ChurchTools
Gemeinsam Kirche gestalten.

Aufgabe für Backend-Entwickler

Project : Dublettenerkennung

Christal Mangolopa

1 Algorithm

To detect duplicates, I chose to join the tables cdb_person and `cdb_gemeindeperson` in order to have all attributes retrieved from each person. Then, I proceeded to the comparison of pairs of people by evaluating their columns. I did not choose to compare all attributes, but only a sample of the columns. This choice was subjective, even arbitrary, as I had no more information on the meaning of the columns.

The attributes I chose to compare are:

- `'nationalitaet_id'`,
- `'name'`,
- `'vorname'`,
- `'beruf'`,
- `'geburtsname'`,
- `'geburtsort'`,
- `'nationalitaet'`,
- `'taufort'`,
- `'spitzname'`,
- `'titel'`,
- `'strasse'`,
- `'plz'`,
- `'ort'`,
- `'land'`,
- `'telefonprivat'`,
- `'telefonhandy'`,
- `'fax'`,
- `'email'`,
- `'geburtsdatum'`,
- `'hochzeitsdatum'`,
- `'taufdatum'`.

Two strings are considered similar according to the Levenshtein algorithm, with a threshold of 0.85. Their matching value will be 1, otherwise it will be 0.

Pairs with different `nationalitaet_id` are directly considered as not similar. I made this choice because in the database, this value is displayed as non-zero. Moreover, its format is normalized: it is an `int()`.

Finally will be considered as duplicates those who obtain a score higher or equal to 6 matches among these compared attributes.

To accomplish this, I used the recordlinkage package of Python.

2 Framework organisation

2. a) PHP

In the `api.php`, I used PHP to communicate with the Database.
There I make a request to retrieve all the information about each person. This information will be passed to the algorithm function. I chosed to store it in a .json file and call then call that file in the algorithm.

2. b) Python

The algorithm is implemented in Python. This decision was made since the problem of record linkage is quite complex, I choose to use previous work. Therefore, I discovered the package called `recordlinkage` and decided to use it for the given problem.

3 Difficulties in implementation

The first difficulty was in the research part, that is where I spent the more time. After having a small overview of the research in this area, the question was how to implement this in a simplest way. Finding the `recordlinkage` package was a real breakthrough.

When I decided to implement the algorithm in Python, the challenge was to fit the two scripts together. That also let to annoying debugging sessions.

4 Improvements

The algorithm is based on several parameters that have been chosen arbitrarily. These are such as the choice of the attributes to be evaluated, the number of matches to be considered as duplicates, the method of evaluation of the similarity of the strings, etc. Those choices does not guaranty the good quality of the algorithm. It actually does not provide any way to evaluate the accuracy of it.

To do it right, it would have been necessary to have access in some way to some data known for being duplicate. From there, it would now be possible to apply any machine learning instrument to choose the different parameters. Indeed, having access to real labels (true if dublicate, false if not), the parameters would be chosen in order to minimize the error on the real labels.

Also, the project could be improved by providing a `makefile`.

5 Part of the code I am the most proud

I was glad to be able to be able to apply the Python package to make to code more simple. I am also thankful to now be aware of the existing problem of record linkage since it seems up-to-date with the research in the field of data science.

6 Reference

`https://recordlinkage.readthedocs.io/en/latest/`
`https://towardsdatascience.com/how-to-perform-fuzzy-dataframe-row-matching-with-recordl`