MovieLens Recommender System Capstone Project_NF

Nelson Marcelo Ferreira Berg

Contents

1 Summary **2**

2 Exploratory Data Analysis **3**

2.1 Data Preparation 3

2.2 Main basic data exploration 7

2.3 Rating distribution vs Users and Movies 11

2.4 Processing data 13

2.5 Rating exploration 15

3 Modeling **21**

3.1 Loss Function - RMSE 21

3.2 Naive Model 21

3.3 Movie Effect Model 22

3.4 Movie + User Effects Model 24

3.5 Penalized least squares 26

4 Penalized Movie + User Effects Model **27**

5 Validation **30**

6 Results **32**

7 Conclusion **33**

Chapter 1

Summary

The purpose of this project for the Capstone course of the HarvardX Professional Certificate in Data Science (PH125.9x), is to explore and visually examine the MovieLens database of GroupLens Research which features over 10 million film ratings.

MovieLens database contains 10000054 movies ratings. The database then divided in two parts 9 million are used for training and 1 million for validation. In the training database, there are 69878 users and 10677 movies, divided in 20 different genres (a movie can have more than one genre).

Our goal for this project is to minimize the Root Mean Squared Error (RMSE) at least to 0.86549 (RMSE <= 0.86549)

Chapter 2

Exploratory Data Analysis

2.1 Data Preparation

Data preparation consists in installing and loadind the required packages that we are going to use in the project. Also, we download the MovieLens database provided by Proffessor Rafael Irizarry which we are going to use to carry on with the project.

```
# install the necessary packages
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
if(!require(kableExtra)) install.packages("kableExtra")
```

```
## Loading required package: kableExtra

## Warning: package 'kableExtra' was built under R version 4.1.2

##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
if(!require(dplyr)) install.packages("dplyr")
```

```r
if(!require(dslabs)) install.packages("dslabs")
```

```
## Loading required package: dslabs
```

```r
if(!require(tidyr)) install.packages("tidyr")
```

```r
if(!require(stringr)) install.packages("stringr")
```

```r
if(!require(forcats)) install.packages("forcats")
```

```r
if(!require(ggplot2)) install.packages("ggplot2")
```

```r
if(!require(tidyr)) install.packages("lubridate")
```

```r
# Loading libraries
library(tidyverse)
library(caret)
library(data.table)
library(kableExtra)
library(dplyr)
library(dslabs)
library(stringr)
library(forcats)
library(ggplot2)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
# 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
                 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) %>% 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) # 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)
```

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

```r
# Joining columns
edx <- rbind(edx, removed)
```

MovieLens contains 10000054 movies ratings. The data is divided in two parts, one part used for training which is called **edx**, and the other for testing called **validation**.

With the **edx** database we are going to train models in order to select the most appropriate to test it in the validation database.

The **edx** database is divided in **train_set** and **test_set** to train and test the modeling.

```r
# Test_set set will be 10% of Edx data
test_index2 <- createDataPartition(y = edx$rating, times = 1, p = 0.1,
                                   list = FALSE)
train_set <- edx[-test_index2,]
temp2 <- edx[test_index2,]

# Make sure userId and movieId in test_set set are also in train_set
test_set <- temp2 %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test_set set back into train_set
removed_2 <- anti_join(temp2, test_set)
```

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

```r
# Joining the columns
train_set <- rbind(train_set, removed_2)
```

2.2 Main basic data exploration

Edx Database

```r
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

```r
glimpse(edx)
```

```
## Rows: 9,000,061
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 370, 377,
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 838983392, 8389
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumber (1994
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy", "Actio
```

```r
edx %>% summarize(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

validation Database

```r
glimpse(validation)
```

```
## Rows: 999,993
## Columns: 6
## $ userId    <int> 1, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 7,
## $ movieId   <dbl> 588, 1210, 1544, 151, 1288, 5299, 380, 435, 480, 477, 508,
## $ rating    <dbl> 5.0, 4.0, 3.0, 4.5, 3.0, 3.0, 3.0, 3.0, 5.0, 3.0, 3.0, 4.0,
## $ timestamp <int> 838983339, 868245644, 868245920, 1133571026, 1133571035, 11
## $ title     <chr> "Aladdin (1992)", "Star Wars: Episode VI - Return of the Je
## $ genres    <chr> "Adventure|Animation|Children|Comedy|Musical", "Action|Adve
```

```r
validation %>% summarize(n_users = n_distinct(userId),
                         n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   68531     9796
```

train_set Database

```r
head(train_set)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

```r
glimpse(train_set)
```

```
## Rows: 8,100,070
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 370, 377, 420,
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 838983392, 8389
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumber (1994
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy", "Actio
```

```r
train_set %>% summarize(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

test_set Database

```r
head(test_set)
```

```
##    userId movieId rating  timestamp
## 1:      1     364      5  838983707
## 2:      1     616      5  838984941
## 3:      2     736      3  868244698
## 4:      2    1049      3  868245920
## 5:      3     213      5 1136075789
## 6:      3    1246      4 1133570967
##                                        title
## 1:                          Lion King, The (1994)
## 2:                         Aristocats, The (1970)
## 3:                                 Twister (1996)
## 4:             Ghost and the Darkness, The (1996)
## 5: Burnt by the Sun (Utomlyonnye solntsem) (1994)
## 6:                    Dead Poets Society (1989)
##                                        genres
## 1: Adventure|Animation|Children|Drama|Musical
## 2:                         Animation|Children
```

```
## 3:           Action|Adventure|Romance|Thriller
## 4:                            Action|Adventure
## 5:                                       Drama
## 6:                                       Drama
```

```
glimpse(test_set)
```

```
## Rows: 899,991
## Columns: 6
## $ userId    <int> 1, 1, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5,
## $ movieId   <dbl> 364, 616, 736, 1049, 213, 1246, 1674, 5527, 6287, 33750, 25
## $ rating    <dbl> 5.0, 5.0, 3.0, 3.0, 5.0, 4.0, 4.5, 4.5, 3.0, 3.5, 3.0, 5.0,
## $ timestamp <int> 838983707, 838984941, 868244698, 868245920, 1136075789, 113
## $ title     <chr> "Lion King, The (1994)", "Aristocats, The (1970)", "Twister
## $ genres    <chr> "Adventure|Animation|Children|Drama|Musical", "Animation|Ch
```

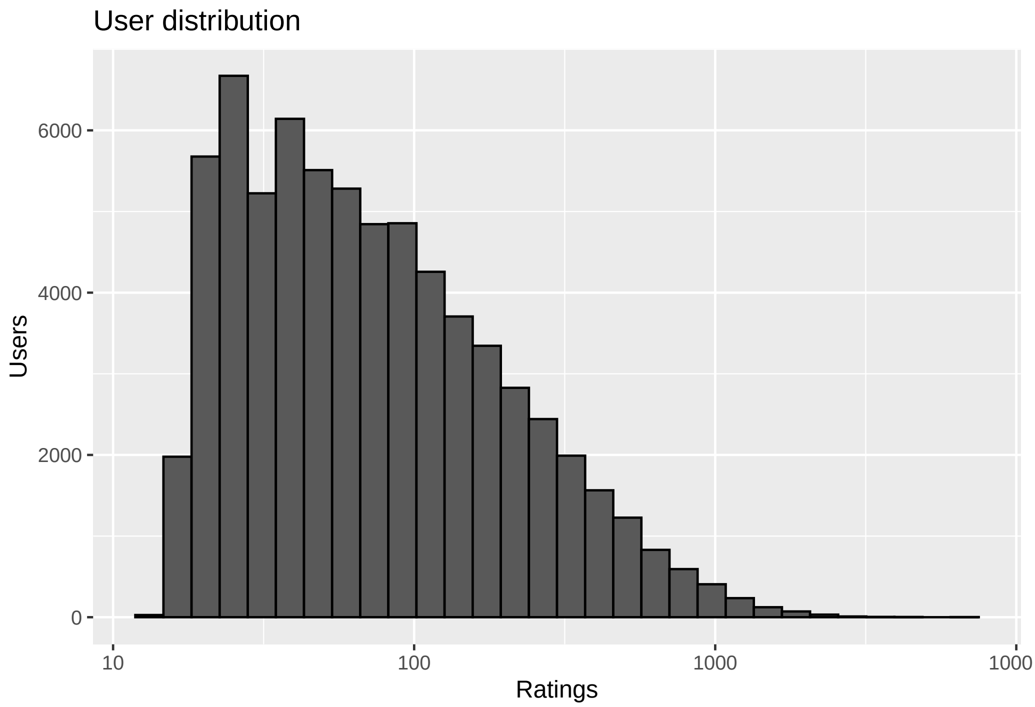
```
test_set %>% summarize(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   68008     9736
```

2.3 Rating distribution vs Users and Movies

Examining distribution of ratings by users.

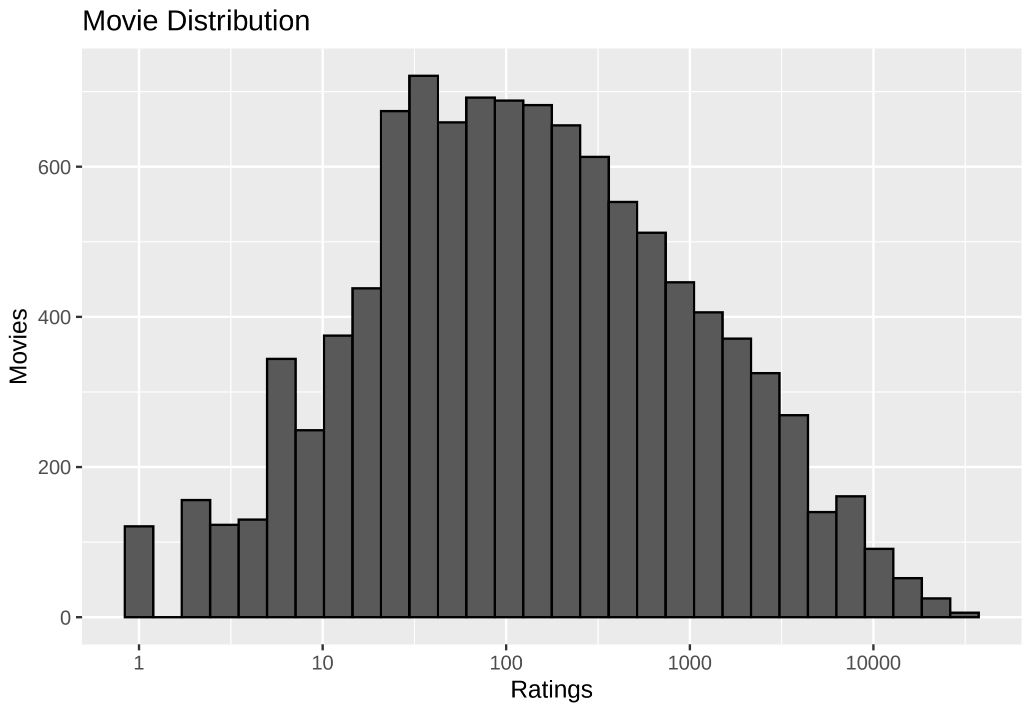
```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, color = "black") +
  scale_x_log10() +
  ggtitle("User distribution") +
  labs(x="Ratings", y="Users")
```



We can observe that most users do not rate a lot of movies.

Examining distribution of ratings by movies.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram( bins=30, color = "black") +
  scale_x_log10() +
  ggtitle("Movie Distribution") +
  labs(x="Ratings" , y="Movies")
```



We can observe a nearly normal distribution which is not a surprise, since the most popular movies are usually rated more often than the least popular movies. Having movies with fewer ratings for sure may affect the recommendation system.

2.4 Processing data

For a better data exploration and analysis, in this section we modify: * the title to separate the release year of the movie * the timestamp column to get the rating year and month

This is done in order to get a more precise prediction of movie rating in our model.

```
# Convert the timestamp to **ratingyear**and ratingmonth in the edx database
edx$date <- as.POSIXct(edx$timestamp, origin="1970-01-01")

edx$ratingyear <- format(edx$date, "%Y")
edx$ratingyear <- as.numeric(edx$ratingyear)
```

```r
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres                date ratingyear
## 1:                Comedy|Romance 1996-08-02 07:24:06       1996
## 2:         Action|Crime|Thriller 1996-08-02 06:58:45       1996
## 3:                        Comedy 1996-08-02 06:56:32       1996
## 4:  Action|Drama|Sci-Fi|Thriller 1996-08-02 06:57:01       1996
## 5:       Action|Adventure|Sci-Fi 1996-08-02 06:56:32       1996
## 6: Action|Adventure|Drama|Sci-Fi 1996-08-02 06:56:32       1996
```

```r
# Convert the timestamp to **ratingyear**and ratingmonth in the validation data
validation$date <- as.POSIXct(validation$timestamp, origin="1970-01-01")

validation$ratingyear <- format(validation$date, "%Y")
validation$ratingyear <- as.numeric(validation$ratingyear)


head(validation)
```

```
##    userId movieId rating  timestamp
## 1:      1     588    5.0  838983339
## 2:      2    1210    4.0  868245644
## 3:      2    1544    3.0  868245920
## 4:      3     151    4.5 1133571026
## 5:      3    1288    3.0 1133571035
## 6:      3    5299    3.0 1164885617
##                                                  title
## 1:                                      Aladdin (1992)
## 2:       Star Wars: Episode VI - Return of the Jedi (1983)
## 3: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
## 4:                                      Rob Roy (1995)
## 5:                             This Is Spinal Tap (1984)
```

```
## 6:                         My Big Fat Greek Wedding (2002)
##                                              genres                 date ratingyear
## 1: Adventure|Animation|Children|Comedy|Musical 1996-08-
02 06:55:39       1996
## 2:                    Action|Adventure|Sci-Fi 1997-07-
06 23:20:44       1997
## 3:     Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-
06 23:25:20       1997
## 4:                   Action|Drama|Romance|War 2005-12-
02 21:50:26       2005
## 5:                             Comedy|Musical 2005-12-
02 21:50:35       2005
## 6:                             Comedy|Romance 2006-11-
30 08:20:17       2006
```

Removing irrelevant columns

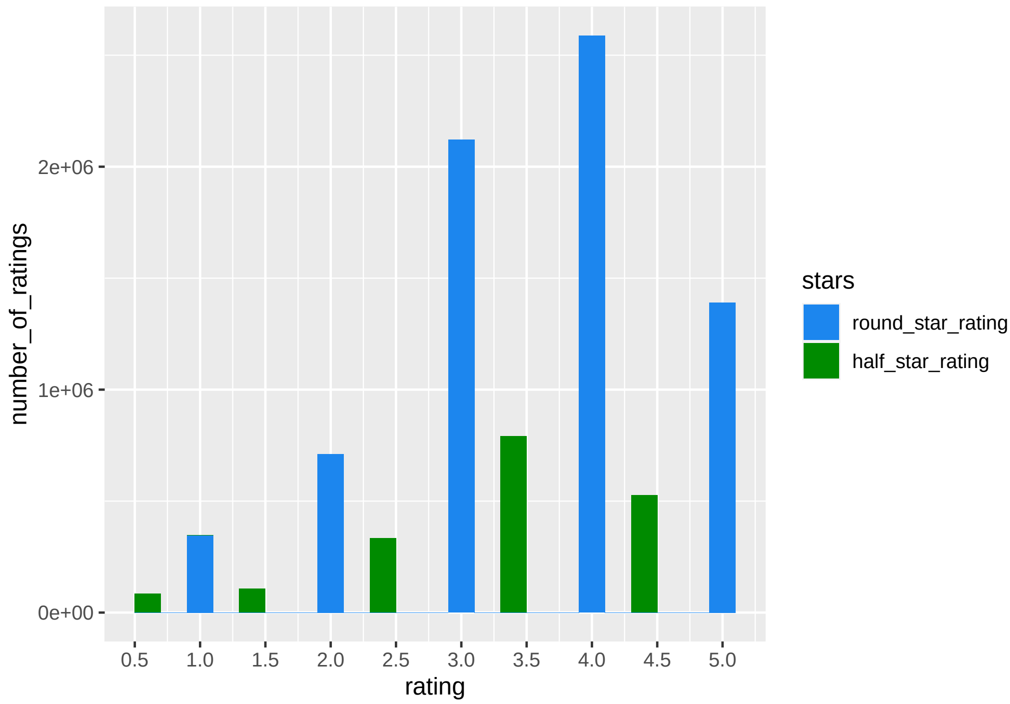
```
# edx database
edx <- edx %>%
select(-timestamp, -date)

# validation database
validation <- validation %>%
select(-timestamp, -date)
```

```
# Release year in a new column

edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

2.5 Rating exploration

```
# Data frame that contains movie ratings from edx data
stars <- ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                   edx$rating == 4 | edx$rating == 5) ,
                "round_star_rating",
                "half_star_rating")
```

```
movie_ratings <- data.frame(edx$rating, stars)

# Number of ratings per rating
ggplot(movie_ratings, aes(x = edx.rating, fill = stars)) +
  geom_histogram(binwidth = 0.2) +
  scale_x_continuous(name = "rating", breaks = seq(0, 5, by = 0.5)) +
  scale_y_continuous(name = "number_of_ratings") +
    scale_fill_manual(values = c("round_star_rating"="dodgerblue2",
                                 "half_star_rating"="green4"))
```



Our data visualization reveals us that most users tend to rate movies when they consider the movie is a 3 star o more. It is not common for users to rate "0" stars for a movie. The histogram also shows us that half-stars ratings are less common.

```
# Summary of rating count
edx %>% group_by(rating) %>% summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 2
```

```
##    rating   count
##     <dbl>   <int>
##  1    4   2588021
##  2    3   2121638
##  3    5   1390541
##  4    3.5  792037
##  5    2    710998
##  6    4.5  526309
##  7    1    345935
##  8    2.5  332783
##  9    1.5  106379
## 10    0.5   85420
```

Additionally, we can observe that 4,3,5, 3.5 and 2 are the top ratings by users.

```r
edx %>% group_by(rating, ratingyear) %>% summarize(count = n()) %>%
  arrange(desc(count))
```

```
## `summarise()` has grouped output by 'rating'. You can override using the `.gr
```

```
## # A tibble: 106 x 3
## # Groups:   rating [10]
##    rating ratingyear  count
##     <dbl>      <dbl>  <int>
##  1    4         2000 397903
##  2    3         1996 388572
##  3    3         2000 297856
##  4    4         1996 278440
##  5    5         2000 259702
##  6    4         1999 248444
##  7    4         2005 246146
##  8    4         2001 238558
##  9    3.5       2005 201679
## 10    3         2001 185806
## # ... with 96 more rows
```

```r
# Top 10 most rated titles
edx %>%
```

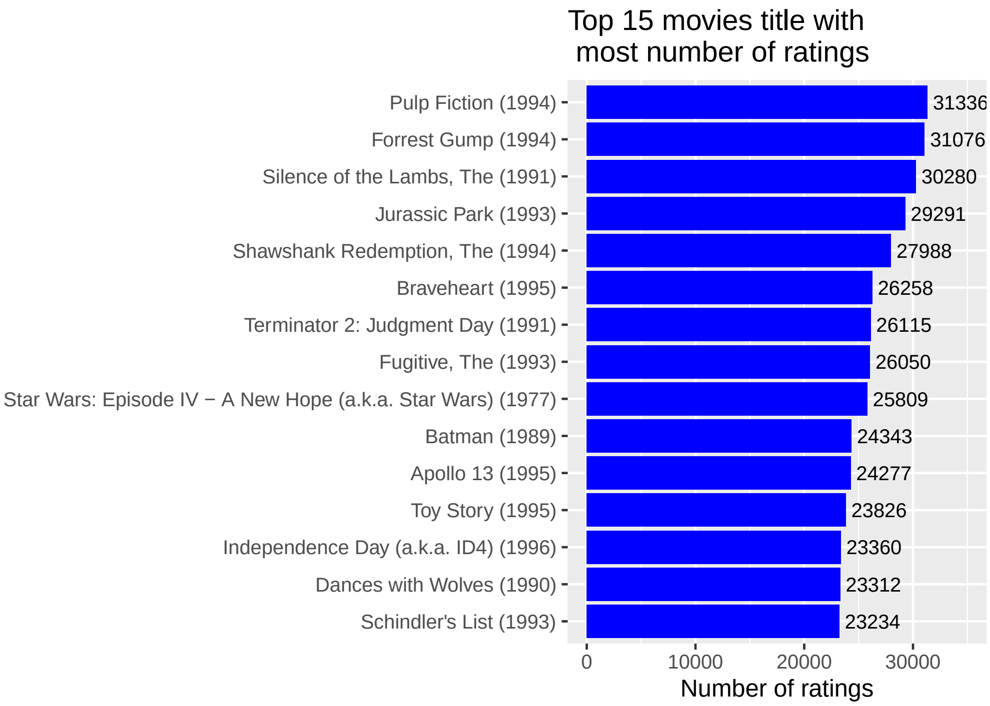
```
  group_by(title) %>%
  summarize(count=n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,676 x 2
##    title                                                        count
##    <chr>                                                        <int>
##  1 Pulp Fiction (1994)                                          31336
##  2 Forrest Gump (1994)                                          31076
##  3 Silence of the Lambs, The (1991)                             30280
##  4 Jurassic Park (1993)                                         29291
##  5 Shawshank Redemption, The (1994)                             27988
##  6 Braveheart (1995)                                            26258
##  7 Terminator 2: Judgment Day (1991)                            26115
##  8 Fugitive, The (1993)                                         26050
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25809
## 10 Batman (1989)                                                24343
## # ... with 10,666 more rows
```

```
# Top listed genres
edx%>%
  group_by(genres) %>% summarize(Ratings_Sum = n(), Average_Rating = mean(rating
    arrange(-Ratings_Sum)
```

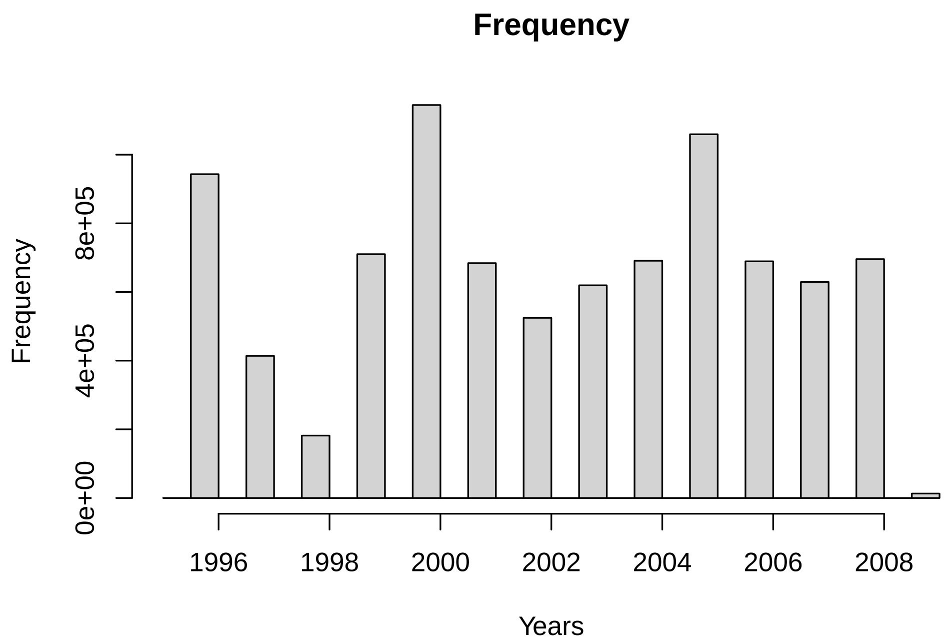
```
## # A tibble: 797 x 3
##    genres                     Ratings_Sum Average_Rating
##    <chr>                            <int>          <dbl>
##  1 Drama                           733353           3.71
##  2 Comedy                          700883           3.24
##  3 Comedy|Romance                  365894           3.41
##  4 Comedy|Drama                    323518           3.60
##  5 Comedy|Drama|Romance            261098           3.65
##  6 Drama|Romance                   259735           3.61
##  7 Action|Adventure|Sci-Fi         220363           3.51
##  8 Action|Adventure|Thriller       148933           3.43
##  9 Drama|Thriller                  145359           3.44
## 10 Crime|Drama                     137424           3.95
## # ... with 787 more rows
```

```r
# Movies with the major number of ratings
most_rated_titles <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(15,count) %>%
  arrange(desc(count))

most_rated_titles %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat= "identity", fill="blue") +
  coord_flip(y=c(0, 35000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Top 15 movies title with \n most number of ratings")
```



We can observe that the movies with the highest amount of ratings are 90s movies

```r
# Frequency of ratings per year
edx$ratingyear %>%
  hist(main="Frequency", xlab="Years")
```



This shows us in which years users gave more ratings.

Chapter 3

Modeling

In this section we use **train_set** for some models for the recommender system and after training the models, we use the best to test it with the validation dataset.

3.1 Loss Function - RMSE

Root-mean-square error (RMSE) is a formula to measure the differences between values predicted by a model or an estimator and the values observed. Our goal in this project is to have RMSE <= 0.86549. RMSE defined as:

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

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

```
RMSE <- function(true_ratings = NULL, predicted_ratings = NULL){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

3.2 Naive Model

The simplest model for the recommendation system consists in predicting the same ratings for all movies despite the users.

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

```
train_mu_hat <- mean(train_set$rating)
train_mu_hat
```

```
## [1] 3.512451
```

If we predict all unknown ratings with muhat the result of the RMSE is the following:

```
nrmse <- RMSE(train_set$rating, train_mu_hat)
nrmse
```

```
## [1] 1.060464
```

With the Naive Model, our RMSE 1.060464 which is very far from our goal 0.86549

```
prediction_results <-  data.frame(model="Naive Model", RMSE=nrmse)
```

3.3 Movie Effect Model

Some movies are rated higher than others which is confirmed by our data. To build a more precise model we will use b_i in the function representing the average rating for movie i. In the Netflix challenge papers b notation is referred as "bias".

The formula is the following:

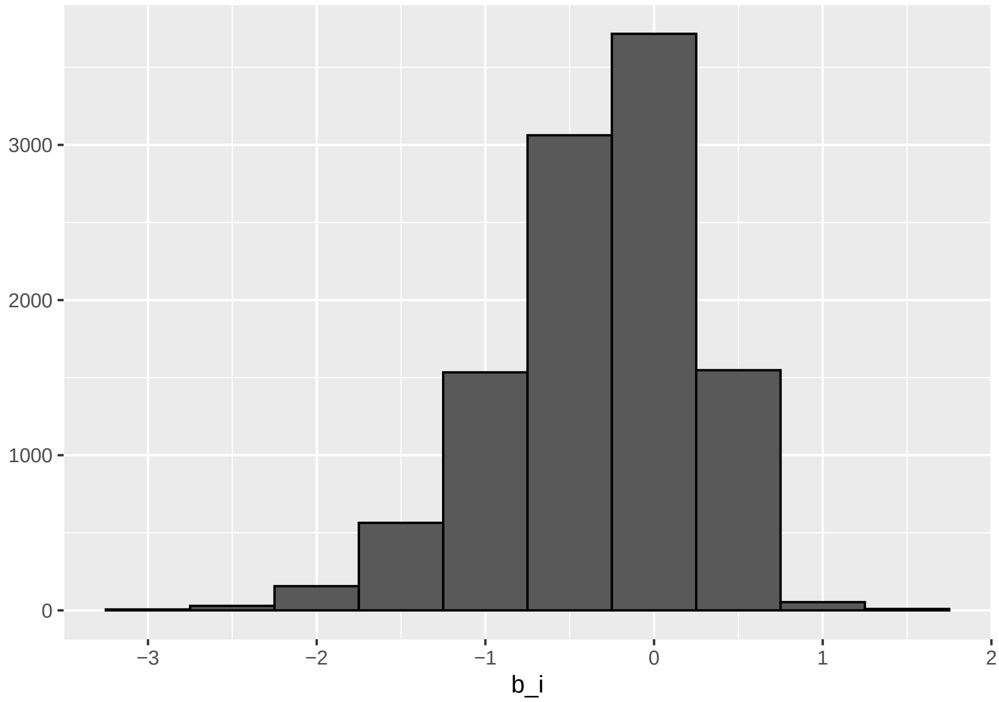
$$Y_{u,i} = \hat{\mu} + b_i + \epsilon_{u,i}$$

```
train_mu_hat <- mean(train_set$rating)
train_mu_hat
```

```
## [1] 3.512451
```

```
bi <- train_set %>% # Average rating by movie
 group_by(movieId) %>%
 summarize(b_i = mean(rating - train_mu_hat))
```

```
qplot(b_i, data = bi, bins = 10, color = I("black"))
```



Building the prediction with the movie model:

```
prediction_bi <-train_mu_hat + test_set %>%
  left_join(bi, by = "movieId") %>% .$b_i
m_rmse <- RMSE(prediction_bi, test_set$rating)
m_rmse
```

```
## [1] 0.9424221
```

The result from Movie Effect Model **(0.9424221)** is much closer to what is our goal.

```r
prediction_results <- prediction_results %>%
  add_row(model = "Movie Effect Model", RMSE = m_rmse)
prediction_results
```

```
##                 model      RMSE
## 1         Naive Model 1.0604640
## 2 Movie Effect Model 0.9424221
```

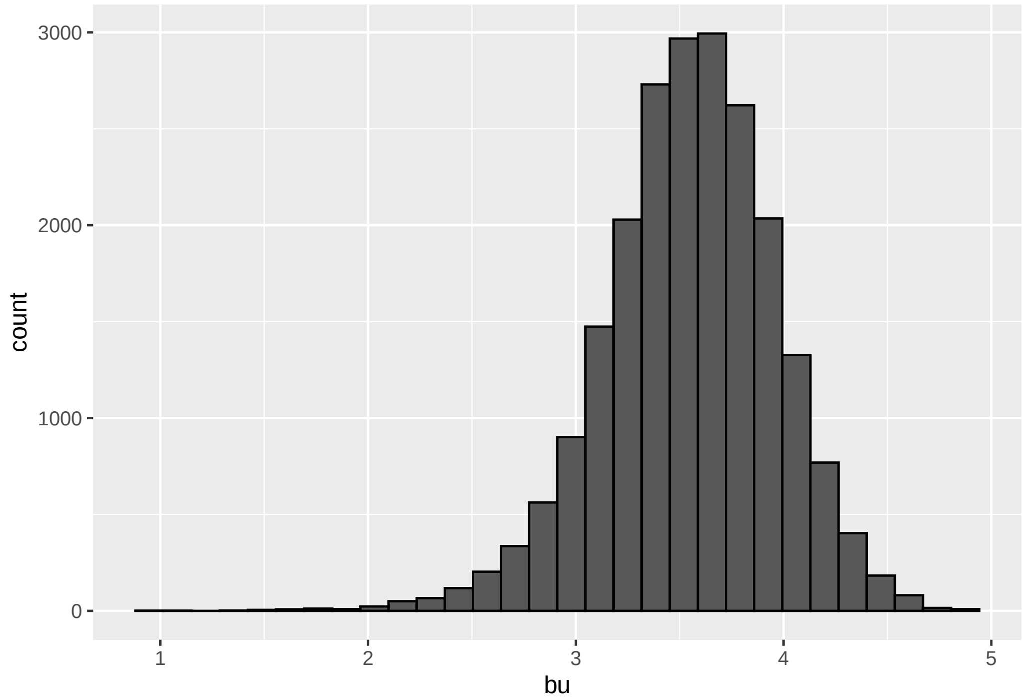
3.4 Movie + User Effects Model

In this model we introduce user effects assuming that all users tend to rate movies according to their personal standards. We add to the formula the user effect b_u

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

We observe the average rating for user u for those that have rated 100 movies or more.

```r
train_set %>%
 group_by(userId) %>%
 filter(n()>=100) %>%
 summarize(bu = mean(rating)) %>%
 ggplot(aes(bu)) +
 geom_histogram(bins = 30, color = "black")
```




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

```
## [1] 3.512451
```

We calculate the average rating per user.

```r
bu <-train_set %>%
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - train_mu_hat - b_i))
```

Building the prediction with users:

```r
prediction_bu <- test_set %>%
 left_join(bi, by='movieId') %>%
 left_join(bu, by='userId') %>%
 mutate(predictions = train_mu_hat + b_i + b_u) %>%
 pull(predictions)
um_rmse <- RMSE(prediction_bu, test_set$rating)
um_rmse
```

```
## [1] 0.8645357
```

```r
prediction_results <- prediction_results %>%
  add_row(model = "Movie + User effects", RMSE = um_rmse)
prediction_results
```

```
##                  model      RMSE
## 1          Naive Model 1.0604640
## 2   Movie Effect Model 0.9424221
## 3 Movie + User effects 0.8645357
```

3.5 Penalized least squares

To penalize is to control the total variability of the movie effect. We minimize an equation that adds a penalty.

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

We see that the Movies + Users effects model is the one that gives us the best results.

```r
prediction_results
```

```
##                  model      RMSE
## 1          Naive Model 1.0604640
## 2   Movie Effect Model 0.9424221
## 3 Movie + User effects 0.8645357
```

Chapter 4

Penalized Movie + User Effects Model

```
train_mu_hat <- mean(train_set$rating)
train_mu_hat
```

```
## [1] 3.512451
```

```
lambdas <- seq(0, 10, 0.5)

# Modeling with Regularized Movie + User Effect Model

rmse_mu <- sapply(lambdas, function(l){
 b_i <- train_set %>%
 group_by(movieId) %>%
 summarize(b_i = sum(rating - train_mu_hat)/(n()+l)) # rating mean by movie
 b_u <- train_set %>%
 left_join(b_i, by="movieId") %>%
 group_by(userId) %>%
 summarize(b_u = sum(rating - b_i - train_mu_hat)/(n()+l)) # mean rating by use
 predicted_ratings <- test_set %>%
 left_join(b_i, by = "movieId") %>%
 left_join(b_u, by = "userId") %>%
 mutate(prediction = train_mu_hat + b_i + b_u) %>%
 pull(prediction)  # RMSE in the test_set database
```

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

rmse_mu_pen <- min(rmse_mu)
rmse_mu_pen
```

```
## [1] 0.8639383
```

```r
prediction_results <- prediction_results %>%
  add_row(model="Regularized Movie + User Effect Model", RMSE=rmse_mu_pen)
prediction_results
```

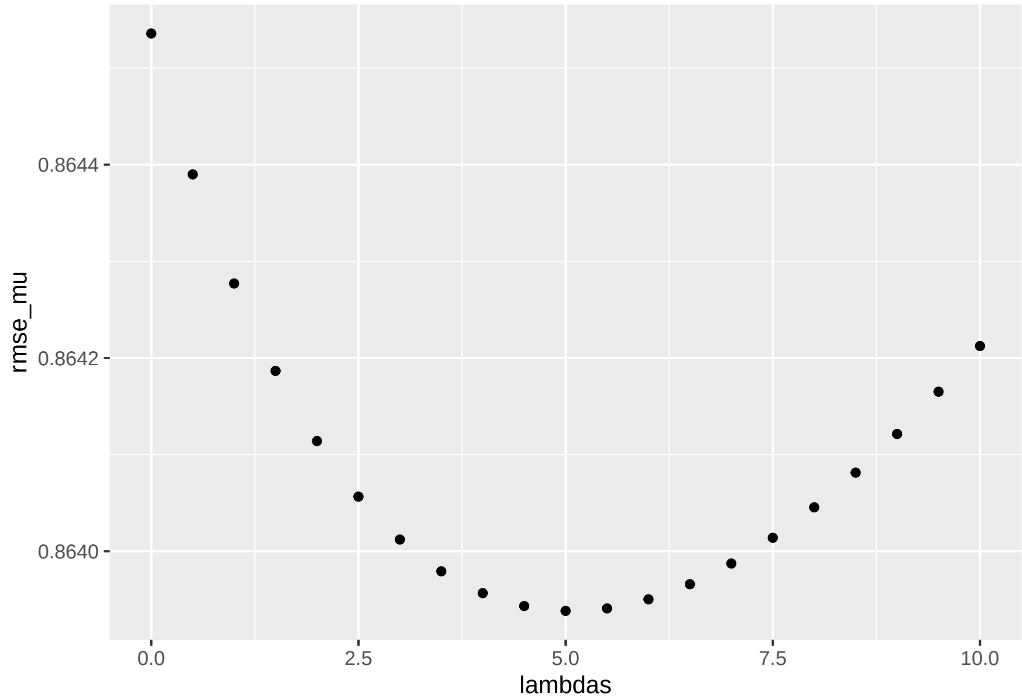
```
##                                   model      RMSE
## 1                           Naive Model 1.0604640
## 2                    Movie Effect Model 0.9424221
## 3                 Movie + User effects 0.8645357
## 4 Regularized Movie + User Effect Model 0.8639383
```

This is the lambda that minimizes the RMSE:

```r
lambdas[which.min(rmse_mu)] # The lambda value that minimize the RMSE
```

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

```r
qplot(lambdas, rmse_mu)
```



0.8644
0.8642
0.8640
rmse_mu
0.0
2.5
5.0
7.5
10.0
lambdas

Chapter 5

Validation

The validation section is the final step for the recommender system. Here we test the best model with the **validation** dataset.

We observe that the best model is the "Regularized Movie + User Effect Model"

```
prediction_results
```

```
##                                  model      RMSE
## 1                          Naive Model 1.0604640
## 2                   Movie Effect Model 0.9424221
## 3                 Movie + User effects 0.8645357
## 4 Regularized Movie + User Effect Model 0.8639383
```

Now we begin validating the most accurate model.

```
edx_muhat <- mean(edx$rating)
```

```
lambdas <- seq(0, 10, 0.1)

# Modeling with Regularized Movie + User Effect Model

rmse_mu_val <- sapply(lambdas, function(l){
 b_i <- edx %>%
 group_by(movieId) %>%
 summarize(b_i = sum(rating - edx_muhat)/(n()+l)) # mean rating by movie
```

```
 b_u <- edx %>%
 left_join(b_i, by="movieId") %>%
 group_by(userId) %>%
 summarize(b_u = sum(rating - b_i - edx_muhat)/(n()+l))  # mean rating by user
 predicted_ratings <- validation %>%
 left_join(b_i, by = "movieId") %>%
 left_join(b_u, by = "userId") %>%
 mutate(prediction = edx_muhat + b_i + b_u) %>%
 pull(prediction) # RMSE prediction on the validation database

 return(RMSE(predicted_ratings, validation$rating))
})

rmse_mu_pen_val <- min(rmse_mu_val)
rmse_mu_pen_val
```

```
## [1] 0.8649857
```

```
prediction_results <- prediction_results %>%
 add_row(model="Regularized Movie + User Effect Model Validated", RMSE=rmse_mu_p
```

Chapter 6

Results

We can observe the results of all the models we built. Finally, we reached to our goal that was to get a RMSE <= 0.86549.

```
prediction_results
```

```
##                                          model      RMSE
## 1                                  Naive Model 1.0604640
## 2                          Movie Effect Model 0.9424221
## 3                         Movie + User effects 0.8645357
## 4           Regularized Movie + User Effect Model 0.8639383
## 5 Regularized Movie + User Effect Model Validated 0.8649857
```

Chapter 7

Conclusion

The Regularized Movie + User Effect Model is the most accurate among all. There is a particular case that when this model is validated, the RMSE is bigger than when it is used in the test_set. Anyways, the results are satisfactory and the model has achieved the main goal. The limitation in this particular case is that the genre and timestamp were not deepened as I wished because my notebook does not have the necessary features to keep up with big codes and it crashes because of the saturation in the RAM. So, I encourage everybody, for future work to analyze those data and see how could they affect in the RMSE.