# *spaCy* for Machine Learning Entity Recognition

**Iker Foronda**
**Iñigo Ortega**

## Abstract

In this document we report the results and key points of our implementation for a NER problem, using *spaCy* software. The aim for this project is to make a model capable of recognizing *Machine Learning* related entities, for example, a paper as test data.

## 1 Description of the problem

The main objective for this project is to develop a model to resolve a NER problem. In order to do so, a library for NLP called *spaCy* will be used.

As we were not able to find an existing dataset for ML named entities, we had to make one by ourselves. The main source of ML named entities for us was the book *"Machine Learning, a probabilistic perspective"*, by Kevin P. Murphy, although the data from that book was also combined with the data we extracted from many papers from Google Scholar.

As the spectrum of ML concepts is incredibly broad, we narrowed the problem to a subset of concepts, and in order to fully use the developing features given by *spaCy*, the problem was defined as a supervised learning problem, as we have to tell which entities are within the data, and later, when the test data is used, we have to compare the prediction with the entities associated to that sample of data.

As it is mentioned before, our approach was to select a subset of entities for the model. in order to assign those entities to the concepts, we selected some "keywords" for each entity, in order to assign the labels.

The definition of our problem:

- 8 ML entities, and some examples of keywords for each of those:
  - SML = "SUPERVISED MACHINE LEARNING ALGORITHMS"
    Keywords: Naive Bayes, decision trees, ...
  - USML = "UNSUPERVISED MACHINE LEARNING ALGORITHMS"
    Keywords: Clustering, latent variable models, ...
  - MLS = "MACHINE LEARNING SOFTWARE"
    Keywords: sklearn, tensorflow, ...
  - NN = "NEURAL NETWORKS"
    Keywords: Perceptron, convolutional network, ...
  - EVM = "EVATUATION METHODS"
    Keywords: Cross validation, accuracy, ...
  - OPM = "OPTIMIZATION METHODS"
    Keywords: Gradient descent, beam search, ...
  - MLP = "MACHINE LEARNING PREPROCESSING"
    Keywords: imputation, pipeline, ...
  - MLA = "MACHINE LEARNING APPLICATIONS"
    Keywords: pattern recognition, autonomous, ...

- Dataset made by ourselves, 7 instances for each keyword.
- The data is perfectly balanced.

## 2 Description of our approach

We organized the implementation of the project according to the tasks:

1. Preprocessing of the dataset.
2. Define and learn models using the training data.
3. Design the validation method to evaluate the accuracy of the proposed classification approaches.

### 2.1 Preprocessing

In order to prepare the data for the model, the first step was to extract the training data from the text, and assign to the keywords the entities related. This process was made with a script, as it was a repetitive task.

Once the training data was prepared, the test data was chosen. The test data is selected, which in our case it is a ML related article.

Once it is all set, using *spaCy* functions for NLP, a pipeline is set, and the model is made out of scratch or loaded from an existing model.

### 2.2 Models

We use two classifiers:

1. NER 'blank' model:
   The model is made at the moment of execution of the python program, and it is trained once with the training data.
2. NER 'Over' model:
   This model has already been trained with the data, but as there is not much data, the process will be repeated several times, in order to train more this model.

### 2.3 Validation

To validate our results we compute the models accuracy in the test data and through implementation of the cross-validation method.

## 3 Implementation

Most of the project steps were implemented in Python, although the scripts used to extract and prepare the data were shell scripts. The main library used for NLP and NER was *spaCy*.

In order to parse text using *spaCy*'s NER module, we implemented the following algorithm:

1. Read or create a new model
2. Create the pipeline with NER
3. Train the model

   After training the model,
4. It can be tested with a given piece of text.
5. Optionally, it can be saved.

We also implemented Cross Validation methods so that we could test the model's performance.

We illustrate how the implementation works in our Python notebook: `https://github.com/initega/spacy-mlnn/blob/master/Spacy_Project.ipynb`.

### 3.1 Search for Overfitting

We tried to cause overfitting in order to see if that phenomenon can also be a problem on this topic. To do so, we ran the main program several times saving the output model and piping it into the next execution of the same program, always using the same piece of text as test set.

That was made with always 30 iterations, and on another try with scaling iterations (starting from 30 and ending at 150).

### 3.2 Cross Validation

In order to test the performance of our model, we used Cross Validation, specifically, Leave-p-out Cross Validation (LPOCV), where $p = 35$.

We cannot use Leave-one-out cross validation because, if we did so, we would be learning the whole training set to search for a keyword that can only be found using a few related samples on the training set. That means, it is basically a waste of time to learn the whole set if almost every sample on it is irrelevant to the keyword in issue.

For example, on this case, we have 245 text samples to learn from, 7 for each keyword, which splits the set on 35 groups of keyword samples.

That means, if we used Leave-one-out cross validation (LOOCV), the keyword on its sample of text used for validation in each iteration of the process would only be found because the other 6 samples of text used for learning that. However, the rest of samples on the training set (238) would not be used, as they have no information about the keyword cross validation method is trying to find on each iteration. So, for each keyword, only samples related to it are relevant.

So, our Cross Validation has 7 folds, each one of 35 samples with information about different keywords.

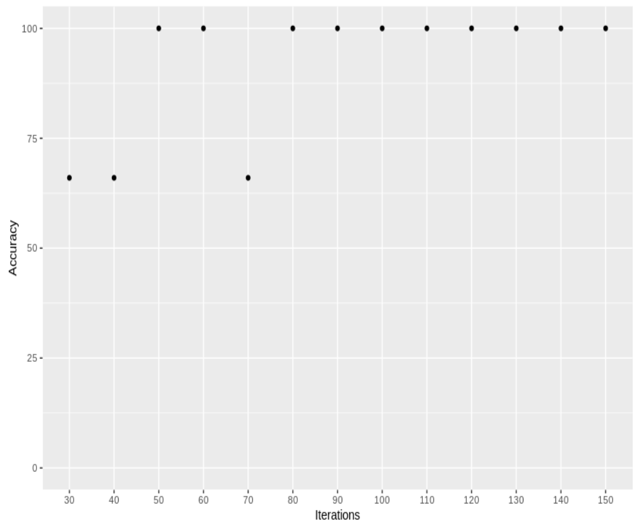
Furthermore, we used cross validation with 6 different iterations on the NER algorithm: 15, 30, 60, 100, 150, 400. The aim was to check the performance of the model, but also to check how it generalizes depending on the number of iterations on a set of 35 keywords.

## 4 Results

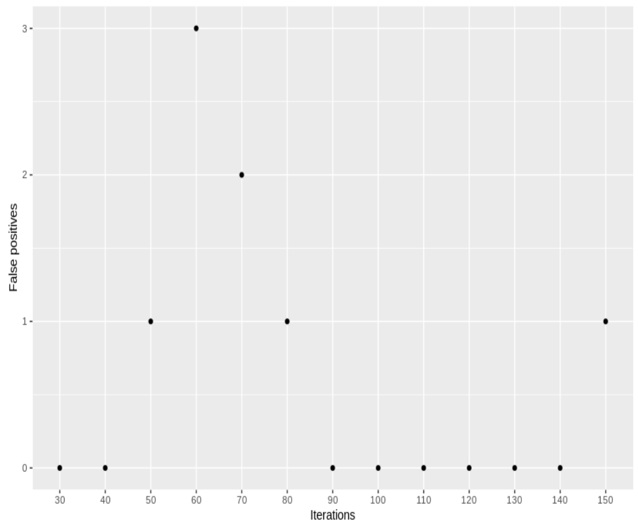
### 4.1 Search for Overfitting: with Incremental Iterations

To test our 'Blank' NER model, a test data will be used, in form of text, extracted from the abstract of one paper about ML implementations. To see the effects of the iteration count in the testing phase, accuracy and false positive count will be evaluated. In this case, accuracy is the ratio between the guesses made by the model against the true entities which are in the model. Besides, false positives will represent the times the model made a guess that it is not between the real entities of the model. This count is not considered inside our definition of accuracy, however, it is important to consider it while evaluating the general performance of the model.

On the one hand, as shown in figure **1a**, 'Blank' model performs relatively well in general, as the structure of the test data is not complex, although there is a correlation between the iteration count and higher accuracy, if figure **1b** is observed, it can be seen that there is a iteration count which makes the best performance, as training less can lead to the model to underperform, and training too much can have the same effect.



(a) Accuracy development as iteration count increases



(b) The amount of false positives given by the model as iteration count increases

Figure 1: NER 'Blank' model's performance in Test data

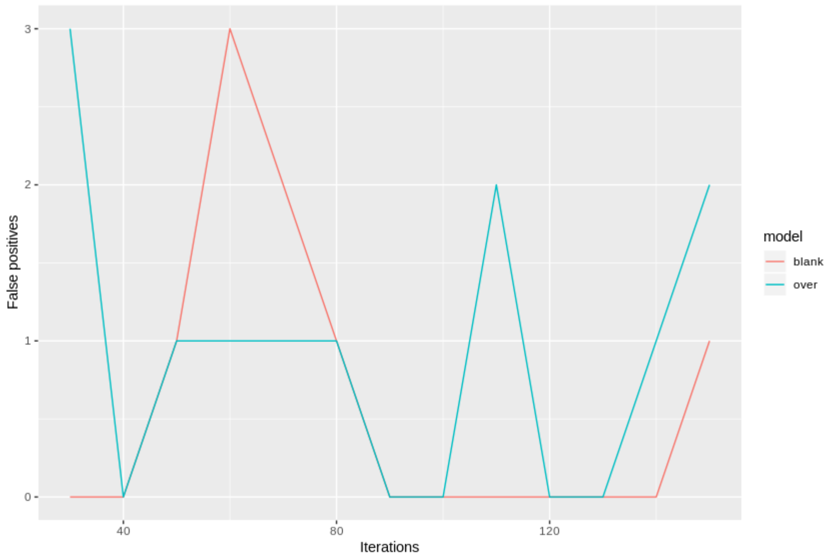


Figure 2: False positives for 'Blank' and 'Over' models

On the other hand, 'Over' model has had a better error while training, as it has been the model which has trained the most, despite the fact that it has been the same data all the the time.

As can be seen in figure **2**, there are several differences between both model's false positive counts. An explanation for this might be the excessive training of the 'Over' model, even though 'Over' model has a perfect accuracy score in this test data, the fact that is always training with the same train data can lead to that increased false positive count.

### 4.2 Search for Overfitting: without incrementing

We also tried the same method as before to check if overfitting was a thing when inputting the previously saved model and saving the learned one for the next execution.

We ran a loop of 80 iterations in order to check this, but the result was not what we expected. It seems like there is no evident overfitting through this method.

The result of using an over trained model was exactly or almost exactly the same as using an empty model on the same test set. Both trained and empty models found all or almost all entities on given test sets.

### 4.3 Cross Validation

As expected, the more iterations on the NER algorithm, the more overfitted it becomes. Then, the less iterations on the NER algorithm, the more generalized it becomes.

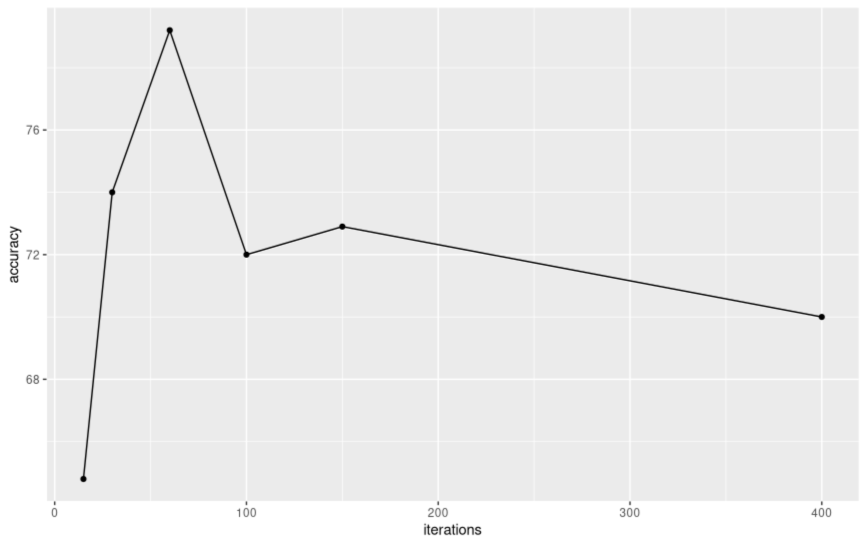


Figure 3: Accuracy obtained from Cross Validation with iterations 15, 30, 60, 100, 150 and 400

However, seems like on iterations below 30, because of not enough training, the loss function is too poor, making the model too generalized, making it unable to have a relevant accuracy.

As can be seen, the Cross Validation with 60 iterations is the best, but algorithms with surrounding iterations like 30 and 100 are also pretty good.

This behaviour's cause is that, when the NER algorithm reaches 30-60 iterations the loss function change from iteration to iteration is not too relevant, which means that the loss function is starting to converge.

It is also worth to mention that, in general, multi-word keywords are less likely to be found that single-word keywords.

What's more, this model is able to find half of the keywords used, generally, pretty easily, but it has problems with the other half.

## 5 Conclusions

*spaCy* is a great tool aimed at natural-language-processing in general. What's more, its Named Entity Recognition parser proved to be a great tool for finding entities throughout given text pieces.

Regarding Cross Validation, it is true that on our case, the results were not perfect, but that might be the result of having such small training data. 245 text samples seems good enough, but it must be taken into account that our task was to recognize 35 keywords, which means that, on overall, only 7 samples of text can be used for training each of them.

On a real situation, much more samples would be needed in order to get a high performance on any given document.

In any case, we saw signs that confirmed overfitting was a thing when testing never seen samples with high iterations. So that problem should be taken into account in a real situation.

Regarding the tests we made, we found out that overfitting was not such a problem as it was with the case of Cross Validation with 400 iterations. Here, results were still good even with an over trained model.

Our hypothesis about this phenomenon is that, when inputting the previously saved model into the next execution constantly doesn't generate overfitting, what it causes is what *spaCy* developers call "catastrophic forgetting". In other words, when inputting a model, the knowledge it had is overwritten in some way, so overfitting is not possible.

However, directly adding more iterations to the NER parser trainer makes the parser think that the huge amount of identical data it is receiving because of the high number of iterations should be taken into account, eventually causing overfitting.

All in all, we consider overfitting could be a problem, but just too many iterations are needed to reach that point. So, we agree on that 30-100 iterations are the best for this case, but we cannot confirm that would be the case for training sets of much more samples.

## References

[1] Wikipedia, *Cross-Validation*, `https://en.wikipedia.org/wiki/Cross-validation_%28statistics%29`

[2] Wikipedia, *Name-entity Recognition*, `https://en.wikipedia.org/wiki/Named-entity_recognition`

[3] spaCy, *Natural Language Processing library*, `https://spacy.io`

[4] GitHub, *The repository used for this report*, `https://github.com/initega/spacy-mlnn`

[5] Kevin P. Murphy, *Machine Learning, a probabilistic perspective*