Language Understanding Systems Course
Final project: Sequence Labelling using CRF and RNN

Marco Omezzolli, 185868, marco.omezzolli@studenti.unitn.it

Abstract

Extracting concepts from a text or a sequence of words is one of the fundamental steps while creating a language understanding system. In this project are presented several methods for labeling a word sequence by using conditional random fields (CRF) and recurrent neural network (RNN). Several approaches for both will be taken in account and a comparison with FST base approach will be presented.

1 Introduction

The scope of this project is to create a word tagger system for a movie domain. In order to perform this operation is necessary a training section on a sequence of sentences that already contains the IOB-tags and next using this train to predict the tags associated with words inside test set. The first section of this document describes the data composition and the words distributions inside train and test documents. In the second section instead are analyzed the different approaches and the result obtained, with errors estimation. The last section contains result comparison and an overall evaluation of the proposed method.

2 Data

The given dataset is called NLSPARQL and contains examples of phrases extracted from the movie domain. The entire dataset is subdivided into two principal classes: one for the train and the second one for testing. In each class we have two different types of document: first of all the .data that contain words and IOB-tags and the feats.txt files that instead contains Lemmas. Each row of the file contains that information in word-token format. For what concerns the IOB-tags each word label can be in one of the following formats: ”I-X”, ”BX” and ”O”, where B states for begin of a statement and I are the inner elements of that statement (for example a movie title). ”O” instead means not a statement, but a simple constitutive element of the phrase.

2.1 Dataset composition

As previous knowledge is important to understand the exact structure of the data and extract several statistical information about the train and test set. In this section are analyzed the numbers of phrases and words inside each document. For the train, we have 21453 words and 3338 phrases. Instead for test 7117 words and 1084 phrases.

2.2 Concepts distributions

Before looking at algorithm used to create the tagging is necessary to reason about the tag frequency. Considering the distribution in the train set we can observe that the O tag is inserted in the 71.74% of the cases so the concepts represent only the 28.26%. Here you can find a plot of the tags and their frequency.

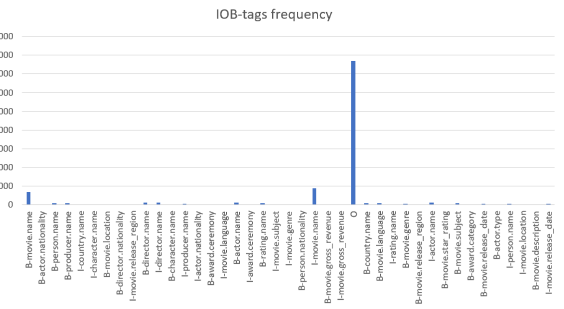


Figure 1: IOB-tags distribution in train

By excluding the ”O” tag from the plots is possible to see what are the most used concept tags. Instead inside the test set the percentage of words tagged as ”O” is 72.15%.

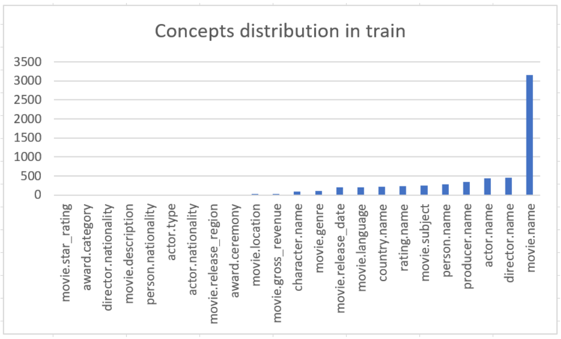


Figure 2: Concepts distribution in train

As before is also important take a look to the frequencies of concepts inside test file to understand what is the current distribution and what type of approach is necessary to improve the accuracy.

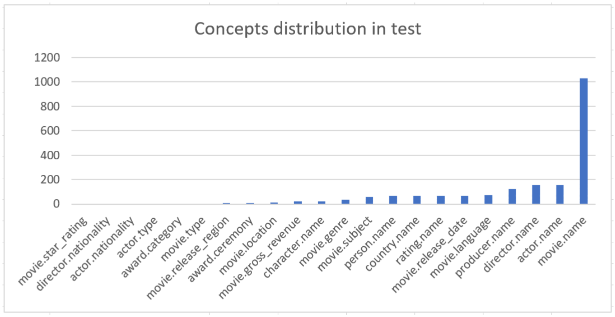


Figure 3: Concepts distribution in test

As is possible to see in *Figure 4* the tags distribution across train and test set is very similar. This is an important point because by analyzing the train set we can obtain also really useful information about the test and is not necessary perform normalization operations between the two.

2.3 Out of vocabolary words

In test set are present words that are not inside the train set and during the analysis are substituted with "unk" word. The percentage of those words out of vocabulary is 23.67%. The value is high and causes several errors during tagging operations. To perform RNN analysis the vocabulary was build by merging train and test, in order to not manage OOV terms.

3 CRF tagging

3.1 CRF Model

Conditional Random Fields are a set of discriminative probabilistic models that intend to calculate the conditional probability distribution of an example to belongs to a certain class(8). They perform this operation by apply the logistic regression to sequential inputs (words that belongs to a sentence). And this sequentiality needs to be evaluated by a specific function that takes in account the previous state and generate a binary result, based on previous and current word. To create the conditional field we can use the following formula:

$$P(y, X, \lambda) = \frac{1}{Z(X)} exp\left(\sum_{i=1}^{n}(\sum_{j}\lambda_j f_i(X, i, y_{i-1}, y_i))\right) \quad (1)$$

Where

$$Z(x) = \sum_{y^a \in y}\sum_{i=1}^{n}\sum_{j}\lambda_j f_i(X, i, y^a_{i-1}, y^a_i) \quad (2)$$

In this case λ_j is a set of weights associated to each feature function, that can be calculated using a maximum likelihood estimation. We can start by using the negative log of the distribution:

$$L(y, X, \lambda) = -log(\prod_{k=1}^{m} P(y^k|x^k, \lambda)) \quad (3)$$

And next derive respect to lambda to find the minimum value. The partial derivatives is then use for a single step in the gradient descent procedure until convergence is reached.

$$\lambda = \lambda + \alpha[\frac{\partial L(X, y, \lambda)}{\partial \lambda}] \quad (4)$$

In addition to that is possible to add features for each token to model conditional distribution in a more sophisticated way. This kind of approach is described in the next sections to achieve a best accuracy over the given set.

3.2 Baseline analysis

First of all a baseline analysis is proposed with CRF using a common templat that considers only unigrams and bigrams.

```
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,1]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]
U07:%x[0,0]/%x[0,1]

B
```

Table 1: Bigram template

With this method the following results are obtained: accuracy 95.50%, precision 92.45%, recall 79.74% and F1 85.63%

3.3 3-grams CRF

The base model was extended with 3-gram template and another column in the train set. Because of the fact that a CRF accepts for each token different features the insertion of a ”parent” features before the final one give us the possibility to increase the overall accuracy of the model.
The new train file merges the IOB-tags and lemmas to create macro-class for each word that next are specified in IOB-tags for output. In particular the process take the original IOB-tag (of the form I/B-X.Y) and split in two pieces based on the ”.”: for example, ”I-movie.name” is splitted in ”I-movie” and ”name”. After that the first element (in our case ”I-movie”) is augmented with the lemma for that word, obtaining a tag like this one: ”I-movie#NN”. For the O tag no division have been used, but only the augmented tag of the form ”O#Lemma”.
By using this method the most frequent tags have now a reduced frequency that allow the method to better discriminate between them, using a more complex features that separate better the words inside the document.
Results obtained are the worst so far: **accuracy 27.06%, precision 7.42%, recall 6.87%, F1 7.14%**, meaning that for CRF the over-specification on labels is not a good methodology to improve performances (it is clear that this set of features do not provide useful information to the building of the conditional random fields inside the model).
The template file used is the following one.

```
U00:%x[0,0]
U01:%x[1,0]
U02:%x[2,0]
U03:%x[3,0]
U04:%x[-3,1]
U05:%x[-2,1]
U06:%x[-1,1]
U07:%x[0,1]
U08:%x[1,1]
U09:%x[2,1]
U10:%x[3,1]
U11:%x[-2,1]%x[-1,1]
U12:%x[-1,1]%x[0,1]
U13:%x[0,1]%x[1,1]
U14:%x[1,1]%x[2,1]
U15:%x[2,1]%x[3,1]
U16:%x[-3,0]%x[-2,0]%x[-1,0]
U17:%x[-2,0]%x[-1,0]%x[0,0]
U18:%x[-1,0]%x[0,0]%x[1,0]
U19:%x[0,0]%x[1,0]%x[2,0]
U20:%x[1,0]%x[2,0]%x[3,0]
U21:%x[2,0]%x[3,0]%x[4,0]
B
```

Table 2: Trigram template

3.4 Trigram template for best analysis

The results obtained in the previous chapter have been used as starting point for thinking about what improvement can be inserted to make the accuracy higher. The template file presented in table 2 have been used on the untouched version of train data (composed by 3 columns with word, IOB-tag and IOB-tag) reaching the following results.

Acc.	**Prec.**	**Rec.**	**F1**
99.75%	99.26%	98.26%	98.76%

Table 3: Result for trigrams

This means that the over-specification presented in the previous chapter is not necessary and gives low level classification performances. The result presented in this last method are the best ones so far and represent a increase of F-score accuracy of 13% respect to the baseline CRF analysis (the reason behind is simply the fact that we keep a better tracking of the dependencies between words in sentences using a trigram approach instead of a bigram, creating a complex and more accurate model).

4 Recurrent Neural Network

4.1 Data preparation

Before training RNN on the dataset a step to create dictionaries and different sets for train and validation is needed. First of all is necessary create a dictionary of the words used inside the model. To perform this operation all the words present inside train and test have been taken. Another file was created by using the tags present inside the train and next the train was split in train and developer set. The second set is composed of 10% phrases of the original set and it is necessary for validate steps during the training.

4.2 The RNN model

Recurrent Neural Network (RNN) are a subclass of Neural Network, a particular machine learning technique that use node from a direct graph along a sequence(4). This specialization of NN uses recurrent connections: this means the input of a layer is feed by the output of the same at the previous time step. Principally exists two different models:

- Elman type: feeds the activation of neuron with the activation of hidden layer at previous step;
- Jordan type: use the output of the previous layer to feed the hidden layer;

In order to succesfully train those models is necessary to produce a modified version of the one-hot-encoding required by NN for input data. In this version the words are encoded in a vector that represent the entire vocabulary with all 0 values except one set to 1 that represent the position of the word inside the vocabulary. If we have a vocabulary of n words we create a vector $V = (x_1, ..., x_n)$ that index all the words: for example the "who" can be encoded as $V = (0, 0, 0, ..., 1, ..., 0)$.
The input of a RNN will be the word representation as described before and the output will be a vector of values that correspond to a class-score. Each node of the last layer will be one of the possible labels presented in the labels set obtained from training and the neuron with the higher activation value will be the prediction. In order to perform a faster and more reliable training the mini-batch approach was applied. Due to the fact that back-propagating weights for each instance of the train set will bring us to a computational overhead a set of words have been selected before updating (for example 100). This allow us to train the network on 100 examples and next update weights on the edges, instead of doing this operation for each example. Another consideration regards the size of the hidden layers that has to be as small as possible to not produce overfitting. In this cases a good rule is to set this parameter between the size of the input layer and the output layer, but the principle is not really applicable to our data because the input layer consist in 1973 possible words encoding and the output of 25, so a really large span. So the decision to keep it as low as possible to avoid the overfit problem. Values from 30 to 150 have been tested and finally 100 was chosen as the best ones.

4.3 Baseline analysis

In this section we will evaluate the best result obtained so far changing the principal parameters for training a RNN. In particular the optimal parameters that have been found were the following:

Learning rate	0.1
Window size	9
Batch size	7
Hidden nodes	100
Seed	3842845
Embed dimension	100
Num. Epochs	25

Table 4: Parameters for RNN

This configuration gives good results on the original train and test set. Here is the summary of the results:

Meth.	**Acc.**	**Prec.**	**Rec.**	**F1**
Elman	95.46%	81.39%	78.26%	79.79%
Jordan	95.14%	81.61%	76.97%	79.23%

Table 5: Result table for elman and jordan type

As is possible to observe the Elman method perform better than the Jordan one, looking to F1 measure. Thus they are very similar and still with a loss of precision F-score of 10% respect to CRF evaluated in the previous section. Inside the "results" folder is possible find other results for both Elman and Jordan type that have parameters changed. See the README file for details on how to interpret the file names.

4.4 Using LSTM

Long short term memory is a class of RNN that prevent the problem of the vanishing gradient and also allow the learning of long-term dependencies(7). The follow formula describe how the machine calculate the hidden state s_t:

$$\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= tanh(c_t) \circ o
\end{aligned} \tag{5}$$

In this case the input unit is represented by x_t at a certain time t and s_{t-1} represents the previous hidden state. The output of the function is clearly another hidden state that iteratively needs to be passed to the layer at $t+1$ if exists one. Also looking in the formulas we hate to analyze several elements:

- i, f, o are respectively the input, forget and output gate. Using the sigmoid we reduce every value from 0 and 1 and by the multiplication we can decide how much information carry to others layers. In particular the input gate tell us how much information from input we let pass. The forget gate instead decide how much we want to let pass from previous state and the output tell us how much the internal state can be exposed to obtain the output;
- g is a inner state, calculated using the current input and the previous hidden state;
- c_t is the memory of the unit. As is possible to see it combines the previous memory of the unit with the forget gate and add the information about the new hidden state g composed with the input state. We can say that is a combination of the previous memory with the new input.
- s_t is the hidden state output, that is calculated by multiply the memory with the output state.

The results obtained with the LSTM model using a window size of 11, to ensure a better capture of long term dependencies, are the following:

- accuracy: 94.63%
- precision: 75.14%
- recall: 75.14%
- F1: 75.14%

Starting from this model the convolutional version have been developed to ensure more accuracy to the trained model and also less pre-processing operations. The new version of CNNLSTM possess several differences from the original one (* represent the convolutional operator that can be calculated as a dot product between matrices):

$$\begin{aligned}
&f = \sigma(x_t * W^f + s_{t-1} * U^f + V^f c_{t-1} + b_f) \\
&i = \sigma(x_t * W^i + s_{t-1} * U^f + V^i c_{t-1} + b_i) \\
&o = \sigma(x_t * W^o + s_{t-1} * U^o + V^o c_{t-1} + b_o) \\
&c_t = f_t \circ c_{t-1} + i \circ \sigma(x_t * W^c + U^c * h_{t-1} + b_c) \\
&h_t = o \circ \sigma(c_t)
\end{aligned} \tag{6}$$

In this case V_f, V_i and V_o are weight matrix that are between the forget gate and the output gate, the cell state and input gate respectively. The results obtained by using this model are the following (using a window size of 11, to catch also long term dependencies):

- accuracy: 95.43%
- precision: 82.41%
- recall: 79.08%
- F1: 80.71%

The results for the window size equal to 9 are not reported. The only relevant thing to notice is that with this value there is an decrease of the F-score of $\sim 5\%$.

4.5 Using GRU

Gated Recurrent Units is quite similar to what a LSTM does and this can clearly be seen from its definition:

$$\begin{aligned}
z &= \sigma(x_t U^z + s_{t-1} W^z) \\
r &= \sigma(x_t U^r + s_{t-1} W^r) \\
h &= tanh(x_t U^h + (s_{t-1} \circ r) W^h) \\
s_t &= (1 - z) \circ h + z \circ s_{t-1}
\end{aligned} \tag{7}$$

In this case the model present only two gates, z that is an update gate and r, a reset gate. The reset tells us how to combine new input with previous

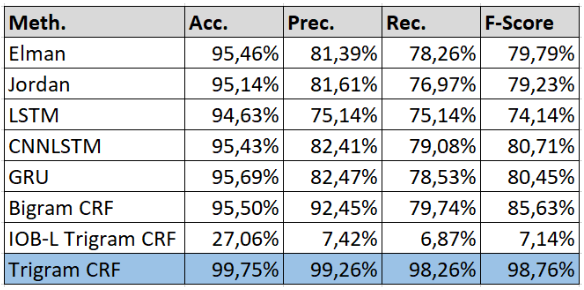
memory and the update tells us how much memory to keep.
Also in this case the code was modified to implement this particular type of RNN and the result obtained is the following one:

- accuracy: 95.69%
- precision: 82.47%
- recall: 78.53%
- F1: 80.45%

The main difference with a LSTM is the fact that GRU should perform better on small dataset(6). Thus in this case the two performs in a very similar way, with only a $\sim 0.3\%$ difference on F-score.

5 Resume on results obtained

In this section is presented a compact visualization of the results obtained with CRF and RNN. In the next chapter a comparison with the WFST model is presented as recall to the previous project.



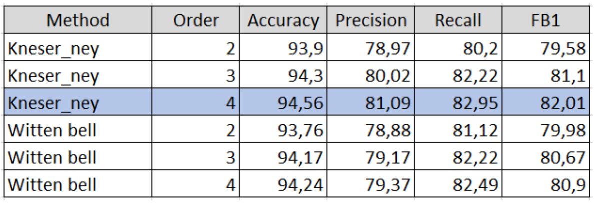
Meth.	Acc.	Prec.	Rec.	F-Score
Elman	95,46%	81,39%	78,26%	79,79%
Jordan	95,14%	81,61%	76,97%	79,23%
LSTM	94,63%	75,14%	75,14%	74,14%
CNNLSTM	95,43%	82,41%	79,08%	80,71%
GRU	95,69%	82,47%	78,53%	80,45%
Bigram CRF	95,50%	92,45%	79,74%	85,63%
IOB-L Trigram CRF	27,06%	7,42%	6,87%	7,14%
Trigram CRF	99,75%	99,26%	98,26%	98,76%

Figure 4: Results for RNN and CRF tests

In the table is marked the best overall method based on F-score. The decision to take the this metric instead of the precision is because it takes in account the unbalancing between the classes so it is more representative of the goodness of the model. As is possible to see CRF performs clearly better that all the types of RNN (except for the IOB#Lemmas features). In this last class of tests the Convolutional version of LSTM gives the best result so far and the differences between the classical LSTM and GRU are evident, with an improvement of $\sim 5\%$ of F-score.

6 Comparison with WFST

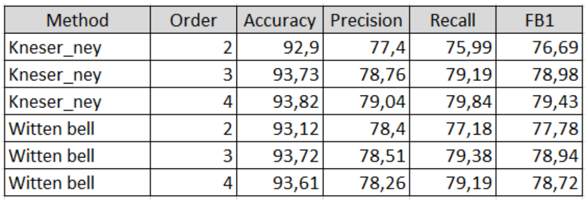
In the following table are reported the performances of weighted finite state automata presented in the previous project with the best accuracy and F-score obtained. To obtain those results



Method	Order	Accuracy	Precision	Recall	FB1
Kneser_ney	2	93,9	78,97	80,2	79,58
Kneser_ney	3	94,3	80,02	82,22	81,1
Kneser_ney	4	94,56	81,09	82,95	82,01
Witten bell	2	93,76	78,88	81,12	79,98
Witten bell	3	94,17	79,17	82,22	80,67
Witten bell	4	94,24	79,37	82,49	80,9

Figure 5: IOB tags partial exclusion results

the original high frequency tags (O, I-movie.name and B-movie.name) were replaced with a word version of the form IOBTag-word. Those performance are comparable with the ones obtained with LSTM approach using RNNs. Instead the basic version of RNN (jordan and elman type) provide results that are closer to the lemma insertion version, in which IOB-tags and lemmas are merged in the form IOB-lemma.



Method	Order	Accuracy	Precision	Recall	FB1
Kneser_ney	2	92,9	77,4	75,99	76,69
Kneser_ney	3	93,73	78,76	79,19	78,98
Kneser_ney	4	93,82	79,04	79,84	79,43
Witten bell	2	93,12	78,4	77,18	77,78
Witten bell	3	93,72	78,51	79,38	78,94
Witten bell	4	93,61	78,26	79,19	78,72

Figure 6: IOB-tag and lemmas test results

CRF instead perform sligtly better, introducing a performance improvement of 3% on F-score without any changes to the original dataset respect to WFST and around 5% respect to RNNs. With some refinement of the train set (changing the features form) the accuracy decrease, instead, by changing the feature function template is possible to reach an improvement of 16% respect to the WFST approach.

7 Conclusions

In conclusion several points have to be put under spotlight:

- RNNs have the worst performance overall: the reason beyond this behave could be the small number of training examples. Usually neural networks requires many data to train a consistent way, and in this case the dataset is not very big. Also choosing the correct parameters clearly influence the result, that has fluctuations from a F-score of 74% to 80%.

- LSTM perform the best in all the RNNs models tested. The reason beyond this is that long short term memory models delete the problem of vanishing gradient and can manage in a better way long dependencies between words inside the phrases. This is important inside a dataset in which the medium length of a sentence is six words.

- CRF shows that they clearly are the best model overall. With a little refinement of the training set that provides more features to the model is possible to achieve an overall increment of 16% respect to the WFST and around 18% respect to RNNs. Also the baseline analysis with CRF performs better than the others method tested.

8 Project references

The project is on Github at the following adress: https://github.com/MarcusOme/LUSProject2

9 References

(1)CRF++ https://taku910.github.io/crfpp/

(2)Theano http://deeplearning.net/software/theano/

(3)CRF https://en.wikipedia.org/wiki/Conditional_random_field

(4)Recurrent Neural Networks https://en.wikipedia.org/wiki/Recurrent_neural_network

(5)Long short term memory RNN https://en.wikipedia.org/wiki/Long_short-term_memory

(6)Gated Recurrent Unit https://en.wikipedia.org/wiki/Gated_recurrent_unit

(7)Understanding LSTM Networks https://colah.github.io/posts/2015-08-Understanding-LSTMs/

(8)Overview of Conditional Random Fields https://medium.com/ml2vec/overview-of-conditional-random-fields\-68a2a20fa541