# Sequence Labelling with WFSTs

Nicolò Alessandro Girardini, Mat. number 203265
nicolo.girardini@studenti.unitn.it

***Abstract*—This is the report of the first project of the course Language Understanding Systems (LUS). It involves the development of a sequence labelling model, trained on a given dataset. The task has been completed by developing and improving a basic model into a more advanced one, achieving a good improvement. All the code and extended results can be found at the public repository [1]**

## I. Introduction

This project focuses on implementing a model able to perform **IOB concept tagging** (Inside, Outside, Beginning). The dataset is **NL2SparQL4NLU** [2] which has as context movies: it is already divided into training and testing set. The resulting model will then be able to distinguish concepts such as actors and movie names. The task has been achieved and in this report the basic model will be described along with its optimal parameters and the process to develop from it a more advanced model will be discussed too.

The tools used to handle the language models and the automata are *openGRM* and *openFST*, while for scripts *Python* has been used.

## II. Dataset Analysis

To help getting started with the task and then improving the models it is necessary to know some basic information about the dataset. It contains sentences within the context of movies and these sentences are in the form of a query. Each of the sentences' words is tagged with the **IOB** format:

- **O:** these are the words out of context
- **B-concept.type:** these are the words that indicate the beginning of a concept (actor, movie director, etc.). Concepts composed by only one word will have this as tag
- **I-concept.type:** these are the words inside the concept. The ending word of the concept will still have this tag

The data is divided in training and test set. More can be found at [2].

The first statistic to look into is how big the training and testing datasets are. It is also interesting to see whether the average length of the sentences is long or short.

Table I
Sentence Number and Average Length

| Data | Number of Sentences | Average Length |
|---|---|---|
| Train | 3338 | 6.43 |
| Test | 1084 | 6.57 |

The test then is around one fourth of the entire dataset. It is interesting to notice that these sentences have the same average length in the two sets and that they are pretty short w.r.t. the average of around 17 of standard english [3]. Part of this is probably because these are mostly structured like queries and it will be important for the model.

The following is to look at the tags distribution, so their frequency:

Table II
Tag Distribution in Train and Test Sets

| Tag | Train Frequency | Test Frequency |
|---|---|---|
| O | 15391 | 5135 |
| I-movie.name | 1755 | 557 |
| B-movie.name | 1402 | 473 |
| B-director.name | 237 | 78 |
| B-actor.name | 227 | 80 |
| I-director.name | 218 | 78 |
| I-actor.name | 210 | 77 |
| B-rating.name | 200 | 61 |
| B-producer.name | 195 | 73 |
| B-country.name | 190 | 62 |
| B-movie.language | 174 | 69 |
| B-movie.subject | 172 | 44 |
| B-person.name | 151 | 34 |
| I-producer.name | 141 | 48 |
| I-person.name | 129 | 32 |
| I-movie.release_date | 114 | 42 |
| B-movie.genre | 94 | 36 |
| B-movie.release_date | 87 | 28 |
| I-movie.subject | 75 | 15 |
| B-character.name | 49 | 15 |
| I-character.name | 48 | 6 |
| I-rating.name | 40 | 8 |
| I-movie.language | 33 | 3 |
| I-movie.gross_revenue | 26 | 15 |
| I-country.name | 22 | 5 |
| B-movie.location | 17 | 7 |
| B-award.ceremony | 10 | 7 |
| B-movie.release_region | 9 | 4 |
| B-movie.gross_revenue | 8 | 5 |
| B-actor.nationality | 5 | 1 |
| I-movie.location | 4 | 4 |
| I-movie.genre | 4 | 1 |
| B-actor.type | 3 | 2 |
| I-award.ceremony | 3 | 0 |
| B-director.nationality | 2 | 1 |
| B-person.nationality | 2 | 0 |
| B-movie.description | 2 | 0 |
| I-actor.nationality | 1 | 0 |
| I-movie.release_region | 1 | 2 |
| B-movie.star_rating | 1 | 1 |
| B-award.category | 1 | 2 |
| B-movie.type | 0 | 4 |
| I-award.category | 0 | 2 |

It will be important for the modelling to notice that the large majority of the tags are **O**, out-of-concept, 71.74% for the training set and 72.15% for the test set. It is also worth noting that a good amount of concepts have a very low frequency, and not only inside concepts (which could be less common

if a concept is usually one word): it is hard to train a model to recognise these. The frequency distribution is pretty much consistent (frequent tags in training set are frequent in test set) but is is worth noting that some tags are not present in both sets: that's why it is important to also handle unknown tags.

## III. Base Model

The final task the model will accomplish is to take a sentence (ordered list of words) as input and output the ordered list of labels (or tags or concepts) corresponding to those words: this is mapping words to concepts. Formalizing, given the sequence of words $(w_1, ..., w_n)$, find its most probable sequence of tags $(t_1, ..., t_n)$:

$$(t_1, ..., t_n) = \underset{t_1,...,t_n}{\arg\max} P((t_1, ..., t_n)|(w_1, ..., w_n))$$

Bayes's Law can be applied and since the probability of word sequence, $P(w_1, ..., w_n)$, is the same for all possible tag sequences it can be omitted, obtaining:

$$(t_1, ..., t_n) = \underset{t_1,...,t_n}{\arg\max} P((w_1, ..., w_n)|(t_1, ..., t_n))P(t_1, ..., t_n)$$

The next step is observing that the tag of the word does not depend from other words' tags. Also, treating the model as a **Hidden Markov Model** (with tags as states) and using a k-order Markov assumption, the probability of the tag is only dependent on the previous k tags, we end up with the final formalization of the model (in this case with a second order assumption):

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

Since a **maximum likelihood** approach is used, the normalized frequencies of word-tag and tag-tag combinations will estimate their probability:

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

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

By the formalization it is easy to see that the model implementation can be split in two parts: the first one can be implemented with a word-to-concept **Finite State Transducer**, while the second one is a **Language Model** so that the Markov assumption is handled by its order, namely by **n-grams**.

### A. Base Model Implementation

The first step is always to compute a **lexicon** of the dataset (a vocabulary of known words). It is also needed to take care of the **unknown words**. The generation of the lexicon can include a **cut-off**, where words that are seen few times are not included because there is not enough information and the model could overfit on them. This model has been tried with and without cut-off, with results discussed in Section III-B.

Following this the FST word-to-concept has been generated. Since they work with costs as edge weights, the negative log-probabilities (natural logarithm) of the pairs word-concept were used. To handle the probability of the unknown words, each of the possible pairs $(< unk >, tag)$ has been given the same probability:

$$cost(< unk >, tag) = -log(\frac{1}{|tags|})$$

Note that this model would work even by not specifying the probabilities of unknown words: in that case, though, sentences containing them couldn't be tagged and only sentences with all words belonging to the vocabulary could be tagged (because the automata wouldn't have any more transitions and it would be in a non-final state).

The second part can now be addressed: a language model will be trained on each sentence's tag sequence. This will model the probabilities of the tag-tag tuples and it's first parameter will be the **order** of the n-gram, so how many of the previous tags are considered to estimate the probability of the current tag. The second parameter is the **smoothing method**: this is needed because when evaluating the probabilities of the sequence many of the possible tag-tag tuples have 0 probability. Using a smoothing method allows to estimate such probabilities and compute a complete language model.

Finally, to test the model, all the test sentences has been transformed into FSA (acceptors) and composed first with the word-to-tag FST and then language model. The result is again an FST and by taking its shortest path (with the smalles cost), thanks to the **Viterbi algorithm**, the result will be the most probable word-concept mapping.

### B. Base Model Results

In this base model only two parameters can be tested: the **n-gram order** and the **smoothing method**. To compare them all the main measure considered is the **F1 score**, which combines precision and recall and it is usually more meaningful than just accuracy or any of the other measures when comparing models.

We can consider as baseline (for both this and the advanced model) the model with **unsmoothed** method and order **1**, reported in Table III.

Table III
Baseline Model

| Method | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Unsmoothed | 0.8900 | 0.5513 | 0.6059 | 0.5773 |

The smoothing methods don't have substantial differences while the order has more impact on the result. In Table IV it is possible to see the results for the different methods used (the order used is 2, which resulted to be the best).

As we can see the Absolute method and Witten Bell are the ones with the better results (almost identical), but there is no a very big difference with the other methods (with Katz performing a bit lower than the others and being the first one to drop when increasing the order). These are also the ones with the best results while increasing the n-gram order: Table V (the method used is absolute) shows that as stated order 2 is

Table IV
BASE MODEL SMOOTHING METHODS

| Method | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Katz | 0.9262 | 0.7803 | 0.7388 | 0.7589 |
| Presmoothed | 0.9265 | 0.7841 | 0.7424 | 0.7627 |
| Kneser Ney | 0.9268 | 0.7841 | 0.7424 | 0.7627 |
| Witten Bell | 0.9268 | 0.7851 | 0.7434 | 0.7637 |
| Absolute | 0.9269 | 0.7851 | 0.7434 | 0.7637 |

the one achieving better results (with a big difference w.r.t. unigrams). This is also probably due to the fact that most of the sentences are not very long and thus bigrams are enough to model this data.

Table V
BASE MODEL N-GRAM ORDER

| Order | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| 1 | 0.8900 | 0.5513 | 0.6059 | 0.5773 |
| 2 | 0.9269 | 0.7851 | 0.7434 | 0.7637 |
| 3 | 0.9265 | 0.7667 | 0.7470 | 0.7567 |
| 4 | 0.9285 | 0.7716 | 0.7525 | 0.7619 |
| 5 | 0.9288 | 0.7697 | 0.7534 | 0.7615 |
| 6 | 0.9289 | 0.7704 | 0.7534 | 0.7618 |
| 7 | 0.9226 | 0.7424 | 0.7424 | 0.7424 |

It was surprising to see the results still being pretty consistent with such high order n-grams. The guess here is that probably there are sentences that very often have the same structure when they have the same concepts and so when handling correctly the tag tuples with 0 probabilities it is still possible to achieve good results.

The last explored option was the **cut-off**. It has been tested with n-gram order 2 and method Absolute, while the cut-off eliminated words with frequencies equal or lower than 1, 2, 3. As Table VI reports that the model gets worse: this is probably caused by the amount of names in the dataset and the fact that many concepts have low frequency (and probably are associated with low frequency words). These are not very likely to repeat but they are for sure associated with concepts and so they are very important for the training of the model.

Table VI
BASE MODEL CUT-OFF

| Cut-Off | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| None | 0.9269 | 0.7851 | 0.7434 | 0.7637 |
| 1 | 0.9178 | 0.7799 | 0.7049 | 0.7405 |
| 2 | 0.9045 | 0.7655 | 0.6581 | 0.7077 |
| 3 | 0.8942 | 0.7577 | 0.6104 | 0.6761 |

The best result with this model was then obtained by using order **2** with the **absolute method** with **no cut-off**. The complete results can be found in [1].

## IV. ADVANCED MODEL

This model starts from the base one and adds several improvements. Firstly, it model the **joint probabilities** of words and tags, as proposed in [4]. This allows to take into considerations the words and not only tags in the transducer: it is very important, because it is a way of adding more information to the *O* tags, which are the most frequent tags and the ones which have no concept information. This can then help to better distinguish between words that are often associated with a concept and words that have a low probability to be associated with concepts. The new model will be trained to generate (word, tag) pairs, so its formalization is now (always with second-order assumption):

$$((w_1,t_1),...,(w_n,t_n)) = \arg\max_{(w_1,t_1),...,(w_n,t_n)} \prod_{i=1}^{n} P(w_i|(w_i,t_i))P((w_i,t_i)|(w_{i-1},t_{i-1}))$$

It is evident that $P(w_i|(w_i,t_i))$ always has the value 1: this changed a bit the implementation, since weights are not needed anymore for the FST. The second probability is still computed by using a language model.

It also adds input normalization and tries to make use of **POS tags** to improve its performance.

### A. Advanced Model Implementation

Following the pipeline order, the first thing is the **input normalization**. This has been done by transforming all digits to a default string, $< number >$. The second normalization was to change all words into their **lemmas**: this is actually a choice of feature which will be discussed in Section IV-B. The input normalization is also applied to the test set before any other operation.

As said in Section IV, the first probability term is always 1, so an unweighted FST has been used.

The $< unk >$ words still have a transition to any of the possible concepts, which are now treated differently. The joint model uses the combination of word and concept: this is simply implemented by using as concept a string with word and concept: $word + Concept.type$. The whole model will then predict word and concept jointly and to retrieve the concept it suffices to split the string. When POS tags are considered they enhance even more the concept, making it $word + Concept.type + POS$.

### B. Advanced Model Results

The evaluation here has to take in account not only the parameters, n-gram order, the smoothing method and cut-off but also features, such as lemmas and POS tags.

The evaluation here has been different. Many combinations of features (lemmas, POS tags) and parameters were possible. The model was run with all these possible combinations (with n-gram order between 1 and 10). Then all the combinations have been ranked w.r.t. their F1 score.

Following this, Table VII shows the top scoring order and smoothing for each feature combination.

From these results it is evident that **lemma** is a feature that improves the model: in fact the first 38 combinations of parameters have lemmas and not POS tags. Overall the models with lemmas perform better than models without it, even if the difference is pretty small. Having, instead, the **POS tag**

Table VII
ADVANCED MODEL FEATURES COMBINATIONS

| Lemmas | POS tags | Order | Smoothing | F1 Score |
|---|---|---|---|---|
| NO | NO | 5 | Kneser Ney | 0.7926 |
| NO | YES | 5 | Kneser Ney | 0.7931 |
| YES | YES | 5 | Kneser Ney | 0.7958 |
| YES | NO | 5-10 | Witten Bell | 0.7996 |

adds some information w.r.t. the model applied on raw data, but it doesn't improve it much: this is most likely because the concepts become more specific and so POS tags increase sparseness, lowering the information the model can learn. It still gives overall good results when combined with lemma, even if worse than having lemma alone. The last model in the table is also the best performing model.

For what concerns the **n-gram order** of the language model, Table VIII shows the best result for each order.

Table VIII
ADVANCED MODEL N-GRAM ORDER

| Lemmas | POS tags | Order | Smoothing | F1 Score |
|---|---|---|---|---|
| NO | NO | 1 | Katz | 0.5750 |
| YES | YES | 2 | Katz | 0.7942 |
| YES | NO | 3 | Kneser Ney | 0.7949 |
| YES | NO | 4 | Witten Bell | 0.7984 |
| YES | NO | 5-10 | Witten Bell | 0.7996 |

As one could imagine, order 1 is the worse performing, with any combination of parameters, because it doesn't model any dependence on previous tokens. Instead, even by just using 2 there is a big improvement, which is very close as a score to the best result, order 5. It is interesting to see that after order 5 the results become almost constant, not just with the Witten Bell method, but with all of them: this happened on a smaller scale with the base method as well. It is probably due to the fact that the sentences have a pretty standard form and the model is able to learn the different forms. At the end it can be said then that the order does not play a big role in improving a lot the model, given that it is at least 2.

The **smoothing method** has a higher impact on the performance, as shown in Table IX.

Table IX
ADVANCED MODEL SMOOTHING METHOD

| Lemmas | POS tags | Order | Smoothing | F1 Score |
|---|---|---|---|---|
| YES | NO | 5-10 | Presmoothed | 0.7854 |
| YES | NO | 5 | Absolute | 0.7928 |
| YES | NO | 4-10 | Katz | 0.7966 |
| YES | NO | 5 | Kneser Ney | 0.7980 |
| YES | NO | 5-10 | Witten Bell | 0.7996 |

As in the base model there is one method that performs much worse than the others, but vice-versa the others change more significantly the result. It is worth noting that all the best results have lemma and no POS tag.

Lastly, frequency cut-off has been tried on the best model to see if there is any improvement. Table X shows the results.

As in the previous model there is degrading of performance with cutoff (even if more slowly). This again might be because there is little data and cutting off what there is weakens the model a lot.

Table X
ADVANCED MODEL CUT-OFF

| Cut-Off | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| None | 0.9399 | 0.7893 | 0.8103 | 0.7996 |
| 1 | 0.9333 | 0.7817 | 0.7910 | 0.7863 |
| 2 | 0.9276 | 0.7557 | 0.7626 | 0.7591 |
| 3 | 0.9210 | 0.7430 | 0.7342 | 0.7386 |

Overall the results with different combination of parameters and features are not very different: the most important are the features (mostly using lemmas) and the smoothing method. The best resulting model has: **lemma, no POS tag, order 5 (to 10)** and **smoothing method Witten Bell**.

## V. CONCLUSION

The task of building a model capable of sequence labelling has been completed. Furthermore the advanced model improved significantly the base model, as showed in Table XI. It also outperforms the base model even without optimizing parameters and features, being, then, an overall better learning structure.

Table XI
MODELS RESULTS

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Base | 0.9269 | 0.7851 | 0.7434 | 0.7637 |
| Advanced | 0.9399 | 0.7893 | 0.8103 | 0.7996 |

Other interesting findings are that the use of certain features (the use of lemmas without POS tags) usually improves the model regardless of the order and smoothing method. This also happens vice-versa: keeping the features the same, the use of a precise smoothing method (i.e. Witten Bell or Kneser Ney) usually gives better results than using the others.

## REFERENCES


[1] "Github repository of the project," https://github.com/Svidon/LUS_Project1, Accessed: 2020-08-20.

[2] "Nl2sparql4nlu dataset," https://github.com/esrel/NL2SparQL4NLU, Accessed: 2020-08-20.

[3] "How many words make a sentence?" https://techcomm.nz/Story?Action=View&Story_id=106, Accessed: 2020-08-20.

[4] R. G. Raymond C., "Generative and discriminative algorithms for spoken language understanding." Antwerp: INTERSPEECH, 2007.