Language Understanding Systems Course
Mid-term project: Sequence Labelling using Finite State Transducers

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 and using finite state transducers, smoothing algorithm and n-grams.

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 Words frequency and distribution

In this section is performed an analysis of the distribution of the data and in search if the words inside train follow a Zipf distribution. By analyzing the words with the higher frequency inside the text (only the ones that appears more than 100 times) we can obtain this graph.

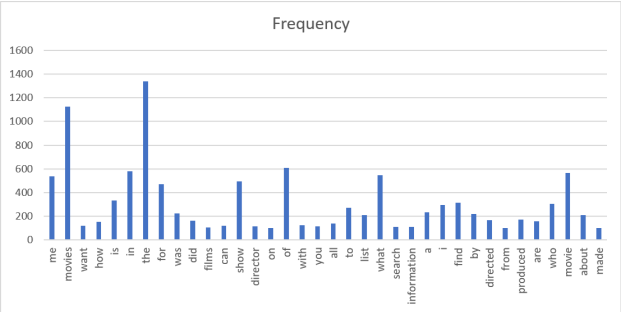

Figure 1: Distribution of most frequent words

It is clear that the text approximately follows the Zipf law. Lower rank words (the ones that have less letter inside, like "the" or "of") appears with higher frequency than high-rank words (for example "information"). Because we are working inside a specific context we also have context-specific words that appear with high frequency, like "movie" or "directed".

2.3 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.

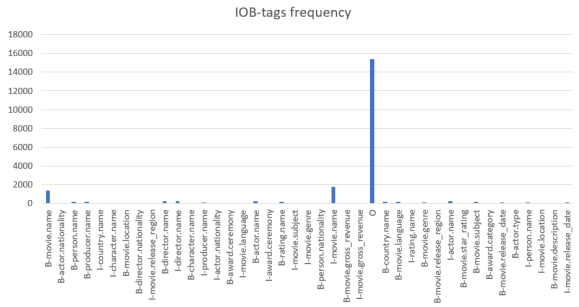

Figure 2: IOB-tags distribution in train

By excluding the ”O” tag from the plots is possible to see what are the most used concept tags.

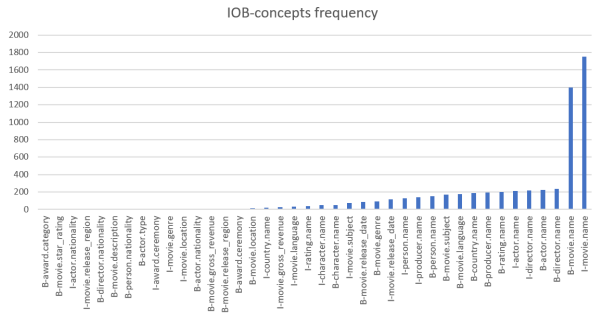

Figure 3: IOB-concepts distribution in train

Instead inside the test set the percentage of words tagged as ”O” is 72.15%. In figure n is possible to see the plot of the distribution of elements.

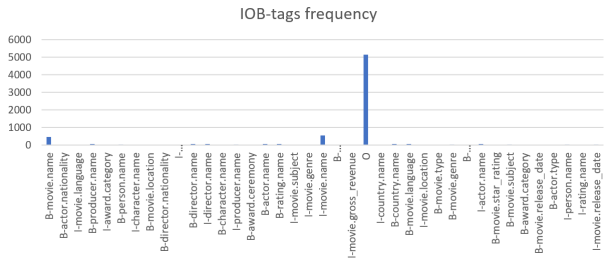

Figure 4: IOB-tags distribution in test

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.

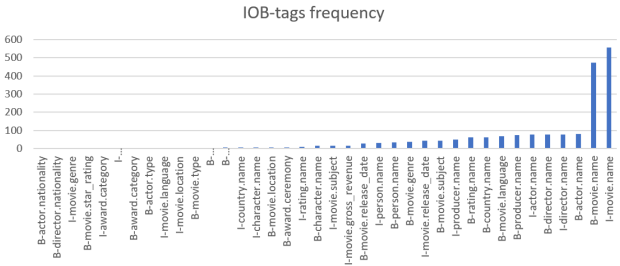

Figure 5: IOB-concepts distribution in test

As is possible to see 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.

2.4 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.

3 Finite state transducers for tagging

In this section, the methodology and partial results are reported for the different approaches used during the analysis. Inside the project, folder is possible to find the complete code for each section or method described in this document.

3.1 Tools

In order to create this project two principal tools where used:

- OpenGRM: a collection of open-source libraries for constructing, combining, applying and searching formal grammars and creating n-grams models as FSTs
- OpenFst: a library to constructing, optimizing and searching weighted finite-state transducers (FSTs)

3.2 The main idea

By using FST is possible to convert a sequence of words inside a tags sequence. By learning a FSM (Finite State Machine) starting from a train set is possible to combine several elements to obtain, for each sequence, a tag representation: starting from

a phrase S and its conversion in tags λ_s is possible to create a decoder in the following way:

$$\lambda_s \circ \lambda_{tagger} \circ \lambda_{SCLM}$$

Where λ_{tagger} is the concept tagger that associate to the words each possible tag with a certain probability and λ_{SCLM} is the concept model, created using OpenGRM.

3.3 Implementation

First of all a lexicon is generated, containing all the words present inside the train set. Also "¡unk¿" is inserted in order to manage unknown words. Notice that all the single quote characters have been replaced with a "%" symbols to avoid format problem in strings while running the model.
After that in order to generate the λ_{tagger} machine is necessary calculate the probability of $P(W_i|t_i)$. To obtain that for each word and its tag was used the following formula:

$$P(W_i|t_i) = \frac{count(W_i, t_i)}{count(t_i)}$$

Where $count(W_i, t_i)$ is the numbers of time that the words W_i appears with tag t_i and $count(t_i)$ is the total time the tag appears in the train set.
After calculating this is possible to proceed with the creation of the λ_{tagger} by inserting a single state 0 in which each edge contains the original word and the probability to generate exactly the t_i tag. Instead, for the unknown words (see section 2.4 for more details) to λ_{tagger} are added several edges in which the probability is equal to:

$$P(W_i|t_i) = \frac{1}{|Concepts|}$$

With this addition is possible to manage also the words that are not present in train set and need to be converted into a tag by the FST.
At this point is possible to create λ_{SCLM} using a certain n-gram order(in this project only order 2,3 and 4 where used) on the tagged version of the train set (each word replaced with the given tag). Also different smoothing algorithms can be used: witten bell, kneser ney, presmoothed, absolute. The only method that have a particular behave is the unsmoothed, that requires unigrams to perform. Clearly choose one of the possible methods will result in significantly changes to the accuracy obtained.

After that is possible to create the FST representation of the string(λ_s) and compose it with the λ_{tagger} and next with the λ_{SCLM} to obtain the final prediction for the string S. The shortest path of this transducer will then be saved to perform later accuracy analysis.

3.4 Baseline analysis

This represent the first and most simple test calculated with all the smoothing algorithm and all the possible orders, to obtain some base values to use in order to compare them with more specific methodologies that are explained in the next section. The methodology to perform this analysis is the same explained in the previous section and unigrams and superior order n-grams are used in combination with smoothing algorithm to obtain those results.

Method	Order	Accuracy	Precision	Recall	FB1
Unsmoothed	1	88,9	54,63	60,04	57,21
Absolute	2	92,69	78,51	74,34	76,37
Absolute	3	92,65	76,67	74,7	75,67
Absolute	4	92,85	77,16	75,25	76,91
Katz	2	92,62	78,03	73,88	75,89
Katz	3	92,09	75,6	72,69	74,11
Katz	4	91,88	72,57	73,97	73,26
Kneser_ney	2	92,68	78,41	74,24	76,27
Kneser_ney	3	92,64	76,67	74,7	75,67
Kneser_ney	4	92,78	76,87	75,53	76,19
Presmoothed	2	92,65	78,41	74,24	76,27
Presmoothed	3	90,74	64,81	71,4	67,95
Presmoothed	4	89,97	62,28	72,5	67,01
Witten bell	2	92,68	78,51	74,34	76,37
Witten bell	3	92,62	76,58	74,61	75,58
Witten bell	4	92,86	77,11	75,34	76,22

Figure 6: Baseline tests

As is possible to see the Witten bell method and the Kneser ney are the most accurate, especially for most high values in orders.

3.5 Cutoff frequency

In order to improve the accuracy for the prediction a test using frequency cutoff is used. In particular all the elements that appears only one time inside the train set where excluded and their probability normalized to 1 (the minimum possible value) and those are the result obtained (in the following table are reported results only for Witten bell and Kneser ney, for the complete list of accuracy you can inspect the file *result_cutoff.txt* placed in *Final_result* folder).

Method	Order	Accuracy	Precision	Recall	FB1
Kneser_ney	2	89,67	62,79	69,75	66,09
Kneser_ney	3	90,36	64,11	71,22	67,48
Kneser_ney	4	90,42	64,19	71,13	67,48
Witten bell	2	90,94	66,87	71,4	69,06
Witten bell	3	91,6	68,62	74,15	71,28
Witten bell	4	91,56	68,56	74,15	71,25

Figure 7: Cutoff test

As is possible to see there is not any improvement respect to the baseline. Also by changing the code and setting a cutoff that is higher than the one that act on frequency one words is possible to observe a bigger decrease of the accuracy for all the methods with any order.

3.6 O tag elimination

One of the problems already presented in section 2.3 is the fact that the ”O” tag have a much bigger probability respect to the others tags. For this reason the ”O” were totally eliminated and substituted with ”O–X” tags, where X represent the original words associated with the tag. In the following table you can see the results of apply this method.

Method	Order	Accuracy	Precision	Recall	FB1
Kneser_ney	2	93,44	77,78	79,65	78,66
Kneser_ney	3	94,31	79,77	82,03	80,89
Kneser_ney	4	94,45	80,81	82,58	81,69
Witten bell	2	93,48	77,69	80,75	79,19
Witten bell	3	94,01	78,57	82,31	80,39
Witten bell	4	94,06	78,52	82,4	80,41

Figure 8: O tag exclusion test

In this case is present a little improvement respect to the baseline test. The best one is the one marked in the table that bring a total gain of 4.67 of FB1.

3.7 Using lemmas

Because of the fact an improvement is noticed in the previous test Lemmas are used in order to specify better the tags for each word in training set. For this step the IOB-tags are merged with the lemmas to create a tag of the form ”IOB–Lemma” and the train is executed on this new train set.

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 9: IOB tags and lemmas

As is possible to see there is little improvement respect to the baseline test, but the results are less precise than the ones see in the previous section.

3.8 Words and IOB-tags

Because the accuracy was high in section 3.6 more considerations where done to try to achieve also a better result. If by eliminating high frequency tags of only a type the overall precision grow maybe extend this method to other IOB-tags will bring to better results. For this reason the *Figure 3* information were used to detect the high frequency concept tags. As is possible to see the *I-movie.name* and *B-movie.name* are the ones most probable inside the train set and also in test set. So a tag replacement have been done, in order to transform them in ”I-movie.name-X” and ”B-movie.name-X”, where X represent the word associated with the tag. Following this idea, that also gives better results than the method proposed in 3.6 (increment of accuracy of 0.10% and FB1 of 1-1.5%) the result obtained are the following:

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 10: IOB tags partial exclusion

For the complete result table for all different methods and orders is possible to consult *Final_results/IOB_exclusion.txt*.

4 Conclusions

In this project different methods have been tried and different results have been obtained.

- The best smoothing algorithm are Kneser Ney and Witten Bell. They have more accuracy respect to others and the FB1 is also higher.
- Higher orders are better, and around 3 or 4 in all the methods is possible to observe a 1/2% improvement in precision and FB1.
- Frequency cutoff do not improve performances.
- Using lemmas do not provide better results, instead lower precision and FB1 by 2-4% and 5-10% respectively.
- By inserting some differences inside the ”O”, ”I-movie.name” and ”B-movie.name” tags is possible to see a good improvement of precision and FB1, that lead to the best method evaluated in this project.

5 Project references

The project is on Github at the following adress: https://github.com/MarcusOme/LUSProject1

6 References

OpenGrm http://www.openfst.org/twiki/bin/view/GRM/NGramLibrary

OpenFst http://www.openfst.org/