Language Understanding Systems
Midterm Project

Riccardo SCILLA - 213723
riccardo.scilla@studenti.unitn.it

Abstract

The aim of this project is to implement a concept tagging model on NL2SparQL4NLU dataset by using weighted finite-states transducers. Different approaches and configurations are used in order to achieve better results.

1 Introduction

Spoken language understanding (SLU) is an emerging field in speech and language processing. We focus on Concept Tagging which task is to assign to each word in a sentence, the correct tag that describes the concept. This task is approached using weighted finite-states transducers which provide an efficient way to represent and process probabilities of sequences. We use *OpenFST* and *OpenGRM*.

The report is structured as follow: we start analyzing the NL2SparQL4NLU dataset that contains sentences related to the movie domain. We move on the description of the methodology used, in particular on how out-of-vocabulary words are processed and how the different pipelines of concatenated WFST are built in order to achieve better performance. The results of these methods are described and discussed in the following section, moving on the conclusions.

2 Data Analysis

The NL2SparQL4NLU dataset contains sentences coming from the movies domain. The dataset is splitted into train and test set. For each set we have an utterances file, which contains one sentence per row, and a conll file, where sentences are splitted into words and a label is assigned to each word using IOB-schemes. IOB refers to " Inside", "Outside" and "Beginning" of span respectively.

Relevant information regarding the analysis of the data in terms of sizes and distributions of types and output labels is are briefly described in Table 1.

Set	Sentences	Unique words	IOB-tags
train	3338	1729	41
test	1084	1039	39

Table 1: Dataset analysis

We can represent the empirical distribution of the frequency of the words in our train set using the Zipf's Law. In Figure 1 are plotted the first 35 most frequent words of the train and test set.

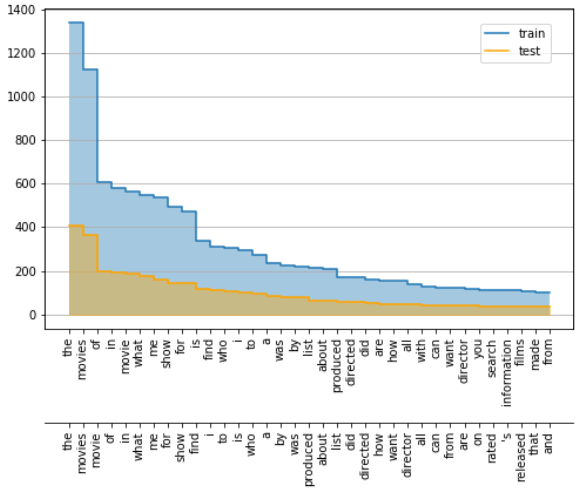

Figure 1: Train and Test set Zipf's Law

In Table 2 we can see how concepts are distributed in train and test set. The most frequent concept is *movie.name* both in train and test set; the concept *movie.type* is not present in the train set while the concepts *person-nationality* and *movie.description* are not present in the test set. The other concepts are more or less distributed proportionally with respect to the sizes of the sets.

Concept	Train	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	21
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
person.nationality	2	0
movie.description	2	0
movie.star_rating	1	1
award.category	1	4
movie.type	0	4

Table 2: Concepts distribution

Looking at the frequency of tag *O* in train and test sets, we can see from data that it is present with frequency of 15391 in train and 5135 in test, so it covers 71.74% and 72.15% of the total number of tags respectively in train and test set. Due to the fact that *O* is the majority in tags, it influence the performance of our models, so we will see that playing with it brings us better results.

The models are trained using the entire train set and using the train set with cut-off 1: here words and tags that have frequency equal to 1 are removed and replaced by <*unk*>. If cut-off is applied in train set, 779 words and 2 tags with frequency 1 are removed. All the words and tags not present in the train set are also replaced by <*unk*>.

3 Methodology

In order to achieve Concept Tagging task, Hidden Markov Model (HMM) is used. HMM allows us to talk about both observed events (like words that we see in the input) and hidden events (like part-of-speech tags). When we are modelling observed sequences, the states of the process are observable so we only need to compute *transition probabilities*. In HMM, we assume that observed sequences have been generated by a Markov Process with unobservable states. Since the states of the process are hidden and the output is observable, each state has a probability distribution over the possible output tokens, called *emission probabilities*.

In formulas, given a word sequence $\mathbf{w} = \{w_1, w_2, \ldots, w_n\}$ the goal is to find the most probable tag sequence $\mathbf{t} = \{t_1, t_2, \ldots, t_n\}$, such that:

$$\begin{aligned}\mathbf{t} &= \underset{\mathbf{t}}{\operatorname{argmax}}\, p(\mathbf{t}|\mathbf{w}) \\ &= \underset{\mathbf{t}}{\operatorname{argmax}}\, \frac{p(\mathbf{w}|\mathbf{t})p(\mathbf{t})}{p(\mathbf{w})} \\ &= \underset{\mathbf{t}}{\operatorname{argmax}}\, p(\mathbf{w}|\mathbf{t})p(\mathbf{t})\end{aligned}$$

where Bayes's Rule is used.

We consider simplifying assumptions on these probabilities:

1. p of a word only depends on its own tag,

$$p(\mathbf{w}|\mathbf{t}) = \prod_{i=1}^{n} p(w_i|t_i)$$

2. p of a tag depends on previous n tags,

$$p(\mathbf{t}) = \prod_{i=1}^{n} p(t_i|t_{i-n+1}^{i-1})$$

These two probabilities are respectively *emission* and *transition* probabilities. They can be estimated using frequency counts:

$$p(w_i|t_i) = \frac{count(w_i, t_i)}{count(t_i)}$$

$$p(t_i|t_{i-1}) = \frac{count(t_{i-n+1}^{i-1}, t_i)}{count(t_{i-n+1}^{i-1})}$$

From this procedure we can understand what are the basic components of the pipeline we need for compute the task: first, a FST is applied to the input sentence and concatenated with a WFST which assigns a tag to each word, considering the weights computed as:

$$cost(w_i, t_i) = -ln(p(w_i|t_i))$$

Last, a language model FST that represent $p(\mathbf{t})$ is concatenated from which we obtain the most likely concept sequence.

We can visualize the whole pipeline of the Concept Tagger as follow:

$$\lambda = \lambda_W \circ \lambda_{W2T} \circ \lambda_{LM}$$

3.1 OOV handling and Smoothing

As said in Data Analysis section, out of vocabulary words (OOV) are replaced by <*unk*> in train set before computing the lexicon file.

Differently from the *emission probabilities* $p(w_i|t_i)$ described before, $p(<unk>|t_i)$ are computed by assigning an equal probability to each pair. Thus, the costs of these transitions are:

$$cost(<unk>, t_i) = -ln\left(\frac{1}{V}\right)$$

Maximum Likelihood Estimation (MLE), used to compute *emission probabilities*, suffers from data sparseness. A more accurate way to compute them would be to discount $p(w_i|t_i)$ using Add-1 Smoothing, using the following formula:

$$p(w_i|t_i) = \frac{count(w_i, t_i) + 1}{count(t_i) + V}$$

3.2 Models Analysis

Baseline

In Baseline model the simple pipeline is implemented and evaluated with different parameters such as different ngram orders, smoothing methods and cut-off.

- ngram orders = {*2,3,4,5*}
- smoothing methods = {*absolute, witten_bell, katz, kneser_ney, presmoothed, unsmoothed,*}
- frequency cut-off = {*1,2*}

In case of no cut-off, λ_{W2T} has transitions from <*unk*> to each tag, with the costs described before.

To create the WFSTs we use libraries *OpenFST* and *OpenGRM*. First the dataset is pre-processed in order to create the lexicon using *ngramsymbols*. The first transducer λ_W is created using

farcompilestrings and then extracted using *farextract*. The second WFST λ_{W2T} is created manually with weights as shown before, then compiled with *fstcompile*. The last transducer λ_{LM} is created using *ngramcount* indicating the ngram order and then using *ngramsymbols* indicating the smoothing method. The composition is achieved using *fstcompose* and *fstshortestpath* which compute the Viterbi algorithm to find the best path.

Improvement 1: Remove *O*-tag

As we have discussed in Data Analysis, the *O*-tag is very frequent in the dataset, consequently, there is not enough context to learn a good ngram model. The aim of this improvement is to implement a pipeline without *O*-tag.

To do this, λ_{W2T} is modified in order to translate "word"-"word" if the tag would be *O*. At the end of the pipeline, a new transducer is composed to bring back *O* for the evaluation. The final pipeline is:

$$\lambda = \lambda_W \circ \lambda_{W2T} \circ \lambda_{LM} \circ \lambda_{W2T_{backO}}$$

Improvement 2: Input generalization

In NLP it is common to normalize the input data to reduce sparsity. This procedure replace all members of the infinite set with a single unique token with respect to a common pattern. For our task of Concept Tagging in NL2SparQL4NLU dataset, different words related to the same category are mapped to the same entity.

In particular, a new transducer is composed to the input in order to map persons names to *PER*, locations to *LOC*, nationalities to *NAT* and numbers to *NUM*. The final pipeline is:

$$\lambda = \lambda_W \circ \lambda_G \circ \lambda_{W2T} \circ \lambda_{LM}$$

This can be enhanced removing *O*-tag as seen in Improvement 1.

Joint Distribution Modelling

Sequence labelling can be approached modelling jointly the sequences of words and tags. To do this, R&R model is used:

$$p(\mathbf{w}, \mathbf{t}) \approx \prod_{i=1}^{N} p(w_i t_i | w_{i-N+1}^{i-1}, t_{i-N+1}^{i-1})$$

Here, λ_{W2T} is modified in order to translate "word"-"word+tag". The final pipeline is:

$$\lambda = \lambda_W \circ \lambda_{W2WT} \circ \lambda_{SCLM}$$

NLTK Model

The model is tested using NLTK, which provides implementations of popular sequence labelling algorithms, including HMM Tagger. We use both IOB-tags and POS-tags, that are available in NL2SparQL4NLU dataset.

3.3 Comparative Analysis

In Table 3 are showed the sizes of the Lexicons and the number of arcs of each transducer used in the models. We can see that the input generalization model has the most number of arcs.

	Base	Impr. 1	Impr. 2	JDM
Lexicon (no cut-off)	1772	1772	1775	3989
λ_{W2T}	2300	3040	1071	-
$\lambda_{W2T_{backO}}$	-	1772	993	-
λ_G	-	-	21453	-
λ_{W2WT}	-	-	-	4520
λ_{LM} ngram 4	884	17580	17580	27568

Table 3: Sizes of models

4 Evaluation

The best results of the models described are shown in this section, together with a comparison between different use of cut-off and Add-1 Smoothing

In the **Baseline Model**, best F1 scores are obtained without cut-off and with Add-1 Smoothing.

Smoothing Method	ngram order			
	2	3	4	5
absolute	76.09	75.49	76.22	75.96
witten_bell	**76.37**	75.67	76.18	76.08
kneser_ney	76.09	75.67	76.19	75.76
katz	75.58	74.25	73.10	62.65
presmoothed	76.15	68.44	68.86	68.45
unsmoothed	75.9	75.52	76.12	75.61

Table 4: Baseline best model scores

Results are almost the same without using Add-1 Smoothing, where the model reaches 76.365 at best and sometimes scores are better than the ones shown in Table 4. Using cut-off, the performances are significantly reduced. The summary of the best scores with all parameters is shown in Table 5.

Cut-off	Add-1		Smoothing Method
	Yes	No	
Yes	74.001	74.060	witten_bell
No	76.371	76.365	witten_bell

Table 5: Baseline best scores

In the **Remove *O*-tag Model**, best F1 scores are obtained without cut-off and Add-1 Smoothing.

Smoothing Method	ngram order			
	2	3	4	5
absolute	78.93	81.37	81.57	81.34
witten_bell	79.31	81.34	81.43	81.26
kneser_ney	79.45	82.04	**82.74**	82.43
katz	78.84	80.85	81.17	81.08
presmoothed	77.58	79.54	79.86	79.47
unsmoothed	76.16	78.40	78.25	78.21

Table 6: Remove *O*-tag best model scores

This improvement produce a better F1 score with all the parameters used, but *kneser_ney* results are the best over all. Also in this case, the use of cut-off reduces the performance. The summary of the best scores with all parameters is shown in Table 7.

Cut-off	Add-1		Smoothing Method
	Yes	No	
Yes	80.089	81.006	kneser_ney
No	81.315	82.740	kneser_ney

Table 7: Remove *O*-tag best scores

In the **Input generalization Model** best F1 scores are again obtained without cut-off and Add-1 Smoothing.

Smoothing Method	ngram order			
	2	3	4	5
absolute	68.42	74.97	75.52	75.62
witten_bell	68.33	74.07	74.89	75.44
kneser_ney	69.72	75.00	**75.80**	75.62
katz	68.33	73.67	73.03	72.71
presmoothed	68.02	71.86	71.80	71.84
unsmoothed	67.34	72.42	73.12	73.51

Table 8: Input generalization best model scores

Although the input generalization should improve the model, in this case we can see that the scores a worse than the one obtained with the other models. Also in this case, the use of cut-off reduces the performance. This might be caused by a wrong use of the entity translation. Best scores are again obtained using *kneser_ney*. The summary of the best scores with all parameters is shown in Table 9.

Cut-off	Add-1		Smoothing Method
	Yes	No	
Yes	70.776	71.629	unsmoothed
No	75.122	75.800	kneser_ney

Table 9: Input generalization best scores

Using **NLTK** with HMM Tagger we obtain Accuracy and F1 scores showed in Table 10.

Tags	Accuracy	F1 score
IOB	90.867	77.470
POS	84.881	87.379

Table 10: NLTK HMM Tagger scores

Another possibility is to use NgramTagger provided in NLTK. Accuracy and F1 scores are showed in Table 11 with different ngram orders and cut-offs.

	IOB-tags		
Cut-off	ngram	Accuracy	F1 score
Yes	1	85.72	71.44
	2	71.59	62.91
	3	68.84	60.47
No	1	87.62	74.03
	2	77.98	70.07
	3	75.48	68.06
	POS-tags		
Cut-off	ngram	Accuracy	F1 score
Yes	1	87.42	89.53
	2	58.35	71.03
	3	49.05	62.67
No	1	89.45	90.62
	2	70.37	80.19
	3	58.07	71.00

Table 11: NLTK nGramTagger scores

From scores we can see that using POS-tags we obtain worse Accuracy while F1 score is better both in HMM and nGramTagger. In nGramTagger the statistics decreases with the ngram orders and applying cut-off.

5 Conclusion

In this project, WFST are used to implement a Concept Tagger in a movie domain dataset. Different methods and approaches are used with the aim to implement the best model.

Results show that removing *O*-tag in language model provides better scores in evaluation with respect to the baseline model. On the other hand, the generalization of the input with entities is harder to use in this domain, therefore it doesn't provide an effective improvement. The best smoothing method used is *kneser_ney*.

References

LUS Course webpage: disi.unitn.it/ riccardi/page7/page13/page13.html

LUS Lab: github.com/esrel/LUS

Gobbi Jacopo, Stepanov Evgeny, Riccardi Giuseppe. (2018). Concept Tagging for Natural Language Understanding: Two Decadelong Algorithm Development.