Concept Sequence Tagging for a Movie Domain
Language Understanding Systems, Mid-Term Project

Davide Pedranz
Mat. number 189295
davide.pedranz@studenti.unitn.it

Abstract

Concept Sequence Tagging is usually one of the first building blocks of a Spoken Language Understanding (SLU). The extracted concepts can be used to build more advanced systems. In this project, we developed a generative model based on Weighted Finite State Transducers (WFST) to extract concepts from sentences taken from the movie domain.

1 Introduction

Concept tagging can be defined as the extraction of concepts out of a given word sequence, where a concept represents the smallest unit of meaning that is relevant for a specific task. The extracted concepts can be used by the following blocks in a SLU pipeline to provide more complex functions, for instance, a personal vocal assistant or an automatic flights reservation system.

In this report, we will present a WFST generative model to extract concepts for a movie domain, build using the `OpenFst`[1] and `OpenGrm NGram`[2] libraries. We will briefly describe the used data set. Then, we will present the model. Finally, we will discuss the WFST implementation and the obtained performances.

2 Data Set

The data set used is NL-SPARQL, a corpora of sentences from the movie domain. For each word, the Part of Speech (POS) and the word stem are provided in addition to the concept tag. Example of concepts are the actor name or the movie release date. The aim is to train a model to assign the most probable sequence of concepts to each sentence.

[1] http://www.openfst.org/twiki/bin/view/FST/WebHome

[2] http://www.openfst.org/twiki/bin/view/GRM/NGramLibrary

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: Frequency of the concepts in the corpora.

2.1 NL-SPARQL Corpora

The corpora is already divided in the train and test sets. The train set consists of 3338 sentences, for a total of 21453 tokens. The test set is approximately a third of the train one and consists of 1084 sentences, for a total of 7117 tokens. The corpora contains 24 different concepts. Table 1 show their frequencies in the train and test sets. 9 concepts, for instance `director.nationality` and `award.category`, are very rare in the train

set, with a frequency smaller then 10. The concept `movie.type` is present only in the test set, so it is very hard for any model to predict it.

2.2 IOB notation

The concept tags follow the Inside, Outside, Beginning (IOB) format. Each concept can spawn on multiple contingent words. The `O` tag indicates that the word has not being assigned any concept. The `B-` prefix denotes the beginning of a chunk, the `I-` prefix indicates a chunk continuation. Each chunk is delimited by a `O`, the next `B-` tag or the end of the sentence.

2.3 Evaluation

The performances of the models are evaluated using the `conlleval.pl`[3], a Perl script originally developed to measure the performance of POS tagging the CoNLL-2000 corpora. The script can handle the IOB notation and computes precision, recall and F1 score for each concept and the overall performances for the entire test set.

3 Model

The model computes the sequence of tags $t_1, ..., t_n$ (also denoted as t_1^n) with the highest probability given the observed sequence of words $w_1, ..., w_n$:

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

Applying the Bayes' Law, we obtain:

$$t_1, ..., t_n = \underset{t_1,...,t_n}{\operatorname{argmax}} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)};$$

since the denominator does not depend on $t_1, ..., t_n$, we can drop it:

$$t_1, ..., t_n = \underset{t_1,...,t_n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n). \quad (2)$$

We assume that the probability of a word only depends on its own tag, and not on the tags of the other words in sentence. Thus, we have that:

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^{n} P(w_i | t_i). \quad (3)$$

Each $P(w_i|t_i)$ can be easily computed from the training data using maximum likelihood as:

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

where $C(t_i, w_i)$ denotes the number of times that word w_i has tag t_i and $C(t_i)$ denotes the frequency of tag t_i.

The term $P(t_1^n)$ represent the probability of a tag given all its predecessor in the sentence. To compute the term, we introduce a Markov assumption: the probability of a tag in the sentence depends only on the m previous tags:

$$P(t_1, ..., t_n) = \prod_{i=m}^{n} P(t_i | t_{i-m}, ...t_{i-1}), \quad (4)$$

where t_0 denotes the beginning of the sentence. A Markov assumption of order m corresponds to a $m+1$ n-gram language model.

Each term $P(t_i|t_{i-1})$ can be computed using maximum likelihood as follows:

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

Using Equation (2), (3) and (4), we can rewrite the model in Equation (1) as:

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

4 Implementation

The model described in Section 3 can be implemented as a WFST. In particular, the model can be decomposed in two transducers, one for each term. The final model is given by the composition of the two transducer.

4.1 First term: $\prod_{i=1}^{n} P(w_i|t_i)$

The first transducer $\lambda_{word2concept}$ assigns the tag with the highest probability to each word. The transducer has a single state 0 with many self transitions. For each unique pair (w_i, t_j), a self transition $w_i \rightarrow t_j$ is added to the transducer. The cost associated to the transitions is computed as the negative logarithm of the probability of the pair $P(w_i, t_j)$:

$$cost(w_i \rightarrow t_j) = -ln(P(w_i, t_j)).$$

In addition, we must take care of Out of Vocabulary (OOV) words, i.e. words that are never observed in the training data and thus not present in lexicon. Since we can not make any assumption about the concepts probability for unknown words, we choose a uniform distribution. Thus, we

[3] http://www.cnts.ua.ac.be/conll2000/chunking/output.html

add one extra self transition $\langle unk \rangle \rightarrow t_j$ for each concept t_j with cost:

$$cost(\langle unk \rangle \rightarrow t_j) = -ln\Big(\frac{1}{n_{concepts}}\Big).$$

$\langle unk \rangle$ is a special token used for the OOV words.

The model is implemented with a Bash script that computes all counts, costs and transition for the transducer. The lexicons are computed using `OpenGrm NGram`. The transducer is compiled to the `FST` format using `OpenFst`.

4.2 Second term: $\prod_{i=1}^{n} P(t_i|t_{i-1})$

The second transducer $\lambda_{concept\_lm}$ is a language model for the concepts. It takes as input a sequence of concepts and computes its probability given the predecessors and the observed training data.

A general problem with language models is data sparseness: since the language has an underlying structure and the training data is limited, not all tuples $(t_i, ..., t_m)$ are observed. As a consequence, most probabilities have value 0, which makes the probability of most words sequences 0 as well. To solve this problem, we introduce smoothing: the unseen tuples are assigned a small probability even though the real observed probability is 0. The probability of frequent tuples is slightly reduced to make the joint probability sum up to 1.

Many different smoothing methods have been proposed in the literature. A simple technique is the Laplace smoothing: we add imaginary training data with all possible n-gram combinations, each of them with frequency 1. For instance, for the bigram case, we can compute the smoothed probability as:

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

where V is the size of the vocabulary of pairs $(t_i|t_{i-1})$. Section 6 compares the performances of different smoothing methods on our model.

Opposite to the first term, we do not need to take care of OOV words, since the first transducer can produce only concepts in the concepts' lexicon.

4.3 Concept tagging

To compute the most probable sequence of tags, each sentence is first compiled to a Finite State Acceptor (FSA) $\lambda_{sentence}$. For each word w_i, we define a state i and a transition from state $i-1$ to state i that translate word w_i in itself. The initial state is 0, the final state is n, where n is equal to the length of the sentence. Figure 1 shows an example.

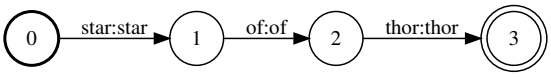

Figure 1: FSA for the sentence "star of thor".

The most probable sequence of tags can be computed as the shortest path in the Finite State Transducer (FST) given by the composition:

$$\lambda_{sentence} \circ \lambda_{word2concept} \circ \lambda_{concept\_lm}.$$

The shortest path is computed by the `OpenFst` library using the Viterbi algorithm. This model is implemented in the file `model/v1.sh`.

5 Language Model Improvement

The model presented in Section 4 does not exploit the information provided by the words that are tagged as `O`. In fact, all information about the original word is dropped by the transducer $\lambda_{word2concept}$. We can preserve it by including the word itself in the `O` tag, so that we can train a more accurate language model. This idea is implemented in the file `model/v2.sh`.

Before building the lexicon, the training data is modified to replace the `O` tags with `O-word`. The lexicon, $\lambda_{word2concept}$ and $\lambda_{concept\_lm}$ are build using the modified version of the training data. Finally, a new transducer $\lambda_{concept2iob}$ is added to translate the `O-word` tags back to `O`. This transducer has a single state 0, a self transition `O-word` $\rightarrow$ `O` for each word in the lexicon and a the self transitions `B-concept` $\rightarrow$ `B` and `I-concept` $\rightarrow$ `I` for each concept. All transitions have cost 0. For each possible input, there is only one possible transition.

6 Performances

6.1 Baseline

We use the first model with a unigram underlying language model without smoothing as the baseline to evaluate the performances of the different n-gram orders and smoothing methods. The baseline model simply assigns to each word the concept with the highest frequency in the training set for the specific word. The model reached a F1 score of 57.13%, as showed in the first line of Table 2.

6.2 n-gram Order

The model without smoothing has the best performances for bigrams and 5-grams, reaching respectively a F1 of 76.21% and 76.05%. The smoothing

n	smoothing	prec.	recall	F1
1	unsmoothed	54.64%	59.85%	57.13%
2	unsmoothed	78.39%	74.15%	76.21%
2	witten_bell	78.51%	74.34%	76.37%
3	unsmoothed	76.55%	74.52%	75.52%
3	witten_bell	76.58%	74.61%	75.58%
4	unsmoothed	77.15%	74.89%	76.00%
4	witten_bell	77.11%	75.34%	76.22%
5	unsmoothed	77.35%	74.79%	76.05%
5	witten_bell	77.03%	75.62%	76.32%
6	unsmoothed	76.72%	74.61%	75.65%
6	witten_bell	74.35%	73.05%	73.69%

Table 2: Comparison of the performances of different n-gram orders for model 1.

n	smoothing	prec.	recall	F1
2	absolute	78.51%	74.34%	76.37%
2	katz	78.03%	73.88%	75.89%
2	kneser_ney	78.41%	74.24%	76.27%
2	presmoothed	78.41%	74.24%	76.27%
2	unsmoothed	78.39%	74.15%	76.21%
2	witten_bell	78.51%	74.34%	76.37%
5	absolute	76.97%	75.34%	76.15%
5	katz	56.95%	71.31%	63.33%
5	kneser_ney	76.79%	75.53%	76.16%
5	presmoothed	62.36%	72.14%	66.89%
5	unsmoothed	77.35%	74.79%	76.05%
5	witten_bell	77.03%	75.62%	76.32%

Table 3: Comparison of the performances of different smoothing methods for model 1.

helps to get slightly better performances, but does not change the ranking of the n-gram orders. All models with n-gram order greater then 1 perform significantly better than the baseline, with an increase in the F1 score of about 20%.

6.3 Smoothing Methods

For both bigrams and 5-grams, the best smoothing method is Witten-Bell, followed by Absolute and Kneser-Ney. Table 3 shows a detailed comparison of all smoothing methods for the best n-gram order choices.

6.4 Features

The model can be trained with a different feature rather than the word. Table 4 compares performances of the 5-grams model with Witten-Bell smoothing for the different input features: POS and word stems. The best option is to use just the original words. Word stems have slightly worse performances than words. POS have terrible performances, with a F1 score of only about 20%.

feature	n	prec.	recall	F1
pos	2	17.91%	11.92%	14.31%
pos	5	22.51%	20.71%	21.58%
radix	2	77.96%	73.60%	75.72%
radix	5	76.12%	74.52%	75.31%
word	2	78.51%	74.34%	76.37%
word	5	77.03%	75.62%	76.32%

Table 4: Comparison of the performances of different input features for model 1, using Witten-Bell smoothing.

n	smoothing	prec.	recall	F1
3	kneser_ney	81.60%	82.49%	82.04%
3	witten_bell	80.04%	82.68%	81.33%
4	kneser_ney	82.44%	83.04%	82.74%
4	witten_bell	80.12%	82.77%	81.42%
5	kneser_ney	82.09%	82.77%	82.43%
5	witten_bell	80.07%	82.49%	81.26%

Table 5: Comparison of the performances of different parameters for model 2, using words as input.

6.5 Improved model

The improved model described in Section 5 achieves significantly better performances than the original model, with a F1 score around 82%. The best performances are obtained with a 4-gram underlying language model with Kneser-Ney smoothing, with a F1 value of 82.74%. Similarly to the original model, the additional features obtain worse performances. Opposite to the previous case, the best smoothing methods seems to be Kneser-Ney. More detailed results are showed in Table 5.

7 Conclusions

In this assignment, we created a generative model to do concept tagging. The best model was the one described in Section 5, using 4-gram and Kneser-Ney smoothing for the underlying language model. The inclusion of words in the O tags brought a significant performance improvement: the model reached 82.44% precision, 83.04% recall and a F1 score of 82.74% on the test set.