Prediction of Google Local Users' Restaurant ratings

Shunxin Lu · Muyu Ma · Ziran Zhang · Xin Chen

Abstract Since mobile devices and the Internet play a great part in people's daily lives, they greatly affect the way in which people evaluate places. Instead of directly going to a place without any knowledge about it, people now can easily search its rating on any online rating websites. Also, it is always convenient to leave a review and rating once one has visited a place, and this not only helps build a richer dataset about this place (thus providing more help to others regarding this database) but also acts as a kind of social interaction: One can share where one has been, what one has done, and how one feels about it with others. In this context, Google introduced its service on which users can give reviews to business around the world. Google promises that businesses will be connected "directly with customers, whether they're looking for you on Search, Maps or Google+" and "customers can show their appreciation with ratings and reviews" [1]. We want to investigate which features are highly correlated with the rating and which are not, as this might help estimate ratings when data are missing or help businesses categorize reviews by high and low ratings in order to analyze its strength and weakness.

Keywords Rating prediction • Google Local • Model fitting and optimizing

Instructor: Professor Julian McAuley
Room 4102, Computer Science Department, University of California, San Diego
e-mail: jmcauley@cse.ucsd.edu
Authors:
Muyu Ma, email: muma@ucsd.edu, PID A99132320
Xin Chen, email: xic085@ucsd.edu, PID A12067598
Ziran Zhang, email:ziz025@ucsd.edu, PID A98085112
Shunxin Lu, email: shl252@ucsd.edu, PID A12420176

1. Introduction:

The dataset we used for this project is Google Local (Map & Restaurant) that contains the basic information of businesses around the world and the ratings from the users. There are three json files in the dataset which are *place.json*, *reviews.json* and *user.json*.

The place.json has the information about the *hours, close status, address, coordinate and, name and ID of the business* from all around the world. The table below shows the total number of businesses around the world, and the number of businesses that are still operating.

Total business	3114353
Business already closed	100215
Business still open	3014138

The *reviews.json* has *userName, rating (*from customers*), review text, categories, gPlusPlacedId, text time* and *utime.* The total number of reviews received is shown as below.

Total reviews	11453846

The user.json has *username*, *current Place*, *education* and *previousPlaces*. The total number of users is shown as below.

Total users	3747937

We discovered that the overall data is too large hence we decided to shrink it down. After analyze the distribution of the dataset, we found out that businesses, in particular restaurants(since they consist most of the overall businesses), in San Francisco are dense and thus it is probably easy to have a better-fitting model.

The distribution of restaurants in San Francisco area

In the above picture we can see that the distribution of restaurants in San Francisco area is pretty dense: the color represents the amount, or the density of the restaurants, the denser the they are, the redder the area is on the graph, and the most concentrated area appears to be white. And after we shrink the data the total review amount is still pretty large(see below), so we choose reviews restricted to San Francisco Restaurant to be our data.

We generate four files below:

• *SFRestaurantReviews.json*: Contains all the restaurant reviews in San Francisco area.

• *SFRestaurants.json* : All restaurant names in San Francisco area.

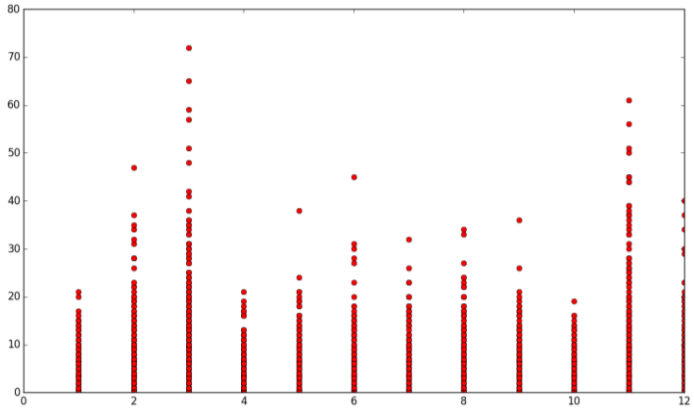
• *SFRestaurantUsers.josn*: All users who gave the reviews.

• *SFRestaurantRatings.json:* The ratings of each restaurant.

The table below shows some properties we found in the new json files.

Total business	11220
Total restaurant	4035
Total users	22018
Total active restaurant	3593
Total reviews	114553
Total restaurant review	66875

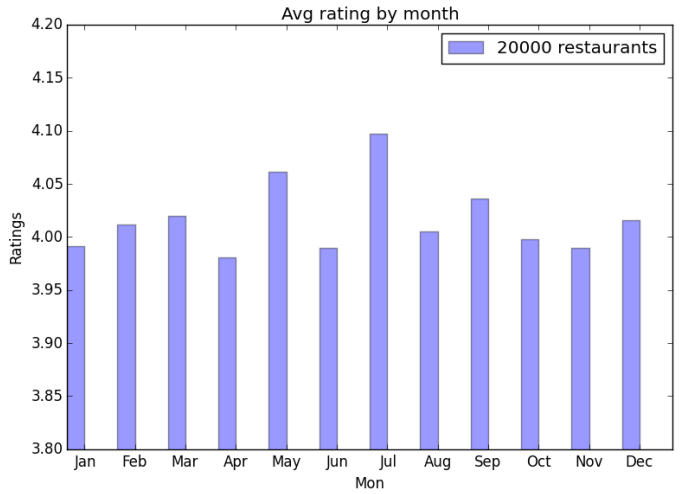
Take a closer look at the data we have, we found out that month might affect people going to restaurant or not:



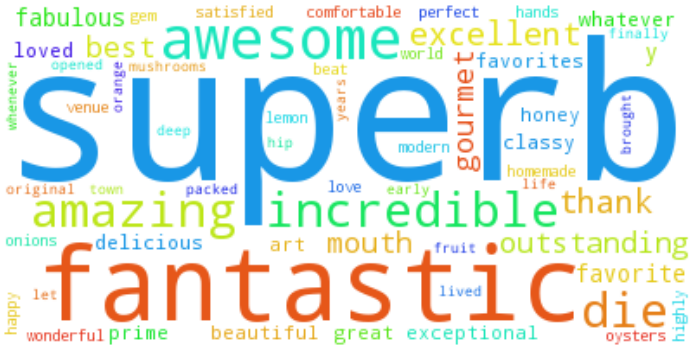
The number of people reviewed for each month

In the above picture, the x axis is month, y axis is the number of reviews people gave. Each point represents a particular restaurant; which means a point at x=3 and y=50 means this restaurant in March receives overall 50 reviews. In order to make this graph as realistic as possible, we tossed out the restaurants that have total reviews less than 20, since their monthly data have little chance to have an accurate predictive pattern.

Inspired by this, since we are predicting rating, we generated a graph of 20000 average ratings by month as follows:



We also looked into the text and its correlation with rating by implementing text mining, and we found out that the most positive-related words, and most negative-related words respectively:



Most positive words

Most negative words

By these preliminary observations, we decided to use average user ratings and average restaurant ratings to be features as they appeared to be helpful in our first assignment, as well as trying months and text, since they appeared to be correlated with the reviews somehow.

2. Predictive Task:

We are trying to predict what rating would a certain review be, given a full training set, and test data with ratings missing.

In order to test the performance of prediction, we came up with two baseline models.

1. Predict each review's rating by this review contributor's average rating. The average rating of each single user in the training data set is calculated beforehand. If the user never appears in the training set, use the overall average rating (average rating of all the reviews in the training set) instead.
2. Predict each review's rating by the average rating of the restaurant being reviewed. Average rating of each restaurant in the training data set is calculated beforehand. If that restaurant never appears in the training set, use the overall average rating (average rating of all the reviews in the training set) instead

Then we started building our model. The features we used are listed below:

• Offset, which represents the intersection of our linear model and y-axis.

•Average rating given by the user in the training data set. If the user never appears in the training set, use the overall average rating (average rating of all the reviews in the training set) instead. These data are processed by making a defaultdic object for each unique user ID, and loop through the training set to average over all the review ratings for each user.

• Average rating of the restaurant in the training data set. If that restaurant never appears in the training set, use the overall average rating (average rating of all the reviews in the training set) instead. Similarly, these data are processed by making a defaultdic object for each unique place ID, and loop through the training set to average over all the review ratings for every restaurant.

• Popularity of the user in training set. These data are processed by looping through the training set and adding up each user's total review count.

• Popularity of the restaurant in training set. These data are processed by looping through the training set and adding up each restaurant's total review count.

• The score of the text review determined by the result of text mining we performed on training set.

• The length of text review.

• Month in which the review is given. We divide the months into 4 parts, with a feature vector of length 4. For example, if the review is given in first quarter of the year (namely Jan, Feb, Mar), we append a [1,0,0,0] to our feature, etc.

To assess the validity of our model, we used the test set to test the performance of our model on unseen data. After we built our model, we used the validation set to tune our model, in order words, used the validation set to find the lambda that can minimized the error.

3. Model:

3.1 Linear Regression

One model we chose is linear regression. The reason we chose this model is that we would like to make a prediction, so we should use one of the supervised learning models. And among them, we are most familiar with linear regression.

"Linear regression is an approach for modeling the relationship between a scalar dependent variable *y* and one or more explanatory variables (or independent variables) denoted *X*."[3]

$$Y = X\theta$$
$$\theta = (X^T X)\text{^} - 1X^T y$$

To divide the data into training, testing, and validation sets, we first shuffled all the reviews randomly and took the first 22000 as our training set, then the next 22000 as testing set, and the next 22000 as validation set.

3.1.1 Text Mining:

We incorporated text mining inside our linear regression model.

In order to get the text score, we first performed a text mining on the reviews in training set. We looped through all the review texts and picked out the top 1000 most frequent unigrams. Then we performed the sentiment analysis on those most common words on the training data. We used ridge regression for this analysis so that the result we got would become more reasonable with regularization. Then after we had the 1000 theta values corresponding to those words, we sorted the theta values to find the words with most positive and negative impact on the model as keywords in out scoring system[4].

Then we basically used the theta values of the keywords to determine how many scores a text review can get. For example, when we were looping through the reviews in validation set, in one review, if one of the most common words appears, then we go to a dictionary, which contains those keywords as keys and their theta values from the sentiment analysis as values, look for the value of that keyword. After we added the values of all the keywords in a text review, we could use the sum as the final score the review gets.

3.2 Optimization:

After we built our model using the features mentioned above(all the features listed in last section plus the text mining), the MSE and MAE of our model were significantly less than that of the baseline models. But since it was just a simple linear regression model, it did not have any optimization and regularization. So we decided to perform a gradient descent on our model to minimize the residual the model gets but also penalize the complexity of the model. The gradient descent equation looks like this:

$$arg\ min_\theta\ = \frac{1}{N}\ \| y - X\theta \|_2^2 + \lambda \| \theta \|_2^2$$

In order to avoid over-fitting, we adjusted the lambda value again and again until we found the value that minimized the error in validation set.

After we performed the gradient descent, we were able to get the optimized theta, and use it to make a new prediction on the validation set. And we got some improvement in terms of accuracy.

The strength of the linear regression model is that it is easy to build, and with optimization, it can get a fairly reasonable result comparing to the baseline models we set. The weakness of this model is that there is not much room for improvement after we reached certain point. Even though we had tried many features, not many of them were useful and some of them even increased the error.

3.4 Other Model Considered-Matrix Factorization

We found a method called Matrix Factorization online with sample provided. It is a method widely used in recommendation system, often related with film rating prediction. The details of the model is described below.

We have a set of users as $\boldsymbol{U}$, and items as $\boldsymbol{D}$. Construct $\boldsymbol{R}$. It has size of $|\boldsymbol{U}| \times |\boldsymbol{D}|$ and inside the matrix we align all ratings if this user-item pair exists in the review data, and append 0 otherwise. And we assume we want to find K latent features. Then we want to get 2 matrices $\boldsymbol{P}$ (size of $|\boldsymbol{U}| \times \boldsymbol{K}$) and $\boldsymbol{Q}$ (size of $|\boldsymbol{D}| \times \boldsymbol{K}$) such that their cross product is approximately $\boldsymbol{R}$:

$$R \approx P \times Q^T = \hat{R}$$

Then since we know the formula we can calculate each entry of R hat in such way:

$$\hat{r}_{ij} = {p_i}^T q_j = \sum_{k=1}^{k} p_{ik} q_{ki}$$

Then we need to find P and Q using gradient descent: first initialize P and Q randomly, then calculate their cross product's difference with real R. And we use squared error as the "difference" we want to compare:

$${e_{ij}}^2 = (r_{ij} - \hat{r}_{ij})\text{^}2 = (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{ki})\text{^}2$$

Since our goal is to minimize this error, we need to know the direction(e.g., by what value should we change pik and qik). So we differentiate the error above with respect to the 2 variables:

$$\frac{\partial}{\partial p_{ik}} {e_{ij}}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{ki}$$

$$\frac{\partial}{\partial q_{ik}} {e_{ij}}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{kj}) = -2e_{ij}p_{ki}$$

Then we finally formulate the rules for the 2 variables in which we wish to update our matrix.

$${p'}_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} {e^2}_{ij} = p_{ik} + 2\alpha e_{ij} q_{ki}$$

$${q'}_{ki} = p_{ik} + \alpha \frac{\partial}{\partial q_{ki}} {e^2}_{ij} = q_{ik} + 2\alpha e_{ij} p_{ik}$$

Then we can calculate the overall error per iteration in order to achieve a small error where we can break the loop(for example, when sum error < 0.001) or reach the maximum iteration times (for example, 50000 times).

However we still need to regularize our model not only to minimize the error, as it is the prime objective, but also penalize the complexity. So we can change a different error calculation method in which beta is the controller of magnitudes of user-feature and item-feature vectors(in practice beta is around 0.02):

$${e^2}_{ij} = (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{ki})\text{^}2 + \frac{\beta}{2} \sum_{k=1}^{k} (||P||\text{^}2 + ||Q||\text{^}2)$$

Then our new update rule is the following:

$${q'}_{ik} = q_{ik} + \alpha \frac{\partial}{\partial q_{ik}} {e^2}_{ij} = q_{ik} + \alpha(2e_{ij}p_{ki} - \beta q_{ik})$$

$${p'}_{ik} = q_{ik} + \alpha \frac{\partial}{\partial q_{ik}} {e^2}_{ij} = q_{ik} + \alpha(2e_{ij}q_{ki} - \beta p_{ik})$$

By this method we might come up with a rating prediction matrix with pretty small error.

In order to test the error on predicted data, we have to extract some existing rating and pretend this user-item pair does not exist in order to calculate the error. In practice we randomly pick out some ratings and change them to 0, and store the real value separately. Then after iteration, we calculate the "difference"(MSE and MAE) in order to see how different the prediction is from the real value.

The strength of such model is that it works well on a dense matrix, since all it does is comparing its result with original data, if there is more to compare, the more accurate the model would be on those unseen data.

4. Literature:

Google local dataset is a popular resource to study for data mining purpose. After research, we found a paper by Idan Izhaki and Diego Cedill[2] described how to use user score to build a restaurant recommendation system for the whole country by using the Google Local data, which is really similar to our task. In their paper, they first merged the restaurant by category and sort them by the number of reviews, as the result they found the top ten categories restaurants in America. On the other hand, they believe: "different restaurants in the US will get different scores for the exact same restaurant in a different location". In order to make sure the accuracy,

they used K-mean algorithm to solve the issue.

For the feature, they decided create a matrix that has 300 columns to represent the location of resultants. But eventually they found the result was not what they expect so they decided to create another feature that used the average of user rating. In this way, their result had impressive improvement.

For the model, they used the simple linear regression. In addition, they defined some terms to explain the problem:

• MSE: mean error square

• FVU: coefficient of determination

• y-bar predictor: prediction of label

In the end, they found out the method hasn't taken regularization into account because the weight of the predication were over fitted the training set.

Comparing to our project, since we only predict the restaurant rating in San Francisco area so we didn't ran into the issue that franchised restaurants would get different rating because they were in the different location. For the feature, as we talked earlier in this report, we chose the restaurant and user average rating that would boost the accuracy of prediction (which, in practice, seemed to have a better result then theirs). Also we used text mining to improve the accuracy of our prediction because we could find the words with the biggest positive and negative impact on the rating given by a user.

5. Conclusion:

Using our final linear regression model to make prediction, the MSE(what we optimized) and MAE(calculate just in order to get a better intuition) were way better then our alternative model and also had significant improvement comparing to the baseline models. The features that made most significant improvement to our model are the average user rating, average restaurant rating, and the score of the text review.

The theta values we got for our model are:

[-0.02344723, 0.55997867, 0.4446833 , -2.67956494, 1.21124313, 1.01243449, -0.11984632, -0.02030658, -0.01441798, 0.01673247, -0.00545516]

Which could be interpreted into:

1.If all the features appear to be zero, the rating would be predicted as -0.02344723.

2.When the average user rating in training set goes up by 1, the predicted rating will go up by 0.55997867.

3.When the average restaurant rating in training set goes up by 1, the predicted rating will go up by 0.4446833.

4.Every review the user made in training set will decrease the predicted rating by 2.67956494.

5.Every review the restaurant received in training set will increase the predicted rating by 1.21124313.

6.When the score of the text review increase by 1, the predicted rating will go up by 1.01243449.

7.Every word in the text review will decrease the rating by 0.11984632.

8.If the review is made during the first quarter of the year, the predicted rating decrease by 0.02030658.

9.If the review is made during the second quarter of the year, the predicted rating decrease by 0.01441798.

10.If the review is made during the third quarter of the year, the predicted rating will go up by 0.01673247.

11.If the review is made during the forth quarter of the year, the predicted rating decrease by 0.00545516.

The features that did not work well were:

1. Length of categories.

2. Using a binary feature vector to represent different categories.

3. Using binary feature for each month. It seems that the rating is only seasonal related, instead of relating to each month directly.

The linear regression model is successful because we chose some of the most useful features for prediction. The Matrix Factorization Model did not work as well is because, based on the reviews we have, the matrix we built which represents the rating giving by users to restaurants is very sparse, in other words, many elements in the matrix are 0s. This made the factorization and prediction difficult to yield reasonable results.

Note: the MAEs are 1000 times larger since the ratings in

Models	MAE	MSE
Baseline1 (user rating)	682.966475465	847612.366343
Baseline2 (restaurant rating)	712.904201073	858728.696218
Linear regression model	634.94165769152	674277.587854
Matrix Factorization	2090.86435978	5143999.789

the file are in the scale of 1000, 2000, etc.

The result is significance as it can be used in various conditions. For example, we can use this model to predict certain reviews' ratings if the rating data are missing. By discovering these features' relationship with ratings an user give, we gave an insight of user reviews that might help the business owners as well as review site builders.

6. Challenge:

As mentioned above, the dataset that Google provided is really sparse:

• There are some users rated only few restaurants, making it hard to train or predict (for instance, nearly half of the user only rated 1 restaurant). We had to overcome this difficulty by tossing out users who went fewer than 3 restaurants.

• Some restaurants only have few rating: similar as the previous one, we had to throw away those restaurants having fewer than 7 reviews.

• In *place.json*, there are many places that repeatedly showing up.

• Some users have review in the *review.json* but we couldn't find this user in the *users.json*. To overcome the challenge, we only count the user ID if it shows up in the *review.json*

7. Reference

[1] https://www.google.com/business, the description of Google Business.

[2]http://cseweb.ucsd.edu/~jmcauley/cse255/projects/Idan_Izhaki_Diego_Cedillo.pdf , the paper written by previous 255 students.

[3]https://en.wikipedia.org/wiki/Linear_regression,the wikipedia page of linear regression.

[4]http://jmcauley.ucsd.edu/cse190/code/week5.py,the code provided on Professor McAuley's website.