# University of Trento - Language Understanding Systems Course

## Spring Project – Sequence Labelling using Stochastic Final State Transducers (SFSTs)

Federico Marinelli, 187419, federico.marinelli@studenti.unitn.it

## Abstract

The extraction of flat concepts out of a given word sequence is usually one of the first steps in building a spoken language understanding (SLU) or dialogue system. This project aims to evaluate the performance of labelling a word sequence using Stochastic Final State Transducers, adopting different features, smoothing algorithms, and other techniques in order to improve the baseline performances.

## 1 Introduction

The objective of this project is to develop a spoken language understanding system for the movie domain. It represents a classifier that, after the training on a sequence of sentences, has to produces predictions of the concepts for each word of the sentences passed as input, during the testing. The first section of this report describes the data analysis on which the project has been developed. The objective pursued is to increment the performances as much as possible, for this reason from the second section I start the description of all the different implementations, the error analysis and the subsequent changes that I've done. The last section regards the evaluation and the comparison between the implementations.

## 2 Data Analysis

The given dataset is named NL-SPARQL and it concerns the movie domain. It is already split in train and test files. In particular, there are two different instances of it: the first one is composed by a train-set and a test-set in token-per-line format, with tokens and concept tags for sequence labelling; the second one is composed by a train-set and a test-set with POS-tag and Lemmas in token-per-line format. This second data-set will be used to embed additional features for sequence labelling. Both of them have concepts that are represented through "BIO" encoding. It is a common format for tagging tokens in a chunking task, where we label each word by "B-X", "I-X" or "O". Here, "B-X" means "begin a phrase of type X", "I-X" means "continue a phrase of type X" and "O" means "not in a phrase".

### 2.1 Numbers about the dataset

First of all, it's important to analyse the numbers that concern the train-set and the test-set files. The train-set file have 21453 tokens and 3337 sentences, the test-set file contains 7117 tokens and 1091 sentences.


*Figure 1: Train-set and test-set words count*

### 2.2 Zipfian distribution

In this section I analyse if the trainset can be approximated with a Zipfian distribution. In the following figure is plotted the distribution of the words across the train-set, where in the X axis we have the words sorted by their length (on the right side of the graph we have the shortest words), and in the Y axis we have the relative word's frequency. It is possible to see that the words frequency into our dataset follow approximately the Zipf's Law, in fact lower rank words (i.e "the", "in") appears with a higher frequency (~6%) with respect to the higher rank words (i.e. "restrictions"). Since we're dealing with a movie-specific dataset, there are also present context-words like "movie", "movies", "produced" with a high frequency.

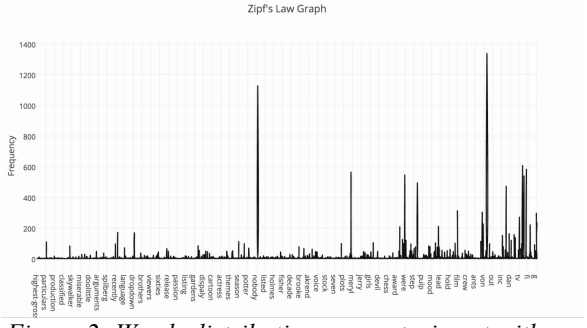


*Figure 2: Words distribution across train-set with respect to relative frequency*

### 2.3 Concepts distribution

During data analysis it is important to reason about the concepts distribution into the train-set and test-set (without B and I encoding). Considering the train-set, the percentage of words tagged with a concept is 28,25%, the remaining part is tagged with O's. In the next figure is it possible to see the plot of the distribution, where in the X axis we have the concepts, and in the Y axis the relative concept's frequency.

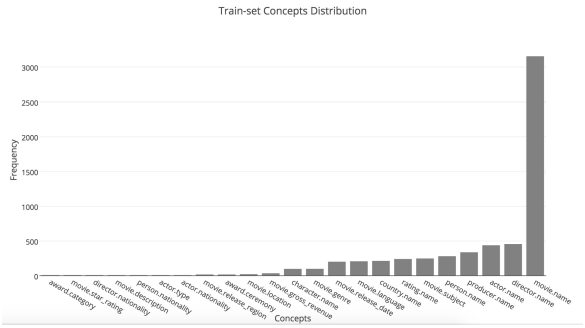


*Figure 3: Concepts distribution across the train-set*

Considering the test set, the percentage of words tagged with a concept is 27,84%. Some as above, in the next figure we can see the concepts distribution across the test-set.

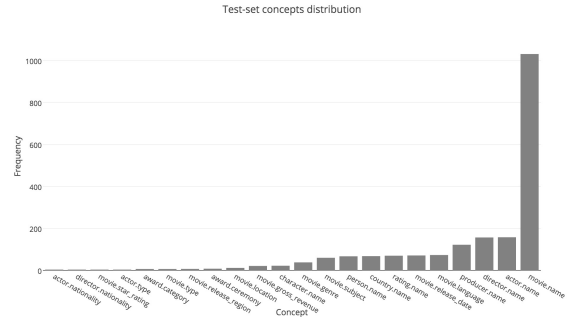


*Figure 4: Concepts distribution across the test-set*

As we can notice, the percentage of tagged concepts into the train and test set are quite similar. The most frequent concepts are the same for both and they follow the same distribution. This means that we are working with a dataset structured in a good way.

### 2.4 Dataset errors

During the inspection of the dataset some errors were found, there was either words typing errors or concepts tagging errors. They were left, as they weren't so many and, in addition, they represent noise. An example is the movie name "appolo thirteen" instead of "apollo thirteen", that is the correct one.

### 2.5 Stop-words

The analysis of stop-words could be useful in order to improve the performances, filtering them out before the classification. The ten most common words into the train-set are reported in the following table.

| Word | Frequency |
|---|---|
| *the* | *1337* |
| *movies* | *1126* |
| *of* | *607* |
| *in* | *582* |
| *movie* | *564* |
| *what* | *545* |
| *me* | *539* |
| *show* | *494* |
| *for* | *472* |
| *is* | *335* |

*Table 1: Stop-words - The 10 most common words into the train-set*

### 2.6 OOV Ratio

The out-of-vocabulary ratio indicates how many words in the test-set are not present in the train-set lexicon. The OOV ratio of the test set is 23.67% over 1039 unique tokens of the test set, which seems be high and this can cause several errors (empirically each OOV word cause 1.5/2 errors).

## 3 Stochastic Final State Transducers (SFSTs)

The tools used in this project are the following:

- *OpenGrm*: used for making and modifying n-gram language models encoded as weighted finite-state transducers.
- *OpenFst*: used for compiling and composing transducers.

The project code is written in Bash language. Python and Perl languages are used during the evaluation phase. In this phase we used a script called conlleval.

### 3.1 General idea

SFST based SLU is a translation process in which stochastic language models are implemented by Finite State Machines (FSM). These FSMs are transducers that take words as input and output the concept tag conveyed by the accepted phrase. Given a new sentence W and its FSM representation λW, the translation process is to find the best path of the transducer resulting of the next composition:

$$\lambda SLU = \lambda W \circ \lambda w2c \circ \lambda SLM$$

Where λw2c represents the likelihood machine that, given a words sequence, outputs the relative concept for each word. The λSLM represents the language model, it is the prior probability relative to each concept.

### 3.2 Implementation

In order to generate the λw2c machine we have to calculate the probability of $P(w_i|c_i)$. For a known word $w_i$, the probability is equal to:

$$P(wi|cj) = \frac{Count(wi, ci)}{Count(ci)}$$

Where $Count(w_i,c_j)$ is the count of the word with a particular tag and $Count(c_j)$ is the number of appearance of the considered tag in the train-set. For an unknown word the probability is equal to:

$$P(wi|cj) = \frac{1}{|Concepts|}$$

Where $|Concepts|$ is the number of concepts within the train-set. Note that the unknown words have been considered as well.

We have to finally apply the –ln to the resulting probabilities in order to embed it as a cost into the transducer.

The lexicon is generated from the concepts and tokens that the λw2c machine take as inputs from the train-set.

Once we have the likelihood machine ready, we have to generate a FST for each sentence within the trainset (they represent the transducer named λW), and we compose each of them with the λw2c transducer. At this point we compose this resulting transducer with the language model λSLM, generated from the sequences of all concepts within the train-set, where each concept is related to a specific word into a words sequence, that represents a sentence. The shortest path of this final transducer will be the output, where we predict, for each word of a sentence, the related concept. During the compilation of the language model we can choose different parameters, like the order (unigram, bigram, trigram, etc..) or the smoothing algorithms (kneser ney, witten bell, etc..). These choices can of course drastically change the final performances.

### 3.3 Baseline

The baseline tells us whether a change is adding value. Once we have a baseline, it's possible to add or change the data attributes, the algorithms we are trying or the parameters of the algorithm, and know whether we have improved our approach or solution to the problem. Our baseline will be the performances obtained with a basic SFSTs implementation explained in this section.

The baseline performances are obtained using the implementation explained in the previous sub-section. The LM machine is compiled using unigrams probabilities and without a smoothing algorithm. The results are reported in the following table.

| Method | Order | Accuracy | Precision | Recall | FB1 |
|---|---|---|---|---|---|
| Unsmothed | 1 | 88.90% | 54.85% | 60.58% | 57.58% |

*Table 2: Baseline performances without smoothing algorithm and unigrams probabilities for the LM*

## 4 Experiments

In this section I report the results obtained by using difference features, smoothing algorithms and other techniques in order to improve the baseline performances.

### 4.1 Changing smoothing algorithms and orders

In this section I report the results using the basic implementation, changing the smoothing algorithms and the related orders. In the last column I report the absolute gain obtained from the baseline FB1.

| *Basic Implementation* | | | | | | |
|---|---|---|---|---|---|---|
| Method | Order | Accuracy | Precision | Recall | FB1 | Absolute Gain FB1 |
| Kneser Ney | 2 | 92.68% | 78.41% | 74.24% | 76.27% | 18.69% |
| Kneser Ney | 3 | 92.64% | 76.67% | 74.70% | 75.67% | 18.09% |
| Kneser Ney | 4 | 92.78% | 76.87% | 75.53% | 76.16% | 18.58% |
| Witten Bell | 2 | 92.68% | 78.51% | 74.34% | 76.37% | **18.79%** |
| Witten Bell | 3 | 92.62% | 76.58% | 74.61% | 75.58% | 18.00% |

*Table 3: Results obtained from the basic implementation*

I obtained an absolute gain of 18.79% from the baseline, these results seem a-priori reasonable good.

### 4.2 Frequency cut-off smoothing

Given a hyper parameter as threshold T, I've cut off all the words appearing less or equal to such a threshold, accumulating a counter of deleted words. Then we have to subdivided the accumulated counter between all the concepts and use them to compute the probabilities. It could be interpreted as a noise cut-off. The results are reported in the following table. The threshold T was set to 1. As it is possible to see, with this method we didn't obtained any reasonable improvements, compared to the baseline performances.

| *Frequency Cut-off Smoothing* | | | | | | |
|---|---|---|---|---|---|---|
| Method | Order | Accuracy | Precision | Recall | FB1 | Absolute Gain FB1 |
| Kneser Ney | 3 | 88.66% | 51.23% | 66.82% | 58.00% | 0.42% |
| Witten Bell | 2 | 88.93% | 52.08% | 67.64% | 58.85% | **1.27%** |
| Witten Bell | 3 | 88.63% | 51.27% | 66.36% | 57.85% | 0.27% |

### 4.3 POS as a feature

In order to improve the performances, I modify the train-set substituting each word with a joint union of a word and the related POS, obtained from the second instance of the train-set ("feast.txt" instance). We have to do the same edit also for the test-set in order to perform the evaluation. As we can see we obtained an absolute gain of 16.92% from the baseline, but compared to the previous work it's worse (section 4.1).

| *POS as feature* | | | | | | |
|---|---|---|---|---|---|---|
| Method | Order | Accuracy | Precision | Recall | FB1 | Absolute Gain FB1 |
| Kneser Ney | 2 | 92.13% | 77.19% | 71.95% | 74.48% | 16.90% |
| Kneser Ney | 3 | 92.08% | 75.19% | 72.50% | 73.82% | 16.24% |
| Kneser Ney | 4 | 92.20% | 75.52% | 73.51% | 74.50% | **16.92%** |
| Witten Bell | 3 | 92.09% | 75.26% | 72.50% | 73.86% | 16.28% |

*Table 5: Results obtained by using POS as a feature.*

### 4.4 POS and Lemmas

In this section I report the performances obtained by using POS and Lemmas jointly, instead of the original

words. Same as above they are taken from the second instances of the train and test set ("feats.txt" instance). The results are reported in the following table. Same as above, we did not obtain better results compared to the previous work (section 4.1).

| *POS and Lemmas* | | | | | | |
|---|---|---|---|---|---|---|
| Method | Order | Accuracy | Precision | Recall | FB1 | Absolute Gain FB1 |
| Kneser Ney | 2 | 92.15% | 77.26% | 71.95% | 74.51% | **16.93%** |
| Kneser Ney | 3 | 92.03% | 75.12% | 72.23% | 73.64% | 16.06% |
| Kneser Ney | 4 | 92.20% | 75.52% | 73.33% | 74.42% | 16.84% |

*Table 6: Results obtained by using POS and Lemmas instead of original words.*

### 4.5 Lemmas instead of words

Also in this case, using lemmas instead of words, I did not get better results, compared to the ones obtained in the section 4.1.

| *Lemmas insted of Words* | | | | | | |
|---|---|---|---|---|---|---|
| Method | Order | Accuracy | Precision | Recall | FB1 | Absolute Gain FB1 |
| Kneser Ney | 2 | 92.40% | 77.79% | 73.51% | 75.59% | **18.01%** |
| Kneser Ney | 3 | 92.29% | 75.94% | 73.79% | 74.85% | 17.27% |
| Kneser Ney | 4 | 92.47% | 76.19% | 74.52% | 75.35% | 17.77% |
| Witten Bell | 3 | 92.27% | 75.85% | 73.69% | 74.76% | 17.18% |

*Table 7: Results obtained by using lemmas instead of words.*

### 4.6 Error and data analysis

During the inspection of the errors done by classifier, and looking into the train-set itself, I noticed different insights.

1. The classifier often confused concepts like *procuer.name*, *person.name* and *director.name*, which represent also the most distributed concepts in the train-set. It's a problem related to the Name Entity Recognition.

2. I've discovered that, in the train-set, a sequence of words that represents a concept, like *producer.name* and *director.name,* was usually preceded, by the words "produced by" and "directed by". Another insight, for instance, is the word "starring" that precedes many times the concept *actor.name*, and this kind of "behavior" appears also for other concepts.

### 4.7 Concept clustering

During the error analysis, as described in the previous section, I noticed that the classifier wasn't very good when it had to discriminate words that could represents the same concept (i.e. director.name, person.name, producer.name). In order to solve this issue, I thought to create "cluster of concepts", instead to tag directly words to concepts. In order to do this implementation, I've inserted an intermediate step in which words are grouped into clusters of concepts and then these clusters are re-mapped into the concepts, the transducer used for this operations are three. The first one is the transducer that map words into an intermediate-concept that represents a cluster (i.e name). The language model used is the same one already explained in the Section 3. Transducers compositions are done with the order as they are presented and, given a word sequence, the most probable labels are retrieved by using the shortest path. Unfortunately, this idea didn't give reasonable results in term of performances (compared to the results obtained in the Section 4.1) and they have not been inserted into the report.

### 4.8 Adding a-priori knowledge as prior probability

The insights raised after the error analysis lead me to reason about if, same concepts, were usually preceded by the same "word pattern", and then find the discriminative power of these patters. (i.e. the concept *producer.name* was usually preceded by the words "produced by"). In order to this evaluation I've computed the ratio of appearances of a word behind a concept, over the whole appearances of the word in the training set. This ratio helped me to understand, for each concept, which are the most related preceding words and how much these words interfere with other concepts (i.e. the higher the ratio, the most is the word related to the concept and the less interfere, since it appears the most of the times behind such concept). This reasoning was also extended to 2 preceding words, and I found very interesting insights. I used these insights to embed a-priori knowledge into the language model. To add these evidences, I have to modify the train-set, simply changing the concepts related to the words analysed, from the O notation, to the word itself. (for simplicity I substituted the O's with $-"related-word"). Of course before the evaluation I have to restore these concepts to the original one (the O notation). With this method I got better results in term of performances (the system is able to discriminate better while labelling, for instance, between *director.name* and *producer.name*) and this reasoning lead me to a new idea, discussed in the next section.

### 4.9 Adding full knowledge as prior probability

While I was adding a-priori knowledge, as described in the previous section, I saw that more O's concepts I substituted, more the overall performances were increasing. Then I decided to modify all the concepts, that appears as O's within the train-set, with the related word itself. This implementation gave me the best results and they are reported in the following table.

| Method | Order | Accuracy | Precision | Recall | FB1 | Absolute Gain FB1 |
|---|---|---|---|---|---|---|
| **Kneser Ney** | **4** | **94.94%** | **82.35%** | **82.95%** | **82,65%** | **25,07%** |
| Kneser Ney | 3 | 94.83% | 81.50% | 82.40% | 81,95% | 24,37% |
| Kneser Ney | 5 | 94.90% | 82.00% | 82,68% | 82,34% | 24,76% |
| Witten Bell | 4 | 94.41% | 80.04% | 82,68% | 81,33% | 23,75% |

*Table 8: Results obtained by substituting all the concepts that appears as O's with the line-related word*

As it is possible to notice, the best performances are obtained by using the Kneser Ney smoothing algorithm with 4-grams for the language model. I obtained an **F1 measure of 82,65%,** with an **absolute gain, form the baseline, of 25,07%.** As soon as I got this results I thought if this methodology could cause overfitting, that's because I'm adding a lot of relations between words and concepts. In order to check if I was overfitting the train-set I performed a cross-validation, in particular a 10-fold cross-validation with the train-set shuffled. The results are reported in the following table.

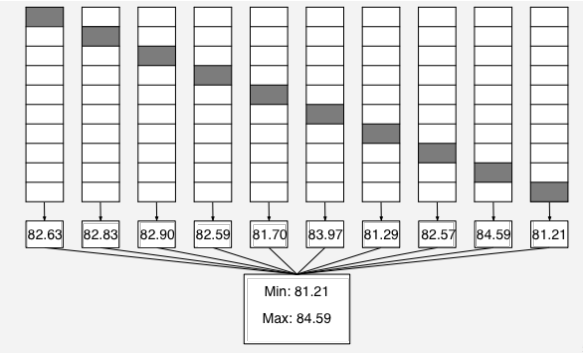


*Figure 5: Cross-Validation Results. I've reported the F1-measure as metric.*

It's possible to say that there aren't signs of overfitting, in fact the F1-measure vary between 81.21% and 84.59% (with an average of 82.9%).

## 5 Evaluation and Conclusion

During this report I've reported the results with accuracy, precision, recall and F1-measure as metrics. The most important between them is the F1, since the accuracy don't take into account the unbalancing between the classes.
During this project I've collected several results and I've derived the following deductions:

- The best smoothing algorithms, concerning this assignment, are *Witten Bell* and *Kneser Ney*, in fact I've reported as results only the ones that I've obtained with them. With the other available algorithms (pre-smoothing, absolute-discounting, Kats) I've obtained a drop of F1-measure of 5/6%, probably Witten Bell and Kneser Ney work better because they are interpolation algorithms.

- In general, 2-grams and 3-grams appeared very good, with respect to, primarily unigram, but also higher n-grams. By the way, the best performances are obtained with a 4-grams.

- A frequency cut-off strategy into the word-to-concept machine didn't help, probably we need more data to see better results.

- Filter the stop-words out before the classification didn't help.

- Adding features like lemmas and POS's into the train-set didn't improve the performances, rather they dropped the F1 of 1/2%. The reasons could be several, it's also important to remember that they were automatic generated.

- The data analysis and mostly the error analysis helped me to find very good insights into the train-set. It definitely represents a good lesson for me, and lead me to improve the overall performances.

- The idea of cluster of concepts seemed very good a-priori, mostly after the error analysis, but it did not add any additional value to the project.

- The best results are reached using 4-grams and Kneser Ney as smoothing algorithm, with a full knowledge setting as prior probability (as explained in Section 4.9). It increased the F1-measure of 25.07% with respect to the baseline, with a value of 82.65%. This is probably due to the high frequency of O's concepts which in this version has been completely translated into concepts, giving useful statistics to the overall classification.

## Appendices

The project can be found on GitHub at the following link: *https://github.com/feedmari/LUS-Spring-Project*

## References

Kyle Gorman. 2016. Opengrm.
*http://www.opengrm.org/*
*Kyle* Gorman. 2017. Openfst.
*http://www.openfst.org/twiki/bin/view/FST/WebHome*