UNIVERSITY OF BARCELONA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Assignment 2: Named Entity recognition

Student name: *Noel Rabella Gras, Marcos Moreno Blanco, Pablo Álvarez Cabrera, Aitor Lucas Castellano & Adrián Fernández Cid*

Course: *Natural Language Processing*
Due date: *June 20th, 2021*

1. Introduction

The main objective of this project was to fully understand the structured perceptron (SP) algorithm, applied to a Named Entity Recognition (NER) problem. To that end, we have adapted the code provided in the course to our problem, and considered the case with extended features (including suffix sensitivity) along with the default (the baseline).

In addition, other approaches have been tested and compared to the original solution. On one hand, we wanted to check whether the linearity of the SP constituted an important limitation to performance, so we also implemented a fully-fledged neural network (namely a bidireccional LSTM model), from which it seemed natural to expect a better result. On the other hand, to gauge the importance of properly accounting for OOV words (noting it was a central point of this assignment), we have tried a HMM as well[1] (a generative model, *i.e.* one that ignores OOV words).

The rest of this report is structured as follows. Below we provide a description of the files included in the delivery[2]. Next, we include an explanation of the SP, stressing its adaptability to OOV words (in contrast with approaches like that of HMM). Subsequently, we briefly comment on the data provided and issues that came up during the assignment. Finally, we present and discuss our results and conclude with some final remarks.

2. Folder structure

In the submitted `zip`, along with this report (`maindoc.pdf`), you will find:

- `train_models.ipynb`. The main notebook, where all models are trained and stored in `fitted_models` to be tested later.

[1]We have completed and adapted the code provided in class, including the functions for counting, computing the forward and backward tables , posterior decoding and some smoothing to prevent errors from OOV words (basically we assign any OOV word uniform probability across states in the emission scores).

[2]Check our GitHub repository for all the code.

- `reproduce_results.ipynb`. Loads the models trained in the previous notebook to evaluate them and computes the required metrics.
- `utils/`. Folder with code used in the notebooks. In particular, it contains `utils.py`, where we define all the functions developed for this delivery (except those specific to the HMM). It also contains the module `sqsek`, which implements the structured perceptron and sequence functionalities, and `hmm.py`, where we put all code used only for the HMM.
- `data/`. Contains the train and test datasets used for this problem.
- `fitted_models/`. Folder containing trained models.
- `predictions/`. This is where we store model predictions when necessary.

3. The structured perceptron

The structured perceptron is a linear example of a *discriminative sequence model*. This kind of models have a similar structure to its generative alternative, Hidden Markov models (HMM), but with a key difference: while HMM are generative, i.e. they seek to mimic the joint distribution of observations and hidden states (in the case seen in class using occurrence counts) to generate a similar process for prediction; discriminative models produce a decision function based on the conditional distribution, which has the advantage of allowing arbitrary functions of the observations as features.

More specifically, in a sequential tagging problem, an HMM will take as input probabilities computed by counting (P_{init}, P_{trans}, P_{final} and P_{emis}) and a given sequence x (e.g. a sentence), returning a label y_i for each token of the input sequence. These labels are computed either individually (by marginalising over the rest and finding the optimal label for a given position, i.e. through posterior decoding) or simultaneously (by finding the most likely sequence of tags, i.e. through Viterbi decoding).

On the other hand, a discriminative model such as the structured perceptron will get its scores from optimising the weights w of a given feature vector $f(x, y)$, that may depend locally on the output y (the tags) and globally on the input x (the whole sentence). On the contrary, in an HMM the probability of a state depends only on the previous state. The discriminative scores are thus much more general, allowing for extra, informative features provided by domain knowledge to improve the performance of the model. Moreover, since the procedure by which the tags are optimised is the same as for HMM (only the scores vary) and we can write the discriminative scores in the same terms as the HMM's (initial, transition, final and emission scores), one can exploit the same efficient trellis structure of the forward-backward and Viterbi algorithms (for posterior or Viterbi decoding, respectively), as for HMM.

3.1. What happens with out-of-vocabulary words?.

A particular boost in performance derived from the possibility of arbitrary features involves the treatment of out-of-vocabulary (OOV) words when testing a model.

In the case of an HMM, since scores are computed by counting the number of occurrences in the training dataset, OOV words at test will simply give an error. One solution to this would be some kind of smoothing, as the Laplace or Lidstone techniques seen in class or the strategy we applied in our case[3], although these have the undesirable property of assigning the same score to every OOV word.

In the case of discriminative models, however, one can easily generate features that fire when a word is capitalised, contains a hyphen, a digit, a given prefix or suffix... hence providing a non-vanishing and adapted representation of OOV words. Moreover, such feature production can be automatised with feature templates, to be instantiated by a mapping when fitted to a train dataset, and the potentially high dimensionality of the feature space can be tacked by exploiting the sparse structure of the feature matrix.

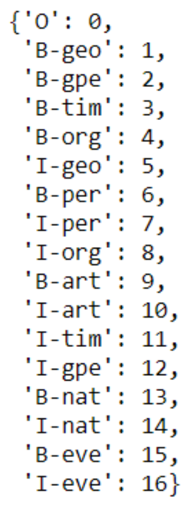
4. Dataset overview

As stated before, we are dealing with a NER problem. Concretely, the data for this problem consists of a train dataset with 38.358 sentences and a test dataset with 38.359 sentences, all of them with the format `sentence_id, words, tags`, where:

- `sentence_id` indicates the sentence to which the word in question belongs
- `words` contains the word
- `tags` is the corresponding label

To read the data, we have added the functions `read_sequence_list` and `read_data_instances` to the provided `PostagCorpus` class, adapting a function that was used to read the data from other dataset with a different structure.

Once we read the data (as a `SequenceList` object), we can see the labels considered in the problem, which are shown below.



```
{'O': 0,
 'B-geo': 1,
 'B-gpe': 2,
 'B-tim': 3,
 'B-org': 4,
 'I-geo': 5,
 'B-per': 6,
 'I-per': 7,
 'I-org': 8,
 'B-art': 9,
 'I-art': 10,
 'I-tim': 11,
 'I-gpe': 12,
 'B-nat': 13,
 'I-nat': 14,
 'B-eve': 15,
 'I-eve': 16}
```

For each category, 'B' is used for marking the Beginning, while 'I' is used for marking the Inside. The 'O' stands for words without a specific label.

[3]To avoid the error with emission scores when computing the forward and backward tables, we assigned to every OOV word uniform probability across states (see `reproduce_results.ipynb`).

5. Issues while working on the assignment

We have tried to reproduce a bidirectional LSTM from this Pytorch tutorial because we have a basic knowledge of Deep Learning and we wanted to try a complicated example. This example from the offical web of Pytorch is a Bi-LSTM Conditional Random Field for named-entity recognition. The author says that the LSTM tagger with a sequence model like the CRF is really essential for strong performance on NER. The problem we have faced is that we have not been able to make it work because the accuracies from each set has been 0, additionally, it has been computationally expensive as we have waited around 13 hours to have the trained model (that model has been trained in a cluster that Aitor and Noel are using for their master thesis for speeding up the code). As we do not have a basic knowledge on the part of LSTM and CRF we cannot have a clean vision on the error we are facing in this case.

Another (minor) issue we have encountered involves the computation of the confusion matrix for the HMM. As explained in `reproduce_results.ipynb`, the axes of the train and test matrices are off, meaning that entries do not match the corresponding tags. Unfortunately, simple though it may seem, we have not been able to solve this bug on time for the submission, although it does not prevent us from drawing the relevant information in the case of the HMM: that most non-"O" words at test are predicted as "O".-

6. Results

The table below shows the train and test accuracies (computed excluding "O"s, as requested) and weighted f-scores for all relevant models considered (excluding the LSTM):

Metric (%) \Model	SP (baseline)	Extended SP	HMM
Train accuracy	83.5	81.0	**86.3**
Test accuracy	32.2	**42.2**	14.5
Train f-score	96.6	96.1	**97.2**
Test f-score	87.1	**89.4**	82.4
Tiny test accuracy	45.2	61.3	**64.5**

The bold figures indicate the best of each row.

Let's make some comments regarding the previous table. First, from the low test accuracies we confirm that, as illustrated in class, sequential tagging is a difficult problem. Second, we see that adding some features such as those based in prefixes/suffixes to the SP reduces overfitting, improving overall performance at test. Third, we confirm the expectation that HMM perform worse than the discriminative alternatives, but with a subtlety. Indeed, as we argue in `reproduce_results.ipynb`, the poor performance of the HMM at test comes from most non-"O" words being classified as "O". However, this misclassification is not caused by those words' being OOV (as we have checked the dictionaries at train and test are the same) but by the information-lacking feature representation provided by the generative model.

The above results are consistent with the confusion matrices (that we do not include in this report for concision; you can see them in `reproduce_results.ipynb`). Those matrices are more or less diagonal for the SP but show an inflated prediction of "O"s for the HMM at test (in the form of a vertical column).

There is, however, the unexpected result of a better tiny test accuracy with the HMM than with any of the SP, especially considering the data contain many OOV words. In this regard, we note that: i) we have manually tagged the tiny test sentences for the purpose of getting the accuracy, so these measures should be taken with a grain of salt; ii) the tiny test only includes a few sentences, which means its results are sensitive to statistical fluctuations. These two are the reasons why we deem this anomaly unimportant[4], and we consider the best model to be the extended-features SP (the one with best results on the larger, reference test dataset).

Finally, to complement this section, we include the tiny test prediction of the extended SP (you can check the predictions with other models in `reproduce_results.ipynb`).

```
The/O programmers/O from/O Barcelona/B-per might/O write/O
a/O sentence/O without/O a/O spell/O checker./B-per

The/O programmers/O from/O Barchelona/B-per cannot/I-per write/O
a/O sentence/O without/O a/O spell/O checker./B-per

Jack/B-per London/B-geo went/O to/O Parris./B-per

Jack/B-per London/B-geo went/O to/O Paris./B-per

Bill/B-per gates/O and/O Steve/B-per jobs/O never/O though/O Microsoft/B-org
would/O become/O such/O a/O big/O company./B-per

Bill/B-per Gates/I-per and/O Steve/B-per Jobs/O never/O though/O Microsof/O
would/O become/O such/O a/O big/O company./B-per

The/O president/O of/O U.S.A/B-org though/O they/O could/O win/O the/O war./B-per

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./B-per

The/O king/O of/O Saudi/B-geo Arabia/I-geo wanted/O total/O control./B-per

Robin/B-per does/O not/O want/O to/O go/O to/O Saudi/B-per Arabia./I-per

Apple/O is/O a/O great/O company./B-per
```

[4]Another possibility we have considered is the tiny test results depending on the decoding, since we use Viterbi decoding with the SP and posterior decoding with the HMM. However, we have checked the tiny test results for the SP with posterior decoding and the accuracy is the same, so this does not seem to have influenced the result.

```
I/O really/O love/O apples/O and/O oranges./B-per

Alice/O and/O Henry/B-per went/O to/O the/O Microsoft/B-org store/O to/O buy/O
a/O new/O computer/O during/O their/O trip/O to/O New/B-geo York./I-geo
```

7. Conclusion

With this assignment we have been able to gain a deeper insight in NER problems, and fully understand the perceptron algorithm. Furthermore, we have worked with more complex models based on deep learning like RNNs, obtaining some interesting knowledge that could help in developing working solutions beyond the attempt reflected here.

In particular, we have confirmed the advantage of more general, richer feature representations, both when comparing the two SP (with Viterbi decoding) to the HMM (with posterior decoding), and when comparing the extended SP with the default one.

Further developments that we envisaged but lacked the time to implement include: the mentioned working version of the LSTM, perhaps also to try and produce an embedding to feed to the SP as feature representation; the addition of yet extra features to the SP through new feature templates; and the optimisation of the code (namely, cythonising it and probably replacing some suspicious numpy arrays we have seen in the SP by their more efficient sparse counterparts). It would also be interesting to find a sufficiently large, labelled dataset containing OOV words to properly check the SP against the HMM in such a scenario (as we consider the tiny test results inconclusive).

Overall, despite the struggle (or perhaps because of it) we think this has been an interesting task, and we look forward to any feedback on code optimisation, code cleaning or the pipelines in general.