Spark! Heyrick Research Fuzzy Matching Deliverable 3

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.

Datasets:

We received two datasets from Heyrick Research: Rubmaps and Google Places. We scraped Yelp using the Yelp Fusion API.
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

Included Files:

There are several Excel files included in the folder for this deliverable. They are as follows:

- high_chance_MA.xlsx and high_chance_MO.xlsx are the results files for MA and MO, respectively, using Google Places data
- high_chance_MA_yelp.xlsx is the results file for MA using Yelp data
- rm_all.xlsx has the standardized Rubmaps data, for the whole country
- nat_MA.xlsx and nat_MO.xlsx has the standardized Google Places data, for MA and MO, respectively

We have also included the following python and Jupyter notebooks:

- phone_rm.py is the file that standardized all phone numbers in Rubmaps
- phone_nat.py is the file that standardized all phone numbers in Google Places
- addr_std.py standardized all addresses from Google Places and Rubmaps
- Matching_byState.ipynb handles the matching and categorization of tiers

Code and Results:

The column headers for the results files (results files saved by state) 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 and Tiers:

All comparison scores were calculated with fuzz.token_set_ratio in the fuzzywuzzy python library. 100 is a perfect match. 0 is a complete non-match. 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 the closest match to how we would score matches and what we would consider to be 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 are therefore comparing name and address together, and verifying the match by individually matching name, address, and phone.

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

Reproducing Results:

To reproduce our results, you must first standardize the data files. This includes standardizing column names, formatting the phone number in E164, and splitting the address into first line plus city, state, and zip code. (See "Included Files" section for the description of what files to use for each step.) In the Jupyter notebook Matching_byState.ipynb, the data set being matched to (in our case Rubmaps) gets loaded into the variable df_rm. If you are using Rubmaps, be sure to filter the full dataframe by the state of choice. The data set that you are

using as the other dataset for the match gets loaded into the variable df_nat (in this case, the file with the data of your chosen state). Running all cells in the Jupyter notebook will give the scores and tiers for your chosen state and will save all columns to a file called high_score_statename.xlsx.

Datasets from Google Places are separated into different spreadsheets by state. Output will be data by states as well. Eventually, we will have a for loop processing all dataset together. For now, we have test sample output of Massachusetts and Missouri.