Duplicate recognition for restaurant dataset*

*

1st Alexander Christoph
OTH Regensburg
Regensburg, Germany
alexander.christoph@st.oth-regensburg.de

***Abstract*—This document describes the analysis and removal of duplicates from the restaurants dataset[2]. The aim is to remove as many duplicates as possible from this data and store it in a cloud hosted mongodb instance without duplicates. A problem searching for duplicated entries of restaurants (or nearly any other dataset) is the format and the different writing of the entries in the data. This problem was already researched by several IEEE members[3]. Within my research I used different approaches to preprocess and detect duplicates. The accuracy of my results is measured with the metrics accuracy, precision and recall. After removing duplicates, the cleared dataset is stored into a mongodb cluster so that it can be accessed any time.**

***Index Terms*—duplicate detection, restaurant dataset**

I. Introduction

In times where big data gets more and more attraction from the industry, it is very important to learn how to deal with it. Especially when it comes to the structure and format of the data. When looking at big data, it is most of the time a problem that there are duplicates and unclean entries in a dataset. This gives inaccurate results when analyzing or working with that data. This happens because most of the time there isn't that much preprocessing happening and there aren't even checks for a standard data format. To learn how to deal with duplicate data, the restaurants dataset is used. This isn't actually big data, but to understand the importance of the preprocessing task, it is pretty good because it's considered a well researched dataset to play with and compare to the gold standard.

In this research, the focus was on how to preprocess the data to detect as many duplicated entries as possible and remove them from the dataset. The hard task was to find ways to prepare the data so that duplicates can be found easily. In order to achieve this, generic cleansing processes, which apply to each column of the data were first used. Then each column was considered individually and specific methods were applied to them.

Once the entire clearing process was completed, ways of finding duplicates had to be investigated. For this purpose, it was chosen that an entry would be recognized as duplicate if a certain column combination had exactly the same values as another entry.

In order to measure the results achieved, various metrics such as accuracy, recall and precision were used (see *section IV Metrics used to measure results*). The achieved metrics can be found in *Table I Metrics for the reached results*

A cloud-hosted mongodb cluster was used to store the resulting data set.

II. Why duplicate detection is important

Duplicate recognition is an important task in data science for several reasons. When a data scientist receives data for research, it is often a problem that the data is not pre-processed and contains incorrect or duplicate entries. Detecting and removing these is a difficult but important task. If you work with impure data, errors can happen. For example, let's look at the restaurant record described in *section III The restaurants data set*. This data can be used to recommend restaurants to a user of an application. If the recommendation contains duplicates with even different fonts for the same restaurant, the user could get confused and stop using our application. Duplicate detection is also important when solving machine learning tasks because most machine learning algorithms cannot handle duplicates and therefore give higher priority to duplicate records, which would result in a less accurate machine learning model.

Because of these problems, duplicate recognition is such a big issue in the field of computer science.

III. The restaurants data set

The restaurant data set investigated in this thesis is a .tsv data set describing the location and type of different restaurants. It contains 864 rows of data with six columns. The columns of the record are:

- id: The unique id of each row
- name: The name of the restaurant
- address: The address where the restaurant is located
- city: The city of the restaurant
- phone: The phone number of the restaurant
- type: The kind of the restaurant (i.e. french or american)

There are 122 duplicated restaurants in the data. These duplicates were hand-selected by some researchers to define a gold standard that would provide the best possible result after removing all duplicates[4]. These duplicates have different variations from each other. For example, some have a different order of words in their name field, such as ”the palm” and ”palm the”. Others have different separators for the phone number like ”310/659-9639” and ”310-659-9639”. Sometimes the duplicate city field is a district of a larger city and sometimes it is the city name itself, like ”los angeles” and

”hollywood”. There are more different variations in the data set, the solution of which is discussed later in the section *section V Preprocessing the data set* for recognition.

This investigation focused mainly on the Name, City, Address and Phone columns, as these have the most useful information for duplicate recognition, while the ID column was omitted because it is only a unique identifier that would not be useful for duplicate recognition. The Type column was omitted because there were too many restaurants of the same type, which would result in an unclean target record.

In addition to the pure restaurant record, a record is also specified that contains all duplicates of the data per id. In this duplicate record there are only two columns, ”id1” and ”id2”, which define the original and the duplicate ID. A record without all these duplicates is considered the gold standard. The achieved results were measured according to this gold standard.

IV. Metrics used to measure results

To measure the accuracy of the results of duplicate recognition, there are various metrics that are frequently used. For all these, it is important to pre-calculate four values using the gold standard data and the archived results:

- True positives (TP): The correct classified true entries
- True negatives (TN): The correct classified false entries
- False positives (FP): Incorrect classified true entries
- False negatives (FN): Incorrect classified false entries

In a gold standard dataset there are no false positives and false negatives[4].

A first metric that is important is the well known accuracy.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Only the calculating the accuracy was not sufficient for this task, since it only provides reliable results if a data set is balanced. In this case, this means that there are as many duplicate entries as there are non-duplicate entries. As an addition to accuracy, I use two more metrics, precision and recall.

Precision measures the accuracy for the minority class by considering only the positive classifications of the result set. It is therefore a good measure of unbalanced data.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recal instead gives the accuracy for the fraction of relevant data.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

The results obtained are measured after each important step of the pre-processing task and are presented in the *section VI Results and possible improvements*.

V. Preprocessing the data set

The research was written in Python, a programming language widely used for data science tasks. The Python-Pandas framework, which provides good data analysis capabilities was used for this research[1]. While the code was written, care was taken to ensure that it would work for other records in the same format.

The goal of the methods used was to minimize the number of unique values in each column representing the same ”real” value so that an equality match for different column combinations (see *subsection V-F Duplicate detection method*) could match them as duplicates.

After each step of the pre-processing pipeline, all double spaces in the strings were removed as precaution.

A complete conclusion of the pre-processing flow can be found in Fig. 1

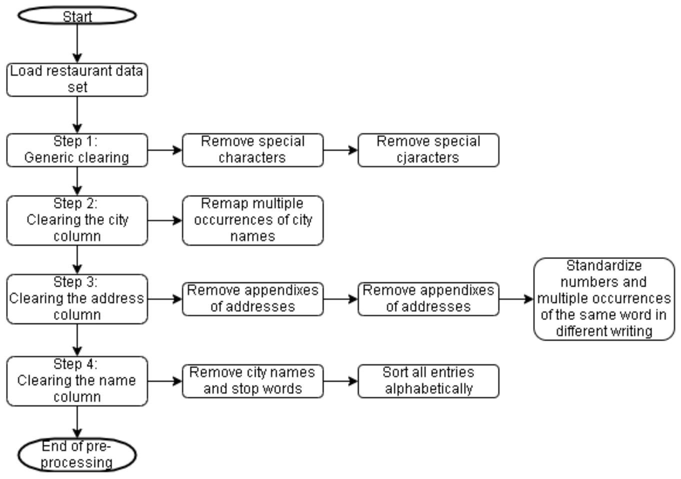
Start
Load restaurant data set
Step 1: Generic clearing
Remove special characters
Remove special cjaracters
Step 2: Clearing the city column
Remap multiple occurrences of city names
Step 3: Clearing the address column
Remove appendixes of addresses
Remove appendixes of addresses
Standardize numbers and multiple occurrences of the same word in different writing
Step 4: Clearing the name column
Remove city names and stop words
Sort all entries alphabetically
End of pre-processing

Fig. 1. Program flow

A. Studying the dataset

The first step in data science tasks is to examine the structure and the different columns of a data set.

This was done by looking at the first lines of the record itself and looking at the values of the different entries to understand their meaning. Afterwards it was necessary to look at each column on its own. This was done by looking at the unique values of each column in particular. Here it was found that e.g. the column ”Type” is very inconsistent with its values (i.e. sometimes the value is ”Pizza” and sometimes ”Italian” for the same type of restaurant) and would not fit as a column to find duplicates, because after normalization there would be too many restaurants of the same type. Therefore the column ”Type” was dropped for this investigation.

The results of researching the other columns name, phone and address are described in the subsections below.

B. Step 1: Generic clearing of the columns

As a first approach it was necessary to perform as many generic operations for all columns as possible.

Many errors in the data came from special characters. So all special characters except spaces were removed from the four columns considered (”Name”, ”Address”, ”City”, ”Phone”). Upper and lower case letters did not have to be considered, since all the data was in lower case. Otherwise it would have been necessary to convert all strings to lower case.

Further investigation of the alphabetic columns has shown that some values contain cardinal points and others do not. Sometimes the cardinal points were even written differently (e.g. ”n” and ”north”). So the next step was to write a regex that recognizes every occurrence of a cardinal point. The regex that recognizes them is:

```
'(( |^)((south)|(east)|(west)|(north)|
(ne)|(se)|(nw)|(sw)|s|w|e|n)( |$))'
```

This was then used to replace everything that matches with an empty string.

After applying generic data clearing, the duplicates for each column have grown. Especially for the ”phone” column, it was noticeable that the detected duplicates have grown from seven to 116, which is quite good.

C. Step 2: Clearing the city column

After the generic clearing, it was necessary to clean up the specifics of each column on its own. It was found that some city names were written inconsistently. For example, sometimes ”la” was written instead of ”los angeles”. So every duplicate occurrence of a city name was mapped to a default value. This map looked like:

```
"la", "west la" -> "los angeles"
"new york city" -> "new york"
```

After applying this mapping, the ”city” column was considered as cleared.

When looking at the resulting metrics, it is noticeable that due to this allocation only two more duplicates were found in this column compared to the previous step. But this is okay, because the city column already contained 817 duplicates before and it is common that there aren’t that many different values when viewing at restaurants that are only located at several citys.

D. Step 3: Clearing the address column

The next step was to clean up the address column. When looking at the entries, it was found that many addresses have unnecessary attachments to describe the location of a restaurant more precisely (e.g. ”3125 piedmont rd. near peachtree rd.”). These had to be removed from the entries. To do this, it was decided to collect the words that separate the actual address from its description and remove everything behind it. The words found were:

```
"between", "off", "near", "at" and "in"
```

Another inconsistency to be resolved was the different spelling of numbers (i.e. ”1”, ”1st” or ”first”). To solve this, a regex was used to detect and then remove appendages of numbers. The regex used to recognize attachments of numbers is:

```
'(?<=\d)(st|nd|rd|th)\b'
```

To normalize all numbers, it was subsequently necessary to map occurrences of string numbers to their numerical representation. This mapping looked like:

```
first -> 1
second -> 2
...
twelfth -> 12
```

For the further cleaning of the ”address” column it was necessary to map different occurrences of the same word to a single one. The map to accomplish this was

```
"la" -> "los angeles"
"ave" -> "avenue"
"rd" -> "road"
"blv", "blvd" -> "boulevard"
"st" -> "street"
```

After applying these clearing steps to the ”address” column, the unique values have decreased from 761 to 739, and the duplicates in this particular column have increased from 116 to 125 compared to the previous step, meaning that nine more duplicated entries have been found in this column compared to the state before applying these clearing steps.

E. Step 4: Clearing the name column

The last column to be cleaned up was ”name”. Initially, a similar procedure as for the ”address” column was used to remove more precise descriptions for the location. So everything after the words ”between”, ”off”, ”near”, ”at”, ”in” and ”of” was removed.

Then it was discovered that the name of a restaurant sometimes contains the name of the city in which it is located. This information is already present in the city column and only adds noise to the data, so it was decided to remove these city names. In addition, some other words were found that would not have been useful in duplicate detection, which were also removed. So all cities in the column ”city” and additionally ”the”, ”restaurant”, ”and” and ”new york city”, were selected and removed. To get these so called ”stop words” as regex, the following code was used:

```
unique_city_list = restaurant_data.name.unique()
city_regex = re.compile('the|restaurant|and|new yo
```

The last step was to arrange the values of the name entries alphabetically, because sometimes the restaurant names were written in different order.

After applying these transformations to the name column, the unique values have been reduced from 773 to 746. The duplicate values in this column have increased from 91 to 118. The results that were achieved as a whole are discussed in *section VI Results and possible improvements.*

F. Duplicate detection method

To detect duplicates, research has shown that most of them could be detected by selecting a combination of the columns ['address', 'name', 'phone'], ['address', 'city', 'name'] or ['name', 'city', 'phone'], in which each value must be the same to be considered a duplicate. All these three duplicate lists are then merged to obtain a complete set of duplicates from the record.

VI. Results and possible improvements

The results obtained are displayed in *Table I Metrics for the reached results*. In total, without viewing the gold standard data set, 101 out of 112 duplicates were found in the restaurants data. This resulted in a recal of 0.90 and a precision of 1.0.

The precision of 1.0 means that there were no false positives in our dataset, which is very good.

The recal of 0.90 means that after applying all preprocessing tasks and selecting duplicates with a combination of different columns, 90% of the real duplicates were found.

The approach used was rather simple and could be improved e.g. by better deciding which combination of columns to use to determine if an item is a duplicate.

Another way to improve the results could have been to better clear the "city" column, so that districts like "hollywood" are mapped to the city they belong to.

Several researchers from the IEEE researched the issue of duplicate record detection and showed lots of different techniques to detect duplicates. They also tried a machine learning approach[3].

TABLE I
Metrics for the reached results

Metric	At beginning	After step 1	After step 2	after step 3	After step 4
Dupliactes in address	92	116	116	125	125
Dupliactes in name	88	91	91	91	118
Dupliactes in city	815	817	819	819	819
Dupliactes in phone	7	116	116	116	116
Detected duplicates	24	68	82	83	101
True Positives	24	68	82	83	101
True Negatives	752	752	752	752	11
False Positives	0	0	0	0	0
False Negatives	88	44	30	29	21
Accuracy	0.9	0.95	0.97	0.97	0.99
Precision	1.0	1.0	1.0	1.0	1.0
Recall	0.21	0.61	0.73	0.74	0.90

VII. Storing data into mongodb

After researching and achieving a reliable result, it was important to store the obtained data somewhere. For this purpose a cloud-hosted mongodb database was chosen. This was done because it is easy to use and mongodb is easily populated from python. With mongodb it is possible to simply push python dictionaries into the database to store them correctly and reliably. The cloud hosting approach was chosen to be able to access the data anytime and anywhere.

Before the collected data was sent to mongodb cluser, it was necessary to change the resulting entries back to more readable values, because in the pre-processing tasks many changes were made to the data, which made them almost unreadable for humans. It was decided to select the first entry for a duplicate from the original data and append the values of "address", "city" and "phone" from the duplicate if they differ from the original. To decide whether one of the columns is the same, the pre-processed data was used.

VIII. Conclusion

In this thesis I have presented my results of duplicate recognition with the restaurant data set. This research was done on a learning basis, so the methods used are rather naive and can be improved. While writing this report, I have gained a clear understanding of why the task of duplicate recognition is so important and so difficult to achieve.

Either way, the results obtained with an precision of 1.0 and a Recal of 0.9 for this data set are really good. Especially because there are no false positive results, which would have been a big disadvantage in this research.

This report has not focused on performance. Many of the methods used could be improved so that the script works better with large amounts of data. One way to improve performance could be to use the mongodb aggregation framework, which is very powerful and could be used for many tasks in the preprocessing pipeline instead of the Pandas library. Another improvement would be the implementation of multi-threading, since many preprocessing tasks could easily be executed on multiple compute units.

For further researches in this topic it would be intersesting to look into machine learning methods that could solve this issue. The other research which was refered in this paper[3] already looked into different methods to solve duplicate detection with machine learning. This report is from 2007, so a similar approach while using new machine learning algorithms could be a possible next step.

References

[1] "Python pandas library documentation", Accessed on: Jan. 9, 2020.[online]. Available: https://pandas.pydata.org/pandas-docs/stable/

[2] "Restaurants dataset", Accessed on: Jan 9, 2020.[online]. Available: https://hpi.de/naumann/projects/repeatability/datasets/restaurants-dataset.html

[3] Ahmed K., Panagiotis G. and Vassilios S., "Duplicate Record Detection: A Survey", IEEE Transactions on knowledge and data engineering, vol. 19, no. 1, January 2007. Accessed on: Jan. 9, 2020.[Online]. Available: https://www.cs.purdue.edu/homes/ake/pub/TKDE-0240-0605-1.pdf

[4] Tobias V., Arvid H., Uwe D., Dustin L. and Felix N., "Reach for gold: An annealing standard to evaluate duplicate detection results" Journal of Data and Information Quality (JDIQ), No. 5, September 2014. Accessed on: Jan. 9, 2020.[Online]. Available: https://dl.acm.org/doi/10.1145/2629687