NLP - Named Entity Recognition

Artur Xarles Esparraguera
Enric Azuara Olivera

Master in Fundamental Principles of Data Science
Universitat de Barcelona
2022

Contents

1 Purpose of the Task

The objective of this project is to fully understand the structured perceptron algorithm applied to Named Entity Recognition (NER). NER problems are very useful in many contexts, from information retrieval to question answering systems. The goal of this project is not to achieve the best results, but to fully understand all the details about a simple solution.

To do this project, we will work with a training dataset with 38366 sentences that contains 839149 words. Test set has 38367 sentences and 837339 words.

2 Introduction

NER is a subtask of information extraction that seeks to locate and classify named entities from a corpus. Those tags can be anything, such as person names, organizations, locations, medical codes, etc. In this problem, the tags are following the IOB Tagging system. This system contains tags of the form:

- **B- {CHUNK_TYPE}**: Word in the beginning chunk
- **I -{CHUNK_TYPE}**: Word inside the chunk
- **O-**: Outside any chunk

Besides the IOB Tagging system, we have some tags that preceed IOB to give a more precise information about the word:

- **geo:** Geographical Entity
- **org:** Organization
- **per:** Person
- **gpe:** Geopolitical Entity
- **tim:** Time indicator
- **art:** Artifact
- **eve:** Event
- **nat:** Natural Phenomenon
- **O:** No chunk token

For instance, a sentence like "Jack London went to Paris." has the following tags: ["B_per", "I_per", "O", "O", "B_geo", "O"].

3 The Algorithm: Structured Perceptron

Before training the model and showing the results obtained on the train and test set, we will do an overview of the algorithm we are using. Structured Perceptron (SP) is an algorithm that combines the perceptron algorithm for learning linear classifiers with an inference algorithm, which in our cause will be the Viterbi. After estimating model probabilities, sequence labels can be obtained using Viterbi decoding. We will start introducing all this concepts talking about the generative models and the Hidden Markov Models.

3.1 Hidden Markov Models

Sequence labeling tasks can be solved following a generative approach. One of the most known generative approaches are HMM. This types of models assumes that the system is a Markov process with unobservable (hidden) states. Thus, their main goal is maximize the hidden state probability given that depends on an initial state. We can define its probability definition as follows:

$$P(X,Y) = P_{init}(y_1|start) \cdot \prod_{i=1}^{N-1} P_{trans}(y_{i+1}|y_i) \cdot P_{final}(stop|y_N) \cdot \prod_{i=1}^{N} P_{emiss}(x_i|y_i)$$

This model works with the first order HMM independence assumptions over the joint distribution:

1. Independence of previous states
2. Homogeneous transition
3. Observation independence

If we take the logarithms of the previous expression and we introduce a set of indicator functions, certain terms can be interpreted as counts. Thus, the set of indicator functions are:

$$I_{[e]} = \begin{cases} 1 & \text{e is true} \\ 0 & \text{e is false} \end{cases}$$

If we introduce the indicator functions in the log-probability expression and we order terms, we get the following:

$$\begin{aligned} logP(X,Y) = &\sum_y logP_{init}(y|start) l_{[y_1=y]} + \sum_y \sum_{y'} logP_{trans}(y'|y) \sum_i l_{[y_i=y,y_{i+1}=y']} + \\ &\sum_y logP_{final}(stop|y) l_{[y_N=y]} + \sum_x \sum_y logP_{emiss}(x|y) \sum_i l_{[x_i=x,y_i=y]} \end{aligned} \tag{1}$$

Certain terms can be interpreted as counts:

- $\sum_i l_{[y_i=y,y_{i+1}=y']}$: Number of times transition y to y' occurs in Y
- $\sum_i l_{[x_i=x,y_i=y]}$: Number of times x is emitted from state y in (X,Y)

This results in a matrix form as the next one:

$$\log P(X, Y) = \begin{bmatrix} \log P_{\text{init}}(\cdot) \\ \text{vec}(\log P_{\text{trans}}(\cdot)) \\ \text{vec}(\log P_{\text{emiss}}(\cdot)) \\ \log P_{\text{final}}(\cdot) \end{bmatrix}^T \begin{bmatrix} e_{y_1} \\ \text{vec}(C_Y) \\ \text{vec}(C_{X,Y}) \\ e_{y_N} \end{bmatrix}$$

From the previous matrix, the left vector contains the model parameters (does not depend on X or Y) and the right vector describes X and Y (knows nothing about the model). Thus, we have a linear model, since the left vector can be seen as the weight vector W and the right one can be seen as the feature vector.

3.2 Viterbi algorithm

Now that we have seen the HMM model, we know how to estimate the probabilities but we need to obtain labels. For this decoding task we are going to use Viterbi algorithm.

Viterbi decoding can be described as:

$$y_i^* = argmax P(Y_1 = y_1, ..., Y_N = y_N | X_1 = x_1, ..., X_N = x_N)$$

So we will need to compute previous expression in order to acknowledge the maximum value and find the correct label. We will not jump into detail how the Viterbi is defined at each position. We want only to recall that once the Viterbi value at N position is computed, the algorithm is able to backtrack using the following recurrence:

- backtrack$(N+1,\, x,\, stop) = argmax(P_{final}(stop|y) \cdot Viterbi(N, x, y))$
- backtrack$(i,\, x,\, y_i) = argmax(P_{trans}(y_i|y_{i-1}) \cdot Viterbi(i-1, x, y_{i-1})$

Viterbi algorithm is going to work as long as we can map the feature vector onto each edge.

It is possible to define more general features. However, the labeling system would be incomplete without a mechanism to estimate the value of the weight vector. One way to learn this weights is the Structured Perceptron Algorithm.

3.3 Structured Perceptron Algorithm

This algorithm is based on the perceptron algorithm. This algorithm is used for structured prediction and works with objects that have a structure (like sequences) as opposed to discrete or real values. We can see the algorithm next:

```
StructuredPerceptron(E, D, φ(·), r)
# E → number of epochs, D → dataset
# Φ(·) → feature vector, r → learning rate
    W = 0
    for i = 1 to E
        for all X^(j), Y^(j) in D
            Y* = arg max_Y W^T Φ(X^(j), Y)
            if Y* ≠ Y^(j)
                W ← W + r(Φ(X^(j), Y^(j)) − Φ(X^(j), Y*))
```

From the previous algorithm, X will be sequences of words and Y from labels. $\phi(X, Y)$ can be defined as a sum of HMM-like features and the argmax operation can be implemented by means of the Viterbi algorithm. We can map the features to the different HMM distributions:

- $y = start$, $y^{'} = c$ and $i = 1$ refers to initial features.
- $y = c$ and $y^{'} = \hat{c}$ refers to transition features.
- $y^{'} = c$ and $i = N$ refers to final features.
- $y^{'} = c$ and $X = w$ refers to emission features.

These previous features are the four types of events we had before on the HMM model. However, since we are focusing on the discriminative approach, we are modelling $P(X|Y)$ instead of $P(X, Y)$. This will allow us to construct new features. We will be able to construct more than one feature at the same time for a given word and position.

Finally, the decoding part stays the same as for the HMM model. This means that we can use the Viterbi algorithm.

4 Implementation

Once we have covered in detail what is and how the structured perceptron algorithm works we will explain how we implemented this algorithm.

We have trained two different Structured Perceptrons. The first one has the default features and the second one has some extra features in order to improve a little bit the accuracy of the model. After we trained both models, we compare them taking account different factors:

- Accuracy on the train and test set taking into accound the "O" labels and not.
- Confusion matrix on the train and test set.
- F-score.
- Accuracy on a TINY_TEST provided by the professor.

4.1 Feature mapper

In order to map all the different features we are going to use skseq package provided by the professor. The function we are using is based on the concatenation of specific prefix strings with the corresponding label indicator. By doing so, we allow the model to identify to which part of the formula corresponds each feature.

As we said before, we have 4 types of features. Those features are the following ones:

- **Initial features:** This features are under the "init_tag" tag. It refers to features at the beggining of the sequence.
- **Transition features:** It has the tag "prev_tag". This features captures the iteraction between the previous tag and the following one.
- **Emission features:** They are under the tag "id". This feature activates for each word of our sequence and it determines the tag associated to a certain word.
- **Final features:** With the "final_prev_tag" tag, this features activates when its the last word of the sequence.

The first model we are going to train will have this four features. Since those features might not be enough, as we have seen, structured perceptron allows us to add more features. We will train our second model with a few added features that we will describe next.

4.1.1 Extra features

In the second model we add features related to information about the word we are studying or about the previous one. So, in addition, we deal with the following features that are combined with the tags related to the word being studied:

1. *start_upper*: if the first letter of the word if uppercase (typical in person or city names).
2. *upper*: if the whole word is uppercase (typical in some organizations).
3. *year*: if the word is made of 4 digits (typical in years).
4. *day*: if the word finishes in -day (typical in week days).

5. *dot*: if the word contains dots.

6. *esean*: if the word finishes in -ese, -an or -ans (as in american, japanese...).

7. *'*: if the word contains the character "'".

8. -: if the word contains the character "-".

9. *previous word*: In this case we check the previous word. If it is one of the following list: [through, in, at, from, before, of, the, with, which, when, after, early, late, on, until, by, finished in -ern] another feature is activated for each of them. These are special words that are usually followed by tags different to 'O'. The feature is different for each of the words.

Once we have described all the features we will add, we will show an example of how they activate and they add extra information for the model. We can see their activation in Figure 1

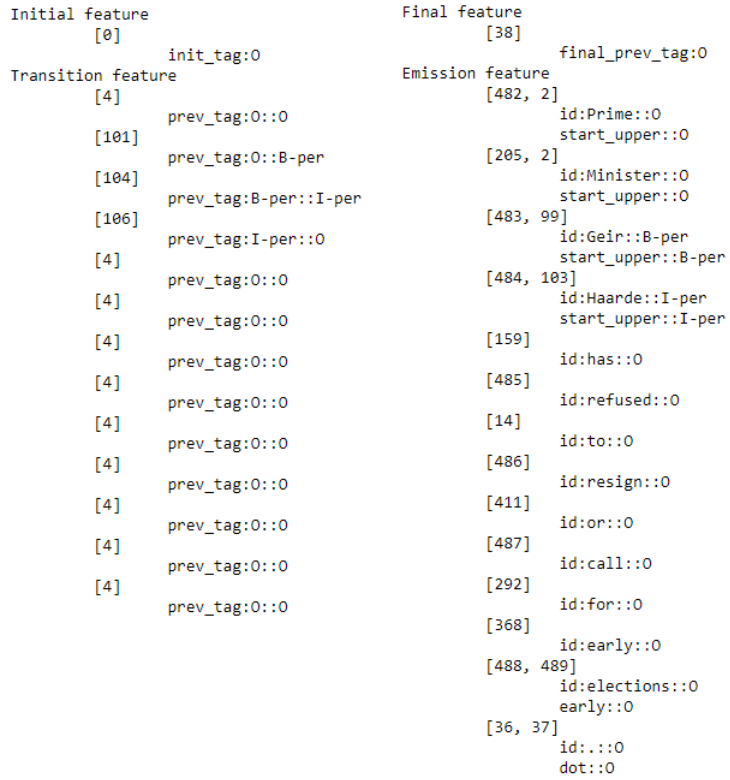

Figure 1: Example of activated features for a given sentece.

As we can see in Figure 1, we have the 4 initial types of features that we described before. The initial, transition and final features stands equal on both models, but not the emission ones. If we look into the emission features, we can observe how we have multiple activations for the different words that contain the sequence. For instance, the name "Geir Haarde" has the extra activation we added when the word has a capital letter. Another example is the word "elections". This word has an activation because it is preceeded by "early". We have seen in our dataset that 67.21% of the cases, when we have a construction of "early + word", that word is not a "O" label. On this case it is a "O", but the only purpose of the example was to show how our extra features behave.

4.2 Non observed words

Once we have seen the features that are activated for a sequence, we can easily analyze what will happen when a word that has not been previously observed appears in the moment of making a prediction. In such case, any feature related to that word and the different tags will be activated. Therefore, if we do not use any extra feature, we will not have any information to tag the word correctly. In these cases it is really important to have meaningful extra features that can define the word without being previously studied. As this extra features relates on characteristics of the word, they can be activated.

We could also try to match the non observed words with other words that already exist in our corpus, specially for those which are misspelled. We could use Levenshtein distance to find the most similar word and use that in substitution of the wrong one. However, it is a much more computational expensive approach.

5 Results

In this section we are going to recap all the results obtained with both models navigating through the different metrics, Accuracy, Confusion Matrix, Precision by tag and F-Score, and also evaluating the predictions on TINY_TEST data.

5.1 Model without extra features

For the structured perceptron without adding extra features we will present the results for the train data, then for the test data and, finally, the predictions for the tiny test set.

5.1.1 Train data results

On the train data, without taking into account the words that have as real tag 'O', we have an accuracy of 83.21%. If we make the analysis by each tag we can observe in Table 1 the precisions.

Tag	O	B-geo	I-geo	B-gpe	I-gpe	B-tim	I-tim	B-org	I-org
Precision	99.3%	83.17%	81.49%	92.39%	58.39%	77.55%	79.29%	77.97%	80.7%
Tag	B-per	I-per	B-art	I-art	B-nat	I-nat	B-eve	I-eve	
Precision	89.98%	88.21%	54.46%	64.88%	54.78%	52.27%	56.15%	54.15%	

Table 1: Table with the precision for each tag in the train set.

The table shows that the tags that are more dificult to predict are the ones related with events, natural phenomens or artifacts. In these cases the precision on the train set is a bit bigger than 50%, while in most of the other tag classes it is around 80%.

We can also look at the confusion matrix shown in Figure 2, which is in logarithmic scale.

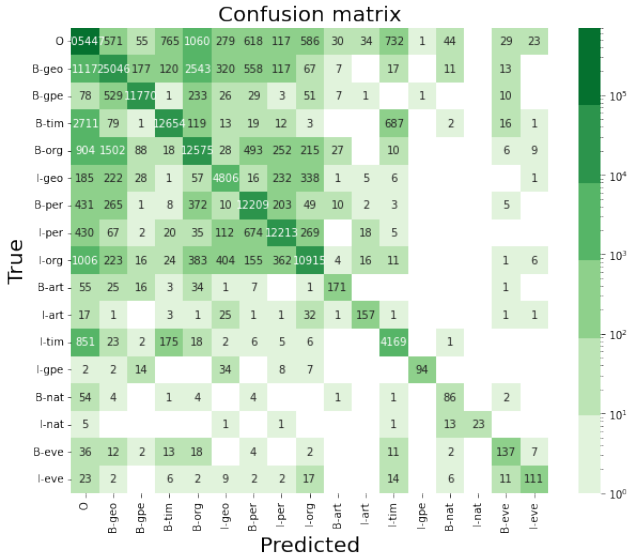

Figure 2: Confusion matrix for structured perceptron predictions on train set.

Here we can see that the model sometimes does not recognize between geographical entities and organizations. It also fails differentiating between persons and organizations in some cases.

Finally, the F1-score obtained is 97%, a pretty good result.

5.1.2 Test data results

If we test the model in unseen data the predictions are worse. For the accuracy, we correctly predict 26.01% of the data, excluding the tag 'O'. We can also see in Table 2 that the precision is lower for all the different tags.

Tag	O	B-geo	I-geo	B-gpe	I-gpe	B-tim	I-tim	B-org	I-org
Precision	99.3%	19.78%	19.34%	22.45%	9.93%	22.16%	31.12%	20.46%	18.77%
Tag	B-per	I-per	B-art	I-art	B-nat	I-nat	B-eve	I-eve	
Precision	37.78%	50.88%	7.89%	27.6%	11.43%	9.52%	20.46%	18.77%	

Table 2: Table with the precision for each tag in the test set.

In this set, the tags that are better identified excluding 'O' are the ones related to person entities. We can appreciate similar information in the confusion matrix shown in Figure 3. In this case it has much more values outside of the diagonal.

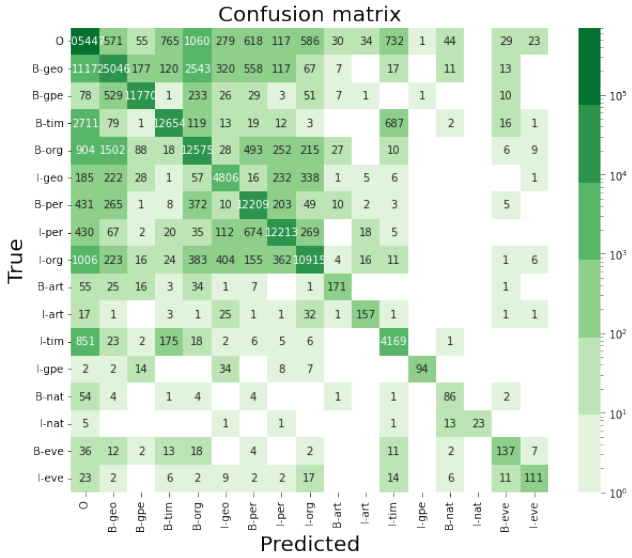

Figure 3: Confusion matrix for structured perceptron predictions on test set.

The F1-score is 86% which is quite good. However, the value is highly influenced by the 'O' tag.

5.1.3 Tiny test results

Here we will discuss the errors in the predictions when dealing with tiny test data. The full predictions for this data is available in the reproduce results notebook. In this case, the accuracy excluding 'O' is 61.76% which is a good result. However we can appreciate some mistakes:

1. 1st pair of sentences: The model can not correctly classify Barcelona when it is misspelled as Barchelona.

2. 2nd pair of sentences: The model can not correctly tag London as the surname of a person. As before, it also has problems when Paris is misspelled.

3. 3rd pair of sentences: It correctly tag the names when they are in the correct format (upper first letters), but when the first letter is not an uppercase letter, it misclassifies jobs. However, gates (Bill gates) is correctly tagged.

4. 4th pair of sentences: In this case the word U.S.A. is not identified as a country and when it is as United States of America, it is tagged as an organization.

5. 5th pair of sentences: the model cannot identify Robin as a name, probably because it doesn't exist in the corpus.

6. 6th pair of sentences: Apple is not identified as a company. In the second sentence, it is correct to assume that it is not a company as it refers to the fruit.

7. last sentence: the model cannot correctly tag the name Alice as it is probably not seen before.

We have seen in these examples that our model has a lot of problems when dealing with unseen words. As explained before, if we don't have added features the model has not information to classify the words correctly. Therefore, we expect that some of these cases are corrected when using the second model, the one with added features.

5.2 Model with extra features

We will now check the results with the second trained model which has the extra features we commented previously.

5.2.1 Train data results

On the train data, without taking into account the words that have as real tag 'O', we have an accuracy of 83.78%. This result is almost the same as the model without extra features. If we make the analysis by each tag we can observe in Table 3 the precisions.

Tag	O	B-geo	I-geo	B-gpe	I-gpe	B-tim	I-tim	B-org	I-org
Precision	98.4%	78.45%	85.77%	93.13%	56.52%	82.04%	83.99%	81.99%	80.28%
Tag	B-per	I-per	B-art	I-art	B-nat	I-nat	B-eve	I-eve	
Precision	91.47%	88.67%	57.32%	65.7%	50.32%	43.18%	52.46%	50.24%	

Table 3: Table with the precision for each tag in the train set with extra features.

The table shows that the tags that are more dificult to predict are the ones related with events, natural phenomens or artifacts. We can see how we have a lower precision on those cases and for the other ones we are arround $80 - 90\%$. Overall, we get similar results than with the previous model without extra features.

We can also look at the confusion matrix shown in Figure 4, which is in logarithmic scale.

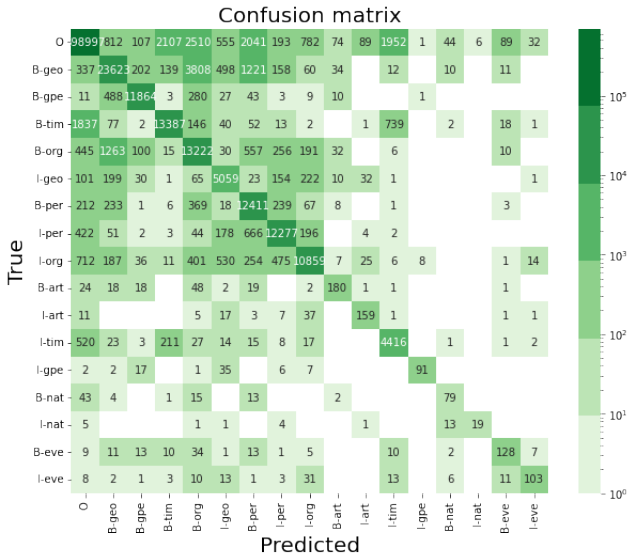

Figure 4: Confusion matrix for structured perceptron predictions on train set with extra features.

Here we can see that the model sometimes does not recognize between geographical entities and organizations. It also fails differentiating between persons and organizations in some cases.

Finally, the F1-score obtained is 96%, a pretty good result and very similar of the previous model.

5.2.2 Test data results

If we test the model in unseen data the predictions are worse. For the accuracy, we correctly predict 45.98% of the data, excluding the tag 'O'. We have an upgrade of almost 20 percentage points, which is huge. We can also see in Table 4 that the precision is hugher in almost all the tags respect to the model without extra features.

Tag	O	B-geo	I-geo	B-gpe	I-gpe	B-tim	I-tim	B-org	I-org
Precision	98.03%	35.67%	48.29%	39.01%	45.71%	38.87%	69.35%	68.18%	41.08%
Tag	B-per	I-per	B-art	I-art	B-nat	I-nat	B-eve	I-eve	
Precision	8.52%	19.91%	30.7%	9.93%	10.86%	7.14%	30.04%	15.23%	

Table 4: Table with the precision for each tag in the test set with extra features.

In this set, the tag that is better identified excluding 'O' is "I-tim", which refers to time indicator. The feature that refers to organization is really close to the time one, and overall almost all the labels have better precision except "I-per" and "I-eve", which we have a worse result. We can appreciate similar information in the confusion matrix shown in Figure 5.

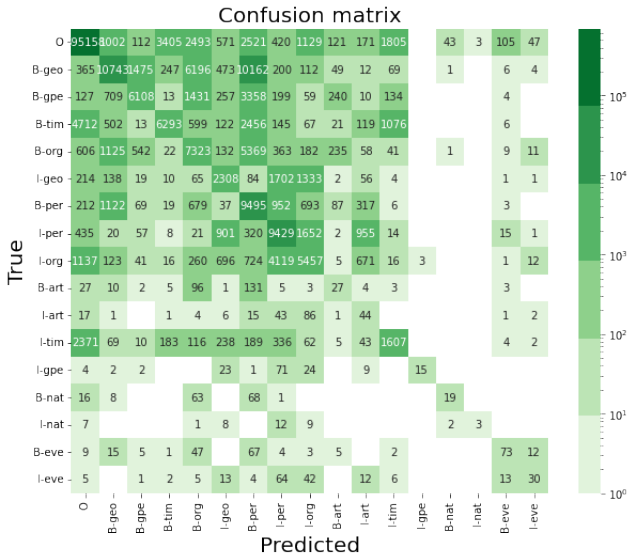

Figure 5: Confusion matrix for structured perceptron predictions on test set with extra features.

We can observe in Figure 5 how we have better test results than the model without extra features. The F1-score is 90% which is 4 percentual points higher than without extra features. Again, it is highly influenced by the 'O' tag.

5.2.3 Tiny test results

Here we will discuss the errors in the predictions when dealing with tiny test data. The full predictions for this data is available in the reproduce results notebook. In this case, the accuracy excluding 'O' is 82.35% which is a way better result respect what we had before. However we can appreciate some mistakes:

1. 1st pair of sentences: unlike the model without features, now we are able to recognize Barcelona as Barchelona. Despite that, it classifies Barcelona as an organization rather than a geographical entity.

2. 2nd pair of sentences: contrasting with the model without features, we recognize Parris as a non "O", but we missclassify it as a person.

3. 3rd pair of sentences: in this case, the model with extra features does worse since "gates" is not with an initial capital letter.

4. 4th pair of sentences: here we have an interesting case. The model with extra features does clearly better since is able to classify "United States of America" as a localization. For "U.S.A" is able to recognize it, but as a organization, which is better than not recognizing it.

5. 5th pair of sentences: the model with added features is able to recognize Robin as a name, unlike the simpler model.

6. 6th pair of sentences: in contrast with the model without features, now we are able to identify "Apple" as an organization and apple as the fruit.

7. Last sentence: with the extra features model we are able to classify "Alice" as a person, in contrast with the primal model.

In this examples we have seen mainly two things. The first one is that we do much better when recognizing names that probably we have not seen before. The second one is that we are able to recognize more words than before, but sometimes we missclassify them, like when we labeled "U.S.A" as an organization.

Given the results we obtained with the extra features model, we consider a success since we upgraded all the metrics we have checked and we do a bit better on the TINY_TEST dataset. We will see this differences on the following section.

5.3 Comparison of models

In Table 5 we can clearly see that adding the extra features previously mentioned clearly improve the results of the structured perceptron.

Metric	Train acc.	Train F1-score	Test acc.	Test F1-score	Tiny acc.
No extra features	83.21%	97.00%	26.01%	86.00%	61.76%
Added features	83.78%	96.00%	45.98%	90.00%	82.35%

Table 5: Comparison of metrics between base model and model with added features.

As we can see in Table 5, we obtained similar results on the train set with both models. This is not the same on the test and the tiny data. The model with extra features obtains a much better result on the test and tiny set and has a slightly better F1-score.

From this results we conclude that the extra features capture information that the simpler one was not able to do, specially on the non-seen data, which is mainly the important one.

6 Conclusions

In the realization of this task we studied how the structured perceptron works, in order to replicate a Hidden Markov Model. We saw that the main problem with the basic HMM is dealing with words that has not seen when training the models, as it has no emission features activated. One way to deal with this problem is adding features related to the characteristics of the words that can also math with their tags without using the whole word. Some of them are features related to uppercase initial letters, numbers in the words or some special words preceding the one that we are studying.

We applied these models in a Named Entity Recognition problem having a great performance in the training split with both models. However, we have seen how the model with extra features clearly outperforms the one without on the test data. In fact, we obtained a 20% better accuracy on the test data, going from 26.01% to 45.98%, which is a clear upgrade. On the tiny data we went from 61.76% to 82.35%. The F1-score went up aswell, but only 4 points (86% to 90%).

As a final thought, we think that the extra features we added contribute to the model for labeling better. However, we have seen on the tiny data that there is still room to improve. We are better than before identifying some words, but we sometimes missclassify them. A possible solution could be finding more and better features for the model. For instance, the misspelled words were causing trouble for the model, so a way to identify which is the actual word from the misspelled one could help the model. We could use Levenshtein distance to find the most similar word and use that in substitution of the wrong one. Another approach could be try different models, but this is out of the task's scope.