User based collaborative Filtering

Tuesday, May 5, 2020 9:05 PM

Designing a Recommendation System

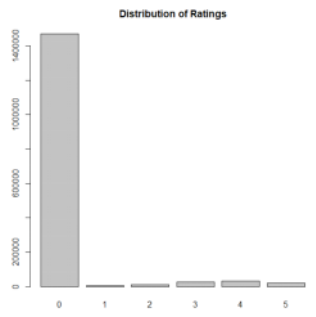
About the Dataset :

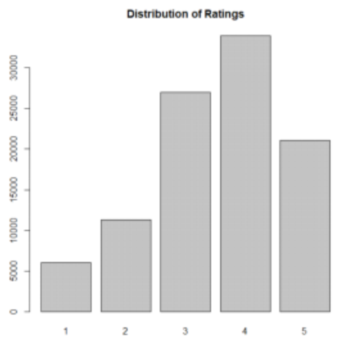
- The 100k MovieLense ratings data set. The data was collected through the MovieLense web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998.
- The data set contains about 100,000 ratings (1-5) from 943 users on 1664 movies.
- The data is in a form of realRatingMatrix of dimension 943*1664. Each row corresponds to a user and each column corresponds to a movie and each value to a rating
- The ratings are integers in the range 0-5

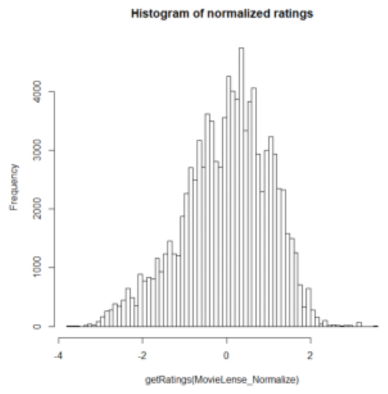
Exploratory Data Analysis

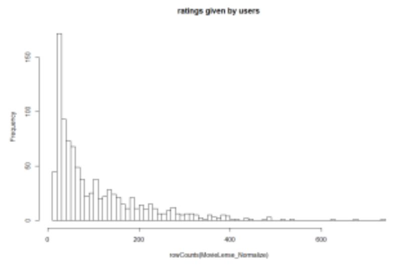
Observations :

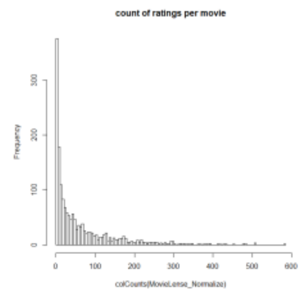
1. Most of the ratings are blank as seen from the distribution of ratings. This makes sense as all the users have not seen all the movies and rated them on the platform. We would be predicting these unknown ratings using User based collaborative filtering
2. Let's remove the rating of 0 from the dataset and visualize the ratings as a rating of zero means a missing value. We see that the low scores are lesser. The majority of the movies are scored a 3 or higher
3. We shall look at the normalized ratings in order to remove the rating bias by individuals. If we look at the histogram of normalized ratings, we see that the range of negative rating is more compared to positive. This makes sense as per general human behavior which is to definitely leave rating/comments when we dislike something.
4. Also we see that most users have not given any rating on the platform or have rated very few movies. Thus normalizing the ratings is required so that out recommendation model does not have lot of bias
5. We see that there are some movies which do not have any ratings assigned to them, user based collaborative filtering will help to assign ratings for these movies. Also most movies have very few ratings assigned. Some movies which are super popular have lots of ratings assigned to them hinting at their popularity among masses.







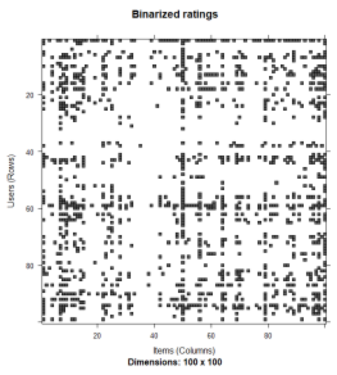


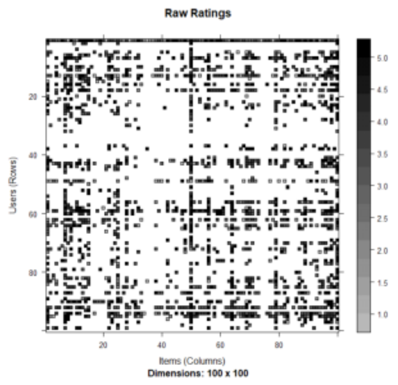


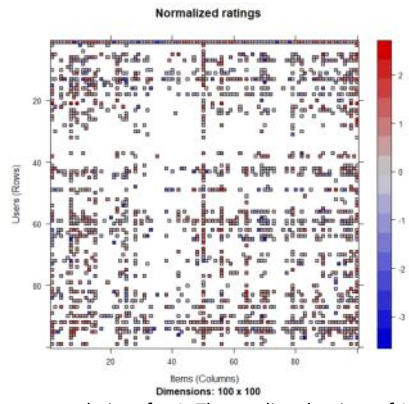
Normalized and Binarized Matrices

1. Ratings are normalized to remove rating bias by individuals and the ratings of 100 users and 100 movies are plotted.
2. Normalization can be done using the mean score for every user and then subtracting this mean score from all the ratings this user has provided. This is called mean normalization
3. Another way to normalize is using the Z-score value for each rating given by user
4. We shall perform experimentations to find the best normalizing method out of the two for our dataset
5. The raw ratings on the scale of 0 to 1 are shown where a rating of 0 indicates unrated movies by individuals using

6. The binary rating matrix is created. All ratings greater than 4 are set as 1 and others are 0. This matrix is sparser than the raw matrix
7. We will be using the normalized matrix for building the recommendation system. Below shown normalized matrix is based on mean centering method.







Building a Recommendation System :

User Based Collaborative Filtering : We will use the ratings provided by similar users to a target user A to make recommendations for A. The predicted ratings of A will be computed as the weighted average values of these “peer group” ratings for each item.

Approach

1. Now we shall build a recommender system using a User based approach. Our aim is to predict the missing ratings of users in the dataset.
2. We will use the normalized ratings for computation. The first step is to identify 'k' similar users for every other user. We shall use Cosine similarity and K-nearest neighbors approach
3. After we have identified 'k' similar users, we will compute the missing ratings for each user by taking the weighted average ratings of the 'k' nearest users as per the similarity score

We will try to find out the optimum value of the Hyperparameters for building the recommendation model using Test - Train split approach.
The Hyperparameters are :

1. **Number of similar users to be considered**
2. **Similarity Measure : Pearson Correlation, Cosing Distance, Jaccard distance**
3. **Normalizing Method**

Evaluation of the Recommendation System :

1. **<u>Preparing the Dataset</u>**
   We shall prepare the train and test dataset in two ways to evaluate the recommendation system :
   i. By splitting the data into train and test set
   ii. Using k-fold cross validation

2. **<u>Evaluation Metrics</u>**

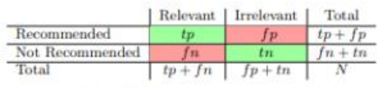
We want to measure the amount of correct and incorrect classifications as relevant and irrelevant items made by our recommendation system.
We will ignore the exact rating and focus on classifying the movies as relevant or irrelevant for a user.

Precision or true positive accuracy is calculated as ratio of recommended items that are relevant to the total number of recommended items.

$$\text{precision} = \text{tpa} = \frac{tp}{tp + fp}$$

Recall or true positive rate is calculated as the ratio of recommended items that are relevant to the number of relevant items. This is the probability that a relevant item is recommended .

$$\text{recall} = \text{tpr} = \frac{tp}{tp + fn}$$



	Relevant	Irrelevant	Total
Recommended	tp	fp	$tp + fp$
Not Recommended	fn	tn	$fn + tn$
Total	$tp + fn$	$fp + tn$	N

Table 1: A contingency table or confusion matrix that accumulates the numbers of true/false positive/negative recommendations.

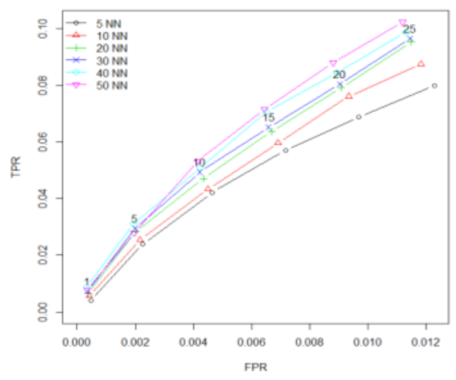
3. **Predict Ratings using the Recommender Model**

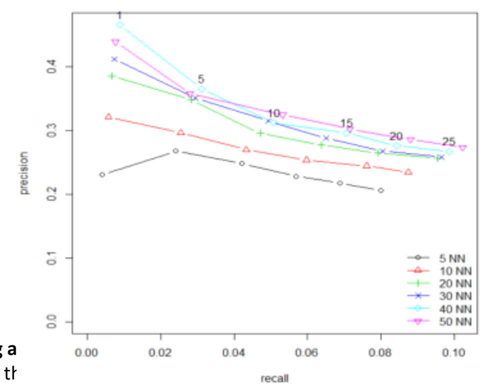
For each new user, we will first Measure how similar each user is to the new one. We will experiment with different similarity measures. For each user, we will Identify the most similar users. The option are - Take account of the top k users (k-nearest_neighbors) and Take account of the users whose similarity is above a defined threshold. Rate the movies rated by the most similar users. The rating is the average rating among similar users and we will use the weighted average rating.

4. Evaluation of the Recommender Model

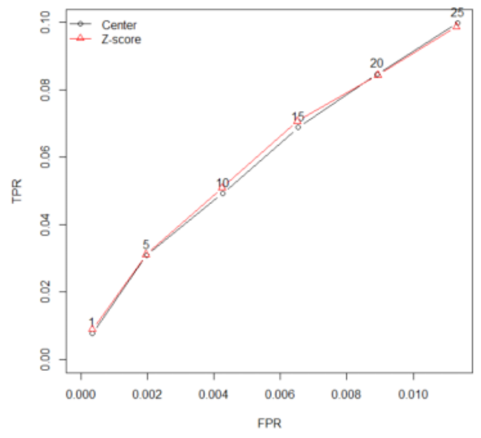
We will use the evaluation functions in recommender lab package to measure our performance. We will create an evaluation sche me that is then applied to our rating matrix. We will be varying the parameters values to attempt to find the best performing model.

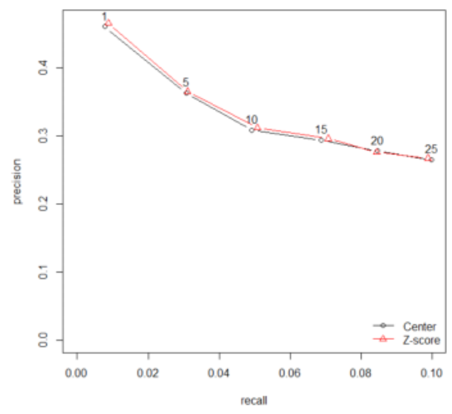
Experiment 1: Varying the Neighborhood Size.
We will look at 6 neighborhood sizes for the recommender model (5,10,20,30,40,50)
Inference :
From the ROC curve and the Precision Recall curve we see that there is little difference between taking 40 and 50 nearest nei ghbors for all number of recommendations. Let's choose 40 neighbors as our best candidate and use this value to check the values of other parameters. The root mean squared error on fitting a recommendation model with 40 nearest neighbors is 1.144



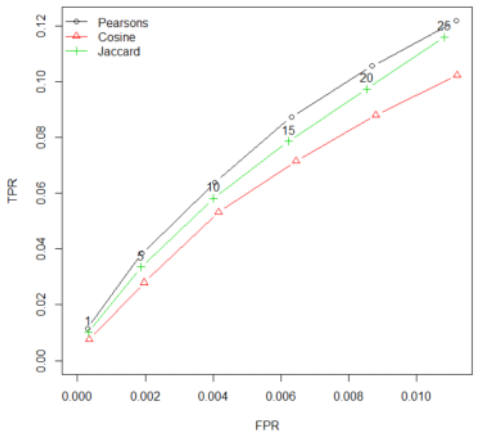


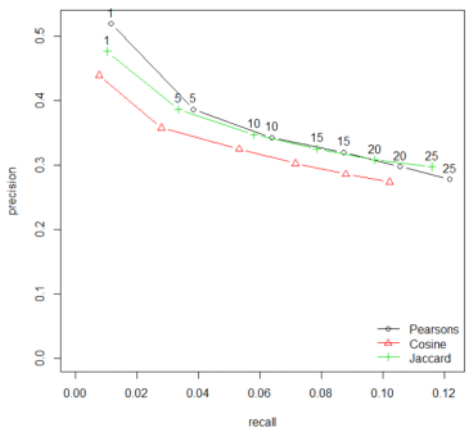
Experiment 2: Using Normalizing Methods - Mean centering a
We will have to normalize the data to reduce the user bias in th data works well for our dataset.
Inference :
Z- score does a slight better job at the different recommendation levels. We shall use Z-score for normalizing the data





Experiment 3: Distance Methods- Pearson, Cosine and Jaccard distance
Building the evaluation models and looking at ROC curve and precision/recall curve, we can see that Pearson performs better than the other similarity measures. The RMSE is 1.12





Inference :

Through the experimentation we found that the best performance of the model is when we are using the following parameters

a. **Use the 40 nearest neighbors**
b. **Use z-score for normalizing the ratings**
c. **Use Pearson's similarity score**

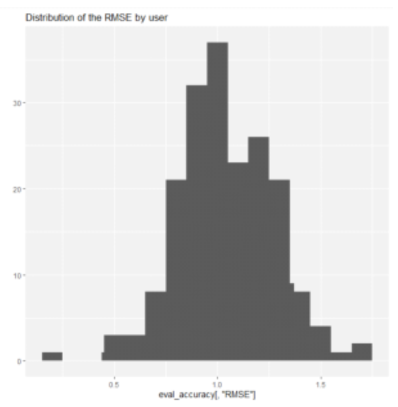
Test the performance of the Recommender System using Cross-Validation approach

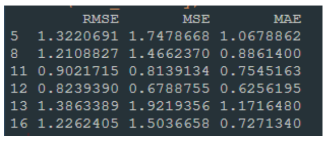
Step 1: Building an evaluation scheme for the cross-validation method.

a) We will create a k-fold cross-validation scheme and redefine the evaluation sets. The data will be split into k parts and in each run k-1 parts are used for training and the remaining parts are used for testing
b) After all the k runs, each part was used as the test set exactly once
c) We will use k as 5 i.e 5 times we will be running the evaluation
d) We can also set the number of items given for each observation. We will use 15 items at each time.
e) Threshold for good rating will be set as 4. Any rating equal to or greater than 4 will be considered a good rating for a user

Step 2 : Evaluating the Ratings

a) We have five sets of same size - 752
b) For each fold, we will use the 4 set for training the recommender model and 1 part to test the model
c) We will compute the accuracy measure for each user. We will use Root Mean squared error.
d) The root mean squared error is in the range 0.8 to 1.38



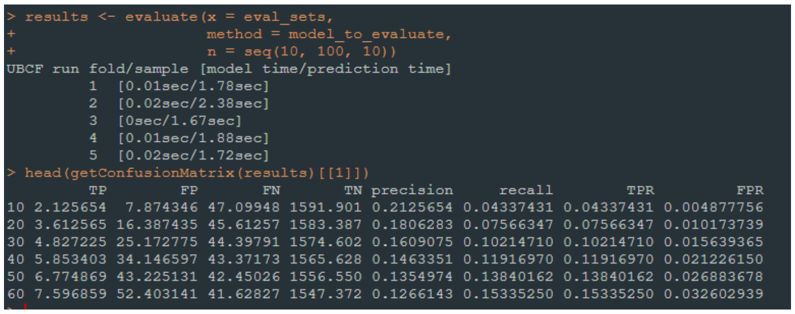


```
        RMSE       MSE       MAE
5  1.3220691 1.7478668 1.0678862
8  1.2108827 1.4662370 0.8861400
11 0.9021715 0.8139134 0.7545163
12 0.8239390 0.6788755 0.6256195
13 1.3863389 1.9219356 1.1716480
16 1.2262405 1.5036658 0.7271340
```

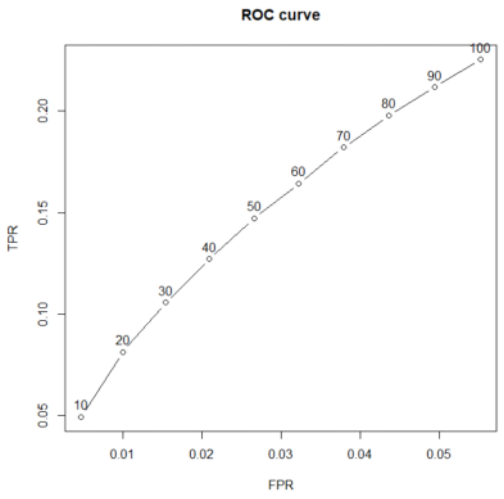
Step 2 : Evaluating the Recommendations

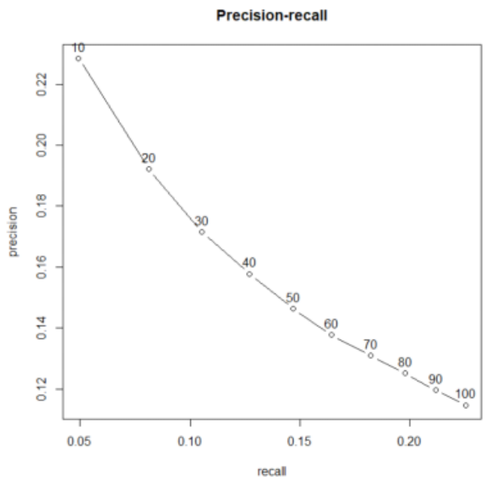
a) We can also compare recommendations with the movies having positive ratings.
b) We will use the inbuilt evaluate function in the recommenderlab library. The function evaluates the recommender performance depending on the number n of items to recommend to each user

c) We will use recommend the following number of movies - 10,20,30,40,50,60,70,80,90,and 100 and evaluate the recommendations
d) Finally we will plot ROC and Precision/Recall curve
e) Higher the percentage of movies recommended, lower is the precision rate and higher the recall rate
f) We see from the ROC curve that as we increase the number of recommendations the True Positive and the False positive rates both are increasing steadily



```
> results <- evaluate(x = eval_sets,
+                     method = model_to_evaluate,
+                     n = seq(10, 100, 10))
UBCF run fold/sample [model time/prediction time]
         1  [0.01sec/1.78sec]
         2  [0.02sec/2.38sec]
         3  [0sec/1.67sec]
         4  [0.01sec/1.88sec]
         5  [0.02sec/1.72sec]
> head(getConfusionMatrix(results)[[1]])
         TP        FP       FN       TN precision     recall        TPR         FPR
10 2.125654  7.874346 47.09948 1591.901 0.2125654 0.04337431 0.04337431 0.004877756
20 3.612565 16.387435 45.61257 1583.387 0.1806283 0.07566347 0.07566347 0.010173739
30 4.827225 25.172775 44.39791 1574.602 0.1609075 0.10214710 0.10214710 0.015639365
40 5.853403 34.146597 43.37173 1565.628 0.1463351 0.11916970 0.11916970 0.021226150
50 6.774869 43.225131 42.45026 1556.550 0.1354974 0.13840162 0.13840162 0.026883678
60 7.596859 52.403141 41.62827 1547.372 0.1266143 0.15335250 0.15335250 0.032602939
```


ROC curve
TPR
FPR


Precision-recall
precision
recall