Building a Simple, Fast Restaurant Recommendation System Using the Foursquare API

Jeremy Nathan

1. Introduction

Previously, neighborhoods in Toronto were segmented using K-means clustering. Making use of the data provided by the Foursquare API, various neighborhoods were grouped based on the categories of restaurants most frequently found in them. Using this method, ethnic enclaves, such as Chinese, Indian, and Greek, were located across the Metropolitan Toronto Area.

My capstone project draws upon the same dataset used for the segmentation assignment to build a restaurant recommendation system. This system would take in a restaurant (ID) as an input and output a list of restaurants that a user might also like to visit based on the input restaurant's attributes, including proximity and popularity.

2. Data

Data was obtained from the Foursquare API. Two endpoints were used: The "Explore" endpoint, which provided the restaurants for each postal code as well as their categories and coordinates, and the "Likes" endpoint. For the purposes of building a prototype recommendation system, a subset of ~500 restaurants within approximately contiguous neighborhoods was chosen. This strategy has multiple benefits. First, obtaining information from the Likes endpoint beyond the count of likes for a restaurant seems to be a rate-limited/premium call to the Foursquare API. As a result, by only examining a subset, the number of API calls that would need to be made to the Likes endpoint would be reduced such that all the relevant data could be obtained in one cell run (as opposed to making multiple calls, each appending to a CSV or json file in the directory). Second, the matrices being manipulated to create the recommendation system would be smaller in size, leading to a lower computational cost and faster iteration in development.

Minimal preprocessing was performed on data retrieved from the endpoints. The Explore endpoint was queried by postal code, and the data returned was loaded into a dataframe consisting of each restaurant's name, ID, category, and coordinates. The most significant step was to drop restaurant ID duplicate rows, which arose when the same restaurant would occur in the results of API calls made for separate postal codes (i.e., if a restaurant was close to the boundary between two postal codes). In addition, all fast-food restaurants were dropped. With these two steps, the number of rows in the restaurant dataframe was reduced by up to 50%.

3. Methodology

To implement a recommendation system for n restaurants most efficiently, a recommendation matrix R of dimension (n, n) was constructed. Each element R_{ij} corresponds to the restaurant J's score relative to restaurant I when restaurant I is the search key. In other words, if a user were to ask for recommendations for restaurants based on restaurant I, R_{ij} would indicate

how strongly restaurant J would be recommended. The algorithm selects the ten top values for $R_{ij}|_{i=I}$ as the restaurants to be returned to the user. Since the matrix R is precomputed, the retrieval of the list of recommendations is nearly instantaneous. In addition, the recommendation matrix can be stored in a database or local .csv file to conserve memory.

Definitions

$$\boldsymbol{R_{ij}} = \left[\log\left(\frac{\mathbf{1}}{\boldsymbol{dist(i,j)}}\right)\right]_{\boldsymbol{rm}} + [\boldsymbol{cat(i,j)}]_{\boldsymbol{rm}} + [\boldsymbol{ref(i,j)}]_{\boldsymbol{rm}} + [\boldsymbol{likes(j)}]_{\boldsymbol{rm}} \quad (\mathbf{1})$$

$$dist(i,j) = \sin^{-1}\sqrt{\sin^2\left(\frac{j_{lat} - i_{lat}}{2}\right) + \cos(i_{lat})\cos(j_{lat})\sin^2\left(\frac{j_{lng} - i_{lng}}{2}\right)} \quad (1a)$$

$$cat(i,j) = \begin{cases} 1 & \text{i, j have same category} \\ 0 & \text{i, j have different category} \end{cases} \quad (1b)$$

$$ref(i,j) = \text{count of users who liked both restaurants i and j} \quad (1c)$$

$$likes(j) = \text{count of users who liked restaurant j} \quad (1d)$$

Each element of R, for a pair of restaurants i and j, was computed according to equation (1). Equation 1 is computed as the sum of sub-equations 1a, 1b, 1c, and 1d. Equation 1a corresponds to the great circle distance between two restaurants (the subscripts lat and lng denote the restaurants' latitude and longitude, respectively). The inverse of this value is used in the equation because closer restaurants should have a higher score. Equation (1c) is included in addition to the number of likes the restaurant has (1d) because a nonzero value for (1c) would indicate that the people who like restaurant i and the people who like restaurant j are not disparate groups.

Since each element of the sum in equation (1) can be calculated independently, R can be constructed as the sum of 4 matrices, each calculating a sub-equation (1a – 1d) listed above. The following names are assigned to the matrices according to the sub-equation they calculate: the proximity matrix (1a), the category matrix (1b), the cross-reference matrix (1c), and the likes vector (1d). Before being added together, they must be scaled, as denoted by the $[\,]_{rm}$ brackets around each term in equation (1). The "r" stands for row-wise scaling, and the "m" stands for min-max scaling, the type of scaling that was applied. The rationale for choosing this method of scaling is discussed below.

Scaling

The most important step in creating the recommendation matrix was choosing exactly how to scale each attribute matrix. Prior to scaling, the raw matrices had values that differed in orders of magnitude. For instance, as seen in Figure 1a, the inverse distance matrix contained values ranging between 10 and 10^7. On the other hand, the category matrix, by definition, only contained 1's and 0's, and the venue likes vector contained values between 0 and 116. By scaling each attribute's values to fall within the same range, each attribute would have an equal influence on the recommendation values.

The first scaling operation was performed on the proximity matrix. The initial distribution of the data in the proximity matrix (i.e., the inverse of the pairwise distances between restaurants) is shown on in figure 1a. This data represents too wide of a range for practical use; even when scaled down, the result would have most points scaled too close to 0 to have any effect on the recommendation matrix. Taking the logarithm of the data yields the distribution on the right, which has a much more desirable distribution since it is on the same order of magnitude.

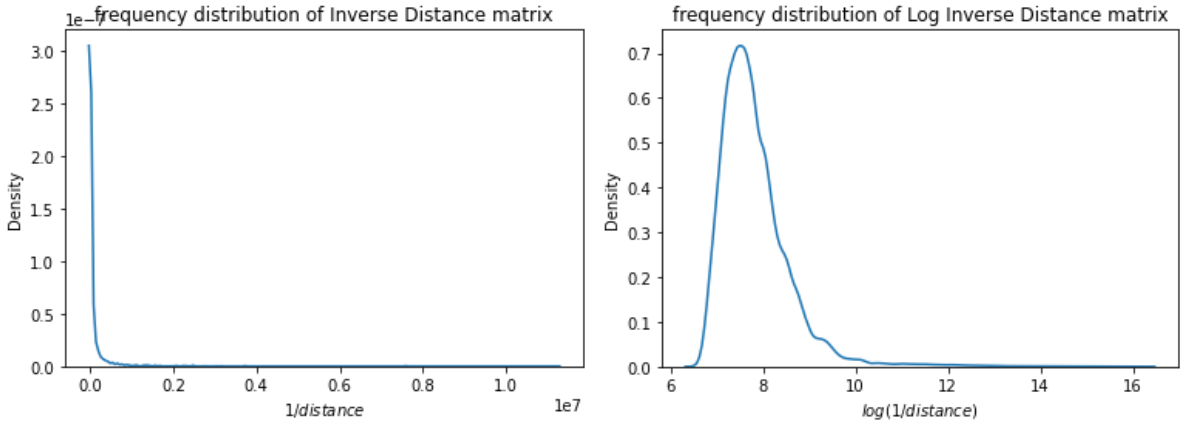

Figure 1a, 1b: frequency distribution of inverse (left, 1a) and log inverse (right, 1b) pairwise distances

Next, the scaling methods must be selected. Since the matrices being scaled are symmetric, and the proximity matrix's diagonal values are all infinite (resulting from 1/0 being calculated), two different approaches to selecting the subarrays for scaling were examined. The first was triangular scaling. This method involved selecting the upper triangle of the square matrix, flattening it, scaling the resulting single-dimensional array, and then repopulating the upper triangle (and copying the values to the lower triangle to maintain symmetry). The next was row-wise scaling, in which each row was scaled (ignoring the index part of the diagonal).

Though both triangular and row-wise subarray selection produced matrices that had relatively balanced impacts on the final recommendation matrix, row-wise scaling was selected. This is because when triangular scaling is used, indexing into a row of a scaled matrix yields a small subset of the scaled distribution of the entire upper triangle that could likely contain small values. In other words, if the scaled upper triangle had values ranging from [-1, 1], an indexed row might range from [-0.2, 0.5], for instance. If this occurs, then the influence of the matrix is unexpectedly diminished in the final recommendation matrix, as shown in Figure 2 below. This behavior would only become more pronounced for a larger sample size n, since each row would make up a smaller fraction of the values in the upper triangle. On the other hand, row-wise scaling does not have this problem since each entire row is scaled, resulting in the same scale for each row.

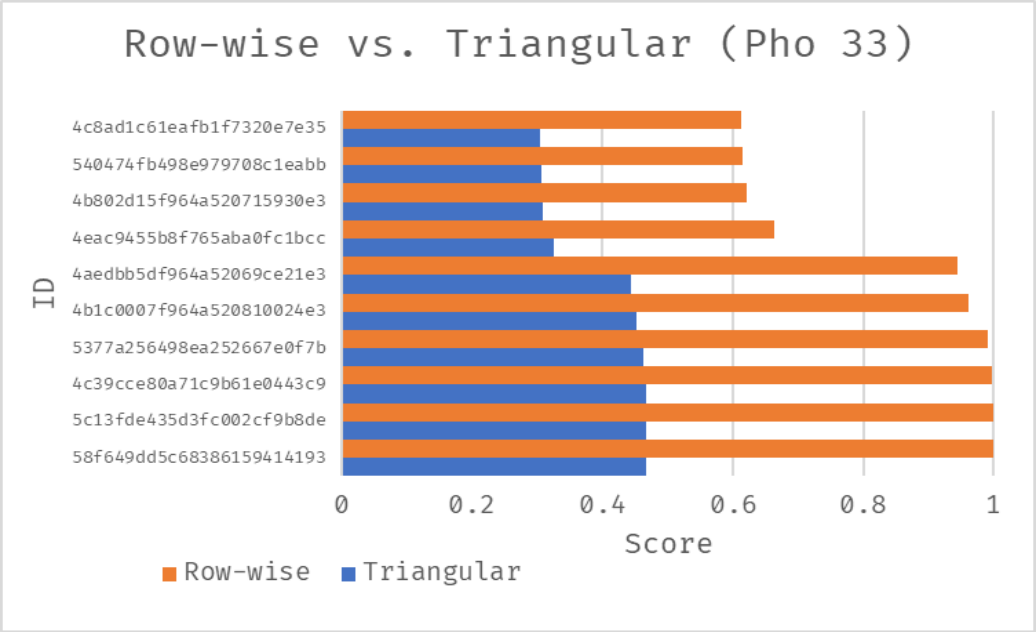

Figure 2: Effect of subarray selection on scaling for sample restaurant 4d3875bb6eef5481ede54048 (Pho 33). Shown are the 10 closest restaurants' proximity scores for both row-wise and triangular subarray selection.

In addition to the subarray selection, the type of scaling must be determined. Two scaling methods were examined: z-score standardization, and min-max scaling. Z-score standardization involves subtracting each value in the sample by the sample mean and then dividing by the sample standard deviation, yielding a distribution whose mean is centered at 0, and whose standard deviation is 1. Min-max scaling subtracts the minimum sample value and divides by the difference between the sample maximum and minimum. Unlike Z-score standardization, min-max scaling preserves the shape of the original distribution, only scaling the distribution to lie between 0 and 1.

After comparing the results of both methods on the attribute matrices, min-max scaling was chosen. Since the data in each matrix came from variables that did not necessarily follow a gaussian distribution, preserving the distribution shape for each restaurant's row in each matrix was an important advantage for min-max scaling. Another advantage of min-max scaling was that for sparse matrices, such as the cross-reference matrix, z-score standardization would cause the nonzero values to be scaled disproportionately higher. This behavior is shown in figure 2b. For a restaurant i that had users who liked both restaurant i and another restaurant j (yielding a positive $ref(i,j)$ score), standardization shifted the cross-reference score to 15, which is much larger than the scores found in the other attribute matrices. As a result, the cross-reference matrix would have a disproportionately large impact on the recommendation matrix. This is avoided when min-max scaling is used, as shown in 2a: the cross-reference score takes on the maximum value (1), giving an advantage to the restaurant that exhibited the positive $ref(i,j)$ score but not preventing other attributes from being considered to generate the top 10 restaurants.

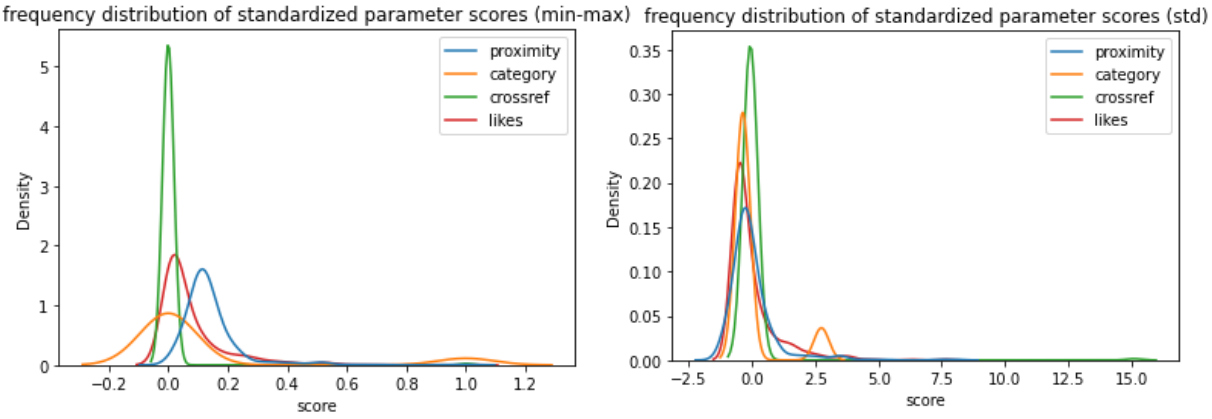

Figure 3a, 3b: Frequency distribution of attribute matrices resulting from z-score standardization (left, 3a) and min-max scaling (right, 3b)

4. Results/Discussion

Unlike a supervised learning task, there was no benchmark to compare the final recommendation matrix to. As a result, the recommendation matrix was tested by inputting different restaurants i, and monitoring the values of the 4 component attributes for the restaurants $j_1 \ldots j_{10}$ with the highest recommendation scores. The results of this analysis can be seen in Figure 4. Across the 6 restaurants shown, restaurants with high values for each attribute were represented in the restaurant lists returned to the user, as shown by the high values for each bar color that can be seen. Out of the four restaurants, the "category" attribute seems to be the most predominant, since most of the restaurants represented in the top 10 scores seem to be in the same category as the input restaurant. This makes sense – in the absence of more data on the user, and more data about each restaurant, users would most likely want restaurants in the same category as the input restaurant. Other predominating factors include proximity and venue likes; if few restaurants in a restaurant's category exist, a user would like to try popular restaurants close by.

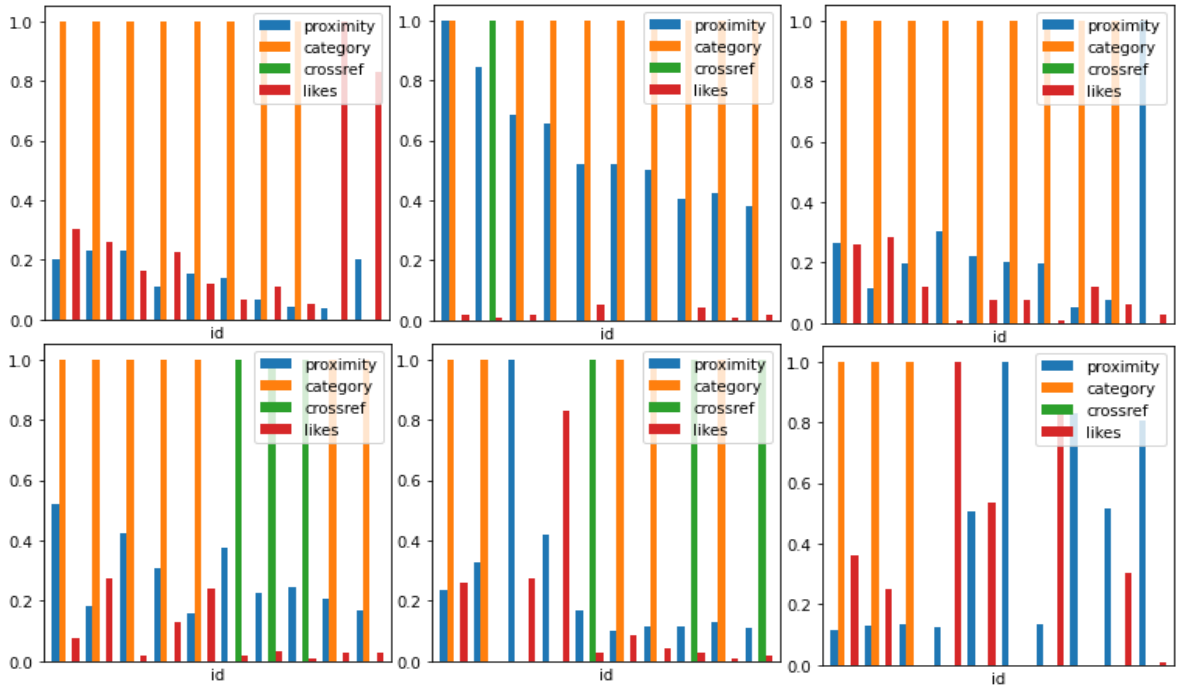

Figure 4: Plots of attributes for restaurants with 10 highest recommendation scores for 6 different input restaurants

5. Conclusion

Using the Foursquare API, an efficient recommendation system was built that factored category similarity, proximity, and popularity into its calculation. This system was based on readily available data in the Foursquare API; with more access to premium calls and different endpoints, more factors could be incorporated into the recommendation system. For instance, menu items could be examined; users may prefer restaurants with more traditional items, or restaurants with a large beverage selection. Overall, since the recommendation matrix is precomputed, results are returned near-instantaneously given a user input. The method used is extremely scalable as well and can be easily expanded to encompass larger sample sizes of restaurants given premium access to the Foursquare API.