Gimme Food by The Yelping Stones Introduction

When using Yelp to explore new restaurants around the area, we often noticed that there were restaurants with high ratings that we didn't particularly like or find that a random restaurant we enjoyed doesn't have such a high rating. We will use the dataset from the Yelp Dataset Challenge that consists of users and restaurants data with their reviews. The intuition behind this project is that people with similar taste tend to agree on which restaurants are good. Our final goal would be to recommend restaurants to users based on their previous reviews/ratings to other businesses. To frame this into a machine learning problem, recommending restaurants to users can be translated into predicting a restaurant's rating of a user. If we are able to predict a user's rating to a specific restaurant, we can easily recommend restaurants to the user based on the top rated restaurants.

Dataset

(Dataset URL: http://www.yelp.com/dataset_challenge)

Data was provided as JSON files for businesses, reviews and users. Since the dataset was too large to run different algorithms, we reduced the size of the data by filtering it down to a meaningful subset.

We filtered the businesses according to a selected location. This was because recommendation should be based on a user's location and the rating matrix should not be too sparse. The restaurants in "North Las Vegas" were selected and only the reviews on these restaurants were selected. Accordingly, we selected users who have more than 10 valid reviews on the filtered restaurants since we need enough reviews per user to get enough data about each user's preferences.

Original dataset: business: 61184, review: 1569264, user: 366715

North Las Vegas datatset: business: 213, review: 1447, user: 139

With the filtered data, we could create a review matrix (users represented as rows and business as columns) with density of \sim 5%.

Related Works

The framework we used for developing a prediction system is generally known as "collaborative filtering," in which items are recommended based on the preferences of other users who have rated the items.

1. Item-Item Collaborative Filtering [neighborhood method] [1]

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j)(r_{u,j} - b_{u,i})}{\sum_{j \in S} |s(i,j)|} + b_{u,i}$$

 $p_{u,i}$ is the predicted rating for user u and item i. $r_{u,j}$ is the rating for item j by user u.

 $b_{u,i}$ is the baseline rating for the user, which we used as the average rating of the user in the training dataset s(i,j) is the similarity metric between item i and item j. Cosine similarity was used in our case.

This algorithm finds restaurants the user has rated already similar to the restaurant whose rating we are predicting and gets the weighted average of the ratings based on the similarity. Item-item collaborative filtering would work better than user-user filtering if it is easier to identify similarities between the items than users (businesses have more overlapping reviews between other restaurants than users have with different users)

2. User-User Collaborative Filtering [neighborhood method] [2]

The same as item-item but instead of using similarities between items, similarities between users are calulated and ratings of other users who rated the restaurants are used. This would work better if finding similarity between users was easier than between restaurants.

3. Matrix Factorization [latent factor method] [3]

$$\min_{q \cdot, p \cdot} \sum_{(u,i) \in \mathbb{K}} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

q is the latent factor matrix for the items. p is the latent factor matrix for the users.

 $\boldsymbol{\lambda}$ is the regularization parameter to avoid overfitting.

This algorithm finds the 'latent factors' that determine the user's rating Only using the observed data, the latent factor matrix for the items and the users are calculated and the rating matrix is obtained via the dot product of the two matrices. Although this is a more complicated algorithm compared to the neighborhood models, it tends to give

better results. Another drawback for this algorithm is that it is nonconvex so even after converging it might not be the globally optimal solution.

4. Hybrid recommender [4]

Because each model has different strengths and weaknesses, the predictions can be mixed together by adding them up with different weights. The weights would change with the dataset, as the review matrix gets more dense the similarity calculations would become more accurate.

Models/Methods

We extended the 3 collaborative filtering algorithms to 11 models by modifying each algorithms with additional features.

- 1. Models using item-item similarity
 - Model 1: Basic item-item similarity with ratings
 - Model 2: Extended item-item similarity with business categories
 Calculates item-item similarity with business categories. Restaurants that contain more common keywords in their category description are considered more similar.
 - Model 3: Extended item-item similarity with business review
 Calculates item-item similarity with business reviews. Restaurants that have more overlapping words in their reviews are considered more similar
 - Model 4: Extended item-item similarity with business reviews with specified keywords
 Calculates item-item similarity with business reviews with words in specific categories. The categories {service, taste, good, bad, price} were considered (For example, for the category 'taste', words such as "spicy", "delicious" were counted). Restaurants with similar category frequencies are considered similar.
- 2. Models using user-user similarity
 - Model 5: Basic user-user similarity with ratings
 - Model 6: Extended user-user similarity with user compliments
 Calculates the user-user similarity with user compliments. User compliments describes what kind of a user he or she is and users with similar description are considered more similar/
 User compliments example in yelp dataset:
 "compliments": {"cute": 2, "plain": 2, "writer": 2, "note": 1, "hot": 1, "cool": 1, "more": 1}
 - Model 7: Extended user-user similarity with user reviews
 - Similar to Model 3 but with user similarity
 - Model 8: Extended user-user similarity with user reviews with specified keywords Similar to Model 4 but with user similarity
- 3. Models using matrix factorization
 - Model 9: Basic matrix factorization method
 Uses matrix factorization where missing matrix values are not considered during the iteration.
 - Model 10: Matrix factorization with initializing missing values with user average rating Modified version of Model 9 to initialize missing values with user average rating.
 - Model 11: Matrix factorization with initializing missing values with restaurant average rating Modified version of Model 9 to initialize missing values with restaurant average rating.

Hybrid Model

The predictions from the models were blended to make the final prediction. The weights of the models to use were determined by using multivariate linear regression with two different regularization methods to prevent overfitting, namely Lasso Regression and Ridge Regression.

Lasso was the more intuitive way since Lasso can zero out weights which eliminates variables that doesn't help fit the data but we decided to use both regularizations and see how they performed differently.

Testing process

i) Testing group

Among 139 users in our filtered dataset, we designated 70% of them to be in the training group and the other 30% to be in the testing group. Among the reviews of the users in the testing group, we designated 30% of the reviews to be testing reviews and they were hidden for testing. The other 70% of the reviews were used to calculate the similarity between the users and predict the hidden ratings.

For testing the hybrid model, we took the predictions from the test set and used 50% for training and 50% for testing. Mean squared error was used for evaluating the models.

ii) Cross-Validation

Cross-Validation was used to determine the regularization parameters for matrix factorization and lasso/ridge regression that led to the smallest MSE.

- Matrix Factorization
 - 10-fold cross validation was used to run different stochastic gradient descents to find the best hyperparameters step size, regularization parameter, and number of latent factors. The below are lists of values examined. step_size: [0.0002, 0.02], regularization_parameter: [0.001, 0.01, 0.02, 0.05, 0.1] number_of_latent_features: [2, 10, 20, 50, 100, 200]
- Lasso regression, Ridge regression Lasso regression and Ridge regression were done using the sklearn python library functions LassoCV and RidgeCV which have built in cross validation to set the regularization parameter α. LassoCV uses coordinate descent and RidgeCV uses Generalized Cross Validation (GCV), which is a more efficient form of leave-one-out cross-validation. During our research, we came upon the fact that cross validation of the form of k-fold or leave-one-out cross-validation doesn't always find the optimal solution for Lasso regression, which is why we used the LassoCV function with the built in cross validation.

iv) Optimizing Matrix Factorization

Stochastic gradient descent for the matrix factorization model finds the local minimum point. However, it is not guaranteed to be the global minimum. Thus, in order to get a value close to the global minimum point, we ran matrix factorization 10 times and used the one that had the lowest mean squared error. This was also reflected in hybrid models. Model 9, 10, 11 were ran 10 times and only the ones with lowest error were considered.

Results & Analysis

1) Test results for 11 distinct models

The below is a bar graph of mean squared error (MSE) of different models. MSE of a naive algorithm which rates any restaurant as the user's average rating was added for comparison.

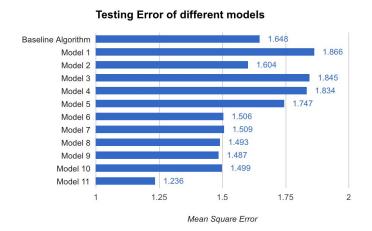


Figure 1. Testing Error with different models

Each model performed as expected. Basic item-item and user-user collaborative filtering performed worse than the baseline algorithm since the data was too sparse to find enough similarity between different users and restaurants. For the item-item collaborative filtering, the restaurant categories seemed to be the feature that was the most effective in finding similarities between restaurants. Model 3 and Model 4, which used the reviews that were left for the restaurants didn't work well when finding similarities between restaurants. However, the same approach seemed to be effective for finding user-user similarity. The assumption was that people who used similar words to describe restaurants had similar criteria for judging a restaurant and using that similarity, Model 7 and Model 8 performed well. Model 6, which characterizes the type of user through the tags provided by other users, was also effective in finding users with similar ratings.

The latent factor models (Models 9 \sim 11), as seen from research, performed better than the neighborhood models (Models 1 \sim 8). The basic matrix factorization model, which only used observed data to minimize the mean squared error performed slightly better than the best of the neighborhood models. For the hyperparameters, step size of 0.0002, regularization

parameter of 0.01, and 2 latent factors worked the best for the basic matrix factorization model after running 10 fold cross validation. The modified version of the latent factorization model by filling in the missing data with the user average rating (step size = 0.0002, regularization parameter of 0.1 and 20 latent factors) turned out to do worse than the basic model but filling it in with restaurant average rating (step size = 0.0002, regularization parameter 0.01, and 20 latent factors) turned out to be the best model for predicting the ratings. The constant step size worked better in our case, after seeing using step sizes such as 1/(number of iterations) or $1/\sqrt{\text{number of iterations}}$ led to significant increase in the error rate. For testing for convergence, we stopped after the error rate started plateauing, especially since waiting until the error rate reaches 0 would overfit the model to the training data. The error in **Figure 3** is measured by the sum of the mean squared error for all considered data in the matrix, which explains the large value.

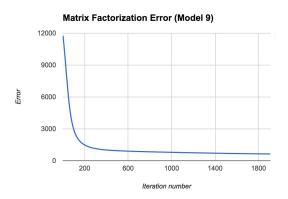


Figure 3. Matrix Factorization error for iterations for Model 9 (10, 11 were omitted)

2) Test results for hybrid models

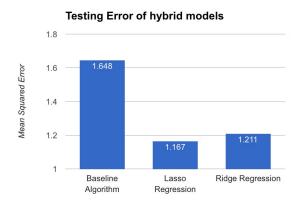


Figure 2. Testing Error with hybrid models

Model	1	2	3	4	5	6	Model	1	2	3	4	5	6
Weight	0	0	0	0	0.185	0	Weight	0.104	0.004	-0.165	-0.08	0.183	-0.03
Model	7	8	9	10	11	Intercept	Model	7	8	9	10	11	Intercept
Weight	0	0	0.237	0	0.282	1.016	Weight	0.0277	0.049	0257	0.085	0.228	1.19

Table 1. Coefficients for Lasso Regression (Left) (alpha = 0.087)/Ridge Regression (Right) (alpha = 59.28)

Both results for Ridge and Lasso regression were considered. As expected, Lasso regression performed slightly better than Ridge regression because Lasso regression got rid of the variables that seemed to introduce too much bias. For Lasso, the models that ended up being used were Model 5 (basic user-user similarity), Model 9 (basic matrix factorization model) and Model 11 (matrix factorization model with missing data filled with restaurant average). It was surprising that the model didn't consider the models with better mean squared error rates, but considering the testing ratio was 50% (50% was used for training, 50% was used for testing) it doesn't seem to be a problem with the evaluation process. The alpha value for Lasso

was determined using coordinate descent, which was built in to the LassoCV function and the mean squared error values for the different alpha values are plotted in **Figure 4**. Unfortunately for Ridge regression, the cross validation data was not available from the built in RidgeCV function, so the plot is not available.

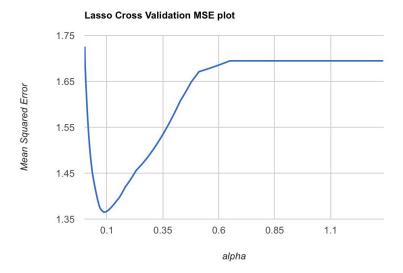


Figure 4. Mean Squared Error plot for different alpha values for Lasso Regression Cross Validation

Conclusion/Future Work

Using the hybrid model, we were able to achieve a mean squared error rate close to 1, which is a fairly accurate model considering the errors are squared. However since the ratings are discrete values in the interval 0.5, the mean squared error was a better representation of the errors than the I-1 norm (in MSE, two errors of 0.25 difference is considered less than one error of 0.5). There certainly could be improvements upon this model, such as using feature-weighted linear stacking for the hybrid model[7]instead of the constant weights for each of the prediction models. The biggest challenge faced was not having enough information about each user and restaurant to accurately find similar ratings and that each user shows significant difference in how they rate restaurants. The feature based weighting of the hybrid model would be able to accommodate for the different users preferences. Another way to improve this model is getting rid of the randomness of matrix factorization as suggested in "Guaranteed Matrix Completion via Non-convex Factorization" by Sun, R. et al [8]. Matrix factorization is a non convex problem that doesn't guarantee the global optimum solution even after converging with stochastic gradient descent. Although we resolved that issue by starting at multiple random points and getting the minimum out of all, that doesn't guarantee the global optimum solution and it also relies on luck.

References

- [1]B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl, "Item-based collaborative filtering recommendation algorithms," in ACM WWW '01, pp. 285–295, ACM, 2001.
- [2]P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: an open architecture for collaborative filtering of netnews," in ACM CSCW '94, pp. 175–186, ACM, 1994.
- [3]Y. Koren, "Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model," Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, ACM Press, 2008, pp. 426-434.
- [4]Ekstrand, M., & Konstan, J. (2010). *Collaborative filtering recommender systems* (2nd ed., Vol. 4, pp. 81-173). Foundation and Trends.
- [5]Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. Computer, 30-37.
- [6]Matrix Factorization: A Simple Tutorial and Implementation in Python. (n.d.). Retrieved December 11, 2015, from http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python
- [7]Sill, J., Takacs, G., & Mackey, L. (2009). Feature-Weighted Linear Stacking.
- [8]Sun, R., & Luo, Z. (2014). Guaranteed matrix Completion via Non-convex Factorization.