MovieLens Project Submission for HarvardX PH125.9x

Jordan Georgiev

2024-12-08

Introduction

The Movielens project is part of the HarvardX PH125.9x Capstone course. The goal of this project is to create a recommendation system for movies based on the 10M version of the MovieLens dataset.

A **recommendation system** is a type of information filtering system that suggests relevant items to users based on their preferences, interests, and past behavior.[1] These systems are commonly used in various online platforms, such as e-commerce websites, streaming services, social media, and news sites, to enhance user experience and increase engagement. The system tries to predict a recommendation or rating a user would give to a product (articles, books, movies etc.) by using machine learning algorithms. The predicted rating can then be used to recommend additional products to the user.

Project goal and evaluation Our goal is to develop and train a machine learning model capable of predicting the ratings a user would provide for a movie in a designated validation set. This model will be trained on a dataset containing historical ratings from a set of users for various movies. In order to evaluate the performance of our model we will use the loss function RMSE (root mean squared error) while aiming at a value of RMSE < 0.86490.

Approach The following steps will be followed and documented during the Movielens project to ensure our model works.

1. **Data collection and preparation**: load and prepare the dataset provided, clean and preprocess the data to handle missing values. Split data into training, validation, and test sets.
2. **Exploratory data analysis**: Analyse and visualize the data to understand the features and predictors.
3. **Model creation and evaluation**: create, fine-tune, test and validate the model.
4. **Reporting**: document the analzsis and the model and create the final report

Data collection and preparation

The code for downloading and preparing the data was provided in the course documentation and is included in the movielens.r script we are using to create our model. The code split the data into two datasets - **edx** and **final_holdout_test**. As per assignment requirement, the later was NOT used for training, developing, or selecting our algorithm and it was ONLY used for evaluating the RMSE of our final algorithm. The tidyverse and caret libraries were loaded using the provided code.

In the next step we downloaded additional packages and loaded the necessary libraries we would use. - knitr - kableExtra - ggplot2 - ggthemes - gridExtra - scales

Since we were not allowed to use the **final_holdout_test** for testing our algorithm, we partitioned the **edx** data into **train_set** (90% of the observations) and a **test_set** (10% of the observations) using the following code.

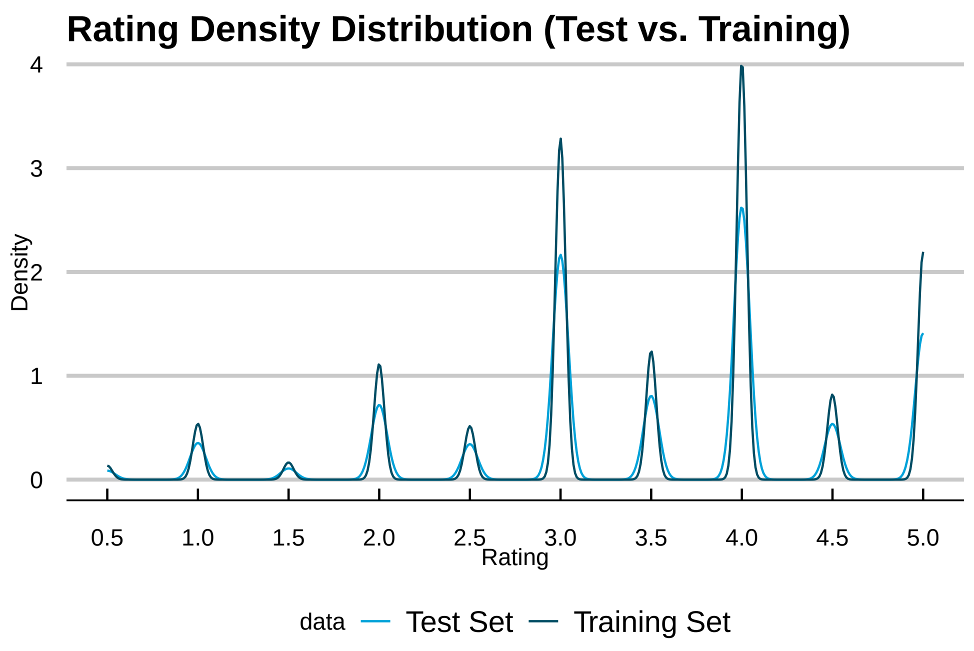
[1] https://www.nvidia.com/en-us/glossary/recommendation-system/

```r
# Split edx into train and test sets
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Here an overview of the number of rows and columns of the datasets to be used:

Dataset	Rows	Columns
edx	9000055	6
final_holdout_test	999999	6
train_set	8100065	6
test_set	899990	6

After preparing our datasets, we examined the rating density distributions across the rating categories in both the **train_set** and the **test_set**. They seem to match.



The datasets were further examined for missing values in order to do data cleaning. No missing values were found, the datasets seem tidy.

Dataset	Missing
edx	FALSE
final_holdout_test	FALSE

We further examined the data structure of the **edx** dataset. The release year of each movie is stored in the column title, so if we would need this information for our model we should split the title into two columns - title and year. The rating timestamp is stored as UNIX time (integer), which measures time by the number of non-leap seconds that have elapsed since 00:00:00 UTC on 1 January 1970. If we decide to use the rating timestamp, we will need to convert it and store it separately.

Here is a summary of the **edx** data:

Table 3: edx Summary

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000055	Length:9000055
1st Qu.:18124	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
Median :35738	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35870	Mean : 4122	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

The first couple of rows look like this:

Table 4: edx Head

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy:Romance
2	1	185	5	838983525	Net, The (1995)	Action:Crime:Thriller
4	1	292	5	838983421	Outbreak (1995)	Action:Drama:Sci-Fi:Thriller
5	1	316	5	838983392	Stargate (1994)	Action:Adventure:Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action:Adventure:Drama:Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children:Comedy:Fantasy

The **edx** dataset contains data about movies, their genres, users and their ratings - the following table shows us a summary of the data included:

Table 5: edx Summary

Users	Movies	Genres
69878	10677	797

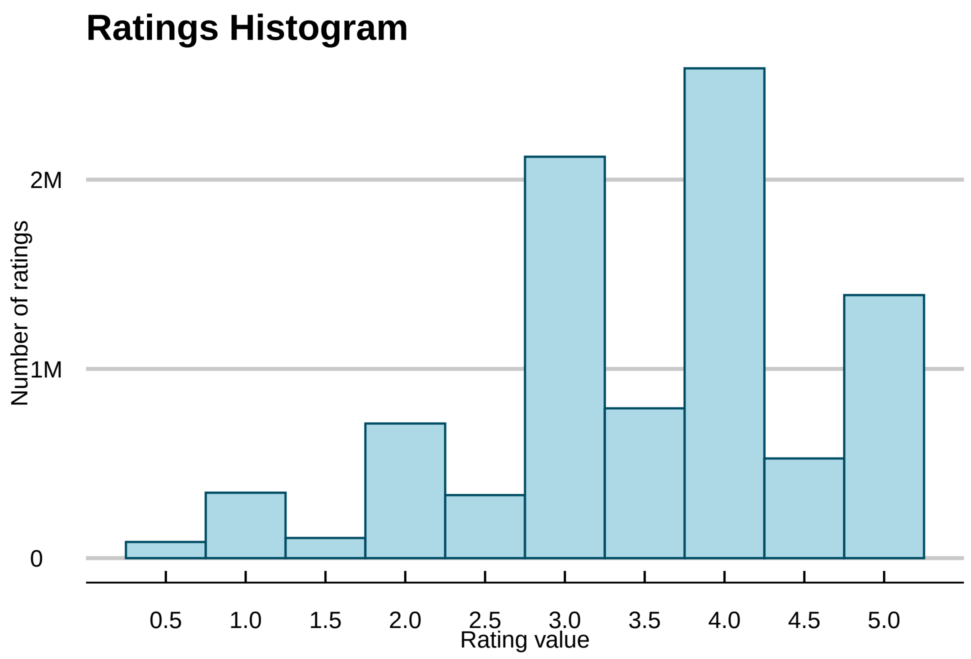
Exploratory Data Analysis

In our model we want to predict the rating a user could give to a movie in the test set. So let us have a look at the rating column. The ratings are from 0 (low) to high (5) with a 0.5 increment - so 10 rating variables in total. The average rating over all records is slightly above 3.5.

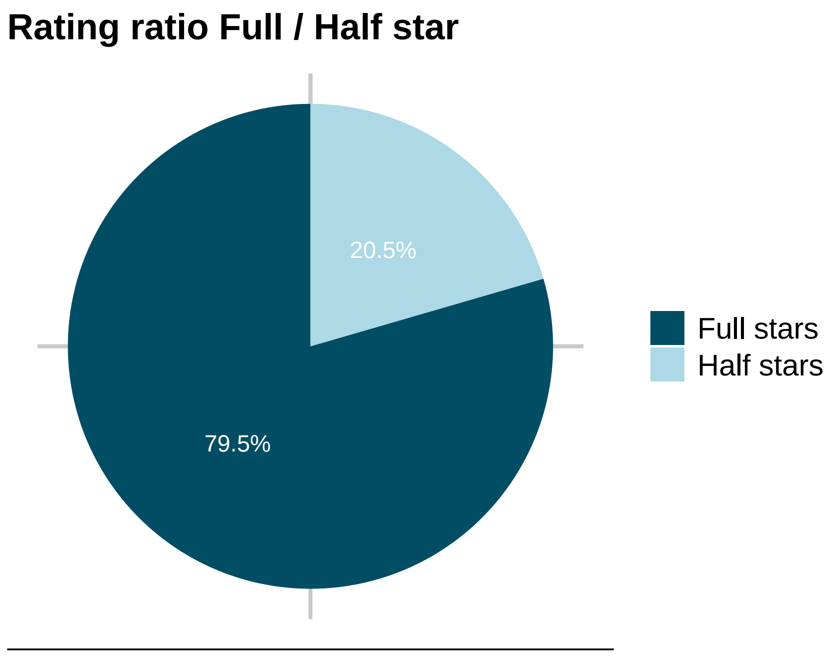
```r
mean(edx$rating)
```

```
## [1] 3.512465
```

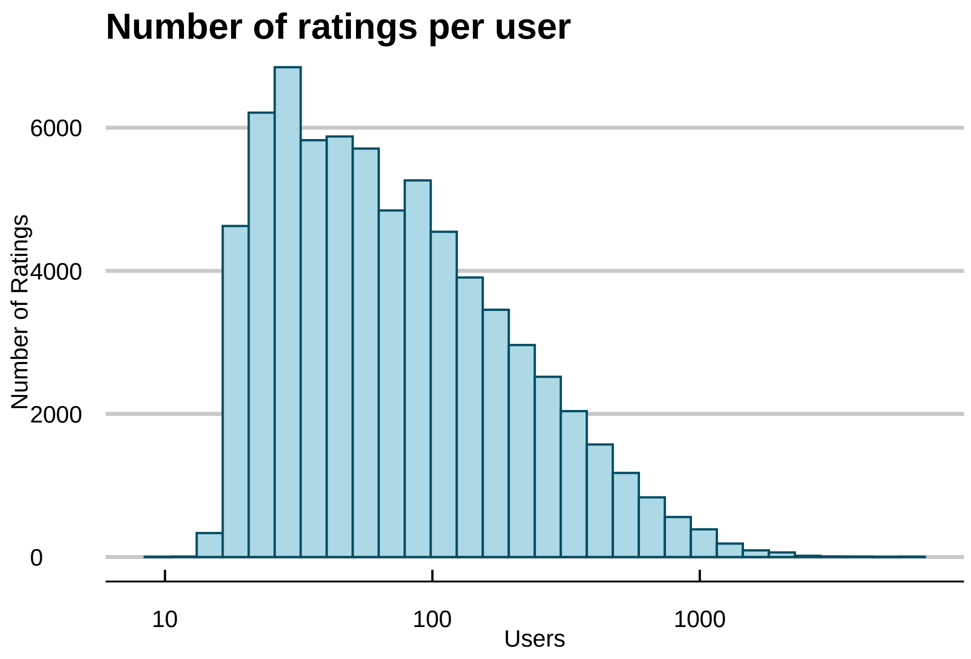
The ratings histogram shows, that the most used rating is 4, followed by 3 and 5.



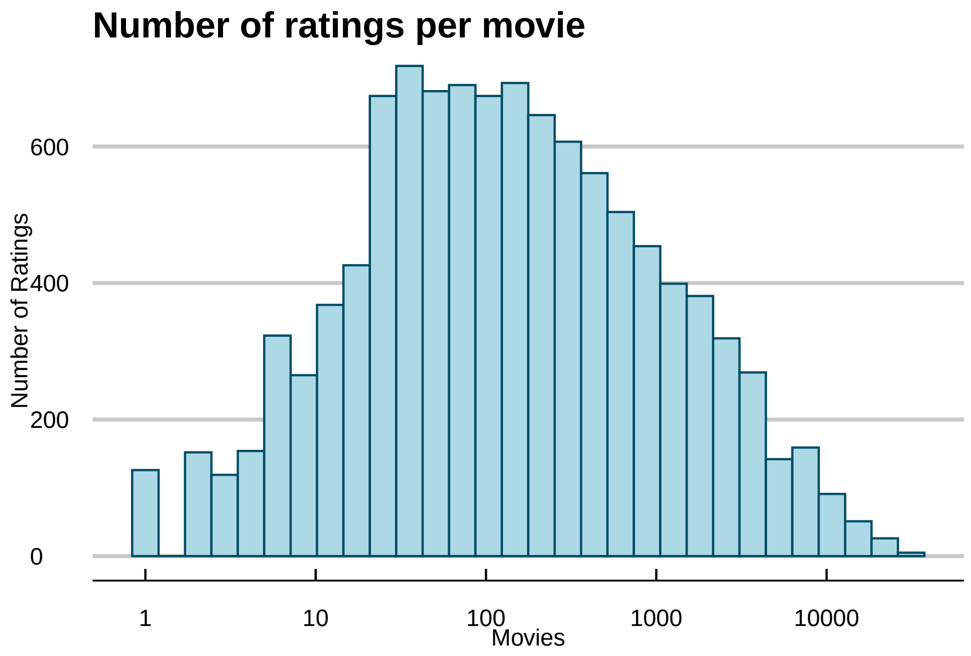
It is obvious, that the users strongly prefer so called full star ratings (1-5), their share is almost 80%. Of all ratings only 20% are in half step (0.5, 1.5, 2.5 etc.). We will see, if this preference can help us create a better estimation.



If we look at the distribution of average ratings per user, we can recognize that some users are more active in rating than others. Most users review very few movies.



We can also confirm, that some popular movies, such as blockbusters attract more viewers and ratings respectively.



Modell creation and evaluation

For evaluating the accuracy of our model we will use a loss-function, which will calculate the root mean squared error (RMSE), which is the square root of the mean of the square of all of the errors. Since RMSE squares the error terms, larger errors have a disproportionately large effect, making RMSE sensitive to outliers. RMSE can be used for comparison of different models and a lower value of RMSE indicates better model performance.

We define a RMSE function, which will then be used to calculate the accuracy of our predictions:

```r
# Define Root Mean Squared Error (RMSE) formula
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

As we are building a recommendation system, we will use a regression model and an approach similar to the one outlined in the course material for PH128.x and in Irizzary, 2019.[2] The mean rating μ is modified by one or more "biases" b.

$$Y_{u,i} = \mu + b_i + b_u + b_g + \varepsilon_{i,u,g}$$

We start with a simple, naive model by just assuming the same rating for all movies and users with all the differences explained by random variation. This will be our baseline, to which we will compare all successive additions of biases.

$$Y_{i,j} = \mu + \varepsilon_{i,j}$$

```r
# Baseline Prediction based on Mean Rating
mu <- mean(train_set$rating)
rmse_baseline <- RMSE(test_set$rating, mu)
```

method	RMSE
Naive Mean	1.060054

Our result is much higher than the one we are aiming to beat (0.86490), so we will add a movie bias to our model. We know, that some movies are generally rated higher than others, so we will try to accommodate this in our model bi adding a movie bias _i.

$$Y_{u,i} = \mu + b_i + \varepsilon_{i,u}$$

```r
# Add Movie effect
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Crete prediction
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by="movieId") %>%
  pull(b_i)

# Calculate RMSE mu + b_i
rmse_movie_effect <- RMSE(predicted_ratings, test_set$rating)
```

method	RMSE
Naive Mean	1.0600537

[2]https://rafalab.dfci.harvard.edu/dsbook-part-2/highdim/regularization.html

Movie Effect	0.9429615

We see how our RMSE is decreasing. Now we are adding the user bias, since some users are more picky and others just love most movies.

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{i,u}$$

```
# Add User effect
user_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

# Crete prediction
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculate RMSE mu + b_i + b_u
rmse_user_effect <- RMSE(test_set$rating, predicted_ratings)
```

method	RMSE
Naive Mean	1.0600537
Movie Effect	0.9429615
Movie + User Effects	0.8646843

We further decreased our RMSE, but not enough. We will also add the effect of the genre of the movie, as we assume, that some genres are watched more and get more ratings than others.

```
# Genres effect
genre_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - mu - b_i - b_u))

# Create prediction
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Calculate RMSE mu + b_i + b_u + b_g
rmse_genre_effect <- RMSE(test_set$rating, predicted_ratings)
```

```
rmse_results <- bind_rows(rmse_results, data.frame(method="Movie + User + Genre Effects", RMSE = rmse_g
kable(rmse_results, booktabs = TRUE, escape = FALSE) %>%
  kable_styling(position = "center", latex_options = c("striped"))
```

method	RMSE
Naive Mean	1.0600537
Movie Effect	0.9429615
Movie + User Effects	0.8646843
Movie + User + Genre Effects	0.8643241

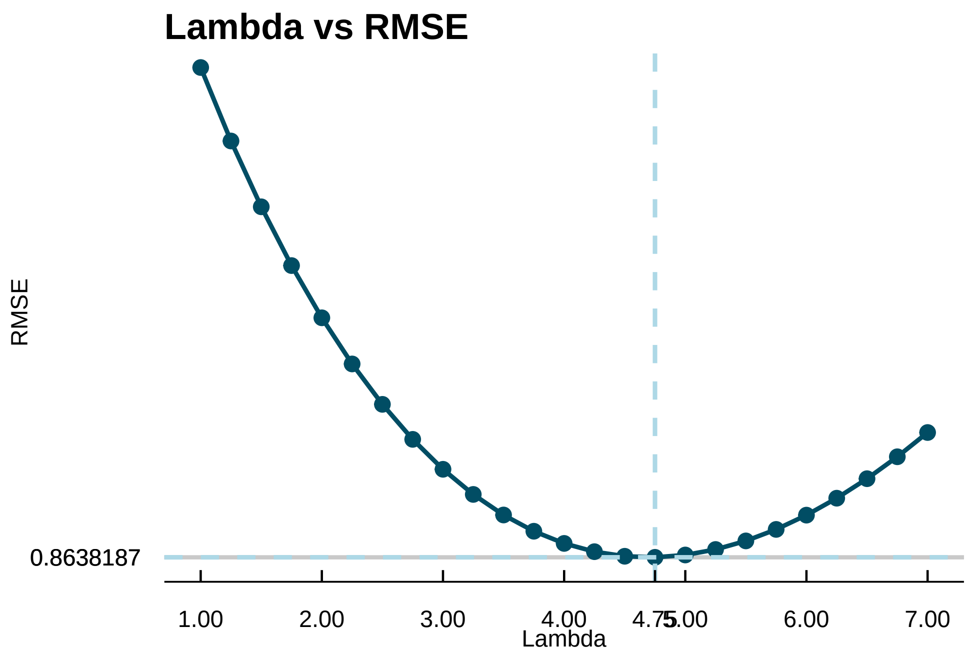
Now we will use regularization (penalized least squares), which permits us to penalize large estimates that are formed using small sample sizes, e.g. estimating a high rating for a movie based on only few (high) ratings. We will use this formula:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-b_g)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2+\sum_g b_g^2)$$

The first term is the mean squared error and the second is a penalty term that gets larger when many variables are large. We will minimize the equation to find the λ (tuning parameter) for which we will get the minimum for RMSE. We can use cross-validation to choose it.

```r
# Regularization - find lambda for min RMSE
lambdas <- seq(1, 7, 0.25)
rmses <- sapply(lambdas, function(lambda) {
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n() + lambda))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu - b_i)/(n() + lambda))
  b_g <- train_set %>%
    left_join(movie_avgs, by="movieId") %>%
    left_join(user_avgs, by="userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - mu - b_i - b_u)/(n() + lambda))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  return(RMSE(test_set$rating, predicted_ratings))
})

# Find optimal lambda and RMSE
opt_lambda <- lambdas[which.min(rmses)]
min_rmse <- min(rmses)
```

We can now plot the λ against the RMSEs and we can see that we achieve the minimum RMSE with $\lambda = 4.75$.



We can now calculate the RMSE after regularization and we observe that it was reduced further.

method	RMSE
Naive Mean	1.0600537
Movie Effect	0.9429615
Movie + User Effects	0.8646843
Movie + User + Genre Effects	0.8643241
Regularization	0.8638187

We achieved RMSE of 0.8638187 after regularization. Now we are ready to fit our model by using $\lambda = 4.75$ and calculate the final RMSE using the **final__holdout__test**.

```r
# Calculate with lambda which minimizes RMSE
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lambda))

b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n() + lambda))

b_g <- train_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n() + lambda))

# Final prediction with final_holdout_test
predicted_ratings_final <- final_holdout_test %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Final RMSE for final_holdout_test
RMSE_final <- RMSE(final_holdout_test$rating, predicted_ratings_final)
```

method	RMSE
Naive Mean	1.0600537
Movie Effect	0.9429615
Movie + User Effects	0.8646843
Movie + User + Genre Effects	0.8643241
Regularization	0.8638187
Final Holdout Test	0.8648545

In our algorithm we used the overall rating average and added the average ratings for movies, users, genres and achieved much better RMSE results, as the naive (baseline) prediction. With regularization, the RMSE further improved further. After applying the algorithm to the ***final_holdout_test** dataset, we achieved RMSE of 0.8648545, which is slightly better than the RMSE of 0.86490 we were aiming for.

Results and conclusion

An algorithm to achieve RMSE < 0.86490 while predicting movie ratings was successfully developed. The usage of linear regression model with regularized effects for movies users and genres is adequate to predict movie ratings in the Movielens 10M dataset.

The approach described above was chosen after several unsuccessful tests with logistic regression and random forest, which failed due to hardware limitations and the large dataset size. Expanding the current model with adding an additional factor to accommodate for the fact that users prefer full-star to half-star ratings did not improve the RMSE.

Future work to improve the model can include fine-tuning and including a “time bias”. Investigation of the relation between movie rating and the release date of a movie could yield further reduction of the RMSE.