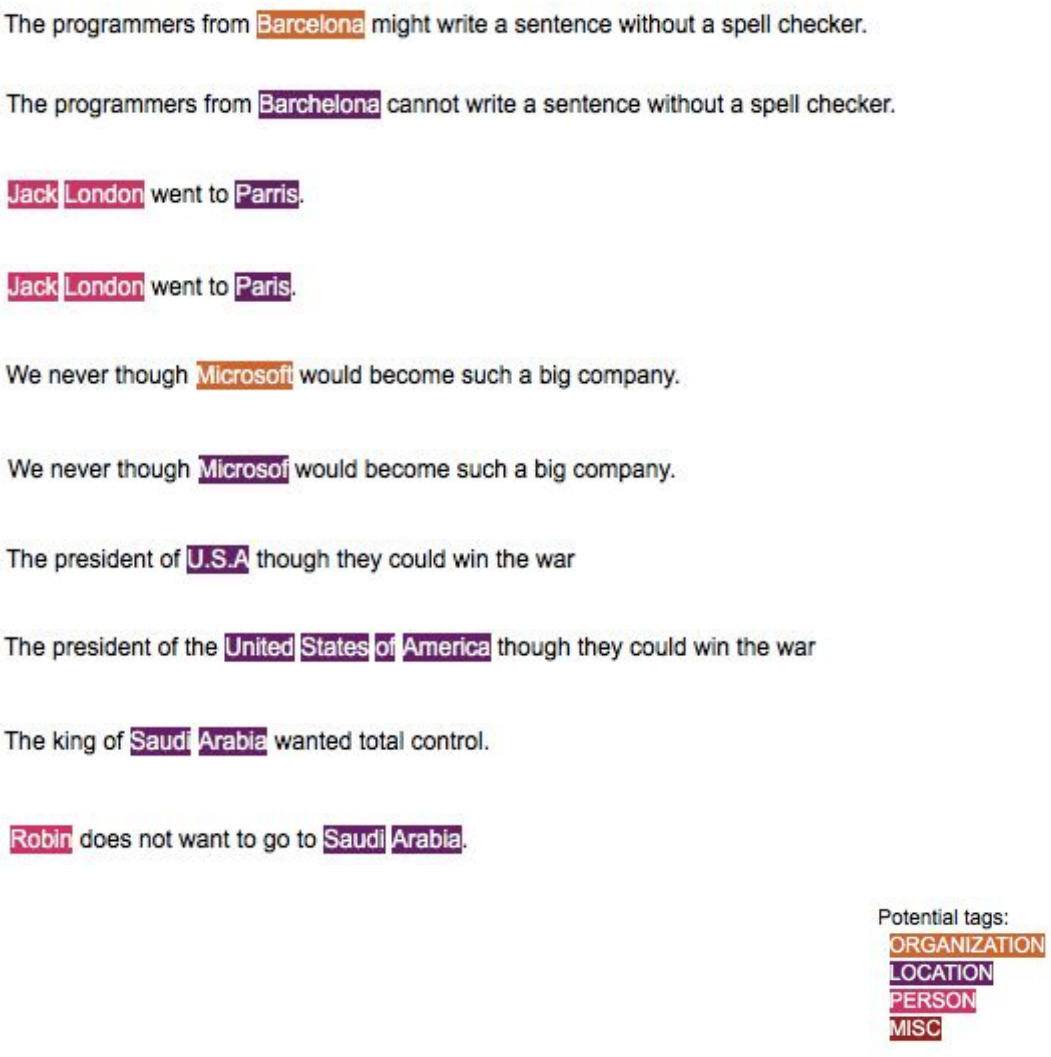

Figure 3. Ideal outcome of our NER system (based on Stanford NLP NER)

Notes on Data

For the sake of computational time, and given the different trials and attempts we were doing, in most cases we have only taken 10.000 sentences, which we divided 75% for training and 25% for test:

- train_size: 7500 test_size: 2500 total: 10000

Only for some final executions, and in order to get the best possible results, we train our models with the complete dataset.

From CSV to Sequences to Features

An important aspect regarding the dataset is how to prepare the data to fit the model. From the data shown in *Figure 1*, it has to be translated to Sequences, in which they are re-formatted into "position/tag" as shown:

```
1/0 2/0 3/0 4/0 5/0 6/0 7/1 8/0 9/0 10/0 11/0 12/0 13/1 14/0 15/0 10/0 16/0
2/0 17/2 18/0 19/0 20/0 21/0 22/0
```

The numbers represent the corresponding ids of word/tag in the dictionary. A similar approach is used for the NN. Interpreting the position (*pos*) and the *tag* into the precise word (or *entity*), we see:

Thousands/O of/O demonstrators/O have/O marched/O through/O London/B-geo to/O protest/O the/O war/O in/O Iraq/B-geo and/O demand/O the/O withdrawal/O of/O British/B-gpe troops/O from/O that/O country/O ./O

From this sequence, we will generate the SP features using the ***skseq*** library provided. See below some of the features for the sentence shown in *Figure 1*.

```
'init_tag:O': 0,
 'id:Thousands::O': 1,
 'istitle::O': 2,
 'is_long_word::O': 3,
 'id:of::O': 4,
 'is_short_word::O': 5,
 'prev_tag:O::O': 6,
 'id:demonstrators::O': 7,
 'id:have::O': 8,
 'is_medium_short_word::O': 9,
 'id:marched::O': 10,
 'is_medium_long_word::O': 11,
 'id:through::O': 12,
 'id:London::B-geo': 13,
 'istitle::B-geo': 14,
 'is_medium_long_word::B-geo': 15,
 'prev_tag:O::B-geo': 16,
 'id:to::O': 17,
 'prev_tag:B-geo::O': 18,
```

'id:protest::O': 19,
'id:the::O': 20,
'id:war::O': 21,
'id:in::O': 22,
...

The features we will consider, along with the initial (default) ones, are:

{'init_tag', 'hyphen', 'id', 'is_medium_long_word', 'allupper', 'isdigit', 'final_prev_tag', 'is_long_word', 'istitle', 'hasdigits', 'dots', 'prev_tag', 'apostrophe', 'is_medium_short_word', 'is_short_word'}

These new features we have built are further described in the following section.

These allows us to see which features are activated by which kind of tags. For instance, from the set of sentences we see that '*init_tag'* is related to the *entities* as follows:

['init_tag:O', 'init_tag:B-gpe', 'init_tag:B-geo', 'init_tag:B-per', 'init_tag:B-org', 'init_tag:B-tim', 'init_tag:B-nat', 'init_tag:B-eve', 'init_tag:B-art']

The set of features will be further developed in other sections of the document.

Notes on Data tagging

We realized that ground-truth data is not always well labeled. This is something we need to account for, as humans neither have a 100% reliability when performing labeling tasks, and the NER problem might not be completely well posed (there might be different opinions on tagging). Here we can find a couple of sentences wrongly tagged on the ground truth. We can observe that, actually, our Structured Perceptron does even better:

WORD:	TRUE:	PRED:
About	O	O
90	O	O
percent	O	O
of	B-geo	O
Mexico	I-geo	B-geo
's	O	O
exports	O	O
go	O	O
to	O	O
the	O	O
United	B-geo	B-geo
States	I-geo	I-geo
.	O	O

WORD:	TRUE:	PRED:
Officials	O	O
with	O	O
King	B-org	B-per
Mswati	I-org	I-per
's	O	O
government	O	O
say	O	O
they	O	O
did	O	O
not	O	O
want	O	O
the	O	O

Figure 4. Example of True Tag vs Predicted Tag

2. Feature generation

In this section, we describe the features that we have implemented and used afterwards in both approaches. First of all, we started by using the models without features, to see the mistakes in the predictions and discuss the features that could help to improve the results. Then, we implemented

the first features related to the case of the letters, such as if the word was capitalized, for instance. We saw that the models detected more entities based on the learning of these features, but the problem was predicting the correct label. Therefore, we started to implement new features, based on the errors in the prediction.

In the following table, for each feature, we include the feature id (how the feature is named in the code), a brief description of the feature itself and the most important errors that motivated the implementation of that specific feature. The code of the implementation of the features is in the following files of the code repository:

- *src/□skseq/□sequences/extended_feature.py* (SP features)
- *src/rnn.py* , function *get_features()* (RNN features)
- □*src/util.py* (functions used for some features)

Feature id	**Description**	**Motivating Errors**
isdigit	The word is a number	The years (B-tim) in the following sentence are missed: *That constitution was nullified by a military coup in **1962** and replaced by another constitution in **1974**.* These features helped to fix these (and other numeric) issues.
hasdigits	The word has numbers	
istitle	The first letter is upper and the rest is lower	Undetected entities and organizations. These features help to properly predict organizations like in the sentence: *Police say a total of six security guards and de-miners who worked for a U.S.-based mine clearing company (**RONCO**) and ...*
allupper	All the characters are upper cased	
hyphen	The word has a hyphen	Organization names, personal identifiers: “Mr.”, “U.S.” or like: Washington accuses Tehran of fueling violence in Iraq by training and supplying weapons to **Shi'ite** insurgents
apostrophe	The word has an apostrophe	
dots	The word has dots	
is_short_word	The word has less than 4 characters	These word length features are to finely tune the others, with the idea to help the model on properly identifying the tags.
is_medium_short_word	The word has between 4 and 5 characters	
is_medium_long_word	The word has between 6 and 7 characters	

is_long_word	The word has more than 7 characters	

3. Models

As mentioned before, two approaches are considered for this problem: a Structured Perceptron (SP) and a Recurrent Neural Network (RNN)

3.1. Structured Perceptron

In the first approach, we use a Structured Perceptron model. We described in the previous section how the feature mapping was generated from a sequence build from the raw data provided. In this model, we have a sequence coming in our system and we need to label that sequence properly. Once properly labeled, we need to update the weights of the structured perceptron where the system makes a prediction mistake. The trick here is that mistakes are part of the sequence, so the overall problem is to find which features to "punish" when the system makes a mistake. If the inference matches the true label, we don't need to update the weights. For this update, we take the features vector of the true label sequence and substract out the features vector of the sequence we predicted to finally get the weights update as outcome.

For the implementation of the SP, the provided ***skseq*** library was used, with some minor changes. For instance, some functions were added to help with the prediction of new test sentences, and a tolerance parameter was added to the model (*src/skseq/sequences/structured_perceptron.py*) in order to stop training when there is no improvement in the accuracy and to avoid overfitting.

The plot below shows the convergence of the accuracy of the SP:

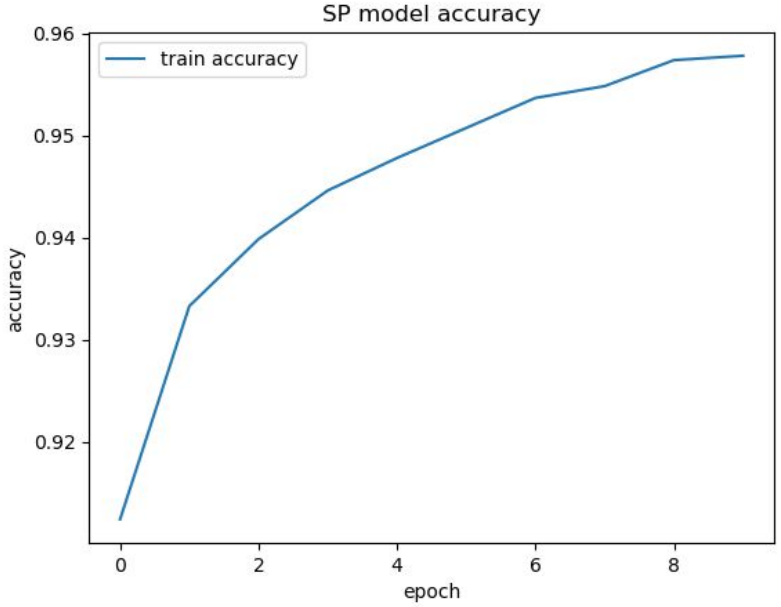

Figure 5. SP accuracy convergence
(train on 10000 sentences, complete set of features)

3.2. Recurrent Neural Network

There has been recently many attempts to increase performance by building NER systems based on Neural Nets. We can find in the literature some works that are arguably the state of the art in the field. For instance, [1] uses a Deep Neural Network architecture that uses jointly word-level and character-level embeddings to perform sequential classification. In [2], the authors use a hybrid bi-directional long-short term memory (BLSTM) and Convolutional Neural Network (CNN) to automatically detect word- and character-level features. In [3], the authors introduce among others a bidirectional LSTMs and conditional random fields to tackle NER, and obtain state-of-the-art performance in four languages without resorting to any language-specific knowledge.

Our Neural Net approach is based on a Recurrent Neural Network (RNN). We use a Bidirectional Gated Recurrent Unit (Bidirectional-GRU), which is a version of RNNs that uses a gating mechanism allowing the network to learn which information needs to be kept through time and which can be forgotten. Find here below a description of the model.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 62)	0	
embedding_2 (Embedding)	(None, 62, 50)	232450	input_3[0][0]
input_4 (InputLayer)	(None, 62, 0)	0	
concatenate_2 (Concatenate)	(None, 62, 50)	0	embedding_2[0][0] input_4[0][0]
dropout_2 (Dropout)	(None, 62, 50)	0	concatenate_2[0][0]
bidirectional_2 (Bidirectional)	(None, 62, 200)	90600	dropout_2[0][0]
time_distributed_2 (TimeDistrib	(None, 62, 17)	3417	bidirectional_2[0][0]

Total params: 326,467
Trainable params: 326,467
Non-trainable params: 0

Figure 6. RNN architecture

As seen, we use an embedding, use a Dropout of 10% to regularize.

When training, the number of epochs and convergence plots are carefully checked in order to avoid overfitting. Around 10 epochs seem to be a reasonable number to achieve a good accuracy.

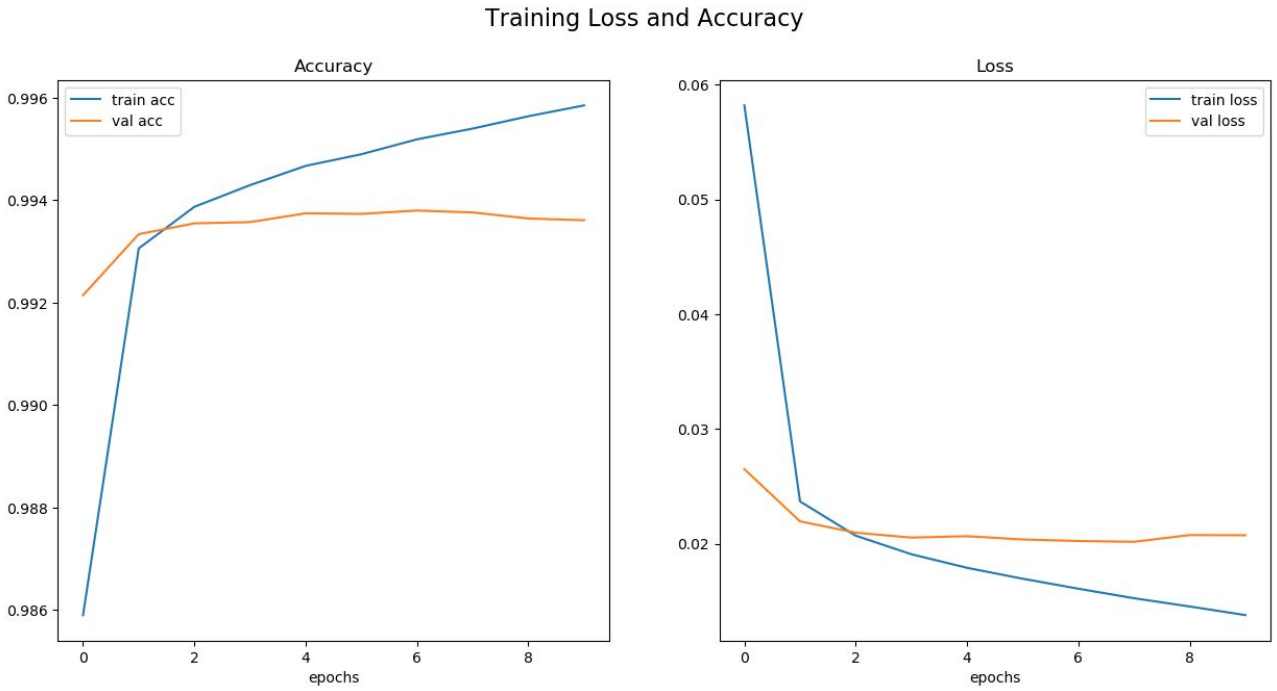

Figure 7. RNN accuracy and loss convergence
(train on complete dataset and set of features)

3.3. Results

Three metrics have been considered to report the performance of both models: accuracy, confusion matrix and the number of predicted sentences without any label error. The three of them are applied to the train and test set. In the file ***src/results.py*** it can be seen the code for each metric. For simplicity, they have been wrapped together in a function called ***metrics_to_report***, which takes as input the true and predicted tags dataset, the previous dataset but flattened as an array, the list of different classes and a suffix in order to save the confusion matrix plot.

Moreover, it was decided to plot the convergence of the model, loss and accuracy during training process, for the Recurrent Neural Network, and the accuracy over epochs for the Structured Perceptron.

Accuracy score

The accuracy score computes the ratio between the number of correctly predicted tags of the whole dataset over the total number of tags. To achieve this computation the accuracy_score function from sklearn library is used. At the input of this function is required to have all the dataset labels (true and predicted tags) as an array, for this reason, the tags datasets are flattened and saved before passing as input of the ***metrics_to_report*** function.

Confusion Matrix

Confusion matrix confronts the predicted labels against the true labels. The dimension is number of classes by number of classes. The interesting thing of a confusion matrix is that it can be seen specifically which labels are being misclassified as others. The quantities at each location of the matrix have been normalized respect the total of each row, in order to have a number between 0 and 1. Let´s illustrate it with a visible example:

Figure 8. Confusion Matrix

As it has been seen previously in Figure 2 there are 17 classes. The rows of the matrix are the True labels and the columns the Predicted labels. The colour intensity of each cell correspond to the normalized quantity. Ideally, it should be an identity matrix, this would mean that the classifier predicts correctly the totality of the tags. For example, if we analyse the first column, it can be seen that the 99% of the true tags ‘O’ are predicted as ‘O’, the 28% of ‘B-geo’ tags are predicted as ‘O’, the 18% of true ‘B-gpe’ are predicted as ‘O’ and successively for the rest of the column. The previous explanation shows us why information provided by the confusion matrix is important in the experiment. Knowing which labels are being misclassified or confused as other labels gives us a starting point to think about feature improvements. When a new feature is implemented we could print the confusion matrix again and observe if some classification improvement appeared.

Number of sentences without any label error

The last metric shows the number of sentences that have been predicted correctly in its totality. To have a better accuracy of how many sentences were predicted correctly, the percentage of these correctly predicted sentences over the total number of sentences is printed.

The table below shows the results of executing the models with different features (new features were added in subsequent executions). All the metrics described above are shown in the table or in the plots below:

	Description [features]	**Model**	**Dataset (train+test) and epochs**	**Accuracy**	**SWE (sentences without any error)**
1	Initial approach (no extra feat.)	SP	10000 sent. 10 epochs	Train: **0.9655** Test: **0.9473**	Train: **60.29%** (4522) Test: **50.80%** (1270)
		RNN		Train: **0.9827** Test: **0.9570**	Train: **77.56%** (5817) Test: **55.00%** (1375)
2	+ Numeric feat. *[isdigit, hasdigits]*	SP	10000 sent. 10 epochs	Train: **0.9636** Test: **0.9475**	Train: **59.16%** (4437) Test: **50.44%** (1261)
		RNN		Train: **0.9831** Test: **0.9570**	Train: **78.00%** (5850) Test: **54.88%** (1372)
3	+ Case-letters feat. *[istitle, allupper]*	SP	10000 sent. 10 epochs	Train: **0.9631** Test: **0.9496**	Train: **60.92%** (4569) Test: **52.20%** (1305)
		RNN		Train: **0.9832** Test: **0.9594**	Train: **77.85%** (5839) Test: **55.88%** (1397)
4	+ Punctuation feat. *[hyphen, apostrophe, dots]*	SP	10000 sent. 10 epochs	Train: **0.9627** Test: **0.9500**	Train: **60.53%** (4540) Test: **52.64%** (1316)
		RNN		Train: **0.9845** Test: **0.9589**	Train: **79.44%** (5958) Test: **55.80%** (1395)
5	+ Word length feat. *[is_short_word, is_medium_short_word, is_medium_long_word, is_long_word]*	SP	10000 sent. 10 epochs	Train: **0.9647** Test: **0.9519**	Train: **61.51%** (4613) Test: **52.84%** (1321)
		RNN		Train: **0.9838** Test: **0.9599**	Train: **78.35%** (5876) Test: **56.92%** (1423)
6	All feat. (as 5), complete dataset	SP	complete dataset 10 epochs	--	--
		RNN		Train: **0.9840** Test: **0.9696**	Train: **79.04%** (28429) Test: **65.30%** (7829)

The results above show that the initial features are already quite good to detect many tags, but adding all the extra features and fine-tunings, we managed to increase the overall accuracy, and improve on the percentage of sentences without any error. Finally, training with the complete dataset made the difference and helped to finally boost the accuracy.

The plots below show the confusion matrices of the best SP and RNN models, when all the features are considered:

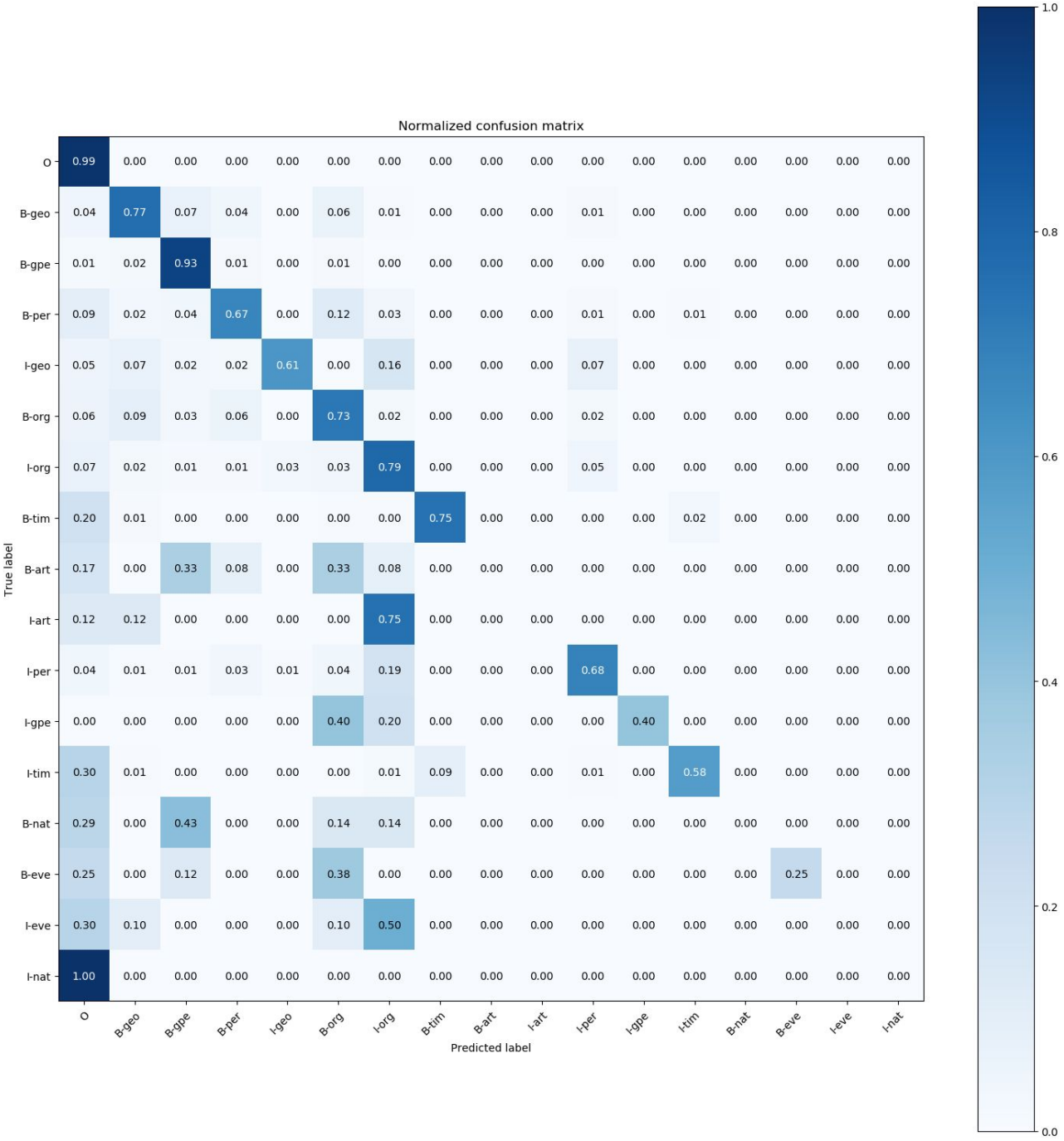

Figure 9. SP Confusion Matrix
(train on 10000 sentences, complete set of features)

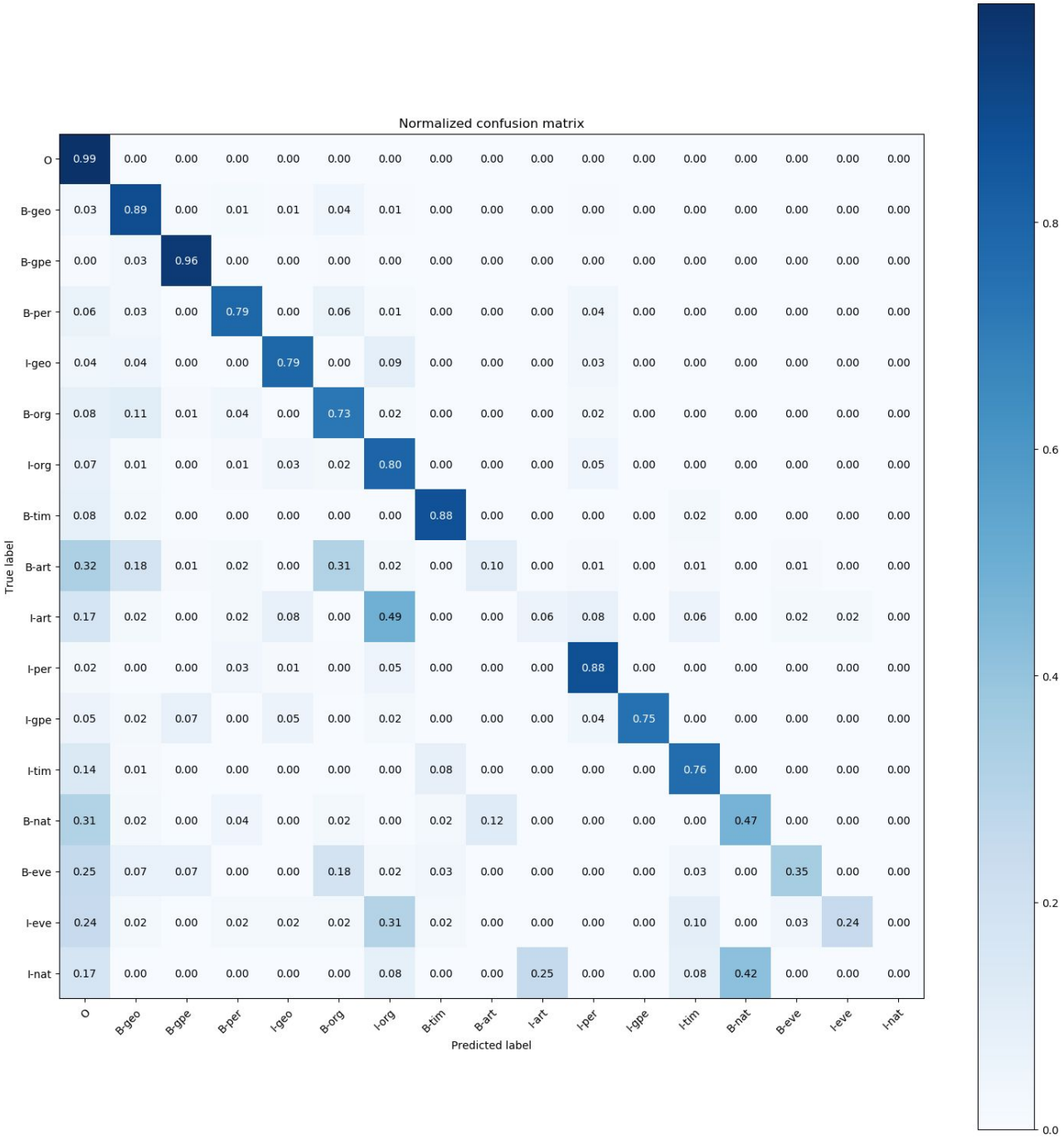

Figure 10. RNN Confusion Matrix
(train on complete dataset and set of features)

Finally, we can see below the result of the SP NER applied to the suggested test sentences:

- *The/O programmers/O from/O Barcelona/B-geo might/O write/O a/O sentence/O without/O a/O spell/O checker/O ./O*
- *The/O programmers/O from/O Barchelona/B-org can/O not/O write/O a/O sentence/O without/O a/O spell/O checker/O ./O*
- *Jack/B-per London/B-geo went/O to/O Parris/B-org ./O*

- *Jack/B-per London/B-geo went/O to/O Paris/B-geo ./O*
- *We/O never/O though/O Microsoft/B-org would/O become/O such/O a/O big/O company/O ./O*
- *We/O never/O though/O Microsof/B-org would/O become/O such/O a/O big/O company/O ./O*
- *The/O president/O of/O U.S.A/B-geo though/O they/O could/O win/O the/O war/O*
- *The/O president/O of/O the/O United/B-geo States/I-geo of/I-geo America/I-geo though/O they/O could/O win/O the/O war/O*
- *The/O king/O of/O Saudi/B-org Arabia/I-org wanted/O total/O control/O ./O*
- *Robin/O does/O not/O want/O to/O go/O to/O Saudi/B-org Arabia/I-org ./O*
- *Today/B-tim ,/O Wednesday/B-tim 12/I-tim at/O 18:00/O ,/O mid-May/O ,/O there/O is/O a/O group/O of/O 4/O students/O working/O on/O the/O NLP/B-org task/O using/O Python/B-org code/O in/O the/O sunny/O N.Y.C/B-geo ./O*

We see how our model is able to generalize and properly predict some words (and mistakes) that were not found in the training set.

4. Conclusions

The first conclusion to remark is that the task of assigning entity tags in some context may not be an easy task, even for a human being. This is due to the fact that in certain sentences, tagging an entity as a “geo” or as a “gpe” or tagging words like “later” as a “tim” entity or as “O”, can be difficult to distinguish. These subtle differences depend a lot on the immediate context of each word and are not so easy to tag and predict, even more for an automatic system.

Finally, as further work, we would evaluate the possibility of preprocess the data in order to be able to clean some entities before the prediction of the model. This would be able to avoid the possibility of having mistaken predictions for misspelled words such as “Barchelona”, “Parris”, among many others. Although the system has learned well certain features, we thought that having a cleaner input for the model to predict would help the prediction metrics. In particular, for these cases, we thought that an edit distance function or any other spellchecker approach would be useful.

Annex

a. Team contributions

In order to do this project, we first met all together to read the requirements and understand properly what we were asked to deliver. Each one of us devoted some time to explore the data, think about how to do the project and research on the topic. Then, we shared our ideas and discussed the two main approaches of the project (the Structured Perceptron and the RNN). We also discussed two other main aspects that we needed to take into account: the features that we wanted to implement and use with the different models, and the metrics that we were going to use to measure the performance of both models.

Afterwards, we distributed the different aspects discussed so that each one of us could work on specific tasks. Nevertheless, we kept sharing the code, problems, solutions and new ideas with the rest of the team, as well as contributing in almost all the parts of the project.

Pilar Santolaria: Report structure, features, structure perceptron approach, report writing.

Eduard Ribas: Report structure, features, structure perceptron approach, code unification and execution flow, report writing.

Alejandro Castrelo: Github repository structure, RNN approach.

Gerard Marrugat: Metrics and models evaluation, code unification, report writing.

Toni Miranda: Metrics and models evaluation, state-of-the art literature, report writing.

b. Files

In order to execute the code and reproduce the results above, the source files are structured as follows:

- *src/sp.py*: implements the SP model with some default parameters to be directly executed
- *src/rnn.py*: implements the RNN model with some default parameters to be directly executed
- *src/main.py*: it is a wrapper for the two modules above that allows to change some parameters:

```
usage: main.py [-h] [-m MODEL] [-d_p DATA_FILE] [-n_s NUMBER_SENTENCES]
               [-n_e NUMBER_EPOCHS] [-m_w_s MAX_WRONG_SAMPLES]

Main script for NER using SP or RNN

optional arguments:
  -h, --help            show this help message and exit
  -m MODEL, --model MODEL
                        Model to be used: SP or RNN (leave empty for BOTH)
  -d_p DATA_FILE, --data_file DATA_FILE
                        Path containing the data
  -n_s NUMBER_SENTENCES, --number_sentences NUMBER_SENTENCES
                        Number of sentences for training (DEFAULT: all
                        dataset)
  -n_e NUMBER_EPOCHS, --number_epochs NUMBER_EPOCHS
                        Number of epochs for training (DEFAULT: 10 epochs)
  -m_w_s MAX_WRONG_SAMPLES, --max_wrong_samples MAX_WRONG_SAMPLES
                        Number (max) of wrong predictions to show for the SP
                        (DEFAULT: None)
```

For instance, we can do:

python main.py -n_s 10000 -m_w_s 30 -m SP

to train a SP with 10.000 sentences and report 30 sentences with mistakes.
Note that the three modules must be executed from the *src* folder.

Other files like *src/data.py*, *src/util.py* and *src/results.py* implement functions that are used for both approaches.

Finally, the output plots of the executions are saved in the *results/* folder.

c. References

[1] Cicero dos Santos, Victor Guimarães, *Boosting Named Entity Recognition with Neural Character Embeddings*, Proceedings of the Fifth Named Entity Workshop, joint with 53rd ACL and the 7th IJCNLP, pages 25–33, Beijing, China, July 26-31, 2015.
[2] Jason P.C. Chiu, Eric Nichols, *Named Entity Recognition with Bidirectional LSTM-CNNs*, 2016
[3] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer, *Neural Architectures for Named Entity Recognition*, 2016