Data Wrangling with MongoDB

Clean and visualize data on a map

Jan Hrdlicka
Ostbayerische Technische Hochschule Regensburg
Department of Informatics
Regensburg, Germany
jan.hrdlicka@st.oth-regensburg.de

***Abstract*—In this age, maintaining a healthy business goes hand in hand alongside working with data. The data is a foundation of any successful business and the entrepreneurs are aware that looking into data is what will make them ahead of the competition. This paper deals with cleaning of a dataset containing restaurants, duplicate record detection and data visualization in a map using Python and MongoDB. Step-by-step solution will be provided to demonstrate, how data wrangling task is solved by a student. The precision of duplicate detection by approach in this paper resulted in 97.2%.**

***Index Terms*—Data cleaning, Python, geomapping, folium**

I. Introduction

The usage of social media is dramatically rising [1]. This phenomena goes hand in hand with the amount of data created. People create data by their social media actions - likes, comments, statuses, photos... The companies use these data (also called big data) to improve their products so they can succeed in the highly competetive market.

Some of the big data can contain mistakes; for example due to mistypes and slangs people use. Data with mistakes are often completely understandable for a human but for a machine unusable. This type of data is called dirty data. Latest research shows that approximately 20% of the database data is dirty [2]. Dirty data have less value for the companies and one of the goals for data scientists is to make these data more understandable for the machines and therefore more valuable. Data wrangling consists of gathering, filtering, converting, exploring and integrating data. These steps will be shown in this paper.

Data corruption due to user input is one of many forms how can dirty data arise. Other possible sources of dirty data are the following:

- Poorly applied or no coding standards
- Different schemas used for the same type of record
- Migrations from legacy data systems
- Evolving applications
- No unique identifiers on records
- Programmer errors
- Transmission corruptions

Another complication related to decreased value of data is record duplication. This problem often arises when the records in relational databases do not share a common unique identifier across all tables [3].

The goal of this paper is to show a step-by-step data wrangling task including:

- Web scraping
- Data cleaning
- Duplicate detection
- *NoSQL* database usage
- Geographical visualization
- *Gold standard* evaluation [4]

Each step will be presented with some information about the libraries I picked and my explained approach. The whole work is made in Python, data are saved in a cloud *MongoDB* database *Atlas* [5].

An example dataset containing 864 real world restaurant records will be used. All of the restaurants are located in the USA. After cleaning the dataset and detecting duplicates, the results will be uploaded to *MongoDB*. Next step is to update the records with latitudes and longitudes of addresses using *OpenStreetMap* API and visualize them in a *Leaflet* [6] map via *folium* [7]. This solution is not limited to restaurants, but all of the datasets consisting of real world datasets (companies, personal addresses, etc.) with the same structure.

TABLE I
EXAMPLE OF RESTAURANT DATASET

ID	Name	Address	City	Phone	Type
1	arnie morton's of chicago	435 s. la cienega blv.	los angeles	310/246-1501	american
2	arnie morton's of chicago	435 s. la cienega blvd.	los angeles	310-246-1501	steakhouses
3	art's delicatessen	12224 ventura blvd.	studio city	818/762-1221	american
4	art's deli	12224 ventura blvd.	studio city	818-762-1221	delis
5	hotel bel-air	701 stone canyon rd.	bel air	310/472-1211	californian
6	bel-air hotel	701 stone canyon rd.	bel air	310-472-1211	californian
7	cafe bizou	14016 ventura blvd.	sherman oaks	818/788-3536	french
8	cafe bizou	14016 ventura blvd.	sherman oaks	818-788-3536	french bistro
9	campanile	624 s. la brea ave.	los angeles	213/938-1447	american
10	campanile	624 s. la brea ave.	los angeles	213-938-1447	californian

As can be seen in Table I above, the dataset was probably created by merging two tables. By looking at the phone numbers, it seems that for this small example, the people who were inserting the records into the tables (assuming the database was not created programatically) were trying to stay consistent. The dataset contains a lot of abbreviations so one of the main goals of data cleaning will be to map these abbreviations to whole words.

II. Solution

The code has its own *GitHub* repository [8] and the results are fully reproducible. My solution consists of four scripts.

The reason behind this is that after the data is cleaned, they are uploaded into *Atlas* and then there is a possibility to update the coordinates or create the map from anywhere.

Here is an architecture used for solving this problem:

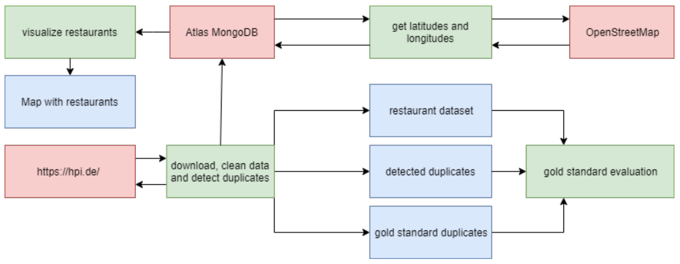

Fig. 1. Solution architecture

Green boxes in Figure 1 above represent Python scripts and their functionalities. Servers are drawn in red boxes and created files from scripts have blue color. To avoid debugging scripts, *Jupyter Notebook* [9] provides a graphical user interface with a possibility to divide the code to single instructions. So it is a good practice to first develop and debug the solution with *Jupyter Notebook* and then rewrite it to a script. Alongside with this application, *Anaconda* [10] contains many useful libraries that simplify work in Python.

Now I will present the solution parts in more detail.

A. *Web scraping*

The files are located on a server [11] in *.tsv* format (tab-separated values). Webpage *https://hpi.de/* also includes some auxiliary model files. In this subsection I will outline how to download three files that are needed - file with restaurants, gold standard duplicate file and non-duplicate file.

First, a client has to send an HTTP request to a server. An easy solution for this provides *requests* library [12]. The HTTP request returns a *Response Object* with all the response data (content, encoding, status, etc). To parse the HTML and to find the hyperlinks to the files, I use *Beautiful Soup* [13].

Another popular web scraping tool used mainly for website testing is *Selenium* [14] library, often paired with *ChromeDriver* [15]. As the website does not require any interaction, usage of these tools is not necessary.

The server response is parsed into *soup* type which helps to extract the content by creating a parse tree from the HTML. The website contains total of 340 elements in two depths. but it takes only two instructions to find required hyperlink, this of course depends on individual structure of the website. Found links are then downloaded into local folder.

B. *Data cleaning*

The quality of this work depends on data cleanliness and therefore this subsection will get proper attention.

I use *pandas* [16] (also a part of *Anaconda* distribution) to convert the dataset to proper Python format. As seen on table I, every record has at least one special character. Characters can often have a special meaning in particular languages [17]. To prevent any problems related to this, I replaced every special character in the dataset with a space. Considering a big variety of special characters present in the dataset, usage of regular expressions [18] makes the most sense.

Now that I got rid of special characters, each of the columns needs to be cleaned under different conditions.

1) Name: One possibility to make sure the strings truly represent real restaurants is to look for similar name with the same address on websites like *Yelp* or *TripAdvisor* through an API and replace the dataset name with the found one. As was stated in introduction, this solution is not limited to restaurants so the name was left as it is.

2) Address: The shortcuts of cardinal directions and street types are mapped to full words. Next step is to get rid of the redundant words and phrases. Any word representing relations to an object (between, near, at, in) and all words that follow this prepositions are deleted. This method is rather risky because the real address could contain these words. Better approach would be to query parts of the address and compare similarities with the real addresses on online maps. Best free solution seems to be the *OpenStreetMap* [19]. However, the geolocation model I use (geocoder Nominatim from module geopy [20]) often does not provide correct results and the structure of geographical response is not consistent - more in the subsection E.

3) City: The process to clean this column is similar to address cleaning. To the list of redundant words also belong cardinal directions. The redundand words are simply deleted from the string. Some of the city names are mapped to new ones, for example ”New York” vs. ”New York City”. This change is only cosmetic and should not make a lot of difference in my work because the geocoder can return and operate with both of these city names.

4) Phone: This columns now contains only numbers and spaces that replaced special characters. To create consistent phone numbers, the spaces were deleted and the numbers were formatted to match the clean dataset model on mentioned website.

5) Type: The people inserting records in this database often provided two phone numbers instead of one. All the second numbers projected into column ”Type”. In my solution, this column will not contain any numbers so all numbers are deleted alongside with meaningless words (e.g. ”traditional”, ”new”, ”classic”, etc.).

TABLE II
CLEANED DATASET

ID	Name	Address	City	Phone	Type
1	arnie morton s of chicago	435 south la cienega boulevard.	los angeles	3 102 461 501	american
2	arnie morton s of chicago	435 south la cienega boulevard	los angeles	3 102 461 501	steakhouses
3	art s delicatessen	12224 ventura blvd.	studio city	818/762-1221	american
4	art s deli	12224 ventura boulevard	studio city	8 187 621 221	delis
5	hotel bel air	701 stone canyon road	bel air	3 104 721 211	californian
6	bel air hotel	701 stone canyon road	bel air	3 104 721 211	californian
7	cafe bizou	14016 ventura boulevard	sherman oaks	8 187 883 536	french
8	cafe bizou	14016 ventura boulevard	sherman oaks	8 187 883 536	french
9	campanile	624 south la brea avenue	los angeles	2 139 381 447	american
10	campanile	624 south la brea avenue	los angeles	2 139 381 447	californian

Table II above displays part of the final, cleaned dataset. Clearly, the dataset still contains duplicates which will be discussed next.

C. Duplicate detection

Duplicates in a database are one of the prime causes of poor data quality and are at the same time among the most difficult data quality problems to alleviate. The evaluation of duplicate detection systems usually needs pre-classified results. As more and more classifiers are evaluated against the annealing standard, more and more results are verified and validation becomes more and more confident.

In my solution, I assume that duplicate records have same phone number (see table II to check that the phone numbers are the same for duplicate records) and similar name. The name similarities are computed by Jaro-Winkler distance [21], which is a string comparison algorithm that is used for measuring edit distance between two sequences. The score is normalized such that 0 equates to no similarity and 1 is an exact match [3]. I use similarity threshold equal to 0.7. Detected duplicates are then saved into a file with same structure as the downloaded Gold standard file. Another similarity algorithm suitable for this could be *WHIRL*. *WHIRL* works well for a large variety of entries and is insensitive to the location of words, thus allowing natural word moves and swaps (e.g., "hotel bel air" is equivalent to "bel air hotel") [3]. This similarity metric does not capture word spelling errors which people often make.

1) Gold standard: In a gold standard, all duplicates are known, and thus, we also know all real-world entities that are only represented by a single record [4]. Figure 2 displays Gold standard Venn-diagram, where downloaded tsv file with duplicates represents red set and duplicates detected by my solution have green color.

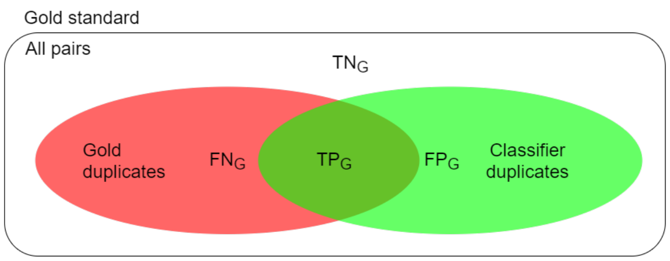

Fig. 2. Gold standard evaluation

I will outline the main ideas behind Gold standard, for more technical depth of this subject see mentioned literature [4]. Table III below explains the Gold standard evaluation.

TABLE III
CLEANED DATASET

Acronym	Meaning
G	A gold standard consists of duplicates DG and nonduplicates NG and is correct and complete).
TP_G	A declared duplicate that is correctly classified (according to the gold standard).
TN_G	A declared nonduplicate that is correctly classified (according to the gold standard).
FN_G	A declared nonduplicate that is actually a duplicate (according to the gold standard).
FP_G	A declared duplicate that is actually a nonduplicate (according to the gold standard).

There are two key figures how to measure classifier quality:

- *Precision* - measure for the correctnes of the result.
- *Recall* - measure for the completeness of the result.

Now let's evaluate my solution:

$$Precision = \frac{TP_G}{TP_G \cup FP_G} = \frac{104}{104+3} = 0.972, \tag{1}$$

as can be seen in equation (1) above, my solution classified 108 duplicate pairs correctly and three pairs as duplicates, but according to gold standard, they were non-duplicate records. Let's see the three false positives:

TABLE IV
FALSE POSITIVES

ID	Name	Address	City	Phone	Type
179	cafe ritz carlton buckhead	3434 peachtree road	atlanta	4 042 372 700	ext international
181	dining room ritz carlton buckhead	3434 peachtree road	atlanta	4 042 372 700	international
180	ritz carlton cafe buckhead	3434 peachtree road north east	atlanta	4 042 372 700	american new
182	ritz carlton dining room buckhead	3434 peachtree road north east	atlanta	4 042 372 700	american
183	restaurant ritz carlton	181 peachtree street	atlanta	4 046 590 400	continental
823	ritz carlton cafe	181 peachtree street	atlanta	4 046 590 400	american

As seen on table IV, these false positives are probably in fact true positives. The reason behind this could be that the records are present in the Gold standard but both of them have a different duplicate therefore they are marked as false positives.

$$Recall = \frac{TP_G}{TP_G \cup FN_G} = \frac{104}{104+8} = 0.929. \tag{2}$$

Here are some of the false negatives:

TABLE V
FALSE NEGATIVES

ID	Name	Address	City	Phone	Type
5	hotel bel air	701 stone canyon road	bel air	3 104 721 211	californian
6	bel air hotel	701 stone canyon road	bel air	3 104 721 211	californian
21	restaurant katsu	1972 north hillhurst avenue	los angeles	4 046 590 400	asian
22	katsu	1972 hillhurst avenue	los feliz	4 046 590 400	japanese
129	uncle nick s	747 9th avenue	new york city	2 123 151 726	mediaterran
130	uncle nick s	747 ninth avenue	new york city	2 122 457 992	greek

these records are truly duplicates. Classification failed often because the distances between names were too high (first two examples) or the phone numbers did not match (last example) or of course both scenarios at once.

As was stated in previously cited article [4] gold standard is the best way to evaluate a classifier, because it gives exact numbers for precision and recall and it is also very easy to apply.

Common practice after duplicate detection is to merge duplicate pairs together. This is not part of this paper and the duplicates will be left as they are now to increase the probability to find coordinates for every restaurant.

D. MongoDB

MongoDB is a *NoSQL* database that suits this task thanks to easy scalability, deployment and free cloud platform provided directly by *MongoDB* team as a service [22]. The data in the database are stored in *JSON* format. The first thing to start using Atlas cloud is to create a cluster. The free cluster provides 512 MB of storage. The set-up of a cluster is fairly easy and requires almost no technical knowledge. I use MongoClient from module PyMongo [23] to connect into the cluster using Python. I can access all the databases that are in the cluster or create new ones so I upload the cleaned dataset to Atlas.

E. Finding coordinates

The database will be updated with latitude, longitude for each record using *OpenStreetMaps* geolocator *Nominatim* from module geopy [20]. To ensure that as much as possible records are updated, first, I get the *Nominatim* response to the city in which the restaurant is. The reason behind this is that the geocode response or required query to find the location successfully is not consistent, as can be seen in table VI below:

TABLE VI
NOMINATIM RESPONSES

Queried city or part of the city	Response
Los Angeles	Los Angeles, Los Angeles County, California, United States of America
Atlanta	Atlanta, Fulton County, Georgia, United States
New York City	New York, United States of America
San Francisco	San Francisco, San Francisco City and County, California, United States of America
Hollywood	Hollywood, Los Angeles, Los Angeles County, California, 90038, United States of America
Malibu	Malibu, Los Angeles County, California, 90265-4797, United States

Even though I have set the country bias to only United States of America, almost 15% of the restaurant addresses were not found. However, only 2% of the geocode city responses were blank so *Nominatim* found the city or city part almost everytime, which shows a little bit of inconsistency in Nominatim responses and queries. Another, maybe better solution for this would be to use *Google Maps*, which uses consistent queries [24] and is probably more successfull finding addresses. Nevertheless, the API is paid.

F. Geographical visualization

The last step is to load the dataset again and create a marker on a map for each restaurant with coordinates. I use *folium* [7] which helps to create *Leaflet* [6] maps. *Leaflet* is a JavaScript open-source library that creates simple interactive maps. There is an option to cluster the markers so the user can for example see how many restaurants are in a city. As said in the section dedicated to duplicate detection, there are still duplicates in the dataset. My solution is to not add the marker with the same name and coordinates on the map. See Figure 3 for the end result.

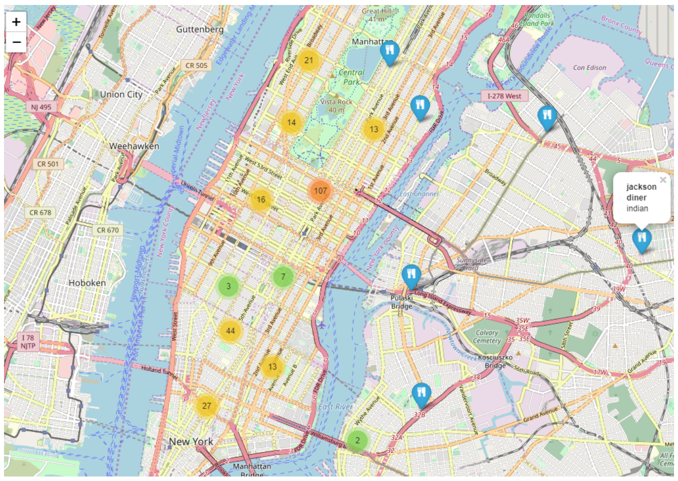

Fig. 3. Data visualization

III. CONCLUSION

This paper showed some basic data wrangling algorithms and aims at students with basic programming knowledge, as the author is a beginner in this field. The web scraping part caused no problems and the files were downloaded every time the script was run. Almost one page is dedicated to data cleaning, which was the main part of this work. The quality of data cleaning is debatable. Duplicate detection using Jaro-Winkler distance provided satisfying results (precision equal to 0.972 and recall 0.929) whereas 15 % of not found coordinates is a fairly high number. This is also caused by the inconsistencies in the geolocator responses and queries.

REFERENCES

[1] L. Hjorth and S. Hinton, *Understanding social media*. SAGE Publications Limited, 2019.
[2] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "Data cleaning: Overview and emerging challenges," in *Proceedings of the 2016 international conference on Management of Data*. ACM, 2016, pp. 2201–2206.
[3] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 1, pp. 1–16, 2006.
[4] T. Vogel, A. Heise, U. Draisbach, D. Lange, and F. Naumann, "Reach for gold: An annealing standard to evaluate duplicate detection results," *Journal of Data and Information Quality (JDIQ)*, vol. 5, no. 1-2, p. 5, 2014.
[5] MongoDB, "Atlas - cloud database," 2020. [Online]. Available: https://www.mongodb.com/cloud/atlas
[6] V. Agafonkin, "Leaflet - a javascript library for interactive maps," 2019. [Online]. Available: https://leafletjs.com/
[7] GitHub, "folium documentation," 2020. [Online]. Available: https://python-visualization.github.io/folium/
[8] J. Hrdlicka, "Data wrangling with mongodb," 2020. [Online]. Available: https://github.com/hrdlickajan/dmdb
[9] Jupyter, "The jupyter notebook," 2015. [Online]. Available: https://jupyter-notebook.readthedocs.io/en/stable/
[10] Anaconda, "Anaconda distribution," 2019. [Online]. Available: https://www.anaconda.com/distribution/
[11] P. D. F. Naumann, "Restaurant dataset," 2020. [Online]. Available: https://hpi.de/naumann/projects/repeatability/datasets/restaurants-dataset.html
[12] K. Reitz, "Requests: Http for humans," 2019. [Online]. Available: https://2.python-requests.org/en/master/
[13] L. Richardson, "Beautiful soup documentation," 2015. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[14] B. Muthukadan, "Selenium documentation," 2018. [Online]. Available: https://selenium-python.readthedocs.io/installation.html
[15] Google, "Chromedriver - webdriver for chrome," 2020. [Online]. Available: https://chromedriver.chromium.org/
[16] pandas, "Python data analysis library," 2019. [Online]. Available: https://pandas.pydata.org/
[17] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.
[18] K. Thompson, "Programming techniques: Regular expression search algorithm," *Communications of the ACM*, vol. 11, no. 6, pp. 419–422, 1968.
[19] M. Haklay, "How good is openstreetmap information? a comparative study of openstreetmap and ordnance survey datasets for london and the rest of england," *Environment & Planning (under review)*, 2008.
[20] GeoPy, "Geopy documentation," 2018. [Online]. Available: https://geopy.readthedocs.io/en/stable/
[21] J. Feigenbaum, "Jarowinkler: Stata module to calculate the jaro-winkler distance between strings," 2016.
[22] M. Stonebraker, "Sql databases v. nosql databases," *Communications of the ACM*, vol. 53, no. 4, pp. 10–11, 2010.
[23] A. Nayak, *MongoDB Cookbook*. Packt Publishing Ltd, 2014.
[24] G. Svennerberg, *Beginning Google Maps API 3*. Apress, 2010.