CS416: DWM Endsem Report

Title: Hotel Recommender System from Zomato review data

Submitted by

Bhadra Giri
181CO113

Feyaz Baker
181CO119

Project Brief

In this project, we'll explore different recommender systems we can develop for hotels and eateries in Bangalore. We're using a dataset from Kaggle that stores the data from more than 50,000 restaurants all over Bangalore, as stored on Zomato. To this dataset, we plan to extract all the usable data, then process and clean the data, visualise the dataset to understand what parts can be used, and then settle on a finalised dataframe. We then settle on a list of characteristics, like cuisine, location, food type, and use these as preferences. We take those preferences to cluster restaurants. When a user presents their preferences, it's embedded into this space of preferences, and the nearest clusters are selected and presented to the user.

COLAB NOTEBOOK:
https://colab.research.google.com/drive/1gQaFM9grXS3QIEm6Oahg08objzEmWS8R?usp=sharing&authuser=1#scrollTo=tA-zpBp2dTe6
GIT REPO LINK: https://github.com/NegaMage/DWDM_zomato_recommender

Since the frontend requires files over 500MB in size, there was no way to upload the data files required for the frontend. The frontend files submitted don't have data, please check the repo for data.zip and follow instructions. The frontend code was only submitted to prove that we made a frontend, and the images attached show it works.

Status of project

The project has been completed successfully. At several points of the project, we have stored the dataframe as csv, and we have also trained a Birch clustering model. With both these, we have also made a frontend for the app in Flask, and put all the information and steps to execute the frontend, in the repo (https://github.com/NegaMage/DWDM_zomato_recommender#readme).

This report is a summary of the techniques used, features extracted, augmentations produced, and models trained.

Data preparation and cleaning

Dropped the 'rest_type' column: restaurant type was submitted by restaurants, and often listed combinations of 26 base types. Rather than analysing each value and choosing which of the

base types suited it best, we dropped the restaurant type and used listed_in(type) instead, because it had just 7 base types and no combinations existed.

Dropped ‘listed_in(city)’ because it's coarser than the ‘location’ column. Each row also stored listed_in(city) as well as location. Closer analysis showed location was much more specific than listed_in(city), so we dropped those columns.

Renamed column ‘approx_cost(for two people)’ to ‘cost’: This was just for ease of typing.
Removed ‘\5’ from the ‘rate’ column: So we can use it as float values directly.
Set all new restaurants as zero-rating: The newly added restaurants had the rating column as ‘NEW’. This was replaced by ‘0.0’.

Removed commas from the values of ‘cost’: Set ‘rate’ to the next 0.1 and ‘cost’ to the next 100.
Rounded off rate and cost values to the nearest unit, to put everything as categorical values.
Extracted all valid entries from ‘menu_item’ column: Most of the entries in the menu column was an empty list. These restaurants didn’t have their menu listed. We extracted only the valid ones.

Listed value counts of the columns: listed_in(type) and cuisines: In order to get a better understanding of these columns and perform visualizations, we listed the value counts of these columns.

Preprocessed the values in ‘menu_items’ and ‘dish_liked’ by converting data to lower-case, getting rid of all punctuation, and removing digits. We cleaned the data that was present in these columns by first splitting the string and processing each of the values present.

Data Augmentation

We used the ‘address’ column to find latitude and longitude using Google Maps Geocoding. Google Maps allows us 40,000 queries per month, for free. Since our dataset initially had 51,000+ entries, this was a strong reason for us to discard NA values instead of using some data correction methods.

We created a ‘food_items’ column by merging ‘menu_item’ and ‘dish_liked’. Some entries didn’t have their menu listed, but had their popular dishes listed. We combined the two columns to form the complete menu, and prioritise queries according to whether a dish is popular there, or just on the menu.

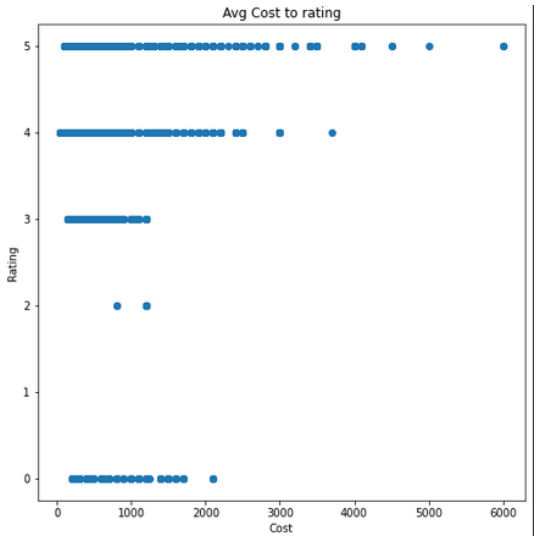
Classified the food items into their appropriate segments using NLP: The ‘food_items’ column was converted into a .csv file which was then processed using NLP techniques. We made meta-labels(topic_keywords) based on the overview of the data. Also, we defined the labels(topic_labels) before training the model. We then converted each keyword to a vector using GloVe. Following this, we converted our data into vectors as well. Finally, we computed a similarity matrix of each keyword to each topic, which gave us the output.

Took relative percentages of categories of food served in a hotel, and if over 20% of the items of a restaurant's menu were of a specific category, that hotel was considered as specialising in

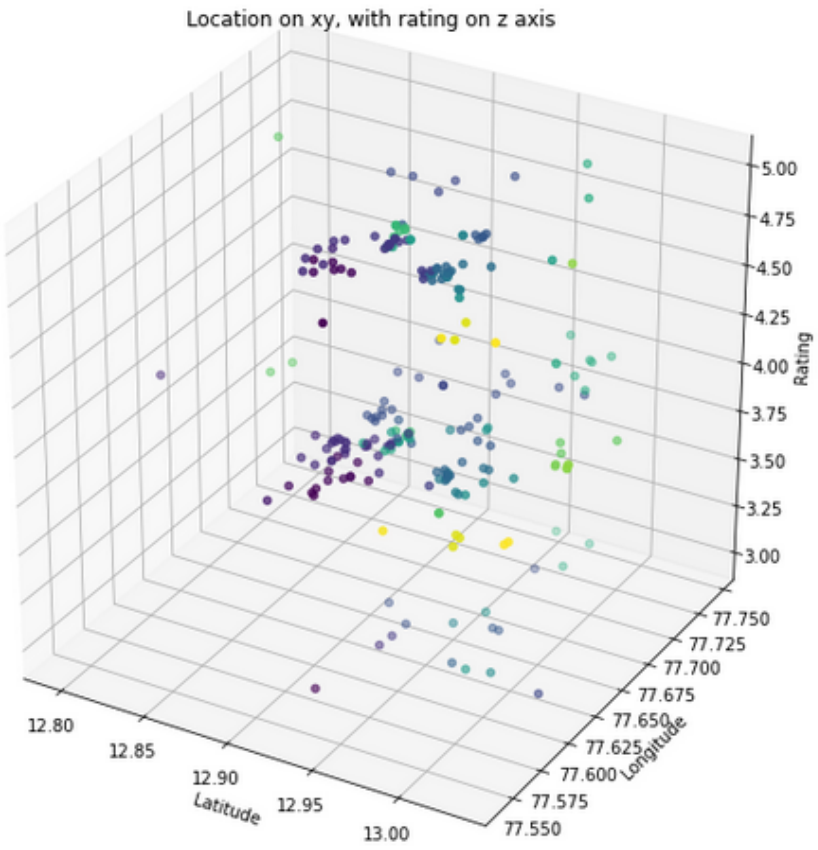
serving that type of food. This gives us more information to cluster with, as well as giving us a way to allow users to choose their food-type.

Used googlemaps to find which locations were close to each other, and used this information to make an adjacency graph. Now, when users specify their location, we can cluster accordingly and isolate those restaurants closest to a user's location.

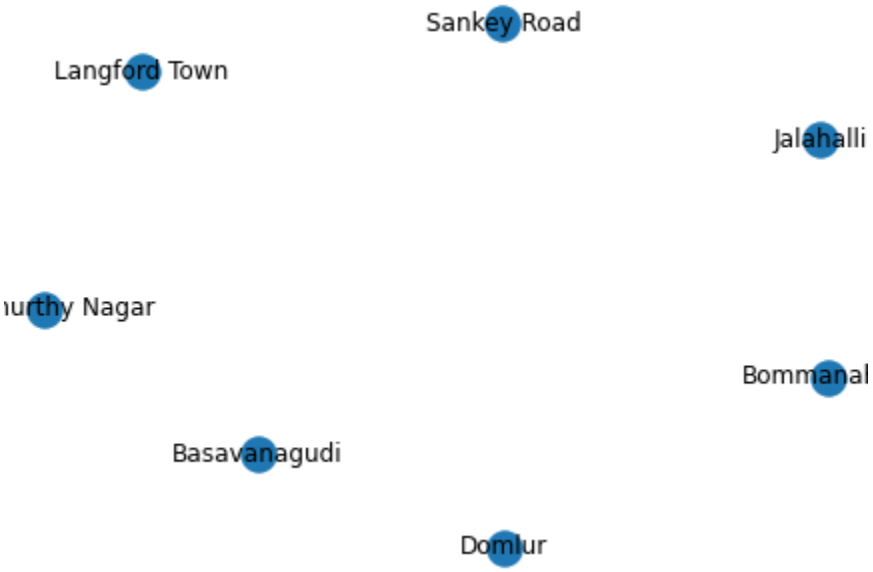
Data Visualization

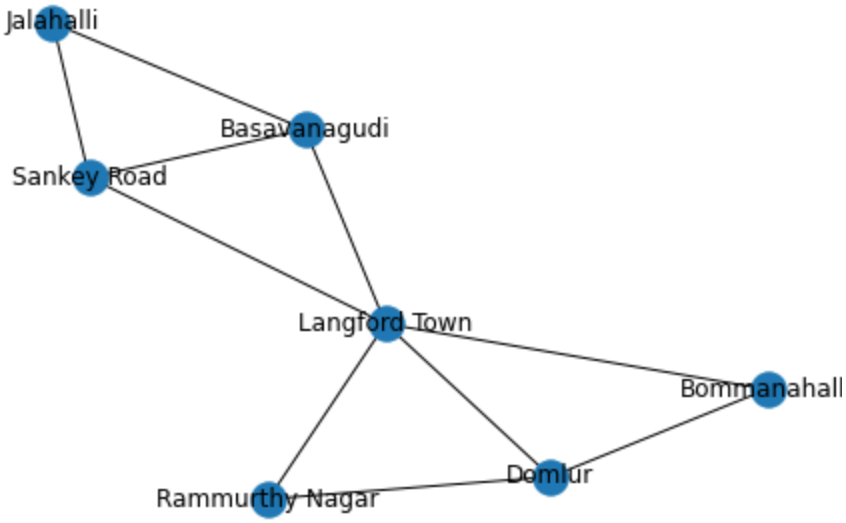


Plotted the cost for two people against rating of hotel. Most hotels of five stars are good, but expensive, as expected.

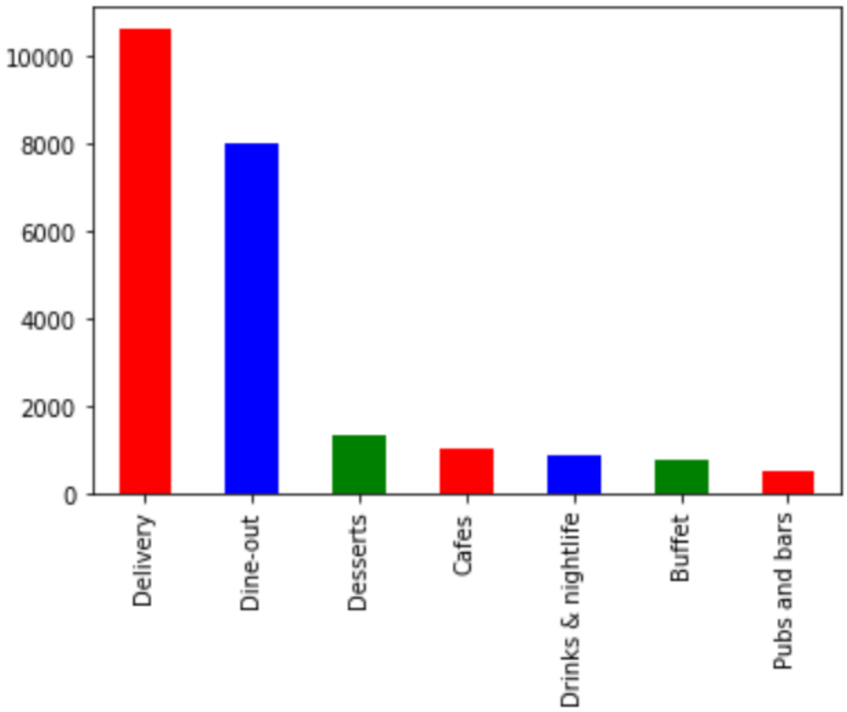


Plotted latitude and longitude of a small sampling of restaurants, with their ratings on z axis, and different colours for each location. There's a lot of points clustering together, so it's hard to analyse, but we can see that most of the high rated hotels are from the same area, so our recommender must adjust accordingly.

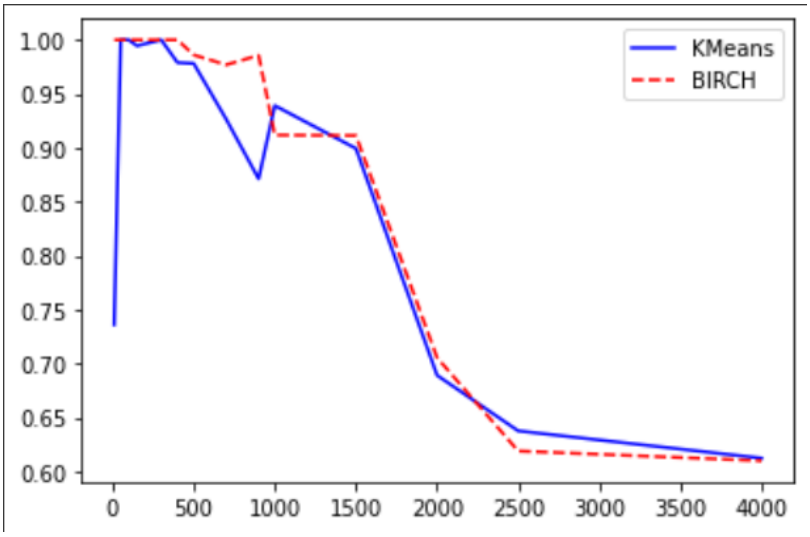




Here, we plotted 7 locations out of 88, and tried to develop the adjacency graph. In the code, we used all 88 nodes for the graph.



Number of restaurants of each listed type. A majority are listed as delivery specific, and this affects our clustering as well.



Training curves for number of clusters(x axis) vs RAND index score(y axis).

Modeling

The goal of our project is to develop a good clustering that can be used to recommend other hotels similar to a user's preferences. But since there's no ground truth clustering labels, we have improvised a solution.

We train the entire dataset on the model, and use a subset of that data as our sample data. We then train another model, using only the leftover data (training data). We then predict the clusters of the sample data according to the second model, and compare those clusters with the clusters already formed in the first model. When we repeat this experiment several times, and always obtain large RAND index scores, it implies that the clustering produced by the second model has high probability of occuring, implying that there exists an underlying structure to the data. However, since this requires us to run the models several times, we have had difficulty picking models that are computationally weak enough to train multiple times, and strong enough to capture the nuance of data.

This strategy, called pseudo-label verification, is only to be used when the data has no inherent ground truths, as it is essentially only verifying that the same model can produce the same results multiple times.

We looked at the following models:

KMeans

KMeans was the easiest model to implement. It starts by assigning random nodes (rows, here) as the centroids of clusters. It calculates the distance between each node and each centroid, and uses this distance to find the new classification of each node, and thus develops a new centroid. Over many iterations, this process reaches a stable point, which is assumed to be the new cluster centroids.

Kmeans has the problem of generating spherical decision boundaries, namely that we consider distance as outwards from the centroid, in a hyperplane, without transformations. This leads to some problems where the fit of the model is not as good as some non-spherical decision boundary models, like Spectral Clustering. Because of this, KMeans is computationally cheap, but here, it gave RAND Scores of 0.78 on average.

Birch

Balanced Iterative Reducing and Clustering using Hierarchies, or BIRCH, clusters large datasets by first making a small summary of the dataset, and clusters the summary instead of the larger dataset. The summary is kept as dense with information as possible, without increasing the strain on the algorithm.

In our trials, Birch clustering regularly gave RAND Index score 1. Hence, the rest of our project was done with this model in mind.

DBScan

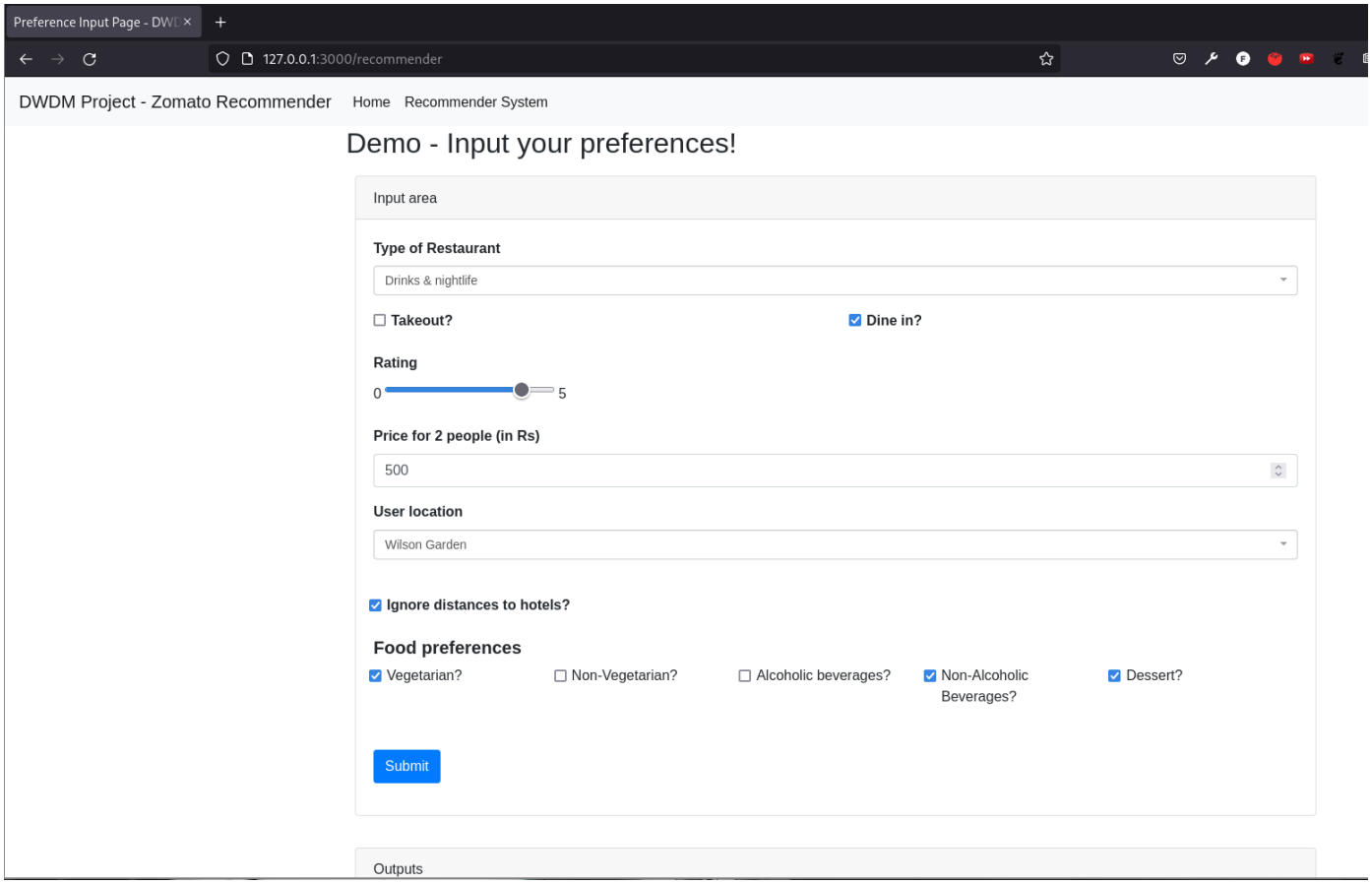
Density Based Scans are a clustering method that automatically finds the optimum number of clusters. It also predicts non-linear decision boundaries, and works by first identifying distinct groups in the data that are separated by regions of low point density. From there, it uses the density reachability and density connectivity as parameters to decide if a point belongs to a particular cluster or not. Once a whole iteration of search based on reachability and connectivity is done, the algorithm runs again, until all points are assigned a cluster.

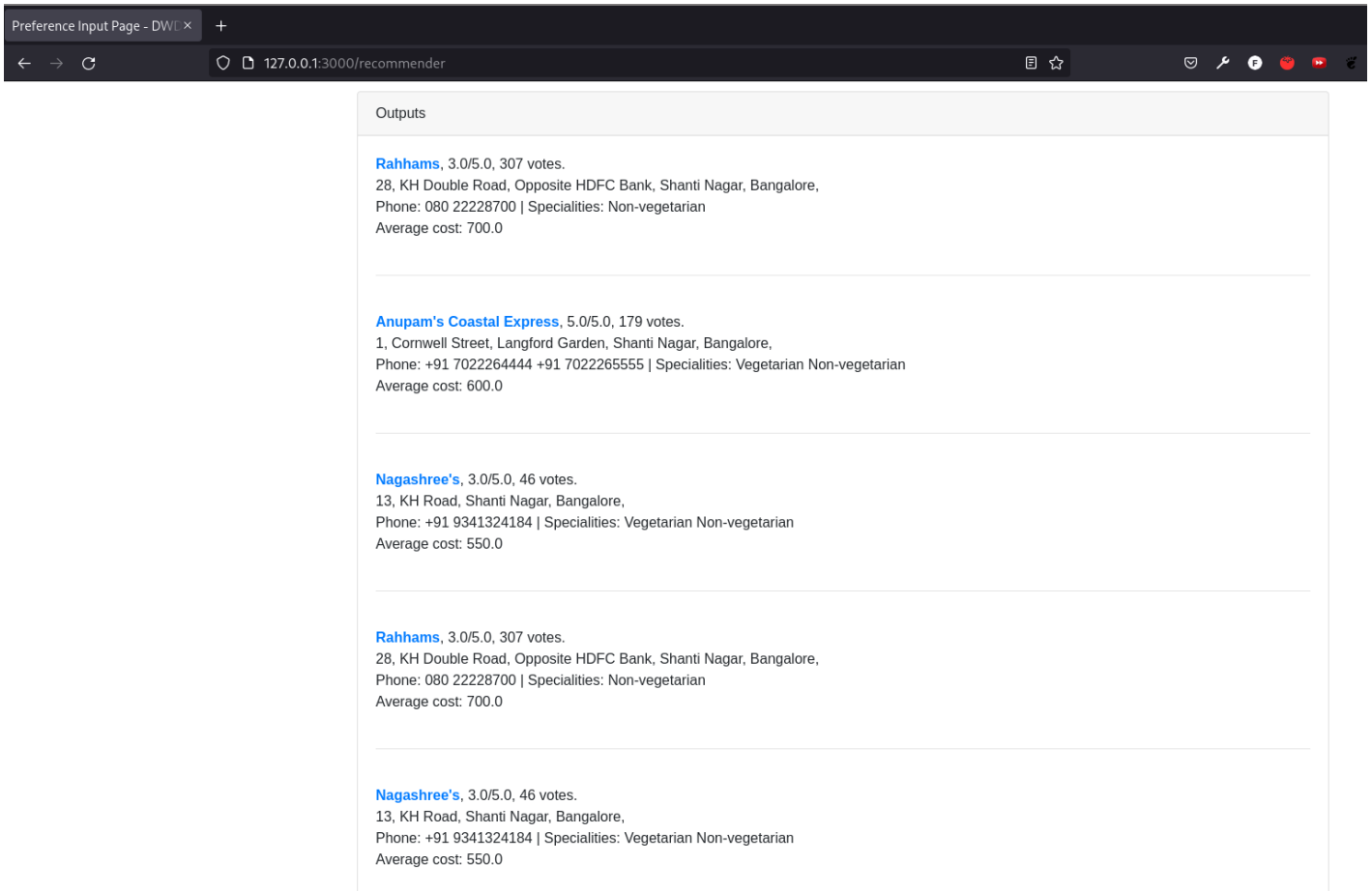
MeanShift

Meanshift iteratively clusters by shifting points towards the highest density area/point. This process repeats until each point reaches a mode, and it is then assigned a cluster. This model is computationally expensive, but it produced a RAND index score of 1.

Frontend

We wrote the frontend in Flask, and the instructions to run it are on the git repo mentioned above. Instead of running the entire notebook on demand, every time we started up the frontend, we decided to store the relevant csv files, pickled graph file, and the weights of a pretrained model, in the zip file on the repo. With these files, we can essentially replicate our results without running the colab notebook. This was done in part because the augmentation process used google maps APIs which require private credentials to run.


Preference Input Page - DWD
127.0.0.1:3000/recommender
DWDM Project - Zomato Recommender
Home
Recommender System
Demo - Input your preferences!
Input area
Type of Restaurant
Drinks & nightlife
Takeout?
Dine in?
Rating
0
5
Price for 2 people (in Rs)
500
User location
Wilson Garden
Ignore distances to hotels?
Food preferences
Vegetarian?
Non-Vegetarian?
Alcoholic beverages?
Non-Alcoholic Beverages?
Dessert?
Submit
Outputs



Frontend screens: The first image is a view of the preference input screen, the second is the outputs produced.

Future areas of work

We can extend along the following areas:

1. History based decisions - By storing the previous restaurants that a user has ordered from, and the ratings given then, we can more accurately appeal to the user's choices, and present results accordingly.

2. Property agnostic clustering: We could explore developing a system where users can preferentially opt out of searching for a specific term. For example, a user may not care about the cost while ordering, or if they're getting food delivered, they would not bother about location. Such clustering would require training multiple models, but there may be a better method for this.