Natural Language Generation

Summer Internship



Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Authors** | **Comments** | **Authorization** |
| 1.0 | 26-08-2020 | Pedro Esteves | Summer Internship | - |
| 2.0 | 28-08-2019 | João Almeida  Inês Alves | Summer Internship | Pedro Reganha |

Index

[Version History 2](#_Toc49514046)

[1 CONTEXT AND OBJECTIVE 4](#_Toc49514047)

[1.1 Objective 4](#_Toc49514048)

[1.2 Context 4](#_Toc49514049)

[1.3 Installation 4](#_Toc49514050)

[2 Features 2019 (NER) 6](#_Toc49514051)

[2.1 Named Entity Recognition 6](#_Toc49514052)

[2.2 Natural Language Generation (Columns roles) 6](#_Toc49514053)

[2.3 Natural Language Generation (Named Entity Recognition Algorithm) 7](#_Toc49514054)

[2.4 Natural Language Generation (Custom NER classes) 7](#_Toc49514055)

[3 Features 2020 (Report) 11](#_Toc49514056)

[3.1 Report structure 11](#_Toc49514057)

[3.2 Python Pipeline 11](#_Toc49514058)

[3.3 New features 12](#_Toc49514059)

# CONTEXT AND OBJECTIVE

## Objective

The Syone’s NLG program consists in generating a Report based on an input table. The improvements built, made it capable of generating richer text instead of repetitive analysis (state of previous work).

## Context

The project was boosted up during the summer internship in 2020. The main focus was building up a nice clean Report using all the mathematical operations that had already been developed. Another feature is **NER(Named Entity Recognition)** analysis of the input to improve the richness of the text. Fragment from a generated report:

*“Still analyzing the same* ***Company*** *value but now, about* ***Yield****, the mean is 600.00 and the standard deviation is 100.00, the maximum is 700.0, and belongs to the Person,* ***Eva****, in Local,* ***Faro****, which has Age* ***25****. Besides that, the minimum value is 500.0, and belongs to the Person,* ***Carlota****, in Local,* ***Coimbra****, which has Age* ***23****.”*

## Installation

Install Docker Desktop

Install Docker 19.03.12

Intall Python 3.8.4

Libraries:

elasticsearch 6.3.1

Flask 1.0.2

googletrans 3.0.0

inflect 4.1.0

nltk 3.5

openpyxl 2.5.5

seaborn 0.10.1

sklearn 0.0

spacy 2.3.2

To run the project:

* In the repository checkout to branch final\_dev;
* Start the project writing “docker-compose -f docker-compose.yml up ” in command line (this will start all necessary processes like nlg, frontend, backend, interlayer and elasticsearch);
* A screenshot of a cell phone

  Description automatically generatedOpen <http://localhost:3000/narrative> to generate narrative (you would see the page o generating narratives as you can see in the image below).

As it is not practical to always build the docker to run the project, you can choose to use JSON data and install Postman for example.

Here is an example of how to use Postman:

* <https://www.youtube.com/watch?v=hAEJajltHxc&feature=youtu.be>

# Features 2019 (NER)

## Named Entity Recognition

NER stands for Named-Entity Recognition and is a subtask of information extraction that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, and percentages.

The NER capabilities of the model are present in file NER.py. The DefaultText module will call the NER functionalities through the function detect\_ner\_classes() which will analyze the names of the columns and assign a number to each column representing the probability of the column belonging to the class Entity, class Location or class Time. (These 3 categories were picked since they are very broad and are common to almost every Excel and every Tableau, however they can be adapted, new or more specialized classes can be chosen). If the probability of a column being a NER class is above a certain threshold (a number chosen by the programmer/user) this means the column will be treated as a NER class, It is pictured in the previous excerpt where “Age” was treated as Time NER class and Local was treated as Location NER class. The NER task was accomplished using WordNet path\_similarity() function.

## Natural Language Generation (Columns roles)

During the information retrieval is important to distinguish different types of columns. The first distinction done in this module is separating numeric columns from categoric columns.

After that is done, the categoric columns are analyzed to check if there is a supergroup.

**Supergroup** – The supergroup is the name of the categoric column with the smallest number of different values. Note: Just because some data has a categoric column it does not mean it has a Supergroup, besides that, the number of different values must also be smaller than a given threshold.

**Supergroup value** – Since the Supergroup is the name of the column, the Supergroup values are the values of that column. Once a Supergroup is detected in the dataframe, the dataframe will be splitted in subdataframes where the criteria to split the dataframe is the different values of Supergroup.

**NER Class** – NER classes aren´t columns but they are an important notion. The NER classes chosen for this project were **Entity**, **Time** and **Location**. One can think of NER classes as a set of words that represents a given concept. For example, for the class Entity one could include the set of words “Customer”, “Brand”, “Animal” etc… And for **Location** “Street”, “Country”, “Region” etc…

**NER Column** – NER columns can be categoric or numeric and have smaller or bigger granularity. A column is considered a NER column if its name shares with any word from a NER Class wordset a path\_similarity value above a certain threshold. That threshold determines if the column is considered a NER column or not. NER columns are useful in information extraction because the report in order to be useful must not contain all the information from the data, (otherwise it would be preferable to look at the data instead). NER columns provide a way to associate maximum and minimum values to the entities, locals and time associated with them.

**Numeric Columns** – There can, and should be numeric columns independently if a supergroup is or not detected and if or not NER classes are detected. If a supergroup is detected the numeric columns are analyzed for each supergroup value and their maximums and minimums are associated with NER classes.

**IMPORTANT NOTE:** A supergroup column cannot be a NER column.

## Natural Language Generation (Named Entity Recognition Algorithm)

Once a column begins the process of Named Entity Recognition, the path\_similiarity() between the column name and the words matching the NER class word set will be calculated. In the designed algorithm this value can be calculated in two different ways depending if the column name has just one word or if it has more than one word.

**Case where the column name(p) has just 1 word:**



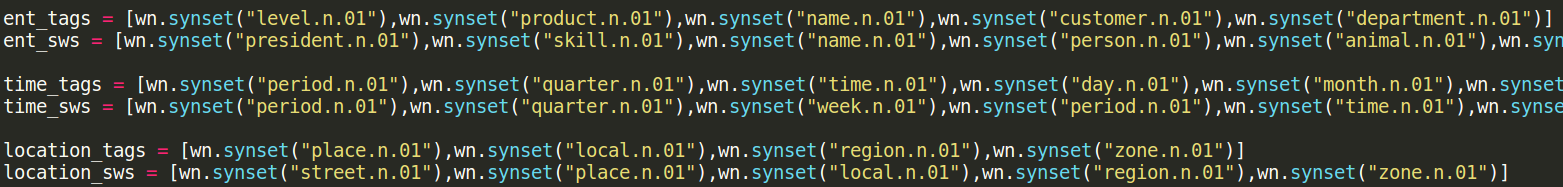
If the column name has just one word then the probability of it being a certain NER class is given by the highest path\_similarity between the column name and a word of the NER class word set.

**Case where the column name has more than 1 word:**

A screenshot of a cell phone

Description automatically generated

For each word of the column name it is calculated the highest path\_similarity between the word in the column name and the words in the NER class word set. After, the highest probabilities for each word are summed and the result is divided by the number of words in the column name. Lastly that value is subtracted by a lambda multiplied by the number of words in the column. (This is done to penalize the probabilities of the columns with a large number of words).



Example of the words matching a given NER class. Here we have the words for the NER classes Entity, Time, and Location. There are 2 arrays for the NER classes, one for the case where the column name is just composed by 1 word and other for the case where the column name is composed by more than 1 word.

## Natural Language Generation (Custom NER classes)

Since the goal of this project is for this model to be flexible enough to produce reports for any excel/tableau of a given domain, one can adapt the model to generate more domain specific reports by following these 3 simple steps:

1.1 – Inside the file DefaulText.py add to the array ner\_classes the name of the new ner\_class you want the application to recognize. **For example “litter”**.

A picture containing object, clock, ball, orange

Description automatically generated

A close up of a clock

Description automatically generated1.2 – It should look like this after adding:

A screenshot of a cell phone

Description automatically generated2.1 – Open NER.py in an IDE and add a new key-pair to the dictionary **ner\_classes\_tags**. (In this case the key must be “**litter**” since the keys in this dictionary have to match the keys in ner\_classes from the previous step).

2.2 – After adding the new ner class “Litter”, ner\_classes\_tags should look something like this. You can choose the words you want to associate to ner\_class as the synset’s present in the array.

A picture containing road, sitting, holding, stacked

Description automatically generated

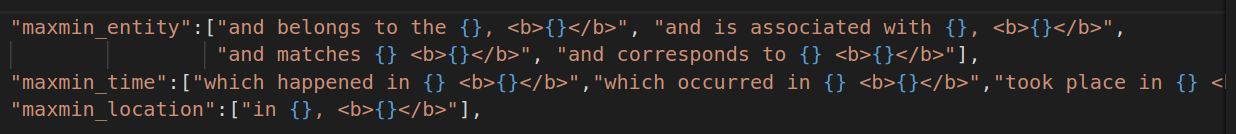
3.1 – After doing the previous steps the new ner\_class will appear in the data format generated by the NLGML module.

A screenshot of a cell phone

Description automatically generated

Excerpt of the output of NLGML.py from mammals.xlsx excel (Here it is observable that for the value “**Water**” of the supergroup “**Domain**” 2 NER classes have been detected. Litter which is associated to column “**Litter Size**” and “**Entity**” which is associated to column **Animal**.

3.2 – To add new templates for the new ner\_classes, simply added them to file Templates.py like this “**maxmin\_<new NER class>”.** Following the example from the previous steps you would add **maxmin\_litter**.



The advantage of having the freedom to pick new NER classes is that you can make the text template more or less generic given the granularity of the NER class.

A close up of a screen

Description automatically generated

Excuse the nonsense of “LIXO LIXO LIXO etc...” It is just for demonstration purposes on the app:

A close up of a newspaper

Description automatically generated

There it is! The new ner\_class Litter has been detected and its info has been presented in the text.

# Features 2020 (Report)

## Report structure

“A **report** is written for a clear purpose and to a particular audience. Specific information and evidence are presented, analysed and applied to a particular problem or issue. The information is presented in a clearly structured format making use of sections and headings so that the information is easy to locate and follow.”

The first measure included creating a Python Class (in ReportTemplate.py) that involves the Report’s structure. It has the following properties:

1. generate\_title
2. generate\_terms\_of\_reference
3. generate\_introduction
4. generate\_super\_group\_body
5. generate\_general\_body
6. generate\_draft
7. generate\_correlation

On top of that, the file also includes auxiliary functions to help writing the syntax of the main body or adding hyperlinks and collapse the text.

## Python Pipeline

F Natural Language Generation can be pictured as a pipeline of different tasks where there is precedence between them. For this model a pipeline was implemented and each code file in the project can been seen as belonging to one of these tasks.

1. Information Extraction/Retrieval

2. Text Structuration

3. Text Generation

**Extraction and Retrieval of Information** consists on the process of extracting from the data relevant information and relations between the present variables. It answers questions such as: What are the variables of interest ? Which variables are numeric and which variables are categoric ? Which variables are NER classes ?

**Text Structuration** is the task responsible for generating the skeleton of the text which will be filled in the Text Generation step with concrete words. During this step the data from the tableaus and excels gets converted to the same format so that during the generation step, the text can be generated independently from the initial data input providing a better encapsulation and less dependencies between the modules.

**Text Generation**. Generates the final report in English from the structured collected from the previous step.

A screenshot of a cell phone

Description automatically generated

Figure 1Association of .py files to one of the three different phases of the NLG pipeline (Information Retrieval, Text Structuration and Text Generation).

1. After the excel/tableau data has been processed by Server.py and Business.py it arrives to DefaultText. Here a number of steps will occur such as translating the column names to english, detecting if there is or not a supergroup, extracting m[aximums and m](https://www.ck12.org/book/CK-12-Precalculus-Concepts/section/1.5/)inimums etc… During this step through the function **detect\_ner\_classes()** the data is sent to the NER module which will perform Named Entity Recognition, possibly assigning the columns to a “Entity”, “Time” or “Location” class.
2. After all values and relations are extracted and stored in a dictionary (called report) the report will be sent to NLGML. This dictionary format is dependent from the data input type which can be excel or tableau. In the NLGML a new dictionary, independent from the data will be generated. This dictionary is fundamentally different from the previous one, namely, it has much more nested keys. Nested keys make text generation easier and are important due to the more information is gathered about a given relation, more complex the sentence will be, and that information is stored in a nested fashion.
3. After the new data dictionary is generated, it is sent to ReportWriter whose responsibility is to generate the Detail Analysis and organize the Narrative structure (class from ReportTemplate). Finally all words, sentences and expressions are in Templates. There are two variables: temps and dictionary which represent the sentences mainly used in Detail Analysis and in Report.

## New features

DefautText.py:

* For each super group’s table, all columns with standard deviation 0 are deleted. – drop\_col\_no\_std()
* Global analysis, in order to get the maximum and minimum value and the column that has the maximum and minimum standard deviation. – advanced\_numeric\_analysis()
* Find if a Super Group has already been chosen (in the data), by having an identifier’s column. – given\_super\_group()
* Improved correlation search. Now it looks for correlation in the column which has the major standard deviation when there is super group. - tableau\_correlation() ; On the other hand it tries to correlate numeric columns when there is no Super Group. – pre\_correlation()
* Creates a better heatmap and saves the path for present it later – calc\_corr()
* Keeps writing more Reports until there is not more super groups. – default\_text\_gen()

MLNLG.py:

* Sorts correlation group by the most influents and organizes by whether has more than one pair or not. – sort\_correlation\_pairs()
* Finds which column as the most and the less standard deviation for each super group value. – std\_mean\_column()
* Computes the standard deviation average for each super group value, and sorts the resut. – std\_variation()
* Standard deviation main function. – std\_analysis()
* Computes the standard deviation’s percentage. – variation\_p()

ReportWrite.py:

* Main function to generate the Report. – write\_final\_text()
* Computes the maximum, minimum and mean for each column and matches with the super\_group value – mean\_max\_min()

ReportTemplate.py:

A screenshot of a cell phone

Description automatically generatedThis file creates features that can integrate any Report. Even though the detail analysis is written in ReportWrite.py, it’s here where all main text is generated. Besides that, by using hyperlinks() or collapse() functions it is possible to introduce simple HTML features that makes the generated text clean, and interactive.

local\_test.py:

* Generate narrative Locally and when WRITE\_FILE is True it creates an HTML with the Report. Truly helpful for DEBUG.

tableu2table.py: (primitive version)

* Rearrange tableau data (json) to create a 2D table (json).

Template.py:

* New phrases are added for the new template.