Movielens Report

Guillermo Ortiz

13/2/2021

Executive Summary

Recommender systems have gained popularity in recent years as useful tools for a variety of uses. They are mostly used for entertainment purposes in music or video applications, suggestions for purchases of various kinds and even matchmaking in dating applications. In this work, a movie recommendation system based on the Netflix challenge for ratings prediction is developed. A version of the movielens database with 10M observations is used. Machine learning algorithms are used to generate predictions for movie ratings. The Root Mean Squared Error (RMSE) is used to assess the quality of the predictions. By using a different algorithms, the final RMSE obtained is 0.8651 for the ratings in the validation set.

Introduction

Recommender systems have gained popularity in recent years as useful tools for a variety of uses. They are mostly used for entertainment purposes in music or video applications, suggestions for purchases of various kinds and even matchmaking in dating applications.

In this work, a movie recommendation system based on the Netflix challenge for ratings prediction is developed. A version of the movielens database with 10M observations is used. Machine learning algorithms are used to generate predictions for movie ratings. The Root Mean Squared Error (RMSE) is used to assess the quality of the predictions.

First, the data is donwloaded and divided into two sets: edx and validation. The edx dataset will be used to train the algorithms while the validation set will serve for evaluation purposes only. Second, different models and algorithms are tested using the edx dataset, which is sub-divided into two: the train set and the test set. In the train set, the models and algorithms are developed while the test set is used to choose the best algorithm possible. The RMSE is used to choose the best possible algorithm as well.

Finally, when the best algorithm is identified, it is applied to the entire edx dataset. Once the predictions are made, the validation dataset is used to obtain the final RMSE of this work.

Data

Loading required packages and options

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

```
## 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   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## Warning: package 'ggplot2' was built under R version 4.1.2

## -- 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(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ggthemes

## Warning: package 'ggthemes' was built under R version 4.1.2
```

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

```
## Loading required package: 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(tidyverse)
library(caret)
library(data.table)
library(ggthemes)
library(lubridate)
options(digits = 4)
options(scipen = 999)
```

Loading Data

```
load("C:/Users/gfeor/Desktop/R Professional Certificate/9. Capstone/movielens/data/movielens_datasets.RD
```

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

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

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

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

movies <- as.data.frame(movies) %>% 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, 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)
```

The 10M dataset is divided in two sub-datasets: "edx" and "validation". The "edx" dataset is the one that it is going to be used for training, while the "validation" dataset will serve to evaluate the machine learning algorithm.

Data Exploration

The edx dataset consists on 9,000,005 observations with 6 variables. Each observation represents a movie that was given a rating from a specific user.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 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 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

The validation dataset consists of 999,999 observations with the same variables as in the edx dataset. These observations will be used to test whether or not the predictions issued by the machine learning algorithm exercise were accurate or not.

```
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
##  - attr(*, ".internal.selfref")=<externalptr>
```

Here, the "edx" dataset will be further explored.

As it was mentioned earlier, the edx dataset has 9,000,005 observations and 6 variables. The variables are userId, movieId, rating, timestamp, title and genres. Timestamps represent seconds since midnight UTC January 1, 1970.

The summary function provides a statistical summary of the data.

```
summary(edx)
```

```
##      userId         movieId          rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   : 789652009
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.00   1st Qu.: 946768283
##  Median :35738   Median : 1834   Median :4.00   Median :1035493918
##  Mean   :35870   Mean   : 4122   Mean   :3.51   Mean   :1032615907
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.00   3rd Qu.:1126750881
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1231131736
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

There are 10,677 unique movies and 69,878 unique users in the edx dataset.

```
length(unique(edx$movieId))     #number of unique movies in edx dataset
```

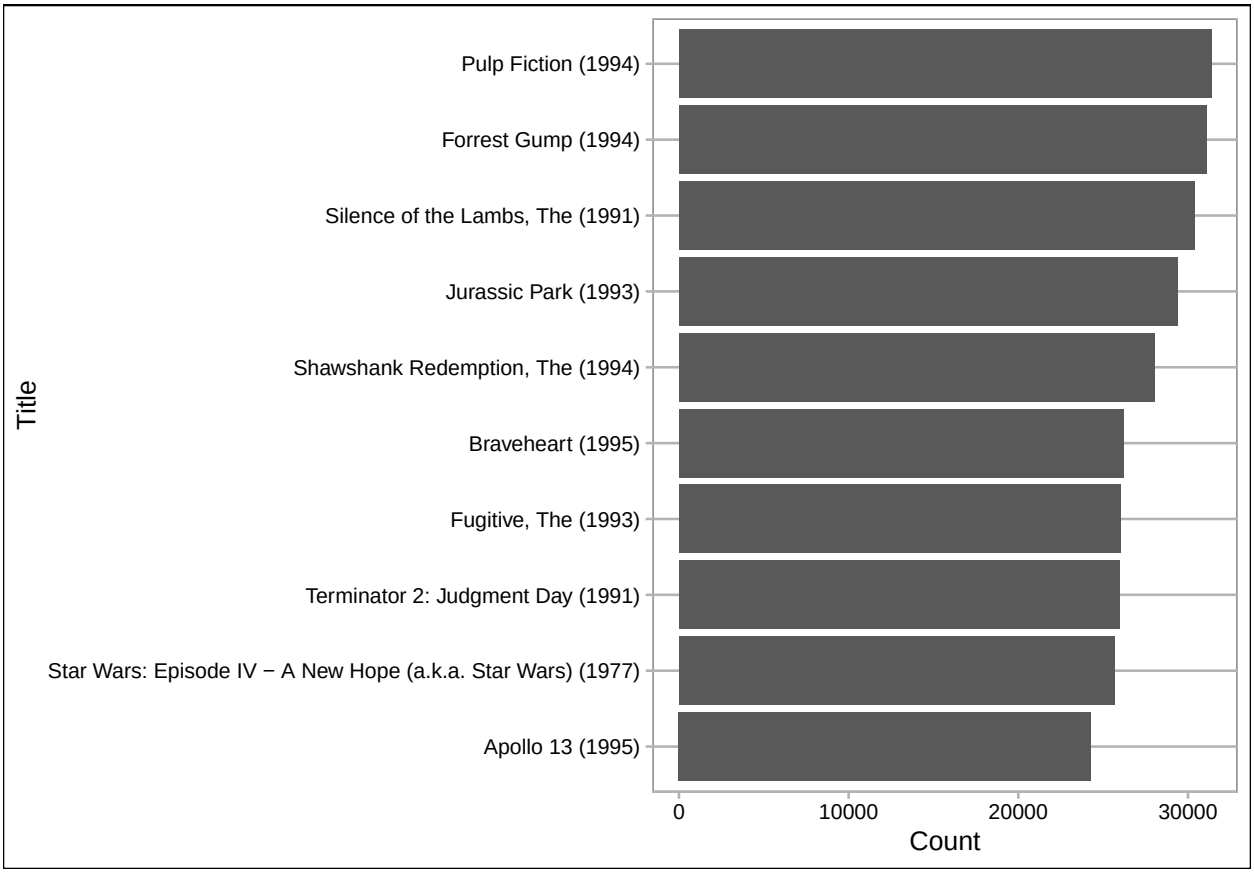
```
## [1] 10677
```

```
length(unique(edx$userId))      #number of unique users in edx dataset
```

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

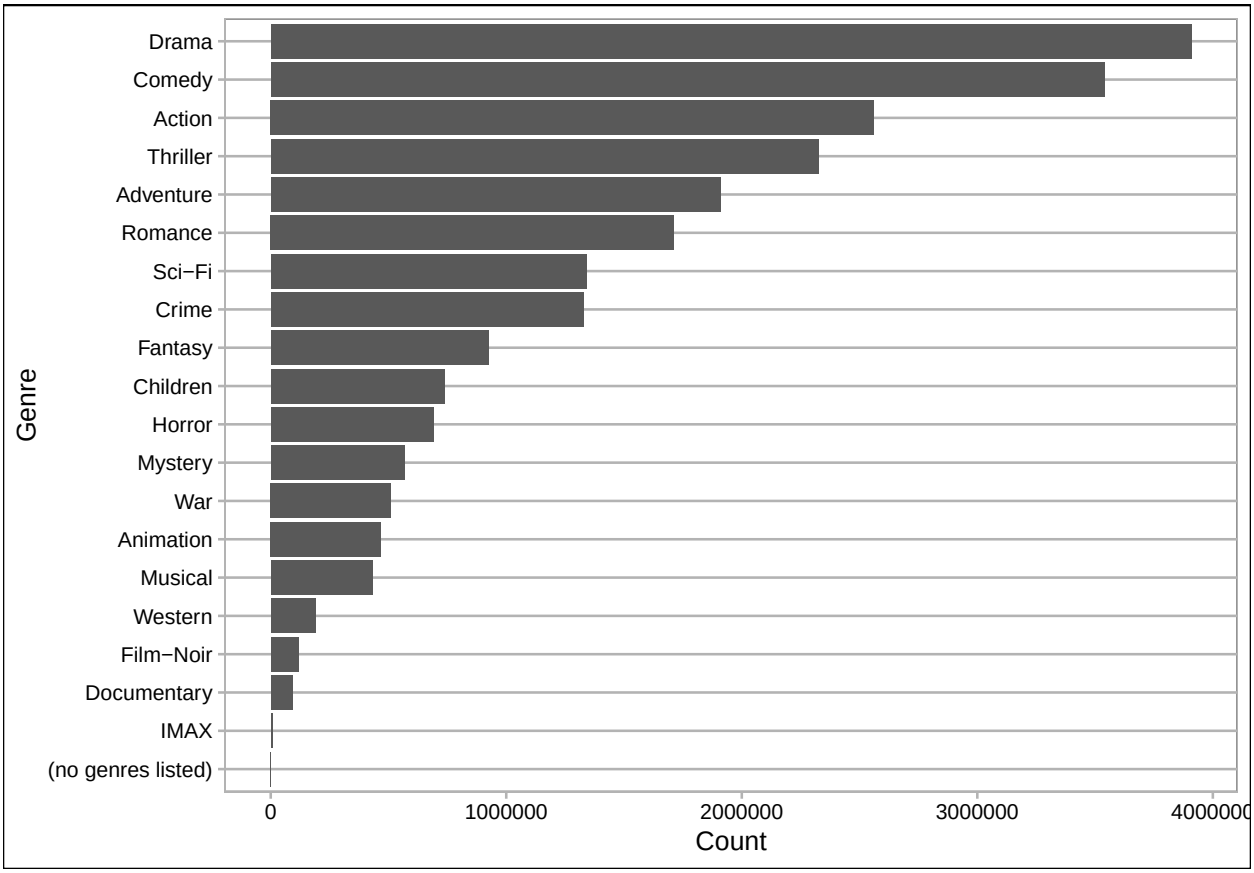
The five most popular movies in the edx dataset are: Pulp Fiction, Forrest Gump, The Silence of the Lambs, Jurassic Park and The Shawshank Redemption.

```
edx %>% group_by(title) %>%
  summarise(ratings=n()) %>%
  top_n(10, ratings) %>%
  ggplot(aes(ratings, reorder(title,ratings))) +
  geom_col() +
  xlab("Count") + ylab("Title") +
  theme_calc()
```



The three most popular genres in the edx dataset are: Drama, Comedy and Action.

```
edx %>% group_by(genres) %>%
  summarize(n=n()) %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count=sum(n)) %>%
  ggplot(aes(count, reorder(genres,count))) +
  geom_col() +
  xlab("Count") + ylab("Genre") +
  theme_calc()
```



The distribution of the movie ratings shows a range of 0.5 to 5 with half star ratings being less common than whole star ratings.

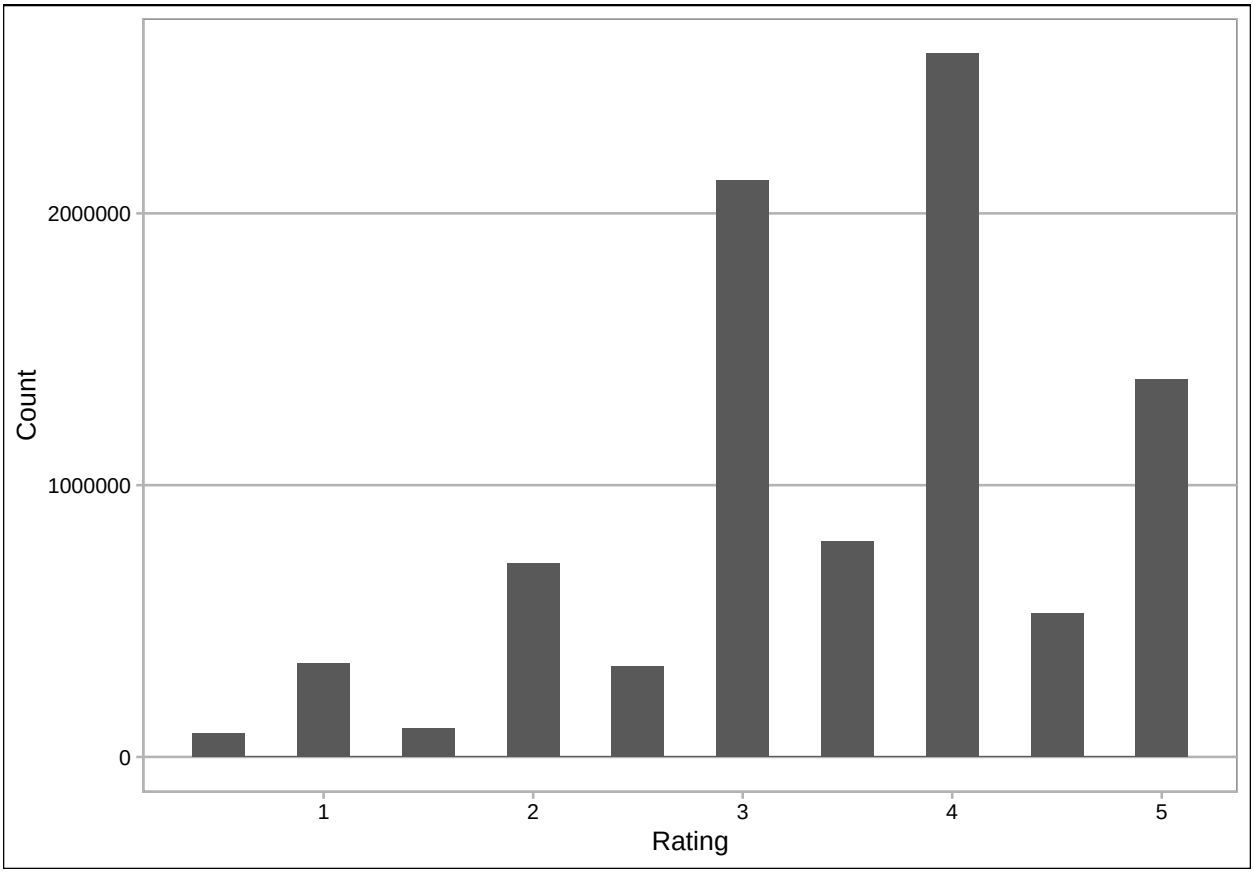
```
min(edx$rating) #minimum rating in edx dataset
```

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

```
max(edx$rating) #maximum rating in edx dataset
```

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

```
ggplot(edx, aes(rating)) +
  geom_histogram(binwidth = 0.25) +
  xlab("Rating") + ylab("Count") +
  theme_calc()
```



Methods

First of all, using the edx dataset, the train and test sets are created, in order to evaluate the algorithms without using the validation dataset.

```
set.seed(1)

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train <- edx[-test_index,]
test <- edx[test_index,]

rm(test_index)
```

To make sure there are no movies nor users in the test set that don't appear on the training set, the following code is used:

```
test <- test %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

The Root Mean Squared Error (RMSE) measure will be used to evaluate the predictions from the algorithm. The RMSE function is created:

```r
RMSE <- function(true, prediction){
  sqrt(mean((true - prediction)^2))
}
```

First Model: Average rating
In this first model, the average rating is used as prediction for all observations in the test set.

```r
mu <- mean(train$rating) # expected value
rmse_avg <- RMSE(mu, test$rating)
rmse_results <- tibble(method = "Average rating",
                       RMSE = rmse_avg)
rmse_results
```

```
## # A tibble: 1 x 2
##   method          RMSE
##   <chr>          <dbl>
## 1 Average rating  1.06
```

The RMSE is approximately equal to 1.06. This is a very high value. More refined methods are needed.

Second Model: Incorporating MOVIE EFFECTS
In this second model, besides taking into account the average ratings, specific movie effects are incorporated into the predictions.

```r
avgs_movie <- train %>% group_by(movieId) %>%
  summarize(bi = mean(rating - mu)) # movie effects

pred_movies <- test %>%
  left_join(avgs_movie, by = "movieId") %>%
  mutate(pred_movies = mu+bi) %>%
  pull(pred_movies)

rmse_movies <- RMSE(pred_movies, test$rating)

rmse_results <- rmse_results %>% rbind(c("Movie effects",
                                         rmse_movies))
rmse_results
```

```
## # A tibble: 2 x 2
##   method         RMSE
##   <chr>          <chr>
## 1 Average rating 1.05990428951531
## 2 Movie effects  0.9437429136878
```

The RMSE now is approximately equal to 0.9437. This is 10.8% lower than the RMSE from the First Model. However, there is still a lot of room for improvement.

Third Model: Incorporating MOVIE + USER EFFECTS
Here, the model takes into account the movies and the user effects, or characteristics, to make predictions.

```
avgs_user <- train %>%
  left_join(avgs_movie, by='movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi)) # user effects

pred_users <- test %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  mutate(pred_users = mu + bi + bu) %>%
  pull(pred_users)

rmse_users <- RMSE(pred_users, test$rating)

rmse_results <- rmse_results %>% rbind(c("Movie + User effects",
                                         rmse_users))

rmse_results
```

```
## # A tibble: 3 x 2
##   method               RMSE
##   <chr>                <chr>
## 1 Average rating       1.05990428951531
## 2 Movie effects        0.9437429136878
## 3 Movie + User effects 0.865931962476235
```

The RMSE is 0.8659. A 8.2% decrease with respect to the previous model.

Fourth Model: Incorporating MOVIE + USER + GENRE EFFECTS
Here, we also add the genre effects.

```
avgs_genre <- train %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  group_by(genres) %>%
  summarize(bg = mean(rating - mu - bi - bu)) # genre effects

pred_genres <- test %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  left_join(avgs_genre, by='genres') %>%
  mutate(pred_genres = mu + bi + bu + bg) %>%
  pull(pred_genres)

rmse_genres <- RMSE(pred_genres, test$rating)

rmse_results <- rmse_results %>% rbind(c("Movie + User + Genre effects",
                                         rmse_genres))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                         RMSE
##   <chr>                          <chr>
```

```
## 1 Average rating               1.05990428951531
## 2 Movie effects                0.9437429136878
## 3 Movie + User effects         0.865931962476235
## 4 Movie + User + Genre effects 0.865594068044791
```

The RMSE is 0.8656. This represents a 0.03% decrease with respect to the previous model.

Fifth Model: Incorporating MOVIE + USER + GENRE + YEAR EFFECTS Here, we also add the year effects.

```
train <- train %>% mutate(y = as.numeric(str_sub(title,-5,-2)))

test <- test %>% mutate(y = as.numeric(str_sub(title,-5,-2)))

avgs_year <- train %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  left_join(avgs_genre, by='genres') %>%
  group_by(y) %>%
  summarize(by = mean(rating - mu - bi - bu - bg)) # year effects

pred_year <- test %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  left_join(avgs_genre, by='genres') %>%
  left_join(avgs_year, by='y') %>%
  mutate(pred_year = mu + bi + bu + bg + by) %>%
  pull(pred_year)

rmse_year <- RMSE(pred_year, test$rating)

rmse_results <- rmse_results %>% rbind(c("Movie + User + Genre + Year effects",
                                         rmse_year))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                              RMSE
##   <chr>                               <chr>
## 1 Average rating                      1.05990428951531
## 2 Movie effects                       0.9437429136878
## 3 Movie + User effects                0.865931962476235
## 4 Movie + User + Genre effects        0.865594068044791
## 5 Movie + User + Genre + Year effects 0.86541892942611
```

The RMSE is 0.86542. This represents a 0.02% decrease with respect to the previous model.

Sixth Model: Incorporating MOVIE + USER + GENRE + YEAR + MONTH EFFECTS Here, we also add the month effects.

```
train <- train %>% mutate(m = as.numeric(month(as_datetime(timestamp))))
```

```
test <- test %>% mutate(m = as.numeric(month(as_datetime(timestamp))))

avgs_month <- train %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  left_join(avgs_genre, by='genres') %>%
  left_join(avgs_year, by = 'y') %>%
  group_by(m) %>%
  summarize(bm = mean(rating - mu - bi - bu - bg - by)) # month effects

pred_month <- test %>%
  left_join(avgs_movie, by='movieId') %>%
  left_join(avgs_user, by='userId') %>%
  left_join(avgs_genre, by='genres') %>%
  left_join(avgs_year, by='y') %>%
  left_join(avgs_month, by='m') %>%
  mutate(pred_month = mu + bi + bu + bg + by + bm) %>%
  pull(pred_month)

rmse_month <- RMSE(pred_month, test$rating)

rmse_results <- rmse_results %>% rbind(c("Movie + User + Genre + Year + Month effects",
                                         rmse_month))
rmse_results
```

```
## # A tibble: 6 x 2
##   method                                    RMSE
##   <chr>                                     <chr>
## 1 Average rating                            1.05990428951531
## 2 Movie effects                             0.9437429136878
## 3 Movie + User effects                      0.865931962476235
## 4 Movie + User + Genre effects              0.865594068044791
## 5 Movie + User + Genre + Year effects       0.86541892942611
## 6 Movie + User + Genre + Year + Month effects 0.865409561361251
```

The RMSE is 0.86541. This decreases the RMSE, albeit marginally.

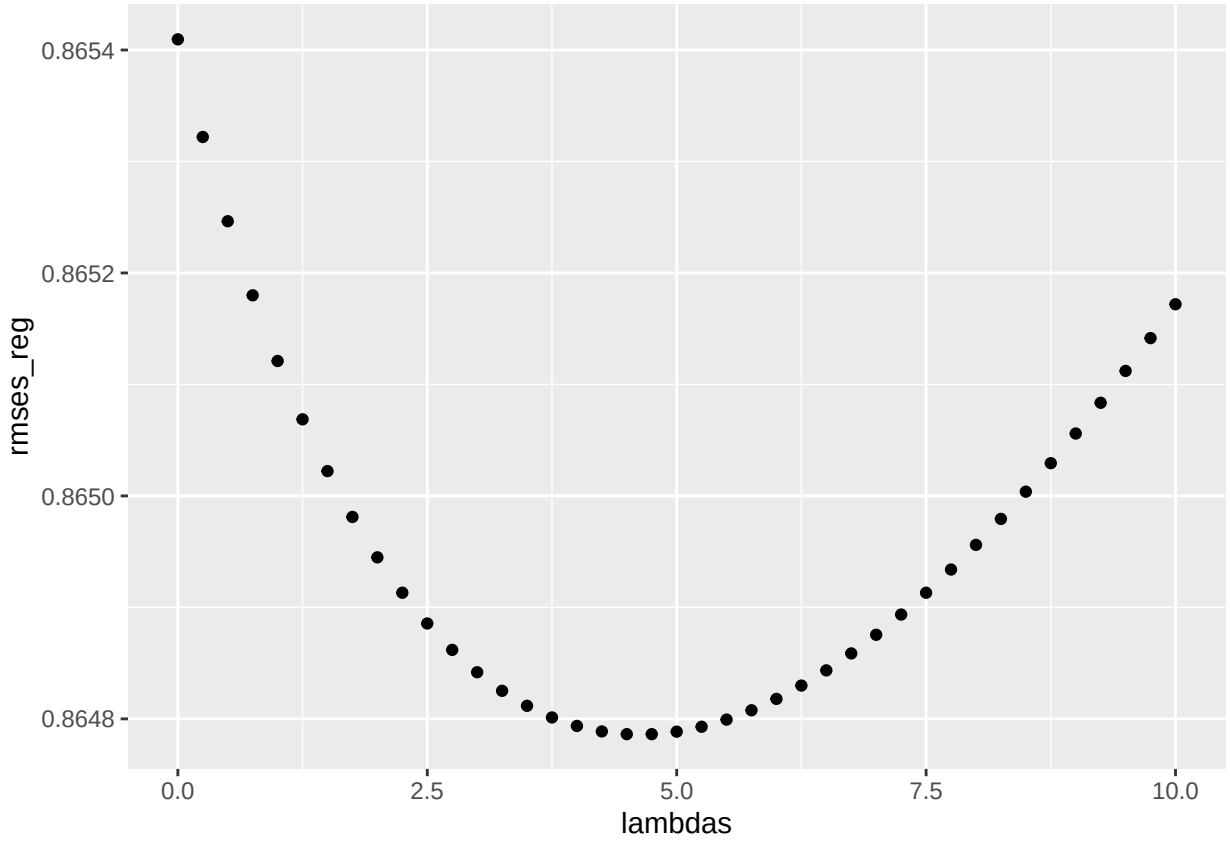
Fifth Model: REGULARIZATION

Here, the regularization method is used, by which a penalty term in the error function is added in order to improve the predictions of movies or users with very few ratings.

First, the penalty term is chosen by Cross Validation.

```
## Choosing lambda by CV

lambdas <- seq(0, 10, 0.25)

rmses_reg <- sapply(lambdas, function(lambda){
  bi <- train %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+lambda))
  bu <- train %>%
    left_join(bi, by="movieId") %>%
```

```
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(n()+lambda))
  bg <- train %>%
    left_join(bi, by="movieId") %>%
    left_join(bu, by="userId") %>%
    group_by(genres) %>%
    summarize(bg = sum(rating - mu - bi - bu)/(n()+lambda))
  by <- train %>%
    left_join(bi, by="movieId") %>%
    left_join(bu, by="userId") %>%
    left_join(bg, by="genres") %>%
    group_by(y) %>%
    summarize(by = sum(rating - mu - bi - bu - bg)/(n()+lambda))
  bm <- train %>%
    left_join(bi, by="movieId") %>%
    left_join(bu, by="userId") %>%
    left_join(bg, by="genres") %>%
    left_join(by, by="y") %>%
    group_by(m) %>%
    summarize(bm = sum(rating - mu - bi - bu - bg - by)/(n()+lambda))

  predicted_ratings <-
    test %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    left_join(bg, by = "genres") %>%
    left_join(by, by = "y") %>%
    left_join(bm, by = "m") %>%
    mutate(pred = mu + bi + bu + bg + by + bm) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test$rating))
})

qplot(lambdas, rmses_reg)
```




```
lambda <- lambdas[which.min(rmses_reg)]
lambda
```

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

The lambda parameter used in the penalty term that most reduces the RMSE is 4.75.

```
## Predictions using regularization

avgs_moviereg <- train %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(n()+lambda))

avgs_userreg <- train %>%
  left_join(avgs_moviereg, by="movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(n()+lambda))

avgs_genrereg <- train %>%
  left_join(avgs_moviereg, by="movieId") %>%
  left_join(avgs_userreg, by="userId") %>%
  group_by(genres) %>%
  summarize(bg = sum(rating - mu - bi - bu)/(n()+lambda))

avgs_yearreg <- train %>%
```

```
  left_join(avgs_moviereg, by='movieId') %>%
  left_join(avgs_userreg, by='userId') %>%
  left_join(avgs_genrereg, by='genres') %>%
  group_by(y) %>%
  summarize(by = sum(rating - mu - bi - bu - bg)/(n()+lambda))

avgs_monthreg <- train %>%
  left_join(avgs_moviereg, by='movieId') %>%
  left_join(avgs_userreg, by='userId') %>%
  left_join(avgs_genrereg, by='genres') %>%
  left_join(avgs_yearreg, by = 'y') %>%
  group_by(m) %>%
  summarize(bm = sum(rating - mu - bi - bu - bg - by)/(n()+lambda)) # month effects

pred_reg <- test %>%
  left_join(avgs_moviereg, by = "movieId") %>%
  left_join(avgs_userreg, by = "userId") %>%
  left_join(avgs_genrereg, by = "genres") %>%
  left_join(avgs_yearreg, by = "y") %>%
  left_join(avgs_monthreg, by = "m") %>%
  mutate(pred = mu + bi + bu + bg + by + bm) %>%
  pull(pred)

rmse_reg <- RMSE(pred_reg, test$rating)

rmse_results <- rmse_results %>% rbind(c("Regularization",
                                         rmse_reg))
rmse_results
```

```
## # A tibble: 7 x 2
##   method                                    RMSE
##   <chr>                                     <chr>
## 1 Average rating                            1.05990428951531
## 2 Movie effects                             0.9437429136878
## 3 Movie + User effects                      0.865931962476235
## 4 Movie + User + Genre effects              0.865594068044791
## 5 Movie + User + Genre + Year effects       0.86541892942611
## 6 Movie + User + Genre + Year + Month effects 0.865409561361251
## 7 Regularization                            0.864786284324217
```

With this lambda, the RMSE now becomes 0.8648; reducing the RMSE by 0.07% with respect to the previous model.

Final Adjustment

Finally, we will adjust the predictions for them to be not less than zero (the minimum value possible) nor greater than 5 (the maximum value possible).

```
min(pred_reg)
```

```
## [1] -0.5117
```

```
max(pred_reg)
```

```
## [1] 6.02
```

```
max(train$rating)
```

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

```
min(train$rating)
```

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

```
pred_maxmin <- test %>%
  left_join(avgs_moviereg, by = "movieId") %>%
  left_join(avgs_userreg, by = "userId") %>%
  left_join(avgs_genrereg, by = "genres") %>%
  left_join(avgs_yearreg, by = "y") %>%
  left_join(avgs_monthreg, by = "m") %>%
  mutate(pred = mu + bi + bu + bg + by + bm) %>%
  mutate(pred_maxmin = ifelse(pred>5,5,ifelse(pred<0.5,0.5,pred))) %>%
  pull(pred_maxmin)

rmse_maxmin <- RMSE(pred_maxmin, test$rating)

rmse_results <- rmse_results %>% rbind(c("Adjusting max and min values",
                                         rmse_maxmin))
rmse_results
```

```
## # A tibble: 8 x 2
##   method                                    RMSE
##   <chr>                                     <chr>
## 1 Average rating                            1.05990428951531
## 2 Movie effects                             0.9437429136878
## 3 Movie + User effects                      0.865931962476235
## 4 Movie + User + Genre effects              0.865594068044791
## 5 Movie + User + Genre + Year effects       0.86541892942611
## 6 Movie + User + Genre + Year + Month effects 0.865409561361251
## 7 Regularization                            0.864786284324217
## 8 Adjusting max and min values              0.86466387384128
```

Now, the RMSE is 0.8647, the lowest we got with between all the models.

Results

Evaluation using the validation set
In this section, we will use the validation set to evaluate our algorithm and end up with the final RMSE for this project.

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

edx <- edx %>% mutate(month = as.numeric(month(as_datetime(timestamp))))
validation <- validation %>% mutate(month = as.numeric(month(as_datetime(timestamp))))

edx_avgs_moviereg <- edx %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(n()+lambda))

edx_avgs_userreg <- edx %>%
  left_join(edx_avgs_moviereg, by="movieId") %>%
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(n()+lambda))

edx_avgs_genrereg <- edx %>%
  left_join(edx_avgs_moviereg, by="movieId") %>%
  left_join(edx_avgs_userreg, by="userId") %>%
  group_by(genres) %>%
  summarize(bg = sum(rating - mu - bi - bu)/(n()+lambda))

edx_avgs_yearreg <- edx %>%
  left_join(edx_avgs_moviereg, by="movieId") %>%
  left_join(edx_avgs_userreg, by="userId") %>%
  left_join(edx_avgs_genrereg, by="genres") %>%
  group_by(year) %>%
  summarize(by = sum(rating - mu - bi - bu - bg)/(n()+lambda))

edx_avgs_monthreg <- edx %>%
  left_join(edx_avgs_moviereg, by="movieId") %>%
  left_join(edx_avgs_userreg, by="userId") %>%
  left_join(edx_avgs_genrereg, by="genres") %>%
  left_join(edx_avgs_yearreg, by="year") %>%
  group_by(month) %>%
  summarize(bm = sum(rating - mu - bi - bu - by)/(n()+lambda))

edx_pred <- validation %>%
  left_join(edx_avgs_moviereg, by = "movieId") %>%
  left_join(edx_avgs_userreg, by = "userId") %>%
  left_join(edx_avgs_genrereg, by = "genres") %>%
  left_join(edx_avgs_yearreg, by = "year") %>%
  left_join(edx_avgs_monthreg, by = "month") %>%
  mutate(pred = mu + bi + bu + bg + by + bm) %>%
  mutate(pred_maxmin = ifelse(pred>5,5,ifelse(pred<0.5,0.5,pred))) %>%
  pull(pred_maxmin)

RMSE(edx_pred, validation$rating)
```

```
## [1] 0.8642
```

The final RMSE is approximately equal to 0.8642.

Conclusion

In this project, a movie recommendation system based on the Netflix challenge for ratings prediction was developed. A version of the movielens database with 10M observations was used. Machine learning algorithms were used to generate predictions for movie ratings. The Root Mean Squared Error (RMSE) was used to assess the quality of the predictions.

The edx dataset was divided in two: the train set and the test set.

This project started by using the mean ratings of the train set as a predictor for the ratings in the test set. The RMSE obtained with this first model was 1.06. This ment that, on average, average ratings missed the correct rating by 1 point, which was rather high.

To reduce the RMSE (i.e. to improve the predictions), the movie, user, genre and time effects were added. To improve the results, regularization was used. Regularization consisted in constraining the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. Finally, the predictions were adjusted to not include predictions below 0 nor above 5.

After choosing the best performing model/algorithm, it was applied to the entire edx dataset. Once the predictions were made, the validation dataset was used to obtain the final RMSE of the project. The final RMSE obtained is 0.8642 for the ratings in the validation set.