Building Concept Tagger with Weighted Finite-States Transducer

Viet Nhat Nguyen
DISI, Univeristy of Trento
vietnhat.nguyen@studenti.unitn.it

Abstract

By using Weighted Finite-States Transducers, we build a concept tagging model on movie dataset. We compare the performances of our models when using different approaches and various of configurations in Language Models on a given test set. We found that relatively KneserNey smoothing performs better than the others among 6 studied smoothing methods.

1 Introduction

Spoken Language Understanding (SLU) is one of the main problems in language technology research. This system is traditionally designed as a pipeline of a number of components such as automatic speech recognization, intent identification, dependency parsing, etc. In this project, we focus on Concept Tagging Module which assigns each single phrase to a sequence indicating the concept of that word. For example in musical domain, an appropriate system should be able to identify "Queen" as the name of a British rock band rather than a female monarch of a kingdom and "Yesterday" is a song, not an adverb of time in sentence "Queen did not perform Yesterday".

Although current state-of-the-art concept tagging models are based on Deep Learning techniques (Gobbi et al., 2018), we use Weighted Finite-State Transducers (WFST) (Mohri et al., 2002) since its structure is simple and intepretable to human. Because the tremendous variety of possible concepts in practice, within the scope of this project we narrow the output tags to movie domain through experiments on a dataset derived from NL2SparQL corpus (Chen et al., 2014).

The remain of this report is organized as follow: we will start by investigating the properties of provided data and showing the distribution of concepts in section 2. In the section after that we will present the baseline approach including statistical motivation behind, how to transform ideas into WFST and results obtained with this baseline model. Then in section 4 we introduce three variations of the baseline model, namely using lemmas, using word-concept tags and using entity recognizer respectively.

2 Data Analysis

Let us first explore some characteristics of the dataset used in this project. This dataset consists of 4 main files:

- *NL2SparQL4NLU.train.features.conll*: each line contains one token, POS and lemma respectively separated by tab character.
- *NL2SparQL4NLU.train.conll*: each line contains one token and its concept tag. The concept tags are represented in the IOB format. Specifically, except from the tag "O", each concept is added a prefix "B" or "I" meaning "beginning of span" and "inside of span" respectively.
- *NL2SparQL4NLU.test.features.conll*: similar to *NL2SparQL4NLU.train.features.conll* by applied for test set.
- *NL2SparQL4NLU.test.conll*: similar to *NL2SparQL4NLU.train.conll* by applied for test set.

There are 3338 sentences in the training set and 1084 sentences in the test set. The average length of sentences in training set is about 6.5 words. All tokens in this data had been normalized to lower case characters. In addition, the data has been modified from the original with respect to tokenization of possessive apostrophes. Table 1 shows some examples drawn from two files of training set described above.

Token	POS	Lemma	Concept
what	WP	what	O
’s	VBZ	be	O
the	DT	the	O
cast	NN	cast	O
for	IN	for	O
life	NN	life	B-movie.name
is	VBZ	be	I-movie.name
beautiful	JJ	beautiful	I-movie.name
who	WP	who	O
played	VVD	play	O
nick	NN	nick	B-character.name
fury	NN	fury	I-character.name
in	IN	in	O
the	DT	the	B-movie.name
avengers	NNS	avenger	I-movie.name

Table 1: Examples from training data

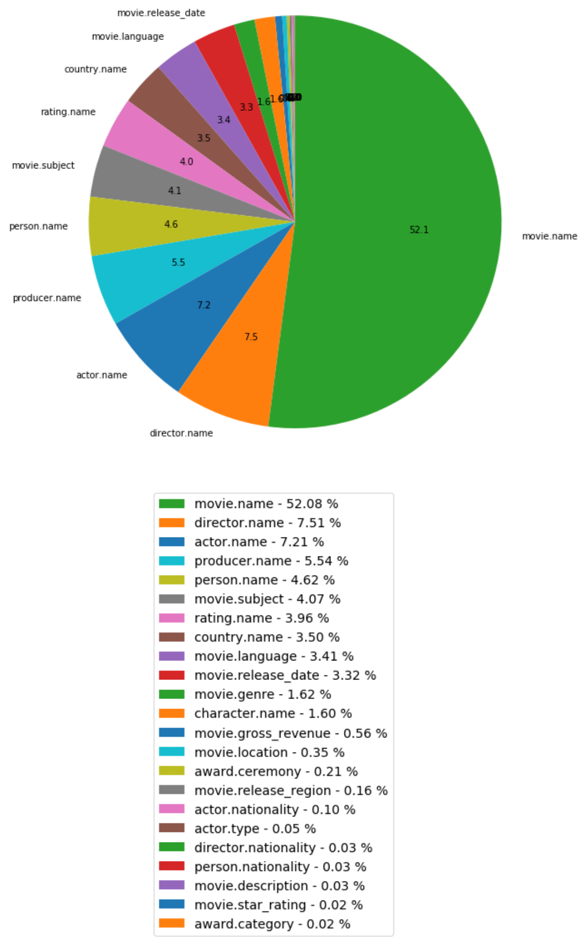

Figure 1: Distribution of concept tags in the training set.

The number of out of concept tag–“O” is 15391 which occupies for 71.7% total number of concept in the training set. Although these “O” tags play an important role in the performance of our model, we are particularly interested in the distribution of meaningful concepts which is visualized in figure 1. We could observe that more than a half of concepts is name of movies, followed by name of directors and name of actors which are around over 7%. There is some concepts that rarely appear in the dataset such as award category (appear only 1 time) or nationality of direction (appear 2 times).

3 Baseline method

3.1 Naïve Bayes classifier

The basic approach to solve the problem tagging is using Maximum Likelihood theory. Given a sequence of n tokens $\boldsymbol{w} = \{w_1, w_2, ..., w_n\}$, our goal is finding the best assignment $\boldsymbol{t} = \{t_1, t_2, ..., t_n\}$, in that each t_i corresponds to one w_i. Formally, our desired solution is casted to a classification problem as follow:

$$\begin{aligned} \boldsymbol{t}^* &= \operatorname*{argmax}_{\boldsymbol{t}} P(\boldsymbol{t}|\boldsymbol{w}) \\ &= \operatorname*{argmax}_{\boldsymbol{t}} \frac{P(\boldsymbol{w}|\boldsymbol{t})P(\boldsymbol{t})}{P(\boldsymbol{w})} \\ &= \operatorname*{argmax}_{\boldsymbol{t}} P(\boldsymbol{w}|\boldsymbol{t})P(\boldsymbol{t}) \\ &= \operatorname*{argmax}_{\boldsymbol{t}} \left(\prod_{i=1}^{n} P(w_i|t_i)\right) P(\boldsymbol{t}) \\ &= \operatorname*{argmax}_{\boldsymbol{t}} \left(\sum_{i=1}^{n} log(P(w_i|t_i))\right) P(\boldsymbol{t}) \end{aligned} \tag{1}$$

In the above equation, we have used Naïve Bayes method. Firstly we apply Bayes’ theorem and then we make a strong (naive) assumption that the probabilities of tokens are independent from each other given their own tags. Since the product of a chain of probabilities possibly leads to an extremely small value, we use the monotonic *log* function for stable computing.

3.2 Build model using OpenFST and OpenGRM

The equation 1 suggests that the basic structure to solve the tagging problem using OpenFST is a pipeline composing of three separable components. The first one is a Finite State Transducer (FST) which simply takes inputs as a sequence of tokens and outputs exactly what it received. The second one is a WFST which plays the role of concept tagger. This WFST takes output of the first

FST as input and produce arcs between states. In that each arc corresponds to one candidate concept tag with weight derived from the first term in the product of equation 1. The last FST is a language model of tags represented by $P(\boldsymbol{t})$. The final model could be written as:

$$sentence_{FSA/FST} \circ concept\_tagger_{FST} \circ \\ \circ\, concept\_langugae\_model_{FST}$$

with $\circ$ stands for compose operation.

Firstly we have to build a lexicon file containing all tokens and concept tags appearing in the training data. This lexicon set should also include token "<unk>" representing unknown words not in the training data, and "<epsilon>" indicating beginning of a sentence. Fortunately, this task could be simply done by calling function *ngramsymbols* provided in OpenGRM. After that we use function *farcompilestrings* to make the first FST in our model.

In next step we have to construct the second FST. The likelihood term in equation 1 is the fraction between number of times a tag appears and the occurrence of that tag and a token come together. This term could be computed by counting as follow:

$$log(P(w_i|t_i)) = \frac{Count(t_i)}{Count(w_i, t_i)} \qquad (2)$$

Nonetheless, there is an issue arise here. Our model works based on the viterbi algorithm (Forney, 1973) which is implemented under *fstshortestpath* function in OpenFST whereas we want to maximize the log likelihood in equation 1. We address this issue by easily converting values in equation 2 to their corresponding negative terms.

Finally, the language model in concept tags is an n-gram model. OpenGRM supports to construct language model through *ngramcount* and *ngrammake* function and allow to choose various number of gram and smoothing methods. All the functions in these libraries are usable through command line. Instead, we write a short wrapper so that our program is entirely written in Python language.

3.3 Experiment result

We run our model with {2,3,4,5}-gram and {"absolute", "katz", "kneser_ney", "presmoothed", "unsmoothed", "witten_bell"} smoothing methods to find best configuration by evaluation in the test set. This evaluation step is done by running an available file named "conlleval.pl" which compares predictions produced by our model with groundtruths then computes F1 score for every single concept tag and for all tags as well. The result is reported in table 2 and figure 2. We could see that in this baseline case, apparently increasing number of grams in language model does not help in improving the F1 score. All smoothing methods achieve best scores when using 2-gram and in several methods, higher grams degrade performances of models such as in "katz" and "presmoothed" methods.

Figure 3 shows how best baseline model–"absolute" method with ngram = 2—performs on each single concept in test set. We could observe that for underpresented concepts, either the model perfoms very effectively with 100% or does not able to learn to predict at all. Due to the domination of tag "O", our model tends to assign tokens with this tag. Therefore the recall scores are quite low consequently.

	ngram			
	2	3	4	5
absolute	**76.37**	75.67	76.19	76.15
katz	75.89	74.11	73.26	63.33
kneser_ney	76.27	75.67	76.19	76.16
presmoothed	76.27	67.95	67.01	66.89
unsmoothed	76.21	75.52	76.00	76.05
witten_bell	**76.37**	75.58	76.22	76.32

Table 2: F1 score obtained from baseline model

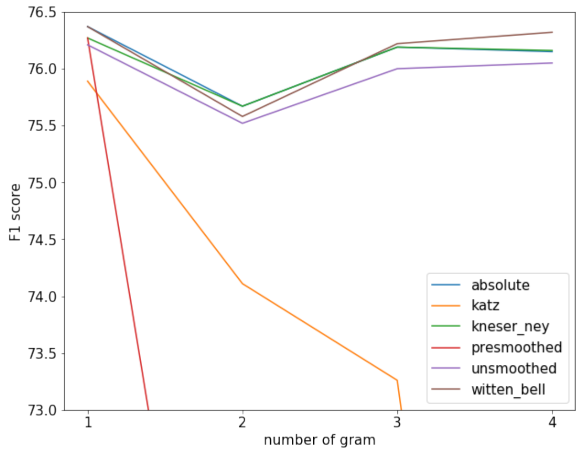

Figure 2: Line graph illustating F1 score of baseline method with various of configurations.

concept tag	precision	recall	F1
actor.name	76.19	80.00	78.0
actor.nationality	100.00	100.00	100.0
actor.type	100.00	100.00	100.0
award.category	0.00	0.00	0.0
award.ceremony	42.86	42.86	42.8
character.name	61.54	53.33	57.1
country.name	61.97	70.97	66.1
director.name	62.12	50.62	55.7
director.nationality	0.00	0.00	0.0
movie.genre	90.62	80.56	85.2
movie.gross_revenue	50.00	100.00	66.6
movie.language	77.19	63.77	69.8
movie.location	50.00	28.57	36.3
movie.name	86.61	82.03	84.2
movie.release_date	75.86	75.86	75.8
movie.release_region	50.00	25.00	33.3
movie.star_rating	0.00	0.00	0.0
movie.subject	72.09	70.45	71.2
movie.type	0.00	0.00	0.0
person.name	48.78	58.82	53.3
producer.name	78.69	65.75	71.6
rating.name	93.55	95.08	94.3

Table 3: F1 score of each concept tags obtained from baseline model with setting “absolute” method and ngram = 2.

	ngram			
	2	3	4	5
absolute	**75.72**	74.72	75.19	75.25
katz	75.40	72.98	72.10	62.48
kneser_ney	75.59	74.85	75.35	75.52
presmoothed	75.68	67.22	66.38	65.76
unsmoothed	75.62	74.70	74.86	74.94
witten_bell	**75.72**	74.76	75.09	75.31

Table 4: F1 score obtained from first variation: using lemmas

	ngram			
	2	3	4	5
absolute	78.93	81.37	81.57	81.34
katz	78.84	80.85	81.17	81.08
kneser_ney	79.45	82.04	**82.74**	82.43
presmoothed	77.67	79.54	79.77	79.47
unsmoothed	76.19	78.52	78.20	78.25
witten_bell	79.31	81.33	81.42	81.26

Table 5: F1 score obtained from second variation: using word-concept tags

4 Variations

4.1 Variation 1: using lemma

Since the lemmas of words are available in the dataset, our first hypothesis is that we can reduce number of concept tags to make our model more consistent and robust with different forms of words as we only pay attention to semantic meaning, not grammar. However, the F1 scores of all configurations drop down when we use lemmas instead of original words, as showed in table 4. We argue that word-form actually does bear information about concepts in our situtation, leading to over-fitting. Elbeit unimprovable, the best configurations are still the same with the baseline, since we does not modify the language model. Because this variation is not beneficial, we will not use it in next variations.

4.2 Variation 2: using word & concept tags

As described in section 2, the majority of tags in our dataset is “O”—out of concept. All the words not related to concepts in movie domain will be marked this tag. Which means that we treat many different words to be the same in our language model. This is not an appropriate approach since although they are not in the movie concept list, these out-of-concept words might bear important information to identify concept tags of surrounding tokens. For example, consider the situtation that we want to find the concept tags for token “X” in two following sentence: “I like all movies made by X” and “I like all movies starred by X”. Despite of the fact that all the first 6 words in both sentence have “O” tag, the word “X” should be director.name in the former and actor.name in the latter. However, in the baseline, our language model is not useful in distinguishing the differences between two sentences since it models the conditional probability $P(t_i|previous\ tags)$ and in this case, previous tags are the same in both sentences. For this reason, it would be proper to take into account the information conveyed by the tokens by combining the tokens and the tag “O” together.

Not surprisingly, as presented in table 5, all the configurations are boosted to higher scores. This time Kneser-Ney with ngram = 4 gives best F1 score since it was confirmed as one of the most effective smoothing methods widely used in community. For the other methods, setting with ngram = 4 also tends to achieve better performances given that the “O” tags now integrated with words therefore interior tags should bear more information about current tag.

	ngram			
	2	3	4	5
absolute	82.80	84.50	84.80	84.71
katz	82.80	83.75	83.99	83.20
kneser_ney	83.09	85.34	**85.92**	85.61
presmoothed	81.57	82.03	82.57	82.36
unsmoothed	80.32	81.54	81.82	82.06
witten_bell	82.62	84.25	84.68	84.76

Table 6: F1 score obtained from third variation: using entity recogizer.

4.3 Variation 3: using entity recogization

In this section, we will try to investigate the performance of our model if we use a named entity recongnizer prior to the concept tagger component, given that the concept tags are overlap. For example, one actor and one director could probably have a same name. For this reason, we hope that replacing some tokens by their named entities could be advantageous with better generalization. There are some libraries providing trained models that have good accuracies such as Spacy with 92.6% accuracy in NER problem for English language [1]. However, because all the names in training set are in lower case format, we are not able to use those libraries to detect named entities. Instead, we simulate this process by deriving named entities from concept tags. specifically, we replace phrases having tags related to human names, {"actor.name", "character.name", "person.name", "director.name"}, by "person.name", and replace phrases having tags related to countries, {"movie.location", "country.name"}, by "country.name". By doing this, we assume that there is a ideal model which makes no mistake on NER problem and we build our architecture on top of that NER model.

The result is showed in table 6. As expected, the F1 scores are remarkably improved in all configurations. We reach approximate 86% in F1 score with Kneyser-Ney method and ngram = 4.

5 Conclusion

In this project, we have studied how to apply Weighted Finite-State Transducer to the problem of concept tagging. We ran experiments on a dataset in movie domain and compared the performances between various settings of language model. In addition, we tried different variations derived from baseline model, including using lemmas of words, replace tag "O" with the combination with original tokens, and integrating an named entity recognizer into the model. The first variation worsened the baseline model since it made the model be over-fitting. In contrast, the formers enhanced the performance since they addressed the problem of overlapping concepts and prevented model from over-fitting.

However, due to the characteristic of the dataset that all token are normalized to lower cases, we could not able to plug a real NER module into our model. Instead, we simulated this module by derive named entities from concept tags. Besides, the dataset used in this project is small therefore the obtaind results had probabilties to over-fit the test set. In the future, it is worth to apply NER module from NLP libraries and see the behaviours of our models when generalizing to other datasets in other domains.

[1]https://spacy.io/usage/facts-figures

Acknowledgments

This report is a part of mid-term project in Language Understanding Systems course at University of Trento, 2019.

References

Yun-Nung Chen, Dilek Z. Hakkani-Tür, and Gökhan Tür. 2014. Deriving local relational surface forms from dependency-based entity embeddings for unsupervised spoken language understanding. *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 242–247.

G. D. Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Jacopo Gobbi, Evgeny Stepanov, and Giuseppe Riccardi. 2018. Concept tagging for natural language understanding: Two decadelong algorithm development.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech Language*, 16:69–88.