Capstone project - MovieLens

Eli F Mayost

December 17, 2019

Introduction

In this project, we will develop a recommendation model for movies.

Recommendation models are useful, to companies that sell products and wish to maximize their benefits by recommending similar items based on other user's buys.

It is also useful to their customers to be able to discover what like minded people consume.

A concrete example, is the recommendations a user get for books on Amazon, or movies on Netflix.

Aim

The target of this project is to use regression to minimize RMSE. Both training (edx here) and test (validation) sets are provided.

We will rate movies from 0.5 to 5, with steps of 0.5, keeping in mind that we are asked to minimize the RMSE [1], between the predicted value and the observed value, to be lower than, or equal to, 0.8775 (the lower the better).

[1] RMSE. Root Mean Square Error. Can simply be written as sqrt((prediction - observation)^2). We will apply it to vectors using the RMSE function in the caret package.

The data

The MovieLens 10 million dataset is automatically downloaded and split into a training subset, called edx, and a validation subset called validation in a variables carying these names.

The following code to achieve these partitions is given by the course's staff:

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

# 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")

# Validation set will be 10% of MovieLens data

set.seed(1)

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)
```

Data wrangling

Checking that we do not have missing values in both sets:

```
edx %>% is.na() %>% any()
```

```
## [1] FALSE
```

```
validation %>% is.na() %>% any()
```

```
## [1] FALSE
```

Verifying that the ratings are indeed between 0.5 and 5 with 0.5 steps:

```
edx %>% count(rating, name = "total")
```

```
## # A tibble: 10 x 2
##    rating   total
##     <dbl>   <int>
##  1    0.5   85420
##  2    1    345935
##  3    1.5  106379
##  4    2    710998
##  5    2.5  332783
##  6    3   2121638
##  7    3.5  792037
##  8    4   2588021
##  9    4.5  526309
## 10    5   1390541
```

```
validation %>% count(rating, name = "total")
```

```
## # A tibble: 10 x 2
##    rating  total
##     <dbl>  <int>
##  1    0.5   9568
##  2    1    38245
##  3    1.5  11899
##  4    2    79308
##  5    2.5  37395
##  6    3   235038
##  7    3.5  87727
##  8    4   287829
##  9    4.5  58713
## 10    5   154271
```

Both datasets have 6 features [2] with the following number of observations:

```
edx %>% dim()
```

```
## [1] 9000061       6
```

[2] A unique user ID (userId).
A unique movie ID (movieId).
A rating (0.5 to 5 with 0.5 step).
A timestamp (Unix epoch).
A title (includes film's year).
A list of genres (separated by a pipe).

```
validation %>% dim()
```

```
## [1] 999993      6
```

Data analysis (edx set)

The first 10 observations:

```
edx %>% head(10) %>% kable()
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical

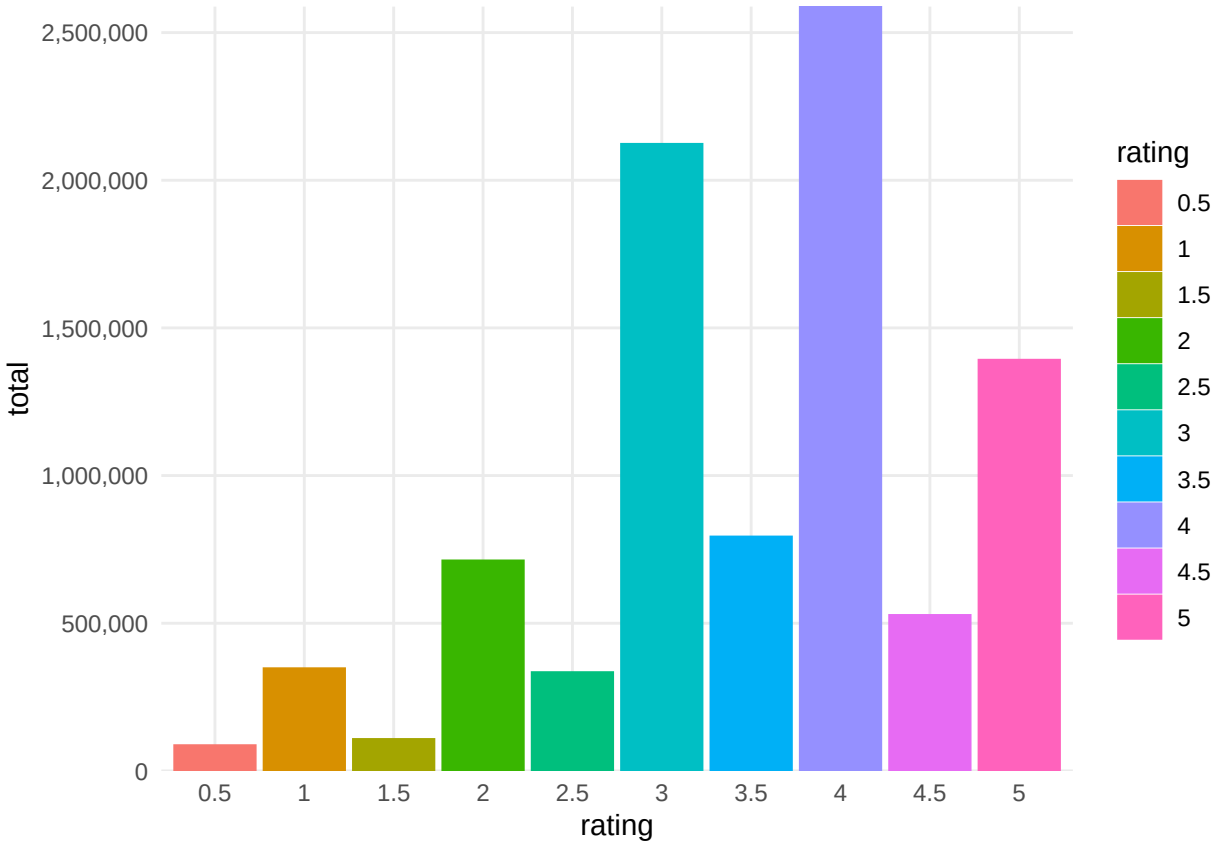
The number of unique users, and unique movies:

```
tibble(users = length(unique(edx$userId)), movies = length(unique(edx$movieId))) %>%
    kable()
```

users	movies
69878	10677

Rating distribution:

```
edx %>%
    mutate(rating = as.factor(rating)) %>%
    ggplot(aes(x = rating, color = rating, fill = rating)) +
        geom_bar() +
        scale_y_discrete(name = "total", labels = scales::comma, limits = seq(0, 3000000, 500000)) +
        theme_minimal()
```



The rating distributon is right skewed with ratings 3, 4 and 5 used more often.

Genres distribution:

Tydyverse, separate_rows was very slow, therefore, I have used linux sed, awk and tr to split the genres (about 20 seconds).

If we were to do it with tidyverse:

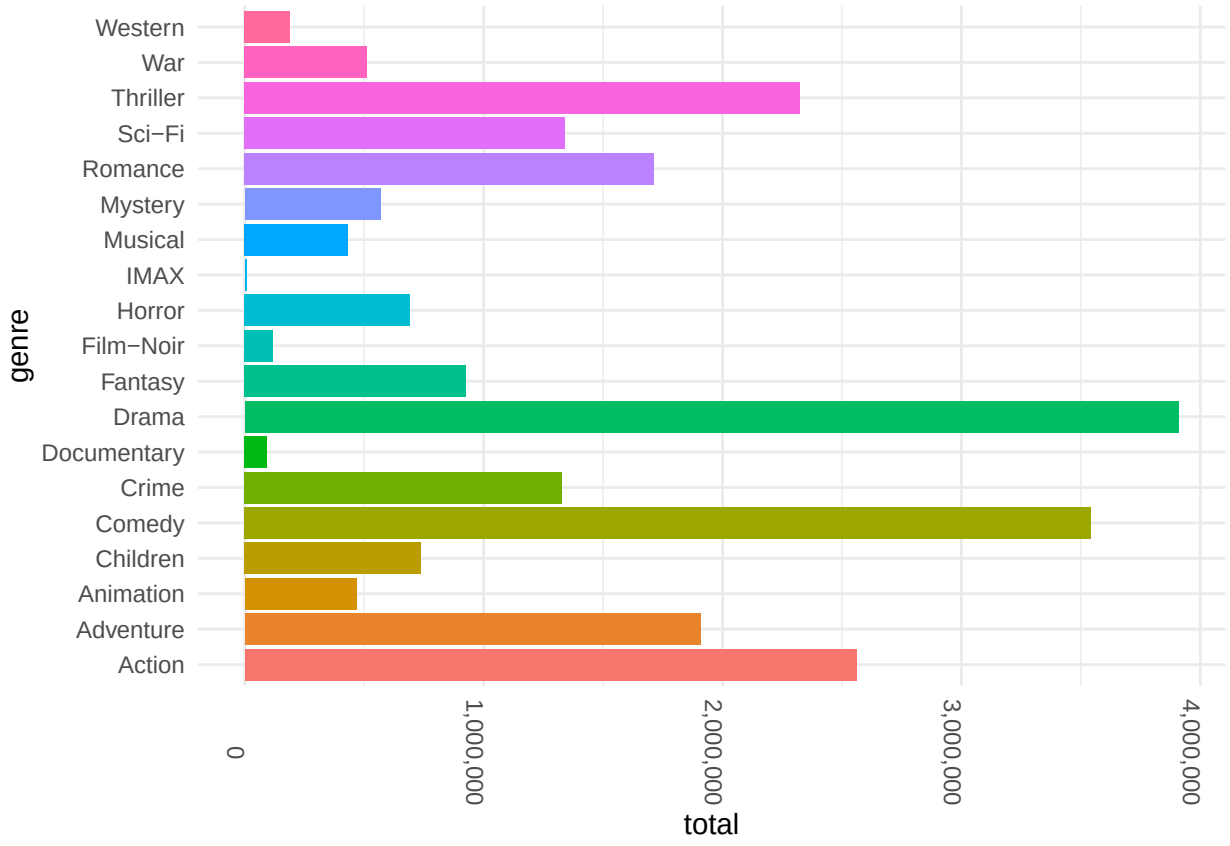
```
edx %>% select(genres) %>% vroom_write(path = "genres.csv", delim = ",")

genres_df <- read.table(pipe("sed '1d' genres.csv | tr '|' '\n' | sort | uniq -c | awk '{OFS=\",\"; pri
                        sep = ",",
                        col.names = c("genre", "total"))

genres_df %>% kable()
```

genre	total
Action	2560649
Adventure	1908692
Animation	467220
Children	737851
Comedy	3541284
Crime	1326917
Documentary	93252
Drama	3909401
Fantasy	925624

genre	total
Film-Noir	118394
Horror	691407
IMAX	8190
Musical	432960
Mystery	567865
(no	6
Romance	1712232
Sci-Fi	1341750
Thriller	2325349
War	511330
Western	189234

We remove the rows without genre, and graph the genres distribution:

```
genres_df %>%
  filter(! str_detect(genre, "\\(no")) %>%
  ggplot(aes(x = genre, fill = genre)) +
    geom_bar(aes(y = total), stat = "identity") +
    scale_x_discrete(name = "genre") +
    scale_y_continuous(name = "total", labels = scales::comma) +
    coord_flip() +
    theme_minimal() +
    theme(legend.position = "none", axis.text.x = element_text(angle = -90))
```



Number of ratings per year, adding a column with the year the movie was rated:

```
edx <- edx %>%
    mutate(year_rated = lubridate::year(as.Date(as.POSIXct(timestamp, origin="1970-01-01"))))

edx %>% count(year_rated, name = "total")
```

```
## # A tibble: 15 x 2
##    year_rated   total
##         <dbl>   <int>
##  1       1995       2
##  2       1996  942976
##  3       1997  414218
##  4       1998  181845
##  5       1999  709978
##  6       2000 1144666
##  7       2001  683412
##  8       2002  524826
##  9       2003  619707
## 10       2004  691191
## 11       2005 1059807
## 12       2006  689447
## 13       2007  628845
## 14       2008  696027
## 15       2009   13114
```

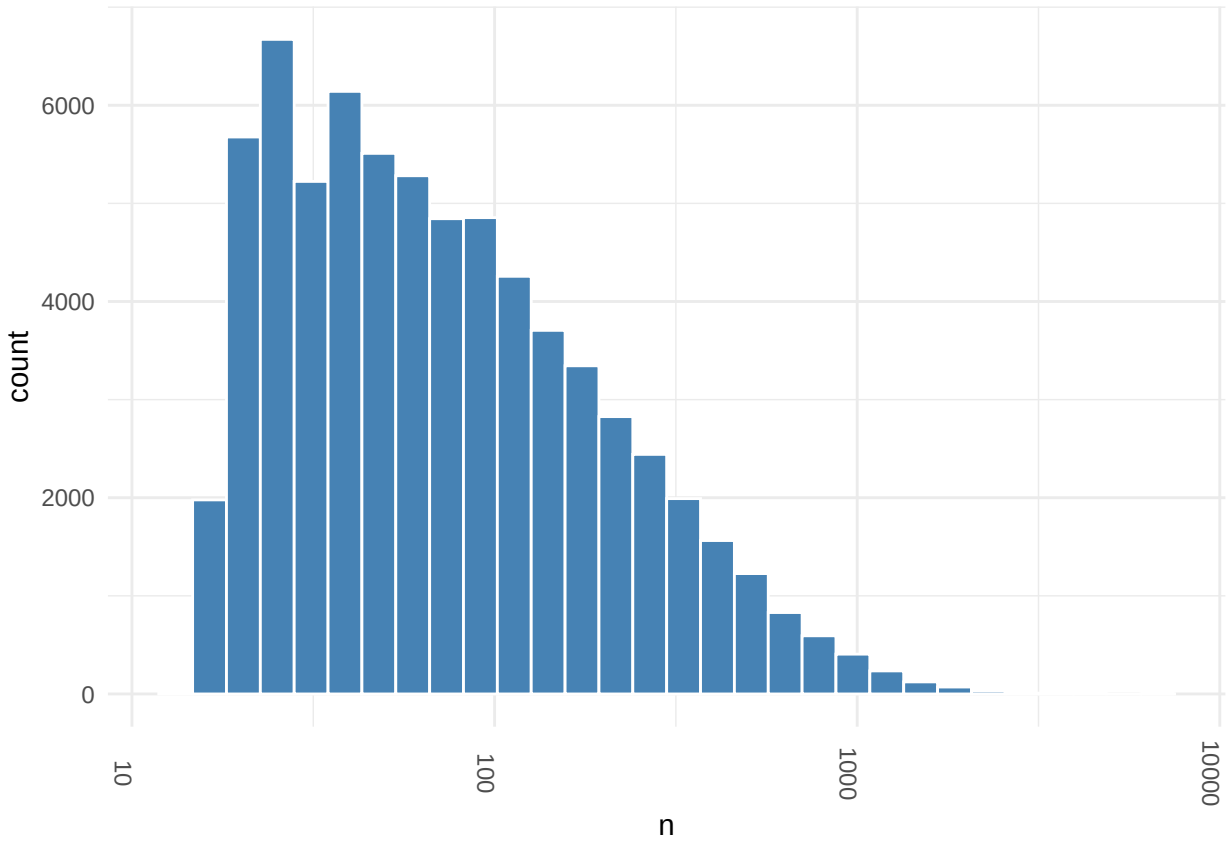
As 1995 has only two ratings, we can remove 1995:

```
edx <- edx %>% filter(! year_rated == 1995)

edx %>% count(year_rated, name = "total")
```

```
## # A tibble: 14 x 2
##    year_rated   total
##         <dbl>   <int>
##  1       1996  942976
##  2       1997  414218
##  3       1998  181845
##  4       1999  709978
##  5       2000 1144666
##  6       2001  683412
##  7       2002  524826
##  8       2003  619707
##  9       2004  691191
## 10       2005 1059807
## 11       2006  689447
## 12       2007  628845
## 13       2008  696027
## 14       2009   13114
```

Let's see the distribution of total of rating per user:

```
edx %>% count(userId) %>%
  ggplot(aes(n)) +
    geom_histogram(bins = 30, color = "white", fill = "steelblue") +
    scale_x_log10() +
    theme_minimal() +
    theme(legend.position = "none", axis.text.x = element_text(angle = -90))
```

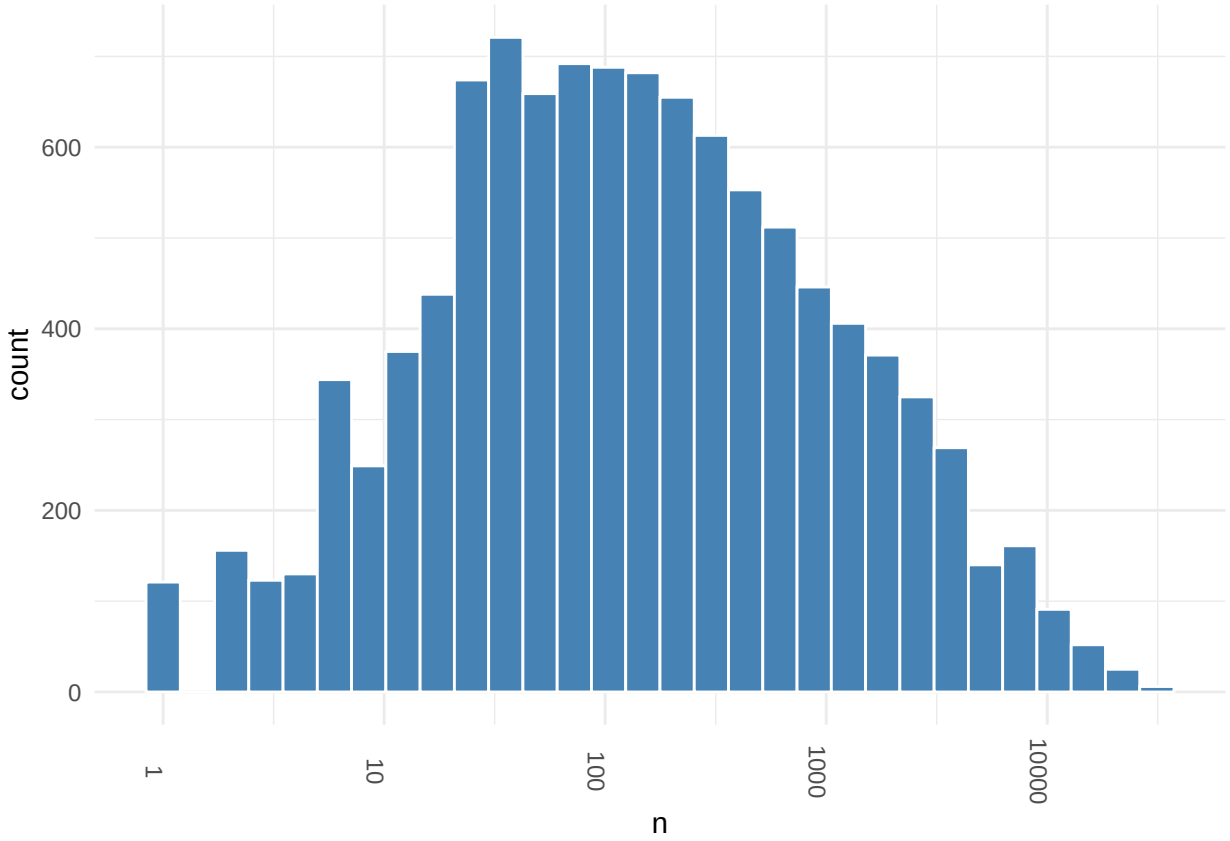


We can see that some users are very prolific and have rated many movies, and that some users have rated very few movies.

We will take this bias into account when modelling and call it the "user activity bias".

Let's do the same with the movie IDs:

```
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
    geom_histogram(bins = 30, color = "white", fill = "steelblue") +
    scale_x_log10() +
    theme_minimal() +
    theme(legend.position = "none", axis.text.x = element_text(angle = -90))
```



Same here, some movies are more rated than others, and that will be taken into account during modelling under the name “movie bias”.

Modelling

Simplest possible model

The simplest model is to predict the average rating for every movie:

```
avg <- mean(edx$rating)
rmse_simple <- caret::RMSE(pred = avg, obs =validation$rating)
rmse_simple
```

```
## [1] 1.060651
```

The RMSE of this method is 1.0606506.

Introducing user activity bias

The user activity bias is calculated as the average rating by user minus the average rating of the whole edx set:

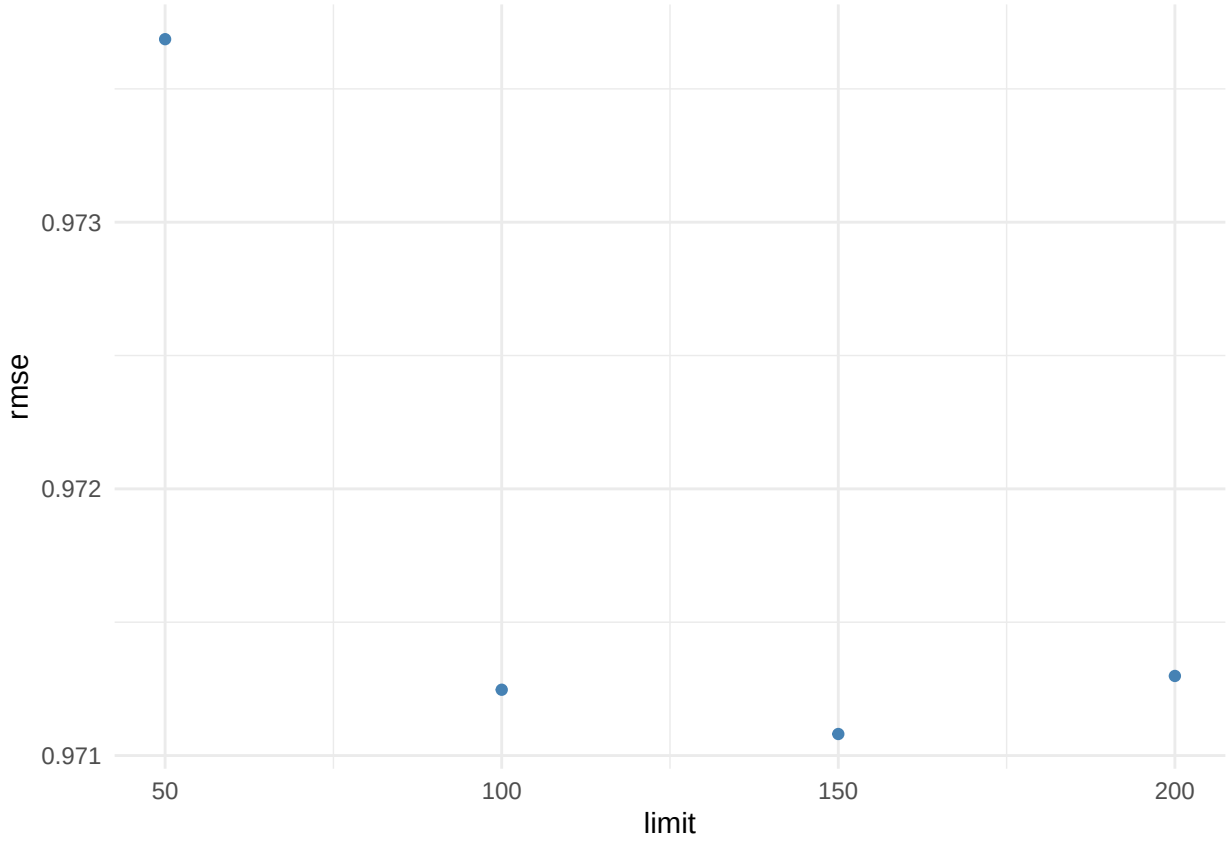
```
user_bias <- edx %>%
  group_by(userId) %>%
  summarise(ub = mean(rating - avg))

data <- validation %>% select(userId, rating) %>% left_join(user_bias, by = 'userId') %>% mutate(pred= a

user_bias_rmse <- RMSE(pred = data$pred, obs = data$rating)

user_bias_rmse
```

```
## [1] 0.9785992
```

As there are users who have a very low number of ratings, let's try and get a number of ratings under which we will exclude these users:

```
limits <- seq(50, 200, 50)

rmses <- sapply(limits, function(limit){
  users <-
    edx %>% count(userId, name = "total") %>%
    filter(total >= limit)

  user_bias <- edx %>%
    filter(userId %in% users$userId) %>%
    group_by(userId) %>%
    summarise(ub = mean(rating - avg))

  data <- validation %>%
    filter(userId %in% users$userId) %>%
    select(userId, rating) %>%
    left_join(user_bias, by = 'userId') %>%
    mutate(pred= avg + ub)

  RMSE(pred = data$pred, obs = data$rating)
})

tibble(limit = limits, rmse = rmses) %>%
  ggplot(aes(x = limit)) +
  geom_point(aes(y = rmse), colour = "steelblue") +
```

```
  theme_minimal()
```



The RMSE is lowest when we filter away the users with less that 150 ratings, so we apply this and can see a slight improvement in our RMSE:

```
users <-
  edx %>% count(userId, name = "total") %>%
  filter(total >= 150)

user_bias <- edx %>%
  filter(userId %in% users$userId) %>%
  group_by(userId) %>%
  summarise(ub = mean(rating - avg))

data <- validation %>%
  filter(userId %in% users$userId) %>%
  left_join(user_bias, by = 'userId') %>%
  mutate(pred= avg + ub)

user_bias_rmse <- RMSE(pred = data$pred, obs = data$rating)

user_bias_rmse
```

```
## [1] 0.9710808
```

Let's calculate a movie bias:

```
movie_bias <- edx %>%
  group_by(movieId) %>%
  summarise(mb = mean(rating - avg))

data <- validation %>%
  left_join(movie_bias, by = 'movieId') %>%
  mutate(pred = avg + mb)

movie_bias_rmse <- RMSE(pred = data$pred, obs = data$rating)

movie_bias_rmse
```

```
## [1] 0.9437046
```

We will combine both biases:

```
data <- validation %>%
  select(userId, movieId, rating) %>%
  left_join(movie_bias, by = 'movieId') %>%
  left_join(user_bias, by = 'userId') %>%
  mutate(pred = avg + ub + mb)

user_movie_bias_rmse <- RMSE(pred = data$pred, obs = data$rating, na.rm = T)
user_movie_bias_rmse
```

```
## [1] 0.8640211
```

The RMSE (0.8640211) is already under the target RMSE so no further improvement is needed.

Conclusion:

I have achieved the goal of the project (RMSE < 0.8649 for full points), and therefore will not continue lowering the RMSE, but further improvements can be gained by defining a coeficient for each user so we have a weighted average based on his ratings' total compared to the total of ratings.

All of that will be achieved depending on CPU and RAM limits, and as with every improvement in accuracy, we need to wwight the spend on hardware against the difference in accuracy gained.