Case Study 4.1 - Movies

Note: If you close this notebook at any time, you will have to run all cells again upon re-opening it.

BEGINNER R

As this is a beginner version, we include a lot of code here to help you along the way.

Identification Information

In [ ]:

```
# YOUR NAME              = Sewar Khalifeh
# YOUR MITX PRO USERNAME = Sewarkhalifeh98
# YOUR MITX PRO E-MAIL   = sewarkhalifeh@gmail.com
```

Setup

Run these cells to install all the packages you need to complete the remainder of the case study. This may take a few minutes, so please be patient.

If you see red messages, don't worry! See **this FAQ**.

In [12]:

```
install.packages('recommenderlab')
print('Installation successful!')
```

```
Installing package into ‘/home/nbuser/R’
(as ‘lib’ is unspecified)
```

```
[1] "Installation successful!"
```

Import

Import the required tools into the notebook.

In [13]:

```
library('recommenderlab')
print('Import successful!')
```

```
[1] "Import successful!"
```

Data

Load the MovieLens data.

In [14]:

```
data = read.table('u.data.txt')
colnames(data) = c("user_id", "item_id", "rating", "timestamp")
```

```
data = data[ , -which(names(data) %in% c("timestamp"))]
print('Data loading successful!')
```

```
[1] "Data loading successful!"
```

We also want to get a sense of what the data looks like. Let's create a histogram of all the ratings we have in the dataset.

In [15]:

```
hist(data$rating)
print('Histogram generation successful!')
```

```
[1] "Histogram generation successful!"
```

We must now subset the data into train and test sets to use for our different models. Here, we split the entire dataset into 70% train and 30% test.

In [16]:

```
# First, we convert our data type to a recommenderlab compatible type
data_new = as(data, "realRatingMatrix")

# Now, we apply a split of 0.7 train, 0.3 test
# You can ignore the given parameter
data_split = evaluationScheme(data_new, method="split", train=0.7, goodRating=3, given=-1)
train = getData(data_split, "train")
test_X = getData(data_split, "known")
test_Y = getData(data_split, "unknown")

print('Data subset successful!')
```

```
[1] "Data subset successful!"
```

QUESTION 1: DATA ANALYSIS

Describe the dataset. How many ratings are in the dataset? How would you describe the distribution of ratings? Is there anything else we should observe? Make sure the histogram is visible in the notebook.

According to the Histogram above, almost 5000 ratings are giving "1 star",10000 ratings are for "2 stars",25000 ratings for "3 stars", 35000 (that's the most occurring rating) are for "4 stars" & finally "20000" for "5 stars", by summing all of these ratings we can conclude that there are 95000 (approx:100k) ratings in this dataset, we can observe that the most frequent ratings are for (3 & 4) stars,& the least frequent rating was "1 star" , it seems to be "central tendency" distribution were most of the ratings are in the middle & other ratings are laying before & after it.

Model 1: Random

In [17]:

```
# Create model object on our training set
model_random = Recommender(train, method="RANDOM")
print('Model creation successful!')
```

```
[1] "Model creation successful!"
```

In [18]:

```
# Evaluate RMSE on test set
predicted_random = predict(model_random, test_X, type="ratings")
error_random = calcPredictionAccuracy(predicted_random, test_Y)
error_random
```

RMSE
1.47086370084792
MSE
2.16344002647204
MAE
1.18291758795689

Model 2: User-Based Collaborative Filtering

In [19]:

```
# Create model object on our training set
model_user = Recommender(train, method="UBCF")
print('Model creation successful!')
```

```
[1] "Model creation successful!"
```

In [20]:

```
# Evaluate RMSE on test set
predicted_user = predict(model_user, test_X, type="ratings")
error_user = calcPredictionAccuracy(predicted_user, test_Y)
error_user
```

RMSE
1.09852549449401
MSE
1.20675826205331
MAE
0.882107914220877

Model 3: Item-Based Collaborative Filtering

In [27]:

```
# Create model object on our training set
# Note that this may take a while to train
model_item = Recommender(train, method="IBCF")
print('Model creation successful!')
```

```
[1] "Model creation successful!"
```

In [28]:

```
# Evaluate RMSE on test set
predicted_item = predict(model_item, test_X, type="ratings")
error_item = calcPredictionAccuracy(predicted_item, test_Y)
error_item
```

RMSE
1.33970794833474
MSE
1.79481738683128
MAE
0.91820987654321

QUESTION 2: COLLABORATIVE FILTERING MODELS

Compare the results from the user-user and item-item models. How do they compare to each other? How do they compare to our original "random" model? Can you provide any intuition as to why the results came out the way they did?

First of all, The user-user model recommends items by finding users with the same interests, it is harder to scale because the users interests keep changing, The item-item is a bit easier because items are fixed with their features,it makes recommendations based on the similarity between items & the random recommendation algorithm,does not utilize any data, which uses any of the ratings at random and does a prediction. let's take a look at each of the RMSE values for each model,the item-item model has the RMSE value of 1.33970794833474 which is almost 1.33 & the user-user model with RMSE of 1.09852549449401 which is almost 1.098, we can observe that the user-user model has less error value (RMSE) than the item-item model because it's more personalized for each user,this model focuses on the dynamic nature of users and how each one specifically prefers what type of movies, in the item based model we look at the similarity of the items and it's not always a user likes a movie which has almost the same features and content as a one they liked before,both of these models came to a result better than the random model which got the highest RMSE value: 1.47086370084792 (which is almost 1.47) & that was expected due to the concept of the random model which does not rely on any data and predicts randomly, hence a larger amount of error may occur in this case.

Model 4: Matrix Factorization

In [29]:

```
# Create model object on our training set
model_matrix = Recommender(train, method="SVD")
print('Model creation successful!')
```

```
[1] "Model creation successful!"
```

In [30]:

```
# Evaluate RMSE on test set
predicted_matrix = predict(model_matrix, test_X, type="ratings")
error_matrix = calcPredictionAccuracy(predicted_matrix, test_Y)
error_matrix
```

RMSE
1.08627858700783
MSE
1.18000116859172
MAE
0.86908498854798

QUESTION 3: MATRIX FACTORIZATION MODEL

The matrix factorization model is different from the collaborative filtering models. Briefly describe this difference. Also, compare the RMSE again. Does it improve? Can you offer any reasoning as to why that might be?

No code is required for this question.

Yes, It improved with an RMSE of 1.08627858700783 which is almost 1.08 , the matrix factorization model improved all of the above models based on how it works, the matrix factorization transforms both items and users into vectors by factors which is given by the item ratings,when the item vector and the user vector has a high correspondence then this will lead to a recommendation that is more specific and that will lead to very accurate personalized recommendations with less RMSE,and it's more flexible to apply to the real-life events.

Precision and Recall @ `k`

We now want to compute the precision and recall for 2 values of `k`: 5 and 10. The `recommenderlab` library makes this super easy!

Note that some of these values may take some time (a few minutes) to compute.

QUESTION 4: PRECISION/RECALL

Compute the precision and recall, for each of the 4 models, at `k` = 5 and 10. This is 2 x 2 x 4 = 16 numerical values. Do you note anything interesting about these values? Anything differerent from the RMSE values you computed above?

We can use the precision, recall to evaluate the performance of the recommender system. The precision of recommendation describes the proportion of items that users prefer, and the recall describes the proportion of the user favorite items that are not missed. Precision is "how useful the results are", and recall is "how complete the results are". for k=5 In the random model ,precision=0.0002120141 ,recall=0.01351351 In the item based model,precision=0.001413428 ,recall=0.009009009 In the user-based model,precision=0.002826855 ,recall=0.01801802 In the matrix factorization model,precision=0.009187279 ,recall=0.05855856 The recall value in the matrix factorization model was the biggest value from other models because as we know recall is the fraction of the relevant data that are successfully retrieved, And in this model, the results are the most accurate. The precision value in the matrix factorization model was the biggest value from other models=0.009187279 because precision shows how useful the search results are, And in this model, the results are the most useful and precise. and we can notice how it's related to the RMSE values above in terms of showing which model is more precise and which returns the best possible prediction based on certain calculations taking into consideration the error margins which is the best when it is the least,and here when the precision is higher it shows that the results are more accurate and same for the recall which shows that the results are complete.

In [31]:

```
results_random = evaluate(data_split, method="RANDOM", type="topNList", n=c(5, 10))
results_random@results
```

```
RANDOM run fold/sample [model time/prediction time]
  1  [0.014sec/4.005sec]


[[1]]
An object of class "confusionMatrix"
Slot "cm":
           TP       FP        FN       TN   precision     recall        TPR
5  0.01060071 4.989399 0.7738516 1677.226 0.002120141 0.01351351 0.01351351
10 0.01413428 9.985866 0.7703180 1672.230 0.001413428 0.01801802 0.01801802
           FPR
5  0.002965968
10 0.005936139

Slot "model":
NULL
```

In [32]:

```
results_user = evaluate(data_split, method="UBCF", type="topNList", n=c(5, 10))
results_user@results
```

```
UBCF run fold/sample [model time/prediction time]
  1  [0.046sec/12.264sec]


[[1]]
An object of class "confusionMatrix"
Slot "cm":
           TP       FP        FN       TN   precision     recall        TPR
5  0.01413428 4.985866 0.7703180 1677.230 0.002826855 0.01801802 0.01801802
10 0.02120141 9.978799 0.7632509 1672.237 0.002120141 0.02702703 0.02702703
           FPR
5  0.002963868
10 0.005931937

Slot "model":
NULL
```

In [33]:

```
results_item = evaluate(data_split, method="IBCF", type="topNList", n=c(5, 10))
results_item@results
```

```
IBCF run fold/sample [model time/prediction time]
```

```
  1   [101.015sec/1.16sec]

[[1]]
An object of class "confusionMatrix"
Slot "cm":
            TP       FP        FN       TN   precision      recall         TPR
5  0.007067138 4.992933 0.7773852 1677.223 0.001413428 0.009009009 0.009009009
10 0.010600707 9.989399 0.7738516 1672.226 0.001060071 0.013513514 0.013513514
           FPR
5  0.002968069
10 0.005938239

Slot "model":
NULL
```

In [34]:

```
results_matrix = evaluate(data_split, method="SVD", type="topNList", n=c(5, 10))
results_matrix@results
```

```
SVD run fold/sample [model time/prediction time]
  1   [0.649sec/3.283sec]


[[1]]
An object of class "confusionMatrix"
Slot "cm":
            TP       FP        FN       TN   precision     recall        TPR
5  0.04593640 4.954064 0.7385159 1677.261 0.009187279 0.05855856 0.05855856
10 0.06007067 9.939929 0.7243816 1672.276 0.006007067 0.07657658 0.07657658
           FPR
5  0.002944960
10 0.005908828

Slot "model":
NULL
```

Top-n Predictions

Finally, we can see what some of the actual movie ratings are for particular users, as outputs of our model.

QUESTION 5: TOP N PREDICTIONS

Do the top n predictions that you received make sense? What is the rating value (1-5) of these predictions? How could you use these predictions in the real-world if you were trying to build a generic content recommender system for a company?

A recommender system usually recommends a list of items for users. They are generally arranged in horizontal or vertical. Users generally only care about the front parts of several items and few users will care about the back parts of the items, such as the items listed on page 20. This way of recommendation is called Top-n recommendation. In this example we took the first 10 recommendations , in the random model all of the recommendations had rating "5" exactly,in the item based model it improved a bit were the recommendations had rating between (4 and 4.5), the user based model was among the best models in terms of accuracy, as you can see, the recommendations got more personalized ratings (3.352174 3.346068 3.253959 3.239554 3.233603) for example,the best model (as expected) was the matrix factorization model which had more precise results, in conclusion we can see how the results we calculated before as the RMSE, precision and the recall for each model gave us a pretty good hint to guess which model has the best performance and give the results we're aiming for.

In [35]:

```
# Pick a user and value of n
user_id = 5
n = 10
print('Variable update successful!')
```

```
[1] "Variable update successful!"
```

In [36]:

```
# Repeat this process for the all models
for (model in c(model_random, model_user, model_item, model_matrix)) {
    # Get recs for model
    user_recs = predict(model, test_X, n=n, type="ratings")

    # Print the top n recs for that user
    print(model)
    print(user_recs@data[user_id,][order(-user_recs@data[user_id,])][c(1:n)])
}

print('Top N predictions successful!')
```

```
Recommender of type ‘RANDOM’ for ‘realRatingMatrix’
learned using 660 users.
 22  24  50  53  57  75  77  81  92 109
  5   5   5   5   5   5   5   5   5   5
Recommender of type ‘UBCF’ for ‘realRatingMatrix’
learned using 660 users.
     288      333      258      347       50      191      181      185
3.352174 3.346068 3.253959 3.239554 3.233603 3.225148 3.206410 3.205814
     315      480
3.205459 3.199933
Recommender of type ‘IBCF’ for ‘realRatingMatrix’
learned using 660 users.
1496 1589  351  691  779  822  904 1085 1111 1173
 5.0  4.5  4.0  4.0  4.0  4.0  4.0  4.0  4.0  4.0
Recommender of type ‘SVD’ for ‘realRatingMatrix’
learned using 660 users.
      50      168      172      268      181      173      288      184
3.330113 3.287511 3.279936 3.244969 3.225602 3.223932 3.214909 3.210598
     122       42
3.196490 3.195227
[1] "Top N predictions successful!"
```

Great job! Now, make sure you check out the **Conclusion** section of the instruction manual to wrap up this case study properly.