Movie Recommendation System

Pablo Dinamarca

23/6/2020

Contents

Chapter 1

Introduction

When you are watching TV or listening to music on Youtube, you never asked yourself how these websites know or at least come close to what you like to see / hear?

Data science is playing an increasingly important role in almost all areas. From economics to medicine, in companies, in the government or even for the agricultural sector, data science is a fundamental tool to project, analyze and even predict all kinds of different data such as consumer behavior or the spread of a disease.

Large online streaming service companies like Netflix, YouTube and Amazon have invested large sums of money in this science that they used to create recommendation algorithms for users.

The recommendation systems collect the data of the users, from any type of interaction they have with a web page or app until their ratings and comments. Then they “clean” this information to be able to work with it, they graph it, project it and analyze it.

In the end, they use these analyzes to project statistical models based on the probability that a user chooses one product or another and use them to improve their products, provide other products of interest, or anticipate the needs of their clients.

Companies follow the idea of offering *the right product* in *the right place*, that is achieved by implementing recommendation systems.

In fact, in 2006 Netflix offered a million dollar prize for the person or group that could improve their recommendation system by at least 10%. We can see this at https://www.netflixprize.com/.

In general, scoring systems are based on a rating scale of 1 to 5 degrees or stars, where 1 indicates the lowest satisfaction and 5 is the highest satisfaction.

In this case, our main objective is to predict the ratings of users for different movies as best as possible.

To do this, we created a film recommendation system from the MovieLens dataset and applied the knowledge obtained from the HarvardX Data Science Professional Certificate Program.

1.1 MovieLens Dataset

GroupLens is a research laboratory at the University of Minnesota that specializes (among other things) in recommendation systems and has provided qualification data for different movies at https://movielens.org/.

The entire MovieLens dataset consists of 27 million ratings from 58,000 movies by 280,000 users. The research presented in this document is based on a subset MovieLens 10M Dataset with 10 million ratings on 10,000 movies by 72,000 users.

1.2 Model Evaluation

The evaluation of machine learning algorithms consists of comparing the predicted value with the actual result. The loss function measures the difference between the two values.

There are several ways to evaluate an algorithm, one of the best known is the root of the root mean square error (RMSE) that will be exposed and explained later.

The central idea is that our loss function (RMSE) is as low as close to zero as possible. The objective of this project is to create a recommendation system with an RMSE < 0.8649.

1.3 Process and Workflow

Our work process will consist of the following steps:

1. Data preparation: Download, analyze, modify and prepare the data to be processed and analyzed.
2. Data exploration: Explore the data by creating graphs, tables, and stradistic summaries to understand the characteristics, relationship, and predictors it contains.
3. Data cleaning: In this section, unnecessary data is removed and data sets are ready to start modeling.
4. Data analysis and modeling: Models are created from the modified data sets that are evaluated with the RMSE and the final results are presented.
5. Final report: The report, its implications and possible future uses of it are exposed.

Chapter 2

Data Preparation

In this section, we download and divide the dataset for use in the analysis.

2.1 Load the Data

We first downloaded the dataset from the MovieLens website and divided it into two subsets used for training and validation. The training subset is called “edx” and the validation subset is called “validation” with 90% and 10% of the original dataset, respectively.

```r
#-------------------------------------#
# Create edx set and validation set #
#-------------------------------------#

# Note: this process could take a couple of minutes

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

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

2.2 Create the test set and train set

Then we divide the “edx” set into two subsets used for algorithm training and testing:

1. The “train” set with 90% of the “edx” data.

2. The “test” set with 10% of the “edx” data.

The model is created and trained in the “train” set and tested in the “test” set until the `RMSE` goal is reached.

```r
library(tidyverse)
library(lubridate)
library(plotly)
library(caret)
set.seed(1, sample.kind = "Rounding")

index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
test_set <- edx[index,]
train_set <- edx[-index,]
rm(index)

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

# Verify the data is correct
head(train_set)
```

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thrille
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|S
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
8	1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance

2.3 Modify the data

If we look at the dataset, we can notice that it contains a column called “timestamp” from which we can extract the years to later perform the exploration with timelines.

```r
# Modify the year as a column
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

```r
train_set <- train_set %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
test_set <- test_set %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
```

When the model reaches the `RMSE` target in the test set, we train the complete set “edx” with the model we created and use the “validation” set for the final test.

This process is called `Model Validation` and is mainly used to estimate how accurately a predictive model will perform in practice. For this we assume that the **validation set** is completely new data with completely unknown results.

Chapter 3

Data Exploration

Before starting to build our model, we must study how our data is composed from different points of view and with different variables.

3.1 Pre-visualize the Data

```
names(edx)
```

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

```
head(edx, 5)
```

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|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-F

We begin to explore our data and how they are organized.

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  7 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 838983
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thri
##  $ year     : num  1992 1995 1995 1994 1994 ...
```

We look at basic statistics of each variable in the dataset.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres               year
##  Length:9000055     Length:9000055     Min.   :1915
##  Class :character   Class :character   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :1994
##                                        Mean   :1990
##                                        3rd Qu.:1998
##                                        Max.   :2008
```

This table is useful as it displays essential statistical information such as rating and year.

```
dim(edx)
```

```
## [1] 9000055       7
```

```
class(edx)
```

```
## [1] "data.frame"
```

```
edx %>% summarize(Users = n_distinct(userId), Movies = n_distinct(movieId))
```

Users	Movies
69878	10677

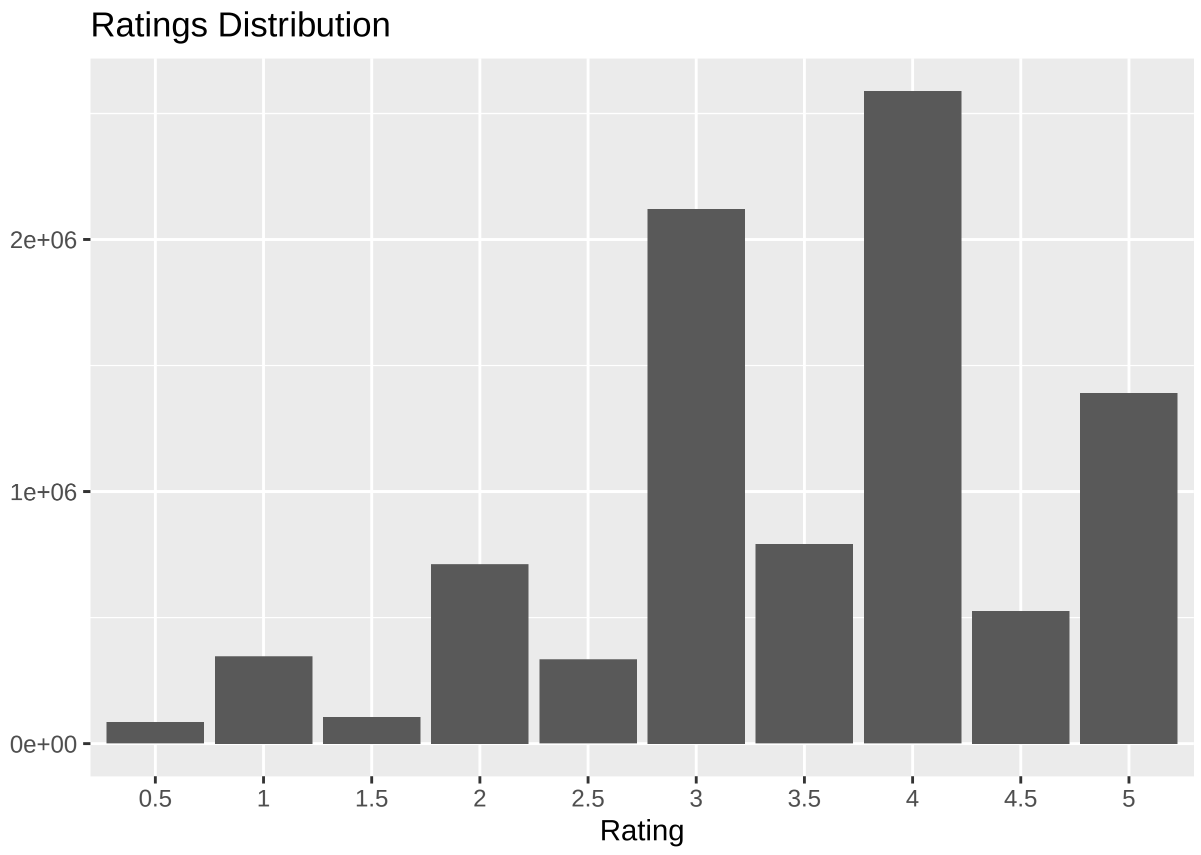
We note the number of Users and movies contained in the “edx” dataset.

3.2 Exploration by each feature

3.2.1 Rating

The following plot shows that users tend to rate movies between 3 and 4. This may be due to different factors and trying to generalize this observation may be wrong so we are going to analyze other variables.

```
# Note:This process could take a couple of minutes.
qplot(as.factor(as.vector(edx$rating))) +
  ggtitle("Ratings Distribution") + xlab("Rating")
```



3.2.2 Date

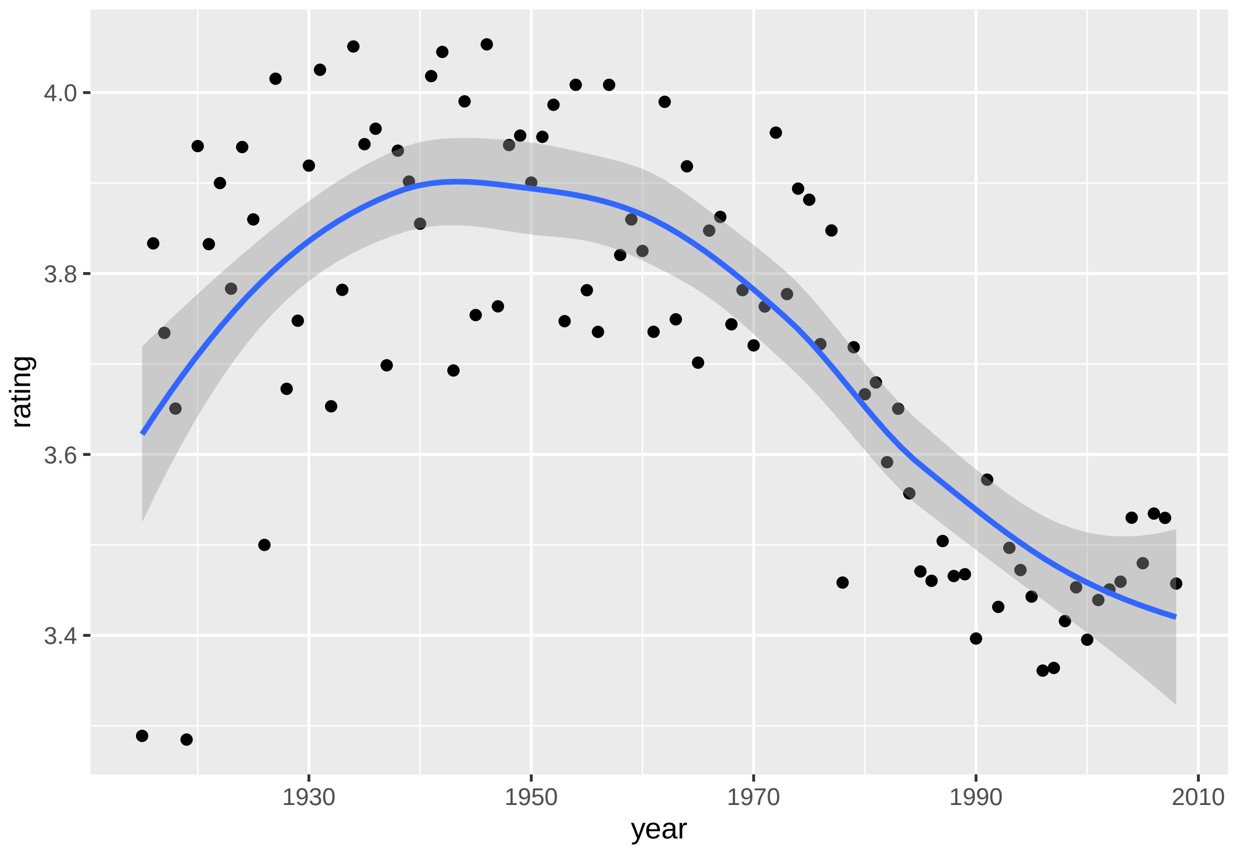
The following table lists the highest rated days for each movie. We note that movies that are box office hits have higher ratings as they are better known.

```r
# Show the highest date count of ratings for each movie
edx %>% mutate(date = date(as_datetime(timestamp))) %>%
  group_by(date, title) %>%
  summarise(Count = n()) %>%
  arrange(-Count) %>%
  head(5)
```

date	title	Count
1998-05-22	Chasing Amy (1997)	322
2000-11-20	American Beauty (1999)	277
1999-12-11	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	254
1999-12-11	Star Wars: Episode V - The Empire Strikes Back (1980)	251
1999-12-11	Star Wars: Episode VI - Return of the Jedi (1983)	241

Below, we can explore a general trend of movie viewers and their rating habits.

```r
edx %>% group_by(year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year, rating)) +
  geom_point() +
  geom_smooth()
```

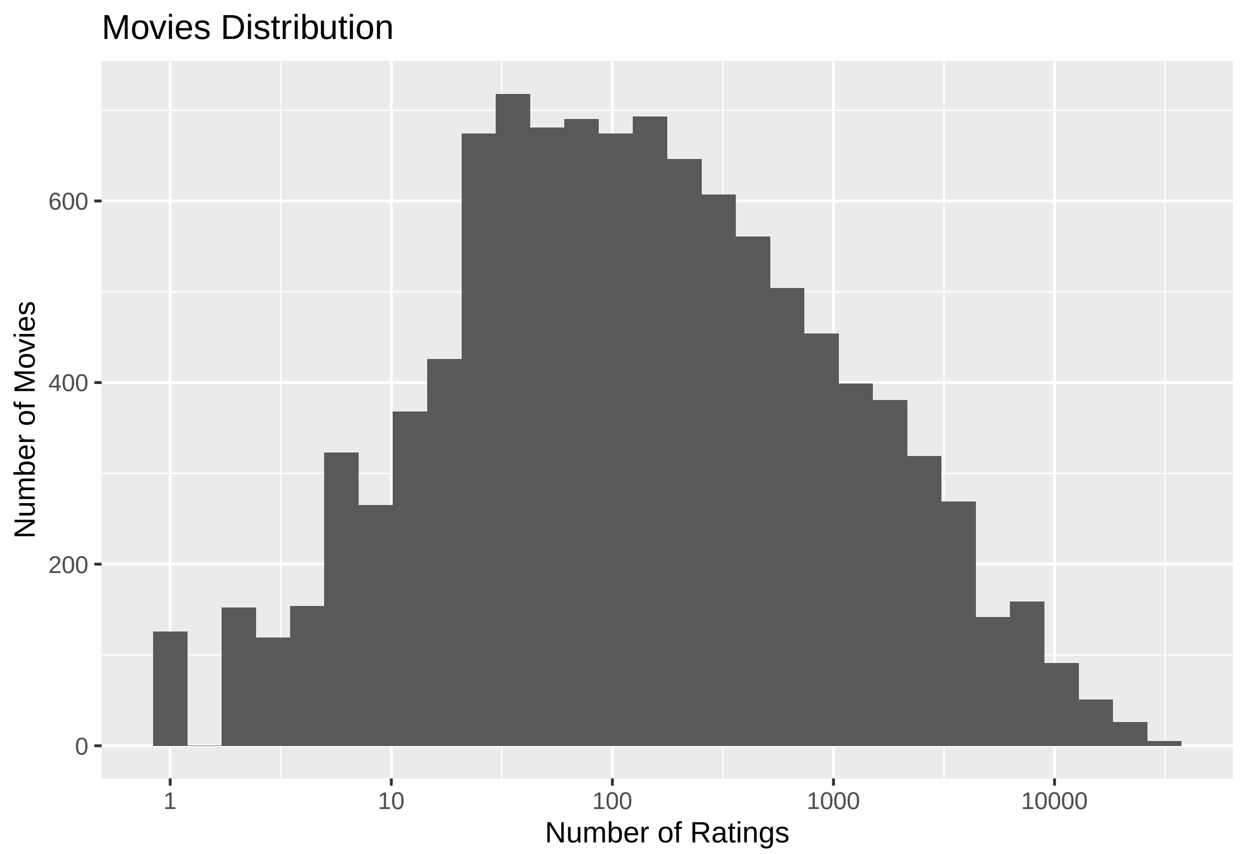


We notice that people have lowered their grade point average over the years and have become more critical when it comes to watching movies.

3.2.3 Movie

We know from date's exploration that some of the movies are rated higher than others, as many movies are watched by few users and blockbusters tend to have higher ratings.

```r
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  ggtitle("Movies Distribution") +
  xlab("Number of Ratings") +
  ylab("Number of Movies")
```

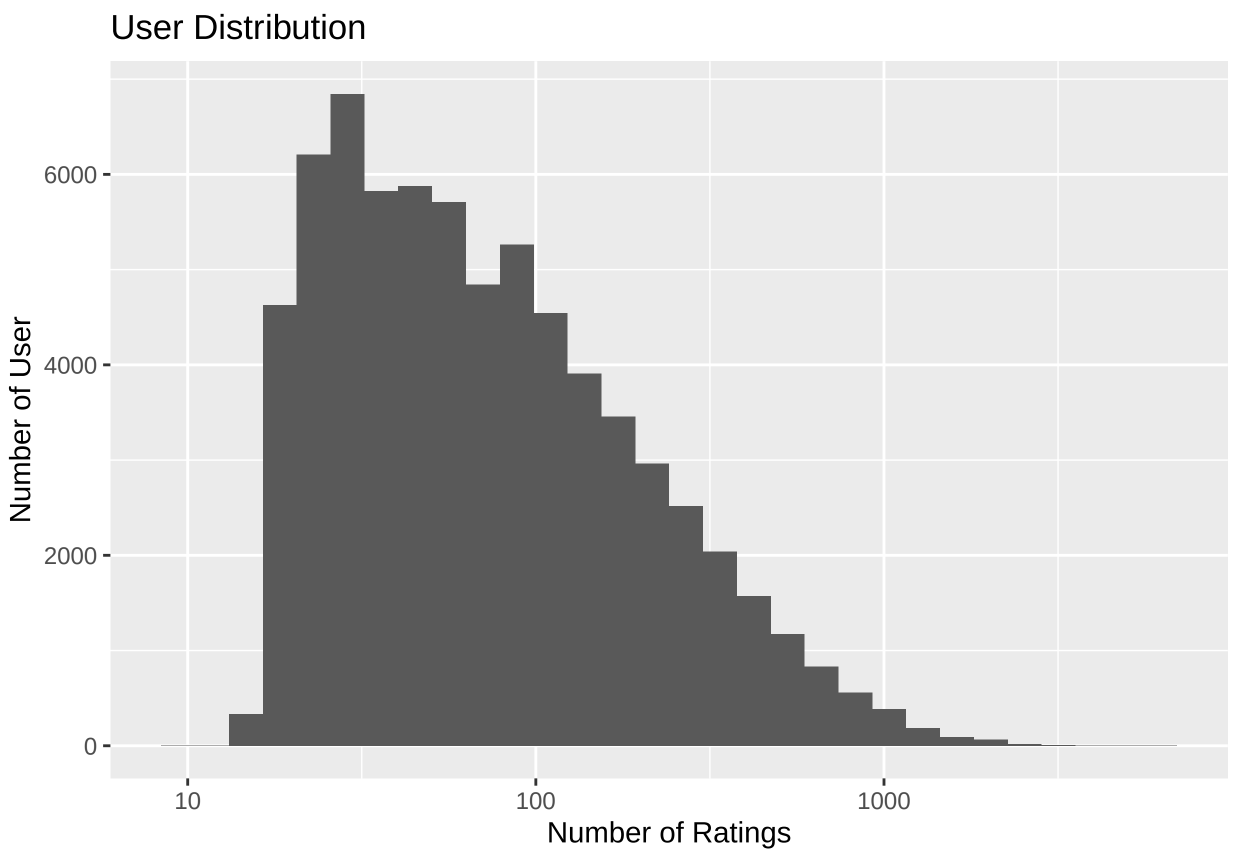


The histogram shows that some movies have been rated very rarely. Therefore, they should be given less importance in movie prediction.

3.2.4 User

Most of users rate few movies, while a few users rate more than a thousand movies. We can see this in the next graph.

```r
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30) +
  scale_x_log10() +
  ggtitle("User Distribution") +
  xlab("Number of Ratings") +
  ylab("Number of User")
```



The graph above shows that not all users are equally active. Some users have rated very few movies.

3.2.5 Genres

The Movielens dataset contains different combinations of genres. Here is the list of the total movie ratings by genre.

```
edx %>%
  group_by(genres) %>%
  summarize(N = n()) %>%
  arrange(-(N)) %>% head(5)
```

genres	N
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425

Chapter 4

Data Cleaning

Now it is time to remove extra variables that will not be useful in our modeling.

4.1 Remove extra Column

```
edx <- edx[,-4]
validation <- validation[,-4]
train_set <- train_set[,-4]
test_set <- test_set[,-4]
```

Chapter 5

Modeling to Use

We can describe the statistical model to be used in the project.

5.1 Linear Model

The first model we will run will be the simplest model that predicts that all users will give **the same rating** to all movies. To find the value that minimizes the `RMSE` we must choose the average score of the ratings given by users and use it to predict all the ratings, as shown in this formula:

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

Where $\hat{Y}$ is the prediction of unknown ratings, μ is the mean of the ratings and $\epsilon_{u,i}$ is the random-error. Anything other than the mean value will increase the `RMSE`.

Part of the variability in the movie data can be explained by the fact that **different movies have different ratings**. This is because some movies are more popular than others and user preference varies. This is called the movie effect or movie bias, and is expressed as b_i.

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

This bias can be calculated as the mean of the difference between the observed rating y and the mean μ. If the average was 3 and a movie was rated 5 stars, then the bias of the movie would be 2 stars.

$$\hat{b}_i = \frac{1}{N}\sum_{i=1}^{n_i}(y_i - \hat{\mu})$$

We can see from data exploration that **different users have different rating patterns**. For example: Some users like most movies and rate around 4 or 5, while other users are more critical and rate around 1 or 2.This is called user effect or user bias and is shown in the following formula:

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

This bias can be calculated in a very similar way to the previous one.

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

So you can also add year biases, they can be grouped into release periods where **release years play a key role**, the earlier the movie comes out the more users could rate it.

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_y + \epsilon_{u,i}$$

And it is calculated in a similar way:

$$\hat{b}_y = \frac{1}{N} \sum_{i=1}^{n_y} (y_{u,i} - \hat{b}_u - \hat{b}_i - \hat{\mu})$$

Finally, **the movies also depend on the genre that each user likes**. For example: A user who likes romantic comedies may not like mafia movies or horror movies and vice versa. This is called gender bias.

The model looks like this:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_y + b_g + \epsilon_{u,i}$$

And it is calculated in a similar way:

$$\hat{b}_g = \frac{1}{N} \sum_{i=1}^{n_g} (y_{u,i} - \hat{b}_y - \hat{b}_u - \hat{b}_i - \hat{\mu})$$

5.2 Regularization

The linear model provides a good estimate of the ratings, however it does not consider that:

1. Many little-known movies have very few ratings.
2. Some users rate very few movies.
3. Some users prefer little-known genres such as artistic or cultural films from each country.
4. The type of movies we like to watch may vary, so does the average rating over the years.

For example: If before we liked to watch cowboy movies, now it doesn't have the same impact as before.

Statistically, this could lead to a large estimated error.

The estimated value can be improved by adding a factor that penalizes small sample sizes that otherwise have little or no impact. Therefore, the estimated effects can be calculated using the following formulas:

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

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

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

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

We add λ `lambda` as a parameter to fit the model with very small values and not affect large values.

An effective method of choosing λ that minimizes the `RMSE` is to run simulations with multiple values of λ.

For practical purposes, we fit the user and movie effect regularized model and then add the year and regularized gender biases.

5.3 Evaluation Results

5.3.1 Root Mean Squared Error - RMSE

The Root Mean Squared Error, `RMSE`, is the square root of the MSE. It is the typical metric to evaluate recommendation systems, and is defined by the formula:

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

Basically subtract the predicted values from the real ones and add all the results to find the average of them, but not before squaring them to avoid negative values and biases, this increases the `RMSE`. Therefore it uses the square root to adjust the weight of the `RMSE` to its real values.

`RMSE` penalizes large deviations from the mean and is appropriate in cases where small errors are not relevant.

Here we define the loss functions:

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

Chapter 6

Data analysis and Modeling

6.1 Model by Average

The first model is just the mean of the ratings, μ.

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

```r
# Average of Ratings
mu_hat <- mean(train_set$rating)

# Create the error dataframe
Model_Results <- data_frame(Method = "By Average",
                            RMSE = RMSE(test_set$rating, mu_hat))
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
# Show the RMSE by average method
data.frame(Model_Results)
```

Method	RMSE
By Average	1.060054

6.2 Model by Movie Effect

We include the effect of the movie represented by b_i.

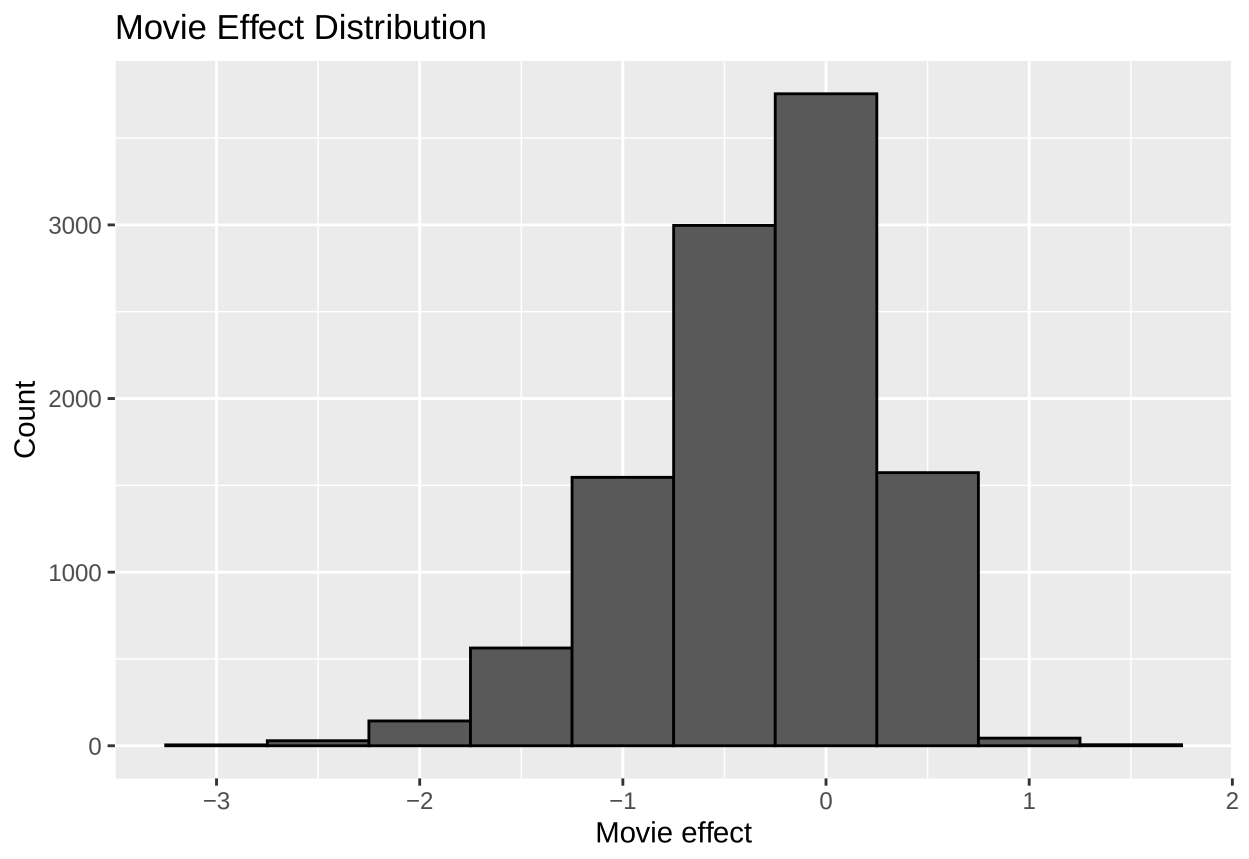
$$\hat{Y} = \mu + b_i + \epsilon_{u,i}$$

```r
#The Bias of movie is represent by b_i
Movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
head(Movie_effect, 5)
```

movieId	b_i
1	0.4150029
2	-0.3064068
3	-0.3613963
4	-0.6372820
5	-0.4416069

We explore the movie bias.

```r
#Plot of Movie Effect
Movie_effect %>% ggplot(aes(b_i)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
  ylab("Count")
```



We can observe the ranges of the values and their distribution of b_i.

```r
#Predict the model by this effect
Predicted_b <- mu_hat + test_set %>%
  left_join(Movie_effect, by="movieId") %>%
  .$b_i

#Aggregate to Models Dataframe
Method_bi <- RMSE(test_set$rating, Predicted_b)
Model_Results <- bind_rows(Model_Results,
                           data_frame(Method="By Movie Effect",
                                      RMSE = Method_bi ))



data.frame(Model_Results)
```

Method	RMSE
By Average	1.0600537
By Movie Effect	0.9429615

RMSE decrease shown.

6.3 Model by Movie & User Effect

We include the effect of the user represented by b_u.

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

```r
#The Bias of user is represent by b_u
User_effect <- test_set %>%
  left_join(Movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

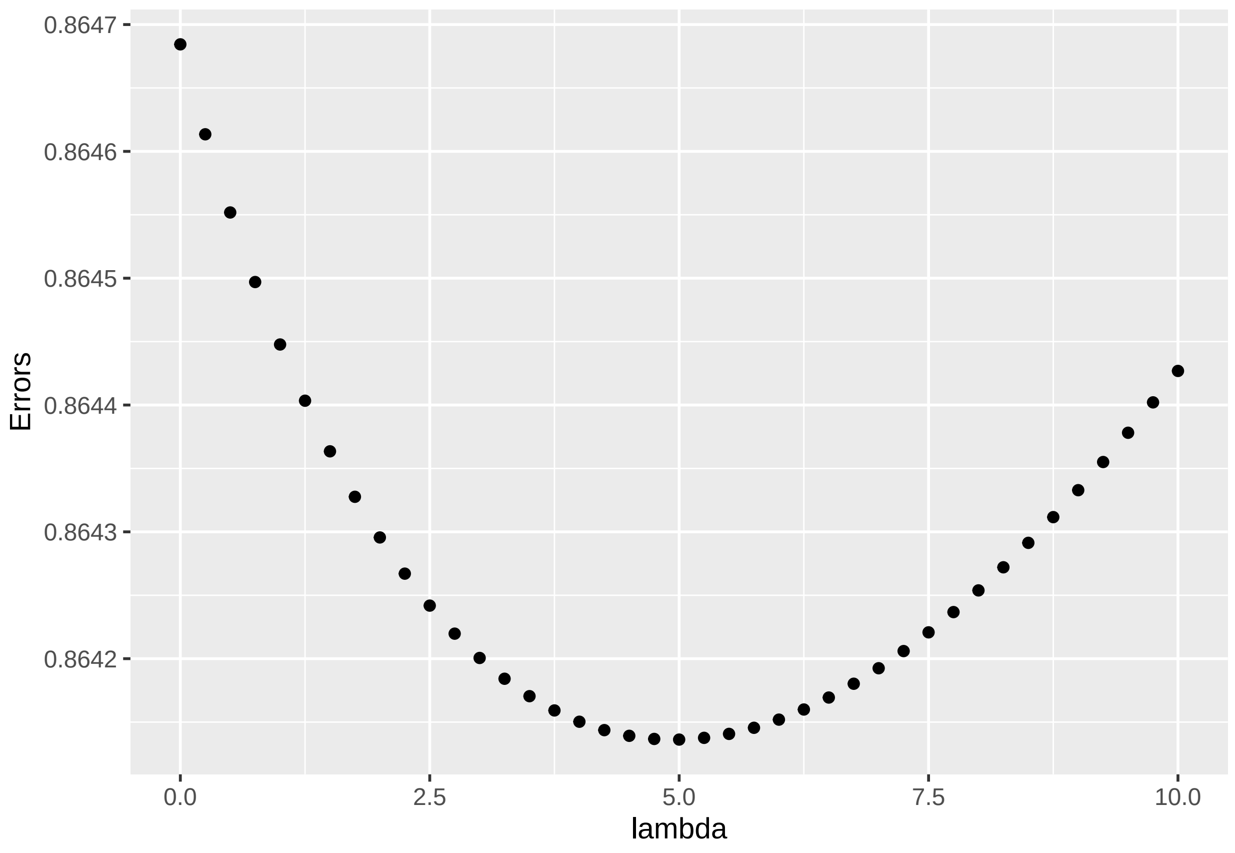
```r
#Predict the model by Movie and User effect
Predicted_b <- test_set %>%
  left_join(Movie_effect, by="movieId") %>%
  left_join(User_effect, by="userId") %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred
Method_MUE <- RMSE(test_set$rating, Predicted_b)


#Aggregate to Models Dataframe
Model_Results <- bind_rows(Model_Results,
                           data_frame(Method="By Movie & User Effect",
                                      RMSE = Method_MUE ))
Model_Results
```

Method	RMSE
By Average	1.0600537
By Movie Effect	0.9429615
By Movie & User Effect	0.8247682

6.4 Model by Regularized Movie + User Effect

For this model, we regularize the bias by adding a penalty factor λ, which is an adjustment parameter.We define a series of values for λ and use `regularization` to choose the best value that minimizes `RMSE`.

```r
#Select lambda by cross-validation
lambda <- seq(0, 10, 0.25)
set.seed(1, sample.kind = "Rounding")

# Note: this process could take a couple of minutes
Errors <- sapply(lambda, 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_b <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(test_set$rating, Predicted_b))
})

# Select the optimal lambda
qplot(lambda, Errors)
```



```r
lambda_1 <- lambda[which.min(Errors)]

#Save the results
Model_Results <- bind_rows(Model_Results,
                           data_frame(Method=
                                        "By Reg. Movie+User Effect",
                                      RMSE = min(Errors)))

data.frame(Model_Results)
```

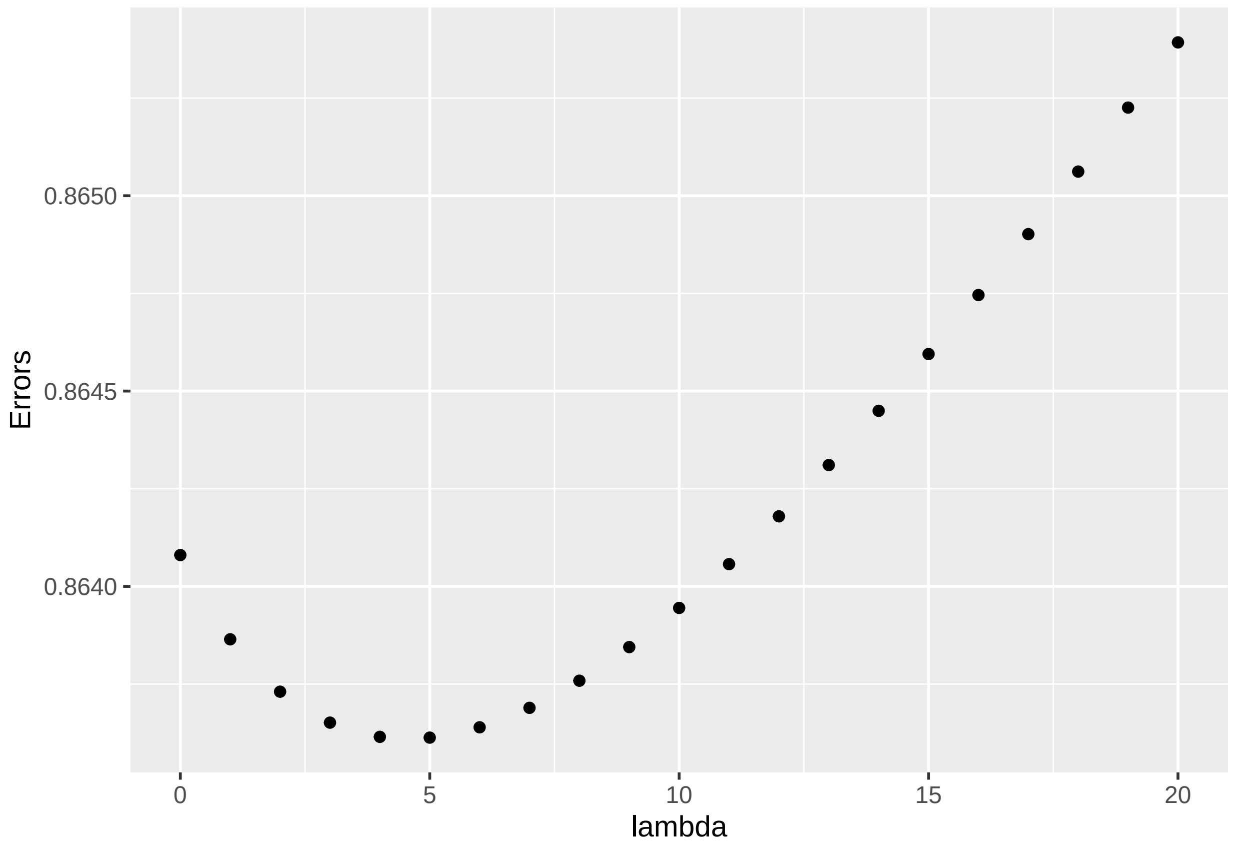
Method	RMSE
By Average	1.0600537
By Movie Effect	0.9429615
By Movie & User Effect	0.8247682
By Reg. Movie+User Effect	0.8641362

6.5 Model Aggregate Regularized Year and Genres effect

For the final model, we add the regularized Year and Genre effect with a penalty factor λ.

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_y + b_g + \epsilon_{u,i}$$

```r
# Select lambda by cross-validation
lambda <- seq(0, 20, 1)
set.seed(1, sample.kind = "Rounding")

# Note: this process could take a couple of minutes
Errors <- sapply(lambda, 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))

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

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

  Predicted_b <- test_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_y, by = "year") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    .$pred
  return(RMSE(test_set$rating, Predicted_b))
})

# Select the optimal lambda
qplot(lambda, Errors)
```



```
lambda_2 <- lambda[which.min(Errors)]

# Save the result
Model_Results <- bind_rows(Model_Results,
                           data_frame(Method=
                                        "Agg. Reg. Year+Genres Effect",
                                      RMSE = min(Errors)))

data.frame(Model_Results)
```

Method	RMSE
By Average	1.0600537
By Movie Effect	0.9429615

Method	RMSE
By Movie & User Effect	0.8247682
By Reg. Movie+User Effect	0.8641362
Agg. Reg. Year+Genres Effect	0.8636129

We can see that the error with the train and test data decreased significantly and achieved our goal of obtaining an RMSE < 0.8649 so we are now ready to implement the algorithm with the **Validation data**.

6.6 Remove variables

Remove all unnecessary variables to assess the final model with Validation set.

```r
rm(Model_Results, Movie_effect, Rating_plot, User_effect, Method_MUE,
   Method_bi, Method_bu, Predicted_MUE, Predicted_ME, Predicted_UE,
   train_set, test_set, Errors, lambda, mu_hat,Predicted_b)
```

Chapter 7

Final Validation

On this occasion we simply apply three models:

1. The average to have a reference.
2. Those regularized by User and Film.
3. The regularized ones including by Year and Gender.

This is due to ease in computing and with previous knowledge obtained from previous models.

7.1 Model by Average

```
set.seed(1, sample.kind = "Rounding")

Model_Results <- data_frame(Method = "By Average",
                            RMSE = RMSE(validation$rating, mean(edx$rating)))
data.frame(Model_Results)
```

Method	RMSE
By Average	1.061202

When observing the error we notice that it is very similar to the one obtained previously with the train and test data, this gives us an idea of the similarity with the validation data.

7.2 Model by Regularized Movie + User Effect

```
# Select lambda_1 of test_set
set.seed(1, sample.kind = "Rounding")

# Note:This process could take a couple of minutes
Errors <- sapply(lambda_1, function(l){
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  Predicted_B <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

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

#Save the results
Model_Results <- bind_rows(Model_Results,
                           data_frame(Method=
                                        "Reg. Movie+User Effect",
                                      RMSE = min(Errors)))
data.frame(Model_Results)
```

Method	RMSE
By Average	1.0612018
Reg. Movie+User Effect	0.8648177

7.3 Model Aggregate Regularized Year and Genres effect

```r
# Select lambda_2 of test_set
set.seed(1, sample.kind = "Rounding")

# Note: this process could take a couple of minutes
Errors <- sapply(lambda_2, function(l){

  mu <- mean(edx$rating)

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

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

  b_y <- edx %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))

  b_g <- edx %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_y, by = 'year') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+l))

  Predicted_B <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_y, by = 'year') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    .$pred

  return(RMSE(validation$rating, Predicted_B))
})
```

```r
# Save the result
Model_Results <- bind_rows(Model_Results,
                           data_frame(Method=
                           "Agg. Reg. Year+Genres Effect",
                                    RMSE = min(Errors)))
data.frame(Model_Results)
```

Method	RMSE
By Average	1.0612018
Reg. Movie+User Effect	0.8648177
Agg. Reg. Year+Genres Effect	0.8642529

We can see that the error is 0.8642529 < 0.8649. So we managed to meet the objective of this project and created an effective predictive model for movie recommendation systems.

Chapter 8

Final Report

8.1 Conclusion

Creating a recommendation system project is one of the most important applications of data science today.

In this document we illustrate a movie recommendation system based on user ratings in which we exposed and explained the different points of view from which algorithms can be created.

We only use four approaches in this document, but we can also add more and there are more effective techniques such as matrix factorization, random forests and other statistical theories.

Regularization and model validation concepts were applied throughout it and we used the linear model of $\hat{Y}_{u,i} = \mu + b_i + b_u + b_y + b_g + \epsilon_{u,i}$ regularized to predict the different qualifications.

This model achieved an RMSE of 0.8642529, successfully exceeding the proposed target of 0.8649 and successfully concluding this draft recommendation system.

8.2 Limitations

Some of the algorithms mentioned above are computationally expensive to run on a laptop and therefore could not be implemented. The amount of memory required far exceeded that available on the computer.

In this case, only four predictors were used, however there are many more that can be used to improve the model, such as playlists, comments, characteristics of the users, the actors that participate, etc.

The model works only for existing users, movies and current rating values, if new users or movies are added, or if the rating changes, the model will have to be updated again and for large amounts of data this could be a problem.

It is worth noting also that we left without an initial recommendation for new users or for users who did not rate the movies. This can be solved by adding more predictors to our model or by using other statistical techniques.

8.3 Future Work

Machine learning models can be widely modified limited by computing power. So we present some approaches to expand the work of this project.

There are many R libraries that can be downloaded and applied to generate algorithms, some of the most used are listed here R libraries for recommender systems.

For widely used approaches not discussed here like content-based filtering and collaborative filtering. The package recomenderlab is widely used to implement them.

To process data from this project or apply matrix factorization, you can use the recosystem package.

In addition, other packages for building recommendation systems are available from The Comprehensive R Archive Network (CRAN).