Movie Recommendation System using the 10M version of the MovieLens dataset

HarvardX PH125.9x

Data Science: Capstone

The MovieLens project R code, R Markdown code and the PDF version are available on GitHub.

Introduction

Recommendation systems changed the way inanimate websites communicate with their users. Rather than providing a static experience in which users search for and potentially buy products, recommendation systems increase interaction to provide a richer experience. The movie recommendation systems help in predicting the choice of movie for the users based on the interests and the historical data and it is one of the most popular application of big data processing.

Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. The Netflix data is not publicly available, but the GroupLens research lab113 generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users.

Project Overview

For this project, we make a small subset of this data available via the dslabs package. We will be creating a movie recommendation system using this MovieLens dataset. We will use the 10M version of the MovieLens dataset. We split the dataset into a training set (edx) and a final hold-out test set (validation) using the code provided. The objective for the final algorithm is to predict ratings with a residual mean square error (RMSE) less than 0.86490, the RMSE returned by testing your algorithm on the validation set.

Process:

1. Data Preparation
2. Data Exploration
3. Data Cleaning
4. Data Modeling
5. Final Validation

1. Data Preparation

We will use the following code to generate our datasets. Develop our algorithm using the edx set. For a final test of our final algorithm, we predict movie ratings in the validation set (the final hold-out test set) as if they were unknown. RMSE will be used to evaluate how close our predictions are to the true values in the validation set (the final hold-out test set).

```
# Create edx set, validation set (final hold-out test set)

library(tidyverse)
library(caret)
library(data.table)
```

```
library(ggplot2)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(movieId),
         title = as.character(title),
         genres = as.character(genres))


movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

2. Data Exploration

General overview of the dataset.

```
head(edx)
```

```
##    userId movieId rating timestamp                    title
## 1:      1     122      5 838985046         Boomerang (1992)
## 2:      1     185      5 838983525          Net, The (1995)
## 3:      1     292      5 838983421          Outbreak (1995)
## 4:      1     316      5 838983392          Stargate (1994)
```

```
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The structure of the data.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

Number of different movies are in the edx dataset.

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

Number of different users are in the edx dataset.

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

Number of movie ratings in each of the following genres in the edx dataset.

```
genres <- c("Drama", "Comedy", "Thriller", "Romance")
sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
})
```

```
##    Drama   Comedy Thriller  Romance
##  3910127  3540930  2325899  1712100
```

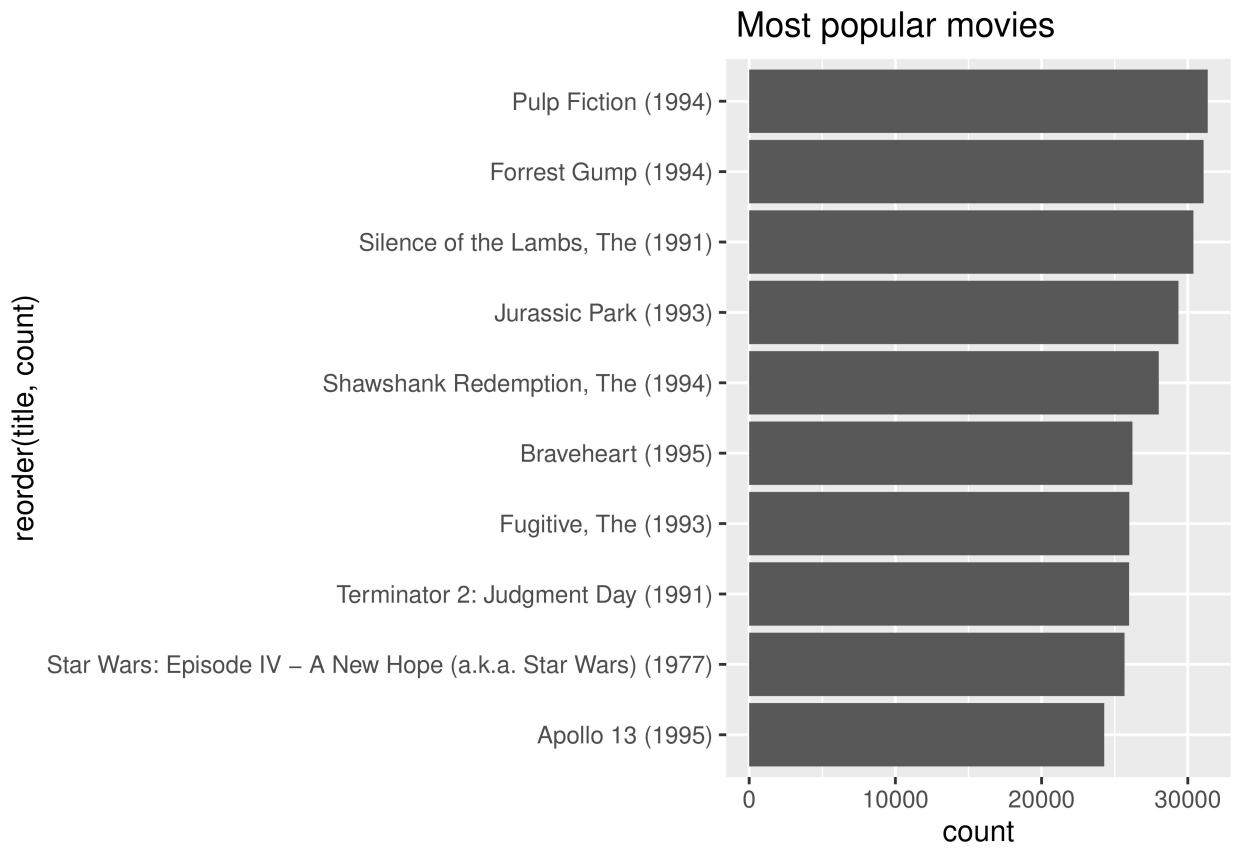
Movies with most ratings.

```
edx %>%
  group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                     count
##      <dbl> <chr>                                                     <int>
##  1     296 Pulp Fiction (1994)                                       31362
##  2     356 Forrest Gump (1994)                                       31079
##  3     593 Silence of the Lambs, The (1991)                          30382
##  4     480 Jurassic Park (1993)                                      29360
##  5     318 Shawshank Redemption, The (1994)                          28015
##  6     110 Braveheart (1995)                                         26212
##  7     457 Fugitive, The (1993)                                      25998
##  8     589 Terminator 2: Judgment Day (1991)                         25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                          24284
## # ... with 10,667 more rows
```

```
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(10, count) %>%
  ggplot(aes(count, reorder(title, count))) +
  geom_bar(stat = "identity") +
  ggtitle(" Most popular movies")
```

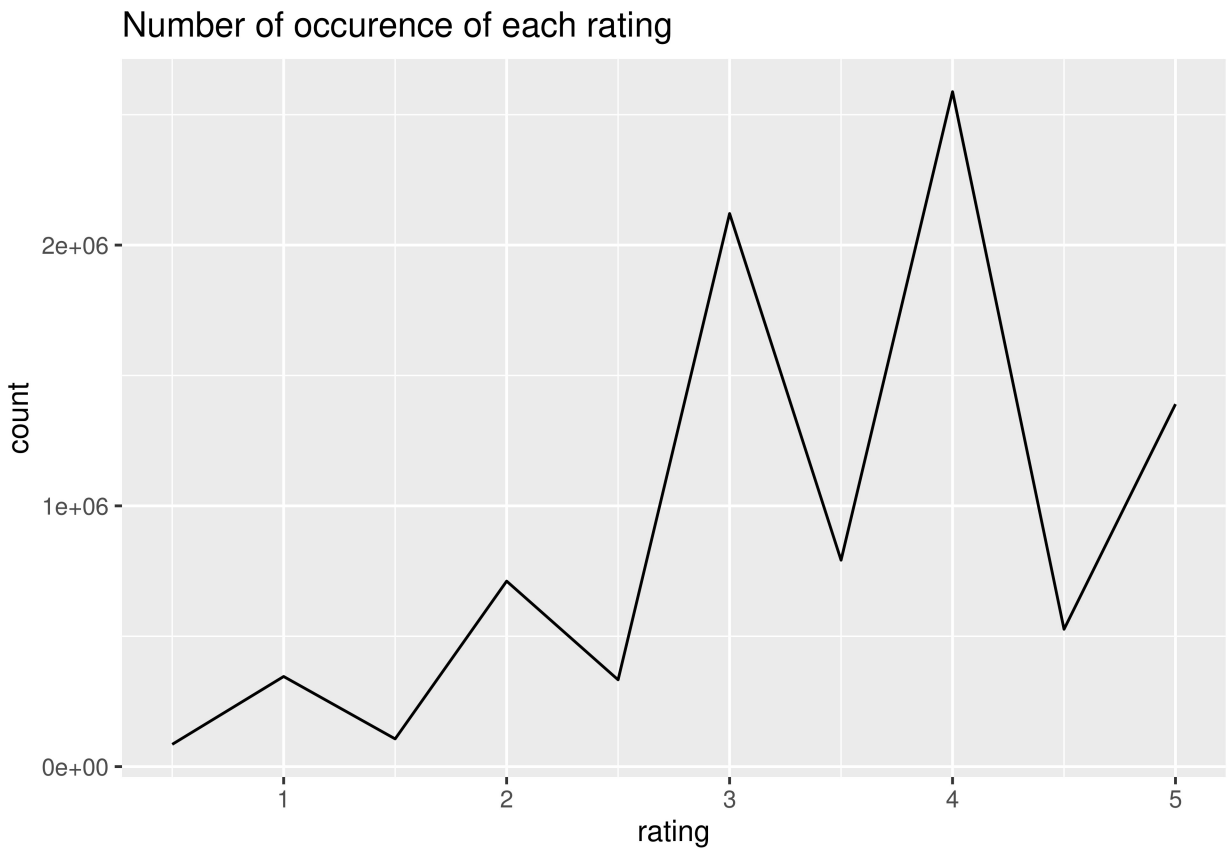


Most common ratings.

```
edx %>%
  group_by(rating) %>%
  summarize(count=n()) %>%
  top_n(5, count) %>%
  arrange(desc(count))
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

```
edx %>% group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line(color="black") +
  ggtitle("Number of occurence of each rating")
```



The graph clearly shows that the common rating is 4 and most of the users are giving a positive rating rather than a negative rating

Loss Function - RMSE

The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set. We define $y_{u,i}$ as the rating for movie i by user u. The RMSE is then defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

Partitioning edx into test set and train set

We split the edx set in 2 parts: the training set and the test set. The model building is done in the training set, and the test set is used to test the model. When the model is complete, we use the validation set to calculate the final RMSE. We use the same procedure used to create edx and validation sets.

```
# Preparing data
# Validation set will be 10% of edx data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

3. Data Cleaning

Data cleansing is a process in which you go through all of the data within a database and either remove or update information that is incomplete, incorrect, improperly formatted, duplicated, or irrelevant. Data cleansing does and can involve deleting information. Here we will be using only userID, movieID, rating and title for our model, so we are removing timestamp and genres.

```
train_set <- train_set %>%
  select(userId, movieId, rating, title)
test_set  <- test_set  %>%
  select(userId, movieId, rating, title)
```

4. Data Modeling

Methods used to train and test the algorithm

Model 1 : Using Mean Model 2 : Using Mean + Movie Effects Model 3 : Mean + Movie Effects + User Effects Model 4 : Regularization: Movie + User Effects

Model 1 : Using Mean

We start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

We know that the estimate that minimizes the RMSE is the least squares estimate, in this case, is the average of all ratings:

```r
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512456
```

```r
rmse_results <- tibble(Method = "Mean",
                       RMSE = RMSE(test_set$rating, mu_hat))
rmse_results
```

```
## # A tibble: 1 x 2
##   Method  RMSE
##   <chr>  <dbl>
## 1 Mean    1.06
```

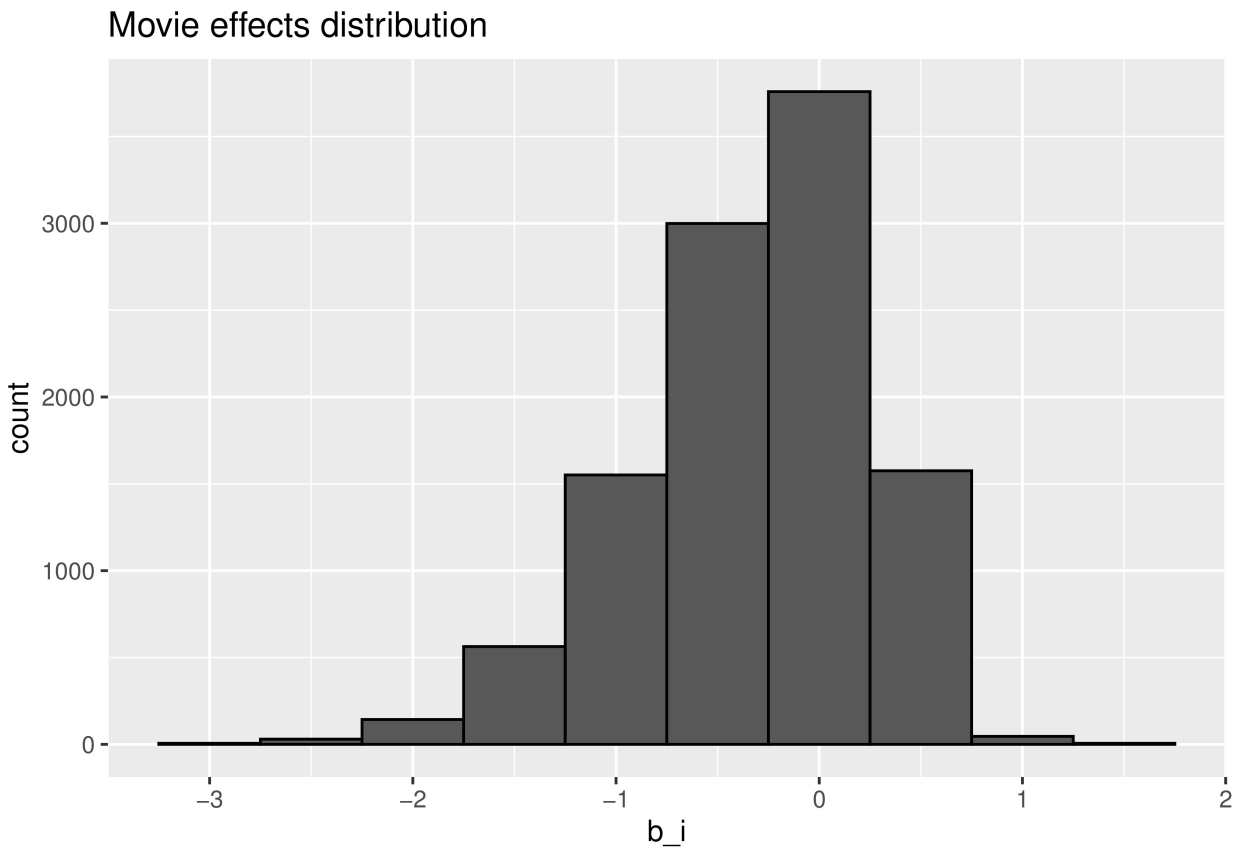
Model 2 : Using Movie Effects

We can improve model 1 by adding movie effects, b_i. Considering the fact that some movies are just generally rated higher than others. We can augment our previous model by taking movie bias, by adding the term b_i to represent average ranking for movie i:

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

```r
# Estimating movie effects
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

```r
# Movie effects distribution:
bi %>%
  ggplot(aes(b_i)) +
  geom_histogram(bins=10, color = "black") +
  ggtitle("Movie effects distribution")
```



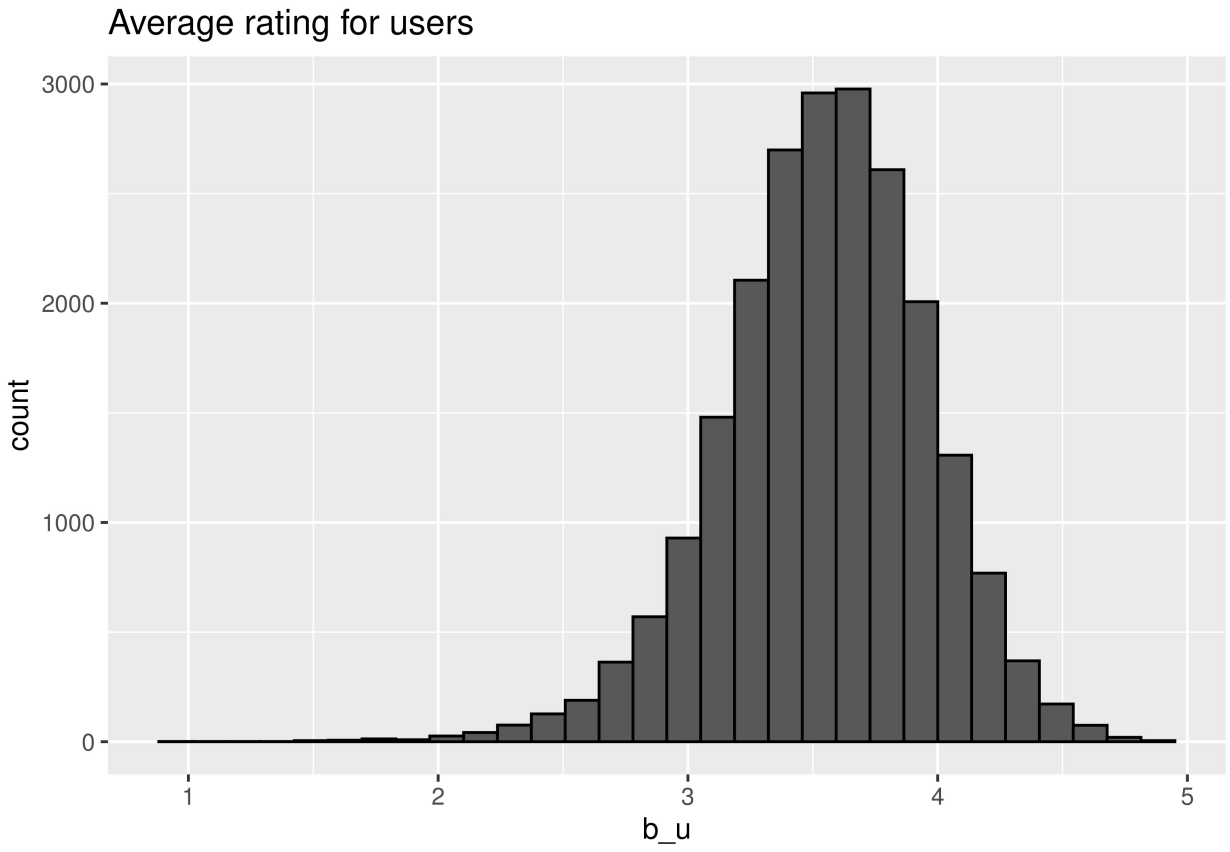

```
# Predicted ratings for Movie effects:
predict_bi <- mu_hat +
  test_set %>%
  left_join(bi, by = "movieId") %>%
  pull(b_i)

# RMSE for Mean + Movie effects
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Mean + Movie effects",
                                 RMSE = RMSE(test_set$rating, predict_bi)))
rmse_results
```

```
## # A tibble: 2 x 2
##   Method                RMSE
##   <chr>                <dbl>
## 1 Mean                 1.06
## 2 Mean + Movie effects 0.943
```

Model 3 : Using User Effects

We can also add the user effects to our model, b_u, because we can find variability across user as well. Visualizing the average rating for user u for those that have rated 100 or more movies.

```
train_set %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")+
  ggtitle("Average rating for users")
```



A further improvement to our model may be:

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

We can estimate $\hat{b}_u$ as below:

$$\hat{b}_u = mean\left(\hat{y}_{u,i} - \hat{\mu} - \hat{b}_i\right)$$

```
# Estimate User Effects:
bu <- train_set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

# Predicted ratings for movie + user effects
predict_bu <- test_set %>%
```

```
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)

# RMSE for Mean + Movie effects + User effects:
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Mean + Movie effects + User effects",
                                 RMSE = RMSE(test_set$rating, predict_bu)))
rmse_results
```

```
## # A tibble: 3 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Mean                                 1.06
## 2 Mean + Movie effects                 0.943
## 3 Mean + Movie effects + User effects 0.865
```

Model 4: Regularization

Regularization is one of the basic and most important concept in the world of Machine Learning. The general idea behind regularization is to constrain the total variability of the effect sizes. In this model we build a function by adding lambda to minimize the RMSE. Note that lambda is a tuning parameter, we can use cross-validation to choose it.

```
# Generate a sequence of values for lambda ranging from 3 to 6 with 0.25 inc.

lambdas <- seq(3, 6, .25)
# Regularized model, predict ratings and calculate RMSE for each value of lambda
rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(test_set$rating, predicted_ratings))
})
```

```r
# Assign optimal tuning parameter (lambda)
lambda <- lambdas[which.min(rmses)]

# Minimum RMSE achieved
regularised_rmse <- min(rmses)

# Results table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Regularised Movie + User effects",
                                 RMSE = regularised_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Mean                                 1.06
## 2 Mean + Movie effects                 0.943
## 3 Mean + Movie effects + User effects 0.865
## 4 Regularised Movie + User effects    0.864
```

Final validation: Linear Model with Regularization.

Now for the final validation we will train the complete edx set and calculate the RMSE in the validation set. The project goal is achieved if the RMSE stays below the target.

```r
mu <- mean(edx$rating)

# Movie effect (bi)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

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

# Prediction
predicted_ratings_final <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Results table
rmse_results <- bind_rows(rmse_results,
                    tibble(Method = "Final Regularization (edx vs validation)",
                           RMSE = RMSE(validation$rating, predicted_ratings_final)))

# The RMSE returned by testing the algorithm on the validation set
final_results <- bind_rows(tibble(Method = "Project objective",
```

```
                                    RMSE = 0.86490), rmse_results)

final_results %>% knitr::kable()
```

Method	RMSE
Project objective	0.8649000
Mean	1.0600537
Mean + Movie effects	0.9429615
Mean + Movie effects + User effects	0.8646843
Regularised Movie + User effects	0.8641362
Final Regularization (edx vs validation)	0.8648177

Conclution

We started by building the simplest possible recommendation system by estimating the Average of all the ratings. Then we added the Movie Effects and then added User Effects for the next. We constructed the predictors and saw how much the RMSE improved. For our final model we used User, Movie and Movie Age Effects with Regularization, we added a penalty value for the movies and users with few number of ratings. The linear model achieved the RMSE of 0.8648177, successfully achieving the project objective of 0.8649.