Language Understanding System: Mid-term Project

Montagner Andrea
andrea.montagner@studenti.unitn.it

Abstract

The aim of this project is to develop o Spoken Language Understanding module for the Movie Domain capable of performing PoS tagging. In order to achieve this goal a language model is firstly created and then trained to predict unseen utterances from a testing pool of data. In the following sections multiple approaches with respective solutions are presented, each one evaluated in term of accuracy, precision, recall and f1 score. Finally these results are compared to each other in order to determine the best approach to this problem.

1 Introduction

A Concept Tagger is a module which is capable of assigning a concept, in this case a tag, to words belonging to a sentence. This operation, though, requires the presence of a dataset composed by couples (words, tag) large enough in order to have a sufficient number of utterances to perform both the training and testing phase.
In particular, the material that was provided for this project was:

- A train dataset, both for the Movie Domain and for the additional features,
- A test dataset, both for the Movie Domain and for the additional features,
- A Perl script, "`conlleval.pl`", to evaluate the performances in terms of accuracy, precision, recall and f1 score.

2 Data Analysis

The provided dataset is the Microsoft NL-SPARQL dataset and it was already split into two parts: one for training and one for testing. In addition, each of these groups of data was divided into main data for the Movie Domain and additional features. The latter contains the lemma and the PoS (Part of Speech) tag for each word.
To summarize, what was given were four files:

- **NL-SPARQL.train.data** containing a two columns set of data, tab separated, where the first column represent the words and the second tags,
- **NL-SPARQL.train.feats.txt** containing a three columns set of data, tab separated, where the first column represent words, the second PoS tags and the third lemmas,
- **NL-SPARQL.test.data** containing a two columns set of data, tab separated, where the first column represent words and the second tags,
- **NL-SPARQL.test.feats.txt** containing a three columns set of data, tab separated, where the first column represent the words, the second tags and the third lemmas.

In these files, each line contains a word (with the respective tag/lemma) of a sentence and the end of each sentence is marked with an empty line as delimiter. (2).

2.1 Data Distribution

A deeper analysis, instead, shows how both IOBs and words distributions follow an empirical statistical law, called *Zipf's Law*, for which the frequency of a word is inversely proportional to its rank in the frequency table.

The next two subsections will illustrate a more detailed analysis of the data, focusing individually first on IOBs and then on words.

who	O
plays	O
luke	B-character.name
on	O
star	B-movie.name
wars	I-movie.name
new	I-movie.name
hope	I-movie.name
show	O
credits	O
for	O
the	B-movie.name
godfather	I-movie.name

Table 1: An example of the structure of train data

who	WP	who
plays	VVZ	play
luke	NN	luke
on	IN	on
star	NN	star
wars	NNS	war
new	JJ	new
hope	NN	hope
show	NN	show
credits	NNS	credit
for	IN	for
the	DT	the
godfather	NN	godfather

Table 2: An example of the structure of feats data

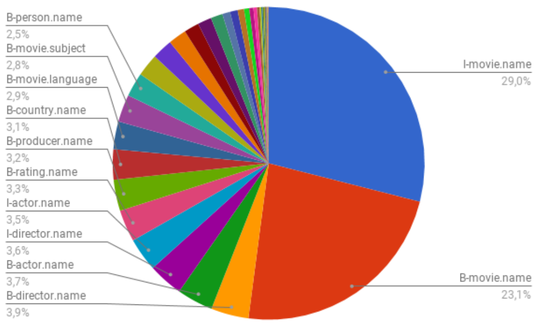
2.1.1 IOB Definition & Distribution

Concept Tags are represented using an *IOB notation*, which is a common tagging format when dealing with chunking tasks like identifying different parts of a string.
The used prefixes are:

- **B** - Beginning of a span
- **I** - Inside of a span
- **E** - End of a span
- **O** - Outside of a span

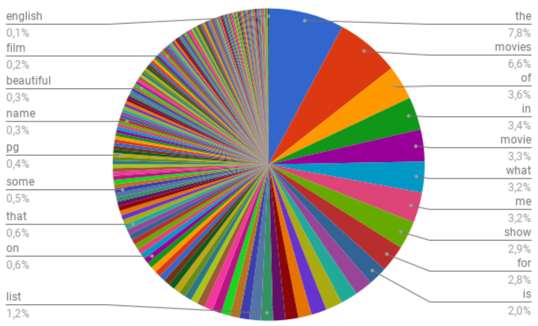
As long as IOBs distributions is concerned, the analysis mirrors the prediction and outputs as the most frequent tag `movie.domain`, divided in `I.movie.domain` and `B.movie.domain`.

In this analysis the tag `O` is not taken into consideration because it can be considered a ”stop-word” for tags.



2.1.2 Words Distribution

Almost all of most frequent words are identified as stop words, like `the` (1337), `of` (607), `in` (582). Nonetheless there are exceptions, like `movies` (1126) and `movie` (564), which are relevant to understand the topic of the considered dataset.



3 Tools and Implementation

The whole project was developed using Bash and Python (v2.7) as programming languages. In addition, two external tools were provided, in order to perform specific actions (*i.e.* creation of the transducer and of the language model). This tools are:

- **OpenFST** library (fst)
- **OpenGrm** Ngrm library (ngr)

All tried training algorithms are performed for 5 different levels of ngram order (1 to 5) and for each smoothing method offered by OpenFST, namely *absolute*, *katz*, *kneser_ney*, *presmoothed*, *unsmoothed*, *witten_bell*.

The case of a transducer words to IOBs will be considered as baseline method for this project and its results compared to other approaches to evaluate which one best suites the problem.

3.1 Baseline method

This first attempt takes into consideration only the basic data files, containing words and concepts. The idea is to see concepts as classes to be predicted and train the model using the probability of words given concepts. For this reason, the best approach seemed to be the *Bayesian Decision Rule* to compute the Maximum Likelihood estimation:

$$P(word \,|\, IOB) = \frac{C(word,\, IOB)}{C(IOB)} \quad (1)$$

Where $C(word,\, IOB)$ is the number of occurrences of the couple (word, IOB), and $C(IOB)$ is the number of total occurrences of that IOB. However, since the final output will be an automaton, probabilities are not the best value to estimate moves between states. For this reason, instead of the Maximum Likelihood, it is used its negative logarithm:

$$Cost(word \,|\, IOB) = -log(P(word \,|\, IOB))$$

Unknown words are handled assigning to all concepts the same probability to generate them. In this way, the costs became:

$$Cost(< unk > \,|\, IOB) = -log(\frac{1}{\#IOBs}))$$

3.2 Other approaches: different training methods

3.2.1 Cut-off

The first idea to improve the accuracy of the prediction was the insertion of a *frequency cut-off* value on words not to take into consideration all those words (with their respective IOB) whose frequency lay below a certain threshold (empirically set to 50). With this add-on, each method (both the baseline and the once explained below) could be tested with or without this threshold in order to have a more accurate idea on how evaluation changes.

Interesting to report is how unknown words are handled. Unlike the baseline solution, here assigning the same probability to all the concepts would not be right. The reason is that the number of times an IOB is deleted in known in advance, giving the possibility to perform a more accurate computation of the probability. When using a cut-off, the steps are:

1. Compute the probability of each IOB as:

   $$P(IOB) = \frac{C(IOB)}{\#IOBs_{tot}}$$

   where $\#IOBs_{tot}$ is the total number of occurrences of each IOB in the dataset, after the removal of the words with low frequency.

2. Assign $1 - P(IOB)$ to the unknown word:

   $$P(< unk > |IOB) = 1 - P(IOB)$$

   In this way it is possible to assign a value that is proportional to the number of times an IOB is deleted because below the threshold.

3.2.2 Lemma2IOB

The second attempt for trying to boost the performances was using lemmas instead of words. Lemmas can be found in the additional features files (2). The Maximum Likelihood is computed as follow:

$$P(lemma \,|\, IOB) = \frac{C(lemma,\, IOB)}{C(IOB)} \quad (2)$$

As can be seen, this method is very similar to the baseline, just replacing words with lemmas. Probabilities are then transformed into costs as before (negative log likelihood), and unknown words are handled with the same policy.

3.2.3 O-removal

Another attempt to enhance the performances was trying to remove the `"O"` tag from the set. The reason is that it brings no valuable information (as stated before, can be seen as a ”stop word” for tags) since most of words are tagged as ”out of span”. The idea was to replace it with the word itself (*i.e.* (show, O) became (show, O-show)). With this little trick the model gains a lot of more information about other tags allowing a better training phase and consequently better results in the prediction.

This method was applied only to the baseline Word to IOB solution. The structure is the same as before, both for probabilities computation and handling of unknown words. The only difference is a modified training set where there are no `"O"` tags

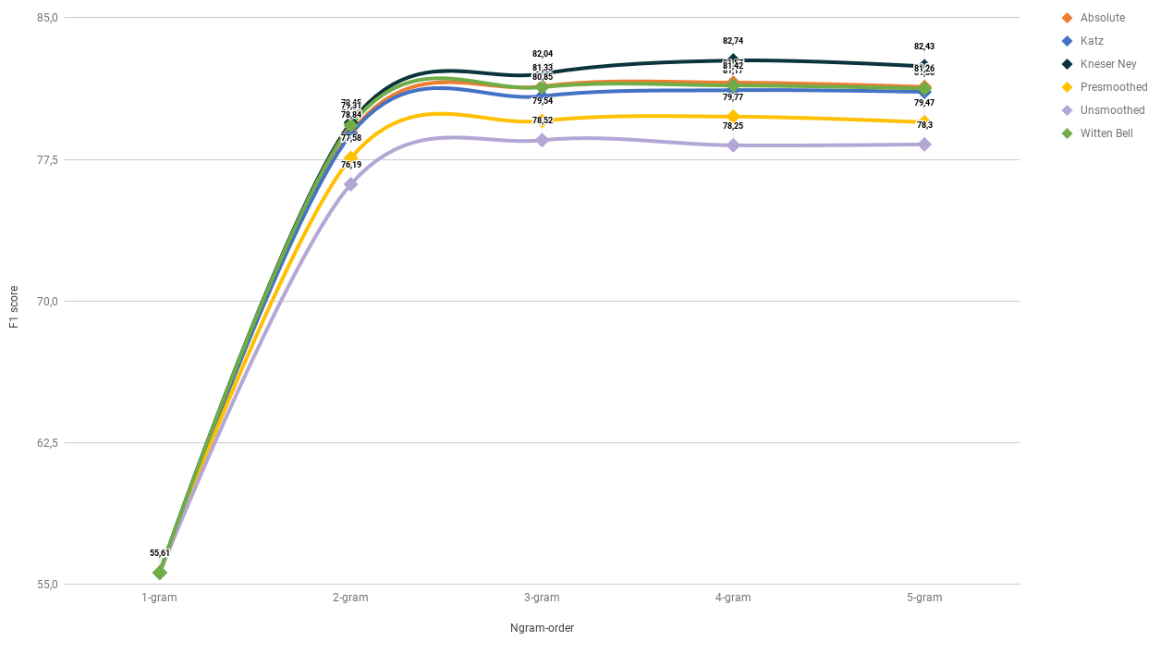


Figure 1: F1 scores for the Word to Concept method without 'O'

4 Evaluations Analysis

Evaluations in performed by a Pearl script, `conlleval.pl`, which evaluates four different scores: *Accuracy*, *Precision*, *Recall*, *F1*.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1[1] = \frac{Precision * Recall}{Precision + Recall}$$

4.1 Word2IOB Evaluation

Analyzing the output of the evaluation script for the baseline approach, best performances are:

- Accuracy: $92, 68$
- Precision: $78, 51$
- Recall: $74, 34$
- F1: $76, 37$

[1]This is a simplified version of the F1 Score formula where $\beta = 1$, the complete formulation is:

$(1 * \beta^2)\frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$

and correspond to the *witten_bell* method with *bigrams*. Similar values are found using *bigrams* with the *kneser _ney* method. These values drop drastically when considering unigrams, while float around the same value for ngram order from 2 to 5. More in general, it is not the best approach to solve this problem, there are solutions which present better performances.

4.2 Cut-off

The insertion of a cut-off value as threshold was an experiment that did not give the expected results. For this reason, it was tested only on the baseline solution. Performances drop a lot throughout all the methods, independently from which ngram order is used. Just as a report, best performances are:

- Accuracy: $71, 73$
- Precision: $731, 91$
- Recall: $32, 391$
- F1: $32, 40$

and correspond to the *unsmoothed* method with *4-grams*.

4.3 Lemma2IOB Evaluation

Also this approach, like the use of a cut-off frequency, does not give good results as output. Very likely the reason lay in how the probability is computed, which may not be the best way. Also here, just as a report, best performances are:

- Accuracy: $89,59$
- Precision: $71,17$
- Recall: $60,86$
- F1: $65,61$

and correspond to the *witten_bell* method with *bigrams*.

4.4 Word2IOB without 'O'

This is the last approach that has been tried and the one who actually gives the best results at all. Performances, with this method, rise of almost 20% as far as F1 score is concerned. Best values are found for *Kneser Ney* method using 4-grams. More in details, values are:

- Accuracy: $94,96$
- Precision: $82,44$
- Recall: $83,04$
- F1: $82,74$

Comparing these values to the ones of the baseline method, it can be seen that the worst value of this approach, that is $76,19$ using *Unsmoothed* with bigrams (unigram values are not taken into consideration due to the fact that are too out of range to be comparable) are very similar to the best results of the baseline solution. As already mentioned, the reason of this sharp raise is that, increasing the generalization with the removal of the `'O'` tag, the system gain the capability learning more information about other concepts.
Figure 1 shows the complete evaluation of these results, for all considered smoothing methods, each of them with ngram order from 1 to 5.

5 Conclusions

In this project different methods develop a Spoken Language Understanding module for the Movie Domain are presented. Each of them has been tested and compared to the one taken as 'Baseline Solution', namely the Word2IOB approach. After a pre-processing phase where data have been organized so to be able to be used in he training, model has been first trained and then tested on different datasets. As mentioned before, best solution are given from the Word2IOB without 'O' approach, it allowing more generalization of the tags. In future, more approach could be performed, using also PoS Tags or combining words with lemmas. The issue would be to individuate a good formulation of the computation of the probability, which could maybe lead to better performances.

References

OpenFST. `http://www.openfst.org/twiki/bin/view/FST/WebHome`.

OpenGrm - Ngrm. `http://www.opengrm.org`.