HarvardX DS Capstone MovieLens Project

Ernest Kollieguwor

7/11/2021

Preface

The capstone project of HarvardX's Data Science Professional Certificate program on the Edx's website served the basis for this report. The R Markdown code used to generate the report and its PDF version are available on GitHub. HTML version may also be available on RPubs.

Introduction

A user is able to predict the rating or other preferences of a given item using a subclass information filtering system called a "Recommendation System". Customers rating is used by companies with huge customers group to predict their rating or preferences of their products. Movie companies like Netflix predict user rating for specific movies using a recommendation system. The Data Science Community was challenged in 2006 of October, to enhance the Netflix recommendation algorithm by 10% for a million dollars award. The winners were announced in September of 2009. Considering some of the data analysis tactics the winning team used, you can read a good summary with detailed narrative here in this assignment which has similar goal to recommends movies on a rating scale using a recommendation system.

Data set

The MovieLens Data set will be used for this project. The GroupLens Research collected this data set and it can be found at this web site (http://movielens.org).

Loading the Data set

The course structure provided code in this link (https://bit.ly/2Ng6tVW) the data is split into an edx set and 10% validation set using this link. The edx data set will be further split into a training set and testing set and the final evaluation will be made on the validation set.

```
if(!require(tidyverse)) install.packages("tidyverse", repos =
"http://cran.us.r-project.org")
## Loading required package: tidyverse
## ── Attaching packages ─────────────────────────────────────── tidyverse
1.3.1 ──
```

```
## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.2     ✓ dplyr   1.0.7
## ✓ tidyr   1.1.3     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.1
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-
project.org")
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
if(!require(data.table)) install.packages("data.table", repos =
"http://cran.us.r-project.org")
## 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
if(!require(recosystem)) install.packages("recosystem", repos =
"http://cran.us.r-project.org")
## Loading required package: recosystem
if(!require(ggthemes)) install.packages("ggthemes", repos =
"http://cran.us.r-project.org")
## Loading required package: ggthemes
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-
project.org")
## Loading required package: scales
##
## Attaching package: 'scales'
## The following object is masked from 'package:purrr':
##
##     discard
## The following object is masked from 'package:readr':
##
##     col_factor
library(tidyverse)
library(caret)
library(data.table)
library(recosystem)
library(knitr)
library(ggthemes)
library(scales)
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
library(tinytex)
library(rmarkdown)
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

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

# if using R 4.0 or later:

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
set.seed(1, sample.kind="Rounding")
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
sampler
## used
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")
edx <- rbind(edx, removed)

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

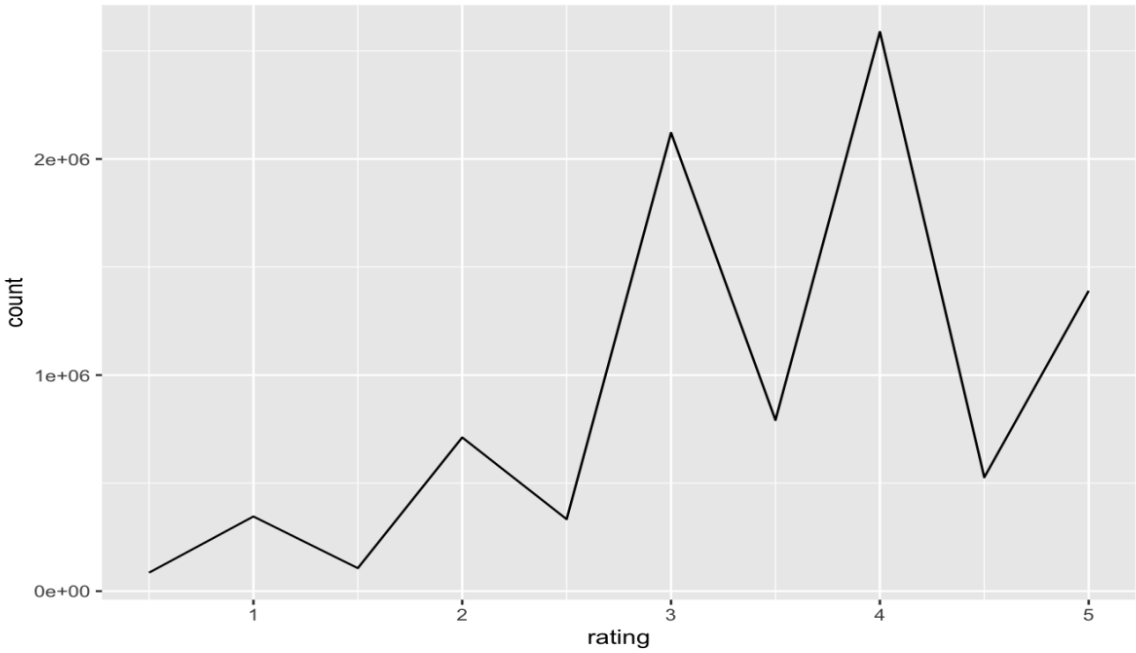
Partitioning the data set to define the train_set and the test_set

```
set.seed(1, sample.kind="Rounding")
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
sampler
## used
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list =
FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Matching userId and movieId in both train and test sets

test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Adding back rows into train set

removed <- anti_join(temp, test_set)
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
"genres")
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

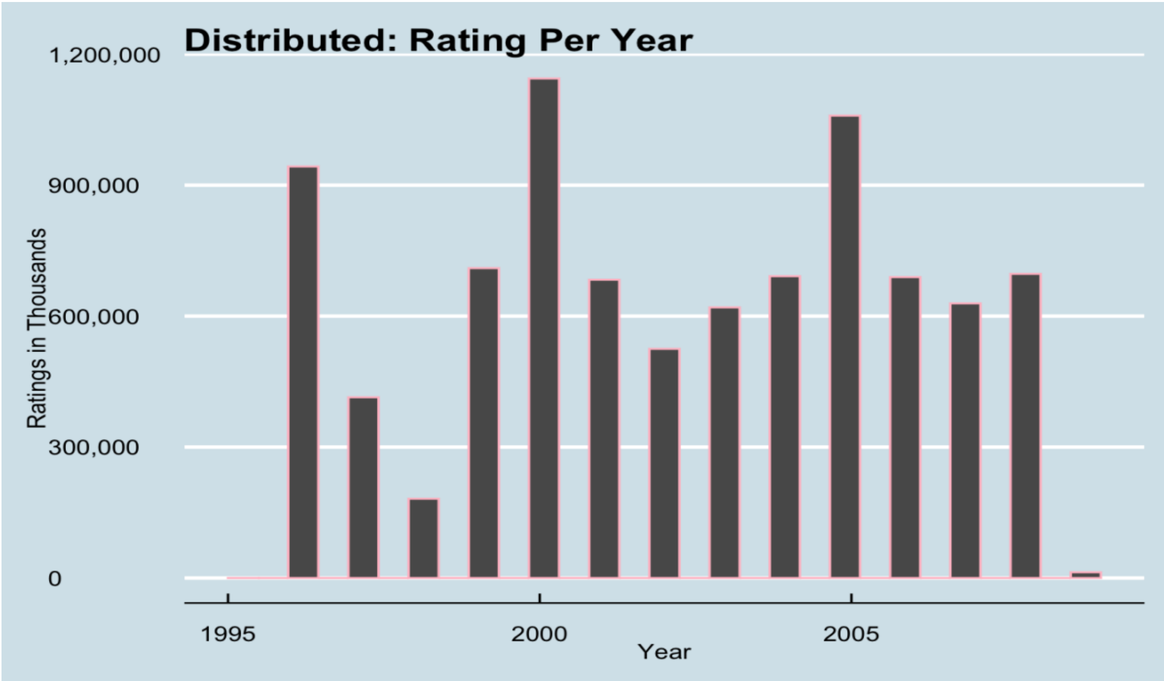
Exploratory Data Analysis

```
# Number of rows and columns in the edx dataset?

nrow(edx)
## [1] 9000055
ncol(edx)
## [1] 6
# Number of zeros given as ratings in the edx dataset?

edx %>% filter(rating == 0) %>% tally()
##   n
## 1 0
# Number of threes given as ratings in the edx dataset?

edx %>% filter(rating == 3) %>% tally()
##         n
## 1 2121240
# Different movies in the edx dataset?

n_distinct(edx$movieId)
## [1] 10677
# Different users in the edx data set?

n_distinct(edx$userId)
```

```
## [1] 69878
# Detecting the structure of the edx data set.

edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  head()
## # A tibble: 6 x 2
##   genres                                              n
##   <chr>                                           <int>
## 1 (no genres listed)                                  7
## 2 Action                                          24482
## 3 Action|Adventure                                68688
## 4 Action|Adventure|Animation|Children|Comedy       7467
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy   187
## 6 Action|Adventure|Animation|Children|Comedy|IMAX     66
# genres in ascending order

tibble(count = str_count(edx$genres, fixed("|")), genres = edx$genres) %>%
  group_by(count, genres) %>%
  summarise(n = n()) %>%
  arrange(count) %>%
  head()
## `summarise()` has grouped output by 'count'. You can override using the
`.groups` argument.
## # A tibble: 6 x 3
## # Groups:   count [1]
##   count genres                  n
##   <int> <chr>               <int>
## 1     0 (no genres listed)      7
## 2     0 Action              24482
## 3     0 Adventure            2276
## 4     0 Animation             329
## 5     0 Children              745
## 6     0 Comedy             700889
# In general, half star ratings are less common than whole star ratings

edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line()
```




```
# Distributing ratings per year

edx %>% mutate(year = year(as_datetime(timestamp, origin="1970-01-01"))) %>%
  ggplot(aes(x=year)) +
  geom_histogram(color = "pink") +
  ggtitle("Distributed: Rating Per Year") +
  xlab("Year") +
  ylab("Ratings in Thousands") +
  scale_y_continuous(labels = comma) +
  theme_economist()
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Movie with the greatest number of ratings?

edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
## `summarise()` has grouped output by 'movieId'. You can override using the
`.groups` argument.
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title
count
##      <dbl> <chr>
<int>
##  1     296 Pulp Fiction (1994)
31362
##  2     356 Forrest Gump (1994)
31079
##  3     593 Silence of the Lambs, The (1991)
30382
##  4     480 Jurassic Park (1993)
29360
##  5     318 Shawshank Redemption, The (1994)
28015
##  6     110 Braveheart (1995)
26212
##  7     457 Fugitive, The (1993)
25998
##  8     589 Terminator 2: Judgment Day (1991)
25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
25672
```

```
## 10     150 Apollo 13 (1995)
24284
## # … with 10,667 more rows
# The ten most given ratings in order from most to least?

edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(10) %>%
  arrange(desc(count))
## Selecting by count
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1     4   2588430
##  2     3   2121240
##  3     5   1390114
##  4     3.5  791624
##  5     2    711422
##  6     4.5  526736
##  7     1    345679
##  8     2.5  333010
##  9     1.5  106426
## 10     0.5   85374
```

Method and Evaluation

Five models will be built and evaluated starting with the simplest, and then the Root Mean Square Error (RMSE) will be used to evaluate each model's accuracy. Finally, the fifth model's accuracy will be evaluated with the validation set created earlier to derived the lowest RMSE.

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

Building Model

Let's see how prediction begins at this moment by replicating the x value at 2 and halt times of the test_set up to maximum of 1,799,966 rows, and counting the number of rows in the predicted test_set and then predicting the RMSE of its rating. The function starts with replicating values in the test_set.

Model 1:

The simplest model assumes a random distribution of error from movie to movie variations, when predicting that all users will rate all movie the same.Considering statistics theory, the mean, which is just the average of all observed ratings, minimizes the RMSE, as described in the formula below. $\hat{Y}$ u,i=μ+ϵi,u

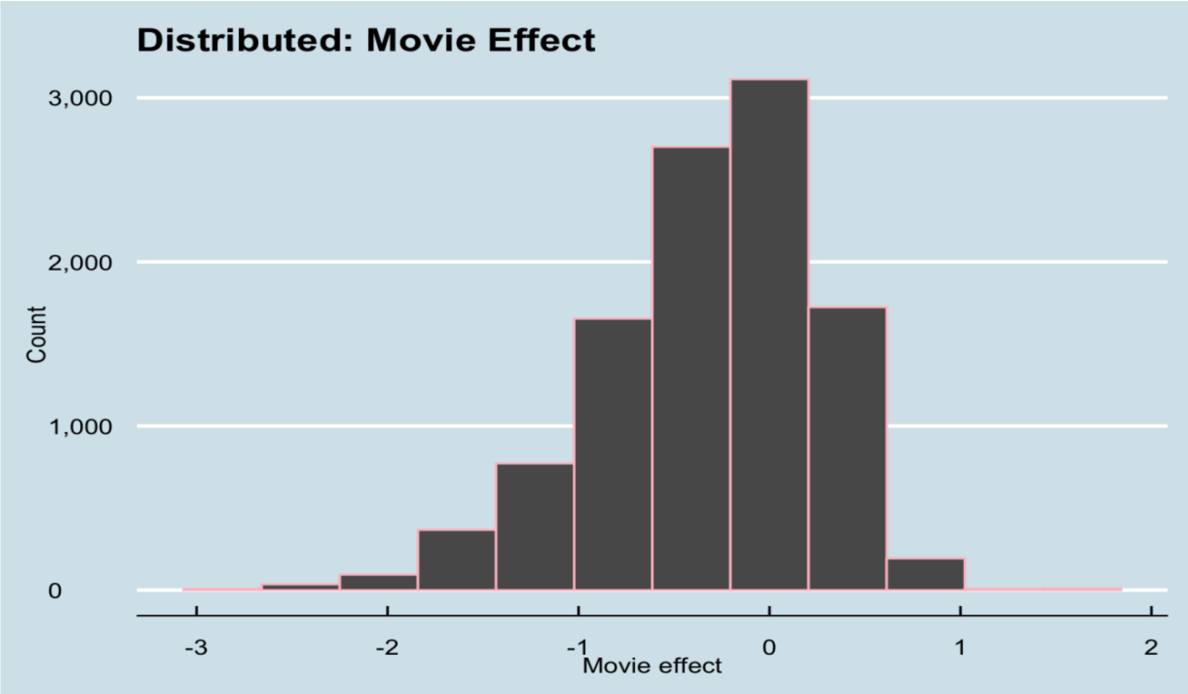
```
# Model 1: Just the average of the data set observations.

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

```
rmse1 <- RMSE(test_set$rating, mu)

# replicating the x value at 2 and halt times of the test_set

predictions <- rep(2.5, nrow(test_set))
RMSE(test_set$rating, predictions)
## [1] 1.465736
# creating a table to store the rmse results every step along the way

naive_rmse <- RMSE(test_set$rating, mu)

rmse_outputs <- tibble(Method = "Model 1: The Overall Average", RMSE =
naive_rmse)
```

Model 2

From exploratory data analysis, it was observed that some movies are more popular than others and receive higher ratings. Considering the movie effect, this model will be improved by adding the term bi to the formula used to determine the average of all movies like this; Yu,i = μ + bi + ϵu,i

```
# Model 2: the movie effect on ratings

bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Visual description of the movie effect normal distribution

bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=12, col = I("pink")) +
  ggtitle("Distributed: Movie Effect") +
  xlab("Movie effect") +
  ylab("Count") +
  scale_y_continuous(labels = comma) +
  theme_economist()
```




```
# Predicting the rmse of the movie effect

predicted_ratings <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i
movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_outputs <- bind_rows(rmse_outputs, tibble(Method = "Model 2: The Movie
Effect", RMSE = movie_effect_rmse))
rmse_outputs %>% knitr::kable()
```

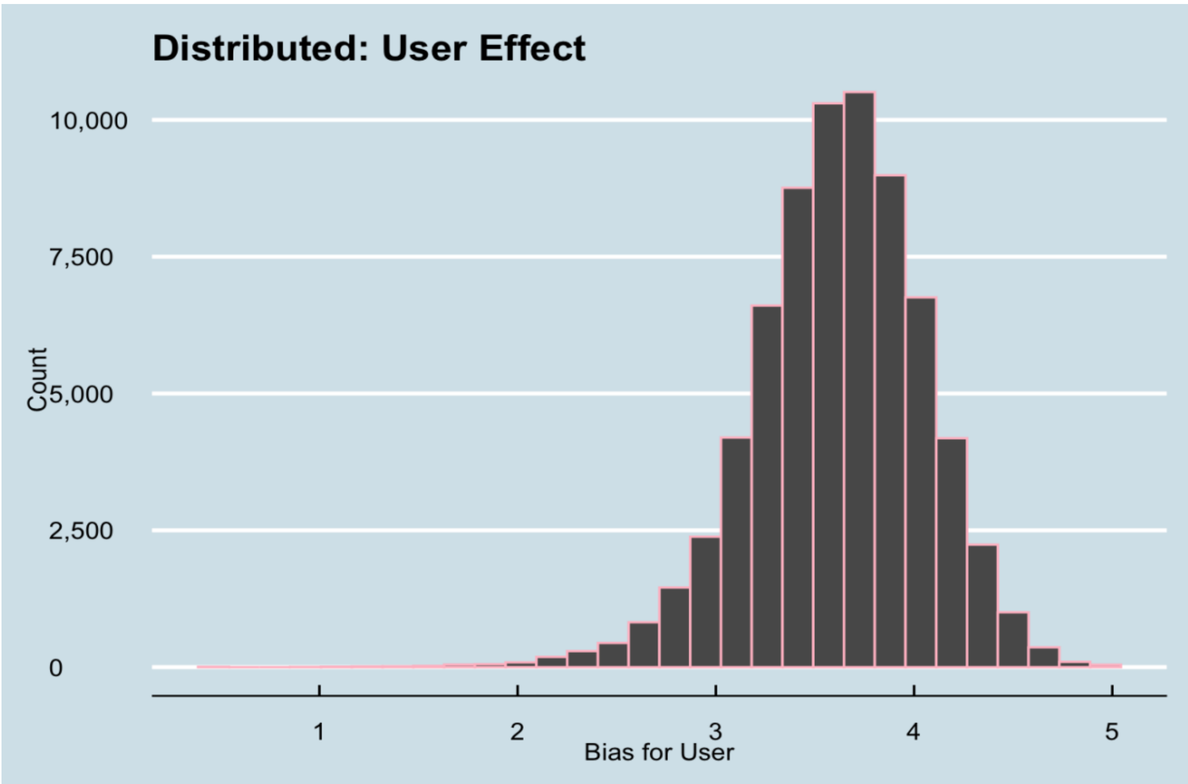
Method	RMSE
Model 1: The Overall Average	1.0599043
Model 2: The Movie Effect	0.9437429

Model 3

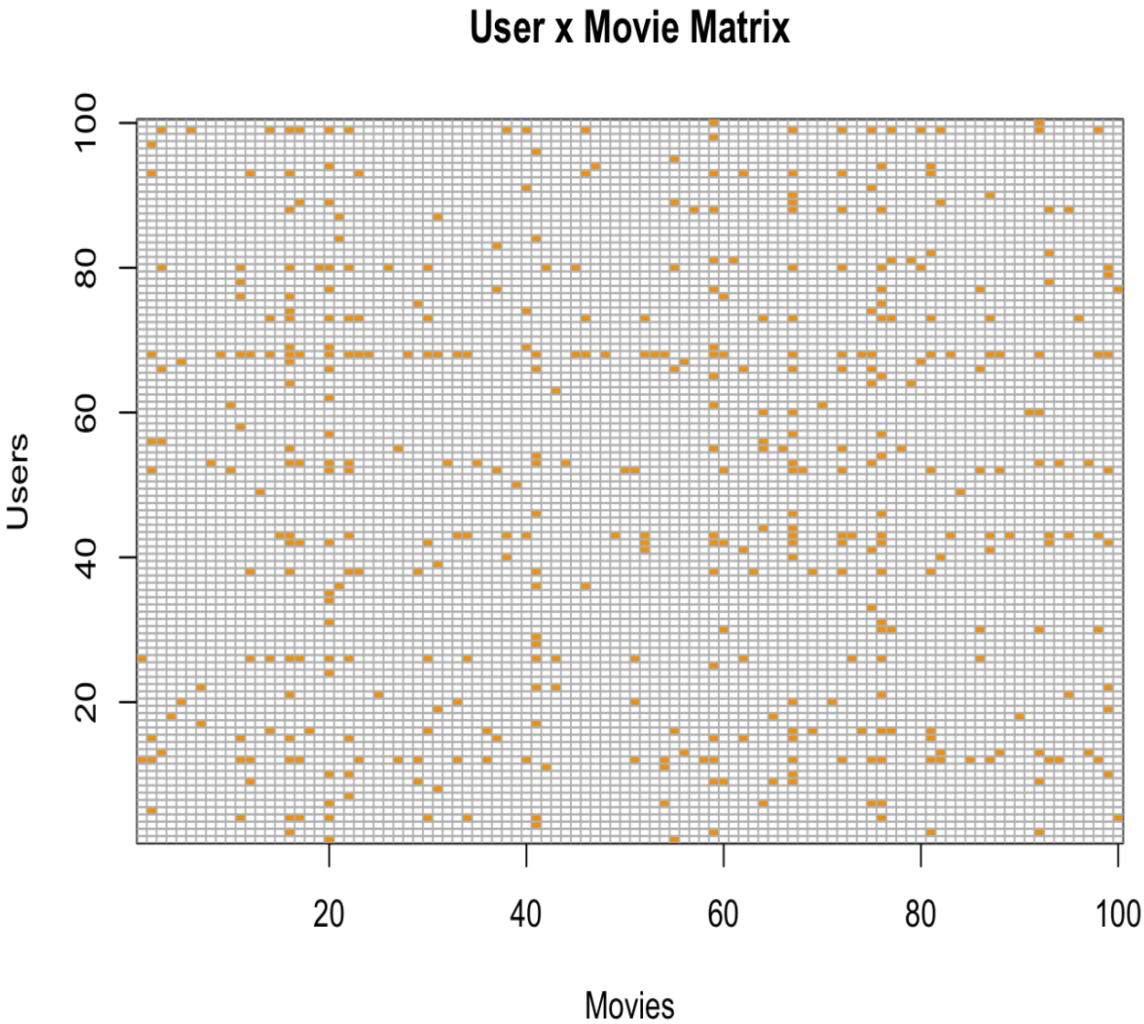
Considering the user's effect, this model can be improved by adding the term "bu" to the formula used in previous model like this; $Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

```
# Model 3: the user's specific effect on ratings

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

```
# Normal distribution for the user effect

train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(color = "pink") +
  ggtitle("Distributed: User Effect") +
  xlab("Bias for User") +
  ylab("Count") +
  scale_y_continuous(labels = comma) +
  theme_economist()
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# plotting the movie and user matrix

users <- sample(unique(edx$userId), 100)
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
title("User x Movie Matrix")
```




```
# Predicting the rmse of the user effect

predicted_ratings <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_outputs <- bind_rows(rmse_outputs, tibble(Method = "Model 3: The Movie &
User Effect", RMSE = user_effect_rmse))
rmse_outputs %>% knitr::kable()
```

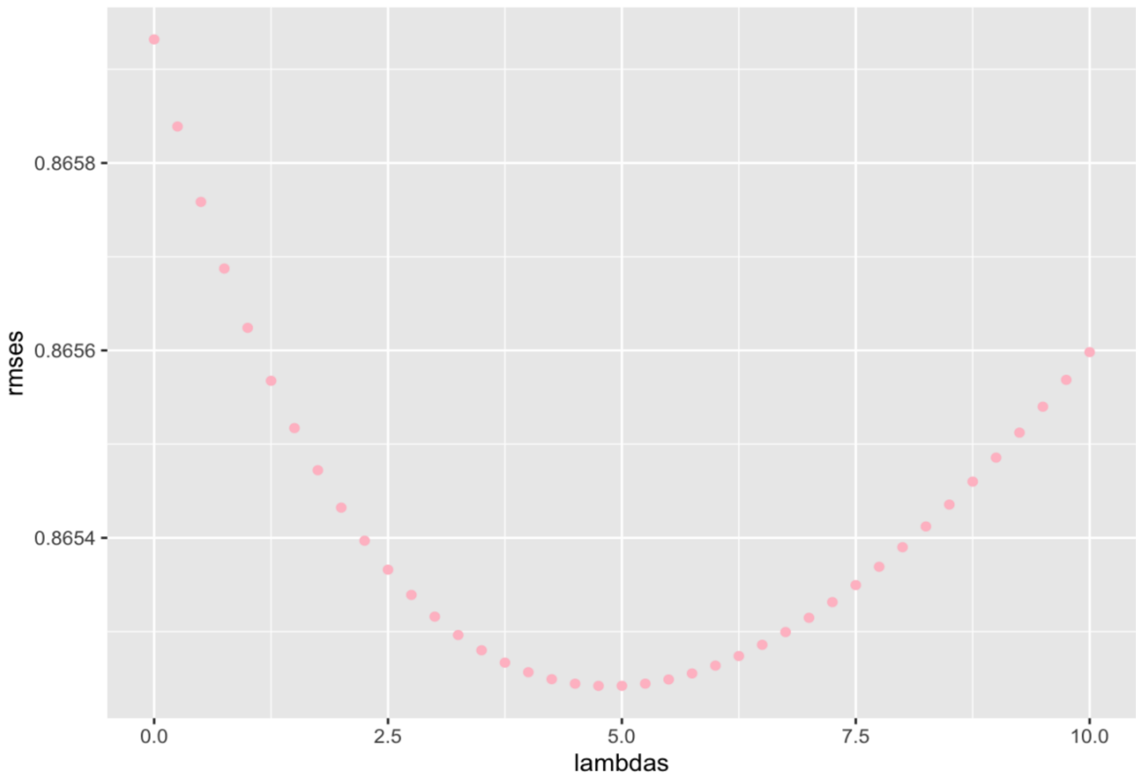
Method	RMSE
Model 1: The Overall Average	1.0599043
Model 2: The Movie Effect	0.9437429

Method	RMSE
Model 3: The Movie & User Effect	0.8659319

Model 4:

Regularizing the movie and user effects to penalize or reduce noisy data. Here, three sets of lambdas are defined to tune lambdas beforehand.

```
# Model 4: regularizing the mean, movie and user effects on rating using the
best parameters from lanbdas

lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(x){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+x))
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+x))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})


# plotting lambdas vs. RMSE

qplot(lambdas, rmses, color = I("pink"))
```




```
# Picking lambdas with the lowest RMSE to be used for regularizing the movie
and user effects.

lamb <- lambdas[which.min(rmses)]
lamb
## [1] 4.75
# Predicting the rmse from a regularized movie and user effects

rmse_outputs <- bind_rows(rmse_outputs, tibble(Method = "Model 4:
Regularizing - the Movie and User Effects", RMSE = min(rmses)))
rmse_outputs %>% knitr::kable()
```

Method	RMSE
Model 1: The Overall Average	1.0599043
Model 2: The Movie Effect	0.9437429
Model 3: The Movie & User Effect	0.8659319
Model 4: Regularizing - the Movie and User Effects	0.8652421

Model 5:

Matrix Factorization - the alternative Recosystem will be used instead due to the memory gap on commercial computer currently in use. Here, the best tuning parameters is used from an R suggested class object called Reco(). The train() method allows for a set of parameters inside the function and then, the $predict() is used for predicted values.

Model 5: matrix factorization - alternatively using the recosystem for tuning due to memory gap.

```
set.seed(1, sample.kind="Rounding")
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
sampler
## used
train_reco <- with(train_set, data_memory(user_index = userId, item_index =
movieId, rating = rating))
test_reco <- with(test_set, data_memory(user_index = userId, item_index =
movieId, rating = rating))
rec <- Reco()

alt_reco <- rec$tune(train_reco, opts = list(dim = c(20, 30),
                                             lrate = c(0.01, 0.1),
                                             costp_l1 = c(0.01, 0.1),
                                             costq_l1 = c(0.01, 0.1),
                                             nthread = 4,
                                             niter = 10))

rec$train(train_reco, opts = c(alt_reco$min, nthread = 4, niter = 40))
## iter      tr_rmse          obj
##    0       0.9724   1.0184e+07
##    1       0.8780   8.9614e+06
##    2       0.8500   8.6477e+06
##    3       0.8299   8.4391e+06
##    4       0.8132   8.2794e+06
##    5       0.7990   8.1480e+06
##    6       0.7865   8.0355e+06
##    7       0.7756   7.9441e+06
##    8       0.7661   7.8658e+06
##    9       0.7577   7.7975e+06
##   10       0.7504   7.7407e+06
##   11       0.7437   7.6866e+06
##   12       0.7377   7.6405e+06
##   13       0.7325   7.6010e+06
##   14       0.7281   7.5689e+06
##   15       0.7238   7.5352e+06
##   16       0.7201   7.5061e+06
##   17       0.7168   7.4795e+06
##   18       0.7137   7.4536e+06
##   19       0.7111   7.4325e+06
##   20       0.7087   7.4121e+06
##   21       0.7066   7.3937e+06
##   22       0.7046   7.3760e+06
##   23       0.7029   7.3596e+06
##   24       0.7013   7.3433e+06
##   25       0.6998   7.3274e+06
##   26       0.6986   7.3126e+06
##   27       0.6976   7.3001e+06
##   28       0.6965   7.2853e+06
##   29       0.6956   7.2728e+06
##   30       0.6948   7.2597e+06
##   31       0.6941   7.2484e+06
##   32       0.6935   7.2362e+06
##   33       0.6929   7.2233e+06
##   34       0.6926   7.2142e+06
```

```
##   35        0.6922   7.2031e+06
##   36        0.6918   7.1919e+06
##   37        0.6915   7.1805e+06
##   38        0.6913   7.1713e+06
##   39        0.6912   7.1614e+06
results_alt_reco <- rec$predict(test_reco, out_memory())

mat_factor_rmse <- RMSE(results_alt_reco, test_set$rating)
rmse_outputs <- bind_rows(rmse_outputs, tibble(Method = "Model 5: Matrix
factorization - alternative recosystem", RMSE = mat_factor_rmse))
rmse_outputs %>% knitr::kable()
```

Method	RMSE
Model 1: The Overall Average	1.0599043
Model 2: The Movie Effect	0.9437429
Model 3: The Movie & User Effect	0.8659319
Model 4: Regularizing - the Movie and User Effects	0.8652421
Model 5: Matrix factorization - alternative recosystem	0.8000766

Finalizing rmse prediction on the validation set

The lowest thus far, has been obtained on the fourth of four models using matrix factorization with the recosystem. Finally, the edx data set will be used to train result fromm the fourth model, while the validation set will be used to test for accuracy.

```
set.seed(1, sample.kind="Rounding")
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
sampler
## used
edx_reco_sys <- with(edx, data_memory(user_index = userId, item_index =
movieId, rating = rating))
valid_reco <- with(validation, data_memory(user_index = userId, item_index =
movieId, rating = rating))
rec <- Reco()

alt_reco <- rec$tune(edx_reco_sys, opts = list(dim = c(20, 30),
                                               lrate = c(0.01, 0.1),
                                               costp_l2 = c(0.01, 0.1),
                                               costq_l2 = c(0.01, 0.1),
                                               nthread = 4,
                                               niter = 10))

rec$train(edx_reco_sys, opts = c(alt_reco$min, nthread = 4, niter = 40))
## iter      tr_rmse          obj
##    0       0.9728   1.2007e+07
##    1       0.8723   9.8747e+06
##    2       0.8386   9.1707e+06
##    3       0.8164   8.7432e+06
##    4       0.8011   8.4685e+06
##    5       0.7893   8.2728e+06
##    6       0.7797   8.1233e+06
##    7       0.7717   8.0051e+06
```

```
##    8        0.7649   7.9070e+06
##    9        0.7590   7.8233e+06
##   10        0.7539   7.7594e+06
##   11        0.7493   7.7032e+06
##   12        0.7450   7.6506e+06
##   13        0.7413   7.6067e+06
##   14        0.7379   7.5689e+06
##   15        0.7347   7.5328e+06
##   16        0.7318   7.5007e+06
##   17        0.7292   7.4756e+06
##   18        0.7266   7.4483e+06
##   19        0.7243   7.4257e+06
##   20        0.7222   7.4049e+06
##   21        0.7203   7.3874e+06
##   22        0.7184   7.3688e+06
##   23        0.7167   7.3527e+06
##   24        0.7150   7.3377e+06
##   25        0.7135   7.3226e+06
##   26        0.7121   7.3103e+06
##   27        0.7107   7.2993e+06
##   28        0.7094   7.2867e+06
##   29        0.7082   7.2749e+06
##   30        0.7071   7.2658e+06
##   31        0.7060   7.2564e+06
##   32        0.7050   7.2475e+06
##   33        0.7040   7.2399e+06
##   34        0.7032   7.2322e+06
##   35        0.7023   7.2253e+06
##   36        0.7015   7.2192e+06
##   37        0.7006   7.2101e+06
##   38        0.6999   7.2051e+06
##   39        0.6992   7.1998e+06
valid_reco <- rec$predict(valid_reco, out_memory())

valid_final_rmse <- RMSE(valid_reco, validation$rating)
rmse_outputs <- bind_rows(rmse_outputs, tibble(Method = "Final validation:
Matrix factorization - alternative recosystem", RMSE = valid_final_rmse))
rmse_outputs %>% knitr::kable()
```

Method	RMSE
Model 1: The Overall Average	1.0599043
Model 2: The Movie Effect	0.9437429
Model 3: The Movie & User Effect	0.8659319
Model 4: Regularizing - the Movie and User Effects	0.8652421
Model 5: Matrix factorization - alternative recosystem	0.8000766
Final validation: Matrix factorization - alternative recosystem	0.7805196

Conclusion

A naive approach have been implemented together with the movie effect and user- movie effect taken as second and third models respectively. Furthermore, the regularization and an alternative matrix factorization were considered as fourth and fifth models respectively, and the lowest RMSE of 0.7805 was derived using the fifth and final validation data set.