Spark! Heyrick Research Fuzzy Matching Final Deliverable

Team: Suzy Kirch, Yang Hu, Chengyang He, Haoran Kang

Introduction:

Background on Heyrick Research:

Heyrick Research focuses only on the illicit massage industry, attacking it by understanding, disrupting, and defeating the business models of these illicit massage businesses, all by data-driven means.

Goals:

Our primary goal was to create a dynamic, scalable script for fuzzy matching between the Rubmaps dataset (a dataset for the known illicit massage businesses) and any other dataset they want. As part of this, we needed to create match criteria, with a confidence score for each match.

Data:

We received Rubmaps and Google Places data from Heyrick Research. These data sets still needed to be standardized. To standardize, we converted the phone number to E164 format. The addresses were split into a zip code column, a state column, and first line of address plus city. This standardization happened with these two data sets in addition to the Yelp data we scraped.

Data Scraping: Yelp Data

We used Yelp fusion API to scrape Yelp for massage places. Because Yelp fusion API has a limit of 5000 calls per day, it may take several days to get all the data. (It took us 3 days to get the data we needed.) We gathered the requested states of Massachusetts and Missouri.

All information in Yelp is scraped and stored in a json file. The json file is then converted to a csv file. (If you are using our scraping file, json2csv.py cuts down the columns to just the relevant ones for matching. If you want to adjust what columns get kept when converting to csv, that is the file to adjust.) All data files are stored by state and are further organized by zip code. Because Yelp's data's address contains geographical location as well, we dropped the latitude and longitude from address and made new attributes for them in the dataframe.

Most of the data is complete. The few missing are filled with 0 or NaN. As long as the state and zip code columns are correct, the row can still be used for matching, though with missing values, it will have a lower match score.

The relevant columns of the datasets are:

- Business ID: location_hash (Rubmaps), googlePlaceID / googleID (Google Places), and id (Yelp)
- Business name
- Business address: this has been split into first line of address plus city, state, and zip code
- Business phone number

Code and Reproducing Results:

Code Files:

- standardize_phone.py: standardizes the phone numbers
- addr_std.py: standardizes the addresses and splits them into the appropriate columns
- match.py: runs through all 50 states plus DC and computes match scores, saving the high chance matches of each
- merge.py: merges all 51 high chance match files into one xlsx file

Reproducing Results:

Three folders (raw, clean, data) should be created under the same directory as scripts. Input data, Rubmaps data, and Google Places data should be stored in the data folder. Scraped Yelp data can go directly into the raw folder. First run standardize_phone.py to get standardized phone numbers of Rubmaps and Google Places—the phone numbers in Yelp are already standardized. It also splits Google Places data into smaller chunks (by state) as standardization phone. It outputs files to the folder raw. To standardize addresses, run functions from addr_std.py. In this file, it takes input from the raw folder and outputs to the clean folder. The file matching.py includes all matching functions. It takes input from clean and outputs it to the data folder. (Note: all initial input and final output are in the data folder.) Finally, merge.py takes all 51 high chance match files and merges them into one xlsx file.

On library and algorithm choice:

We specifically chose fuzzywuzzy for a couple of reasons. One reason is that it seemed easier to use, which meant it was a good starting place. More importantly, though, from our research, it appeared as though the other library of choice, spaCy, was really only good at identifying identical matches, which is not what we need for this project.

We settled on fuzz.token_set_ratio after trying several different fuzzy matching score calculators. We found that the results of fuzz.token_set_ratio was a very close match to how we would score matches and what we would consider to be matches. Thus, we used fuzz.token_set_ratio to score all of our matches.

In our trials of fuzz.token_set_ratio, we quickly realized that an address match would not be sufficient, as you could have a shared address, but a different suite in the building, for example, and a completely different business name, and these would disqualify the pair from being a match. We, therefore, compare name and address together and verify the match with individually matching name, address, and phone number.

Results:

The column headers for the results files are as follows:

- index
- location_hash: the unique ID for the business in the Rubmaps dataset
- googleID: short for googlePlaceID, the unique ID for the business in the Google Places dataset
- rubmap_ad: the business' name and address, separated by a comma, as stated in the Rubmaps dataset

- google_ad: the business' name and address, separated by a comma, as stated in the Google Places dataset
- name_score: the calculated score of name comparison
- addr_score: the calculated score of address comparison
- name_addr_score: the calculated score of the comparison of the combination of name and address
- g_phone: the business' phone number, as stated in the Google Places dataset
- r_phone: the business' phone number, as stated in the Rubmaps dataset
- phone_score: the calculated score of phone comparison
- city: the city in which the business is located
- tier: value 1-5, see below for details about this column

Match Score Cut Off:

We do not have as large of a portion of Rubmaps rows matched as we had hoped. Lowering the match threshold, however, will not fix this problem. We set the name_addr_score to a threshold of 90 because if it is any less than that, we actually wind up with more false positives, not more true matches. On the flip side, if we set name_addr_score higher than 90, we miss a fair number of true matches. We decided on this threshold by parsing through our spreadsheet of all scores, looking at each row, deciding what were matches, and keeping track of the scores for each category in our matches.

Match Tiers:

For fuzz.token_set_ratio, 100 is a perfect match, and 0 is a complete non-match. Parsing through our scores spreadsheet, we found that we had a lot of match scores that were 100. We decided to utilize this fact in creating our match tiers. We have created match tiers, spelled out below, for a straightforward way to know the confidence of a match.

We have saved all comparisons where name_addr_score was at least 90. From there, we divided all results into match tiers, with 1 having the highest confidence of being a match and 5 having the lowest confidence of being a match.

- Tier 1: name_score==100 and addr_score==100 and phone_score==100
- Tier 2: name_score==100 and addr_score==100 and phone_score=/=100
- Tier 3: (name_score==100 or addr_score==100) and phone_score==100
- Tier 4: addr_score==100 and name_score=/=100 and phone_score=/=100
- Tier 5: all others, with name_addr_score >= 90

Limitations:

We had a few limitations on our work for this project.

1. As already discussed, we did not have as many matches as we might have expected, despite the large number of businesses in the Google Places dataset.
2. Due to the property of fuzzy matching and large number of data, it is hard to check the correctness of matches because it must be checked by hand and is therefore really time consuming.
3. Our match criteria is generated only based on MA and MO data.

4. Scraping is time consuming. In addition, the larger states need a couple of days to scrape everything due to the 5000 call per day limit.

Future Work:

We have run our match script for all 50 states plus DC for Google Places, but we have only done a couple of states for Yelp. Future work can include scraping the rest of the 50 states in Yelp and running the matching script on that data. The matching criteria may be altered, depending on the overall results of all states. Also, it is possible to develop a fuzzy matching algorithm by hand to increase the matching accuracy based on the source code of the current fuzzywuzzy library.

We did not create any visualization of our matches. It may be interesting to see all matches on a map, with color coded dots based on tiers. This could help to answer the question of whether there are clusters of these illicit businesses.