UNIVERSITY
OF TRENTO - Italy

Department of Information
Engineering and Computer Science

LANGUAGE UNDERSTANDING SYSTEMS

mid-term project

Andrei Catalin Coman
andreicatalin.coman@studenti.unitn.it
197812

Academic Year 2017/2018

Language Understanding Systems
Spoken Language Understanding Module

Andrei Catalin Coman
andreicatalin.coman@unitn.it

Abstract

This report aims to describe the various phases that led to the development of a Spoken Language Understanding Module required as mid-term project within the Language Understanding Systems course[1], held by Professor Giuseppe Riccardi[2] at the University of Trento[3] in the academic year 2017/2018. A **data analysis** section is initially introduced to provide different insights related to the data underlying the module. The approach to the **development** is then presented, which includes data manipulation and parameters search. Finally, a section dedicated to performance **evaluation** follows, with the aim of comparing the developed module with a reference baseline.

1 Introduction

The scope of the Spoken Language Understanding Module that has been developed, is to receive a series of sentences as input and then output the same sentences with the addition of so-called tags on each word. This process is summarized in the following figure, where tags are applied to the words of the sentence received as input.

Figure 1: Sentence tagger

[1]**LUS UNITN**: https://goo.gl/4RBv5K
[2]**Giuseppe Riccardi**: http://disi.unitn.it/ riccardi/
[3]**UNITN**: http://www.unitn.it/

For the development of such module, a train file was provided containing a series of sentences consisting of terms separated by tabs, where the first element represents the word (`token`) and the second element represents the tag (`concept`). An supplementary train file was also provided containing additional features such as the POS-tag[4] and the lemma of each `token`.

As for the evaluation, a test file (and also the one with additional features) was provided which follows the same format as the train file. Both train and test sets come from the NL-SPARQL dataset containing utterances related to the movie domain.

2 Data analysis

Since the development of the module is entirely data-driven, it is important to carry out an analysis of the data in order to reveal any insights and issues. For this purpose, the following files were examined:

- `NLSPARQL.train.data`
- `NLSPARQL.train.feats.txt`
- `NLSPARQL.test.data`
- `NLSPARQL.test.feats.txt`

The first simple analysis consists of reporting the number of different elements that describe the above-mentioned files. The following tables show the results obtained.

TRAIN TABLES

NLSPARQL.train.data	counts
# of lines	24791
# of sentences	3338
# of tokens	21453
# of unique tokens	1728
# of unique concepts with prefix	41
# of unique concepts without prefix	24

[4]**POS-tag**: Part-Of-Speech tag

NLSPARQL.train.feats.txt	counts
# of unique POS-tags	49
# of unique lemmas	1582

TEST TABLES

NLSPARQL.test.data	counts
# of lines	8201
# of sentences	1084
# of tokens	7117
# of unique tokens	1039
# of unique concepts with prefix	39
# of unique concepts without prefix	23

NLSPARQL.test.feats.txt	counts
# of unique POS-tags	46
# of unique lemmas	952

A subsequent analysis consisted in verifying whether the terms were following a simple mathematical form known as Zipf's law. The latter claims that a small number of events occur with high frequencies and a large number of events occur with low frequencies (Baayen, 2001).

With regard to the train, the distributions of `tokens`, `concepts`, `POS-tags`, and `lemmas` were analyzed, giving the following graphs as results.

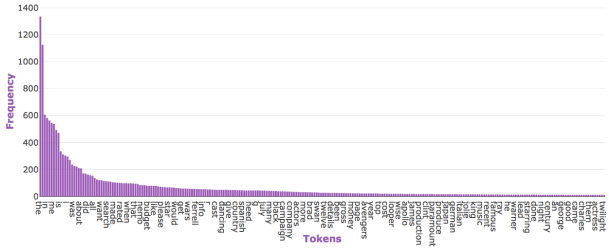

Figure 2: Train tokens distribution

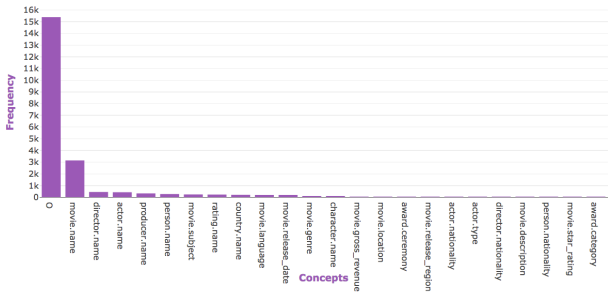

Figure 3: Train concepts distribution

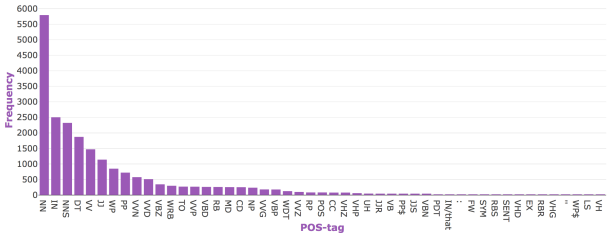

Figure 4: Train POST-tags distribution

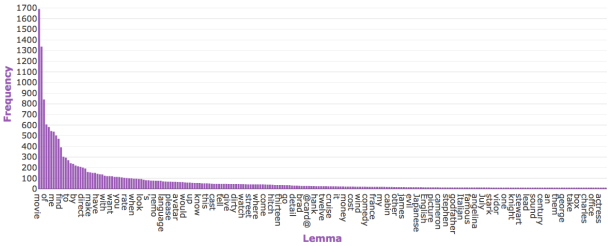

Figure 5: Train lemmas distribution

The same analysis was also performed on the test file. The aim is to highlight eventual discrepancies between the two sets. If the distribution of terms in the test is very different from that of the train, then the model, which is based on train data, would be ineffective.

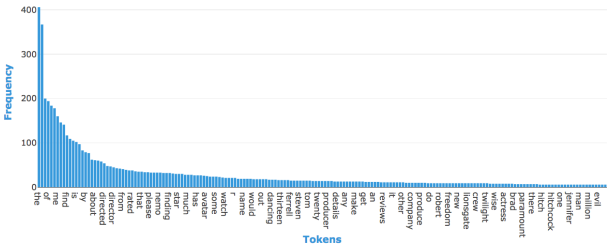

Figure 6: Test tokens distribution

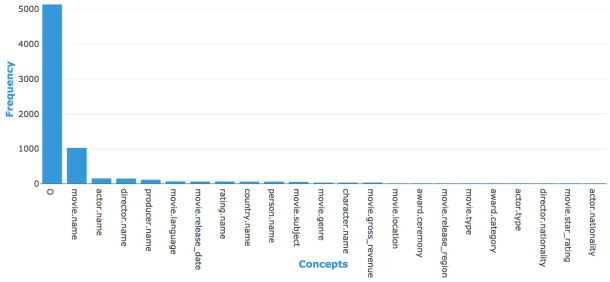

Figure 7: Test concepts distribution

The distributions presented above are in accordance with Zipf's law. In addition, the distributions between train and test are very similar concerning the terms analyzed.

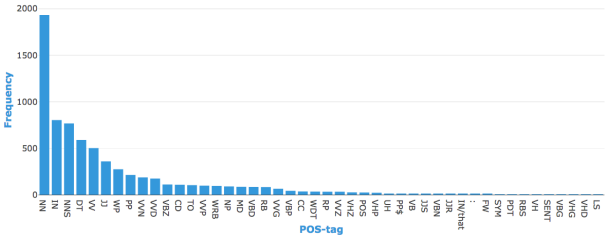

Figure 8: Test POST-tags distribution

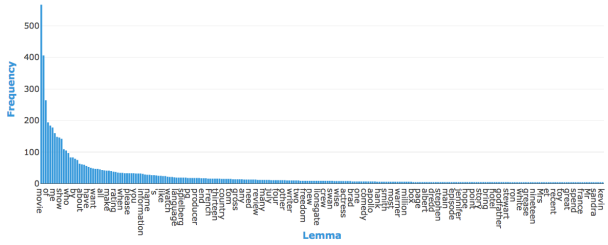

Figure 9: Test lemmas distribution

The OOV[5] rate, i. e. the percentage of words that are present in the test but not in the train amounts to `23.68%`.

As a side note, the occurrence of some typos in both sets could be pointed out, such as Pretty Women instead of Pretty Woman, Martin Scorcese instead of Martin Scorsese, dispaly instead of display, Liongate instead of Lionsgate, Appollo thirteen instead of Apollo thirteen etc.

3 Module development

Given a word sequence $w_1, ..., w_n$ that represents a sentence, the task of the module is to find the most likely tag sequence $t_1, ..., t_n$.
We can express this in a more formal way as:

$$t_1, ..., t_n = \underset{t_1,...,t_n}{\operatorname{argmax}} P(t_1, ..., t_n | w_1, ..., w_n) \quad (1)$$

Moreover we can make some simplifying assumptions and claim that the probability of a word only depends on its own tag, not tags of other words in sentence.

A similar assumption can be made with regard to tags, claiming that the probability of a tag depends only on the previous tag and not on all the previous tags in the sentence (bigram case). All this allows reformulating the problem as follows:

$$t_1, ..., t_n = \underset{t_1,...,t_n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1}) \quad (2)$$

[5]**OOV**: Of Vocabulary

where

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \quad (3)$$

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_i)} \quad (4)$$

and

$$C(x) = count\ of\ x \quad (5)$$

The SLU Module consists of three main components which are:

- `sentence FST`[6]: The sentence to be tagged is represented by a transducer as shown in the following figure.

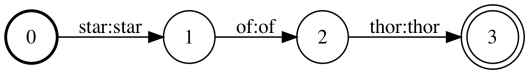

Figure 10: Sentence as a transducer

- `concept tagger WFST`[7]: This component encodes 3 as a transducer, where the probability of a tag given a concept is represented by a transition, and the cost of this transition takes as its value result of $P(w_i|t_i)$.

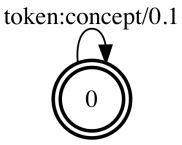

Figure 11: Token:Concept/Weight transducer

- `concept LM`[8]: This component encodes 4 as a transducer known as Language Model. The probability of a concept conditioned on one or more previous concepts is represented by a series of weighted transitions. Tools such as OpenFst[9] and OpenGrm[10] were used to create this component as they allow to switch from bigrams to trigrams or n-grams in general, with extreme ease.

[6]**FST**: Finite State Transducer
[7]**WFST**: Weighted Finite State Transducer
[8]**LM**: Language Model
[9]**OpenFst**: http://www.openfst.org/
[10]**OpenGrm**: http://opengrm.org/

The creation of the second component should take into account the case in which $P(t|w)$ is equal to zero because the token w has never been observed in the train data. To address this problem, as a first step, a special token indicated as <unk>, which stands for unknown token has been inserted into the set of tokens. The second step involves defining the weights of the transitions <unk>:concept, which means defining the probabilities. The two approaches that have been adopted are:

- uniform: The probability of a concept t given that token <unk> has been seen, is simply computed as $\frac{1}{\#\ of\ concepts}$
- cut-off: All (t, w) pairs that are below a previously imposed threshold θ are collected. Subsequently, for each concept t in this category, the amount of tokens A_t related to this specific concept is determined by $\sum_{i=1}^{n}(w_i, t)$. Finally a <unk>:t transition is created, whose weight is computed as $\frac{A_t}{\sum_{i=1}^{n}\#(t_i,w_i)}$. This gives a higher weight to those concepts t that have a larger number of tokens w satisfying the $(w, t) < \theta$ condition.

With regard to the application of the module, it is necessary to make the composition of the three components and take the path that minimizes the cost within the final transducer.

Since cost calculation implies multiplication where factors are $\mathbb{R}$ numbers between 0 and 1, the underflow problem may be relevant. To overcome this problem, it is sufficient to apply the negative natural logarithm to the probability value. A higher probability value will imply a lower cost and vice versa.

4 Evaluation

Each of different versions of the SLU Module has been compared to a baseline defined as follows:

- random: the concept to be assigned to a token is randomly chosen among the set of train concepts. The performance on the test set

Accuracy	Precision	Recall	F1-score
0.0246	0.0032	0.0202	0.0056

- chance: the choice of the concept follows the probability distribution of the concepts within the train set.

Accuracy	Precision	Recall	F1-score
0.5325	0.0132	0.0229	0.0167

- majority: the choice of the concept relies on the most common concept in the train.

Accuracy	Precision	Recall	F1-score
0.7215	0.0	0.0	0.0

The first version of the SLU Module was structured as follows:

- <unk>: handleded with uniform probability
- ngram_size: [2, 3, 4, 5]
- smoothing: witten_bell, kneser_ney, katz, absolute and presmoothed

and the best results were reached with:

ngram_size	smoothing	F1-score
2	witten_bell	0.7637
2	kneser_ney	0.7627
2	absolute	0.7637
2	presmoothed	0.7627
4	witten_bell	0.7622
4	kneser_ney	0.7619
4	absolute	0.7619
5	witten_bell	0.7632
5	kneser_ney	0.7616
5	absolute	0.7615

As an attempt to improve performances, files containing additional features were used. This has led to three new strategies including:

- lemma: the token is replaced with the corresponding lemma
- lemmapos: the token is replaced by the concatenation of the lemma and POS-tag
- tokenpos: the token is replaced by the concatenation of the token and POS-tag

However, expectations were not met resulting in a drop ranging from 0.005 to 0.02 in terms of F1-score.

References

R. Harald Baayen. 2001. *Word Frequency Distributions*, volume 1. Springer Netherlands.