Data Science in the Wild: Final Report

Daniel Nissani dn298 | Ziyu Qiu zq64 | Wei Duan wd255

1 The Problem

Moving to a new city or an unfamiliar city can be very frustrated. Even with clear critera of what they are looking for, it can still be very time-consuming and sometimes people don't even know where to start with. And this is actually a headache for a lot of people. More than 40 million Americans move at least once a year, which occpies over 14 % of the US population.

They have to look at safety, restaurant quality, etc. across various resources. The process of finding the data is very tedious and sometimes it can be very hard to understand the raw data without proper visualization even when they find it. They also have to make trade-off between different neighborhoods, just like manually solving an optimization problem by hand. We aim to use data science to solve this for them by aggregating the information across resources and build intelligent optimization models to recommend the best places to live to users.

2 Data Collection

To do this project, we knew we wanted to include amenity data as well as safety data, transporation data, and pricing data. However, since we wanted to be able to pick neighborhoods, we needed a consistent way to pick out neighborhoods. Some data used latitude and longitude, others used precincts, and others still used neighborhoods in New York City. In the end, we chose to use neighborhoods from NYC Open Data's Neighborhood Tabulation Areas dataset. This dataset gave us 195 neighborhood names for the neighborhood boundaries in NYC.

From there, we collect two different public datasets from the Google API. We collect laundry, library, park, pet store, pharmacy, car repair, car wash, restaurant, school, dentist, doctor, gym, train station, and veterinary care data for each neighborhood in NYC. This is collected through the Google maps places API, where we change each neighborhood name into a geocode and then query the key terms listed above. For each key term, we get 20 results that contain

rating information collected from people who review the places on Google. We take the ratings for each neighborhoods amentities and find the average rating of each amenity in each neighborhood. If a neighborhood amenity has an average rating of 0 (meaning the amenity does not have any ratings), we impute the average value of that amenity among all neighborhoods. This then became a json file for us to query at run time for the optimizer. All scores are between 1 and 5, except for price which is between 1 and 4.

We also collected crime statistics data. This file came from NYC Open Data's Citywide Crime Statistics dataset. We dorpped a lot of the columns as they were unnecessary for our project, keeping the geocodes for the locations of the crimes and the types of crime that occurred. We used the coarser types (Felony, Misdemeanor, and Violation) as these were going to be used to provide us an actual score. To calculate the neighborhood from the geocodes in the crime dataset, we encoded the neighborhoods as geocodes and found the nearest geocode to generate the neighborhood that the crimes took place in. We will go over how we calculate the actual crime score in the optimizer section.

The last dataset we were able to collect was the distances from a neighborhood to a person's work address. As people are moving to NYC, the only data they have is where they are working and what preferences they have for where they live. Thus, we calculate the distance of each neighborhood to the workplace that the person specifies on the fly using the google maps API. Distances were calculated by the amount of time it took for someone to get from a neighborhood their place of work using four methods of transportation: walking, driving, transit (all modes of public transit), and bicycling. We discuss how this influences the optimizer in the optimizer section.

We sadly were not able to use the Zillow API. Although it contains all the data we want, collecting it would be intractable. What we would want to get is an average rent for different types of bedrooms in each neighborhood. However, what Zillow provides is the price of rent of units at a specific *address*. Thus, we need to apriori know all the addresses and map them to an individual neighborhood and then calculate the average. Given the time parameters of this project, we saw this as intractable and left it out of this first iteration. We hope to include such information in the future.

3 Development

The app takes in user input from a frontend built based with Ruby on Rails, and pass the data to a backend server built with Flask. At backend, queries are sent to APIs from sources like GoogleMaps, and get realtime information about fields like commute time, restaurant information, etc. The data is aggregated and fed into an optimizer model based on heuristics.

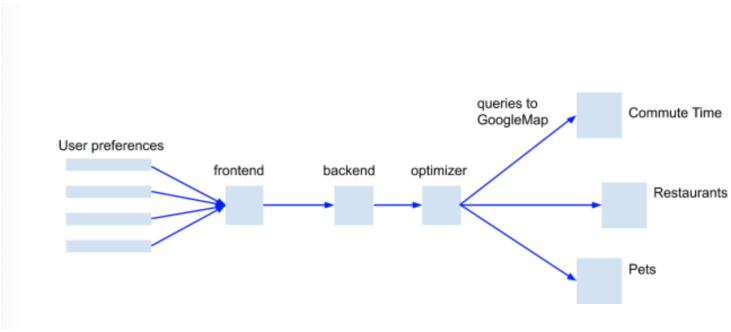

Figure 1: Forward information flow

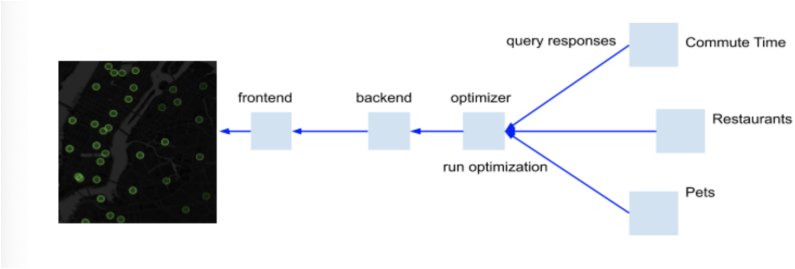

Figure 2: Backward information flow

3.1 Front End

We used Ruby on Rails to write the front end and send HTTP requests to the hosted optimizer. Essentially it is formed by two parts: a form that allow users to enter their preferences on the left and a map on the right. After the user submit his/her preferences, the recommended neighborhoods will be highlighted by circles in different colors, where the brighter the color is, the higher the recommendation score is. As shown by Fig 4, the user will also see a list of recommended neigherhoods in text on the left span.

3.2 Back End

The backend will take a POST request from frontend, containing company address, means of transportation, safety concern, restaurants, gyms, laundromat, pet friendliness, healthcare, and school information. The request data is converted from string to numeric and passed to optimizer. After the optimizer gives a score of each neighborhood, the backend will visualize the scores with circles of different brightness on the map and send the map back to frontend to show recommendation to user.

Figure 3: Frontend Demo

3.3 Optimizer

The optimizer is a linear function that takes a kind of average over all the different features collected. Here, we describe how amenities are combined and normalized within the optimizer.

For restaurants, we don't only care about the quality of the restaurants, but we also care about the price of the restaurants. Thus, we needed to find a way to penalize restaurants for having high prices even though they had good food. We decided to do the following:

$$\text{restaurant quality} - .25 \times \text{restaurant price}$$

This gave us a score that empirically corresponded to how much we care about the price of a restaurant. Moreover, this made it so at most, the price would be penalized by 1, thus not terribly effecting the restaurant score and keeping it in line with the other scores.

For pet friendliness, health, and driver friendliness, we had multiple data points to combine together and decided to take the average of those data points.

$$\text{Pet Friendliness} = \frac{\text{Park quality} + \text{Pet Store Quality} + \text{Vet quality}}{3}$$

$$\text{Health} = \frac{\text{Doctor quality} + \text{Dentist Quality} + \text{Pharmacy quality}}{3}$$

$$\text{Driver Friendliness} = \frac{\text{Car repair quality} + \text{Car wash Quality}}{2}$$

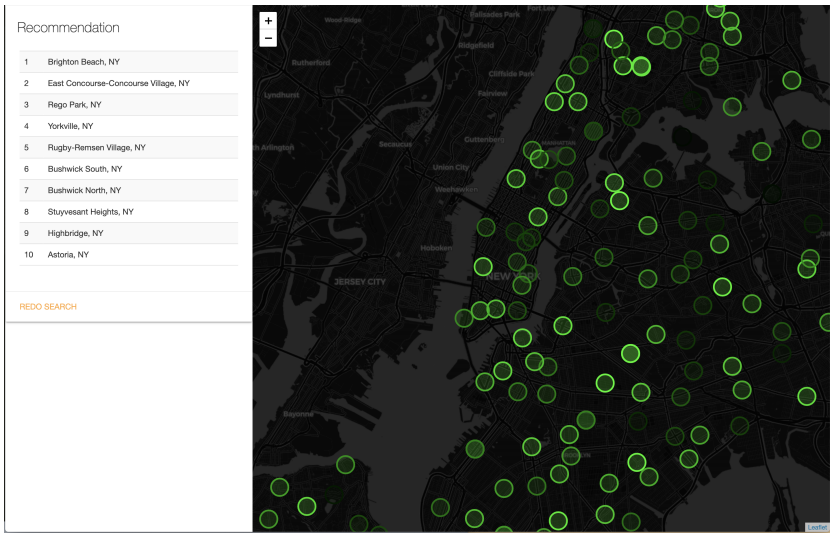

Figure 4: Recommendation List

Notice that these are some basic averages to get a baseline for the optimizer. The normalization makes it so they stay within the original parameters of the features. The rest of the amenity features had no weighting to them, and were taken as the raw averages computed during the data collection phase.

The distance data was calculated on the fly, where distances were calculated using four types of transportation: driving, walking, transit (all forms of public transit), and bicycling. The user got to choose one of these or decide they do not care. If they did not care, all four were computed and the fastest one was chosen. To weight this, we forced the user to determine how long they want it to take to commute to work. Using this time, we compute a specific score for the time it took. If the time it took to commute to work was less than the users desired time, we calculated:

$$\frac{\text{desired commute time}}{\text{actual commute time}}$$

Notice that this value will increase as the actual commute time calculated decreases. This is intentional, as the closer you are to your job the better (this is an assumption, but we believe it holds for most cases). Moreover, if the actual commute to work was more than the desired time, we calculate the following:

$$-\frac{\text{actual commute time}}{\text{desired commute time}}$$

Notice here that the score increases in absolute value as the actual commute time increases, but since this is in absolute vlaue, the value actually decreases. Thus, we are attributing a “more negative” score as we move farther away.

Lastly, for the security score, we compute it by taking the ratio of felonies, the worst type of crime in a neighborhood, over all crimes that have happened in the dataset in that neighborhood. However, for this, we also compute how much a person cares about security scores. If they don't care at all, we give every neighborhood a score of 0. If they do care a little bit, then we give them a score of -2 times the ratio described above. If they just care about it, then we multiply the ratio by -4. And if they really care about it, then we multiply the score by -8. Notice that when the person cares about the safety score, we weight it enough so that it matches the amenity parameters. We use the amenity parameters to adjust the weights of these (by halfing and doubling) so that the score corresponds to the person's preferences.

Once we collect all these values, we average them over the number of preferences the person has, and then output a score for each neighborhood. We then rank those scores to tell the individual the top neighborhood choices, and provide a heatmap for their exploration.

Note that many advancements can be made here. Specifically, we could do some machine learning to adjust the weights and really personalize the experience to each user as we learn what they want over time. We discuss this in more detail in the future work section.

3.4 Delopyment

In order to serve more people in need, we decided to make the tool open-source on github at `https://github.com/ziyuqiu/LIVEASY_RoR`. We also deployed the website to heroku at `https://ct-liveasy.herokuapp.com/` so that people can easily use it.

4 Data Analysis

Much of our analysis was done through visualizations and motivates further work that we want to do later on. Our visualizations are of the actual distributions of the different scores we computed. Most of these distributions were normally distributed, which we believe indicates that we correctly made a baseline for the scores. Here is an example of a distribution that was fairly normal:

Notice here that the distribution looks quite normally distributed, with few outliers. This means that the scores are well distributed. And since many scores have this type of distribution, we know that the distribution of the scores themselves will be fairly normal as well.

However, a few scores are not normally distributed and further work needs to be done to check how to weight their scores so that they will contribute correctly.

Note that the most erratic of these is the train scores. This will greatly affect the scores if someone chooses transit as their mode as transit. Moreover, the double hump of the pet friendly and driving distributions tell us that neighborhoods will get bimodal scores that may bias the optimizer to certain

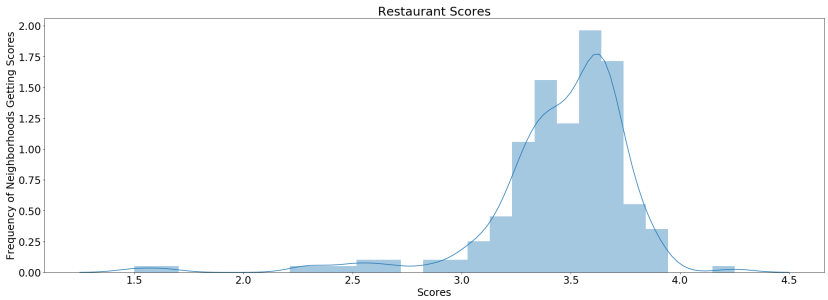

Figure 5: Distribution of Restaurant Scores

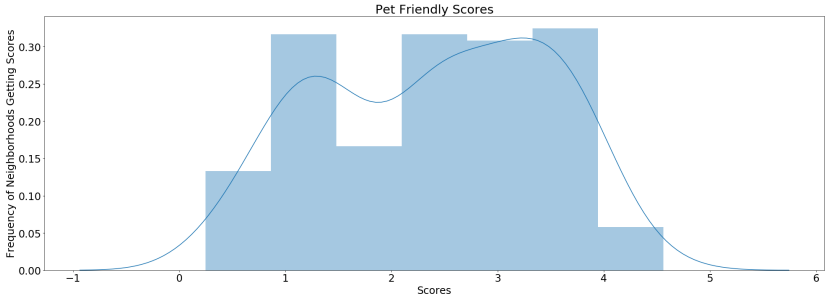

Figure 6: Distribution of Pet Friendly Scores Scores

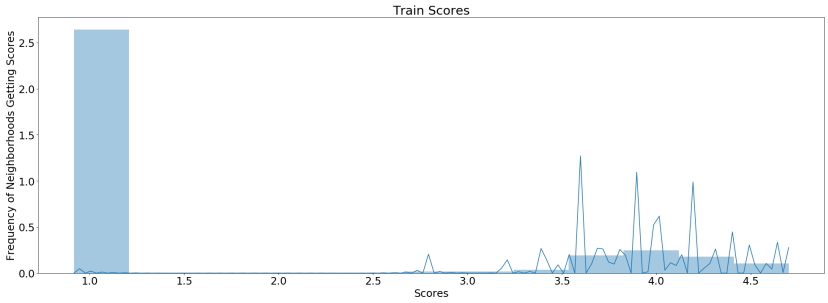

Figure 7: Distribution of Train Scores

neighborhoods disproportionately. Further statistical analysis needs to be done on how to make this better.

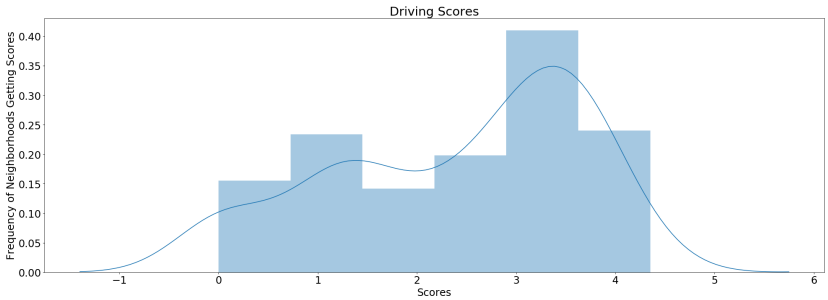

Figure 8: Distribution of Driving Scores

5 Future Work

5.1 Architecture

In the future we can add a recommendation evaluation system by collecting users' feedback about whether they finally chose to live in the places we recommend. By collecting the simple binary data of whether a user likes a place or not, we can optimize our current heuristics based optimization model in an autonomized way. One potential approach is to build a machine learning model to classify good heuristics and bad heuristics.