CS229 Final Project
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 versions of collaborative filtering and developed two new algorithms, namely, review-based and text-based recommendation models to predict the rating from a user to a restaurant. The input is the historical interactions between users and restaurants, including the textual reviews and numerical ratings, and the output is the predicted ratings. In this paper, we compare the results of multiple models, and try to improve the overall accuracy by building a hybrid system.

II. Related Work

Recommendation system is a topic both of academic as well as commercial interest, and is well studied in the area of data mining and machine learning. Currently, recommendation systems could be broadly classified into three main categories: collaborative filtering (CF), content-based filtering (CB), and the hybrid system of the previous two [1].

Collaborative filtering makes recommendations based on the previous ratings from users to items. The general idea is that the ratings for similar items from a given user should be similar, and the ratings to one item by similar users should also be similar [2]. One of the most famous examples of collaborative filtering is the recommendation system of Amazon.com, which utilizes the item-item similarities [3]. Though collaborative filtering is one of the most widely used recommender systems [4]. The major challenges of this method including the cold start problem, scalability, and sparsity, mainly because the number of items in modern application is extremely large, and each user could have only rated a tiny fraction of it. Also, when a user is newly entered the system, there is no previous ratings to be readily used [5].

Content-based filtering algorithms are based on the similarities between a set of predefined features for items and a profile of the user preferences [6]. One example for this algorithm is the music recommendation system of Pandora, within which a database of user preferences is built, and the songs are tagged using more than 400 musical attributes [7]. The content-based filtering algorithms performs well when the system is small and structural. However, as the size and complexity of the system grow, feature extraction could become either too labor intensive or error prone [8].

Lastly, in order to improve precision, hybrid recommendation systems bridging collaborative filtering and content-based filtering are developed. The hybrid system could be a simple combination of the results independently predicted by collaborative and content-based filtering, or be one unifying model that fully integrates the two algorithms [9]. At the research front, the state-of-the-art hybrid system combines collaborative and content-based filtering via probabilistic latent semantic analysis [10], and could be further augmented by knowledge-based techniques [11]. While in production, the hybrid recommendation system of Netflix recommends movies that is similar to other movies the user liked, as well as the movies that share features correlated with the user's preferences [12].

III. Dataset

The dataset is from Yelp Dataset Challenge [13], which contains about 3.7 million reviews from 687,000 users for 86,000 restaurants, among which, only the reviews would be used in this project. The raw data has been preprocessed and only the following information is preserved for each review: the user ID, the restaurant ID, the numerical rating (integer values ranging form 1 to 5), and the original text of the review. One concrete example of the preprocessed data looks like this: {"user_id": 554, "restaurant_id": 3256, "stars": 4, "text": "Great local spot. I used to come here every Saturday after Japanese school with my mom."}.

This dataset includes information about restaurants on 10 cities all across the world. We've picked the city of Las Vegas, since it 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. The final dataset contains 259143 reviews (from 6058 users to 6978 restaurants). 25% of the data is randomly picked out to be the testing set. There are 62958 reviews in the testing set, and 196185 in the training set. Since we have multiple models built for this project, the corresponding feature extraction would be described in detail when introducing the models.

IV. 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 this paper, we try to build models that output the prediction of the rating from a user to a restaurant. For the purpose of recommendation, we can just recommend the restaurants with the highest predicted ratings.

To evaluate the accuracy of the predictions, we choose to use one of the most popular metrics for the recommendation systems[14], the root mean squared error (RMSE), which is also the metric used in the famous Netflix grand prize [15]:

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

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.

V. Methods

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

$$\hat{r}_{ub} = \frac{\bar{r}_u^U + \bar{r}_b^B}{2}$$

By this average, we hope to encapsulate both the rating pattern of the user, and the overall reputation of the restaurant.

B. Memory-based Collaborative Filtering

For the memory-based collaborative filtering, we want to extract the similarities between users, and between restaurants. Only the historical ratings would be used in this model. 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 in the training set. If there is no such rating, the corresponding element in the matrix would be the placeholder 0, since a actual Yelp rating is at least 1.

We name each row of X to be the user characteristic vector, 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 the characteristic vectors, we can compute the cosine similarities between two users or two restaurants. Specifically, the similarity between user i and j is computed by:

$$S_{ij}^U = \text{sim}(X_i^U, X_j^U) = \frac{(X_i^U)^\top X_j^U}{||X_i^U||_2||X_j^U||_2}$$

and the similarity between the restaurant i and j is:

$$S_{ij}^B = \text{sim}(X_i^B, X_j^B) = \frac{(X_i^B)^\top X_j^B}{||X_i^B||_2||X_j^B||_2}$$

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 is based on the similarities between users:

$$\hat{r}_{ub} = \bar{r}_u + \frac{\sum_{i:X_{ib}\neq 0} S_{ui}^U (X_{ib} - \bar{r}_i^U)}{\sum_{i:X_{ib}\neq 0} |S_{ui}^U|}$$

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 term $(X_{ib} - \bar{r}_i^U)$ could be viewed as the extent to which the user i likes the restaurant b.

The second way to make prediction is based on the item-item similarities:

$$\hat{r}_{ub} = \frac{\sum_{i:X_{ui}\neq 0} S_{bi}^B X_{ui}}{\sum_{i:X_{ui}\neq 0} |S_{bi}^B|}$$

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

In this paper, we implement one model-based collaborative Filtering based on matrix factorization. One widely-used way of matrix factorization for recommendation system is singular value decomposition[16], which can be expressed as $X = USV^\top$, where U and V are orthogonal matrices, and S is a 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.

In order to predict ratings. We first fill in the missing values in X by the baseline estimates, then factorize X by singular value decomposition. We pick the largest k singular values and use the product of reduced U, S, and $V^\top$ to estimate the missing values in X.

D. Collaborative Filtering by Gradient Descent

Instead of using SVD. We can also factorize X by stochastic gradient descent. In this method, the objective is to transform X into the product of two arbitrary matrices $P \in \mathbb{R}^{U\times L}$ 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:

$$L = \sum_{(u,b)\in\mathcal{D}_{\text{train}}} (X_{ub} - P_u^\top Q_b)^2 + \lambda(||P||_2^2 + ||Q||_2^2)$$

where X_{ub} is an actual rating in the training set, and $P_u^\top Q_b$ is the prediction for that rating if we multiply P and Q. The intuition is that if the 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 estimates for the missing values in X. To minimize the loss, we can take the gradient of P and

Q for one training example, and 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 (RB)

We present another way of making recommendations, we call it review-based recommendation. The previous methods only use the ratings as features. From now on, we are going to use the textual reviews. This method is inspired by the content-based recommendation, where features for each item is extracted, and being compared with the user's preference.

To construct the feature vector for each restaurant, we first concatenate all the reviews it has received into a long sentence. The sentence is then vectorized via the bag-of-words technique (unigrams). Each element in the resulting vector represents the count of that word (standard stop words and stemming techniques are used). We then construct the feature matrix $F \in \mathbb{R}^{B \times W}$ (W is the total number of words) containing the feature vectors for all the restaurants, each in a row. To get a better representation of the relevance between words and document (restaurant), for each element in F, we compute the tf-idf (term frequency?inverse document frequency) value of term t in the document(restaurant) d by the following formula:

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

after which each row of the matrix F is then normalized to have a Euclidean norm of 1. Each row of F is now the final feature vector for each restaurant. For the users, we construct the preference matrix:

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

Each row of P is the preference vector for a user, which is a linear combination of the feature vectors of restaurants he/she has rated before, with the corresponding ratings as the coefficients. After normalizing P, we can compute the similarities between all of the feature-preference pairs by one matrix multiplication:

$$S = PF^\top \quad R \in \mathbb{R}^{U \times B}$$

The element S_{ub} would be the similarity between restaurant b's features and the user u's preference.

The outcome of the similarity matrix S is that now we can rank all the restaurants for each user by the similarities. For the purpose of recommendation, we can just recommend the restaurants being ranked at the top. However, we still want to validate the recommendations, as well as compare the performance with other models, therefore, we need a way of transforming the rankings into real-valued ratings.

To do this transformation, we can pick any regression learning algorithm, and do a regression between the predicted rankings and the actual ratings in the training set. For example, the restaurant ranked at 100 is rated 5 in the training set, and another restaurant ranked at 300 is rated 3, then a reasonable rating prediction for the restaurant ranked at 200 could be somewhere within the range between 5 and 3, probably 4. In our implementation, we used the locally weighted linear regression algorithm, which is more flexible compared with ordinary linear regression, and is capable of extracting the non-linearity from the data.

F. Text-based Recommendation (TB)

The text-based recommendation is very similar to the review-based recommendation system described in the previous section. The only difference is, instead of constructing the preference vector for each user using the history ratings, we now directly construct 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 use the same technique as constructing the feature vectors for restaurants. The preference vectors of users would have the same dimension as the feature vector of restaurants, since we are always using one set of features (words). Just like the review-based recommendation, we could then compute the similarities between users and restaurants, and ranking restaurants for each user.

The intuition for this method is that we think the reviews could be descriptive for a restaurant, as well as for a user. For example, the word "price" is very frequent both in the reviews for a restaurant, and also in the reviews written by a user. The reason 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.

G. Hybrid Model (HM)

In the hybrid model, we combine the predictions from all of the previous models by a linear combination (with intercept). Some of the previous models use the ratings, while others use the reviews; some use the similarities between users, while others use the similarities between restaurants. We think that each of the models can grasp some part of the data and the underlying relationship. It would be worth trying to combine them, and see if we can make more accurate predictions.

VI. Experiments

Among all of the models, only two of them has hyper-parameters, namely, collaborative filtering by SGD and the model-based CF by SVD. When tuning hyper-parameters, 20% of training set is separated as the validation set. After the optimal hyper-parameters have been found, the model is retrained on the whole training set. Next, we'll discuss the details of hyper-parameters for those two models.

A. Collaborative Filtering by Stochastic Gradient Descent

The collaborative filtering by SGD method has three hyper-parameters, the learning rate η, regularization coefficient λ, and L the number of latent features to include. We use the alternating optimization strategy (coordinate descent) to tune those hyper-parameters. Firstly, we choose the initial values of those three parameters based on experience. Fix two of them and try to optimize the third one. For example, figure 1 shows that when λ and η are fixed, the effect of changing L:

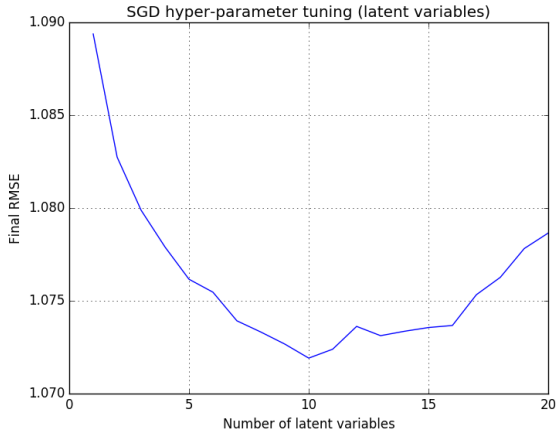

Fig. 1. Tuning the number of latent features for the CF-SGD

It's clear that the optimal value for L is 10 under the current situation. After this we can update L to 10 and switch to optimize another parameter. We keep optimizing alternating parameter for a certain amount of iterations or until convergence. The final set of optimal hyper-parameters are $L = 10$, $\lambda = 0.18$ and $\eta = 0.11$. There is also another strategy of setting η to $\frac{1}{\sqrt{t}}$, where t is the current number of iteration. Figure 2 is the learning curve using the optimal set of hyper-parameters. Since we only need a small number of iterations, and don't have convergence problem, therefore for simplicity, we keep η as a constant.

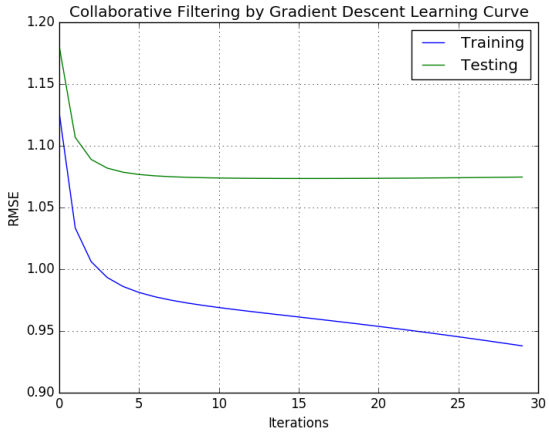

Fig. 2. Learning curve for the CF-SGD method

B. Model-based Collaborative Filtering by SVD

Only one parameter needs to be tuned for the singular value decomposition, which is the number of singular values we want to keep k. For SVD, higher k usually means higher precision, since more information could be preserved from the original matrix. We have experimented different values of k, the testing error is stable and seems not depending on k. We choose $k = 20$ as the final value to save running time, since larger k means a more expensive matrix decomposition.

VII. Results and Discussion

The performance of all of the models are listed below:

Model	Training RMSE	Testing RMSE
Baseline	1.532	1.587
Memory-based CF	1.099 / 1.132	1.107 / 1.133
Model-based CF	0.061	1.128
CF-SGD	0.509	1.072
RB	1.255	1.380
TB	1.467	1.392
HM	0.584	0.609

We used the abbreviations in the table, where RB stands for review-based recommendation, TB stands for text-based recommendation, and HM stands for the hybrid model. Also, two values are listed for the memory-based coillabrative filtering, since the predictions can be made either using the similarities between users, or the similarities between the restaurants. The algorithm is the same, but we can make two predictions for each example, and therefore have two RMSE listed.

Among the models we've built, the current best one is the collaborative filtering by gradient descent (CF-SGD), which has a testing RMSE of 1.072. Also, one of the advantage of this model is that we don't need to know what exactly the latent features represent, as they can be automatically chosen by the algorithm.

The other CF models also have good performance, with the testing RMSE around 1.1 to 1.2. Since collaborative filtering is a well-studied area, we've implemented these models mainly because we want to compare their performance with our own models. But still, CF relies heavily on the previous interactions between users and items. For example, the cold-start problem is an issue since the newly-registered users do not have any previous activity within the system. Even for the existing users, as the system becomes larger (more users and more restaurants), the previous activities convey less information since the network is becoming significantly more sparse. In fact, this is the direct motivation for our own models proposed in this paper, namely, the review-based and text-based recommendation.

In these two models, we construct a feature vector for each restaurant and a preference vector for each user. Then use the similarities between the feature vector and the preference vector to rank the restaurants for each user, and recommend the restaurants ranked at the top.

The feature vector for a restaurant is always constructed by the reviews it has received. For the review-based recommendation model, the preference vector for users are constructed by combining feature vectors of restaurants and the previous ratings made by the user. This model requires previous ratings, therefore could be used for the existing users in the system.

On the other hand, for the text-based recommendation model, the user's preference vector is constructed directly form the reviews the user has written. At the first glance, this model still need previous reviews from the user. However, since the reviews are just some text the user has written, we can essentially get any text associated with the user from any source. 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. Thus, this model can be used for newly-registered users (cold-start problem).

As a result, these two models can be combined in production to give recommendations to all the users. The performance of the review-based and text-based recommendation models are not as good as the CF models, but the ability of coping with newly-registered users is an advantage.

Lastly, the hybrid model we've built is a linear combination of all the predictions made by the previous models. It achieves a testing RMSE of 0.609. Figure 3 is a visualization of the predictions made by the hybrid model (for the testing set).

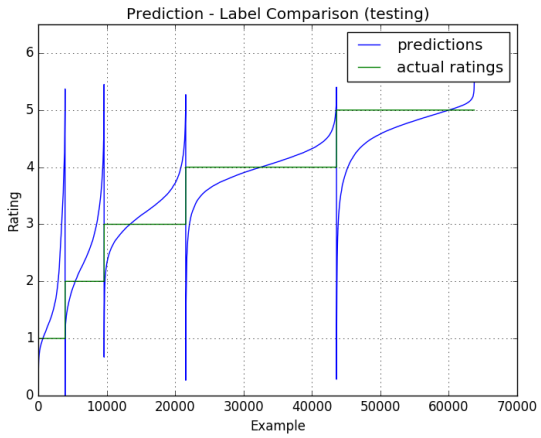

Fig. 3. Predictions made by the hybrid model, compared with the real ratings

In this figure, the actual ratings are sorted. Since the actual ratings only contain integers, the resulting curve contains 5 discrete steps. The predictions for examples of the same actual rating are then sorted among themselves. We can see that our predictions are centered around the real ratings. Most of the predictions have an error less than 1. The error of the examples with actual rating of 1 is the largest among others. We think the reason is that rating 1 is the minority class since people seldom give the lowest rating. Therefore, we resampled the training set to balance the number of examples for all the ratings, and the testing RMSE decreased to 0.553.

Finally, to give a more concrete illustration of the output of our model (hybrid model), we've pick one random user in the dataset, sorted the predicted ratings among all the restaurants, and recommend the highest rated ones. The top-ranked recommendations are:

- Stage Bar (bar)
- Yard House LINQ (american)
- Lucky Streak at LAS (bar)

The highest ranked restaurants that has actual ratings in the testing set for this user include:

- Restaurant El Rey (mexican, ranked 105, actual rating 4)
- The Blind Pig (nightlife, ranked at 232, actual rating 5)
- Hot Feel (asian, ranked at 269, actual rating 4)

In the whole dataset, there are 6978 restaurants being ranked for this user. Therefore the above result shows that the restaurants ranked at the top are indeed liked by the user.

VIII. Conclusion and Future Work

In this project, we've implemented the widely-used collaborative filtering model for recommendation, developed two new recommendation models (review-based and text-based recommendation) that, to some extent, could relieve the cold start problem, and built a hybrid recommendation system that could provide decent recommendations.

In the future, we would like to further optimize the hybrid system. Linear combination is a simple but effective way to hybrid. However, we still want to figure out what are the relationships between the output of each models. Interesting questions to ask include: Does any model always tends to overestimate or underestimate the ratings? Can we identify the errors for two models are somehow positive or negative correlated? and why?

If we can answer those questions, we could have a better understanding of what underlying relationships are each model trying to extract. Then it would be easier and more intuitive to figure out a more creative way of hybridization.

References

[1] Park, Deuk Hee, et al. "A Review and Classification of Recommender Systems Research." International Proceedings of Economics Development & Research 5.1 (2011).

[2] Herlocker, Jonathan L., et al. "An algorithmic framework for performing collaborative filtering." Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 1999.

[3] Linden, Gregory D., Jennifer A. Jacobi, and Eric A. Benson. "Collaborative recommendations using item-to-item similarity mappings." U.S. Patent No. 6,266,649. 24 Jul. 2001.

[4] McLaughlin, Matthew R., and Jonathan L. Herlocker. "A collaborative filtering algorithm and evaluation metric that accurately model the user experience." Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2004.

[5] Lee, Sanghack, Jihoon Yang, and Sung-Yong Park. "Discovery of hidden similarity on collaborative filtering to overcome sparsity problem." International Conference on Discovery Science. Springer Berlin Heidelberg, 2004.

[6] Pazzani, Michael J. "A framework for collaborative, content-based and demographic filtering." Artificial Intelligence Review 13.5-6 (1999): 393-408.

[7] Howe, Michael. “Pandora’s Music Recommender.” A Case Study, I (2009): 1-6.
[8] Lops, Pasquale, Marco De Gemmis, and Giovanni Semeraro. ”Content-based recommender systems: State of the art and trends.” Recommender systems handbook. Springer US, 2011. 73-105.
[9] Adomavicius, Gediminas, and Alexander Tuzhilin. “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.” IEEE transactions on knowledge and data engineering 17.6 (2005): 734-749.
[10] Hofmann, Thomas. ”Collaborative filtering via gaussian probabilistic latent semantic analysis.” Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. ACM, 2003.
[11] Burke, Robin. ”Integrating knowledge-based and collaborative-filtering recommender systems.” Proceedings of the Workshop on AI and Electronic Commerce. 1999.
[12] Gomez-Uribe, Carlos A., and Neil Hunt. ”The Netflix recommender system: Algorithms, business value, and innovation.” ACM Transactions on Management Information Systems (TMIS) 6.4 (2016): 13.
[13] Yelp Dataset Challenge www.yelp.com/dataset_challenge
[14] Joost de Wit. “Evaluating Recommender Systems”, Delft: 1980.
[15] Koren, Yehuda. “The bellkor solution to the netflix grand prize.” Netflix prize documentation 81 (2009): 1-10.
[16] Sarwar, Badrul, et al. “Incremental singular value decomposition algorithms for highly scalable recommender systems”, Fifth International Conference on Computer and Information Science. 2002.
[17] Scikit-learn library: http://scikit-learn.org/stable/
[18] Natural Language Toolkit (NLTK): http://www.nltk.org