# Concept Tagging Module for Movie Domain
# Mid-Term Project Language Understanding System

**Alberto Carbognin**
Università degli Studi di Trento
mat. 211420
alberto.carbognin@studenti.unitn.it

## Abstract

In this mid-term project, we developed a concept tagging model by using a **weighted finite state transducer** from a movie domain. We then created several **language models** with different configuration ngram orders and smoothing algorithms. Finally, we evaluated it with the conlleval script. In addition, we explored other datasets: the ATIS (Airline Travel Information System) and the datasets considered in the paper (Braun et al., 2017).

## 1 Introduction

The main focus of this project was to develop a Concept Tagging Module, which is a small piece of a typical SLU[1] pipeline. Usually, this kind of pipeline may include automatic speech recognition, count vector featurizer, intent classifiers, and so forth. The role of the Concept Tagger is to extract notions from a sequence of words (or tokens). Given the complexity of building WFSTs[2] over heterogeneous domain, we used a specific topic provided by the NL2SparQL corpus that happens to be movie related. We used the OpenFst libraries to build the model and the language models with different ngram orders and smooth algorithms. As a step forward we iterate the same process over different datasets, without using additional features as POS or LEMMA, and compared the performance F1 score to research. The report is structured in three sections; in the **Data Analysis** it is reported a brief description of the datasets used in the project. In the second section, **Evaulation**, a description of the model, the experimental results with error analysis are shown. In the last section, **Discussion**, the obtained results are discussed.

[1] Spoken Language Understanding
[2] Weighted Finite-State Transducers

## 2 Data Analysis

The dataset provided by the *NL2SparQL corpus* has four main important files:

- *NL2SparQL4NLU.train.utterances.txt*: this file contains a list of train sentences, one in each row separated by the control character new line;
- *NL2SparQL4NLU.train.conll.txt*: this file contains the same list of train sentences of the first file, but splitted by each word; plus it has the IOB annotation in the same row of the word separated by the control character horizontal tab;
- *NL2SparQL4NLU.test.utterances.txt*: this file is similar to the first one and it contains a list of test sentences, one in each row separated by the control character new line;
- *NL2SparQL4NLU.test.conll.txt*: this file contains the test sentences with rows splitted by word plus the IOB annotation.

There are a total of 3338 sentences or 21453 tokens in the former set, the training one, and 1084 or 7117 tokens in the latter, the test one. In the table 1 a list of concepts is reported by name and dataset; some of them are very rare like `person.nationality` or absent like `movie.type`.

The *NLU-Evaluation-Corpora* dataset used in (Braun et al., 2017) were in `json`; we then proceded to convert them to the desired format. These datasets are very small like reported in table 2. In order to evaluate the WFSTs over these datasets we split them in *test* and *train* by the 30% and the 70% respectively like reported in the table 3.

The ATIS dataset was in *json* format and already splitted into *train* and *test* for a total number of sentences of 5871. The train set has 4978 sentences,

| concept | train | test |
|---|---|---|
| actor.name | 437 | 157 |
| actor.nationality | 6 | 1 |
| actor.type | 3 | 2 |
| award.category | 1 | 4 |
| award.ceremony | 13 | 7 |
| character.name | 97 | 21 |
| country.name | 212 | 67 |
| director.name | 455 | 156 |
| director.nationality | 2 | 1 |
| movie.description | 2 | 0 |
| movie.genre | 98 | 37 |
| movie.gross$_revenue$ | 34 | 20 |
| movie.language | 207 | 72 |
| movie.location | 21 | 11 |
| movie.name | 3157 | 1030 |
| movie.release$_date$ | 201 | 70 |
| movie.release$_region$ | 10 | 6 |
| movie.star$_rating$ | 1 | 1 |
| movie.subject | 247 | 59 |
| movie.type | 0 | 4 |
| person.name | 280 | 66 |
| person.nationality | 2 | 0 |
| producer.name | 336 | 121 |
| rating.name | 240 | 69 |

Table 1: *NL2SparQL* concepts.

| dataset | total sentences |
|---|---|
| ATIS | 5871 |
| AskUbuntuCorpus | 162 |
| ChatbotCorpus | 206 |
| WebApplicationsCorpus | 89 |

Table 2: *NLU-Evaluation-Corpora* and ATIS datasets.

| dataset | train | test |
|---|---|---|
| ATIS | 4978 | 893 |
| AskUbuntuCorpus | 113 | 49 |
| ChatbotCorpus | 144 | 62 |
| WebApplicationsCorpus | 62 | 27 |

Table 3: *NLU-Evaluation-Corpora* and ATIS splitted datasets.

while the test one 893; the latter is about the 15% of the total number of sentences.

## 3 Evaluation

The evaluation of the performance has been performed by the *conlleval* script by using the IOB notation and gathering insightful metrics like: precision, recall and F1.

### 3.1 Model

In order to solve the problem of tagging we use the maximum likelihood approach. We must compute the best sequence of tags $t_1, ..., t_n$ given a sequence of words $w_1, ..., w_n$, formally can be expressed as:

$$t^* = \underset{t}{\operatorname{argmax}} P(t|w)$$

$$= \underset{t}{\operatorname{argmax}} \frac{P(w|t)P(t)}{P(w)}$$

$$= \underset{t}{\operatorname{argmax}}(\prod_{i=1}^{n} P(w_i|t_i))P(t)$$

$$= \underset{t}{\operatorname{argmax}}(\sum_{i=1}^{n} log(P(w_i|t_i)))P(t) \quad (1)$$

Above we have applied the Bayes' theorem to make a strong assumption that the probabilities of tokens are independent from one another given their tags. We use the log function to address the fact that there could be very small values.

The first Finite State Transducer receives the inputs as a sequence of tokens and simply outputs what has been received. The second is a Weighted Finite State Transducer that has the role of concept tagger, it takes the input and generate arcs between states. The last component is the language model; this pipeline could be written as, where $\circ$ is the compose operation:

$$\lambda_{fst} \circ \lambda_{concept\_tagger} \circ \lambda_{language\_model} \quad (2)$$

We built the lexicon using the *ngramsymbols* tool from the dataset, it contains all the tokens and concept tags plus the $< UNK >$ for the unknown words[3] and the $< epsilon >$ and finally we call the *farcompilestrings* to create the first $\lambda_{fst}$. The second component is working using the viterbi algorithm as reported in the OpenFST documentation,

[3] Also know as OOV (Out of Vocabulary words).

so that to be able to maximize the log likelihood of the following equation:

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

it is required to multiply by negative factor of -1. The last component is a language model based on n-grams; in order to gather more information multiple models with different ngram sized and smoothing algorithms were built. Python has been used to write configurable scripts that automatically perform the tests and gather the results we need.

### 3.2 Results

The experiments were performed by the *main.py* script; multiple ngram orders and smoothing algorithms have been tried. Across all the datasets the we tried the ngram orders: 1, 2, 3, 4 and 5; and the smoothing algorithms: absolute, katz, kneser, presmoothed, unsmoothed and witten. The movie domain dataset results are reported in the table 4; the language model configuration that has the best F1 score **76.37** is the one with ngrams equal **2** and the smoothing **absolute**. We can observe that with ngrams equal 3 and the smoothing witten we obtain a similar score. It seems that improving the ngrams for most smoothing algorithms leads to worst performance. In the ATIS dataset we obtain different results as reported in the table 5; the best F1 score **86.50** is given by the combination of ngrams equal **5** and the **kneser** algorithm. We also observed a significant improvements only by 2 to 3 ngrams and a stagnant performance improvement afterwards; in some cases the score decreased. In this dataset the precision and recall values are higher than the movie domain one; this confirm us that the model is able to perform the task better within this domain. The tables 6 7 8 were obtained from the datasets available at the the NLU-Evaluation-Corpus github repository[4]; we performed the same combination of ngrams orders and smoothing algorithms, but we omit on purpose the last rows of table as no significant results were obtained in terms of F1 score performance[5]. In the *AskUbuntuCorpus* domain the F1 performance score of **71.19** in the gathered results is very low; nevertheless the best performance was found with ngrams equal to **3** and the **absolute** algorithm; at the increasing of the ngrams we observed lower precision. The experiments on the *WebApplicationCorpus* went even worse; the best parameters are ngram equal to **2** and **absolute** algorithm with a F1 score of just **57.14**. Furthermore we observed a rapid decreasing of the precision when increasing the ngrams. In the other hands we obtained better results for the *ChatbotCorpus* with the configuration of ngrams equal **3** and the **katz** algorithm; we observed a F1 performance score of **88.67**.

| n | smoothing | precision | recall | F1 |
|---|---|---|---|---|
| 2 | absolute | 78.51% | 74.34% | **76.37** |
| 2 | katz | 78.03% | 73.88% | 75.89 |
| 2 | kneser | 78.41% | 74.24% | 76.27 |
| 2 | presmoothed | 78.41% | 74.24% | 76.27 |
| 2 | unsmoothed | 78.39% | 74.15% | 76.21 |
| 2 | witten | 78.51% | 74.34% | 76.37 |
| 3 | absolute | 76.67% | 74.70% | 75.67 |
| 3 | katz | 75.60% | 72.69% | 74.11 |
| 3 | kneser | 76.67% | 74.70% | 75.67 |
| 3 | presmoothed | 64.81% | 71.40% | 67.95 |
| 3 | unsmoothed | 76.55% | 74.52% | 75.52 |
| 3 | witten | 76.58% | 74.61% | 75.58 |
| 4 | absolute | 77.16% | 75.25% | 76.19 |
| 4 | katz | 72.57% | 73.97% | 73.26 |
| 4 | kneser | 76.87% | 75.53% | 76.19 |
| 4 | presmoothed | 62.28% | 72.50% | 67.01 |
| 4 | unsmoothed | 77.15% | 74.89% | 76.00 |
| 4 | witten | 77.11% | 75.34% | 76.22 |
| 5 | absolute | 76.97% | 75.34% | 76.15 |
| 5 | katz | 56.95% | 71.31% | 63.33 |
| 5 | kneser | 76.79% | 75.53% | 76.16 |
| 5 | presmoothed | 62.36% | 72.14% | 66.89 |
| 5 | unsmoothed | 77.35% | 74.79% | 76.05 |
| 5 | witten | 77.03% | 75.62% | 76.32 |

Table 4: *NL2SparQL* performance comparison with multiple parameters combinations.

## 4 Discussion

In the movie domain dataset we met the objective baseline for the F1 score of 76; nevertheless there are multiple improvements that can be done to the model, for instance by training the language model with words and concept tags. Using the ATIS dataset we got a F1 score of 86.50 with higher precision and recall, this means that in this case the model, even without improvements is able to perform the concept-tagging task very well. In the other datasets (Chatbot, AskUbuntu and WebAppli-

[4] https://github.com/sebischair/NLU-Evaluation-Corpora

[5] To view the full table please refer to the the project's repository on github.

| n | smoothing | precision | recall | F1 |
|---|---|---|---|---|
| 2 | absolute | 74.77% | 74.49% | 74.63 |
| 2 | katz | 74.88% | 74.76% | 74.82 |
| 2 | kneser | 74.62% | 74.40% | 74.51 |
| 2 | presmoothed | 74.38% | 74.06% | 74.22 |
| 2 | unsmoothed | 74.33% | 73.96% | 74.15 |
| 2 | witten | 74.58% | 74.36% | 74.47 |
| 3 | absolute | 85.11% | 84.66% | 84.89 |
| 3 | katz | 84.21% | 84.27% | 84.24 |
| 3 | kneser | 85.53% | 84.80% | 85.16 |
| 3 | presmoothed | 84.46% | 84.10% | 84.28 |
| 3 | unsmoothed | 84.85% | 83.87% | 84.36 |
| 3 | witten | 85.18% | 84.73% | 84.96 |
| 4 | absolute | 85.39% | 84.60% | 84.99 |
| 4 | katz | 77.81% | 78.54% | 78.17 |
| 4 | kneser | 85.55% | 85.09% | 85.32 |
| 4 | presmoothed | 82.76% | 83.01% | 82.88 |
| 4 | unsmoothed | 85.31% | 83.27% | 84.28 |
| 4 | witten | 85.19% | 84.56% | 84.87 |
| 5 | absolute | 86.55% | 85.46% | 86.00 |
| 5 | katz | 80.70% | 80.49% | 80.60 |
| 5 | kneser | 86.74% | 86.25% | **86.50** |
| 5 | presmoothed | 83.68% | 83.90% | 83.79 |
| 5 | unsmoothed | 85.69% | 82.74% | 84.19 |
| 5 | witten | 86.40% | 85.46% | 85.93 |

Table 5: *ATIS* performance comparison with multiple parameters combinations.

| n | smoothing | precision | recall | F1 |
|---|---|---|---|---|
| 2 | absolute | 82.61% | 59.38% | 69.09 |
| 2 | katz | 82.61% | 59.38% | 69.09 |
| 2 | kneser | 82.61% | 59.38% | 69.09 |
| 2 | presmoothed | 82.61% | 59.38% | 69.09 |
| 2 | unsmoothed | 82.61% | 59.38% | 69.09 |
| 2 | witten | 82.61% | 59.38% | 69.09 |
| 3 | absolute | 77.78% | 65.62% | **71.19** |
| 3 | katz | 16.67% | 40.62% | 23.64 |
| 3 | kneser | 75.00% | 65.62% | 70.00 |
| 3 | presmoothed | 74.07% | 62.50% | 67.80 |
| 3 | unsmoothed | 67.74% | 65.62% | 66.67 |
| 3 | witten | 77.78% | 65.62% | 71.19 |

Table 6: *AskUbuntuCorpus* performance comparison with multiple parameters combinations, full results can be found in the github repository.

| n | smoothing | precision | recall | F1 |
|---|---|---|---|---|
| 2 | absolute | 87.29% | 66.88% | 75.74 |
| 2 | katz | 75.97% | 75.97% | 75.97 |
| 2 | kneser | 87.29% | 66.88% | 75.74 |
| 2 | presmoothed | 75.97% | 75.97% | 75.97 |
| 2 | unsmoothed | 75.66% | 74.68% | 75.16 |
| 2 | witten | 87.29% | 66.88% | 75.74 |
| 3 | absolute | 85.16% | 85.71% | 85.44 |
| 3 | katz | 88.39% | 88.96% | **88.67** |
| 3 | kneser | 85.16% | 85.71% | 85.44 |
| 3 | presmoothed | 73.58% | 75.97% | 74.76 |
| 3 | unsmoothed | 84.46% | 81.17% | 82.78 |
| 3 | witten | 85.16% | 85.71% | 85.44 |

Table 7: *ChatbotCorpus* performance comparison with multiple parameters combinations, full results can be found in the github repository.

| n | smoothing | precision | recall | F1 |
|---|---|---|---|---|
| 2 | absolute | 82.35% | 43.75% | **57.14** |
| 2 | katz | 82.35% | 43.75% | 57.14 |
| 2 | kneser | 82.35% | 43.75% | 57.14 |
| 2 | presmoothed | 82.35% | 43.75% | 57.14 |
| 2 | unsmoothed | 82.35% | 43.75% | 57.14 |
| 2 | witten | 82.35% | 43.75% | 57.14 |
| 3 | absolute | 68.75% | 34.38% | 45.83 |
| 3 | katz | 68.75% | 34.38% | 45.83 |
| 3 | kneser | 70.59% | 37.50% | 48.98 |
| 3 | presmoothed | 77.78% | 43.75% | 56.00 |
| 3 | unsmoothed | 68.75% | 34.38% | 45.83 |
| 3 | witten | 70.59% | 37.50% | 48.98 |

Table 8: *WebApplicationsCorpus* performance comparison with multiple parameters combinations, full results can be found in the github repository.

cations) we observed low performance according to the F1 score for the AskUbuntu and the WebApplications; this may be given from the small size of the datasets. In the Chatbot dataset the F1 score was very high, but this results cannot be conclusive; the limited amount of availability of both training and test data could have lead to a model that is not able to generalize very well outside the experimental lab. Taking as reference the results obtained with rasa in the paper (Braun et al., 2017) we observed that our model is performing less good for the chatbot dataset with an F1 score of 94.3 against 88.67. Regarding the WebApplications dataset our model is performing slightly better with a F1 score of 57.14 againts 56.7; in the AskUbuntu dataset the paper F1 score of 80.7 is better than our 71.19. Even the author of the paper cannot conclude if the obtained results can be trusted for a production environment; so that given the fact that our results are very similar, future papers should focus on expanding the datasets and perform further investigations.

## References

Daniel Braun, Adrian Hernandez-Mendez, Florian Matthes, and Manfred Langen. 2017. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, Saarbrücken, Germany. Association for Computational Linguistics.