Language Understanding Systems

Abstract

This paper provides the description of the task performed in order to fulfill the first assignment of the Language Understanding Systems course. The project consisted in building a prediction tool meant to associate to each word of a set of sentences its **concept tag**, that is, a tag that indicates what kind of concept is expressed by the word. The goal was achieved by building a **finite state automata** on a training set that was provided at the beginning of the project.

1 Introduction

The input of this task consisted in a **training set**, formatted as a tab-separated values document, containing a group sentences of tagged words. The sentences belong to the *movie domain* of the NL-SPARQL Data Set. Each word of such sentences is associated in the training set with a tag that indicates the **conceptual category** the word belongs to. In the set, the following categories are present:

Concept tag	Meaning
character.name	name of a character
movie.name	name of a movie
actor.name	name of an actor
person.name	name of a person
director.name	name of a director
movie.release_date	release date of a movie
movie.location	location of a movie
producer.name	name of a producer
country.name	name of a country
movie.genre	genre of a movie
movie.language	language of a movie
rating.name	rating of a movie
movie.subject	subject of a movie
actor.type	type of actor
actor.nationality	nationality of an actor
director.nationality	nationality of a director
movie.gross_revenue	gross revenue of a movie
person.nationality	nationality of a person
award.ceremony	award ceremony
movie.release_region	release region of a movie
movie.description	description of a movie
movie.star_rating	star rating of a movie
award.category	award category of a movie
O	other

Except for the `O` category, the training set provided a **prefix** for each tag, either `B-`, `I-` or `E-`, indicating whether the word is at the beginning, in the middle or at the end of the concept, for those cases in which the concept is itself expressed by a periphrasis.

A **test set** was also provided, for purposes of evaluation. The test set obviously matches the format of the training set. The purpose of the project consisted of the following goals:

- providing a generic **analysis** of size and distribution of the data

- **training** a finite state automaton and a language model
- **tuning** the parameters of smoothing and n-gram size
- handling **unknown** words
- optimizing the **evaluation** score

2 Data Analysis

The training set consists of `3338` sentences, for a total of `21453` words. The test set, instead, consisted of `1084` sentences, for a total of `7117` words. In the two sets, respectively, the concepts present the frequency described in the following two graphs.

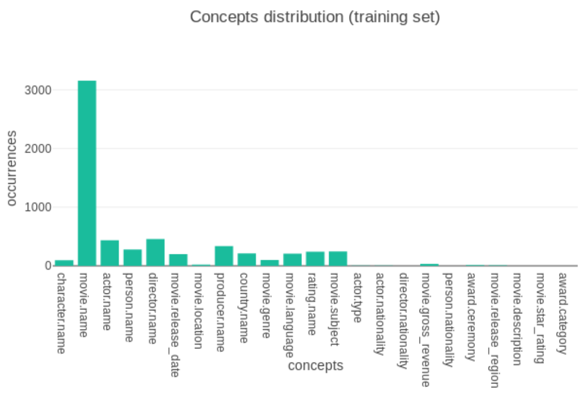

Figure 1: Training set distribution

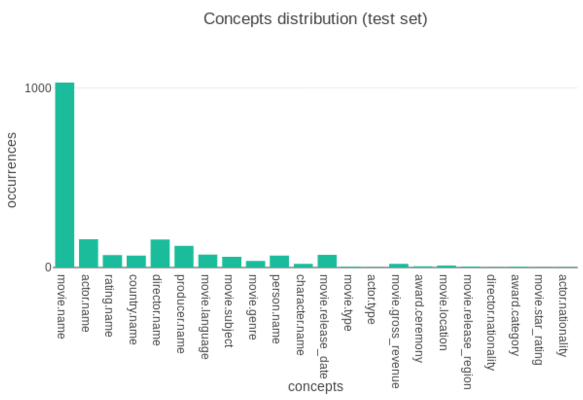

Figure 2: Test set distribution

The frequency of the `O` tag has been omitted in the graphs, since it disruptively outnumbers the occurrences of the other tags: the training set counts `15391` instances, while the test counts `5135`.

As for the **unknown words**, that is, the words which are in test set but not in the training set, they represent about `23.7%` of the whole test set.

3 Problem statement

In this section, each problem present in the task will be formally described, providing an outline of what goals need to be achieved in each phase.

3.1 Building the finite state automaton

In order to model the language, we need to build an automaton that takes into account the probability of a certain word being associated with a certain tag, for **unknown** words too, as well as the probability of a certain **sequence** of tags.

3.2 Building a lexicon

In order to handle all the words in the vocabulary, we need to build a **lexicon** containing each one of them.

3.3 Building language model

In order to take into account the relations between subsequent words, we need to build a language model that codifies all the **sequences** of tags that are present in the training set, creating a transducer.

3.4 Encoding each sentence

We need to express each sentence encountered in the test set as an **automaton** itself, since it needs to be composed with the other automaton, which codifies the probabilities of each couple `(word, tag)`.

3.5 Applying the model to the test set

Once the model is ready, we can apply it to each codified sentence of the test set, in order to predict the tag for each word. The idea is that this solution should maximize the probability of the tag to be associated with that word, as well as the probability of the resulting sequence of tags.

3.6 Maximize the score

By tuning the **parameters** and applying **improvements** to our data, we need to make the prediction as good as possible, with particular focus on maximizing the F1-score on the test set.

4 Solution

In this section, we will be going through every step of the solution that has been found in order to achieve the goals listed in the previous section. In order to solve the whole task, I have been using `python 3` as a **programming language**, and several **bash commands** provided by the `opengrm` [2] and `openfst` [1] libraries.

4.1 Building the finite state automaton

Our automaton should encode the probabilities of each word being associated with a certain concept tag. In order to achieve this, I simply computed each value as follows:

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

Basically, the probability is computed as the count of occurrences of the association between word and tag, over the total count of occurrences of the tag only. Plus, we should also take care of all the words that are encountered in the test set, but are not present in the training set. I have chosen to associate to each of these word a probability of $1/n$, where n is the number of concepts.

Once the probabilities were computed, I simply built the transducer as an automaton with a single state, and a transition for each probability, as follows:

in_state	fin_state	i_word	o_word	weight
0	0	word	tag	prob
0	0	<unk>	tag	prob
...	...	...	...	...

4.2 Building a lexicon

In order to build a lexicon, I simply used the `opengrm` library:

```
ngramsymbols < training_set.data
> lexicon.lex
```

4.3 Building a language model

The first step for the generation of a language model consisted of extracting from the training set the **sequences** of tags, one for each sentence, and this has been easily achieved with python. Such sequences have then been encoded with the following command:

```
farcompilestrings
--symbols=lexicon.lex
--unknown_symbol='<unk>'
concepts_sequences.txt >
encoded_concepts_sequences.far"
```

where `concepts_sequences.txt` is simply the file where the raw sequences were collected. Once having done this, I have used the following command in order to generate the count of *n*-grams (with arbitrary value for *n*) in the training set:

```
ngramcount --order=ngram_size
--require_symbols=false
encoded_concepts_sequences.far
> encoded_concepts_sequences.cnt
```

Finally, by the following code I have generated the language model:

```
ngrammake --method=mymethod
encoded_concepts_sequences.cnt
> encoded_concepts_sequences.lm
```

The idea consists of encoding the count of *n*-grams as an automaton, and then making a model out of it. We will see soon how it has been used.

4.4 Encoding of a sentence

In order to encode each sentence of the test set as an automaton, I have simply been applying the following code:

```
echo "my sentence"
| farcompilestrings
--symbols=lexicon.lex
--unknown_symbol='<unk>'
--generate_keys=1 --keep_symbols
| farextract
--filename_suffix='.fst'
```

The code, as we can see, takes care of **unknown words**, and keeps track of the original symbols in the lexicon.

4.5 Applying the model to the test set

We have now all the elements we need in order to perform the **prediction**. For each sentence in the test set, I have applied the following code:

```
fstcompose compiled_string.fst
transducer.fst
| fstcompose -
encoded_concepts_sequences.lm
| fstrmepsilon
| fstshortestpath
| fsttopsort
> result.fst
```

Basically, the idea consists of composing the encoded string with the transducer described in section 4.1, which gives as result an automaton which essentially takes into account every possible tag for each word, with its probability. Then, such result is composed with the language model described in section 4.3, giving as result an automaton which also takes into account the probability of the ***n*-grams**, that is, the probability of a certain sequence of concepts.

The result of this operation consists in the sequence of final predictions of the concept for each word of our sentence, based on maximizing its probability in the training set (which corresponds to finding the lightest path in terms of the inverse log of the probability).

5 Experiments: maximizing the score

In order to define a **baseline** to start from, I have computed the scores with three different trivial approaches:

- **random**: assign a random value as prediction for each word → *F1*-score ∼ `0.6%`
- **chance**: assign a random value as prediction for each word, according to distribution of concepts → *F1*-score ∼ `1.5%`
- **majority**: assign to each item the most represented prediction in the training set → *F1*-score = `0%`, accuracy = `72.15%`

The very first implementation of the code has been applied with the following parameters:

- *n*-gram size: `3`
- smoothing method: Witten-Bell

The Witten-Bell smoothing exploits the method of modeling probabilities of seeing events (in our case, *n*-grams) for the first time. This parameters led to an *F1*-score equal to `75.58`. From this point, I have been performing a set of **experiments** with the purpose of maximizing the score.

The first significant improvement has been achieved by handling the **unbalance** derived from the vast majority of `O`s in the training set. This unbalance, indeed, increases the probability of errors. In order to tackle this problem, I replaced each `O` in the training set with a tag based on the word itself:

`who O` → `who _who_`

This way, we essentially **distribute** what was once the probability of being tagged with an `O` over all the possible values which fall into the "other" category of the table in the introduction. For the evaluation of the result, each one of those predictions is converted back into an `O`.

This improvement alone increased the *F1*-score to `81.33`. After having applied this, I have started iterating over some combinations of the *n*-gram size and smoothing method parameters:

***n*-gram size**	**method**	***F1*-score**
2	witten-bell	79.31
2	katz	78.84
2	absolute	78.93
2	kneser ney	79.45
3	witten-bell	81.33
3	katz	80.85
3	absolute	81.37
3	kneser ney	82.04
4	witten-bell	81.42
4	katz	81.17
4	absolute	81.57
4	kneser ney	82.74
5	witten-bell	81.26
5	katz	81.08
5	absolute	81.34
5	kneser ney	82.43

6 Discussion and conclusions

The Kneser-Ney smoothing method, which operates by subtracting a fixed value from the probability's lower order terms to **omit** *n*-grams with lower frequencies, and exploits the **history** of *n*-grams within documents, turned out to outperform the alternative methods that have been tested.

On the other hand, the *n*-gram size parameter hasn't seemed to be significantly determinant within the range of values that has been analyzed.

As for the **errors** in the prediction, most of them turned out to be either related to a person's name, or to the position within a periphrasis:

real concept	**predicted concept**	**error count**
B-movie.name	I-movie.name	19
B-movie.language	B-country.name	13
B-producer.name	B-director.name	12
I-producer.name	I-director.name	9
B-actor.name	B-person.name	7
I-actor.name	I-person.name	7

Comprehensibly, the classifier occasionally failed in determining whether a concept is at the beginning of a sentence or in the middle of it, although the concept itself was often correctly-predicted.

Also, when it comes to **names** of people, it is obviously hard for the classifier to determine whether it's an actor, a director, a producer or a generic person: for the unseen words, the prediction is likely to fail. Finally, a good number of errors involved **languages** and **country names**.

Finally, the experiments performed generally outperform the **baseline** approaches described at the beginning of section 5. Although the majority approach provides a fair level of **accuracy**, the tested methods significantly improve also the more relevant metric of ***F1*-score**.

References

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

[2] Opengrm. http://opengrm.org/.