Movie Recommendation System

ggh725

Introduction

The aim of this project is to create a movie recommendation system using the 10M version of the MovieLens dataset. Specifically, the goal is to train a machine learning algorithm which uses the inputs in one subset (i.e. the edx subset) to predict the movie ratings in the validation subset. The accuracy of the algorithm is assessed using the root mean squared error (RMSE).

The edx subset contains 9,000,055 observations and 6 variables, i.e. userId, movieId, rating, timestamp, title, and genres. The validation subset is 10% of the total MovieLens data, and contains 999,999 observations. The code needed to create these data subsets were provided in the online course materials.

For the first part of this project, the features of the edx dataset is explored to determine the possible trends in movie ratings. The characteristics and trends which were observed in the data exploration phase will guide the data analysis section. Transformations which are needed to organize and clean the data are also performed in this first part.

This is followed by the analysis section. The movie recommendation algorithm is based on a model which assumes that movie ratings($\hat{Y}$) is a function of the mean movie ratings(μ), movie effect(b_i), the user effect(b_u), the time of rating(b_t), and movie genre(b_g), to wit:

$$\hat{Y} = \mu + b_i + b_u + b_t + b_g$$

This projects concludes with a summary of its findings, as well as a discussion of its limitations.

Analysis

In this section, the edx subset is explored and analyzed in order to determine the features of the data and to detect possible trends.

First, the edx and validation sets are downloaded and created.

```
#################################
# Create edx set, validation set
#################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse

## -- Attaching packages ---------------------- tidyverse 1.3.0 --

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## Warning: package 'ggplot2' was built under R version 3.6.1

## Warning: package 'tidyr' was built under R version 3.6.1
```

```
## Warning: package 'readr' was built under R version 3.6.1

## Warning: package 'purrr' was built under R version 3.6.1

## Warning: package 'dplyr' was built under R version 3.6.1

## Warning: package 'forcats' was built under R version 3.6.1

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

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

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.6.1

## Loading required package: lattice

##
## Attaching package: 'caret'

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

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

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

## Warning: package 'data.table' was built under R version 3.6.1

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

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

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

```r
# 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(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

```r
# Validation set will be 10% of MovieLens data

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

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

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

# Make sure userId and movieId in validation set are also in edx set

validation <- temp %>%
     semi_join(edx, by = "movieId") %>%
     semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
```

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

```r
edx <- rbind(edx, removed)

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

This is followed by an examination of the edx dataset.

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```r
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
##  Length:9000055     Length:9000055
```

```
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```r
str(edx)
```

```
## '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|Ad
```

The edx subset contains 9000055 observations of 6 variables, i.e. userId, movieId, rating, timestamp, title, and genres. Two variables (userId and timestamp) are encoded as integers, another two (movieId and rating) as numeric, and the last two (title and genres) as characters. In order to be able to properly use the timestamp variable, this must be later converted into the date format.

```r
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736 1390114
```

```r
edx %>% summarize(n = n_distinct(movieId))
```

```
##       n
## 1 10677
```

```r
edx %>% summarize(n = n_distinct(userId))
```

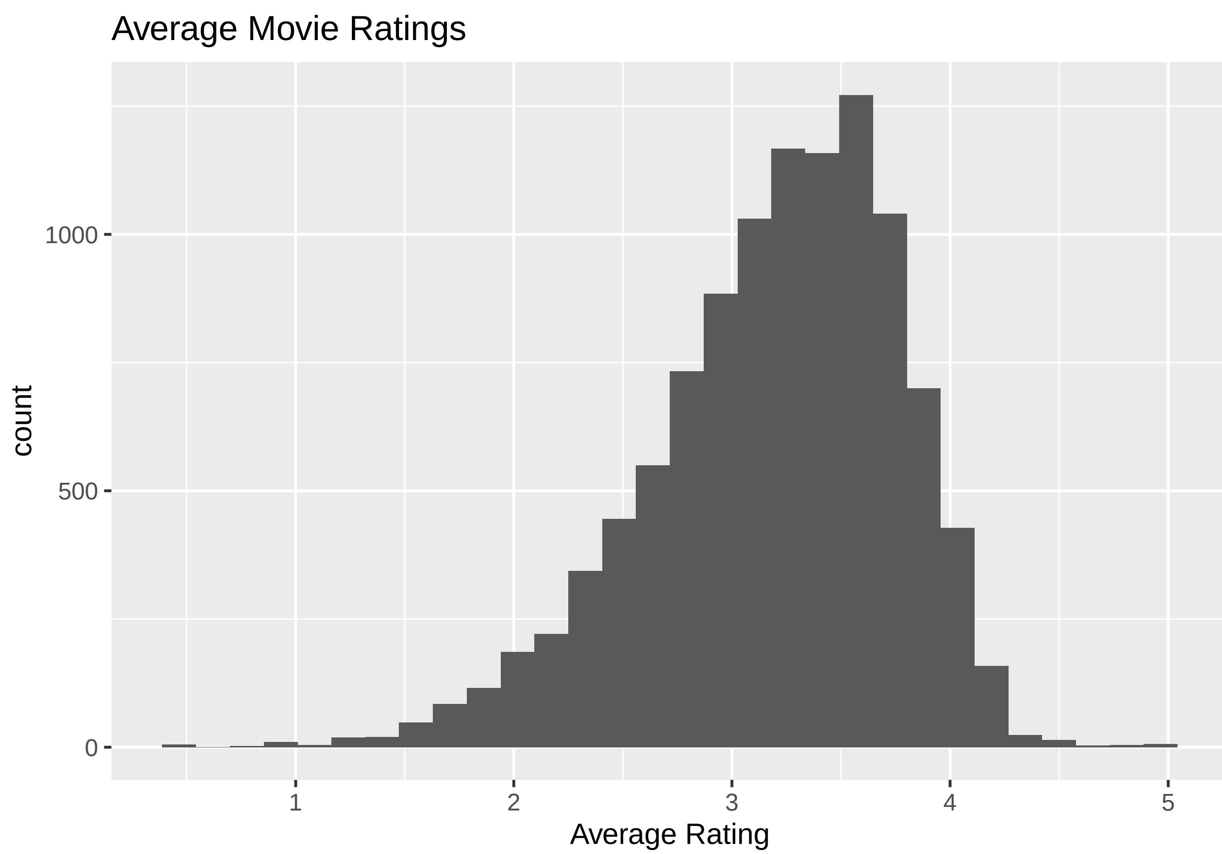
```
##       n
## 1 69878
```

As seen in the summary, there are no missing values. The most common ratings are 4, 3, 5, 3.5, and 2. There are 10,677 movies and 69,878 users in the dataset. Considering that there are 9000055 observations, we can see that not all users submitted a rating for every movie.

The figure below shows that the average rating received by movies likewise varied.

```r
edx %>% group_by(movieId, title) %>%
    summarize(avg_rating = mean(rating)) %>%
    ggplot(aes(avg_rating)) + geom_histogram() +
    labs(title = "Average Movie Ratings", x = "Average Rating")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
edx %>% group_by(movieId, title) %>%
    summarize(n = n()) %>%
    arrange(desc(n)) %>%
    top_n(10)
```

```
## Selecting by n

## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                     n
##      <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 most reviewed movies are Pulp Fiction, Forrest Gump, and The Silence of the Lambs.

However, a number of movies received only 1 rating.

```
edx %>% group_by(movieId, title) %>%
    summarize(n = n()) %>%
    arrange(desc(n)) %>%
    top_n(10)
```

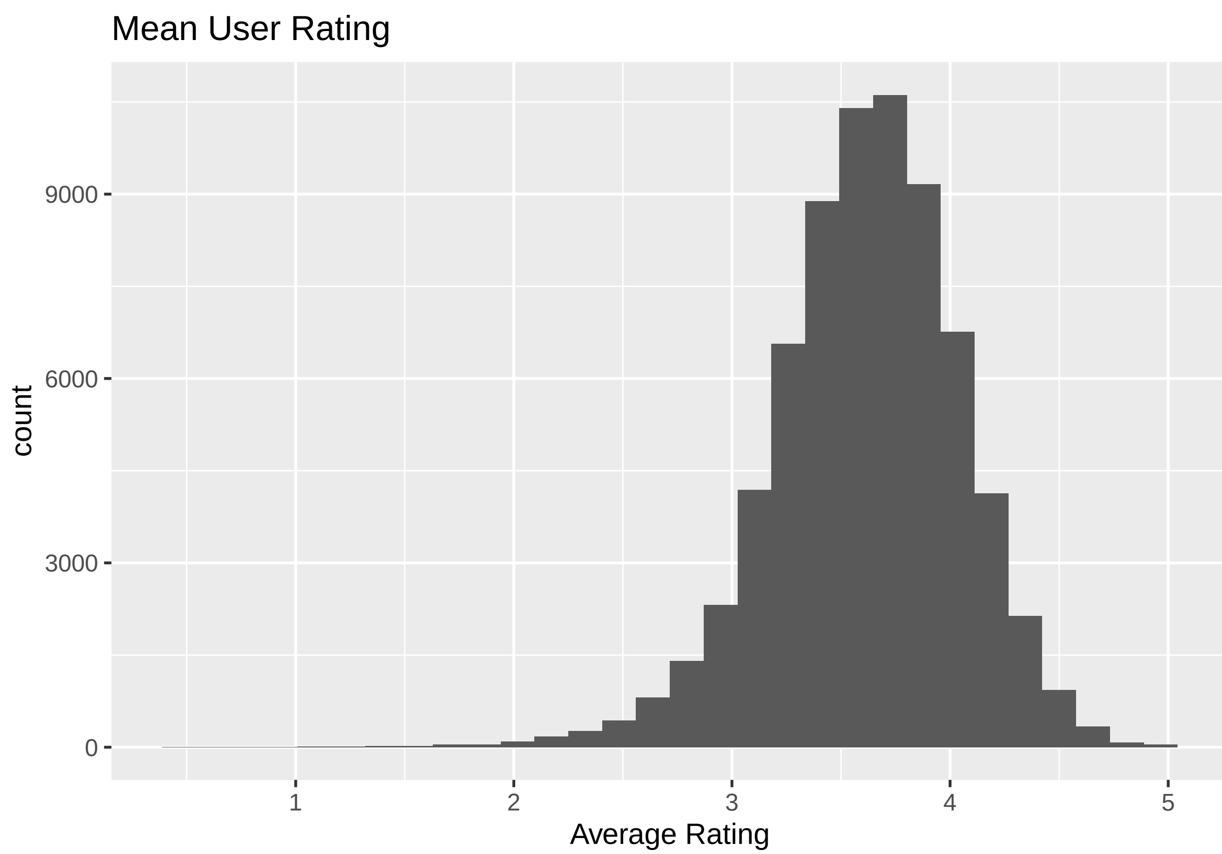
```
## Selecting by n
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                        n
##      <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 average rating per user likewise varied, as shown by the figure below:

```
edx %>% group_by(userId) %>%
    summarise(avg_rating = mean(rating)) %>%
    ggplot(aes(avg_rating)) + geom_histogram() +
    labs(title = "Mean User Rating", x = "Average Rating")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



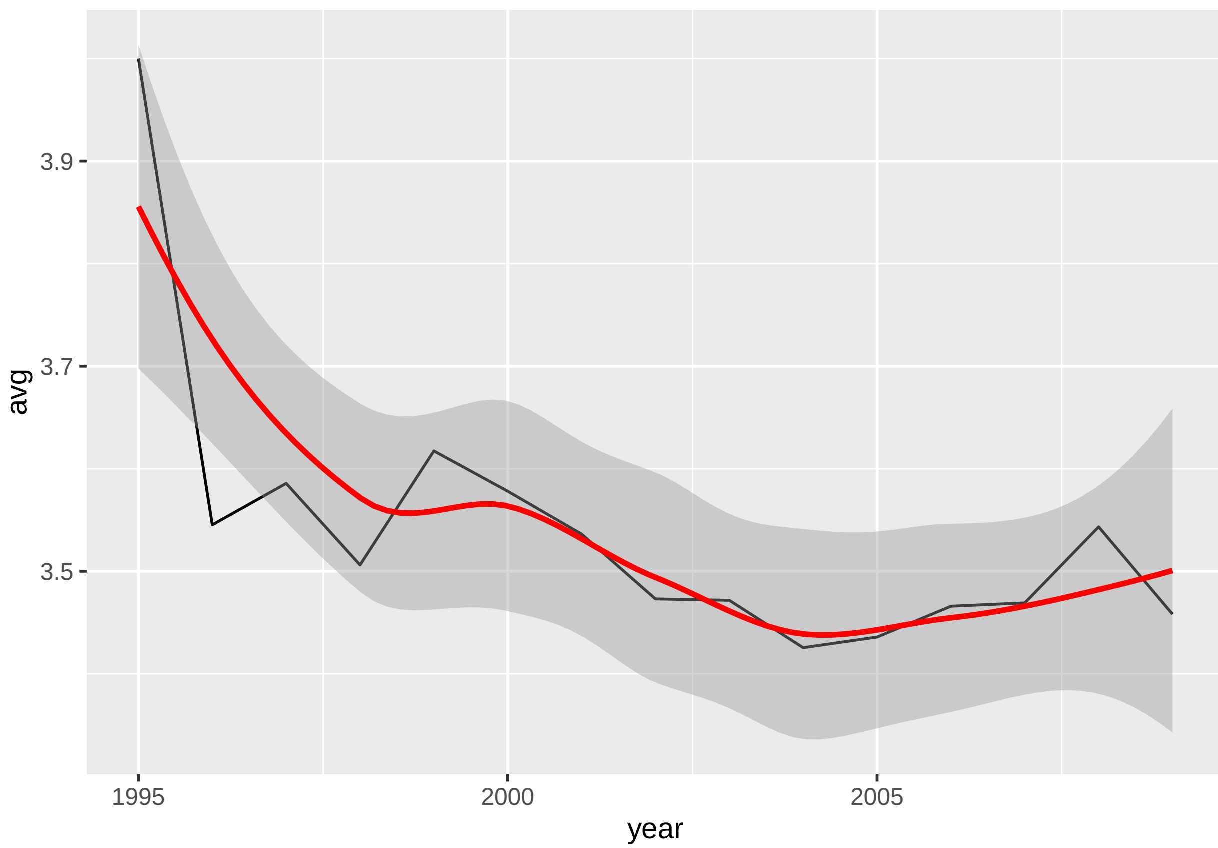
The variable timestamp identifies the date and time when the rating was posted. The first reviews were posted in 1995. Since then, there has been a general downward trend in the annual average of the ratings.

```r
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 3.6.1

##
## 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 object is masked from 'package:base':
##
##     date
```

```r
edx %>% mutate(year = year(as_datetime(timestamp))) %>%
    group_by(year) %>%
    summarize(avg = mean(rating)) %>%
    arrange(year) %>%
    ggplot(aes(year, avg)) +
    geom_line() +
    geom_smooth(color = "red")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Finally, we examine the genres variable.

```
edx %>% group_by(genres) %>%
    summarize(avg = mean(rating)) %>%
    arrange(desc(avg)) %>%
    top_n(10)
```

```
## Selecting by avg
```

```
## # A tibble: 10 x 2
##    genres                                    avg
##    <chr>                                   <dbl>
##  1 Animation|IMAX|Sci-Fi                    4.71
##  2 Drama|Film-Noir|Romance                  4.30
##  3 Action|Crime|Drama|IMAX                  4.30
##  4 Animation|Children|Comedy|Crime          4.28
##  5 Film-Noir|Mystery                        4.24
##  6 Crime|Film-Noir|Mystery                  4.22
##  7 Film-Noir|Romance|Thriller               4.22
##  8 Crime|Film-Noir|Thriller                 4.21
##  9 Crime|Mystery|Thriller                   4.20
## 10 Action|Adventure|Comedy|Fantasy|Romance  4.20
```

```
edx %>% group_by(genres) %>%
    summarize(avg = mean(rating)) %>%
    arrange(avg) %>%
    top_n(-10)
```

```
## Selecting by avg

## # A tibble: 10 x 2
##    genres                                          avg
##    <chr>                                         <dbl>
##  1 Documentary|Horror                             1.45
##  2 Action|Animation|Comedy|Horror                 1.5
##  3 Action|Horror|Mystery|Thriller                 1.61
##  4 Comedy|Film-Noir|Thriller                      1.64
##  5 Action|Drama|Horror|Sci-Fi                     1.75
##  6 Adventure|Drama|Horror|Sci-Fi|Thriller         1.75
##  7 Action|Adventure|Drama|Fantasy|Sci-Fi          1.90
##  8 Action|Children|Comedy                         1.91
##  9 Action|Adventure|Children                      1.92
## 10 Adventure|Animation|Children|Fantasy|Sci-Fi   1.92
```

A number of genres, notably Animation|IMAX|Sci-Fi, receive on average high ratings (4.71). Certain genres, such as Documentary|Horror and Action|Animation|Comedy|Horror, receive on average low ratings (1.45 and 1.5, respectively).

Methods and Results

From the exploratory data analysis, we saw that the data needs to be preprocessed before the movie recommendation algorithm can be trained. Specifically, the timestamp variable needs to be converted from integer into date format. I chose to convert timestamp into years, to allow a more tractable analysis.

```
edx <- edx %>% mutate(year = year(as_datetime(timestamp)))
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 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  $ year     : num  1996 1996 1996 1996 1996 ...
```

```
validation <- validation %>% mutate(year = year(as_datetime(timestamp)))
str(validation)
```

```
## 'data.frame':    999999 obs. of  7 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
##  $ year     : num  1996 1996 1996 1997 1997 ...
```

Next, we need to define the RMSE function which will be used to assess the performance of the algorithm. The RMSE refers to the residual mean squared error of the predicted ratings against the true ratings. The aim is to minimize this RMSE.

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

We can now proceed with developing the recommendation algorithm.

The first, and simplest, model is to predict the same movie rating for all users, with random variation explained by an error term.

$$Y_u, i = \mu + \varepsilon_u, i$$

The rating estimate that would minimize the RMSE is the average of all ratings.

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

```
## [1] 3.512465
```

If mu is used to predict ratings, the RMSE is:

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

```
## [1] 1.061202
```

```
rmse_results <- tibble(Method = "Mean", RMSE = rmse1)
rmse_results
```

```
## # A tibble: 1 x 2
##   Method  RMSE
##   <chr>  <dbl>
## 1 Mean    1.06
```

We saw earlier that some movies are rated highly (3.5 to 5), while others are given low ratings (0.5 to 3). To account for this, the algorithm will add a term b_i that represents the average ranking of movies:

$$Y_u, i = \mu + b_i + \varepsilon_u, i$$

```
movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))

prediction2 <- mu + validation %>%
    left_join(movie_avgs, by = "movieId") %>%
    pull(b_i)
```

The RMSE of this Movie Effects model is lower:

```
rmse2 <- RMSE(validation$rating, prediction2)
rmse2
```

```
## [1] 0.9439087
```

```
rmse_results <- bind_rows(rmse_results, tibble(Method = "Movie Effects", RMSE = rmse2))
rmse_results
```

```
## # A tibble: 2 x 2
##   Method          RMSE
##   <chr>          <dbl>
## 1 Mean           1.06
## 2 Movie Effects 0.944
```

The so-called User Effects should also be accounted for. Specifically, we saw earlier that the mean ratings varied per user. In light of this, the algorithm will add a term b_u that represents the average rating per user:

$$Y_u, i = \mu + b_i + b_u + \varepsilon_u, i$$

```
user_avgs <- edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))

prediction3 <- validation %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    mutate(predict = mu + b_i + b_u) %>%
    pull(predict)
```

The RMSE for this model, which includes User Effects, is even lower than the previous one:

```
rmse3 <- RMSE(validation$rating, prediction3)
rmse3
```

```
## [1] 0.8653488
```

```
rmse_results <- bind_rows(rmse_results, tibble(Method = "Movie and User Effects", RMSE = rmse3))
rmse_results
```

```
## # A tibble: 3 x 2
##   Method                  RMSE
##   <chr>                  <dbl>
## 1 Mean                   1.06
## 2 Movie Effects          0.944
## 3 Movie and User Effects 0.865
```

The third variable that should be accounted for is time. We earlier saw that since 1995, average ratings have been decreasing. To account for this, the algorithm will also include a term b_t to represent the effects of time when the rating was made:

$$Y_u, i = \mu + b_i + b_u + b_t + \varepsilon_u, i$$

```
time_effects <- edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    group_by(year) %>%
    summarize(b_t = mean(rating - mu - b_i - b_u))

prediction4 <- validation %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(time_effects, by = "year") %>%
    mutate(predict = mu + b_i + b_u + b_t) %>%
    pull(predict)
```

The RMSE for the model which includes Time Effects is marginally better:

```
rmse4 <- RMSE(validation$rating, prediction4)
rmse4
```

```
## [1] 0.8653369
```

```r
rmse_results <- bind_rows(rmse_results, tibble(Method = "With Time Effects", RMSE = rmse4))
rmse_results
```

```
## # A tibble: 4 x 2
##   Method                  RMSE
##   <chr>                  <dbl>
## 1 Mean                   1.06
## 2 Movie Effects          0.944
## 3 Movie and User Effects 0.865
## 4 With Time Effects      0.865
```

We saw earlier that some genres received, on average, higher ratings than others. As such, this variable should also be accounted for in the algorithm. To do so, we add b_g to capture this genre effect:

$$Y_u, i = \mu + b_i + b_u + b_t + b_g + \varepsilon_u, i$$

```r
genre_effects <- edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(time_effects, by = "year") %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu - b_i - b_u - b_t))

prediction5 <- validation %>%
    mutate(year = year(as_datetime(timestamp))) %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(time_effects, by = "year") %>%
    left_join(genre_effects, by = "genres") %>%
    mutate(predict = mu + b_i + b_u + b_t + b_g) %>%
    pull(predict)
```

This addition further improves the RMSE:

```r
rmse5 <- RMSE(validation$rating, prediction5)
rmse5
```

```
## [1] 0.8649347
```

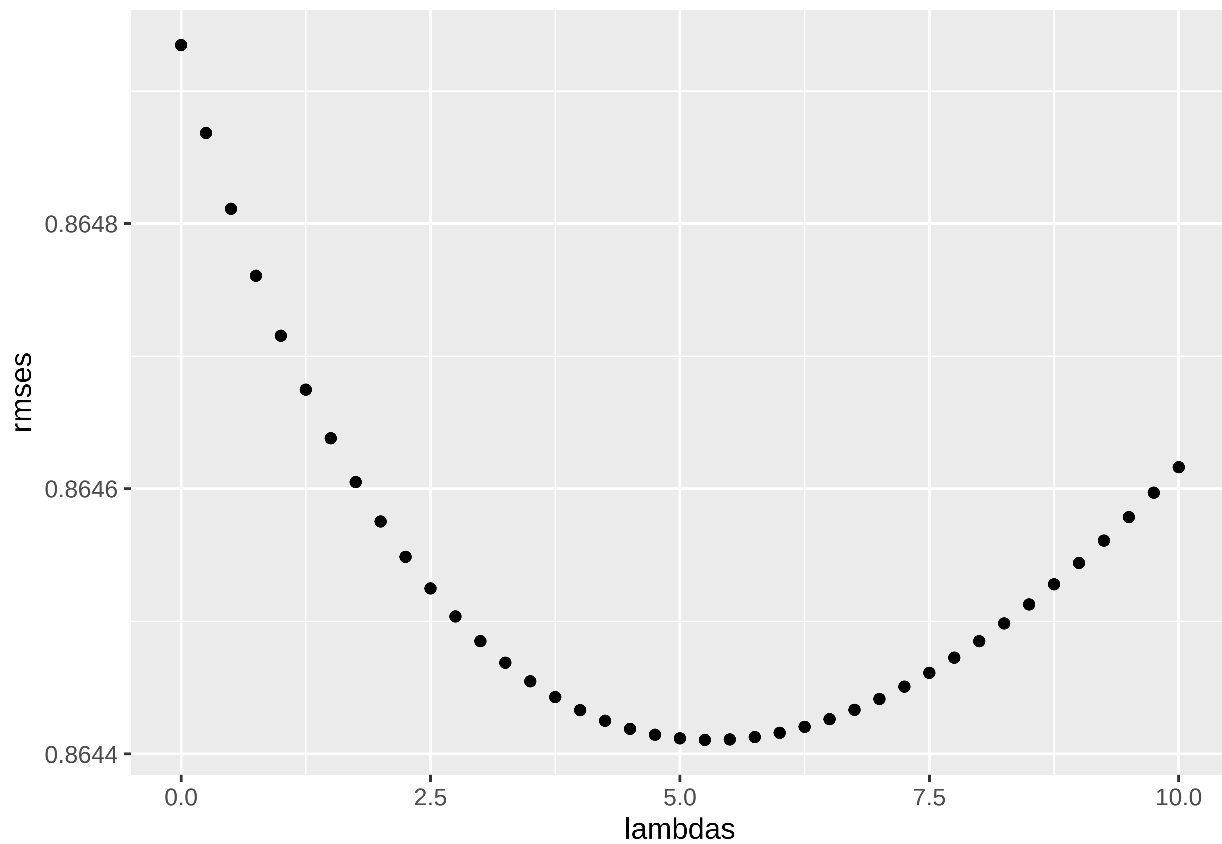
```r
rmse_results <- bind_rows(rmse_results, tibble(Method = "With Genre Effects", RMSE = rmse5))
rmse_results
```

```
## # A tibble: 5 x 2
##   Method                  RMSE
##   <chr>                  <dbl>
## 1 Mean                   1.06
## 2 Movie Effects          0.944
## 3 Movie and User Effects 0.865
## 4 With Time Effects      0.865
## 5 With Genre Effects     0.865
```

The algorithm can be further improved by regularization. Regularization would prevent 2 things from unnecessarily influencing the model, namely: (i) movies with only a few ratings and (ii) users with only a handful of ratings.

First, we determine the appropriate penalty term.

```r
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, 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 - mu - b_i)/(n() + l))

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

    b_g <- edx %>%
        mutate(year = year(as_datetime(timestamp))) %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_t, by = "year") %>%
        group_by(genres) %>%
        summarize(b_g = sum(rating - mu - b_i - b_u - b_t)/(n() + l))

    predicted_ratings <- validation %>%
        mutate(year = year(as_datetime(timestamp))) %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_t, by = "year") %>%
        left_join(b_g, by = "genres") %>%
        mutate(prediction = mu + b_i + b_u + b_t + b_g) %>%
        pull(prediction)

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

qplot(lambdas, rmses)
```



The lambda is thus:

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

```
## [1] 5.25
```

We next see how the algorithm performs with regularized estimates:

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

prediction6 <- validation %>%
    left_join(movie_reg, by = "movieId") %>%
    mutate(prediction = mu + b_i) %>%
    pull(prediction)
```

```
rmse6 <- RMSE(validation$rating, prediction6)
rmse6
```

```
## [1] 0.9438805
```

```
rmse_results <- bind_rows(rmse_results, tibble(Method = "Regularized Movie Effects", RMSE = rmse6))
rmse_results
```

```
## # A tibble: 6 x 2
##   Method                        RMSE
##   <chr>                        <dbl>
```

```
## 1 Mean                      1.06
## 2 Movie Effects             0.944
## 3 Movie and User Effects    0.865
## 4 With Time Effects         0.865
## 5 With Genre Effects        0.865
## 6 Regularized Movie Effects 0.944
```

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

prediction7 <- validation %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    mutate(predict = mu + b_i + b_u) %>%
    pull(predict)
```

```r
rmse7 <- RMSE(validation$rating, prediction7)
rmse7
```

```
## [1] 0.864817
```

```r
rmse_results <- bind_rows(rmse_results, tibble(Method = "With Regularized User Effects", RMSE = rmse7))
rmse_results
```

```
## # A tibble: 7 x 2
##   Method                         RMSE
##   <chr>                         <dbl>
## 1 Mean                          1.06
## 2 Movie Effects                 0.944
## 3 Movie and User Effects        0.865
## 4 With Time Effects             0.865
## 5 With Genre Effects            0.865
## 6 Regularized Movie Effects     0.944
## 7 With Regularized User Effects 0.865
```

```r
time_reg <- edx %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    group_by(year) %>%
    summarize(b_t = sum(rating - mu - b_i - b_u)/(n() + lambda))

prediction8 <- validation %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    left_join(time_reg, by = "year") %>%
    mutate(predict = mu + b_i + b_u + b_t) %>%
    pull(predict)
```

```r
rmse8 <- RMSE(validation$rating, prediction8)
rmse8
```

```
## [1] 0.8647958
```

```r
rmse_results <- bind_rows(rmse_results, tibble(Method = "With Regularized Time Effects", RMSE = rmse8))
rmse_results
```

```
## # A tibble: 8 x 2
##   Method                         RMSE
##   <chr>                         <dbl>
## 1 Mean                          1.06
## 2 Movie Effects                 0.944
## 3 Movie and User Effects        0.865
## 4 With Time Effects             0.865
## 5 With Genre Effects            0.865
## 6 Regularized Movie Effects     0.944
## 7 With Regularized User Effects 0.865
## 8 With Regularized Time Effects 0.865
```

```
genre_reg <- edx %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    left_join(time_reg, by = "year") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_t)/(n() + lambda))

prediction9 <- validation %>%
    left_join(movie_reg, by = "movieId") %>%
    left_join(user_reg, by = "userId") %>%
    left_join(time_reg, by = "year") %>%
    left_join(genre_reg, by = "genres") %>%
    mutate(predict = mu + b_i + b_u + b_t + b_g) %>%
    pull(predict)
```

```
rmse9 <- RMSE(validation$rating, prediction9)
rmse9
```

```
## [1] 0.8644106
```

```
rmse_results <- bind_rows(rmse_results, tibble(Method = "With Regularized Genre Effects", RMSE = rmse9))
rmse_results
```

```
## # A tibble: 9 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Mean                           1.06
## 2 Movie Effects                  0.944
## 3 Movie and User Effects         0.865
## 4 With Time Effects              0.865
## 5 With Genre Effects             0.865
## 6 Regularized Movie Effects      0.944
## 7 With Regularized User Effects  0.865
## 8 With Regularized Time Effects  0.865
## 9 With Regularized Genre Effects 0.864
```

We see that the best RMSE is obtained with this regularized model:

$$Y_u, i = \mu + b_i + b_u + b_t + b_g + \varepsilon_u, i$$

Conclusion

This project aimed to train an movie recommendation algorithm which minimizes the loss as measured by the RMSE. This project used the MovieLens dataset to train and validate the algorithm.

We found that the regularized model which accounts for the movie effects, user effects, time, and genre is the optimal model (RMSE of 0.8644106). This model is however limited by the constraints of the given dataset.

Improvements to this model can be obtained by considering other factors, such as: the year of movie release; the difference between the year of release and the year of review; and a more specific identification of genres. The exploration of other machine learning models may also bring improvements to this model.