NLU project - sequence labeling

Paganin Martina
Università degli studi di Trento
martina.paganin@studenti.unitn.it

Abstract

This NLU projects aims to create a Spoken Language Understanding module for the movie domain, based on a NL-SPARQL dataset. The module is built by using functions provided by two open-source libraries (such as OpenFST and OpenGRM), Python and bash scripts.

Basically, the module has to assign concept tags (in the IOB-tagging format) to different parts of sentences by taking into account the experience learned on a training set. In order to accomplish this task, two different approaches were taken. The first is a basic one, while the second is an improved version of the previous one. Performances, which are measured on a test set by computing the F1-score index, rise from 76% points for the first approach to approximately 82% for the second one.

The project is based on the knowledge provided during the Language Understanding Systems course hold by professors G. Riccardi and E. Stepanov (spring 2018)[5].

1 Introduction

This paper describes how to use the NLU module for the movie domain and what are the results obtained on the test set. Initially, it will be give an introduction to the language understanding problem, followed by an explanations about the language model. Then, it will be explained what final-state automata are, why they are helpful for this scope, and lastly, what is the IOB-tagging format used in this project.

In the second section, the instruments used to build the module will be listed. Then, it will be explained how the module works, in order to tag unseen sentences (coming from a test set) and get the evaluation results. Finally, scores coming from the evaluation phase will be showed and compared in order to make some considerations on the parameters used in the building of the language model and get a final discussion.

1.1 The concept tagging phase

An important step of the language understanding problem is the phase during which sentences have to be interpreted. This means that words have to assume a specific meaning in the context of that particular sentence. In order to accomplish this task, tokens (words) in a sentence have to be mapped to a more abstract representation, or concept, which depends on the structure of the language. So, the most probable sequence of abstract concepts (more formally speaking, labels) has to be found for each sentence.

Learning a language model from data come to us useful, as it allows to model the probability distribution over sequences of words. This can be easily done by the instruments provided by the OpenGRM library[4] which permits to create a language model by using few command-line instructions and setting different parameters, such as the ngram-order (the size of sub-sequences) and the smoothing method useful to solve the zero count problem. Language model are encoded in structures called weighted final-state transducers (WFST), which are particular final-state automata.

1.2 Weighted final-state transducers

Final-state automata[1] are basically graphs, with states and transitions, and they are extremely useful in language understanding tasks, as they model the relations between set of words, or more precisely, languages.

Transitions in weighted final-state transducers have this kind of structure: input token, output token, weight. Transducers are a particular case of

final-state automata, and they help to translate an input token into its respective output token. Acceptors are a different type of final-state automata which are mostly used to check if a word belongs to a language. In this case transitions have equal input and output tokens. In order to understand weather a token belongs to a language or not, it needs to be accepted by an acceptor (an automaton which encodes the relations between the words in that language). If a path from an initial state to an accepting one exists, the token is in the language, otherwise it is not.

Hence, to map words into concepts, WFSTs are exactly what it needs to accomplish such a task, as they encode the mapping (word-concept) in their transitions. A sequence of tokens may have more than one transduction, but the most probable one has to be chosen. This is possible because each transition is associated to a weight, which depends on a probability value. By taking the path with the lower cost during the sequence labeling phase, the most probable sequence of tags is selected. In such a way, sentences are tagged word-by-word and it is possible to assign to each of them a concept.

1.3 The IOB tagging format

In this particular task, it is required to identify sequences of words coming from the movie domain, representing concepts such as movie names, characters, producers, directors and many others. There are many different types of tagging formats: examples of these are the POS-tagging and the IOB-tagging formats. In this project the concept tag phase is based on the second one.

The IOB tagging format[2] is used to tag tokens in chunking tasks, which requires to split a sentence in parts and assign to each of them a specific meaning. The B- and the I- prefixes denote respectively the beginning and the continuation of a chunk. The O label is assigned to all the tokens which are not part of any chunk. So, this tagging method distinguishes tokens which are in a chunk from tokens which are outside a chunk.

2 Instruments

This module is mainly composed by scripts written in Python language and bash commands.

Moreover, two open-source libraries (OpenFST[3] and OpenGRM) are also used, in order to accomplish all those frequently NL processing steps. These two library are extremely useful because they simplify the creation of finite-state automata necessary to the language recognition task and to encode the language model.

Generally, bash scripts are used to call the main functions, while Python scripts are used to parse and prepare text files in such a way they can be given as input to the necessary functions. OpenFST is useful as it allows to build finite-state acceptor and transducer in an easy way, while OpenGRM is used in combination with the previous library to encode a language model into a weighted finite-state transducer.

3 Data analysis

The given dataset is split in 2 parts: training and test set. The training set is composed by 3338 sentence, while the test set has size almost equal to 1/3 of the training set: it contains 1084 sentences (totally 7117 tokens). The dataset comes together with an additional text file containing additional features such as Part-Of-Speech tags and lemmas for each word.

The set of sentences (all in the english language) is given in this form: tokens from each sentence are tagged with a label in the IOB format. Both the train and the test set are structured on two columns, the first one is the list of tokens, while the second one contains their related IOB tag. In the training set there are 41 different tags, all related to the movie domain. As can be seen in table 1, tags are very sparse: many show very low percentage values, while only few reach a considerable percentage.

Table 1 reports the distribution of tags in the training set, in percentages. Words labeled with the O tag are a consistent proportion of the whole dataset: there are 15391 O tags, so 70% of the dataset is composed of words which are not part of any chunk. As can be seen, movie name is the second most frequent tag, reaching almost the 15%, followed by director and actorn names. The other types of tags count instead a small number of occurrences.

4 Methodology

The program consists of two main steps: the training phase and the test phase, at the end of which, the evaluation phase is performed.

During the training phase, it is necessary to build an algorithm which learns to tag unseen se-

IOB tag	percentage
O	71.74%
movie.name	14.72%
director.name	2.12%
actor.name	2.03%
producer.name	1.57%
person.name	1.30%
movie.subject	1.15%
rating.name	1.12%
country.name	0.98%
movie.language	0.95%
movie.release-date	0.93%
character.name	0.45%
movie.genre	0.45%
movie.gross-revenue	0.16%
movie.description	0.0093%
person.nationality	0.0093%
director.nationality	0.0093%
movie.location	0.09%
award.category	0.0047%
award.ceremony	0.06%
movie.star-rating	0.0047%
movie.release-region	0.04%
actor.nationality	0.02%
actor.type	0.014%

Table 1: IOB tags distribution

quences of words, so two automaton are created and combined together.

The first one is a transducer which has to map each word into an IOB-tag, by choosing the most probable one. The cost on the edges of this WFST is computed in the calc_probab_edges.py Python script as the negative log of $P(token_i|tag_i)$, where $P(token_i|tag_i) = Counts(tag_i, token_i)/Counts(tag_i)$. It has also to be added edges for unknown words, with $(P = 1/total\ \#\ of\ \ tags)$.

The second automata represents the language model for the tags, so it encodes the probability of a concept tag given the previous n tags.

By composing these automaton, the two above probability assumptions can be linked together in order to infer the most probable sequence of tags taking into account both the probability of seeing a specific tag given the current token, and the probability of seeing a tag given the previous n tags.

4.1 First Method

To run the nlu.sh bash script, three parameters have to be given as input:

- a number in the range 1-5, to specify the ngram size used during the creation of the language model
- a string (one among witten_bell, kneser_ney, unsmoothed, presmoothed, katz)
- a name of file, with .txt extension, in order to save evaluation results on a text file

This script has to call the train.sh script, which takes as input the training set (on a text file), another text file, and the two previous parameters, the ngram order and the method used for the smoothing phase during the construction of the language model. The second text files is necessary to the construction of the language model, as it contains tagged sentences in the IOB format.

The language model is built in the train.sh script, and it has to construct an automaton for each sentence taking into account the relation of each tag inside the sentence, in order to compute the probability of each tag given the previous one.

After the training phase, the nlu.sh calls the test_results.sh giving the test file as input (where tokens are organized in sentences) and the final-state transducer. In the test_result.sh script the two automata are combined together, than, the test file is read, line-by-line. For each line an acceptor is built, in order to make the composition with the transducer built on the training set and get a final acceptor which encodes the transduction in tags for the test sentence. In order to obtain a suitable acceptor, some actions are performed on that. Epsilon transitions are removed and the shortest path is selected on the transducer.

Results are saved on text files and these are then parsed to get a final text file which can be evaluated with the perl script conlleval.pl. Finally, the output of this final script is written on the text file given as input to the nlu.sh script, to make easier the consultation of the evaluation. This first approach, allows to reach an F-score ranging from 74% to 76%.

4.2 Improvements

The second approach works basically like the previous one, so it takes as input the same three parameters as before. The main difference consists

in the training set text files. Before launching the module, the train.data file is parsed in the following way: 'O' tags are substituted with the word itself (e.g. the pair (who, O) is turned into (who, who)).

The same edit is applied to the second text file, which is necessary to the creation of the language model, during the training phase. With this simple modification performances of the algorithm can be improved. This shrewdness allows the model to better classify chunks with the correct tag, because they are no more preceded by generic O-tags. For example, before this modification, sentences were of this type:

who / O
plays / O
luke / B-character.name
in / O
star / B-movie.name
wars / I-movie.name
new / I-movie.name
hope / I-movie.name

As can be noticed, both the tokens which identifies the character and the sub-sequence which denotes the movie title, are preceded by an O tag. After the modification, O tag are removed and substituted with the token itself, so the sentence has this form:

who / who
plays / plays
luke / B-character.name
in / on
star / B-movie.name
warsI-movie.name
new / I-movie.name
hope / I-movie.name

Hence, the character and movie chunks are respectively preceded by different prepositions and verbs, such as "who plays", "on", "the", "movie". A sequence of tags is built by taking those which gives the higher probability, according to the formula

$$t1, .., tn = \arg\max_{t1,..,tn} \prod_{i=1}^{n} P(token_i|tag_i)P(tag_i|tag_{i-1}).$$

As can be seen, the sequence is determined by considering both the probability of seeing the current word given its tag, and the probability of seeing the current tag given the previous one. After the modification, this second probability changes substantially, as many informative tags are no more preceded by the always same O tag. Hence, if a tagged chunk starts after one or more O tags, it can be confused easier with an incorrect tag (e.g. the character may be classified as a movie title). On the other hand, if a specific sequence of tokens comes before a chunk, the probability of being correctly tagged rises.

5 Results and discussion

In the next page, table 2 shows results obtained with the basic approach, while the table 3 lists the results obtained by the improved algorithm. As can be seen, the first one stays in a range of 74-76% of F-score, while the second one reaches values which stay in a higher range, showing a peak of 81.81%. F-score increases of about 0.5% points from the first approach to the second one, proving that the majority of O tags prevents the language model of discriminating more chunks. By substituting the O tags with the respective words, the language model becomes more discriminative and this helps reaching better results.

In the first table it can be seen that the smoothing methods which give the best performances are absolute, unsmoothed and witten_bell. Among these, absolute is the best one. The poorest one is clearly the presmoothed, which doesnt reach a 70% of accuracy. On the other hand, the second table shows that the best performance is given by setting the parameters for the language model as ngram order equal to 4, with kneser_ney as smoothing method. In both the table it can be seen that values for the smoothing method rise with the rising of the ngram order parameter, until to 4. With ngram order equal to 5, Fscore values decrease again, suggesting that size of ngram with n=5 is not a good choice, maybe due to the fact that the majority of chunks are not longer than 3-4 tokens.

Finally, some considerations about the most common mistakes found after the evaluation phase: by analyzing some samples of results, it emerges that the country name of a movie is often incorrectly classified as the language. Words like "spanish" and "english" are clearly name of idioms, but in the context of a sentence as "display a list of spanish films", they are instead referred to movie origins. In some other cases, idiom adjectives are instead thought as the language of the movie, not its origin.

ngram	absolute	kneser_ney	katz	presmoothed	unsmoothed	witten_bell
2	76.31	76.27	75.84	76.21	76.15	76.31
3	75.62	75.67	74.05	67.95	75.46	75.52
4	76.04	76.04	73.12	67.01	75.85	76.07
5	76.04	76.01	63.19	66.75	75.90	76.17

Table 2: Fscore(%) values with the first approach

ngram	absolute	kneser_ney	katz	presmoothed	unsmoothed	witten_bell
2	78.93	78.59	77.99	77.58	76.19	78.46
3	81.37	81.12	79.95	79.54	78.44	80.43
4	81.57	81.81	81.17	79.86	78.28	80.48
5	81.34	81.50	80.23	79.47	78.25	80.36

Table 3: Fscore(%) values after the improvement

Another common mistake is given by confusing people roles, such as tagging actors as director, or producers. In sentences where verbs like to direct or to produce come before people names, these are more correctly classified as director names, or producer names.

References

[1] Final-state automata. http://https://en.wikipedia.org/wiki/Finite-state_machine. Accessed: 2018-04-22.

[2] IOB tagging. https://en.wikipedia.org/wiki/Inside-outside-beginning_(tagging). Accessed: 2018-04-22.

[3] Open FST library. http://www.openfst.org/twiki/bin/view/FST/WebHome. Accessed: 2018-04-22.

[4] OpenGRM library. http://www.opengrm.org/twiki/bin/view/GRM/NGramLibrary. Accessed: 2018-04-22.

[5] E. Stepanov G. Riccardi. LUS course, spring 2018, unitn.