Université Libre de Bruxelles

INFO-Y404

Natural Language Processing

Intelligent Word Order Handling

Daoud Yemni

16 juin 2018

Table des matières

1 Introduction

Natural language production aims to formulate semantic representations into concrete sentences. The order of words of the produced sentence is very important. The word order can significantly change the meaning of a sentence. For the project, we need to provide a rendering program that takes as input a set of semantic representations and produce a correct sentence. The project is dividing into 2 part. The first part consists of reading the input file and construct the constraint problem by translating the semantic representations into constraints and variables. The second part consists of building a language model and compute the perplexity of a sentence. By using both part, the program is able to provide a good and correct sentence corresponding to the semantic representations.

2 Constraint Programming

The constraint programming is used in the project to solve the problem. The library python-constraint was added in the project to create, construct and solve a constraint problem. Each semantic representation, given in the input file, has been translate into a constraint or variable. When all variables and constraints are implemented. The program will find none or one or multiple solutions. If there is one solution, the solution will be simply formatted and printed but if there is multiple, the program will compute the perplexity of each one and print the sentence with the best perplexity.

2.1 Variables

In the program, Each variables of the problem are represented by their name, for example "var-1", and a integer value. The value of the variable represents the position of the word in the sentence. The variable is created with the semantic representation : (string, var-1, value-1). For each string-semantic representation, the program create and add a variable in the problem.

$$(string, a, value) \rightarrow a \in [0, ..., (N-1)]$$

where N is the number of words/variables.

2.2 Constraints

In the project, you can define your own constraints corresponding to a semantic representation in the JSON file. There is some default definition of semantic representation and constraints. For each semantic representation, I will define the description of the representation, his representation and his constraint programming version :

2.2.1 (meets, var-1, var-2)

The grammar constraint **meets** describes that the first variable need to be before the second variable. In this case, the first word needs not to be necessarily directly behind the second word.

$$(meets, a, b) \rightarrow a < b$$

2.2.2 (precedes, var-1, var-2)

The grammar constraint **precedes** describes that the first variable need to be directly before the second word. Example : (precedes, the-1, game-1) means that in the sentence, the word corresponding to the "the-1" is next to the word corresponding to the "game-1" : "(...) the game (...)".

$$(precedes, a, b) \rightarrow a == (b - 1)$$

2.2.3 (first, var-1)

The constraint **first** describes that the variable need to be the first word in the output sentence.

$$(first, a) \rightarrow a == 0$$

2.2.4 (last, var-1)

The constraint **last** describes that the variable need to be the last word in the output sentence.

$$(last, a) \rightarrow a == (N - 1)$$

where N is the number of words/variable.

2.2.5 (follows, var-1, var-2)

The constraint **follows** describes that the first word need to be after (not necessarily direct) the second word.

$$(follows, a, b) \rightarrow a > b$$

2.2.6 (next, var-1, var-2)

The constraint **follows** describes that the first word need to be **directly** after the second word.

$$(next, a, b) \rightarrow a == (b + 1)$$

2.2.7 (between, var-1, var-2, var-3)

The constraint **follows** describes that the second word need to be between the first word and third word.

$$(between, a, b) \rightarrow a < b \wedge b < c$$

2.2.8 (chain, var-1, var-2, var-3, var-4)

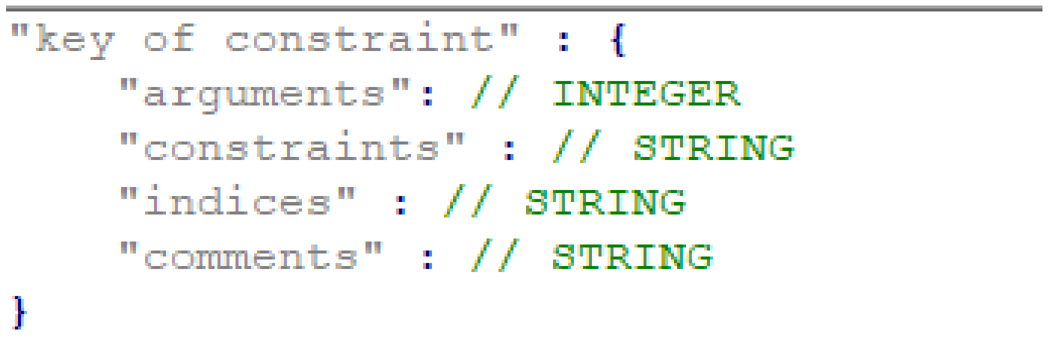
The constraint **follows** describes a chain of words in defined order. You can change the order.

$$(chain, a, b, c, d) \rightarrow (a < b) \wedge (b < c) \wedge (c < d)$$

There is a last constraint not linked with the semantic representation. This constraint defines all variables need to be different of each one. Each pair of variables are different.

2.3 Adding or updating constraints

You are free to add, delete or modify the constraints of the program. You're not limited on the number of arguments but you need to respect this format :



```
"key of constraint" : {
    "arguments": // INTEGER
    "constraints" : // STRING
    "indices" : // STRING
    "comments" : // STRING
}
```

FIGURE 1 – Format JSON of a constraint

The definition of each element is defined as following :

— **arguments** : Number of arguments used for the constraint

— **constraints** : Represents an array of mathematical formulation of your constraints with the respective variables (x1, x2, ..., xn) where n is the number of arguments. For example, ['x1 < x2', 'x2 < x3'] for a constraint with 3 variables. The maximum of variables to use in each element of the array is 2.

— **indices** (optional) : If you have only 1 argument, you cannot omit this section. It corresponds of an 2-D array in which each subarray contains the indices of variables used in each constraint defined in **constraints** section. For example, if you have in **constraints** is "['x1 < x2', 'x2 < x3']", you will have in **indices** : "[[1,2], [2,3]]".

— **comment** (optional) : Express the behavior of the constraint in a human language.

3 Language Model

In this section, we will explain the language model used for the perplexity. In the project, the teachers provide a website[1] where we can directly find a n-gram of words with frequencies. With that, the language model can be easily build. I choose the Bi-gram with non case-sensitive. The file is represented as follow :

1. https://www.ngrams.info/download_coca.asp

frequency   first-word   second-word

We read the file, store the frequency and the corresponding Bi-gram in the *count* matrix.

$$count(w1|w2) = freq$$

We use the 1-add smoothing to escape of the zero probabilities, normalize all of bi-grams and store the probability in the probability matrix.

$$P(w1|w2) = \frac{count(w1|w2) + 1}{count(w2) + V}$$

where V is the size of the vocabulary.

Then we compute the perplexity of a sentence as follow :

$$PP(W) = \sqrt[n]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

where W is the sentence and N is the number of word in the sentence W.

The perplexity is the inverse probability and normalized by the number of words. So the lowest perplexity of a sentence is, the most correct and best is.