Report

Contents

1 Introduction

This report is part of the final course in the HarvardX Data Science Professional Certificate, Capstone. The aim of the project is to create a movie recommendation system using the MovieLens dataset. Thus, to achieve this goal we will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

According to Michael Hahsler: "Predicting ratings and creating personalized recommendations for products like books, songs or movies online came a long way from Information Lense, the first system using social filtering created by Malone, Grant, Turbak, Brobst, and Cohen (1987) more than 20 years ago. Today recommender systems are an accepted technology used by market leaders in several industries (e.g., by Amazon, Netflix and Pandora). Recommender systems apply statistical and knowledge discovery techniques to the problem of making product recommendations based on previously recorded data (Sarwar, Karypis, Konstan, and Riedl 2000). Such recommendations can help to improve the conversion rate by helping the customer to find products she/he wants to buy faster, promote cross-selling by suggesting additional products and can improve customer loyalty through creating a value-added relationship (Schafer, Konstan, and Riedl 2001). The importance and the economic impact of research in this field is reflected by the Netflix Prize, a challenge to improve the predictions of Netflix's movie recommender system by more than 10% in terms of the root mean square error. The grand price of 1 million dollar was awarded in September 2009 to the Belcore Pragmatic Chaos team."

We will be creating our own recommendation system using the tools we have learned throughout the courses. Thus, we will use R language to write the code, and its libraries, which will help us to complete the task. We will wrangle data, visualize it, and create a machine learning model, which will run using probability and linear regression concepts.

Because the computation will be run on a personal computer, We will use the 10M version of the MovieLens dataset to make it a little easier. This dataset is available on "http://files.grouplens.org/datasets/movielens/ml-10m.zip". Thus, to start we will: install the libraries that will help us to perform our task; download the dataset; create a training and a test set, which will be called edx set and validation set respectively. Then, we will do some data exploration. In sequence we will create a moded.

After that, we will measure the quality of our models by RMSE, which stands for root mean square deviation.It is the same measurament the Netflix used in its Challenge, yet in our case the goal is to reach a RMSE less than 0,87750.

2 Methods/ analysis

The first step is to define what libraries we are going to use. We will need the tidyverse package to manipulate the dataset, and the caret package to develop our machine learning model.

Then, we will create the train set, which will be called edx, and the test set, which will be called validation.

```
knitr::opts_chunk$set(echo = TRUE)

################################################################
# Create edx set, and 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 1.2.1

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0

## -- 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
```

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

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

```
ratings <- read.table(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)
```

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

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

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

Once our datasets are created, we can examine it to get a better view of what we are dealing with.
First we can check names of the collumns:

```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

We can also see that each row shows the rating that an user has given to a movie. For isntance, we can see the first ten rows of the data:

```
head(edx, n = 10L)
```

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller

	userId	movieId	rating	timestamp	title	genres
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-F
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
8	1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
9	1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
10	1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children
11	1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

To find out the number of movies, we will use the code below:

```
length(unique(edx$movieId))
```

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

Similarly we can discovery how many unique users there are in the dataset:

```
length(unique(edx$userId))
```

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

Using the stringi, and stringr packages, we can work with strings. As a consequence, we can determine the genres of the movies.

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

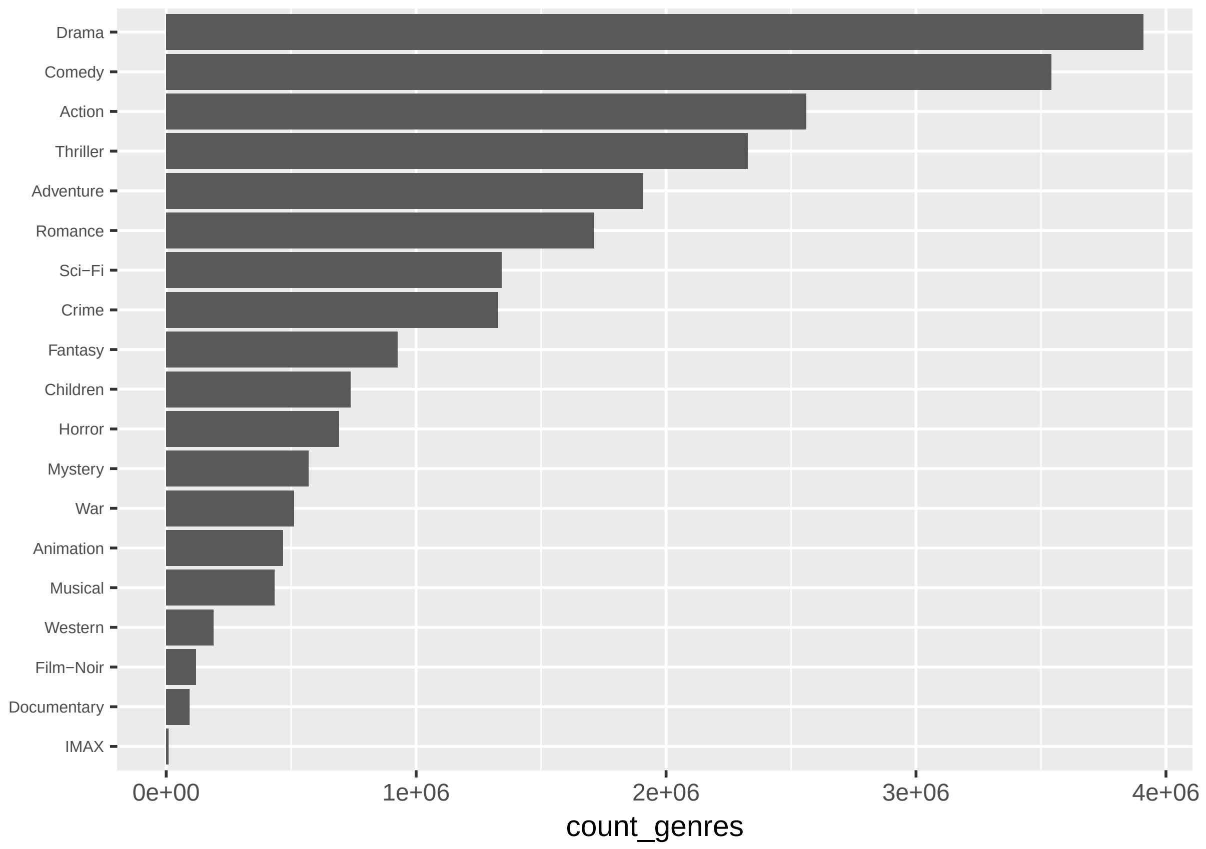
```
## Loading required package: stringi
```

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

genres = unique(unlist(str_extract_all(edx$genres,"[A-Z][^|]+")))
genres
```

```
##  [1] "Comedy"      "Romance"     "Action"      "Crime"       "Thriller"
##  [6] "Drama"       "Sci-Fi"      "Adventure"   "Children"    "Fantasy"
## [11] "War"         "Animation"   "Musical"     "Western"     "Mystery"
## [16] "Film-Noir"   "Horror"      "Documentary" "IMAX"
```

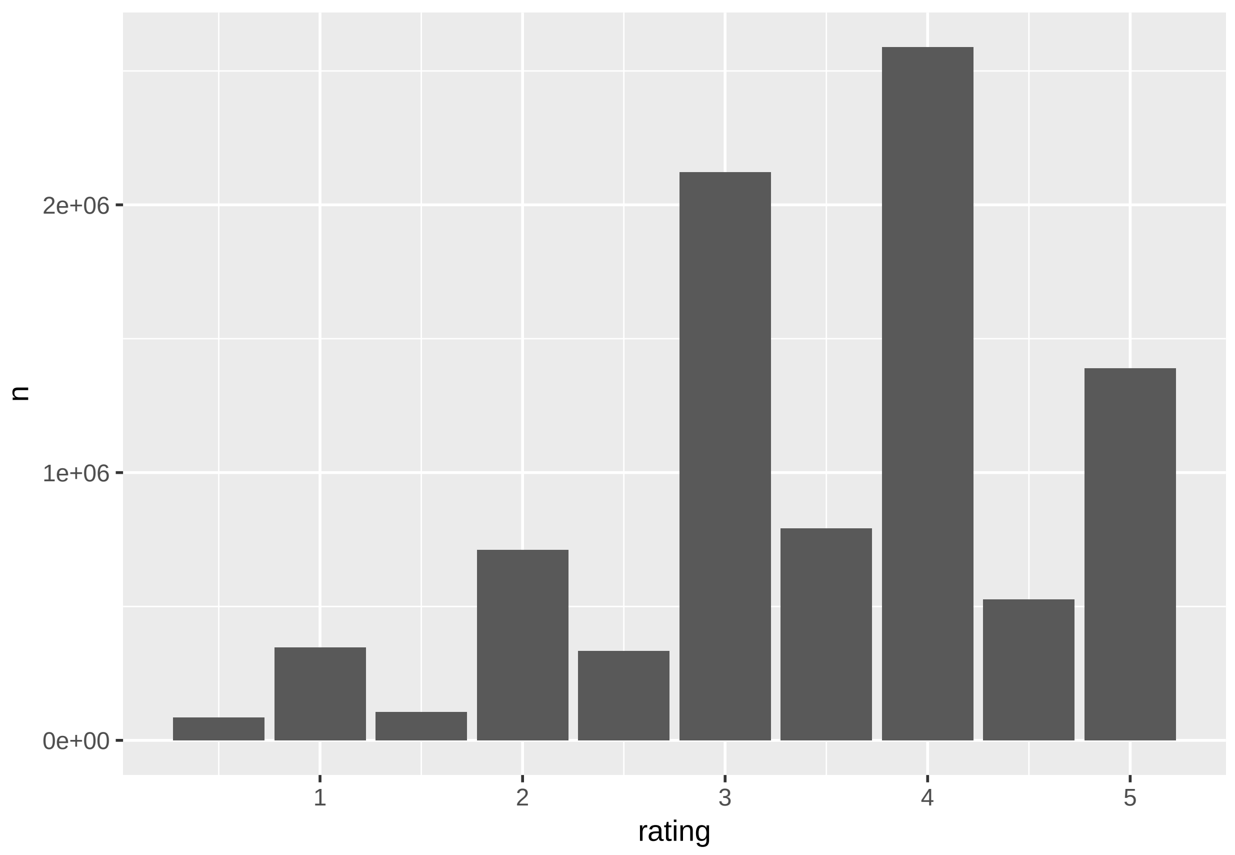
Then, we can see the number of movie grouped by genre.

```
#count the number of movies group by genre
count_genres = sapply(genres, function(x){
  n = sum(str_count(edx$genres,x))
})

#plot the number of movies group by genre (descending order)
data.frame(genres, count_genres) %>%
  mutate(genres = reorder(genres, count_genres)) %>%
  ggplot(aes(genres, count_genres)) +
  geom_bar(stat="identity") +
  coord_flip() +
  theme(axis.text.y = element_text(size = 6)) +
  xlab("")
```



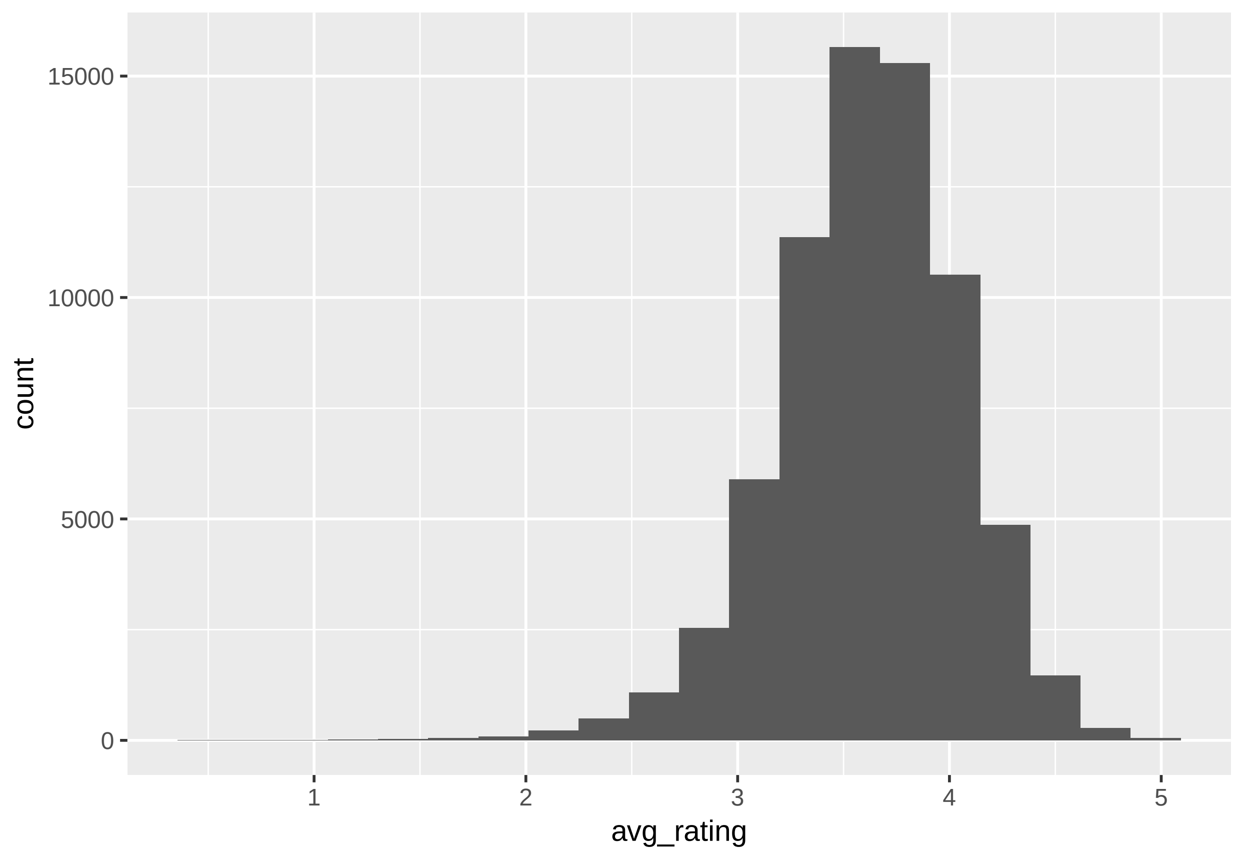
Then, we can find out the distribution of ratings.

```
edx %>% group_by(rating) %>% summarise(n = n()) %>%
  ggplot(aes(rating,n)) + geom_bar(stat = "identity")
```



We can see that there is a user effect using the code below:

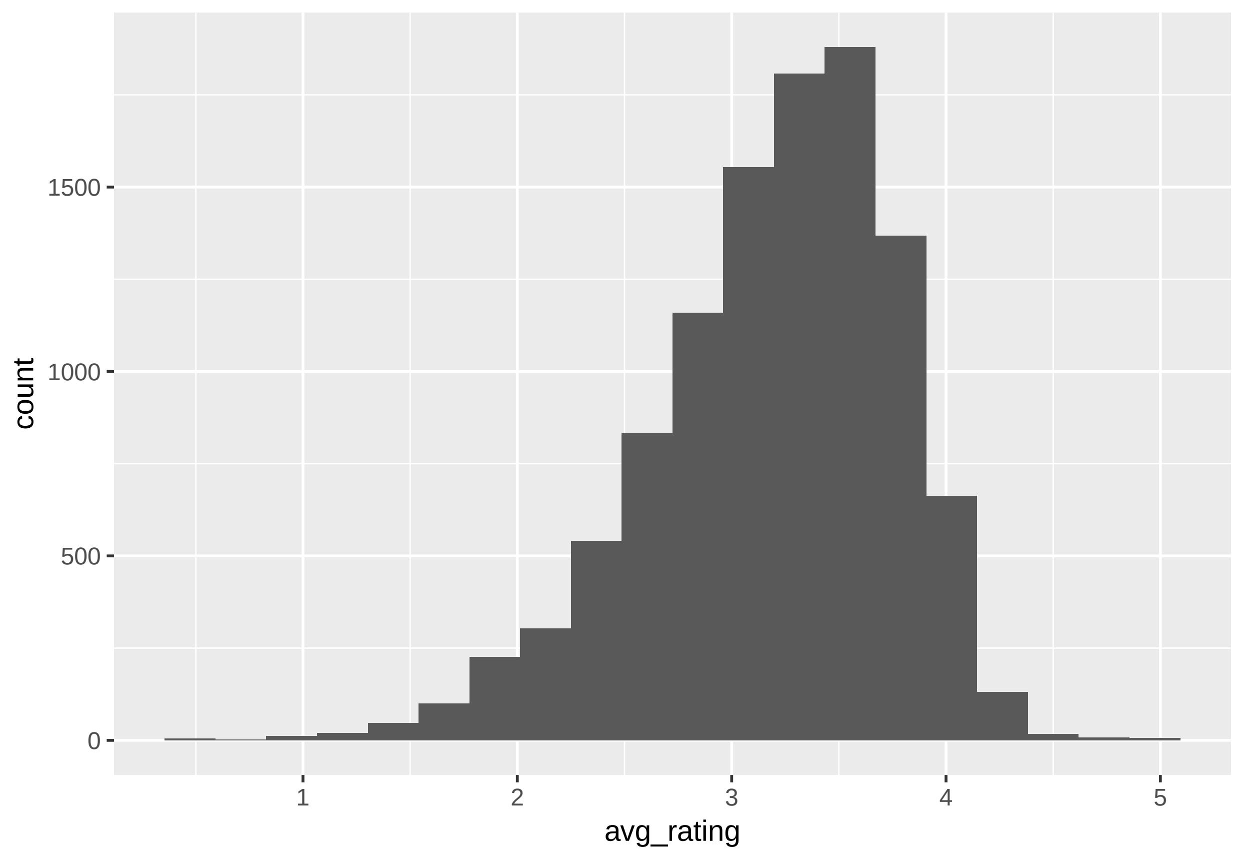
```
user_rating = edx %>% group_by(userId) %>% summarise(avg_rating = mean(rating),sd_rating = sd(rating))
user_rating %>% ggplot(aes(avg_rating)) + geom_histogram(bins = 20)
```



```
rm(user_rating)
```

Similarly, there is a movie effect

```
movie_rating =  edx %>% group_by(movieId) %>% summarise(avg_rating = mean(rating),sd_rating = sd(rating
movie_rating %>% ggplot(aes(avg_rating)) + geom_histogram(bins = 20)
```



```r
rm(movie_rating)
```

And, genre effect

```r
ratings_genres = map_df(genres, function(x){
  n = edx %>% filter(str_detect(edx$genres,x)) %>% summarise(Genre = x, avg = mean(rating), sd= sd(rati
})
ratings_genres %>% knitr::kable()
```

Genre	avg	sd
Comedy	3.436908	1.0746511
Romance	3.553813	1.0304136
Action	3.421405	1.0665828
Crime	3.665925	1.0119106
Thriller	3.507676	1.0311492
Drama	3.673131	0.9953970
Sci-Fi	3.395743	1.0927764
Adventure	3.493544	1.0529353
Children	3.418715	1.0923977
Fantasy	3.501946	1.0654636
War	3.780813	1.0118036
Animation	3.600644	1.0193206
Musical	3.563305	1.0568704
Western	3.555918	1.0237553

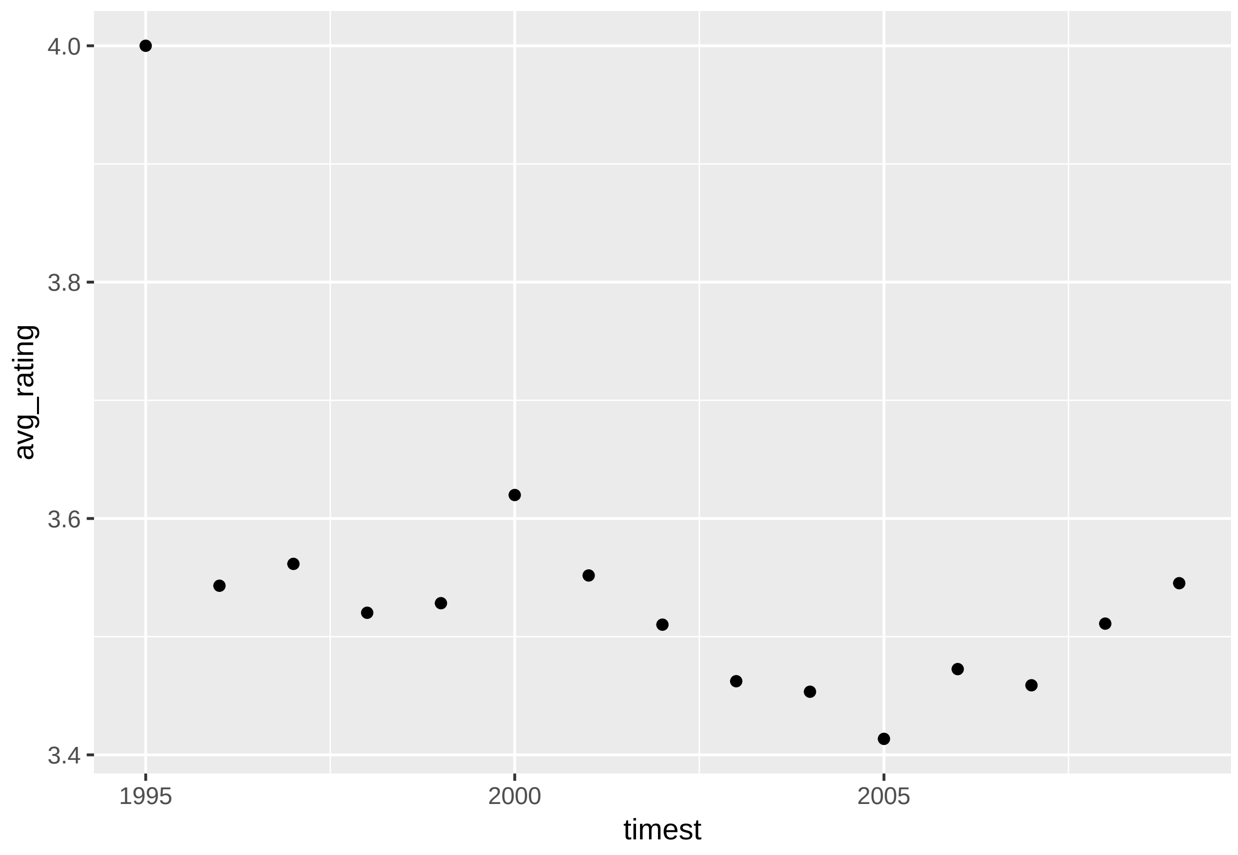
Genre	avg	sd
Mystery	3.677001	1.0002628
Film-Noir	4.011625	0.8871659
Horror	3.269815	1.1499549
Documentary	3.783487	1.0038488
IMAX	3.767693	1.0323171

And also, a time effect

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

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

```r
edx$date = as_datetime(edx$timestamp, origin = "1970-01-01")
edx$year = year(edx$date)

edx %>% mutate(timest = round_date(date, "year"))%>% group_by(timest) %>%
  summarise(avg_rating = mean(rating)) %>%
  ggplot(aes(timest,avg_rating)) + geom_point()
```



Therefore, we can create a model combining these effects. In fact, we are going to try the follow model to predict the rating $Y_{u,i}$, that a user u can give to a movie i, which has some genre g.

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Although this model is similar to a linear regression model, we are not going to use the lm fuction because of this code will run on a personal computer.

3 Results

First, we can calculate the average and considiring this as a model we can calcule RMSE, which will help us to evaluate the improvemment of our models when we complete the following steps.

```
####just the average
mu_hat <- mean(edx$rating)
mu_hat
```

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

```
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

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

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.061202

Since, there is a genre effect, we can calculate the average for it genre. Because, of each movie has more than one genre, we will considerate the genre effect as the mean of average of all genres that define a movie.

```
#####genre effect
genres_avgs = map_dbl(1:length(unique(edx$genres)), function(x){
  mu = mean(ratings_genres$avg[str_detect(unique(edx$genres)[x],ratings_genres$Genre)])
})
genres_mu = data.frame(genres = unique(edx$genres), mu = genres_avgs, stringsAsFactors = F)
avg_genres = edx %>%
  left_join(genres_mu, by="genres")

predicted_ratings = validation %>%
  left_join(genres_mu, by="genres") %>%
  .$mu


model_1_rmse = caret::RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Genre Effect Model",
                                     RMSE = model_1_rmse ))

rmse_results
```

method	RMSE
Just the average	1.061202
Genre Effect Model	1.048384

Altough, it is possiblie to see a improvement in the RMSE, it was not significant considering just the genre effect. Thus, the next step is to use the movie effect.

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

predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu_hat + b_i ) %>%
  .$pred


model_2_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse ))
rmse_results
```

method	RMSE
Just the average	1.0612018
Genre Effect Model	1.0483841
Movie Effect Model	0.9439087

Once we have added the movie effect, we can improve the model considering the user effect.

```
#####users effect
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, validation$rating)
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="User + Movie Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results
```

method	RMSE
Just the average	1.0612018
Genre Effect Model	1.0483841
Movie Effect Model	0.9439087
User + Movie Effect Model	0.8653488

We can see that the model that uses the genre, the movie, and the user effect has given us a RMSE of 0,865, which is less than our target of 0,87750.

4 Conclusion

In this project, we have created a recommender model. It works similar to a linear regression and has achived a final RMSE of 0,865, which is less than our initial target of 0,8775. Thus we have had sucess in our project. Although we have reached our goal, further improvement could be achieved by other techniques, such as regularization and boostraping. The regularization will help us to remove the impact than some movies are more rated than others.

5 References

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[2] Michael Hahsler (2019). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-4.https://github.com/mhahsler/recommenderlab