Cesar Carino
BrainStation: Data Science

Yelp Restaurant Recommender System

The goal of my capstone project, creating a Restaurant Recommender System with the use of Yelp restaurant and rating information, was to use machine learning on known reviewer restaurant ratings to then predict ratings for the set of restaurants respective reviewers are yet to visit. In other words, using a dataset of known ratings to generate another of unknown ratings. This second dataset of predicted (yet to occur) ratings would then be used to populate restaurants for the recommender system based on the system's input variables, which I will later explain. In other words, I opted to take a collaborative filtering approach in creating my recommender system, and used a matrix factorization based model to create the predicted ratings dataset for restaurants respective reviewers were yet to rate.

From the perspective of companies looking to drive user engagement or a type of conversion, examining how users interact with a service or platform can be pivotal for them. In the context of Yelp and its platform, business reviews very much influence its users on a decision to visit a place of business. This creates a great subject area to apply numerous data science techniques in a setting where a company is interested in enabling user engagement, specifically trying new content or services. Recommender systems can be created in both a static or hybrid manner, through content, collaborative and additional techniques.

The data used for my project derived from Yelp's open Dataset webpage, in which it makes a subset of JSON files specific to businesses, reviews, users, and others available. At this site, Yelp also provides documentation for the files and the information contained in them. I only used the businesses (about 153 MB in size) and reviews datasets for the purpose of my analysis, but still had to figure a way to facilitate importing the ratings found in the 'reviews' file as it was large in size (about 6.33GB).

The size of the 'businesses' file allowed the full import into a jupyter notebook to be feasible. I first gauged which geographic area in the file had the highest amount of business available to use in my analysis, and found that it was that of Phoenix, Arizona in the United States. I then examined the summary statistics of the dataset's features, and found that on average a business had about 37 total reviews. Because I wanted my recommender system to mainly populate what can be thought of as growing businesses, I decided to filter the JSON file to only include non-fast food Phoenix, AZ based restaurants with 75 or less total reviews; and were listed as being open in operations. The filtering was done through a command line processing tool helpful in parsing large JSON files called JQ.

Unfortunately, the 'businesses' file was too large in size to read it directly into my jupyter notebook. My initial approach of parsing down this file was to use the output of unique business IDs from the filtered 'businesses' file as input into a conditional statement aimed at filtering the 'reviews' file in a SQL style manner. This proved to be difficult for me because of my limited

experience in JQ and time constraints, so I ultimately decided on the simple yet effective solution in splitting the 'reviews' file into 11 sub-files to then import in an iterative manner. I then merged each sub-review file with the filtered restaurants dataset, and concatenated all of the joins into one to arrive at my final working dataset to apply the machine learning model of choice on.

However, the cleaning and exploratory data analysis was not complete as I found that the number of total reviews for a number of restaurants was inaccurately listed in the Yelp files as certain restaurants had more reviews than their stored value was listed as in that column. I worked to obtain an accurate count of total reviews respective to each restaurant and update the column accordingly. Lastly, I studied the distribution of the 'total_reviews' column once more and found that it followed that of a normal one around the mean of about 44. In addition, I used Yelp's own "new restaurants" page to study what they constituted as new or growing restaurants, and found that the page would rarely populate those with more than about 50 total reviews. I made a final decision, and opted to only include restaurants with an accurate count of 'total_reviews' no more than 65 in my final dataset to be used in my modeling.

Because I wanted to build my recommender system with information that first learned information of known ratings to then best predict those of unknown (not visited or reviewed) values, I understood that a user-item, or in this instance reviewer-restaurant, utility matrix would have almost 17 million (17,615 distinct reviewers and 948 restaurants) combinations; thus better suiting the application of the FunkSVD model to create the predictions. Matrix Factorization models look to leverage hidden factors, called latent variables or embeddings, of both users and items that may impact a rating value in question. The model treats the problem as an optimization one to find the values behind the latent variables that drive assessed ratings. One drawback of the hidden variables is that we lose the ability to infer on what exactly they represent.

The model was first fit to a subset of the filtered dataset containing restaurant attributes and reviews information, which **only** had the unique reviewers, restaurants, and assessed ratings. After fitting the model to the known ratings data and its respective scoring scale of one through five stars, I leveraged the GridSearchCV function from the surprise package to find the optimal parameter values for a passed model and evaluation metric based on a set of parameter values of one's choice. The parameters I optimized for include:

- "n_factors": number of latent variables or embeddings desired to learn.
- "n_epochs": number of iterations the algorithm should run for.
- "lr_all": represents the learning rate for all parameters, and affects how quickly our model can converge to a local minima (aka arrive at the best accuracy).

The evaluation metric I chose for the GridSearch to optimize the parameters for was the Fraction of Concordant Pairs (FCP) metric. The FCP metric is composed of concordant pairs that contain pairs of both known (true) and predicted ratings, which work to imply that if the known rating of Restaurant A is > the known rating of Restaurant B, then the predicted rating of Restaurant A will also be greater than that of Restaurant B. FCP is the ratio of concordant rating pairs to all rating pairs, such that its score ranges between 1.0 (best) and 0.0 (worst).

GridSerach found the parameter values that optimized the FCP metric to a result of about 0.5455 (i.e. about 55%). Though the value could have been better, the result is understandable given the size of the training set. With the optimal parameter values now known, I split the full subset of known ratings into a train and test set to first train the model on the former and create predictions using the latter set on known ratings. Though predicting known ratings was not my target goal, it enabled me to then use the “build_anti_testset” function from the surprise package to create a dataset of predicted ratings for reviewer-restaurant rating combinations yet to occur. The model was able to predict ratings within one ranking point (1-5) from the actual rating of the test set about 25% of the time, which again was an acceptable performance given the size of the testset.

With my predicted ratings for unknown (yet to occur) reviewer-restaurant combinations dataset, I felt comfortable with the model’s results to create a recommender system through a function that:

- Takes the input of a desired restaurant food category (i.e. Fusion, Bagels, Mexican, etc.).
- Requires a tolerance or cap the user is willing to accept between a restaurant's average rating and the predicted score a reviewer (with similar input values) may have rated the restaurant they are yet to visit.
- Populates the top 15, if available, restaurants.

Though the recommender system can be used by anyone, actual business applications of this analysis could relate to edging a set of reviewers to visit other restaurants (or places of business) which Yelp believes they may be content in visiting. In other words, recommender systems are often valuable when they can be assessed in a conversion framework. In the context of Yelp, a conversion can possibly be viewed as getting its active reviewers or users to continue to visit and rate other businesses on its platform.

In the context of my project, the next step I would like to complete is to create a WebApp for my recommender system for ease-of-use.