MovieLens Project Report

Nelscy Acevedo

Contents

Chapter 1

Introduction

Recommendation systems are designed to predict user preferences and recommend potential customers products that they may like to buy or use. These systems are built using machine learning techniques and their usage between companies has increased in recent years.

This document presents a recommendation system elaborated for movies based on users' ratings. To evaluate the system performance, we will use the Root Mean Squared Error (RMSE) as the loss function. RMSE calculates the typical error we make when predicting a movie rating. The lower the RMSE, the better the system performance. For this project, the goal is to create a recommendation system with RMSE less than 0.86490.

To create the recommendation system, we will use the MovieLens dataset. This dataset was generated by the GroupLens research lab at the University of Minnesota. It contains 27,753,444 ratings for 58,098 movies by 283,228 users made between January 09, 1995 and September 26, 2018. In this project, we will use a subset of this database, consisting in 10,000,054 ratings. This subset was split into two sets: edx and validation. Edx set contains 9,000,055 ratings on 10,677 movies by 69,878 users. Validation set contains 999,999 ratings on 9,809 movies by 68,534 users.

Edx set will be used to train different models. For this reason, we will split again edx set into two sets: train and test. Then, we will select the model that performs better in terms of RMSE to elaborate the final validation.

The rest of the document is structured as follows. Section 2 presents the preliminaries about process and techniques used, data cleaning, data exploration and visualization, insights gained, and modeling approach. Section 3 details the modeling results, discusses the model performance and evaluate the final model. Finally, section 4 exposes the main conclusions, limitations and future work.

Chapter 2

Methods and Analysis

2.1 Preliminaries

After installing the required packages, the first step to perform is to download the MovieLens database and select a subset consisting in 10,000,054 movie ratings. After that, we split this subset into two sets: edx and validation, with 90% and 10% of the observations, respectively. Edx set contains 9,000,055 ratings on 10,677 movies by 69,878 users. Validation set contains 999,999 ratings on 9,809 movies by 68,534 users.

Edx set will be used to train different models, while validation set will be used only for evaluating the RMSE of the final model. Therefore, we split edx set again into two sets: train and test. Similarly to how MovieLens database was split, we will consider 90% of the observations in edx set for the train set and 10% for the test set. Train set will be used to build different models and test set will be used to evaluate them.

Next we can see the code used to carry out these first steps.

```
##################################
# Preliminaries                  #
# Create edx and validation sets #
##################################

# Note: this process could take a couple of minutes

if(!require(tidyverse))
```

```r
  install.packages("tidyverse",
                   repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret",
                   repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table",
                   repos = "http://cran.us.r-project.org")

# 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")
movies <- as.data.frame(movies, #For R 4.0, set stringsAsFactors as TRUE
                        stringsAsFactors = TRUE) %>%
  mutate(movieId = as.numeric(levels(movieId))[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")
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)

###########################################
# Create train and test sets from edx set #
###########################################
# Test set will be 10% of edx data
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)
```

After creating train and test sets, we can do some data exploration that will help us to understand better their structure.

As training will be done with the edx set, we can explore it. This set has 9,000,055 observations (rows) and 6 variables (columns). Variables included are userId, movieId, rating, timestamp, title and genres.

```
# Data exploration in edx set
dim(edx)
```

```
## [1] 9000055       6
```

```
head(edx) %>% knitr::kable()
```

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

Variable userId refers to user information, users that rate movies. As the same user can rate several movies, we can calculate how many users are in the edx set as follows. There are 69,878 different users.

```
# Number of users in edx set
n_distinct(edx$userId)
```

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

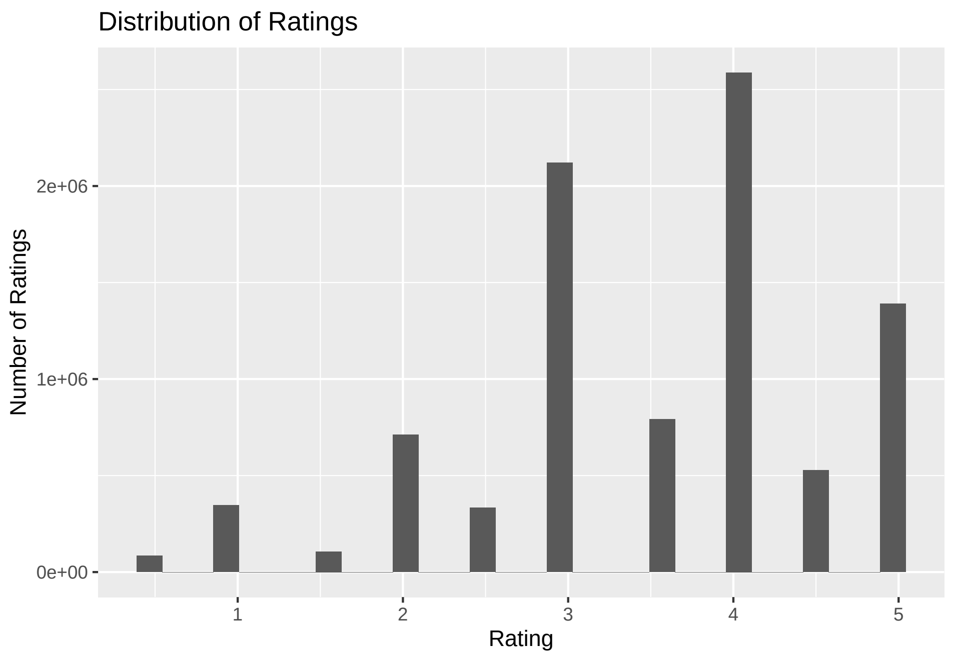
Each movie has its own identification number. This information is captured by variable movieId. As the same movie can be rated by several users, we can calculate how many movies are in the edx set as follows. We observe 10,677 different movies.

```
# Number of movies in edx set
n_distinct(edx$movieId)
```

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

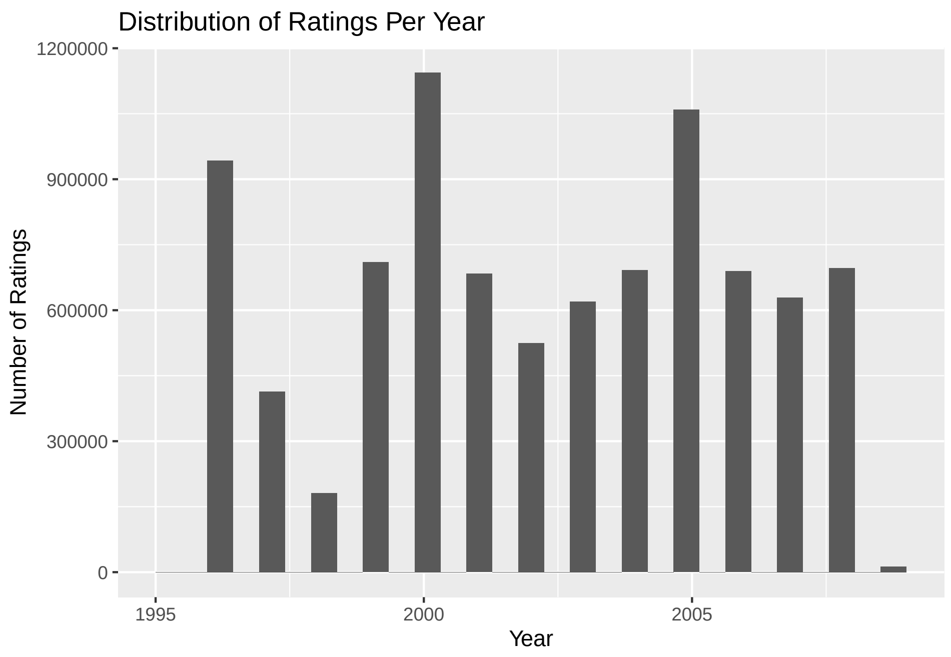
Movie ratings (variable rating) range from 0 to 5, with increases of 0.5 to 0.5. We can observe the distribution of ratings in the following graph. We see that most of ratings are integers (1, 2, 3, 4 or 5) and that 3 and 4 are the most common ratings chosen by users.

```
# Rating distribution in edx set
edx %>% group_by(rating) %>%
  ggplot(aes(x = rating)) +
  geom_histogram() +
  ggtitle("Distribution of Ratings") +
  xlab("Rating") +
  ylab("Number of Ratings")
```



Variable timestamp represents the time and data in which the rating was provided. This information is presented in seconds since January 1, 1970. In the following graph we can see the distribution of ratings per year. Ratings were done between 1995 and 2009, with 2000 and 2009 being the years with most ratings.

```r
# Rating distribution by year in edx set
if(!require(lubridate))
  install.packages("lubridate",
                   repos = "http://cran.us.r-project.org")
edx %>% mutate(year = year(as_datetime(timestamp,
                                       origin = "1970-01-01"))) %>%
  ggplot(aes(x = year)) +
  geom_histogram() +
  ggtitle("Distribution of Ratings Per Year") +
  xlab("Year") +
  ylab("Number of Ratings")
```



Variable genre specifies the genre or genres in which each movie belongs. We can calculate the number of ratings of some of the main genres as follows. We see that a large number of movies are categorized into comedy, action and thriller genres.

```r
# Main genres in edx set
genre = c("Action", "Adventure", "Animation", "Children",
          "Comedy", "Crime", "Drama", "Fantasy", "Film-Noir",
          "Horror", "IMAX", "Musical", "Mystery", "Romance",
          "Sci-Fi", "Thriller", "War", "Western")
sapply(genre, function(g) {
    sum(str_detect(edx$genres, g))
}) %>% knitr::kable()
```

	x
Action	2560545
Adventure	1908892
Animation	467168
Children	737994
Comedy	3540930
Crime	1327715
Drama	3910127
Fantasy	925637
Film-Noir	118541
Horror	691485
IMAX	8181
Musical	433080
Mystery	568332
Romance	1712100
Sci-Fi	1341183
Thriller	2325899
War	511147
Western	189394

Previously, we have seen that edx set contains 10,677 different movies. To know which are those movies, we can use the variable title that presents movie titles. In the following table we have the 10 movies that were rated most times by users.

```
# Movie that were rated most times by users in edx set
edx %>% group_by(title) %>% summarize(count = n()) %>%
  arrange(desc(count)) %>% head(10) %>% knitr::kable()
```

title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284

2.2 Modeling

In this part, we will detail the models that were built for training.

The first model predicts that all users will give the same rating to all movies. Therefore, any difference between movies will be due to random variation. This model can be specified like this:

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

Where:

$Y_{u,i}$: predicted rating.

μ: the "true" rating for all movies. Its least squares estimate is the average rating $\hat{\mu}$.

$\epsilon_{u,i}$: error term.

The second model considers movie effects. This effect takes into account the fact that movies are rated differently by users, with movies rated higher than others. We can include movie effects like this:

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

Where:

b_i: movie effects, average rating for movie i. Its least squares estimate $\hat{b}_i$ is the average of the difference between $Y_{u,i}$ and $\hat{\mu}$ for each movie i.

The third model includes another effect, user effects. The idea behind this effect is the fact that users rates differently. Some of them gives high ratings to every movie, others low ratings. This effect can be included like this:

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

Where:

b_u: movie effects, average rating for user u. Its least squares estimate $\hat{b}_u$ is the average of the difference between $Y_{u,i}$, $\hat{\mu}$ and $\hat{b}_i$.

The fourth model adds an approach for a genre effect. Though an appropriate specification for genre effects should consider each genre separately (as we will see in the final section for future work), due to computer capacity limitations, we can take each possible combinations of genre categories to include an approximation for this effect like this:

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

Where:

b_g: genre-specific effects. Its least squares estimate $\hat{b}_g$ is the average of the difference between $Y_{u,i}$, $\hat{\mu}$, $\hat{b}_i$ and $\hat{b}_u$.

The first four models don't take into account the fact that some movies were rated by just a few users, generating small sample sizes for them. A way to include this fact in our analysis is by penalizing small sample sizes using regularization. Thus, the fifth model considers a penalty term for movie effects.

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

Where:

$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i)^2$: least squares estimate $\hat{b}_i$.

$\lambda \sum_i b_i^2$: penalty that gets larger when various b_i are large. λ is the penalty term.

In this model, the estimate that minimizes the equation is:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

.

Where:

n_i: number of ratings made for movie i.

We can also add a penalty term for user effects. So, our sixth model looks like this:

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

Where: $\frac{1}{N} \sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2$: least squares estimate $\hat{b}_u$. $\lambda(\sum_i b_i^2 - \sum_u b_u^2)$: penalty for user effects.

In this model, the estimate that minimizes the equation is:

$$\hat{b}_u = \frac{1}{\lambda + n_u} \sum_{i=1}^{n_u} (Y_{u,i} - \hat{\mu} - \hat{b}_i)$$

.

Where:

n_u: number of ratings made for user u.

Finally, our seventh model adds a penalty term for genre effects like this:

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

Where: $\frac{1}{N} \sum_{u,i} (y_{u,i} - \mu - b_i - b_u - b_g)^2$: least squares estimate $\hat{b}_g$. $\lambda(\sum_i b_i^2 - \sum_u b_u^2 - \sum_g b_g^2)$: penalty for genre effects.

In this model, the estimate that minimizes the equation is:

$$\hat{b}_g = \frac{1}{\lambda + n_g} \sum_{i=1}^{n_g} (Y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

.

Where:

n_g: number of ratings made for genre g.

It is important to note that the penalty term λ is a tuning parameter. This means that we can select different values of λ in our models and see which one minimizes the RMSE. To select the optimal λ, we can use cross-validation.

Chapter 3

Results

In this section, we will see the results of the models explained above and select the one that performs best in terms of RMSE.

Remember that the aim of this project is to create a movie recommendation system with the lowest possible RMSE (specifically, RMSE less than 0.86490). Therefore, we need to create a function that calculates the RMSE to evaluate the performance of each model. We know that the RMSE is defined as:

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

Where:

N: number of user/movie combinations and the sum occurring over all these combinations.

$\hat{y}_{u,i}$: predicted rating.

$y_{u,i}$: observed rating.

Thus, we can use the following code to create the RMSE function.

```r
# Create a function that computes the RMSE for vectors of ratings
# and their corresponding predictors
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

3.1 Model Results

For the first model, we calculate the average rating in the train set. We consider this average as the prediction for all ratings. So, we use it to evaluate the RMSE in the test set. We can denominate it as a naive RMSE.

```
#############################
# Training different models #
#############################

###############
# First model #
###############
# Assumes the same rating for all movies and users
# with all the differences explained by random variation
# Average of all ratings
mu_hat <- mean(train_set$rating)
mu_hat
```

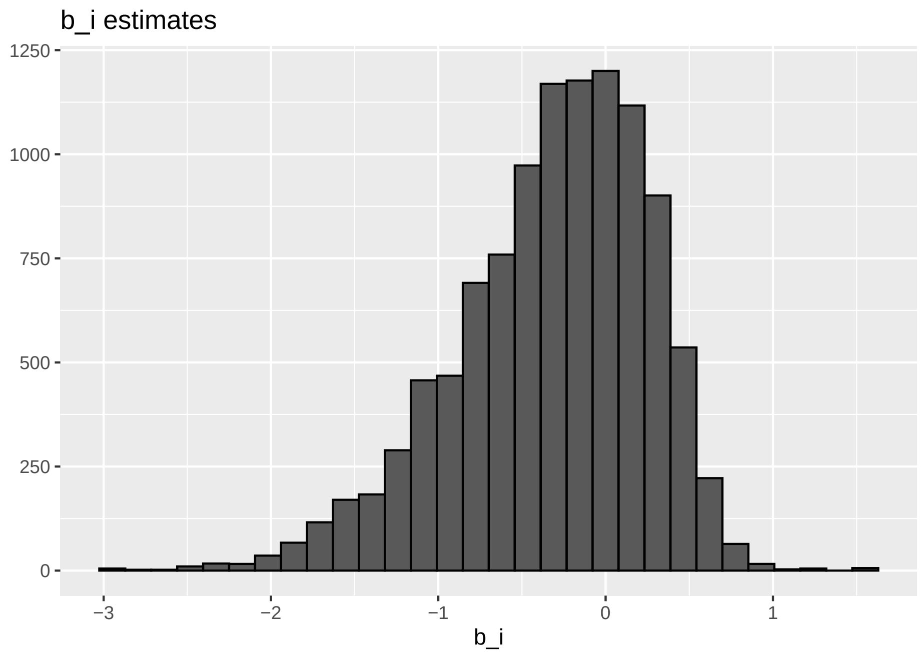
```
## [1] 3.512456
```

```
#Calculate a naive RMSE
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060054
```

In the second model, we include the movie effect. We estimate it in the train set. We can see how it varies graphically. Next, we use it to predict ratings in the test set and estimate the RMSE.

```
################
# Second model #
################
# Model with movie effects
# Estimate average ranking for movie  i
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# See movie effect estimates graphically
qplot(b_i, data = movie_avgs, bins = 30, color = I("black"),
      main = "b_i estimates")
```

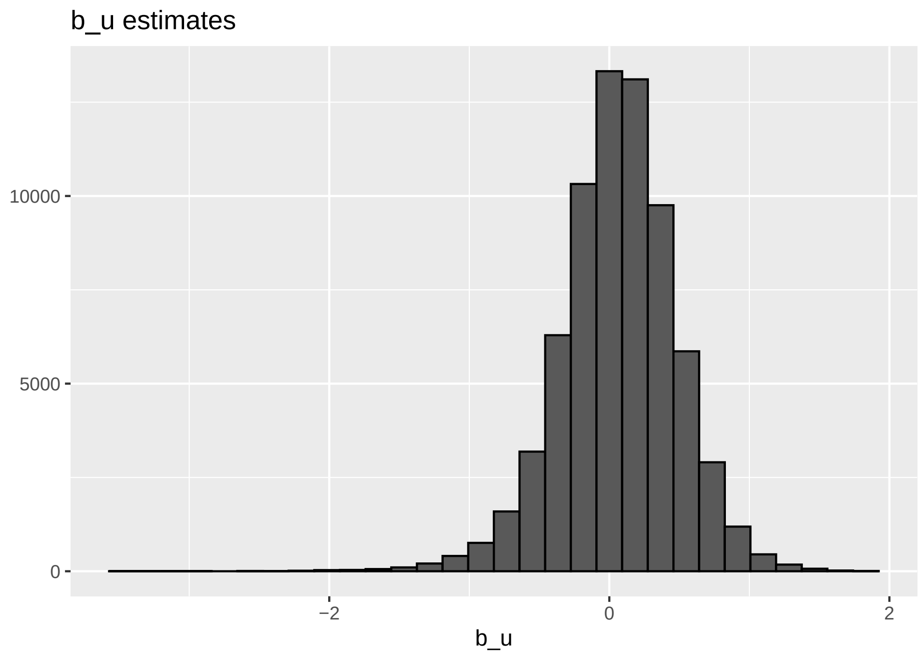


```
# Predict ratings with movie effects
predicted_ratings_m <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# Estimate RMSE
m_rmse <- RMSE(test_set$rating, predicted_ratings_m)
m_rmse
```

```
## [1] 0.9429615
```

For the third model, in addition to the movie effects, we add user effects. Both effects are estimated in the train set and are used to predict ratings in the test set. Then, we estimate the RMSE. We can see also the how user effects are distributed graphically.

```
###############
# Third model #
###############
# Model with movie and user effects
# Estimate average ranking for user u
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# See user effect estimates graphically
qplot(b_u, data = user_avgs, bins = 30, color = I("black"),
      main = "b_u estimates")
```

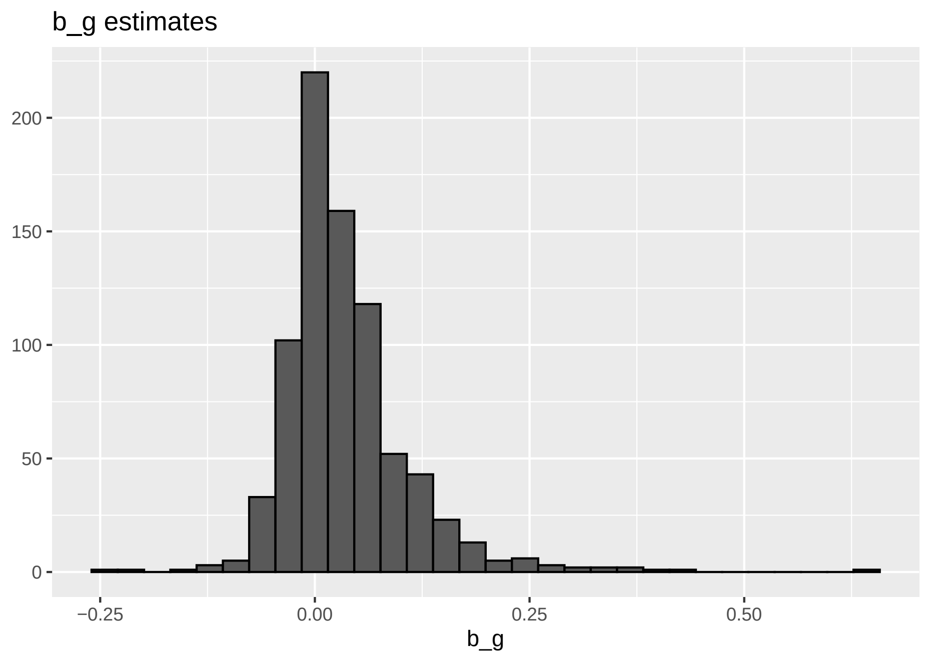


```
# Predict ratings with movie and user effects
predicted_ratings_m_u <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Estimate RMSE
m_u_rmse <- RMSE(test_set$rating, predicted_ratings_m_u)
m_u_rmse
```

```
## [1] 0.8646843
```

For the fourth model, we include an approximation for genre effects. The three effects are estimated in the train set and are used to predict ratings in the test set. Then, we estimate the RMSE. We can see also the how genre effects are distributed graphically.

```
################
# Fourth model #
################
# Model with movie, user and genre effects
# Estimate genre-specific effect
mu <- mean(train_set$rating)
genre_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

# See genre effect estimates graphically
qplot(b_g, data = genre_avgs, bins = 30, color = I("black"),
      main = "b_g estimates")
```

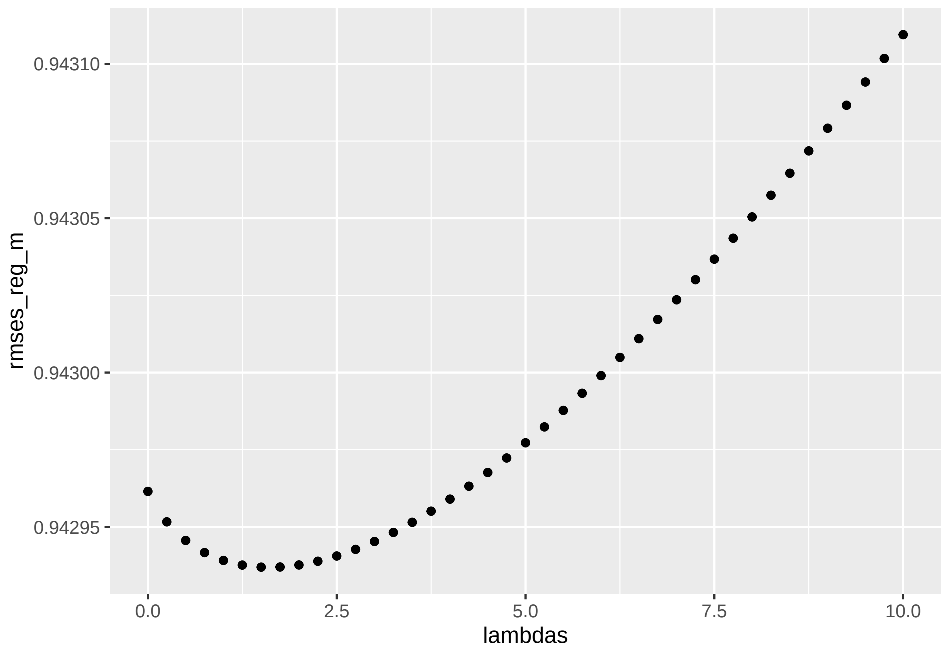


```
# Predict ratings with genre effects
predicted_ratings_m_u_g <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Estimate RMSE
m_u_g_rmse <- RMSE(test_set$rating, predicted_ratings_m_u_g)
m_u_g_rmse
```

```
## [1] 0.8643241
```

For the fifth, sixth and seventh models, we consider penalty terms to take into consideration the presence of movies with very few ratings in the data. In the fifth model, we add a penalty term for movie effects; in the sixth model, we also include a penalty term for user effects and; in the seventh model, we add a penalty term for genre effects. As the penalty term is a tuning parameter, we use cross validation to select the optimal penalty term.

```r
###############
# Fifth model #
###############
# Model with penalized least squares (regularization)
# Choosing the penalty term for movie effects using
# cross-validation
lambdas <- seq(0, 10, 0.25)

mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses_reg_m <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(test_set$rating, predicted_ratings))
})

qplot(lambdas, rmses_reg_m)
```



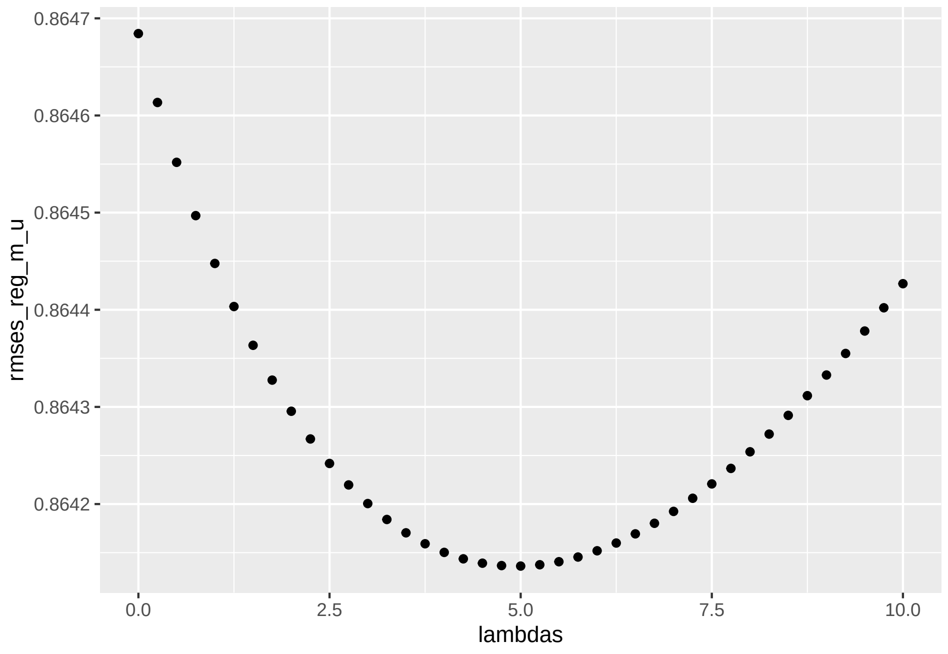
```r
lambdas[which.min(rmses_reg_m)]
```

```
## [1] 1.5
```

```r
min(rmses_reg_m)
```

```
## [1] 0.942937
```

```r
###############
# Sixth model #
###############
# Model with penalized least squares (regularization)
# Choosing the penalty term for movie and user effects using
# cross-validation
lambdas <- seq(0, 10, 0.25)

rmses_reg_m_u <- 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))
})

qplot(lambdas, rmses_reg_m_u)
```



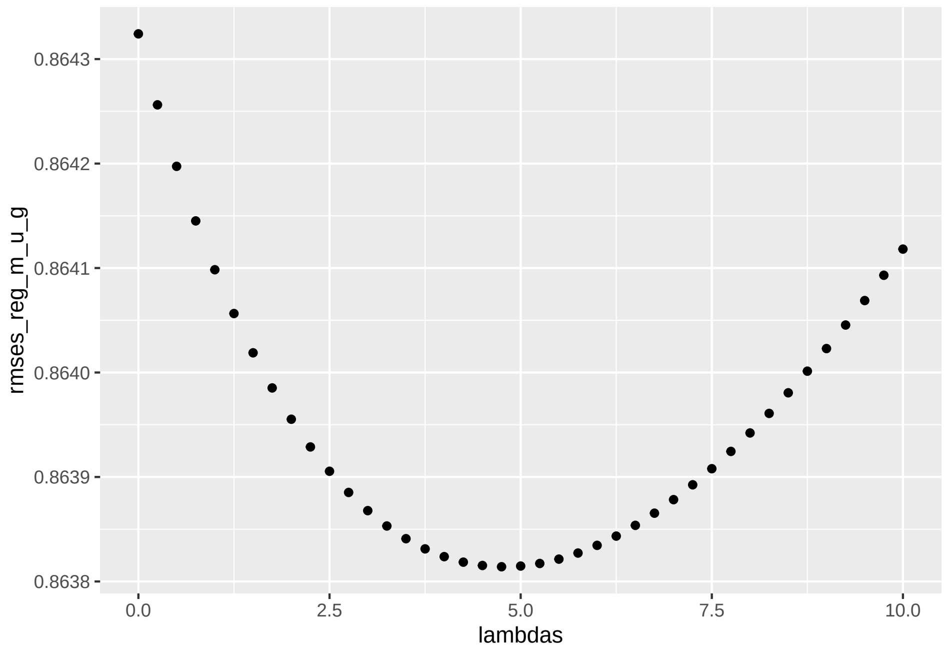
```
lambdas[which.min(rmses_reg_m_u)]
```

```
## [1] 5
```

```
min(rmses_reg_m_u)
```

```
## [1] 0.8641362
```

```
#################
# Seventh model #
#################
# Model with penalized least squares (regularization)
# Choosing the penalty term for movie, user and genre effects
# using cross-validation
lambdas <- seq(0, 10, 0.25)

rmses_reg_m_u_g <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)
  return(RMSE(test_set$rating, predicted_ratings))
})

qplot(lambdas, rmses_reg_m_u_g)
```



```
lambdas[which.min(rmses_reg_m_u_g)]
```

```
## [1] 4.75
```

```
min(rmses_reg_m_u_g)
```

```
## [1] 0.8638141
```

To choose a model for the final evaluation, we create a table that compares RMSE values of each model described above.

```
rmse_results <- tibble(method = c("Just the average", "Movie Effects Model",
                                  "Movie + User Effects Model",
                                  "Movie + User + Genre Effects Model",
                                  "Regularized Movie Effects Model",
                                  "Regularized Movie + User Effects Model",
                                  "Regularized Movie + User + Genre Effects Model"),
                       RMSE = c(naive_rmse, m_rmse, m_u_rmse,
                                m_u_g_rmse, min(rmses_reg_m),
                                min(rmses_reg_m_u),
                                min(rmses_reg_m_u_g)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effects Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Regularized Movie Effects Model	0.9429370
Regularized Movie + User Effects Model	0.8641362
Regularized Movie + User + Genre Effects Model	0.8638141

We see that RMSE improved substantially when adding movie, user and genre effects into the model. The lowest RMSE is achieved with regularized movie, user and genre effects. So, we will use this model in the final evaluation using the entire edx set and the validation set in the next section.

3.2 Final Model

For the final evaluation, we use all the edx set to train the model. Then, we predict movie ratings and calculate the final RMSE with the validation set.

```
######################
# Chosen final model #
######################
# Estimate regularized average ranking for movie, user
# and genre effects with optimal lambda
op_lambda <- lambdas[which.min(rmses_reg_m_u_g)]
movie_reg_bi <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+op_lambda), n_i = n())
movie_reg_bu <- edx %>%
  left_join(movie_reg_bi, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+op_lambda),
            n_i = n())
movie_reg_bg <- edx %>%
  left_join(movie_reg_bi, by="movieId") %>%
  left_join(movie_reg_bu, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+op_lambda),
            n_i = n())

# Predict movie ratings in validation set
predicted_ratings_reg <- validation %>%
  left_join(movie_reg_bi, by='movieId') %>%
  left_join(movie_reg_bu, by='userId') %>%
  left_join(movie_reg_bg, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

# Estimate RMSE in validation set
final_RMSE <- RMSE(validation$rating, predicted_ratings_reg)
final_RMSE
```

```
## [1] 0.8644514
```

Chapter 4

Conclusion

In this document, we developed a recommendation system for movies using the MovieLens database. To this aim, we first split the database into two sets: edx and validation. Then, we did some data exploration to understand better the data structure.

Edx set was used to train different models, so it was split again into two sets: train and test. Performance of each model was evaluated in terms of its RMSE.

For the first model, we considered only the average movie rating, the RMSE was 1.060054, which means we might miss ratings by one or more stars. In the second model, we added movie effects, reducing RMSE to 0.9429615, an improvement of 11%. The third model added user effects, obtaining an RMSE of 0.8646843, an improvement of 18.4% with respect to the first model. The fourth model used an approximation for a genre effect, considering the possible genre combinations for movies. It obtained an RMSE of 0.8643241, an improvement of 18.5%. The fifth, sixth and seventh models incorporated penalty terms for movie, user and genre effects. RMSEs were 0.942937, 0.8641362 and 0.8638141 for them, an improvement of 11%, 18.5% and 18.5%, respectively.

Taking into account these results, we selected the last model as the final model. We used the entire edx set to train it and the validation set for the final evaluation. The final RMSE was 0.8644514, which is less than 0.86490, achieving the goal of this project.

It is important to note that there were some limitations in running this project. The large size of edx and validation sets, much larger than usually manipulated in other courses, required more demanding processing capacity than could be achieved in a basic computer. This situation made it difficult to run more complex models than one would have originally wanted to do.

Considering these limitations, there are two models, presented in the Machine Learning course, that could be contemplated for future work: (i) the incorporation of a smooth function for the day $d_{u,i}$ for user's u rating of movie i and, (ii) a more formal definition of a genre effect $g_{u,i}$ for user's u rating of movie i than the one used in this project. Moreover, there are another packages, such as recommenderlab, which permits to test and develop different recommender algorithms, and recosystem, which uses matrix factorization for recommendation systems. Both packages could also help us to improve our recommendation system.