CS229 Final Project: Milestone
Predicting Yelp User's Rating Based on Previous Reviews

Yue Li (yulelee) Haomiao Song (hmsong)

I. Introduction

Currently, the interactions between the user and the Yelp application is mainly initiated by the user searching for some keywords, and then go through a list of the matches, potentially ranked by ratings, number of reviews, etc. Since the personalized recommendations are also crucial to better user experience, we want to build a model to recommend new places to users. In this project, we implemented several existing algorithms for recommendation systems, and developed 2 new algorithms, namely, review-based and text-based recommendation models. In this paper, we compares the results of those models, and also, try to improve the overall accuracy by building a hybrid system.

II. Dataset

The dataset is from Yelp Dataset Challenge [**?**], which contains about 3.7 million reviews from 687,000 users for 86,000 restaurants. Within the dataset, we can find basic information of the restaurants (opening hours, address, categories, average rating, parking, alcohol, etc), users (name, review counts, average rating, etc), and reviews (which user, wrote to which restaurant, the content, and the corresponding rating).

This dataset includes information about restaurants on 10 cities all across the world. We've picked the city of Las Vegas, which has the most restaurants and users among the whole dataset. Also, in order to make our results more consistent and speed up the computation, we decide to only use the restaurants and users with more than 20 reviews associated with them. As a result, there are 6058 users and 6978 restaurants left, with a total of 259143 reviews.

III. Evaluation Metric

Generally speaking, the output of a recommendation system should be a list products (in this case, restaurants), however, it would be hard to validate the performance if the output is simply a list of items. Thus, in the paper, we try to build models that takes a restaurant and an user as input, and output the prediction of the rating that the user might give to this restaurant. For the sake of recommendation, we can just recommend the restaurants with the highest ratings.

To evaluate the accuracy of the predicted ratings, we choose to use one of the most popular metrics for the recommendation systems[**?**], the root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum (r_{ub} - \hat{r}_{ub})^2}$$

where $\hat{r}_{ub}$ is the predicted rating of user u to the restaurant b, and N is the total number of ratings in the testing set.

IV. Method

A. Baseline

For the baseline recommendation system, we compute the average ratings for each user and restaurant. For an user u, the average rating $\bar{r}^U_u$ is the average rating he/she has given. For a restaurant b, $\bar{r}^B_b$ is the average of all the ratings it has received. Here we use the superscript U to denote something related to an user, and superscript B to denote something related with a restaurant/business, this convention is used throughout this paper. When predicting the rating of user u to the restaurant b, the prediction is computed by an average over the average rating of u and b, which is $\hat{r}_{ub} = (\bar{r}^U_u + \bar{r}^B_b)/2$.

B. Memory-based Collaborative Filtering

For the memory-based collaborative filtering, we construct the user-item matrix $X \in \mathbb{R}^{U\times B}$, where U is the total number of users and B is the total number of restaurants. Each element in this matrix X_{ub} is the rating user u gave to the restaurant b. If the user didn't give a rating to a restaurant, the corresponding element in the matrix would be the placeholder 0 (since a actual Yelp rating is between 1 to 5).

We name each row of X to be the user characteristic vector of the user, denoted by X^U_u for user u, and each column in X to be the item characteristic vector X^B_b, for the restaurant b. With those characteristic vectors, we can compute the similarities between two users or two restaurants, here we define the cosine similarity (for user i and j):

$$S^U_{ij} = \text{sim}(X^U_i, X^U_j) = \frac{(X^U_i)^\top X^U_j}{||X^U_i||_2||X^U_j||_2}$$

By computing the similarities for all of the user-user and restaurant-restaurant pairs, we can construct the similarity matrix for users: $S^U \in \mathbb{R}^{U\times U}$, and for restaurants: $S^B \in \mathbb{R}^{B\times B}$.

For the memory-based collaborative filtering, we have two ways to make predictions. The first one is the user-based prediction, where:

$$\hat{r}_{ub} = \bar{r}_u + \frac{\sum_i S^U_{ui}(X_{ib} - \bar{r}^U_i)}{\sum_i |S^U_{ui}|}$$

In other words, to predict $\hat{r}_{ub}$, we compute the weighted average of the ratings from all the users to the restaurant b, with the similarities between user u and other users as the weights. Here we're subtracting the mean ratings of each user to eliminate the bias that some user tends to always give higher ratings (while some users might always give lower ratings).

The second way to make prediction is item-based:

$$\hat{r}_{ub} = \frac{\sum_i S^B_{bi} X_{ui}}{\sum_i |S^B_{bi}|}$$

here we don't need to correct the ratings since the ratings X_{ui} are all given by the user u.

C. *Model-based Collaborative Filtering*

Model-based Collaborative Filtering is based on matrix factorization, which is the process of transforming the user-item matrix X into the multiplication of two lower-rank matrices. One widely-used way of matrix factorization for recommendation system is singular value decomposition[**?**], which can be expressed as $X = USV^\top$, where U and V are orthogonal matrices, and S is diagonal matrix with the singular values of X on the diagonal. By the factorization, we extract the useful information from X into U, S and V, and then use their product as the estimates for the missing values in X.

D. *Collaborative Filtering by Gradient Descent*

We can also factorize X by stochastic gradient descent. In this method, the objective is to transform X into the product of $P \in \mathbb{R}^{U \times L}$ and $Q \in \mathbb{R}^{L \times B}$ where L is the number of latent features we want to extract. The loss function therefore could be written as:

$$L = \sum_u \sum_b \mathbf{I}(X_{ub} \neq 0)(X_{ub} - P_u^\top Q_b)^2 + \lambda(||P||_2^2 + ||Q||_2^2)$$

Here the indicator function $\mathbf{I}(X_{ub} \neq 0)$ means that the loss is only taken for the known values in the matrix X. If product of P and Q can approximate the known values in X well, then the other values computed by PQ can be used as the estimations for the missing values in X. Take the gradient for the loss of one training example (u, b), we derive the update rule for the stochastic gradient descent:

$$Q_b := Q_b - \eta\left((X_{ub} - P_u^\top Q_b)P_u + \lambda Q_b\right)$$
$$P_u^\top := P_u^\top - \eta\left((X_{ub} - P_u^\top Q_b)Q_b + \lambda P_u^\top\right)$$

E. *Review-based Recommendation*

Here we present another way of making recommendations, we call it review-based recommendation. This method is inspired by the content-based recommendation, where a list of features for each item is extracted, and being compared with the user's preference. For example, for a movie recommendation system, each movie would have several features like genres, years, actors, and each user would have his/her favorite genres and actors being listed as the user preference. In our dataset, we also have a few features listed for each restaurants, for example "Free Wi-Fi" and "Accepts Credit Cards", but those features are fairly non-representative and a vast amount of restaurants do not have any features associated with them.

As a result, we turn to the reviews written by the users. No matter what is written in those reviews, it should be a solid assumption that those reviews are some kind of useful description for a certain restaurant. In this model, we try to extract useful things from the reviews.

Firstly, for each restaurant, we concatenate all the reviews it has received into a long sentence, and then using the bag-of-words technique to vectorize and extract one feature vector for each restaurant, with each element in the vector to represent the count of that one word (basic stop words and stemming techniques are used). The resulting feature matrix $F \in \mathbb{R}^{B \times W}$, where W is the total number of words. For each element in F, we compute the tf-idf value of term t in the document (sentence) d by the following formular:

$$F_{dt} := F_{dt}\left(\log \frac{1 + B}{1 + \sum_b \mathbf{I}(F_{bt} > 0)} + 1\right)$$

after which each row of the matrix F is normalized to have a Euclidean norm of 1. Next, for each user, we also need to construct his/her preference vector:

$$P = XF \in \mathbb{R}^{U \times W}$$

the preference vector for a given user is just a linear combination of the feature vector of restaurants he/she has rated before, with the corresponding ratings as the weights. After normalizing P, we can compute the similarities between all of the feature-preference pairs by one matrix multiplication, $S = PF^\top \in \mathbb{R}^{U \times B}$. The element S_{ub} would be the similarity between restaurant b's features and the user u's preference.

For each user, we can construct a list of all restaurants, sorted by the similarities. This is already enough for the purpose of recommendation, however, for the sake of validation and also to compare the performance with other models, we need a way of transforming the rankings of restaurants into real-valued ratings. To do this, assuming our rankings are correct, for a given user, what we have are several ratings for a few restaurants from the training set, if we plot the rankings against the known ratings on a 2-D figure, we should see an decreasing trend from those points. For example, the restaurant ranked at 100 is rated 5, and the restaurant ranked at 300 is rated 3, then a reasonable prediction for the restaurant ranked at 200 could be somewhere within the range between 5 and 3, probably 4. As a result, the problem is now transformed into a regression problem, therefore various regression models can be used. In this case, we choose to use locally weighted linear regression, which is capable of dealing with the non-linearity within the data.

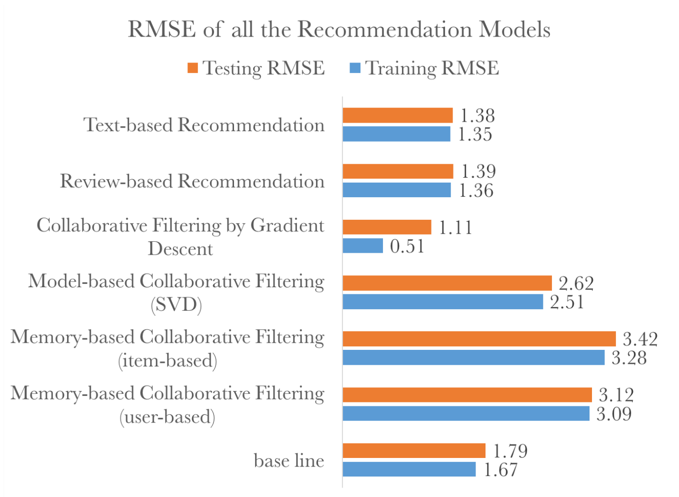
F. *Text-based Recommendation*

The text-based recommendation is very similar to the review-based recommendation system described in the last section. The only difference is, instead of constructing the preference vector for each user using the history ratings, we now directly extract the preference vector from all of the reviews the user has written. In other words, we first concatenate all of the reviews written by a certain user, and then directly extract the preference vector from it. The preference vector would have the same dimension as the feature vector for restaurants.

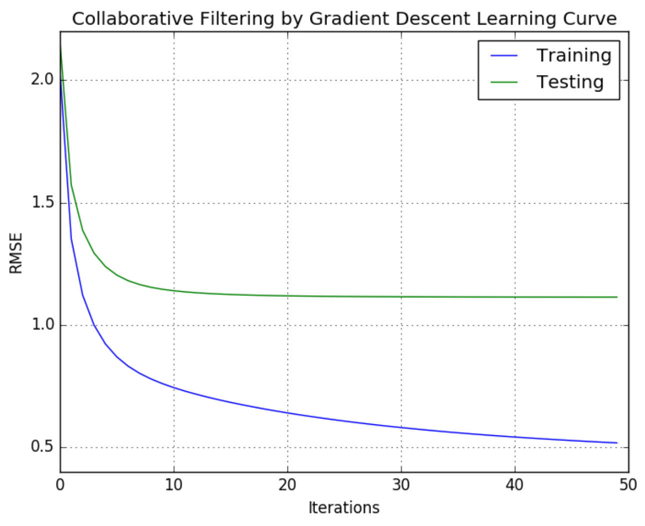
For example, if the word “price” is very frequent both in the reviews for a restaurant, and also in the reviews of an user. The intuition behind this could be that everybody is talking about the price of this restaurant, and this user happens to care about the price a lot, therefore they can be somehow matched up. At this point, we really cannot tell whether this user would like this restaurant (it could be that everybody is actually complaining about the price), but still, there is some relationship between this user and this restaurant. At the regression step, we still use locally weighted linear regression.

V. Results

30% of the data is randomly chosen to be the testing set. The training and testing RMSE for all of the algorithm described in the last section are listed in this figure.



Particularly, this is the learning curve for the gradient descent used in the collaborative filtering by gradient descent model.



VI. Discussion

Among the models we’ve built, the current best one is the collaborative filtering by gradient descent, which has a testing RMSE of 1.11, while other collaborative filtering methods perform worse than the baseline. The reason for this might be that most of the users keep giving similar ratings, for example, one users might give 5 stars to the restaurants he/she likes, and 4 stars to the restaurants he/she doesn’t like, and seldom gives ratings less than 3 stars. As a result, the average could be a fairly good prediction.

The text-based and review-based models are both better than the baseline. Review-based model is slightly better than the text-based model. However, the advantage of the text-based model is that it does not depend on the historical ratings of the users. Though in this case, we’ve used historical reviews from the users to construct the preference vectors, but we can also use any other text records for the users. For example, now the users can use their Facebook account to sign up for Yelp, the preference vectors can be constructed using the text from their Facebook status, or any other sources. Also, one of the main challenges for the recommendation systems is the cold start problem[**?**], which happens when a new user enters the system. Since this text-based model do not depend on historical ratings (only need to gather some text about the user). Thus, to some extent, it can relieve the cold start problem.

Since this is the milestone report, their are still parts of the project not finished. For the next few weeks, there is still one method we want to try out: using the alternating least squares algorithm for the collaborative filtering, which is another method to factorize the user-item matrix. Also, we’re still thinking about how to combine the existing systems into a hybrid model. A simple linear combination is definitely worth a try, but maybe there is other ways.

References

[1] Yelp Dataset Challenge www.yelp.com/dataset_challenge
[2] Joost de Wit, *Evaluating Recommender Systems*, Delft: 1980.
[3] Sarwar, Badrul, et al. *Incremental singular value decomposition algorithms for highly scalable recommender systems*, Fifth International Conference on Computer and Information Science. 2002.
[4] Su, Xiaoyuan, and Taghi M. Khoshgoftaar, *A survey of collaborative filtering techniques*, Advances in artificial intelligence 2009 (2009): 4.