Language Understanding System (145299) - First Assignment

Massimiliano Luca
Univeristy of Trento
massimiliano.luca@unitn.it

Abstract

This document shows the approach to the first assignment of the Language Understanding System class. The goal was to develop a system language understanding (SLU) for the movie domain. To achieve the best result, some solutions have been implemented, tested and are discussed in this report.

1 Introduction

In this document, the pipeline followed to develop the SLU module for the first assignment is exposed. Furthermore, results are analyzed and displayed. Two models are described: the basic model that generate a final state transducer by using some generic features (POS-tag, lemma, IOB-tag and words) and an advanced model that takes care about meta information associated to O IOB-tags. In the last section the performance metric of the two model are briefly discussed.

2 Data Analysis

The dataset used is named NL-SPARQL and it contains sample regards the movie domain. It has already been split into training and testing sets. Furthermore, it has been divided into two instances: the basic one in which each word is associated to a IOB-tag and the one with additional features in which each word is linked to a PoS-tag and to its lemma. Particularly, the PoS tagging follows the Penn Treebank tags. Regarding the IOB tagging system, each token is associated to O, B or I. B indicates the beginning of a phrase, I follows a B tag and annotate that the phrase is continuing, O means that the token is not part of a phrase.

2.1 Dataset Size

The first analysis of the dataset concern the dataset size. Relevant numbers can be found in *Table 1*

Table 1: Dataset size

Source	Words	Unique Words	IOB-tags
train	21453	1728	41
test	7117	1039	39

2.2 Zipf's Law

As it can be seen in *Figure 1*, the training set follows the Zipf's law. Lower rank words such as "the" are the most frequent words in the dataset. As it is expectable, some of the most frequent words are associated with the movie domain i.e., "movie" and "movies" are the second and the fifth most used tokens.

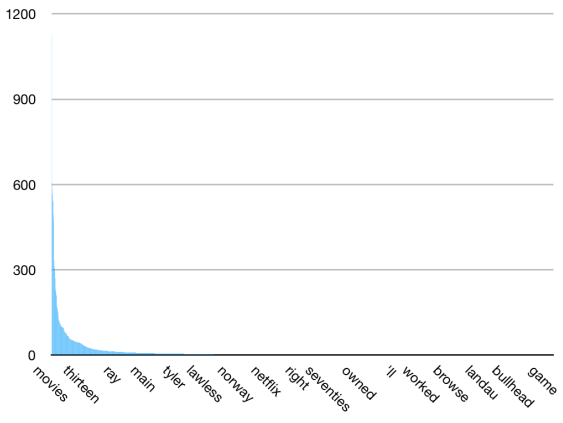

Figure 1: Concept distribution

2.3 Stop-words

Some stop-words have been identified. This step of the analysis could be helpful to improve performances by filtering them out before doing the

classification. The most common stop-words in the dataset are

- the (1337 times)
- movies (1126 times)
- of (607 times)
- in (582 times)
- movie (564 times)

2.4 Concept Distribution

In the training set, only the 28.25% of the words have a tag which is different by "O". By looking at the test set, this percentage falls to 27.84%. In both the dataset, the most common concept is "movie.name" which, for what concern the training set, appears in the 52.07% of the time. The other concepts and the related frequencies are listed in *Table 2*.

Table 2: Concept distribution

Conpet	**Training**	**Test**
movie.name	3157	1030
director.name	455	156
actor.name	437	157
producer.name	336	121
person.name	280	66
movie.subject	247	59
rating.name	240	69
country.name	212	67
movie.language	207	72
movie.release_date	201	70
movie.genre	98	37
character.name	97	21
movie.gross_revenue	34	20
movie.location	21	11
award.ceremony	13	7
movie.release_region	10	6
actor.nationality	6	1
actor.type	3	2
director.nationality	2	1
movie.description	2	0
person.nationality	2	0
movie.star_rating	1	1
award.category	1	4
movie.type	0	4

In other terms the out-of-vocabulary ration is 23.676%. Particularly, "movie.type" can be found only in the test set and, "movie.description" and "person.nationality" are not present in the test set. Another interesting fact that emerges from *Table 2* is that the most common concepts in the training and test sets are very similar and follows the same distribution

3 Model

The goal of the project is to map words to concepts. Given a sequence of tag $(t_1, ..., t_n)$ and a sequence of words $(w_1, ..., w_n)$, the probability of the sequence can be computed as

$$P((w_1, ..., w_n)|(t_1, ..., t_n)) = \prod_{i=1}^{n} P(w_i|t_i)$$

where, using a maximum likelihood approach,

$$P(w_i|t_i) = \frac{C(w_i, t_i)}{C(t_i)}$$

Particularly, $C(w_i, t_i)$ is the frequency of the tuple $< w_i, t_i >$ and $C(t_i)$ is the frequency of t_i. Furthermore, $t_1^n = (t_1, ...t_n)$ and, by treating it as a Markov process, a Markov assumption can be introduced and so it can be stated that the probability of a specific tag of in the sequence t_1^n depends only from k previous tags. Note that by stating this, the concept of n-gram can be introduced in the model. An assumption of order k can be represented as a $(k+1)$-gram. Formally,

$$P(t_1, ..., t_n) = \prod_{i=k}^{n} P(t_i|t_{i-m}, ..., t_{i-1})$$

Given all these assumptions, the original model

$$(t_1, ..., t_n) = \underset{t_1,...,t_n}{\operatorname{argmax}} \sum_{i=1}^{n} P(w_i|t_i) \sum_{i=m}^{n} P(t_i|t_{i-m}^{i-1})$$

From the previous formula emerges that the model can be split in two parts. In fact, $\sum_{i=1}^{n} P(w_i|t_i)$ represent a word to concept transducer while $\sum_{i=m}^{n} P(t_i|t_{i-m}^{i-1})$ is a language model for the concepts.
The first term takes care of

- add transactions to the automata
- compute costs
- take care of OOV words

Costs are computed as

$$c(w_i, t_i) = -log(P(w_i, t_i))$$

if the word is in the vocabulary. Otherwise, *¡unk¿* is defined as the OOV concept and the cost is computed as

$$c(< unk >, t_i) = -log(\frac{1}{|concepts|})$$

In both the cases, "log" has to be considered as the natural logarithm
The second term is a language model for the concepts. It takes a list of concepts as input and it computes a probability (the cost) based on the other concepts that the model found in the training set. Data sparcity comes as a problem: as far as the dataset is limited, most of the sequences are not observed an so most of them will have a 0-probability. As it can be seen in the model definition, this implies that also most of the word sequences will have the probability equals to 0. To solve this issue, smoothing must be used. In the next section, some smoothing method are used and results are compared.

Finally, by using *openfst*, each sentence is compiled into a final state acceptor and then, the most probable $(t_1, ..., t_n)$ is associated to the shortest path of the final state transducer. A more advanced model should take care of the meta-information of the words associated with an O-tag. With a very small improvement, this information can be preserved. The aim is to include the word directly in the O-tag by creating a new temporary tag in the form $O- < word >$. To do this, another final state transducer must be added to the pipeline in order to map concepts to IOB-tags. This new model can be used to improve results as shown in section 4.

4 Performance Evaluation

The baseline can be computed by considering a model without smoothing. Different n-gram orders (ngo in the tables) can be applied to look at differences. In *Table 3*, the results are listed. It emerges that the baseline is $F\beta = 57.1\%$ and is achieved using an unsmoothed model without using ngrams (and considering a single word).Better results are achieved by increasing the ngram-order. Particularly, by using 2grams, the $F\beta$ score jumps to 76.2% By changing the smoothing function, results does not change a lot. In *Table 4*, only results concerning 2grams are shown. The unsmoothed method achieved similar results with 4grams and 5grams but, as for 2grams the F1 score does not change significantly. Other results can be found at (Github). For all the ngram-orders, the best results are produced by the Witten-Bell smoothing method. In all the cases, the F1 score is increased of the 20% with respect to the baseline. By using pos-tags as training features, the result drastically fall. By using lemmas, results remain unchanged. In *Table 5*, some results are listed. The configuration used to generate *Table 5* content is 2grmas smoothed with Witten-Bell. As stated in section 3, an advanced model has been developed. This model improves results by 10%. Some results are listed in *Table 6*. Particularly, the best result has been achieved with 4grams by using Kneser Ney as smoothing method and words as advanced feature. Other results can be found on (Github) but are not significant.

Table 3: Baseline

ngo	prec.	rec.	F1
1	0.546	0.598	0.571
2	0.783	0.741	0.762
3	0.765	0.745	0.755
4	0.771	0.748	0.760
5	0.773	0.747	0.760
6	0.767	0.746	0.756

Table 4: Smoothing function

smoothing	prec.	rec.	F1
absolute	0.785	0.743	0.763
katz	0.780	0.738	0.758
kneser ney	0.784	0.742	0.762
presmoother	0.784	0.742	0.762
witten bell	0.783	0.743	0.763

Table 5: Features

feature	prec.	rec.	F1
lemma	0.779	0.736	0.757
pos	0.179	0.119	0.143
word	0.785	0.743	0.763

5 Conclusion

The original goal to develop an SLU for concept tagging has been achieved. The best result of the

Table 6: Final results

ngo	smoothing	adv. feat.	prec.	rec.	F1
3	Kneser Ney	word	0.816	0.824	0.820
4	Kneser Ney	word	0.824	0.830	0.827
5	Kneser Ney	word	0.820	0.827	0.824
6	Kneser Ney	word	0.742	0.762	0

basic model and the best result of the advanced model are listed in the following table.

Table 7: Advanced model

model	prec.	rec.	F1
base	0.0785	0.743	0.763
advanced	0.824	0.830	0.827

Generally, with the advanced model better results are achieved.

References

Github. Github repository *github.com/MassimilianoLuca/nlsparql-wfst*.