Hybrid Recommender System: Recommending Restaurants to Users

Dharmik Ghoghari
Computer Science
University of Southern California
ghoghari@usc.edu

Lakshya Kejriwal
Computer Science
University of Southern California
lkejriwa@usc.edu

Rishab Kumar
Computer Science
University of Southern California
rishabku@usc.edu

ABSTRACT

The objective of the project is to recommend restaurants to users. We built different collaborative and content-based recommendation models and evaluated their performances using Yelp Dataset. For content-based approaches, we created business and user profiles and computed similarity between them to make personalized recommendations. We evaluated and compared the performance of the algorithms using RMSE (Root Mean Squared Error) metric. We developed a hybrid model by combining all the individual models which improved the overall RMSE.

1 Introduction

Recommendation System is one of the most important parts of e-commerce and online media companies providing online services like Amazon, YouTube, Yelp, etc. Companies are constantly looking to improve their recommendations as better recommendations result in monetary gains. Personalized recommendations based on user interest and history is essential for enhanced user experience. We plan to address that by building a recommender system for recommending new restaurants to users. We focus on predicting the rating a user will give to a restaurant he has not visited. In order to do so, we implemented various Collaborating-Filtering Algorithms, along with Content-Based algorithms. In the paper, we have compared the results of various algorithms and combined them into a Hybrid model to improve the results. Furthermore, based on the final Hybrid Model, we recommend top restaurants to a user. Looking at the skewness and sparsity of the data it is very difficult to measure the quality of the recommendations quantitatively, therefore, we do a subjective analysis of the recommendations made.

2 Dataset

The data we have used is from the Yelp Dataset Challenge Round 12 [1]. The data comprises 5,996,996 reviews for 188,593 businesses in 10 metropolitan areas from 1,518,169 users. For the purpose of our project, we only used reviews (user_id, business_id, rating, review_text) and business information (attributes and categories).

We used data from 2 cities i.e. Pittsburgh and Las Vegas because the data for Pittsburgh is small as compared to data for Las Vegas which allowed us more testing and cross-validation on our systems. We wanted to attest that the models we built would also work well when tested on large data, therefore, we chose to publish results for Las Vegas. We further filtered data to keep users with more than 20 individual reviews for Pittsburgh and more than 50 for Las Vegas. Data was randomly split into 80% for training and 20% for testing. Standard preprocessing was applied to the textual data along with removing stop words [2], rare words (only occurring once in the dataset) and lemmatizing the data. Our processed data for Pittsburgh had 50,893 reviews, 987 businesses, 3192 users and for Las Vegas had 158,880 reviews, 8309 businesses, 5272 users.

3 Methodology

3.1 Baseline

To establish a baseline to compare our recommendation models with, we first compute the average rating for each user. For each user-item pair in test data, we assign prediction as average rating of that user. In case, the user is missing in the train data, the predicted rating for that user is the average of all the ratings in the train data.

3.2 Collaborative-Filtering Based Models

Collaborative-Filtering (CF) Based Recommendation techniques have been quite successful in past giving promising results in this domain. The main idea behind these set of algorithms is that users with similar interests may like similar items. Unlike Content-Based approaches which require the content of the items, to make recommendations, these methods only require ratings various users have given to various items. [3] The following sections describe some of the methods we implemented.

3.2.1 Model-Based Collaborative Filtering

We implemented model-based collaborative filtering based on matrix factorization which can be used to discover latent features underlying the interactions between users U and items I. Our objective is to then factorize $X \in R^{U \times I}$ which is a matrix that contains all the ratings that the users have assigned to the items. Let K be the number of latent features we want to extract. We, therefore, want to find two arbitrary matrices $P \in R^{U \times K}$ and $Q \in R^{I \times K}$ such that their product estimates X. In order to get the user u's estimated rating of item i given by $\hat{r}_{ui}$, we can calculate the dot product of the two vectors corresponding to u and i.

$$\hat{r}_{ui} = p_u^T q_i \tag{1}$$

The main intuition behind this is that if the product of P and Q can approximate the known values in X then the other values computed by their product can be used as the predicted rating for any given item i by user u. To learn the matrices P and Q, we minimize the regularized squared error on the set of known ratings given by the following equation:

$$\min \sum_{(u,i)\in D} (r_{ui} - p_u^T q_i)^2 + \lambda(\|P\|^2 + \|Q\|^2) \qquad (2)$$

where D is the set of (u, i) pairs for which the rating, r_{ui} is known and λ is the regularization parameter.

We used two different approaches to minimize the above equation: stochastic gradient descent and alternating least squares [4].

3.2.1.1 Stochastic Gradient Descent (SGD)

In this approach we first initialize the two matrices P and Q with random values and for each user-item pair in the training set, the system predicts the rating $\hat{r}_{ui}$ and computes the associated error given by the square of the difference between the estimated rating ($\hat{r}_{ui}$) and the actual rating (r_{ui}).

$$e = (r_{ui} - \hat{r}_{ui})^2 \qquad (3)$$

To minimize the error, we can take the gradient of P and Q for each training example and modify the parameters by a magnitude proportional to a learning rate, α in the opposite direction of the gradient.

$$q_i = q_i + \alpha(2ep_u + \lambda q_i) \qquad (4)$$

$$p_u = p_u + \alpha(2eq_i + \lambda p_u) \qquad (5)$$

3.2.1.2 Alternating Least Squares (ALS)

In ALS minimization, we hold one set of latent feature vectors either q_i or p_u constant. By doing so, the optimization problem becomes quadratic. We can then take the derivative of equation with respect to the non-constant feature vectors. We set the derivative equal to zero and solve for the non-constant feature vectors. ALS technique rotates between fixing the q_i's and fixing the p_u's. When all p_u's are fixed, the system computes the q_i 's by solving a least-squares problem and vice versa. This continues decreasing eq. (2) at each step until convergence.

3.2.2 User-based Collaborative Filtering

In User-based collaborative filtering, each row of utility matrix $X \in R^{U \times I}$, represents a user, and each column represents an item. Each entry $X_{u,i}$ is a rating that user u has given to item i. For each user-item pair denoted by u, i whose rating is unknown, we first find all its neighbors i.e. users who have also rated item i. The similarity between u and it's neighbors is given by *Pearson Similarity* as defined in eq. (6). Rating of i for user u is computed by eq. (7). We use this because some users are easy raters as compared to others. This removes bias and bring all the users to the same level.

$$s(u,v) = \frac{\sum_{i\in I}(r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum_{i\in I}(r_{u,i} - \overline{r_u})}^2 \sqrt{\sum_{i\in I}(r_{v,i} - \overline{r_v})}^2} \qquad (6)$$

$$r_{u,i} = \overline{r_u} + \frac{\sum_{i\in I}(r_{v,i} - \overline{r_v}) * s(u,v)}{\sum_{v\in U} s(u,v)} \qquad (7)$$

3.2.3 Item-based Collaborative Filtering

Unlike user-based collaborative filtering, for each user-item pair denoted by u, i whose rating is unknown, we first find all the neighbors of i, i.e. items which are also rated by user u. The similarity is found between i and it's all neighbors using Pearson Similarity. Then rating of i is computed by taking the sum of ratings given by u to the neighbors of i weighted over their similarity as mentioned in eq. (9).

$$s(i,j) = \frac{\sum_{u\in U}(r_{i,u} - \overline{r_i})(r_{j,u} - \overline{r_j})}{\sqrt{\sum_{u\in U}(r_{i,u} - \overline{r_i})}^2 \sqrt{\sum_{u\in U}(r_{j,u} - \overline{r_j})}^2} \qquad (8)$$

$$r_{i,u} = \frac{\sum_{u\in U}(r_{u,j}) * s(i,j)}{\sum_{j\in I} s(i,j)} \qquad (9)$$

However, to compute similarity we tried various other similarity measures [5,6]. Given two vectors α and β, different similarities can be computed as following:

a. Euclidean Similarity

$$sim(\alpha,\beta) = \frac{e^D}{e^D + D} \text{ where } D = \sqrt{\textstyle\sum_{i=1}^{n}(\alpha_i - \beta_i)^2} \qquad (10)$$

b. Manhattan Similarity

$$sim(\alpha,\beta) = \frac{e^D}{e^D + D} \text{ where } D = \textstyle\sum_{i=1}^{n} |\alpha_i - \beta_i| \qquad (11)$$

c. Jaccard Similarity

$$sim(\alpha,\beta) = \frac{\sum_{i=1}^{n} \min(\alpha_i - \beta_i)}{\sum_{i=1}^{n} \max(\alpha_i - \beta_i)} \qquad (12)$$

d. Tanimoto Similarity

$$sim(\alpha,\beta) = \frac{\sum_{i=1}^{n} \alpha_i * \beta_i}{\sum_{i=1}^{n} \alpha_i^2 + \sum_{i=1}^{n} \beta_i^2 - \sum_{i=1}^{n} \alpha_i * \beta_i} \qquad (13)$$

3.3 Content Based Models

Content-based recommendation systems find similarity between users and items by building their corresponding feature vectors. The most important aspect of such models is determining those features which are transformed into vectors and then different similarity measures can be used to compute the similarity between them. We follow three different approaches to build user and item features as described below.

3.3.1 Review Based Model

In this particular model we combine all the reviews a restaurant has received into one single sentence. This single sentence is then transformed into vector using a bag of words model. Each entry in the vector is the count corresponding to a word in the sentence. We then construct the restaurant matrix of all the vectors and use two different techniques to capture the relevance between the words and a restaurant. First, we compute TF-IDF (Term Frequency-Inverse Document Frequency) of each term t (word in the sentence) in the document d (each document vector) using the following formula,

$$idf(t) = log\frac{1 + n_d}{1 + df(d, t)} + 1 \tag{14}$$

$$tf - idf(t, d) = tf(t, d) \times idf(t) \tag{15}$$

where, n_d is the total number of documents, $df(d, t)$ is the number of documents containing the term t and $tf(t, d)$ is the number of terms t in document d.

Apart from TF-IDF, we convert each document into a vector using Gensim's doc2vec implementation. The size of the vector is kept in proportion with the vocabulary of the dataset. To compute the matrix corresponding to a user U, we sum the vectors of businesses a user has reviewed in train data weighted by the rating of each review. Each vector in the user and business matrix is L2 normalized. Now we compute the cosine similarity S by simply multiplying the normalized user and business matrices.

$$B \in R^{R \times W}$$
$$U \in R^{U \times W}$$
$$S = BU^T, \qquad S \in R^{R \times U}$$

where B is the normalized matrix for restaurants, U is the normalized matrix for users, S is the similarity matrix where each value $S_{b,u} \in [0, 1]$ represents the similarity between each restaurant b and user u, R is the number of unique restaurants, W is number of unique words in TF-IDF implementation or vector size specified in doc2vec implementation.

To make recommendations, we can rank the similarities for each user in descending order and recommend the top ones. In order to validate our results, we generated predictions for each user-item pair in test data. Therefore, to transform similarities (0-1) into ratings (1-5), we can use any regressor which captures the non-linearity of the dataset.

3.3.2 Text Based Model

In this model, the restaurant matrix is computed in a similar way as done in the previous model. Instead of computing the user matrix by linearly adding the business vectors which that user has rated, here we follow a different approach. We combined all the reviews a user had given into one single sentence and followed the same approach mentioned above for business matrix. We compute the cosine similarity by multiplying the normalized user and business matrices. Similar to a review-based model we can use regressor to transform the similarity (0-1) values into ratings (1-5).

3.3.3 Category Based Model

Unlike above two models, in this model we come up with entirely new features. Each business in the dataset has categories and attributes in the following json format,

```
Old_string =
{
  "attributes": {
    "BikeParking": "False",
    "BusinessAcceptsCreditCards": "True",
    "BusinessParking": "{'garage': False, 'street':
True, 'validated': False, 'lot': False, 'valet':
False}",
    "NoiseLevel": "average",
    "RestaurantsAttire": "casual",
    "RestaurantsDelivery": "False",
    "RestaurantsGoodForGroups": "True",
    "RestaurantsPriceRange2": "2"
  },
  "categories": "Tours, Breweries, Pizza, Restaurants,
Food, Hotels & Travel"
}
```

We transform the data into a single string where we remove the terms (like Bike Parking, RestaurantsDelivery) whose corresponding value is false and keep the terms whose corresponding value is true. For each term k which has a value v we transform them into the format, k_v. In this data, no stemming or lemmatization was performed only the case of entire data was changed to lower. The transformed data is of the following format,

```
New_string =
businessacceptscreditcards businessparking_street
noiselevel_average restaurantsattire_casual
restaurantsgoodforgroups restaurantspricerange2_2
tours breweries pizza restaurants food hotels travel
```

The motivation behind building these features is that the attributes and categories capture most important information about the business and the user visiting a particular type of businesses would be interested in visiting other businesses with similar features.

For each business we have a string representing its features so similar to the review-based model (described above), we transform this sentence into vectors using TF-IDF and doc2vec. We construct the business matric and user matric similar to the method described in review-based model. To compute the similarity, we multiply the normalized matrices and a regressor can be used to transform the similarities (0-1) to ratings (1-5).

3.4 Hybrid Recommendation System

The hybrid model combines all the predictions generated from the above-mentioned models by their weighted sum. The CF model uses the ratings, while the content-based model uses reviews or categories to capture the similarity between users and businesses to predict the unknown rating. The combination of all the models can grasp different aspects of the relationship between users and businesses to give a more accurate prediction.

4 Results

In our experiments, approximately 80 percent of the data was randomly selected for training, and the remaining data were used to verify the performance of the proposed system. Since most of our models require tuning of hyper-parameters, 20% of the training set is separated as the validation set. After the optimal hyper-parameters have been found, the model is retrained on the entire training data and the performance is reported on the selected test data

4.1 Evaluation Metric

After researching different kinds of evaluation metrics used to evaluate recommendation systems, we chose to evaluate our system based on the root mean square error ($RMSE$). Given that our system generates predicted ratings $\hat{r}_{ui}$ of a user u for a business i for a test set T of user-business pairs for which the true ratings r_{ui} are known, $RMSE$ is given by

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i)\in T} (r_{ui} - \hat{r}_{ui})^2} \tag{16}$$

4.2 Experiments

4.2.1 Model Based Collaborative Filtering

The collaborative filtering by stochastic gradient descent (SGD) has three hyper-parameters namely, the learning rate α, regularization coefficient λ and the number of latent variables K. We use a grid-based search to tune the hyperparameters where we fix two values and try to optimize the third one by incrementing its value by a fixed amount in consecutive steps.

The final set of optimal hyperparameters are $\alpha = 0.005, \lambda = 0.01$ and $K = 12$. Fig 1 shows the learning curve using the optimal set of hyperparameters for Las Vegas.

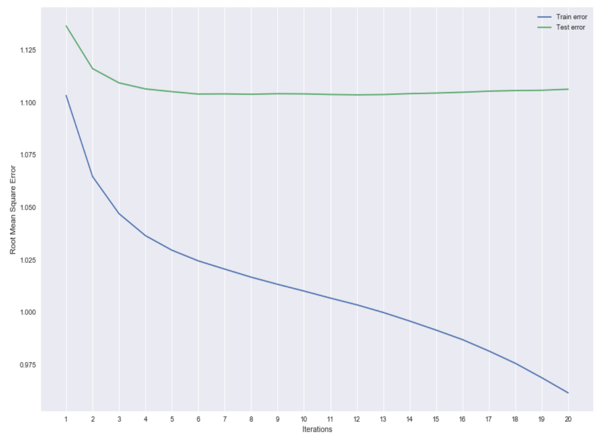

Fig 1. Learning Curve for SGD method

We use the in-built library in Spark Mllib for collaborative filtering by alternating least square (ALS). The ALS function in Spark has three hyperparameters namely, the number of iterations, i, the rank of matrix, K and the regularization coefficient, λ. For ALS, higher number of iterations usually result in lower RMSE. However, i greater than 20 is not possible due to memory constraints. Therefore, we set $i = 20$. The final set of optimal hyperparameters are $i = 20, K = 12$ and $\lambda = 0.1$. The results for model based collaborative filtering are detailed in Table 3.

4.2.1 User Based and Item Based Collaborative Filtering

One critical step in the user based and item-based collaborative filtering algorithm is to compute the similarity between users or items. We implemented different similarity measures (names) as described in Section (3.2.3) and tested them on our data set. These experiments were run on the training data and the performance is reported on the test data. Table 1 shows the experimental results.

4.2.2 Content Based

All the models described in content-based recommendation follow a similar approach, where features are extracted from review text or category depending on the model. The text is vectorized using two main techniques namely TF-IDF and Doc2Vec model. The results for both techniques are detailed in Table 2. In our experiments, the Doc2Vec model performed better than TF-IDF and therefore we employed that for our hybrid model. In order to validate our recommendations on the test dataset, we transformed our similarity values computed using the features into real-values ratings. To do this transformation, we experimented with various regressors like Linear Regression, Random Forest regression and Gradient Boosting Regression to do a regression between the similarity values and the ratings in the training set. We again used a grid-based search to tune the hyperparameters on the validation set and report the final result on the test dataset. The performance of various regressors is detailed in Table 2. The results for hybrid model are shown in Table 3. Fig 2 shows the RMSE values for different models for Pittsburgh city.

To analyze the correctness of our methodology we make recommendations for each user in the test data. Firstly, for each user in test data we generate predictions corresponding to all the businesses in the train data. We consider all the three content-based models described above in section 3.3. Each model computes a similarity matrix between a user and item, where each similarity value in the matrix conveys how closely the user and item are related. We take top K items sorted by decreasing similarity measure for each user. Then we take union of the items from all three models to get one common itemset per user in the test. Now we use the predictions we made using our hybrid model to get the rating for the business in the final itemset. We finally recommend businesses to a user in the order of prediction values.

To look at what our recommendations mean we randomly picked a user (Katie) and analyzed her results. We had 34 train points for her which she rated high. Of all the restaurants she visited 67% of the restaurants served at least pizza, Italian or alcohol, 98% were in price range 1 or 2 (from 1-4), 100% of them accepted credit cards, 70% of them were good for kids, and 65% of them had average noise level (from quiet, average, loud, and very loud). From the recommendations we made using our hybrid model described above we found 80% served Italian, pizza or alcohol, 100% were in price range 1 or 2, all of them accepted credit cards, 70% of them had average noise level. Moreover, 65% of them were also good for kids. Overall, we observed that the restaurants that we recommended were able to incorporate her preferences and recommended nearly similar restaurants which she had been to in the past.

Model	City	Similarity Measures									
		Pearson		Euclidean		Manhattan		Jaccard		Tanimoto	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
User based	Pittsburgh	**0.980**	**1.015**	0.983	1.018	0.983	1.018	1.109	1.081	0.975	1.017
	Las Vegas	**1.129**	**1.185**	1.132	1.187	1.348	1.190	1.277	1.199	1.123	1.187
Item based	Pittsburgh	0.305	0.972	**0.848**	**0.957**	0.849	0.957	0.711	1.056	1.021	0.961
	Las Vegas	0.236	1.238	**1.014**	**1.167**	1.015	1.167	0.823	1.307	1.348	1.190

Table 1. RMSEs for User and Item based Collaborative Filtering

Model	Method	TF-IDF						Doc2Vec					
	City	Linear Regression		Random forest Regression		Gradient boosting Regression		Linear Regression		Random forest Regression		Gradient boosting Regression	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Category	Pittsburgh	1.031	1.036	1.026	1.048	1.031	1.036	1.031	1.038	1.007	1.029	**0.995**	**1.028**
	Las Vegas	1.227	1.241	1.229	1.236	1.230	1.232	1.225	1.239	1.227	1.234	**1.229**	**1.231**
Review	Pittsburgh	1.029	1.041	1.029	1.039	1.031	1.037	1.032	1.037	0.981	1.007	**0.974**	**1.002**
	Las Vegas	1.231	1.227	1.231	1.226	1.233	1.226	1.230	1.226	1.239	1.224	**1.229**	**1.224**
Text	Pittsburgh	1.031	1.038	1.030	1.037	1.032	1.037	1.031	1.037	1.016	1.040	**1.032**	**1.036**
	Las Vegas	1.233	1.227	1.233	1.226	1.232	1.226	1.232	1.226	1.231	1.225	**1.230**	**1.225**

Table 2. RMSEs for Content Based models

Method	SGD		ALS	
City	Train	Test	Train	Test
Pittsburgh	**0.839**	**0.931**	0.830	0.950
Las Vegas	**0.961**	**1.106**	0.957	1.138

Table 3. RMSEs for Model based Collaborative Filtering.

Method	Baseline		Hybrid	
City	Train	Test	Train	Test
Pittsburgh	1.109	1.224	**0.813**	**0.911**
Las Vegas	1.183	1.291	**0.876**	**0.996**

Table 4. RMSEs for Hybrid model

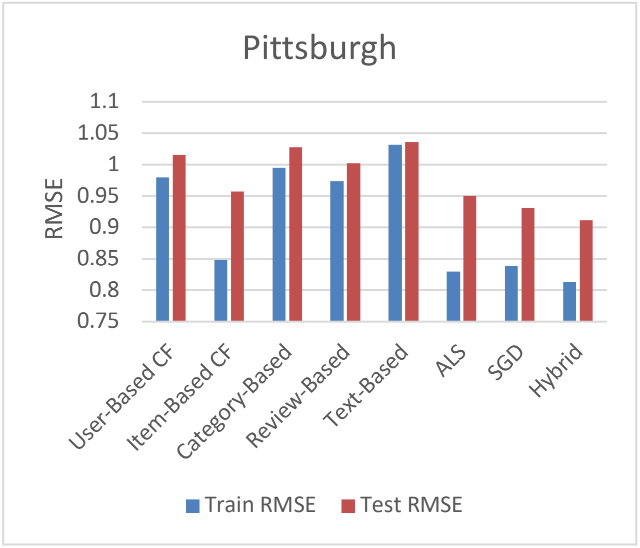

Fig 2. RMSE vs Model for Pittsburgh city

5 Conclusion

In this project, we implemented different recommendation models based on Collaborative-Filtering and Content-Based techniques. To improve our predictions further, we built a Hybrid-Model by combining various predictions using their weighted sum. The essence of our hybrid model is that each individual model in it captures some unique part of the relationship between users and businesses. The final model improves predictions as well as recommendations.

Furthermore, the recommendations can be more personalized and improved if social network of a user is also considered. Additionally, tracking the change in user preferences over time may enhance the recommendations.

APPENDIX

The code is pushed on github and can be found here: https://github.com/Lakshya-Kejriwal/Yelp_Hybrid_Recommender_System
The data is uploaded on google drive and can be found here: https://drive.google.com/drive/folders/1Q6zdfU_k1DJ3I963AlT6KQTGgZNLATR-?usp=sharing

REFERENCES

[1] Yelp Dataset Challenge www.yelp.com/dataset challenge
[2] Natural Language Toolkit (NLTK): http://www.nltk.org
[3] Yingtong Dou (2016). A survey of Collaborative Filtering algorithms for Social Recommender System. International Conference on Semantics, Knowledge and grids (12).
[4] R. Bell, Y. Koren and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no., pp. 30-37, 2009. doi:10.1109/MC.2009.263
[5] Sridhar Sondur, Amit Chigadani. "Similarity Measures for Recommender Systems: A Comparative Study", Journal for Research, Vol 2 Issue 3.
[6] KG Saranya, "Performance Comparison of Different Similarity Measures for Collaborating Filtering Technique", Indian Journal of Science and Technology, Vol 9(29)