Technical Report: Automated Obituary Generation

Ashish Upadhyay

April 17, 2020

Aim

This document gives an overview of a **Flask-Service** created to support the **Case-Based** process for the automated obituary generation.

1 Natural Language Generation

Natural Language Generation (NLG) is one of the sub-field of Artificial Intelligence (AI) where we aim to generate a text/document from non-linguistic input. In our case the main challenging task is to develop an automated **data-driven** process for the generation of obituaries. For this we choose to use a Case-Based Reasoning (CBR) approach which applies lazy learning algorithm for the automated NLG. CBR provides a way to dynamically generate different templates from previous data. A series of experiments were performed to choose best possible algorithm for various sub-tasks in the NLG process.

A technical paper on experiments is accepted at the International Conference on Case-Based Reasoning (ICCBR) - 2020. Please refer to that document for NLG and CBR related stuff.

2 REST API

An RESTful API has been developed to support the NLG process so that it can be easily integrated wherever possible. The service is a RESTful API that takes POST request (input features) from a form and pass it to a CBR component which generates four different obituary texts to be send back.

The code-base for the API is uploaded on a private GitHub repository for sharing with the concerned members. The link to the API is:

`https://github.com/panditu2015/Obituary-Generation`

A brief explanation for the code-base is given in the following sections.

2.1 Directory Structure

The directory structure of the code-base is as follows:

```
Obituary-Generation (root directory)
├── app (flask-services)
│   ├── messages (CBR component)
│   │   ├── data (case-base for the CBR component)
│   │   │   ├── basic.csv
│   │   │   ├── funeral_component.csv
│   │   │   ├── personal_component.csv
│   │   │   └── relations_component.csv
│   │   ├── resources (a list of features categorised in different components)
│   │   │   └── feature_list.csv
│   │   ├── __init__.py
│   │   ├── basic.py
│   │   ├── erros.py
│   │   ├── funeral.py
│   │   ├── main.py
│   │   ├── personal_info.py
│   │   ├── relations.py
│   │   └── README.md (instructions specific to CBR component)
│   ├── templates (HTML templates for flask app)
│   │   └── index.html
│   ├── __init__.py
│   └── routes.py (file for all the routes)
├── labelling (framework for data labelling and creation of case-base)
│   ├── data (final tabular representation created)
│   │   ├── basic.csv
│   │   ├── funeral_component.csv
│   │   ├── personal_component.csv
│   │   └── relations_component.csv
│   ├── resources (jsonlines and xml files)
│   │   ├── tagged.json1 (tagged obituaries downloaded from doccano)
│   │   ├── xml-tagged.xml (tagging in xml version)
│   │   └── xml-tagged-comps.xml (same tagging done component wise)
│   ├── create_csv.py
│   ├── create_csv_comps.py
│   ├── create_xml.py
│   └── README.md (instructions specific to data labelling)
└── reports (technical reports)
    ├── images
    │   ├── basic-out.png (example output from basic-retrieval method)
    │   ├── comps-out.png (example output from component-retrieval method)
    │   └── input-form.png (screen capture of input form)
    └── flask-app-report.pdf
```

```
    └── ICCBR-accepted-paper.pdf
├── migrations (database stuff:  can be easily ignored)
├── .gitignore
├── cbrobit.py (main flask-app file)
├── LICENSE (currently licensed as GNU v3.0)
├── README.md (main instructions file)
└── requirements.txt (required libraries to run the app)
```

It is noted that the files which are not mentioned here and still available in the repository might not be useful and thus can be easily ignored.

2.2 Using the App

To run the app, please install the required libraries by using the following command from inside the `Obituary-Generation` directory:

```
$ pip install -r requirements.txt
```

Now to run on:

- Unix Bash (Linux/Mac):

```
$ export FLASK_ENV=``development''
$ export FLASK_APP=``cbrobit.py''
$ flask run
```

- Windows Powershell:

```
> $env:FLASK_ENV=``development''
> $env:FLASK_APP=``cbrobit.py''
> flask run
```

- Windows CMD:

```
> set FLASK_ENV=``development''
> set FLASK_APP=``cbrobit.py''
> flask run
```

After starting the service, go to the following link where a form will appear:

`https://localhost:5000`

A screen capture of the form is shown in fig. 1. After submitting the form by clicking the submit button, a json file will appear on :

`https://localhost:5000/result`

The input json will contain all the input features in `attribute-value` pair and four generated obituaries from those features. In figs. 2 and 3, we can see the possible outputs for both `basic` and `component` retrieval methods respectively.

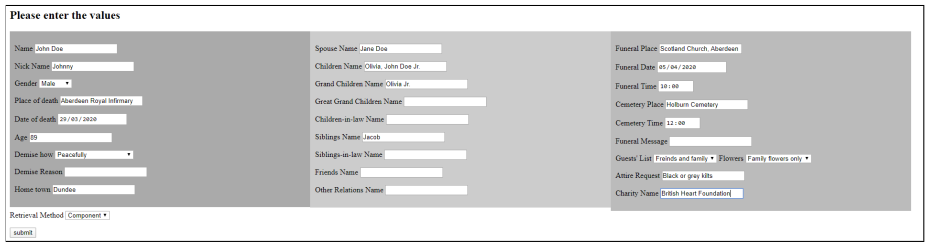

Figure 1: Screen capture of the input form.

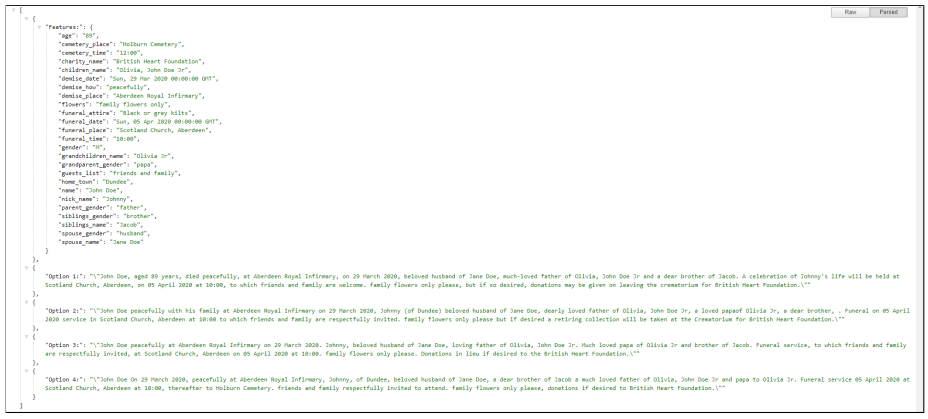

Figure 2: Output in Basic-Retrieval method.

Figure 3: Output in Component-Retrieval method.

2.3 Case-Base

2.3.1 Creation

The case-base is generated using a semi-automated process. First a manual labelling is done for 100 samples to annotate the `attribute-value` pairs using a open-source sequence labelling app named **doccano**[1]. The app generates a `jsonlines` file in the popular CoNLL format that can be found at:

`/Obituary-Generation/labelling/resources/tagged.json1`

The `jsonlines` file contains a `json` object in each line, where each object will have a key `text` with a string non-annotated obituary as value and `labels` with a list of all features in that text followed by there starting and ending index in the obituary string. The `jsonlines` file is then converted into a `xml` file using the following script:

`/Obituary-Generation/labelling/create_xml.py`

which will create the following file:

`/Obituary-Generation/labelling/resources/xml-tagged.xml`

Note this will create the tagged file for `basic-retrieval` method. To create the `component-retrieval` tagged file, a further manual annotation is required to segregate the obituary sentences into different components. The `component` version of above file can be found at:

`/Obituary-Generation/labelling/resources/xml-tagged-comps.xml`

Now that we have the `xml` files, we can run the following code to convert them into a tabular format stored in `csv` files:

For `basic-retrieval` method:

`/Obituary-Generation/labelling/create_csv.py`

this will create the following file:

`/Obituary-Generation/labelling/data/basic.csv`

For `component-retrieval` method:

`/Obituary-Generation/labelling/create_csv_comps.py`

this will create the following files:

`/Obituary-Generation/labelling/data/personal_component.csv`

`/Obituary-Generation/labelling/data/relations_component.csv`

`/Obituary-Generation/labelling/data/funeral_component.csv`

Feature	Component	Frequency
name	Personal Info	125
age	Personal Info	47
demise_place	Personal Info	89
demise_date	Personal Info	101
demise_how	Personal Info	91
demise_reason	Personal Info	11
home_town	Personal Info	44
nick_name	Personal Info	97
work_place	Personal Info	1
parent_gender	Relations	89
spouse_name	Relations	85
spouse_gender	Relations	80
grandparent_gender	Relations	77
children_name	Relations	73
grandchildren_name	Relations	35
great_grandchildren_name	Relations	16
great_grandparent_gender	Relations	41
siblings_name	Relations	16
siblings_gender	Relations	31
children_in_law_name	Relations	17
parent_in_law_gender	Relations	24
siblings_in_law_name	Relations	2
siblings_in_law_gender	Relations	6
other_relations_names	Relations	3
other_relations_types	Relations	22
children_gender	Relations	12
father_name	Relations	7
mother_nane	Relations	9
friends_name	Relations	1
funeral_place	Funeral	90
funeral_date	Funeral	93
funeral_time	Funeral	93
cemetery_place	Funeral	34
cemetery_time	Funeral	13
flowers	Funeral	56
guests_list	Funeral	73
funeral_attire	Funeral	2
charity_name	Funeral	53
reception_place	Funeral	8
reception_time	Funeral	7
reception_date	Funeral	7
funeral_message	Funeral	4

Table 1: Identified features with their frequency.

2.3.2 Features

From the currently labelled 100 samples, the 42 identified features with their respective frequency is shown in table 1.

2.4 Future Work

The possible future works to update the application can be:

- Add a deep learning framework for the solution adaptation.
- Add an active learning framework for data labelling.
- Automate the process of sequence labelling.

[1] `https://github.com/doccano/doccano`