Restaurant Recommender System for Yelp

Panagiotis Karagiannis PKARAGIA@UCSC.EDU

Juraj Juraska JJURASKA@UCSC.EDU

Noujan Pashanasangi NPASHANA@UCSC.EDU

Konstantinos Zampetakis KZAMPETA@UCSC.EDU

Bitbucket Repository: https://bitbucket.org/apropos_/project-ml

1. Problem Statement

In this project we explore some of the capabilities of machine learning to extract useful information from the Yelp dataset. More precisely, we deal with the prediction of whether a user likes or dislikes a previously unvisited business. The prediction is performed based on the ratings of other similar customers who have visited the given business. The similarity of customers is determined by how their ratings of other businesses correspond to each other. We focus on a subset of the businesses, namely the restaurants. Our results are then used for recommending restaurants to the user.

There are two principle ways on how to approach the recommendation process. One utilizes the feature similarity of the individual restaurants, while the other is based on the similarity of users, which assumes that similar users rate restaurants similarly. Hence, the primary data that we are going to use are the ratings of the restaurants (i.e. the stars given by the users to the restaurants), included in the review JSONs. Other features of the reviews (e.g. the date), and of the restaurants (e.g. the attributes), may serve for a more advanced future approach in order to enhance the recommendations.

2. Feature Engineering

The restaurant ratings are extracted from the review JSONs into a utility matrix with rows corresponding to the users and columns corresponding to the restaurants. Obviously, this produces a sparse matrix, as each user has only rated a small fraction of all restaurants.

Our program reads and parses the relevant JSON files and creates an internal sparse matrix to hold the data. We extract the following features from the review JSONs: `business_id`, `user_id`, `stars`, and `date`. As mentioned above, the date property is only extracted for a potential future use, such as enhancing the prediction by reducing the importance of outdated reviews.

Besides creating the utility matrix, we filter only the restaurants, i.e. the businesses having `Restaurants` as one of their categories. We store these in a separate data structure for a future reference, when we require additional information about the restaurants. The `business_id` property serves as a key to match the corresponding restaurant, and we currently store the name of the restaurant in the structure.

In order to normalize the ratings across the user database, the average rating for each user is calculated and subtracted from the individual ratings of that user. Although there is an average rating property provided (see `average_stars` in the user JSON), it is calculated over all the businesses in the dataset, but we only consider restaurants in this paper.

The greater part of the ratings is used as the training set for our learning algorithm, and the rest of them as the testing set. Our current implementations partitions the data at random while trying to approximately use 80% as training and 20% as testing. After training the recommendation system we have it predict the ratings for the user-restaurant pairs in the testing set. Subsequently, we evaluate the closeness of the rating predictions to the true values using the root mean square error (RMSE).

3. Model Formulation

We are using the collaborative filtering method to predict the rating that a user would give to a specific restaurant. In this method we need to relate different objects - users and restaurants. There are two main approaches to do so, namely, the neighborhood approach and the latent factor

models. A successful combination of the two aforementioned methods is successfully used in (1), (2). We focused on the latent factor model for starting our work, following (1).

Using the latent factor models such as Singular Value Decomposition (SVD) we are able to compare users and restaurants together, while avoiding the problem of comparing different objects anymore. We associate each user and each restaurant with f dimensional vectors which should be regarded as a vector of different factors inferred from users' ratings. These factors, such as the cost of the restaurants for example, may or may not be explicit, and it is difficult to be exploited from the user's profile and the restaurant description.

We focus on the SVD method for the rating matrix. A typical model associates each user u with a factor vector $p_u \in \mathbb{R}^f$, and each restaurant with a factor vector $q_i \in \mathbb{R}^f$ (we mostly refer to restaurants as items in the rest of the report). The prediction is done by taking an inner product of p_u and q_i. Conventional SVD is undefined when knowledge about the matrix is incomplete. Only addressing the few ratings that we have or inserting not fully considered numbers in the place of the unknown rating, could cause overfitting or inaccurate results. Instead, we only use the ratings given and we introduce regularizing to overcome overfitting.

Some users tend to give higher ratings than average and some restaurants tend to get higher ratings than others. We take these facts into account by using baseline estimates to adjust the rating that we have in the training set. For this reason we introduce the variables b_u and b_i. These are the offsets of a particular user's ratings or the ratings of a particular item, respectively, from the overall average. For example, let us suppose that the restaurants comprise only the university dining halls and that the average rating of all dining halls is 2.7. Furthermore, assume that students rate Stevenson higher by 0.8 (this corresponds to b_i) because it has better food. However, student Jurik avoids gluten and therefore tends to rate the dining halls lower by 0.5 (this corresponds to b_u) because they often do not offer gluten-free options. Therefore the baseline estimate for Jurik's rating of Stevenson is $b_{ui} = 2.7+0.8-0.5 = 3.0$. Notation is summarized in the following table:

r_{ui}	Given rating of user u for the restaurant i
$\hat{r}_{ui}$	Predicted rating of user u for the restaurant i
μ	Overall mean of all given ratings
b_u	Bias for user u
b_i	Bias for item i
b_{ui}	Baseline estimate for user u and item i
$\mathcal{K}$	Set of all given ratings r_{ui}

To be more precise the prediction will have the following form:

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i$$

As we mentioned above, the loss function we use in order to learn the parameters p_u, q_i and b_{ui} is:

$$\sum_{(u,i)\in\mathcal{K}} (r_{ui} - b_{ui} - p_u^T q_i)^2$$

where $b_{ui} = \mu + b_u + b_i$. Now we add a regularization term:

$$\sum_{(u,i)\in\mathcal{K}} (r_{ui} - b_{ui} - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

where λ is a regularization constant, chosen heuristically.

To minimize the previous expression over p_u, q_i, b_u and b_i we use the gradient descent method. We implemented this method from scratch, using the following updates for the parameters. Each update is done by iterating over the whole set of existing ratings:

$$\begin{aligned} p_u &\leftarrow p_u + \gamma' \left((r_{ui} - \hat{r}_{ui}) q_i + \lambda' p_u\right) \\ q_i &\leftarrow q_i + \gamma' \left((r_{ui} - \hat{r}_{ui}) p_u + \lambda' q_i\right) \\ b_u &\leftarrow b_u + \gamma'' \left((r_{ui} - \hat{r}_{ui}) + \lambda'' b_u\right) \\ b_i &\leftarrow b_i + \gamma'' \left((r_{ui} - \hat{r}_{ui}) + \lambda'' b_i\right) \end{aligned}$$

where γ', γ'' are the learning rates and λ', λ'' are regularization coefficients.

4. Testing on a Simple Example

In implementing the test case, we consider a very simple scenario where we only assume three users: (u_0, u_1, u_2) and three restaurants (i_0, i_1, i_2). The rating of each user for a particular restaurant can be summarized in the following table:

T	i_0	i_1	i_2
u_0	5	4	5
u_1	1	2	2
u_2	5	4	4

The entry $r_{uj} = s$ signifies that user u_i has rated restaurant i_j with s stars.

Now we conceal some of the data and try to predict the values in order to fill in the table and test the accuracy of our algorithm.

T	i_0	i_1	i_2
u_0	5	4	5
u_1	1	2	x_1
u_2	5	x_2	4

This example is deliberately constructed in such a way as to be able to intuitively guess the missing entries in table **T**, even without knowing the true values. More precisely, we see that user u_0 and u_1 have opposite taste in restaurants while u_0 and u_2 seem to be very similar. Finally, from the data available to us it would seem that u_1 and u_2 have differences in the way they would rate the same restaurant. Therefore, given this data one would intuitively predict that x_2 should be relatively high, since u_0 has rated the same restaurant as high. Nevertheless, x_1 would be relatively low since both u_0 and u_2 have rated the business high.

Predicting by using the above algorithm we deduce vectors p_u and q_i as well baseline estimates b_{ui} s.t. $b_{ui}+p_u^T q_i$ gives the predicted rating of user u for restaurant i.

Let r_{ui} be as above. Then after running our algorithm we see that:

$$r_{12} = x_2 = 2.2857$$
$$r_{21} = x_1 = 3.7857$$

calculating the RMSE error we get that:

$$RMSE = \frac{1}{\sqrt{2}}\sqrt{(2.2857-2)^2+(3.7857-3)^2} = 0.4879$$

which shows that our prediction is accurate.

5. Evaluation

As mentioned above, we use the RMSE to evaluate our predictions. First we separate our data into two different parts, training data and test data. After using sparse matrices in our implementation of the SVD algorithm we managed to incorporate all 10^6 available ratings in our program.

In order to evaluate our results we compare the value of the RMSE obtained by SVD to the RMSE of rudimentary methods, namely the *naive* and the *baseline* methods.

By naive method we mean that the prediction for each user-item pair is fixed and is equal to the overall average of the ratings in the training set. The average rating for our training set which was built from the entire relevant Yelp dataset is $\mu = 3.8435$. More precisely, this means that on average each restaurant receives a rating of 3.8435. The RMSE corresponding to this method is 1.3258.

The baseline method improves on the naive method by taking the user biases and item biases into consideration. Here the prediction is computed as the baseline prediction b_{ui} and the corresponding RMSE is equal to 1.3155.

Finally, the SVD algorithm improves even further the baseline method as we described above. We choose the parameters of the gradient descent empirically. Our experiments revealed that the best RMSE is achieved with $\gamma' = \gamma'' = 0.04$ and $\lambda' = \lambda'' = 0.8$, as we can see in the following table:

Table 1. RMSE with different γ, λ values

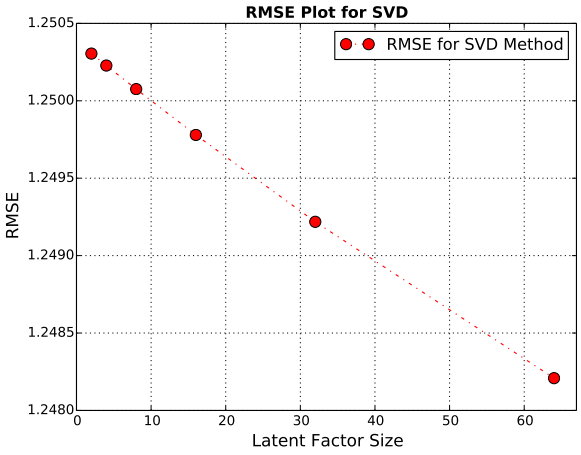
γ	RMSE
0.001	1.1191
0.005	1.0937
0.01	1.0809
0.02	1.0711
0.03	1.0681
0.04	**1.0676**
0.05	1.0682
0.06	1.0694

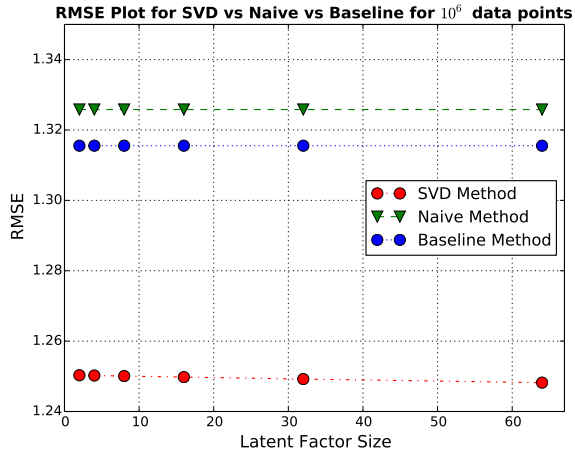
λ	RMSE
0.5	1.0650
0.6	1.0636
0.7	1.0628
0.8	**1.0626**
0.9	1.0628
1.0	1.0633
2.0	1.0743
3.0	1.0852

The RMSE we obtained for the different methods while varying the size of the latent vectors can be summarized in the table below:

Table 2. RMSE for different values of latent factor size

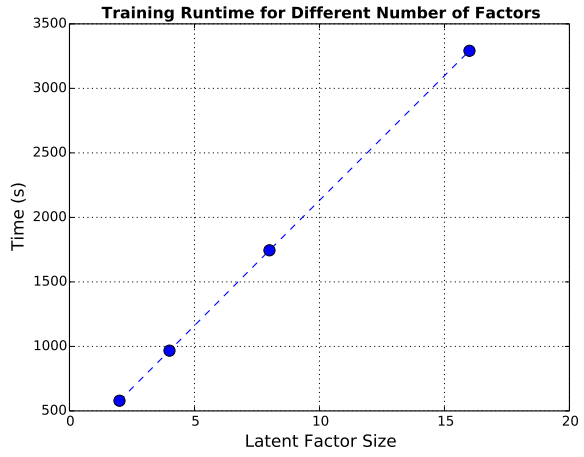
Number of factors	RMSE
2	1.2503
4	1.2502
8	1.2501
16	1.2498
32	1.2492
64	1.2482





6. Results

We consider our results to be relatively accurate since the RMSE obtained by running the algorithm on the complete dataset is significantly lower than the RMSE obtained by the simpler methods similar to what we can observe in (1). Moreover, the RMSE decreases as the number of latent factors in the model increases. Also, we notice that as we increase the number of latent factors the runtime of our program increases in a linear fashion. In the graph below we include the runtime measures for $2, 4, 8, 16$ factors using an Intel Core i7-5600U running at 2.6 GHz:



To test our implementation we recommend restaurants for user with id `PUFPaY9KxDAcGqfsorJp3Q` using 64 factors. The restaurants as well as the rating predictions for this user are in Table 3.

It is worth noticing that the recommended restaurants include ones with a lower rating on Yelp which suggests that our recommender indeed gives personalized results for each user. Our algorithm does not naively suggest only top rated restaurants.

Table 3. Restaurants and Ratings

Restaurant	Predicted Rating	Yelp Rating
Waldschänke	5.00	5
Green Traiteur & Cafe	4.93	5
XO Cafe & Lounge	4.90	3
Roly Poly	4.90	3.5
Asia Food	4.87	4.5

7. Challenges

The main challenge that we faced was mainly to overcome the difficulty of working with all the data. At first, in order to narrow the problem down, we created two different files with 10,000 and 100,000 ratings respectively. Even in that case, it would take our algorithm more than an hour to terminate. As a result, we had to change our implementation approach and we were forced to work with a ratings matrix instead of using json files. Due to the structure of the Yelp Data, the ratings were very sparse therefore we chose to implement a Python sparse matrix in the coordinate (COO) format. We are able to run the program for all the data in roughly two hours for 16 factors. It is obvious that when we increase the number of factors the running time of the program increases. In order to be able to use more factors, we have to improve our running time. So our challenge now is to improve the running time so that we can increase the number of latent factors. Moreover, it would be helpful to extract the location of the user for whom the recommendation is performed so as to be able to recommend nearby restaurants.

References

[1] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[2] Y. Koren, R. Bell, C. Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.