Zip Code Matching Application

Created by: Shashank Bharadwaj (shashab), Caroline Kinnen (ckinnen), Carly Schippits (cschippits)

Project Overview

Our application finds the five zip codes in a user-specified state that are most similar to a user-specified zip code based on the user's parameters. The application helps users determine where to live or travel.

The user interacts with a Django interface and types in a starting zip code, chooses a target state from a drop-down menu, and checks any combination of nine boxes corresponding to the parameters they care most about. The options are demographics, business composition, political ideology, school quality, access to libraries, access to museums, walk score, weather, and housing prices.

Behind the scenes, the application pulls and normalizes zip-code-level data. For each zip code in the target state, the application computes the average squared difference between that zip code's attributes and the starting zip code's attributes.

The five zip codes with the smallest average squared difference are returned via the Django interface, along with a similarity score that ranges from 0% to 100% (0% indicates perfectly dissimilar zip codes and 100% indicates perfectly similar zip codes). The user is also presented with an interactive map featuring all five zip codes. Each map pin is clickable and draws an approximate zip code boundary when clicked.

Structure of the Application

Our project is divided into five main components:

1) Data collection / cleaning
    - This step was only executed once.
    - We used nine data sources:
        i) Demographic data from the U.S. Census Bureau (pulled using Python's CensusData library)
        ii) Business composition data from the U.S. Census Bureau (downloaded as a CSV)
            - The original CSV is way too big to upload to GitHub, but a sample of the data is included in the data/raw_data/ folder.
        iii) Political ideology data based on 2016 presidential election results (downloaded as a CSV)
        iv) School quality data from GreatSchools (scraped from the web)
        v) Libraries data from the Institute of Museum and Library Services (downloaded as a CSV)
        vi) Museums data from the Institute of Museum and Library Services (downloaded as a CSV)
        vii) Walk score data from Walk Score (scraped from the web)
        viii) Weather data from Weatherbase (scraped from the web)

ix) Housing prices data from Zillow (downloaded as a CSV)

2) Database creation
    - This step was only executed once.
    - We aggregated our nine cleaned datasets (available in CSV format in the data/ folder) into a SQL database called zip_db.sqlite3.
3) Django interface
    - The interface is launched every time the user opens the application.
    - The code is available in the ui/mysite/ folder.
4) Algorithm for finding the best zip code matches
    - The algorithm runs every time the user fires a search.
    - The code is available in the algorithm/ folder.
5) Interactive map
    - The map is loaded every time the user fires a search.
    - The map was built using the Google Maps Javascript API.
    - The code is available in the ui/mysite/static folder.

Data cleaning scripts can be run by typing `./install.sh` from the command line (this file takes about 10 minutes to run). Running `./install.sh` also launches the application. Web scraping scripts can be run by typing `./run_scraping.sh` from the command line (this file takes a few days to run).

Accomplishments

For the most part, we accomplished everything we set out to accomplish. But we did run into some issues at the data collection and map building phases.

Lack of data availability caused us to slightly alter our expectations for data quality. Missing data is an issue across most of our datasets. The school ratings dataset has the most missing data, because roughly one third of U.S. zip codes do not contain any schools. Since our final database contains some missing data, our application ignores NAs when computing the average squared difference between zip codes. However, if data is missing for a user-specified zip code for all user-specified parameters, then the application has no data on which to perform calculations, and it returns an error message that asks the user to select additional parameters.

We had also originally planned to display a static map of the target state with the top five zip codes highlighted. However, this approach wasn't very effective, because most zip codes are too small to be visible on a state map. We instead opted for an interactive map that would allow the user to zoom in and out as they please.