MovieLens Recommendation System

HarvardX Data Science Professional Certificate : PH125.9x Capstone

Avijit Sur

03 March, 2024

Introduction

Competing in the era of artificial intelligence requires companies to rethink both their business and operating models [@iansiti2020], placing greater emphasis on the application of data science principles to exploit the availability of big data and the power of machine learning to create value [@mohr2018].

Recommendation systems are among the most important applications of machine learning deployed by digital companies today [@schrage_2017]. Companies such as Amazon and Netflix use these systems to understand their customers better and to target them with products more effectively [@schrage_2018]. In 2009, Netflix awarded a $1M prize to the team of data scientists who had successfully met the challenge of improving their movie recommendation algorithm by 10% [@lohr_2009; @koren2009].

The MovieLens (https://grouplens.org/datasets/movielens/10m/) datasets have provided a popular environment for experimentation with machine learning since their launch in 1997 [@maxwell_2015].

The goal of this project was to develop a recommendation system using one of the MovieLens datasets which consists of 10 million movie ratings. To facilitate this work, the dataset was split into a training set (edx) and a final hold-out test set (validation) using code provided by the course organisers. The objective was for the final algorithm to predict ratings with a root mean square error (RMSE) of less than 0.86490 versus the actual ratings included in the validation set.

This report sets out the exploratory analysis of the data using common visualisation techniques followed by the methods used to develop, train and test the algorithm before providing and discussing the results from each iteration of the algorithm and concluding on the outcome of the final model, its limitations and potential for future work.

The report was compiled using R Markdown in RStudio (https://rstudio.com/products/rstudio/), an integrated development environment for programming in R, a language and software environment for statistical computing.

Section 1: Loading Data

The MovieLens dataset is downloaded from: https://grouplens.org/datasets/movielens/ (https://grouplens.org/datasets/movielens/)

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

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


options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)
```

Section 2: Initial Data Wrangling

The initial wrangling code is provided by the Data Science Course:

```r
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify =
TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TR
UE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

Section 3: Data Loading

The MovieLens dataset will be divided into two subsets:

```
1. “edx,” serving for training, developing, then selecting the best algorithm;
2. “final_holdout_test” that will be used for evaluating the RMSE of the selected alg
orithm.
```

```
# edx dataset: for training & developing best algorithm.
# final_holdout_test: Evaluate RMSE for the selected algorithm.

# Final hold-out test set will be 10% of MovieLens data
set.seed(1)

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FA
LSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Section 4: Exploratory Data Analysis

The objective of this EDA is to:

```
1. Identify potential errors, better understand patterns within the data, detect outl
iers or anomalous events, find interesting relations among the variables.

2. Help defining the most relevant strategy to develop our prediction models.
```

Section 4.1: Dataset Structure

The edx dataset is a datatable, dataframe consists of 9000061 rows and 6 columns.

```
# edx table structure
str(edx)
```

```
## 'data.frame':    9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838
984474 838983653 838984885 838983707 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Ou
tbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama
|Sci-Fi|Thriller" ...
```

There are 6 Variables in this dataset:

```
1. userId: Integer. Movielens users that were selected at random for inclusion. Their
ids have been anonymised.

2. movieId: Integer. MovieID is the real MovieLens id.

3. rating: Numeric. Rating of one movie by one user. Ratings are made on a 5-star sca
le, with half-star increments.

4. timestamp: Integer. When the rating was made, expressed in seconds since midnight
Coordinated Universal Time (UTC) of January 1, 1970.

5. title: Character. Movies' titles + year of its release.

6. genres: Character. Genres are a pipe-separated list, and are selected from the fol
lowing: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama,
Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Wester
n, "no genre listed".
```

Section 4.1: Summary Statistics

```
# Summary of edx
summary_edx <- summary(edx)
kable(summary_edx, booktabs = TRUE, caption = "Summary Statistics") %>%
  kable_styling(position = "center", latex_options = c("scale_down", "striped"))
```

Summary Statistics

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000061	Length:9000061
1st Qu.:18122	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character

Median :35743	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35869	Mean : 4120	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53602	3rd Qu.: 3624	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

The dataset is clean with no missing/NA values. We will now check the number of unique values for the “users”, “movies” and “genre” variables.

```
# Number of unique values for the “users”, “movies” and “genre”
edx %>%
  summarize('Nb Unique Users' = n_distinct(userId),
            'Nb Unique Movies' = n_distinct(movieId),
            'Nb Unique Genres' = n_distinct(genres)) %>%
  kable(caption = "Unique Values") %>%
  kable_styling(position = "center", latex_options = c("scale_down", "scale_down"))
```

Unique Values

Nb Unique Users	Nb Unique Movies	Nb Unique Genres
69878	10677	797

The data contains 9000061 ratings applied to 10,677 different movies of 797 different unique or combination of genre from 69,878 single users between 1995 and 2009.

To better perform our analysis and build our models , we will further wrangle the datasets as following:

```
1. Convert the “timestamp” column to “datetime” format;
2. Extract the “Release Year” observation from the “Title” column.
3. Use the new “Release Year” column to add a “MovieAge” column (using year 2023 to c
alculate the age).
```

Section 5: Advanced Data Wrangling

```
# Convert the "timestamp" column to "datetime" format
  edx <- edx %>%
    mutate(timestamp = as_datetime(timestamp),
           rating_date = make_date(year(timestamp), month(timestamp))) %>%
    select(-timestamp) %>%
    relocate(rating_date, .after = rating) %>%
    as.data.table()

final_holdout_test <- final_holdout_test %>%
  mutate(timestamp = as_datetime(timestamp), rating_date = make_date(year(timestamp),
month(timestamp))) %>%
  select(-timestamp) %>%
  relocate(rating_date, .after = rating) %>%
  as.data.table()

# Extract the "Release Year" observation from the "Title" column
rel_year <- "(?<=\\()\\d{4}(?=\\))"

edx <- edx %>%
  mutate(release_year = str_extract(title, rel_year) %>%
           as.integer()) %>%
  relocate(release_year, .after = title)

final_holdout_test <- final_holdout_test %>%
  mutate(release_year = str_extract(title, rel_year) %>%
           as.integer()) %>%
  relocate(release_year, .after = title)

# Use the new “Release Year” column to add a “MovieAge” column
edx <- edx %>%
  mutate(movie_age = 2023 - release_year) %>%
  relocate(movie_age, .after = release_year)

final_holdout_test <- final_holdout_test %>%
  mutate(movie_age = 2023 - release_year) %>%
  relocate(movie_age, .after = release_year)
```

In the next phase, we will look deeper into some potentially interesting patterns and/or biases that will help at defining our modelling strategy.

Section 6: Review Users

Section 6.1: Number of rating per user

From the “User” perspective, we will explore the number of ratings made by users.

```
# Calculate number of ratings per user
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarize(nb_ratings = n()) %>%
  ungroup()
```

Contrary to the information provided by GroupLens about the dataset, it is apparent that a number of users actually rated less than 20 movies.

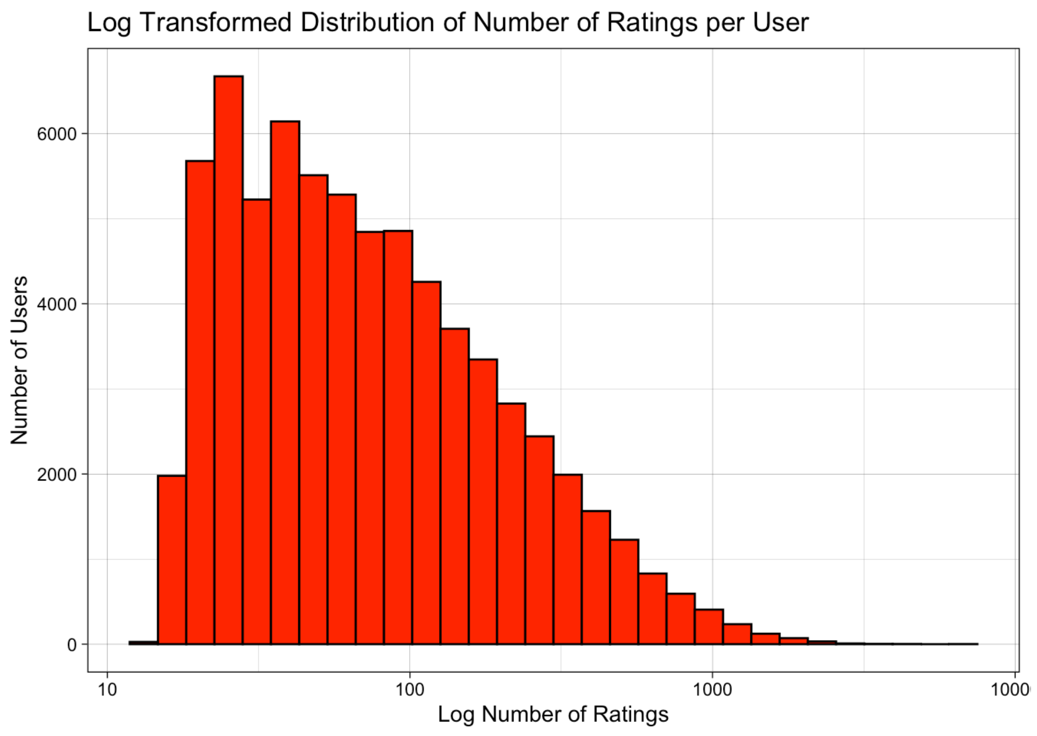
```
# Rating per user less than 20
length(which(ratings_per_user$nb_ratings <= 20))
```

```
## [1] 4986
```

It is confirmed that 4,986 users rated ⩽20 movies. This shall be taken in account as a potential bias affecting our prediction models.

We will now visualise the distribution of the number of ratings per user:

```
ggplot(ratings_per_user, aes(x = nb_ratings)) +
  geom_histogram(bins = 30, fill = "red", color = "black") +
  scale_x_log10() +
  labs(title = "Log Transformed Distribution of Number of Ratings per User",
       x = "Log Number of Ratings",
       y = "Number of Users") +
  theme_linedraw()
```



Right skewed log transformed distribution with median at 62 and mean at 129. It appears that the majority of users rated a number of movies that sits between 25 and 100 which seems somehow low: to be taken in account as a “bias” when developing our prediction models with potentially adding a “user penalty term”.

Section 7: Review Movies

Section 7.1: Number of movies vs number of rating

This section will focus on the relation between movies and their related number of ratings.

```
# Relationship between number of movies vs number of ratings
ratings_movie <- edx %>%
  group_by(movieId) %>%
  summarize(nb_ratings = n()) %>%
  ungroup()
summary(ratings_movie)
```

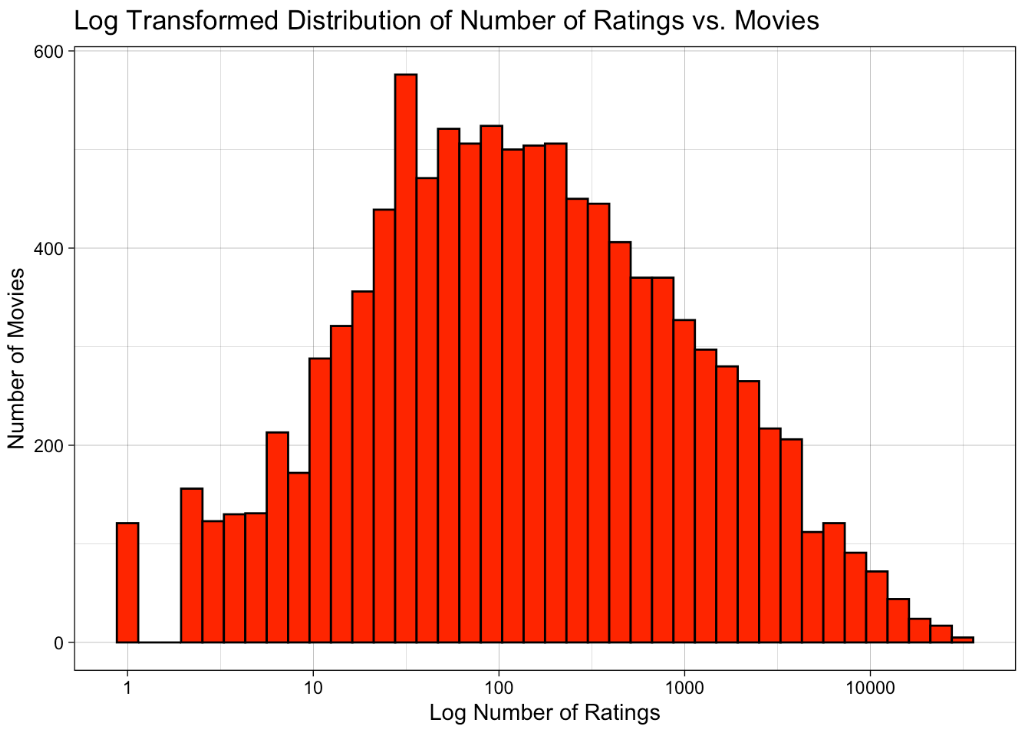
```
##     movieId        nb_ratings    
##  Min.   :    1   Min.   :    1.0  
##  1st Qu.: 2754   1st Qu.:   30.0  
##  Median : 5434   Median :  122.0  
##  Mean   :13105   Mean   :  842.9  
##  3rd Qu.: 8710   3rd Qu.:  563.0  
##  Max.   :65133   Max.   :31336.0
```

```
str(ratings_movie)
```

```
## tibble [10,677 × 2] (S3: tbl_df/tbl/data.frame)
##  $ movieId   : int [1:10677] 1 2 3 4 5 6 7 8 9 10 ...
##  $ nb_ratings: int [1:10677] 23826 10717 7053 1579 6415 12385 7273 811 2280 15250
...
```

Visualization of Movies vs. Ratings:

```
ggplot(ratings_movie, aes(x = nb_ratings)) +
  geom_histogram(bins = 40, fill = "red", color = "black") +
  scale_x_log10() +
  labs(title = "Log Transformed Distribution of Number of Ratings vs. Movies",
       x = "Log Number of Ratings",
       y = "Number of Movies") +
  theme_linedraw()
```



The distribution is not far from being symmetric which tends to show that popular movies are rated more frequently than less popular ones. The fact that there are a number of films with fewer ratings implies potential biases that may affect our recommendation modelling.

Section 8: Ratings Movies

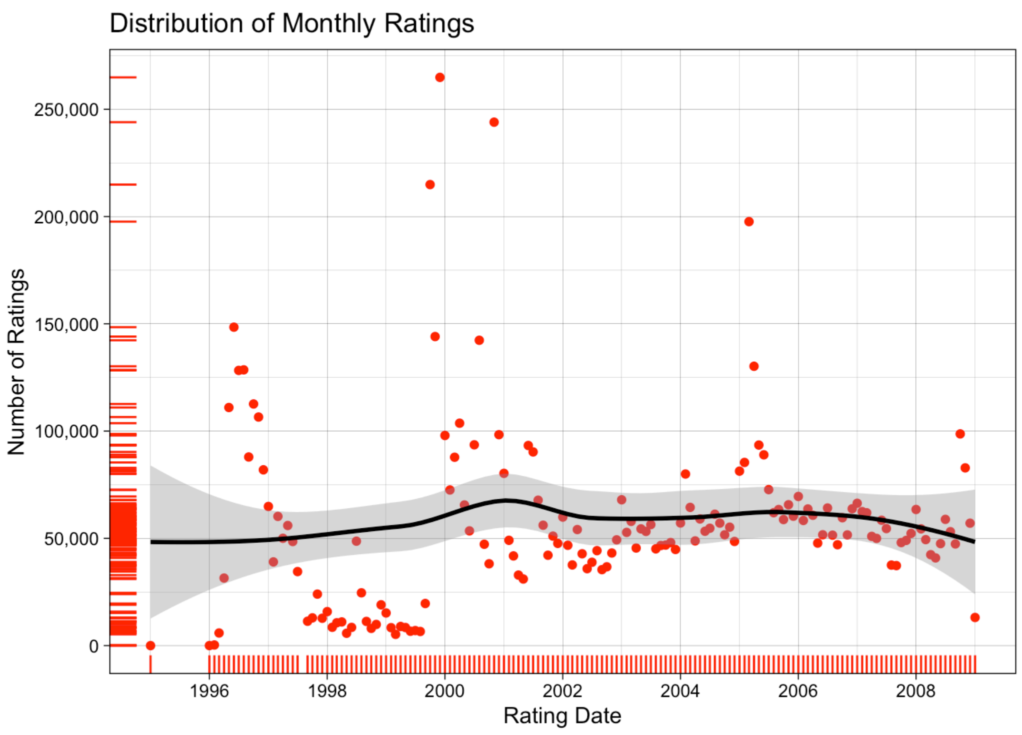
Section 8.1: Number of monthly ratings

Let's explore the "Rating" perspective and start with the number of ratings made on a monthly basis;

```
edx_ratings <- edx %>%
  group_by(rating_date) %>%
  summarize(nb_rating = n())
```

Let's visualize the distribution of ratings made on a monthly basis:

```
ggplot(edx_ratings, aes(x = rating_date, y = nb_rating)) +
  geom_point(color = "red") +
  scale_x_date(date_breaks = "2 years", date_labels = "%Y") +
  scale_y_continuous(breaks = seq(0, 300000, 50000), labels = scales::comma) +
  labs(title = "Distribution of Monthly Ratings",
       x = "Rating Date",
       y = "Number of Ratings") +
  geom_rug(color = "red") +
  geom_smooth(color = "black") +
  theme_linedraw()
```

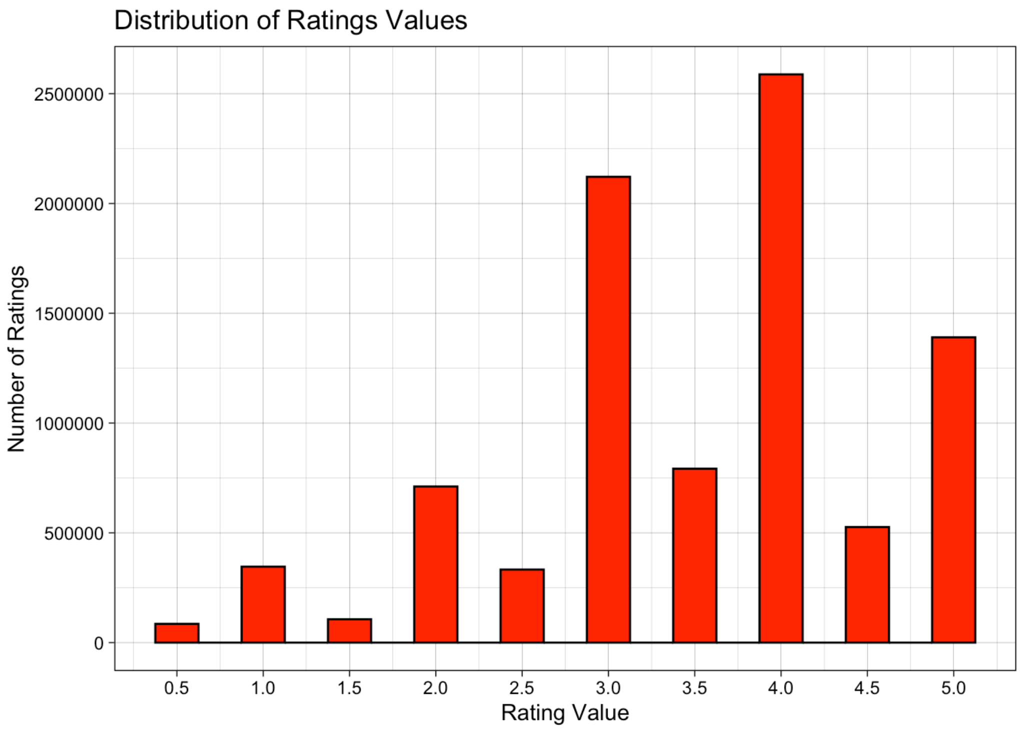


We notice an average of approx. 60k movie ratings made on a monthly basis since 1990. This number seems to have stabilized after 2000. We can notice a number of outliers, high and low before 2000, then high around 2000 (up to 264,856 ratings !) and 2005. To be taken in account when developing our prediction models.

Section 8.2: Rating Values

Let's have a look at the rating values and its distribution:

```
ggplot(edx, aes(rating)) +
  geom_histogram(binwidth = 0.25, fill = "red", color = "black") +
  scale_x_continuous(breaks = seq(0.5, 5, 0.5)) +
  scale_y_continuous(breaks = seq(0, 3000000, 500000)) +
  labs(title = "Distribution of Ratings Values",
       x = "Rating Value",
       y = "Number of Ratings") +
  theme_linedraw()
```

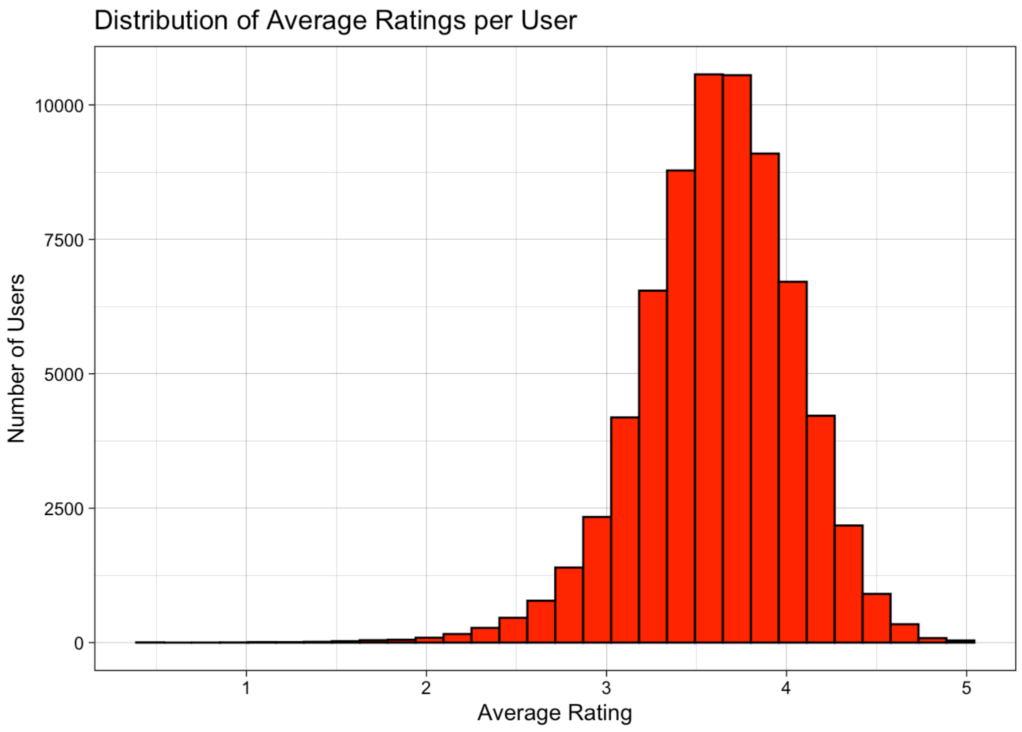


There are 10 different ratings users can award a movie: 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5 and 5. A rating of “4” is the most popular rating followed by “3”, “5”, “3.5”, etc. while “0.5” is awarded the least frequently.

Section 8.3: Average Rating per user

Finally, we will explore the distribution of mean rating made by users.

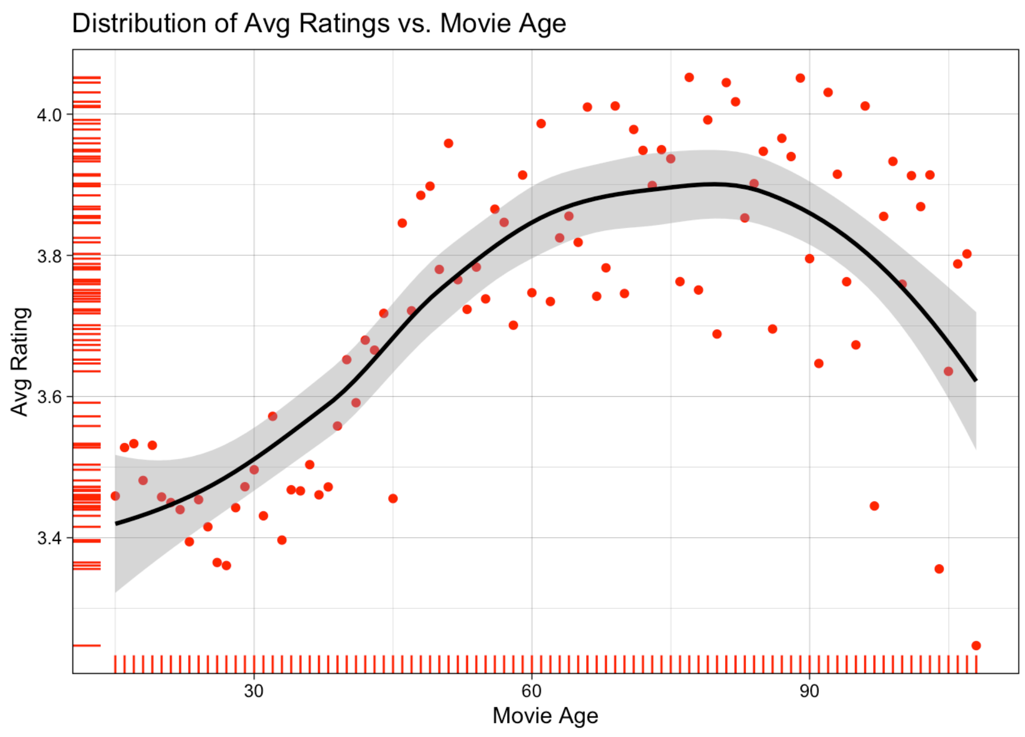
```
avg_ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarize(average_rating = mean(rating)) %>%
  ungroup()

ggplot(avg_ratings_per_user, aes(x = average_rating)) +
  geom_histogram(bins = 30, fill = "red", color = "black") +
  labs(title = "Distribution of Average Ratings per User",
       x = "Average Rating",
       y = "Number of Users") +
  theme_linedraw()
```



The distribution is symmetric and, as seen previously, centred around 3.5 which shows a tendency for users to favour rating movies they appreciated rather than the one they disliked.

Section 8.4: Mean Rating vs Movie Age

```
ratings_avg_age <- edx %>%
  group_by(movie_age) %>%
  summarize(average_rating = mean(rating)) %>%
  ungroup()

ggplot(ratings_avg_age, aes(x = movie_age, y = average_rating)) +
  geom_point(color = "red") +
  labs(title = "Distribution of Avg Ratings vs. Movie Age",
       x = "Movie Age",
       y = "Avg Rating") +
  geom_rug(color = "red") +
  geom_smooth(color = "black") +
  theme_linedraw()
```



The correlation between the two features seems quite clear with users tending to award higher ratings to older movies rather than to newer releases. While probably interesting, this effect / bias will be taken in account if we don't reach our RMSE targeted value through simpler models

Section 9: Modeling

We will focus on the the effects and/or biases in relation with the "users" and "movies" sections we went through.

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

Section 9.1: Benchmark Model

The benchmark model we will use is to naively predict all ratings as the average rating of the training set.

The formula we will use is defined as follow: Yu,i = predicted rating, μ = average rating and ϵu,i = independent errors centred at 0. Yu,i=μ+ϵu,i

```
edx_mu <- mean(edx$rating)
RMSE_1 <- RMSE(edx$rating, edx_mu)

result1_table <- tibble(Model = "Benchmark", RMSE = RMSE_1)
result1_table %>%
  knitr::kable()
```

Model	RMSE
Benchmark	1.060393

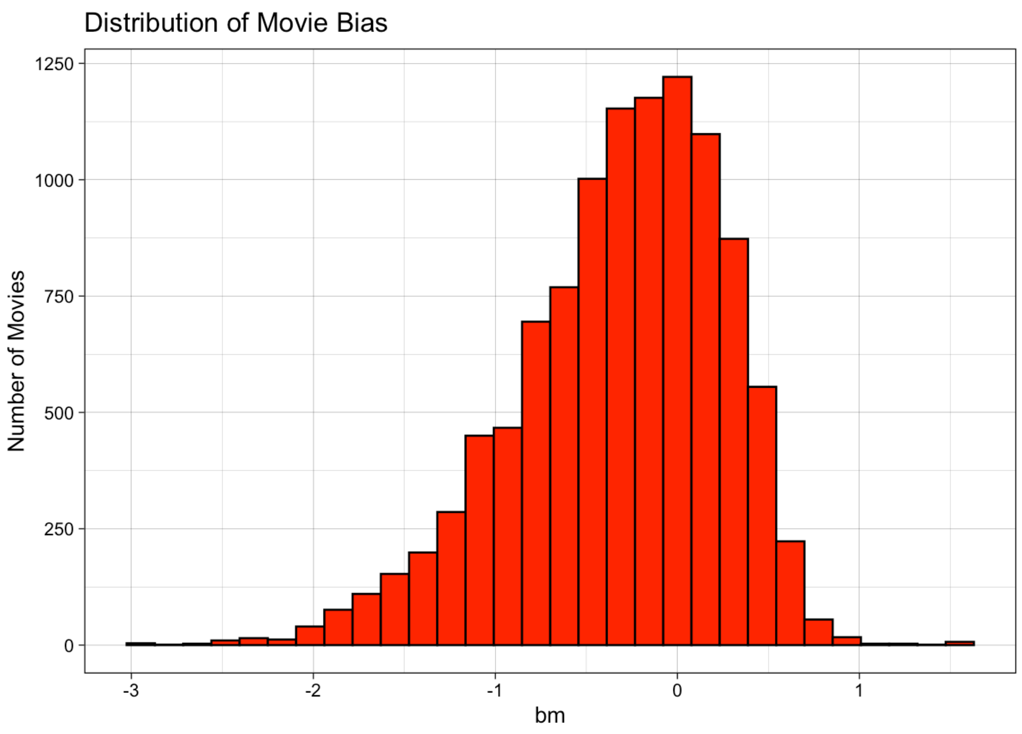
The RMSE for this model is 1.0603, which is unsurprisingly high.This will nevertheless be used as a benchmark for comparison with the different models we will now design.

Section 9.2: Movie-bias Model

To improve our model we will take in account the idea that some movies are subjectively rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movie's mean rating from the total mean of all movies μ. The resulting variable is called bm with "b" for bias and "m" for movie. The formula we will use is defined as: Yu,i=μ+bm+ϵu,i With Yu,i = predicted rating, μ = average rating, bm = "movie bias" variable and ϵu,i = independent errors centred at 0.

Let's first visualize the bm distribution:

```
movie_bias <- edx %>%
  group_by(movieId) %>%
  summarize(bm = mean(rating - edx_mu))
ggplot(movie_bias, aes(x = bm)) +
  geom_histogram(bins = 30, fill = "red", color = "black") +
  labs(title = "Distribution of Movie Bias",
       x = "bm",
       y = "Number of Movies") +
  theme_linedraw()
```



We will now check the prediction against the edx set to determine the related RMSE:

```
pred_bm <- edx_mu + edx %>%
  left_join(movie_bias, by = "movieId") %>%
  .$bm
RMSE_2 <- RMSE(edx$rating, pred_bm)
result2_table <- tibble(Model = "Movie Bias", RMSE = RMSE_2)
result2_table %>%
  knitr::kable()
```

Model	RMSE
Movie Bias	0.942368

The RMSE for this model is 0.9423, which is quite an improvement from our benchmark measure but still far from our initial goal.

We have predicted movie rating based on the fact that movies are rated differently by adding the computed bm to μ. If an individual movie is on average rated worse than the average rating of all movies μ , we predict that it will rated lower that μ by bm, the difference of the individual movie average from the total average.

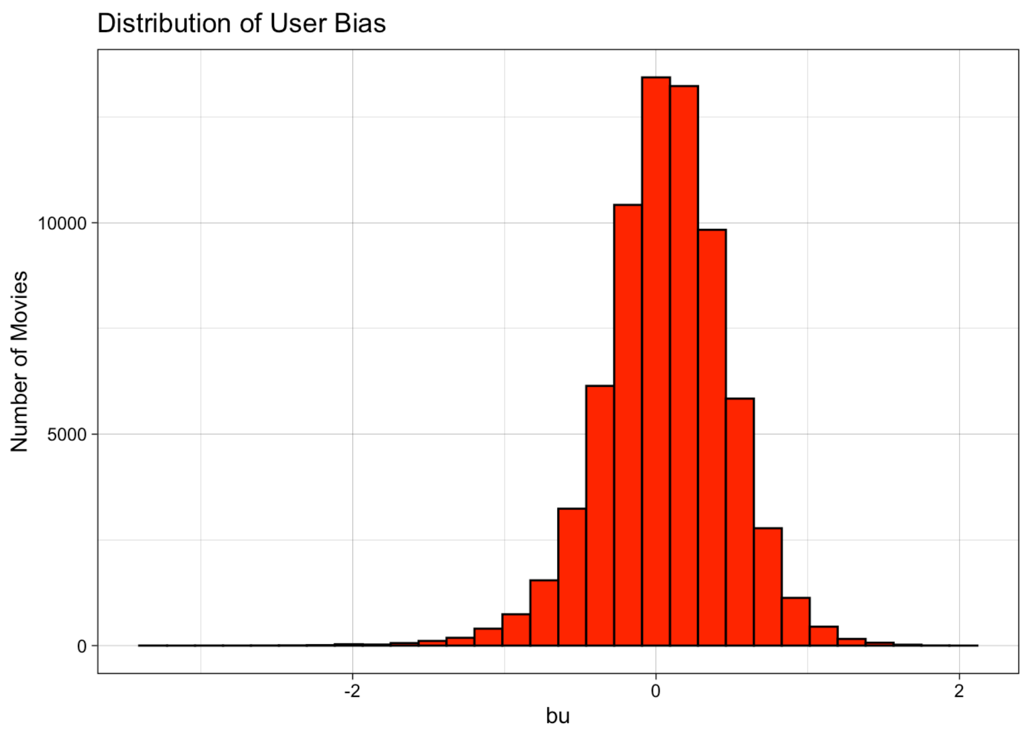
Section 9.3: Movie & User-bias Model

This model introduces “User Effect / Bias” that reflects the fact that individual users tend to rate films according to their own standards (which vary widely in distribution). The formula can be defined as follows with Yu,i = predicted rating, μ = the average rating, bm = the movie effects, bu = the user effects and ϵu,i = independent errors centred at 0.

Yu,i=μ+bm+bu+ϵu,i where bu is the “User Bias” variable.

Let’s first visualize the bu distribution:

```
user_bias <- edx %>%
  left_join(movie_bias, by = "movieId") %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - edx_mu - bm))
ggplot(user_bias, aes(x = bu)) +
  geom_histogram(bins = 30, fill = "red", color = "black") +
  labs(title = "Distribution of User Bias",
       x = "bu",
       y = "Number of Movies") +
  theme_linedraw()
```



We will now check the prediction of this model against the test set to determine the related RMSE:

```
pred_bu <- edx %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(pred = edx_mu + bm + bu) %>%
  .$pred
RMSE_3 <- RMSE(edx$rating, pred_bu)
result3_table <- tibble(Model = "Movie & User Biases", RMSE = RMSE_3)
result3_table %>%
  knitr::kable()
```

Model	RMSE
Movie & User Biases	0.8566699

The RMSE for this model is 0.8567. Quite an improvement from our latest model but this model can be improved further.

Section 9.4: Regularized Movie & User-bias Model

We will now introduce the concept of regularisation: the idea is to add a tuning parameter λ to further reduce the RMSE. The idea is to "penalise" outliers from the Movie Bias and User Bias sets which shall optimise the recommendation system.

Define λ:

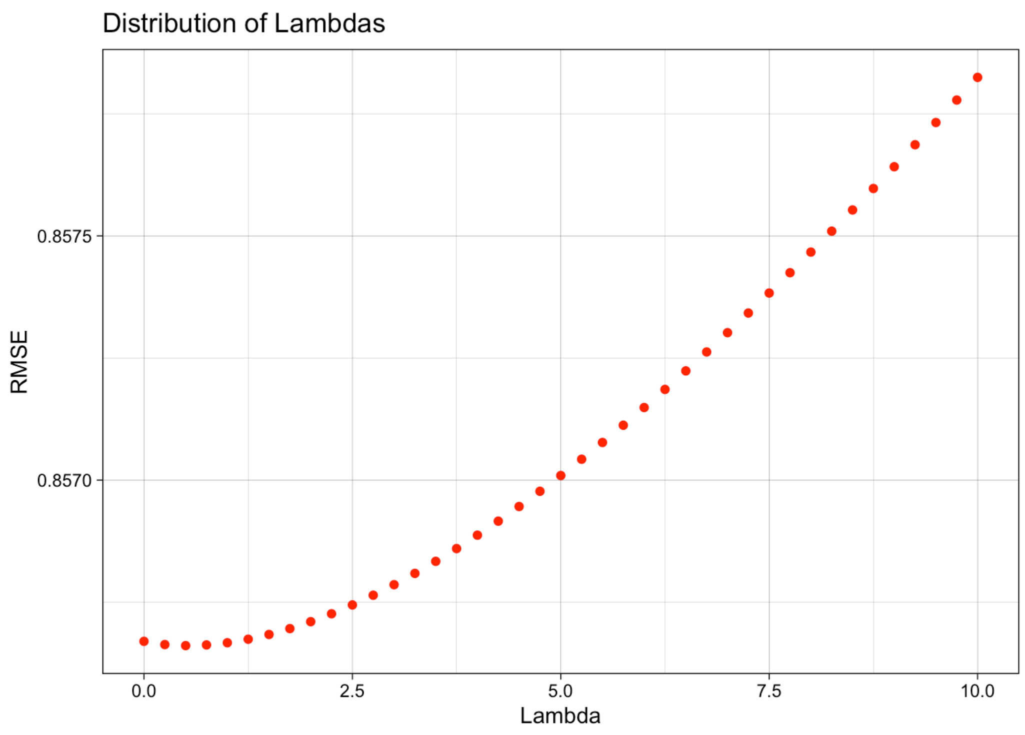
```
lambdasReg <- seq(0, 10, 0.25)

RMSEreg <- sapply(lambdasReg, function(l){
  edx_mu <- mean(edx$rating)
  bm <- edx %>%
    group_by(movieId) %>%
    summarize(bm = sum(rating - edx_mu)/(n() + l))
  bu <- edx %>%
    left_join(bm, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - bm - edx_mu)/(n() + l))
  predicted_ratings <- edx %>%
    left_join(bm, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = edx_mu + bm + bu) %>%
    .$pred
  return(RMSE(predicted_ratings, edx$rating))
})
```

Let's now determine which λ will be best at reducing the RMS.

```
lambda <- lambdasReg[which.min(RMSEreg)]
lambda
```

```
## [1] 0.5
```

```
ggplot(mapping = aes(x = lambdasReg, y = RMSEreg)) +
  geom_point(color = "red") +
  labs(title = "Distribution of Lambdas",
       x = "Lambda",
       y = "RMSE") +
  theme_linedraw()
```



We will therefore use 0.5 as tuning parameter λ to further reduce our RMSE of the following "Regularised Movie & User Biases" model:

```
bm <- edx %>%
  group_by(movieId) %>%
  summarize(bm = sum(rating - edx_mu)/(n() + lambda))
bu <- edx %>%
  left_join(bm, by = "movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - bm - edx_mu)/(n() + lambda))
pred_reg <- edx %>%
  left_join(bm, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(predictions = edx_mu + bm + bu) %>%
  .$predictions
RMSE_4 <- RMSE(edx$rating, pred_reg)
result4_table <- tibble(Model = "Regularised Movie & User Biases", RMSE = RMSE_4)
result4_table %>%
  knitr::kable()
```

Model	RMSE
Regularised Movie & User Biases	0.8566611

This model shows a marginal improvement of from the previous one with a RMSE at 0.8566.

Section 9.5: Summarize RMSE

Let's summarize the RMSEs of the different models we developed along this project:

```r
results_table <- tibble(Model = c("Benchmark", "Movie Bias", "Movie & User Biases", "
Regularised Movie & User Biases"), RMSE = c(RMSE_1, RMSE_2, RMSE_3, RMSE_4))
results_table %>%
  knitr::kable()
```

Model	RMSE
Benchmark	1.0603926
Movie Bias	0.9423680
Movie & User Biases	0.8566699
Regularised Movie & User Biases	0.8566611

Section 10: Results

The “Regularised Movie and User Biases” Model being the one with the lowest RMSE, we will now apply it to the “final_holdout_test” dataset. The fact that the “final_holdout_test” dataset comprises about 10% of the edx dataset observations shall further reduce its RMSE.

The model: Yu,i=μ+bm+bu+ϵu,i With Yu,i = predicted rating, μ = the average rating, bm = the movie effects, bu = the user effects and ϵu,i = independent errors centred at 0.

```
#Defining "lambda" for "final_holdout_test" dataset
lambdasReg2 <- seq(0, 10, 0.25)

RMSEreg2 <- sapply(lambdasReg2, function(l){
  edx_mu2 <- mean(final_holdout_test$rating)
  bm <- final_holdout_test %>%
    group_by(movieId) %>%
    summarize(bm = sum(rating - edx_mu2)/(n() + l))
  bu <- final_holdout_test %>%
    left_join(bm, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - bm - edx_mu2)/(n() + l))
  predicted_ratings <- final_holdout_test %>%
    left_join(bm, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = edx_mu2 + bm + bu) %>%
    .$pred
  return(RMSE(predicted_ratings, final_holdout_test$rating))
})

#Applying model to "final_holdout_test
bm <- final_holdout_test %>%
  group_by(movieId) %>%
  summarize(bm = sum(rating - edx_mu)/(n() + lambda))
bu <- final_holdout_test %>%
  left_join(bm, by = "movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - bm - edx_mu)/(n() + lambda))
pred_reg <- final_holdout_test %>%
  left_join(bm, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(predictions = edx_mu + bm + bu) %>%
  .$predictions

# Computing RMSE
RMSE_final2 <- RMSE(final_holdout_test$rating, pred_reg)
resultfinal2_table <- tibble(Model = "Regularised Movie & User Biases", RMSE = RMSE_f
inal2)
resultfinal2_table %>%
  knitr::kable()
```

Model	RMSE
Regularised Movie & User Biases	0.826011

With a RMSE at 0.826, the objective of the exercise has been achieved.

Section 11: Conclusion

The objective of this project was to develop a recommendation system using the MovieLens 10M dataset that predicted ratings with a residual mean square error of less than 0.86490. Adjusting for a number of estimated biases introduced by the movie, user, genre, release year and review date, and then regularizing these in order to constrain the variability of effect sizes, met the project objective yielding a model with an RMSE of 0.85669. This was confirmed in a final test using the previously unused validation dataset, with an RMSE of 0.826.

While we built a machine learning model to predict movie ratings in the MovieLens Dataset that matches the requirements of this exercise, it is clear that we could have gone further at building a stronger model by taking in account other perspective such as “Movie Age”, “Genre”, etc. We could have as well developed more complex models based on, for example, Distributed Random Forest or many more. We focused on the most basic model development: we achieved our goal.