Developing a Smart Restaurant Recommendation System for Dropbox and Break Through Tech

By: Maura Anish

Business Focus

During the fall of 2023, I worked alongside five other undergraduate students to build a restaurant recommendation system for Dropbox. The goal of our project was to use data about users' dining preferences to generate personalized recommendations of restaurants for each user. This was a representative problem, but recommendation systems like ours are employed by countless companies (like Netflix, Spotify, Amazon, YouTube, and more) to suggest relevant products to customers. For example, Dropbox currently uses a recommendation system to suggest files that a user may wish to work on next, based on their previous activity. Upon receiving good recommendations from a recommendation system, customers would be more likely to interact more extensively with the service and thus generate companies more revenue.

Data Preparation

The Dataset

We retrieved the data that we used for our project from Kaggle. The data was split into nine separate files: `chefmozaccepts.csv`, `chefmozcuisine.csv`, `chefmozhours4.csv`, `chefmozparking.csv`, `geoplaces2.csv`, `userpayment.csv`, `usercuisine.csv`, `userprofile.csv`, and `rating_final.csv`.

The first five data files contained information about restaurants. `chefmozaccepts.csv` listed the different payment methods that each restaurant accepted. `chefmozcuisine.csv` listed the different types of cuisine that each restaurant served. `chefmozhours4.csv` listed the hours of operation of each restaurant. `chefmozparking.csv` listed the type of parking that was available at each restaurant. `geoplaces2.csv` had more than one feature, unlike the other restaurant data files. This last file provided information such as where the restaurants were located, what their atmospheres were like, and how much they tended to cost.

The next three data files contained information about users. `userpayment.csv` listed the different payment methods that each user preferred. `usercuisine.csv` listed the different types of cuisine that each user preferred. `userprofile.csv` had more than one feature, and included information such as the users' demographic information, where the users were located, and what sort of atmosphere they liked in a restaurant.

`rating_final.csv` was different from all of the other data files, in that it contained information about users and restaurants. Each user listed in the data file had provided ratings of certain restaurants that they had previously visited. The users rated restaurants as a 0, 1, or 2 in the categories of overall quality, food quality, and service quality. In total, there were 1,161 restaurant ratings provided by users.

Across the different data files, the number of restaurants on which we had data varied widely. `chefmozaccepts.csv` had information about 615 restaurants, `chefmozcuisine.csv` had information about 769 restaurants, `chefmozhours4.csv` had information about 694 restaurants, `chefmozparking.csv` had information about 675 restaurants, and `geoplaces2.csv` only had information about 130 restaurants. Since `rating_final.csv` also only had information about 130 restaurants, these were the 130 restaurants we used in our model.

All but one of the data files containing user information had information about 138 users. (`userpayment.csv` only had information about 133 users.) Since `rating_final.csv` also had information about 138 users, these were the 138 users we included in our model. The five of these users that weren't included in `userpayment.csv` were given null values for that feature.

Exploratory Data Analysis

The 130 restaurants that we used from our data were all located in Mexico. As demonstrated in Figure 1 below, the restaurants were primarily concentrated in 3 different states – San Luis Potosi, Morelos, and Tamaulipas.

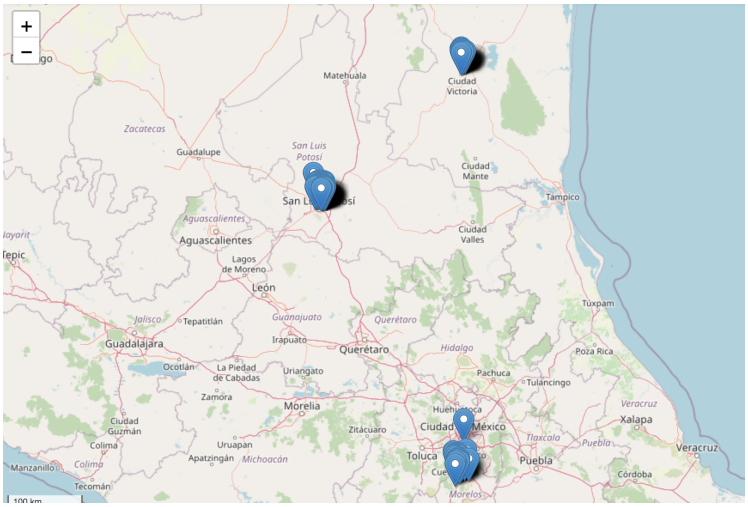

Figure 1: A map showing the restaurants' locations in Mexico.

These restaurants received a varied number of ratings from the 138 users in our data. Some of the restaurants had as few as 3 ratings, while one had as many as 36. According to Figure 2, approximately 72% of the restaurants received less than 10 ratings.

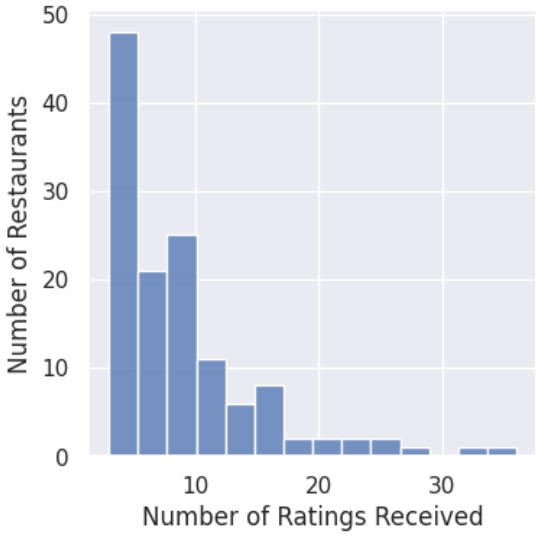

Figure 2: A histogram of the restaurants' ratings.

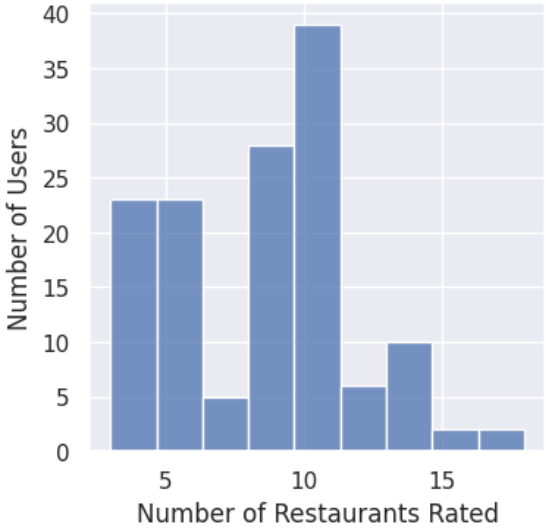

Figure 3: A histogram of the users' ratings.

Likewise, the number of restaurants that each user rated varied. Some users rated as few as 3 restaurants, while two rated as many as 18, as shown in Figure 3. Figure 4 shows that, out of the 1,161 overall ratings given by users to restaurants, nearly 250 were 0s, a little more than 400 were 1s, and almost 500 were 2s.

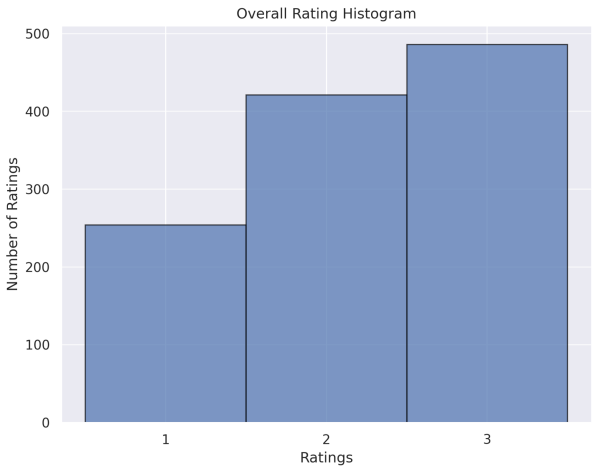

Figure 4: A histogram of the 1161 overall ratings (as 1, 2, or 3 instead of 0, 1, or 2).

Feature Engineering

After deciding that the information about payment methods, parking, and hours of operation wouldn't be necessary for our recommendations, we discarded the `chefmozaccepts.csv`, `chefmozhours4.csv`, `chefmozparking.csv`, and `userpayment.csv` data files. This left us with `chefmozcuisine.csv`, `geoplaces2.csv`, `usercuisine.csv`, `userprofile.csv`, and `rating_final.csv` data files, which we merged into one dataset using a cross-join.

We selected a total of 8 features to clean, based on what we determined to be most useful in predicting whether a user would like a restaurant. These features were: `latitude`, `longitude`, `cuisine_type`, `price`, `alcohol`, `ambience`, `dress_code`, and `smoking`. We decided to use these features because they were included in the restaurant and user data files and they contained few to no missing values.

`latitude` and `longitude` were the only numeric features in our model, and they didn't need to be cleaned before being used. The rest of the features in our model were categorical features, however, so we needed to convert their values from text into numbers before incorporating them.

With the exception of the `price` feature, we converted the other features into 0/1 indicator variables. (For `price`, we used 3 levels: 1, 2, or 3, to indicate low, medium, or high cost.) After taking the initial 59 different categories in `cuisine_type` and grouping them together into 13 broader categories, we one-hot-encoded each category to generate 13 0/1 features for the cuisine types. `alcohol`, `ambience`, `dress_code`, and `smoking` were one-hot-encoded, too.

Then, for `rating`, our label, we re-scaled it from 0, 1, or 2 to 1, 2, or 3. We only used the overall ratings in our model, since the food ratings and service ratings were inherently built into the overall ratings made by each user.

Modeling Approach

Our team took two different approaches to creating a model using the data. One subset of our team developed a content-based filtering approach using Gradient-Boosted Decision Trees (GBDTs), and the other subset developed a collaborative filtering approach using Matrix Factorization. (We initially planned to combine the two models into a hybrid model at the conclusion of our project, but we were unable to achieve this in the time provided to us.)

The content-based filtering model incorporated the cleaned categorical features and label into a `GradientBoostingClassifier` from the `scikit-learn` library in Python. That team tuned hyperparameters such as `n_estimators`, `learning_rate`, and `max_depth` of the trees in the models that they applied to their training dataset, and then they compared the results of the models on their test set.

I was a part of the collaborative filtering team, and our first step was to organize our data into a matrix. To do this, we created a 138 x 130 matrix, in which every row corresponded to a unique user and every column corresponded to a unique restaurant. The values of each (row, column) pair which corresponded to a restaurant that a user had rated were 1, 2, or 3 based on how the user had assessed the overall quantity of that restaurant. Every other (row, column) pair which corresponded to a restaurant that a user hadn't rated received a 0. This resulted in a sparse matrix (as seen in Figure 5), in which only 6.5% of its 17,940 elements were non-zero.

	A	B	C	D	E	F
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	2	0	0
7	0	0	0	2	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0

Figure 5: A 10 x 6 subset of the 138 x 130 matrix containing user-restaurant ratings.

Once the matrix factorization team had our matrix, we sought to apply singular value decomposition (SVD) to it so we would receive three matrices which, when multiplied together, would perfectly recreate our original matrix. After applying dimension reduction to these three matrices, however, we could then multiply them together to generate a matrix with the same dimensions (138 x 130) as our original matrix, but the thousands of 0 elements would be filled in with non-zero numbers. From there, within each row (for each user), we would identify the maximum element that was previously a 0, locate which column that maximum element was in, and recommend that column's restaurant to that user.

First, the matrix factorization team used the `svd` function from the `numpy` library in Python to generate our matrix of predicted ratings for each user and restaurant. When this method produced suboptimal results, we employed a different approach instead, drawing from Denise Chen's work on Medium. This approach allowed us to tune hyperparameters such as `K` (the number of latent factors), `steps` (iterations), `alpha` (learning rate), and `beta` (regularization parameter) before running a gradient descent (GD) algorithm on the matrix to generate the predicted matrix.

Key Findings

Model Results

The content-based filtering team achieved a training accuracy score of 0.84, a validation accuracy score of 0.55, and a testing accuracy score of 0.61. These results were achieved using 100 estimators, a learning rate of 0.1, and a maximum depth of 3 for their GBDTs. Their model seemed to do the best at predicting which restaurants users would rate a 1, as opposed to which they would rate a 2 or 3, but with relatively reliable information about which restaurants users wouldn't like, we would be able to avoid recommending restaurants like those to them.

The collaborative filtering team achieved an RMSE of 0.71 and a correlation of 0.50 on our test set of 232 user-restaurant ratings. These results were achieved using GD with 51 latent factors, 3000

iterations, a learning rate of 0.02, and regularization of 0.02. This was a considerable improvement over the RMSE of 0.87 and correlation of -0.015 on our test set that we received from solely applying SVD to our matrix. The improvement can also be seen in Figures 6 and 7.

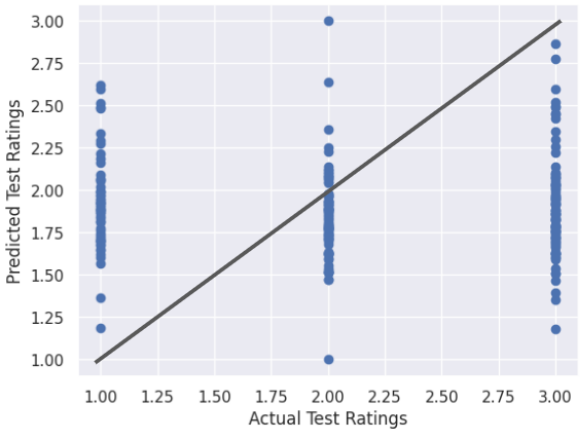

Figure 6: A scatterplot of the actual versus the predicted test ratings from the SVD, with the line y=x.

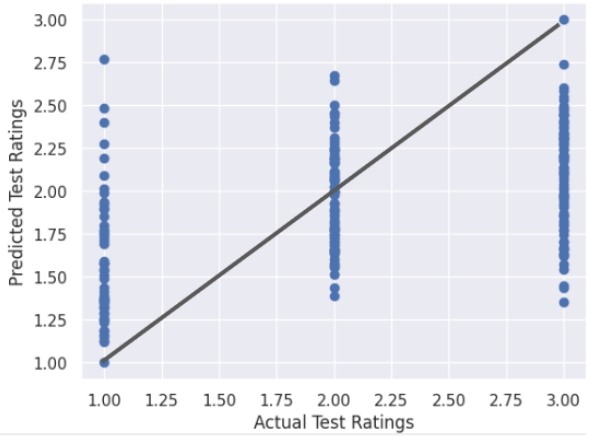

Figure 7: A scatterplot of actual versus predicted test ratings from gradient descent, with a y=x line.

In addition to the improved quantitative results it generated, the application of gradient descent to the matrix also provided better restaurant recommendations to users. Unlike the restaurants recommended to users from the SVD algorithm, the GD approach tended to recommend restaurants that had higher average overall ratings. The combination of these evaluation metrics demonstrates how effective the gradient descent approach was.

Future Work

With more time, it would have been nice to combine the results of the content-based and collaborative filtering models into one hybrid model. Since each model had outputted predicted ratings for each restaurant for each user, we could have averaged all of those ratings together for each user and recommended the restaurant with the highest average rating to each person.

Additionally, we could employ clustering on the users and/or the restaurants in our dataset to get a better sense of which users and restaurants are similar to one another. If we clustered users, we could recommend restaurants that users within each cluster had liked to users who were also in those clusters that hadn't rated those restaurants yet. If we clustered restaurants, we could recommend similar restaurants users hadn't rated yet that were in clusters with restaurants which users had rated highly to those users.

Finally, we could apply a neural network to the data, using Python packages like `TensorFlow` or `PyTorch`. A deep learning approach could be more sophisticated than the methods we applied so far, and could thus detect more subtle relationships between the users and restaurants.

Major Takeaways

Through my work on this project, I learned a lot about recommendation systems. I came to understand how content-based filtering operates on an individual user basis, while collaborative filtering uses information from other users to make recommendations. I also became aware of the cold-start problem that accompanies recommendation systems, whereupon it is difficult to make recommendations to new users who we don't have any information about yet.

Furthermore, I gained an appreciation for how difficult it can be to recommend restaurants to users when there are so many potential restaurants to suggest and so few ratings of those restaurants that users have actually contributed. We were working with a relatively small dataset of 138 users and 130 restaurants in our project, but if we had chosen to pull from the 769 restaurants that we had some data on instead, we would have had hundreds of restaurants that weren't rated by any of our users, but could still have been eligible for recommendation to them.

This project also provided me with the invaluable experience of working with a team of students across several months. At times, we struggled to find common openings in our busy schedules, but the remote weekly meetings that we conducted were quite beneficial to our team bonding and mutual support. I learned how helpful it is to stay organized by creating meeting agendas, taking notes during meetings, and sending out reminder notifications via text and email to keep everyone on the same page. Determining a set of action items for each team to complete by our next meeting helped keep everyone accountable and responsible for their work.

Overall, this project helped me develop my machine learning skills and my professional skills. I feel as though my ability to code in Python has improved, as have my presentation skills and leadership skills. I'm grateful to have had this opportunity to showcase my team's achievements.

Acknowledgements

I'd like to thank my teammates – Maame, Miraya, Sabrina, Sandy, and Vivian – for all of the time and effort they put into this project. I'd also like to thank my Challenge Advisor from Dropbox, April, for guiding us through the project and sharing some of her extensive machine learning knowledge with us. We couldn't have done this without David, our TA from MIT, who continually provided us with helpful and actionable feedback along the way.

Thank you to the Boston team of Break Through Tech AI, especially Chu and Malorie, who have made experiences like these possible for us! I'd like to thank my Career Mentor, Asala, too, for volunteering her time to share career advice with me. I already feel more confident in my ability to find a job in data science as a result of the technical and career guidance that I've received from this excellent program. I can't wait to see what the spring 2024 semester brings!