Sequence labeling with WFST
Language Understanding Systems

Emiliano Tolotti
Mat. 220230
University of Trento
emiliano.tolotti@studenti.unitn.it

Abstract

This document contains the report for the ”Concept Tagging for Movie Domain” project of the Language Understanding Systems course of the University of Trento. Sequence tagging is a type of structured learning which aims to assign meaning labels to words sequences. The approach used in this project to solve the task is shallow parsing with generative model.

1 Introduction

Sequence labeling is one of the most important tasks in NLP, which aims to find the underlaying meaning of a sentence. We need to map sequences of utterances to sequence of labels. To achieve this we will make use of a pipeline composed by 3 elements. In this report a WSFM approach to sequence labeling is presented, with the implementation of a concept tagging pipeline with generative language models using OpenFST[1] and OpenGRM[2] libraries. The report consists is a data analysis part, followed by the description of the pipeline and models architecture. Three models are proposed, one trained on tags, which we will call *basic* model, one with O-tags substitued with words called *advanced* model and one with word+tag pairs, so *joint* model. Then the performance results of the three models are displayed, then briefly compared and discussed.

2 Data analysis

Data analysis has been produced, to help exploring fundamental characteristic of the dataset. The proposed implementation uses the NL-SPARQL[3] corpus which contains lower-cased English tagged sentences about the movie domain.

[1]http://www.openfst.org/twiki/bin/view/FST/WebHome
[2]http://www.opengrm.org/twiki/bin/view/GRM/WebHome
[3]https://github.com/esrel/NL2SparQL4NLU

2.1 Dataset

The dataset follows the CONLL format, with tokens and concept tags pairs. The concept tagging in the dataset is in IOB (Inside, Outside, Beginning) format. The ”I-T” notation indicates that the tag T is inside a chunk. The ”O” tag indicates that the token has no chunk while ”B-T” indicates that the tag T is the beginning of a chunk.

The dataset is already splitted in train and test data partitions. The training set is composed of 3338 sentences, 1728 unique tokens, and 40 tags (+ O-tag). The test dataset is composed of 1084 sentences, 1039 unique tokens, and 38 tags (+ O-tag). As shown in figure 1 the average length of a sentence in the training set is 6.43. There are 2 unseen tags and 246 unseen tokens in the test set so the OOV rate is 23.7%.

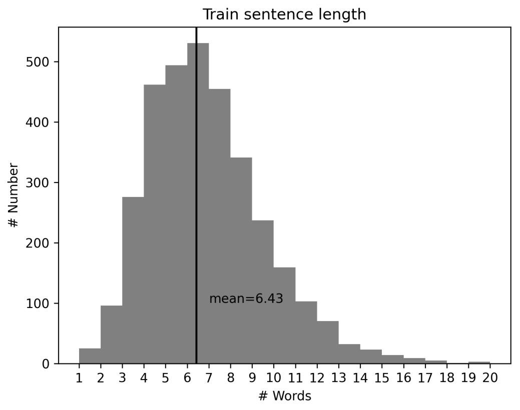

Figure 1: Sentence length distribution in train set

2.2 Concepts

There are 24 unique concepts in the dataset. As it is shown in figure 2, the distribution is similar between the training and test datasets, but there are

Word	Count	Frequency	Predicted
the	1337	0.06232	0.12449
movies	1126	0.05249	0.06225
of	607	0.02829	0.04150
in	582	0.02713	0.03112
movie	564	0.02629	0.02490
...	...	...	...
ryan	1	0.00005	0.00007
reba	1	0.00005	0.00007
kyra	1	0.00005	0.00007
skywalker	1	0.00005	0.00007
emma	1	0.00005	0.00007

Table 1: Words count in train set

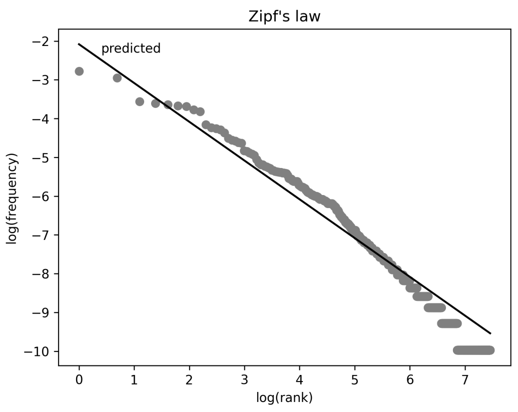

Figure 3: Zipfian distribution of train set

2 concepts in the training set which are not present in the test partition, and 1 concept present only in the latter. While *movie.name* is a very common concept, almost half of them have frequency less than 0.1%.

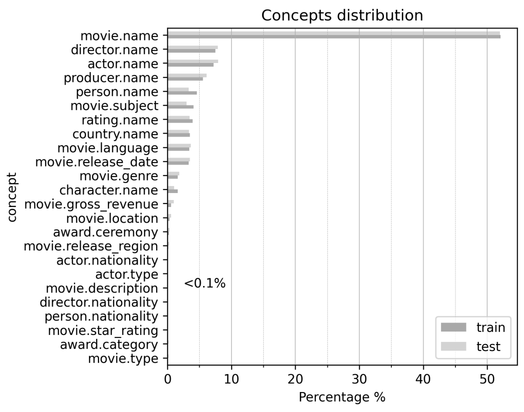

Figure 2: Distribution of concepts

2.3 Zipf's distribution

It can be seen that the train dataset follows a Zipfian distribution for the words, so the frequency of them in the dataset is inversely proportional to their rank. In fact, as shown in table 1, common words such as *the* or words associated to the movie domain such as *movies* have the highest frequency so the lower rank. On the opposite words such as names, for example *skywalker* that appears only once in the corpus, have high rank and low frequency. The graph in figure 3 shows the relationship between rank and frequency for the words in the train dataset. On the X-axis the rank of the words is reposted (in logarithmic scale), while on the Y-axis we have the frequency of the words (also in logarithmic scale).

3 Model

The goal of the model is to compute a sequence of tags $t_1, ..., t_n$ from a sequence of words $w_1, ..., w_n$.

$$t_1^n = \arg\max_{t_1^n} p(t_1^n|w_1^n) \tag{1}$$

Markovian assumption: We can make a Markov assumption and consider a tag as dependent only on the previous $k-1$ tags.

$$t_1^n \approx \arg\max_{t_1^n} \prod_{i=1}^{n} p(w_i|t_i)p(t_i|t_{i-k+1}^{i-1}) \tag{2}$$

3.1 Maximum Likelihood Extimation

WFSMs can be used to model HMMs and we can use weights of edges as probability costs. In this part we estimate probabilities from data using Maximum Likelihood Extimation (MLE) with frequency counts.

Emission probabilities: Emission probabilities of observing word w_i given tag t_i are computed as:

$$p(w_i|t_i) = \frac{c(w_i, t_i)}{c(t_i)} \tag{3}$$

Transition probabilities: Transition probabilities of observing tag t_i observing $k-1$ previous tags $t_{i-1}, ..., t_{i-k+1}$ are computed with a k-gram model of tags:

$$p(t_i|t_{i-k+1}^{i-1}) = \frac{c(t_{i-k+1}^{i})}{c(t_{i-k+1}^{i-1})} \tag{4}$$

3.2 Pipeline

The model described above can be implemented with a WFST. In particular we will make use of a pipeline with three components:

$$\lambda = \lambda_{word} \circ \lambda_{word2tag} \circ \lambda_{tagLM} \tag{5}$$

Sentence FSA: The first component of the pipeline λ_{word} represents a sentence compiled as a Finite State Acceptor (FSA).

Word to Tag transducer: The second component $\lambda_{word2tag}$ is a Finite State Transducer (FST) that maps every word to a tag based on the emission probability. The transducer is composed by a single state with self transitions for every $word, tag$ pair. The weight of the transitions is computed as a cost associated to the emission probability. In particular:

$$weight(w_i, t_j) = -log(p(w_i, t_j)) \tag{6}$$

We also need to map OOV (Out Of Vocabulary) words, which are not present in the train corpus, to tags. We cannot make assumptions on the probabilities, so they are managed with a transition for every tag with an uniform probability distribution.

Tags language model: The third component λ_{tagLM} is the language model for the tags, a finite state acceptor that scores the sentence based on the ngram transition probabilities. We need to take in account data sparsness, so there are sequences of tags $t_i, ..., t_j$ that are not observed in data, and to which the model gives probability 0. To solve this problem smoothing techniques are applied, to increase probability of unseen sequences.

The sequence labeling operation is performed computing the shortest path on λ the with the OpenFST library which applies the Viterbi algorithm.

3.3 Model variations

Three models have been implemented, the *basic* model trained on tags, the *advanced* model with O-tags substitued with the corresponding words and the *joint* model with word+tag pairs.

Basic model: The basic model uses the pipeline described above, using tags as they are. So the $\lambda_{word2tag}$ component maps words to tags with maximum likelihood extimation and the ngram language model λ_{tagLM} is trained on tags.

Advanced model: The advanced model uses words&tags as tags. In particular the O-tags are subsituted with the word they label, so we do not lose information. As described above, $\lambda_{word2tag}$ component maps words to words&tags with MLE and the ngram language model λ_{tagLM} is trained on words&tags.

Joint model: While the basic and advanced models use the pipeline as described above, the joint approach models the **joint probability distribution** of words and tags (Raymond and Riccardi, 2007), so it is implemented differently but it mantains the same 3 components pipeline structure. The $\lambda_{word2tag}$ component is substituted with an unweighted FST that maps words to words+tags and the ngram language model λ_{tagLM} is trained on words+tags pairs.

4 Performance

In this section we expose and compare the performance of the various models as well as the evaluations with different model parameters to find the best configuration. The evaluation has been performed with *conlleval*, a script developed by the CoNLL community for measuring performance of tagging systems using IOB scheme.

4.1 Model parameters

The models have been tested with different parameters to find the best configurations to obtain the best performances.

n-gram order: The language model has been trained with n-gram order form 1 to 5.

smoothing: The smoothing algorithms used are: *witten bell*, *absolute*, *katz*, *kneser ney*, *presmoothed* and *unsmoothed*.

frequency cut-off: The models have been evaluated with full lexicon (cutoff = 0) and with cut-off on words (cutoff = 2), so the lexicon has been truncated by words that appear less than 2 times in the corpus.

4.2 Model comparison

Basic model: Table 2 shows the performance obtained with the basic model with the best configurations of parameters. It is possible to see that the model with tags only performs well with a low ngram order. In fact the best F1 performance of **76.37%** is achieved with a bigram language model.

n	method	c	F1	Prec	Rec
2	witten bell	0	0.7637	0.7851	0.7434
2	absolute	0	0.7637	0.7851	0.7434
5	witten bell	0	0.7632	0.7703	0.7562
2	presmoothed	0	0.7627	0.7841	0.7424
2	kneser ney	0	0.7627	0.7841	0.7424

Table 2: Top-5 (F1-score) configurations performance for basic model.

n	method	c	F1	Prec	Rec
4	kneser ney	0	0.8274	0.8244	0.8304
5	kneser ney	0	0.8243	0.8209	0.8277
3	kneser ney	0	0.8204	0.8160	0.8249
4	absolute	0	0.8157	0.8041	0.8277
4	kneser ney	2	0.8156	0.8144	0.8167

Table 3: Top-5 (F1-score) configurations performance for advanced model.

In addition the top results are obtained with no cut-off. The best smoothing technique for this model is witten bell which performs well also with ngram size = 5.

Advanced model: The F1 performance increases by almost 6.4 points with the advanced model. In fact as it can be seen in table 3, the language model trained on tags and words obtains an F1 score of **82.74%**. The best performances have been reached with ngram size = 4 which is larger than the basic model, while lowering or increasing the value seems to decrease the score on the test set. The best smoothing method for this feature is kneser ney.

Joint model: The joint model scores in the middle between the basic and the advanced model. As it is shown in table 4, the best performances of **79.51%** in F1 score are reached with ngram size = 4 and kneser ney as smoothing algorithm. This smoothing method outperforms the others also when cutoff is applied with large ngram order.

n	method	c	F1	Prec	Rec
4	kneser ney	0	0.7951	0.7883	0.8020
5	kneser ney	0	0.7949	0.7871	0.8029
5	kneser ney	2	0.7905	0.7873	0.7938
3	unsmoothed	0	0.7896	0.7749	0.8048
2	witten bell	0	0.7896	0.7846	0.7947

Table 4: Top-5 (F1-score) configurations performance for joint model.

4.3 Discussion and error analysis

The best smoothing algorithms are *kneser ney* and *witten bell*, as they appear in all the top-5 tables and outperform the other smoothing methods also with multiple ngram orders.

Frequency cut-off does not help improving performance on this dataset, and it could be probably related to the fact that the corpus is of small size, and 45% of tokens appear only once, so the tokens count falls from 1728 to 950 with cutoff = 2.

Larger ngram order increases performance with the usage of more complex tag features, so with a better context. The information enhancement in the advanced and joint models tags allows to have a greater number of meaningful tags, and gives the opportunity to imporove the performance with larger ngrams with respect to the basic model which has a majority of O-tags (71.7%).

There are 2 unseen tags in the test dataset which are not predicted by the model, and many concept have low sample number so it is hard for the algorithm to correctly predict them with such a low context. There are also some misspelled tokens in the corpus that could lead to a lower prediction performance. In addition the high OOV rate can cause several mistagging errors.

5 Conclusion

The concept tagging model for the movie domain has been developed. The advanced model has shown to achieve the best performance among the 3 methods evaluated, and the best F1 score of 82.74% is achieved with 4-grams, kneser ney as smoothing method and no frequency cut-off. A possible improvement for the pipeline could be a generalization element, to preprocess tokens before applying the same pipeline. For example a custom NER model trained on movie-related names, such as movie names and actor names.

6 Appendices

Repository for the project:
`https://github.com/emiliantolo/sequence-labeling-wfst`

References

Christian Raymond and Giuseppe Riccardi. 2007. *Generative and Discriminative Algorithms for Spoken Language Understanding*.