# Capstone MovieLens

Jozsef Toth

2024-05-07

## Introduction

The objective of this project is to create a movie recommendation system based on the MovieLens dataset. The input dataset (edx) contains around 9 million rows (ratings), but the real life dataset can be much larger which makes it challenging to apply a fast and complex machine learning approach. Before model development, the input data set is divided into 2 parts: training and testing data in the ratio of 90% vs 10%. Generally, I will follow the methodology in the book and first calculate the effect of the various properties (movie, user, genres) and then perform a regularization to penalize observations that have very few observations. At the end I will cut the prediction interval to fit better for the rating distribution. The evaluation of the developed models will be done with RMSE (Root Mean Square Error). The validation dataset (final_holdout_test) contains around 1 million ratings and will be only used for the final evaluation of the model. This is my last course for HarvardX's Data Science Professional Certificate program, and I hope I can apply what I've learned in the series.

Original dataset : https://grouplens.org/datasets/movielens/10m/

## Methods and model development

Before we go on let me to introduce two helper functions to evaluate our model and than create the training and test sets. The Evaluate_Model function will be used to calculate RMSE and accuracy. Accuracy is not in the focus of our model development, but still useful to have it. Accuracy will be calculated for half rounded numbers (0,0.5,1,1.5,etc) what are generated by the Round_Rating function.

```
#Helper function to calculate RMSE and accuracy
#For accuracy we use the half rounded numbers (0,0.5,1,1.5,2,2.5, etc)
#For RMSE we use the caret function caret::RMSE
Evaluate_Model <- function(y_hat,model_name="Undefined",y=test_set$rating) {
  accuracy <- mean(Round_Rating(y_hat) == y)
  rmse <- RMSE(y_hat,y)
  data.frame(model_name,rmse,accuracy)
}

#Helper function to half round number to closest possible rating
#Possible rating values : (0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5)
Round_Rating <- function(y) {
  y_rounded <- round(y*2)/2
  y_rounded[y_rounded<0] <- 0
  y_rounded[y_rounded>5] <- 5
  y_rounded
}
```

The edx data set contains around 9 million rows. We divide this dataset into 2 parts: training and testing datasets with the ratio of 90 percent vs 10 percent, by using the following code:

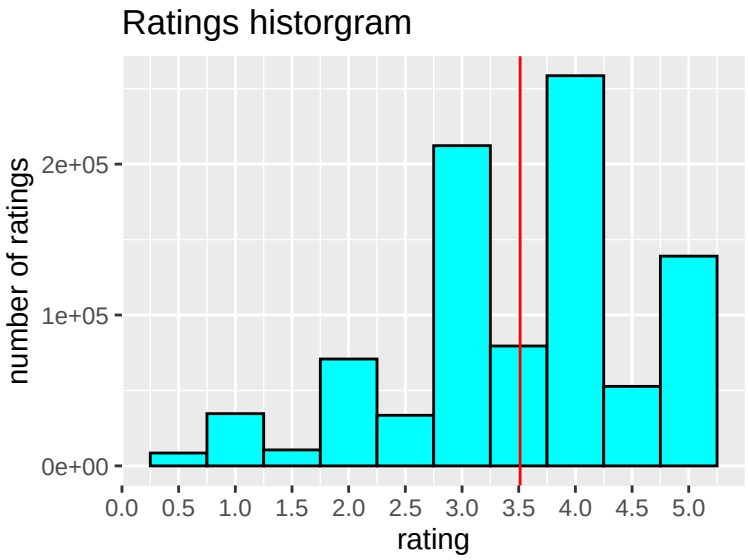
```
#Create train and test sets
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

We will use the training dataset (rownum=8100048) to explore the general statistics of the variables and develop our model. The test dataset (rownum=900007) will be used for evaluating our actual model. We have the following columns:

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

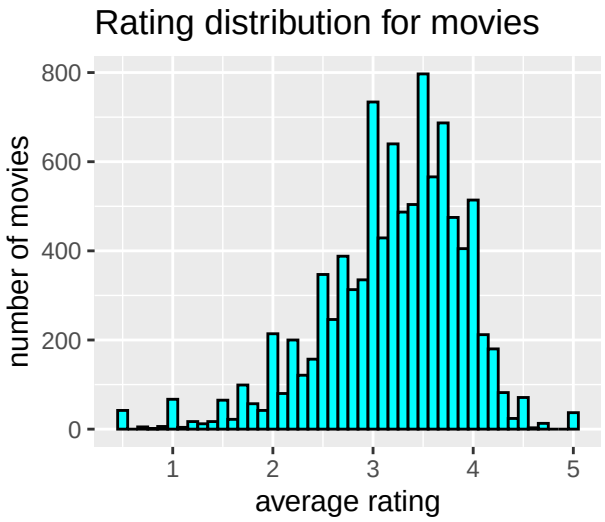
Rating column is our target what we want to predict. Title is not really useful for the prediction because it is different for all movies. Timestamp is an interesting variable, but not easy to include as a predictor. So we have three column what can be easily used as a predictor for rating :
"userId" - Different users like different movies
"movieId" - Some movies are more popular than others
"genres" - The genre of the movie can affect the rating of the movie

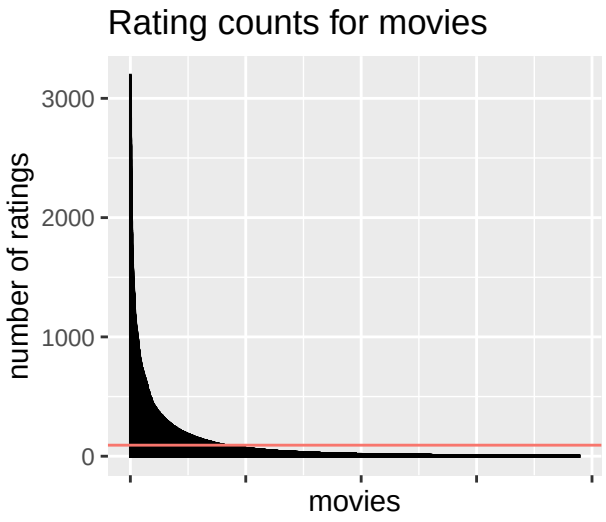
If we look at the distribution of the ratings, we will see some evidence:




| Description | min | max | mean | median |
|---|---|---|---|---|
| Ratings in general | 0.5 | 5 | 3.51 | 4 |

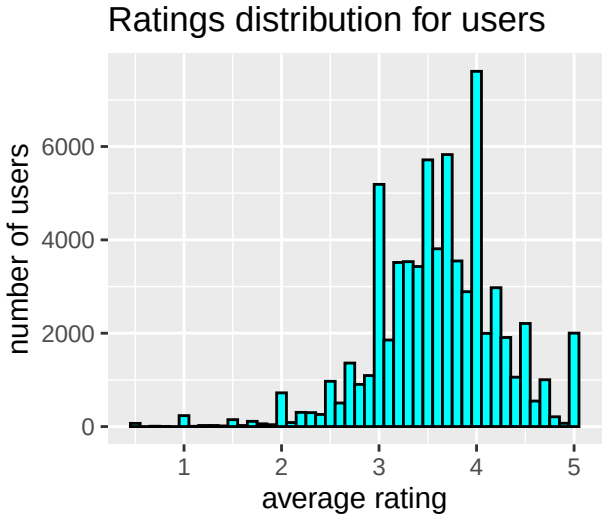
The rating range is from 0.5 to 5. The median rating has the most rates, the average rating is in the center of the distribution. An interesting finding is that users are more likely to give whole numbers (1, 2, etc.) than half-round numbers (0.5, 1.5, etc.). Now explore movie, user and genre ratings in details:

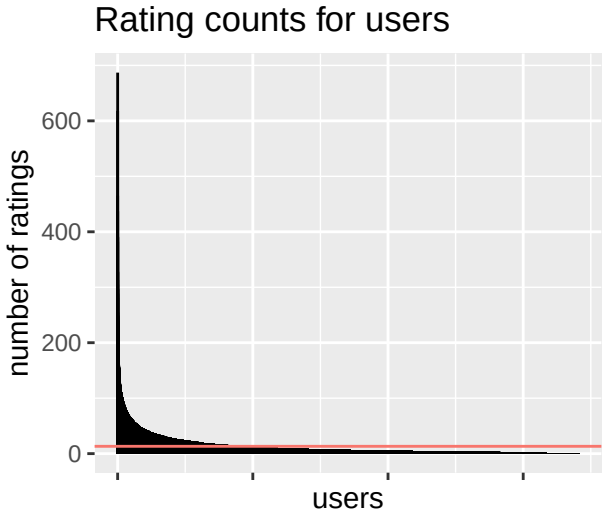



| Description | min | max | mean | median |
|---|---|---|---|---|
| Rating counts for movies | 1 | 3194 | 92.62 | 16 |

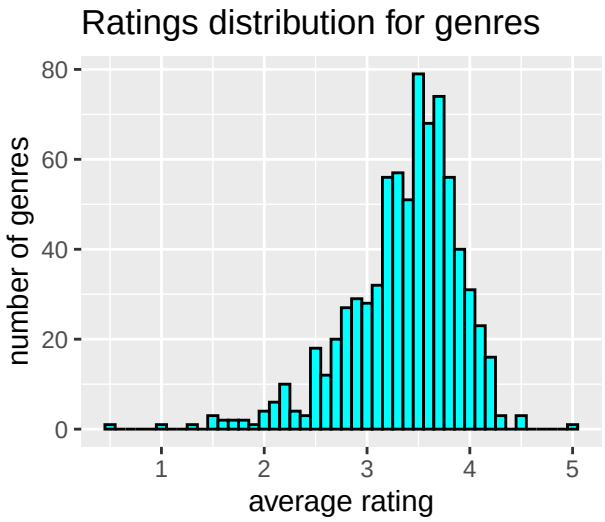
The average ratings shows similar distribution like before. We can see that there is a huge difference between the rating counts of the movies. There are movies that have only a few ratings and others that have over 1000. We know this is normal because famous movies can have more reviewers than others.

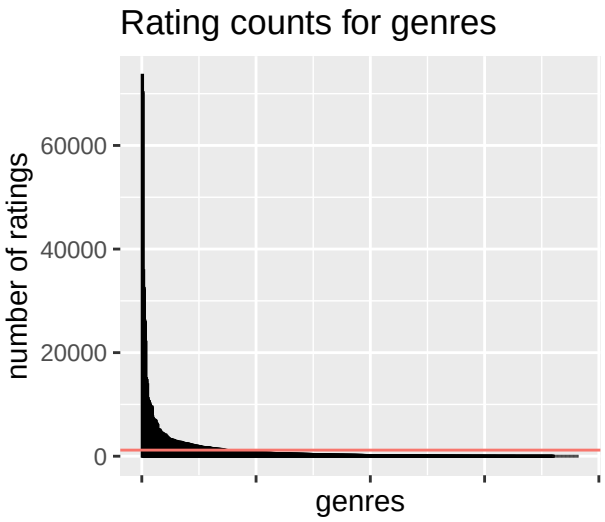



| Description | min | max | mean | median |
|---|---|---|---|---|
| Rating counts for users | 1 | 687 | 13.2 | 7 |

We can see very similar charts for users. There are very active users with more than 600 reviews and some only with a few.




| Description | min | max | mean | median |
|---|---|---|---|---|
| Rating counts for genres | 1 | 73571 | 1178.02 | 170 |

Genres show approximately the same distribution. Users like different movies, for example I like adventure and sci-fi movies. Users can give better ratings for the types of movies they like.

### Preparation of the model development

First, let's examine the difference between choosing the median (4) or the average rating (3.51) as our predictor.

### Model 1 : Median rating alone

```
median_rating <- median(train_set$rating)
m1_eval <- Evaluate_Model(median_rating,"median alone")
m1_eval
```

```
##     model_name     rmse  accuracy
## 1 median alone 1.166763 0.2872967
```

### Model 2 : Mean rating alone

```
mean_rating = mean(train_set$rating)
m2_eval <- Evaluate_Model(mean_rating,"mean alone")
m2_eval
```

```
##   model_name     rmse   accuracy
## 1 mean alone 1.060056 0.08826265
```

Median rating is more accurate, what is not a surprise, because it is an integer matching the most common rating in the test set. Mean is a better choice for RMSE optimization what is our main objective. We will continue the model development with checking the effect of the three selected variable (userId,movieId,genres).

### Model 3 : Mean + User effect

```
user_effect <- train_set %>%
  group_by(userId) %>%
  summarize(u_e = mean(rating-mean_rating),u_n=n())

predicted_ratings <- test_set %>%
  left_join(user_effect,by=c("userId")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,u_e,na.rm = TRUE)) %>%
  pull(y)
m3_eval <- Evaluate_Model(predicted_ratings,"mean+user effect")
m3_eval
```

```
##         model_name      rmse  accuracy
## 1 mean+user effect 0.9777083 0.2051206
```

### Model 4 : Mean + Movie effect

```
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(m_e = mean(rating-mean_rating),m_n=n())

predicted_ratings <- test_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,m_e,na.rm = TRUE)) %>%
  pull(y)
m4_eval <- Evaluate_Model(predicted_ratings,"mean+movie effect")
m4_eval
```

```
##          model_name      rmse  accuracy
## 1 mean+movie effect 0.9429666 0.2251638
```

### Model 5 : Mean + genres effect

```
genre_effect <- train_set %>%
  group_by(genres) %>%
  summarize(g_e = mean(rating-mean_rating),g_n=n())

predicted_ratings <- test_set %>%
  left_join(genre_effect,by=c("genres")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,g_e,na.rm = TRUE)) %>%
  pull(y)
m5_eval <- Evaluate_Model(predicted_ratings,"mean+genres effect")
m5_eval
```

```
##          model_name     rmse  accuracy
## 1 mean+genres effect 1.017505 0.1681687
```

As we see the different models has different effect to RMSE. The order of possible effects will be chosen by RMSE decrease (larger decrease used first), so we will combine the models in the order of : movie, user and genres.

### Model 6 : Mean rating + movie effect + user effect

```
#Movie effect:
movie_effect <- movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(m_e = mean(rating-mean_rating),m_n=n())

#User effect:
user_effect <- train_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  group_by(userId) %>%
  summarize(u_e = mean(rating-mean_rating-m_e),u_n=n())

#Predict (mean+movie+user effect):
predicted_ratings <- test_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,m_e,u_e,na.rm = TRUE)) %>%
  pull(y)
m6_eval <- Evaluate_Model(predicted_ratings,"mean+movie+user effect")
m6_eval
```

```
##               model_name      rmse  accuracy
## 1 mean+movie+user effect 0.8646859 0.2490225
```

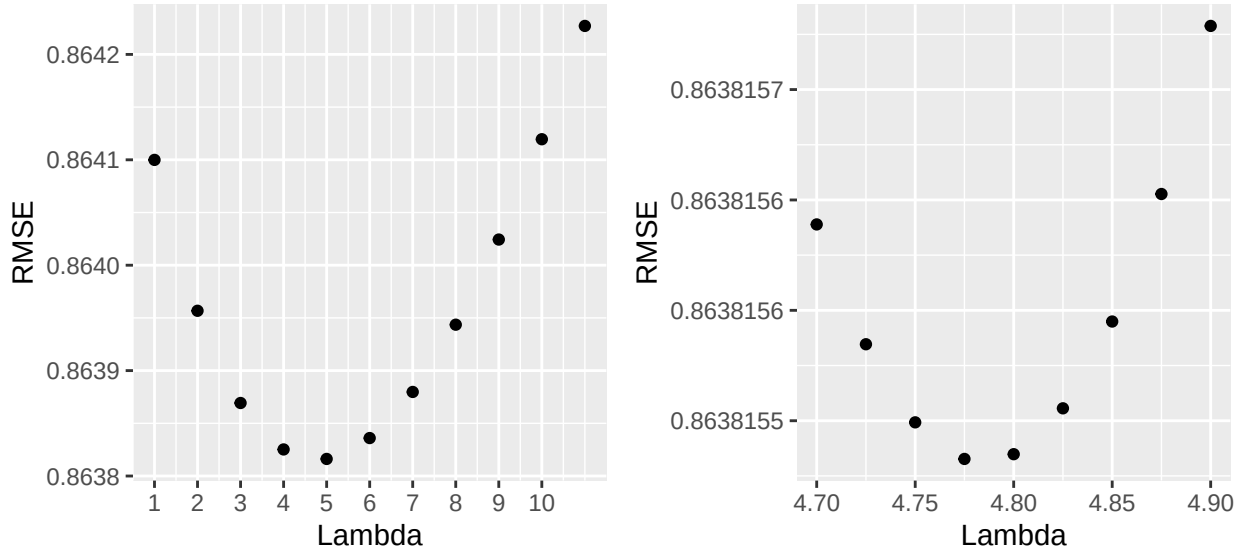
### Model 7 : Mean rating + movie effect + user effect + genres effect

```
#Movie effect and user effect is the same as before...

#Calculate genre effect:
genre_effect <- train_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  group_by(genres) %>%
  summarize(g_e = mean(rating-mean_rating-m_e-u_e),g_n=n())

#Predict (mean+movie+user+genres effect):
predicted_ratings <- test_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  left_join(genre_effect,by=c("genres")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,m_e,u_e,g_e,na.rm = TRUE)) %>%
  pull(y)
m7_eval <- Evaluate_Model(predicted_ratings,"mean+movie+user+genres effect")
m7_eval
```

```
##                       model_name      rmse  accuracy
## 1 mean+movie+user+genres effect 0.8643257 0.2492636
```

We have used the three selected variables (movieId,userId,genres) to optimize our model, now it's time to fine-tune our algorithm. As we saw earlier, there are movies/users/genres that have few ratings. Observations with a small number of ratings are not really reliable, so we introduce a lambda penalty value to reduce these effects. Please check Model 8 for the calculation details, where we simple replace the mean function to a manual calculation where the row number is increased by lambda.

Lets try lambdas from 0 to 10 and than refine it between 4.7 and 4.9 :




We can choose the optimal value based on the charts :

Lambda = 4.755

**Model 8 with Lambda:**

```
# Model with the selected lamba penality (4.75)
lambda <- 4.755
mean_rating = mean(train_set$rating)
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(m_e = sum(rating-mean_rating)/(n()+lambda))
user_effect <- train_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  group_by(userId) %>%
  summarize(u_e = sum(rating-mean_rating-m_e)/(n()+lambda))
genre_effect <- train_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  group_by(genres) %>%
  summarize(g_e = sum(rating-mean_rating-m_e-u_e)/(n()+lambda))
predicted_ratings <- test_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  left_join(genre_effect,by=c("genres")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,m_e,u_e,g_e,na.rm = TRUE)) %>%
  pull(y)
```

```
m8_eval <- Evaluate_Model(predicted_ratings,"mean+movie+user+genres+lambda")
m8_eval
```

```
##                       model_name      rmse  accuracy
## 1 mean+movie+user+genres+lambda 0.8638155 0.2488836
```

If we check the prediction range of our Model 8, we can see that the addition of different effects could produce too low and too high values as well :

```
##                     Description        min      max mean  median
## 1 Rating range after prediction -0.5149524 5.862484 3.51 3.56719
```

Based on the knowledge that the test set has only ratings between 0.5 and 5, we can simple cut the results in the final algorithm to fit this range.

## Results , the final model

### Model 9 : Final model with lambda and [0.5,5] prediction interval

```
# Final model
lambda <- 4.755
mean_rating = mean(train_set$rating)
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(m_e = sum(rating-mean_rating)/(n()+lambda))
user_effect <- train_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  group_by(userId) %>%
  summarize(u_e = sum(rating-mean_rating-m_e)/(n()+lambda))
genre_effect <- train_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  group_by(genres) %>%
  summarize(g_e = sum(rating-mean_rating-m_e-u_e)/(n()+lambda))
predicted_ratings <- test_set %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  left_join(genre_effect,by=c("genres")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,m_e,u_e,g_e,na.rm = TRUE)) %>%
  mutate(y=ifelse(y>5,5,y),y=ifelse(y<0.5,0.5,y)) %>%
  pull(y)
m9_eval <- Evaluate_Model(predicted_ratings,"mean+movie+user+genres+lambda+interval")
m9_eval
```

```
##                                 model_name      rmse  accuracy
## 1 mean+movie+user+genres+lambda+interval 0.8636914 0.2488947
```

**Final model performance on the validation set (final_holdout_test) :**

```
predicted_ratings <- final_holdout_test %>%
  left_join(movie_effect,by=c("movieId")) %>%
  left_join(user_effect,by=c("userId")) %>%
  left_join(genre_effect,by=c("genres")) %>%
  rowwise() %>%
  mutate(y=sum(mean_rating,m_e,u_e,g_e,na.rm = TRUE)) %>%
  mutate(y=ifelse(y>5,5,y),y=ifelse(y<0.5,0.5,y)) %>%
  pull(y)
final_eval <- Evaluate_Model(predicted_ratings,"Final holdout test",y = final_holdout_test$rating)
final_eval
```

```
##           model_name      rmse  accuracy
## 1 Final holdout test 0.8647429 0.2486862
```

## Conclusion

We have successfully developed a model with the following steps :

| model_name | rmse | accuracy |
|---|---|---|
| median alone | 1.1667627 | 0.2872967 |
| mean alone | 1.0600561 | 0.0882626 |
| mean+user effect | 0.9777083 | 0.2051206 |
| mean+movie effect | 0.9429666 | 0.2251638 |
| mean+genres effect | 1.0175053 | 0.1681687 |
| mean+movie+user effect | 0.8646859 | 0.2490225 |
| mean+movie+user+genres effect | 0.8643257 | 0.2492636 |
| mean+movie+user+genres+lambda | 0.8638155 | 0.2488836 |
| mean+movie+user+genres+lambda+interval | 0.8636914 | 0.2488947 |
| Final holdout test | 0.8647429 | 0.2486862 |

Some of our steps decreased RMSE significantly and others only slightly. The limitation of this work is that we have focused on creating a fast solution which needs only couple of minutes to run on an average computer.

I can imagine the following improvements :
- Split genres into individuals like (Action, Romance,etc. . . ) and calculate effects for each.
- As 'Ratings in general' histogram shows users are more likely to give integer values. We can use this information to refine the model.
- Clustering users and movies by similarities can also be an option because this way we minimize the missing values in the User/Movie rating matrix

Thanks you for taking the time to read my report.