Capstone Project - Recommender System

Stanislav Taov

20/12/2020

Introduction

This project was offered as part of the edX - HarvardX Data Science course. The main purpose of this project was to create a movie recommendation system using a small subset of a much larger MovieLens dataset.

Recommender systems are a specific class of machine learning techniques that are widely used in web applications like e-commerce sites, news sites, social networks or streaming services. The main purpose of recommenders is to personalize user experience via suggesting to the end-user (consumer) something that will enhance their overall experience or help to make a decision. These recommendations are based on previous user experience call collaborative filtering. For example, two users who shared similar interests in the past might have a similar interest in the future. Therefore we can offer them certain items that will match their interests. Collaborative recommenders take into account only user preferences and ignore the characteristics or content of items they recommend. The other type of recommenders is content-based systems. These systems take into consideration similarities between items and similarities between user data and require a large amount of data for better accuracy. There are also, knowledge-based systems that take into consideration information about the items/products and match them with user historical data. And finally, there are hybrid systems that combine features of previously mentioned systems into one recommender, this approach can reduce the disadvantages of both systems and create a more robust recommender.

Project Goal

The project goal was to create a machine learning model that can predict movie ratings using data provided in edX data set. To evaluate the model the validation data set was provided as well. Root Mean Square Error was used as a loss function to measure the accuracy of the models (finding the difference (error) between ground truth rankings and predicted rankings).Low RMSE means that the model preforms well and predicts rankings accurately.

Data

In this project the MovieLens 10M data set was used. The data set was created by GroupLens research that has collected and made it available. The data set has 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

```
# 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")

library(tidyverse)
library(caret)
library(data.table)

# 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")
```

Creating Edx (train and test) and Validation Datasets

This code was provided by the edX and modified to generate train, test and validation data sets. The validation data set was used to evaluate model effectiveness in predicting move ratings.

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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)

# Create train and test sets from the edx set with the ratio 80/20
```

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set

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

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

After checking, both data sets don't have any missing values.

```
sum(is.na(edx))
```

```
## [1] 0
```

```
sum(is.na(validation))
```

```
## [1] 0
```

Data Exploration

We use tidy format to display top 5 rows in the edx data set.

```
head(edx, 5) %>%
  knitr::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	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

```
dim(edx)
```

```
## [1] 9000055       6
```

```
dim(validation)
```

```
## [1] 999999      6
```

The complete data set has over 10 million observations with 6 features. The dataset was separated into edx and validation sets containing ~9M and ~1M observations. Each row in the edx data set represents unique

user ratings of movie. We can use summarize function to find number of unique users and number of unique movies. The edx data set was partitioned into train and test sets with 80/20 ratio.

```
edx %>%
  summarize(unique_users = n_distinct(userId),
            unique_movies = n_distinct(movieId))
```

```
##   unique_users unique_movies
## 1        69878         10677
```

The edx data set combines 69878 unique users and 10677 unique movies. So, If we multiply those two numbers, we get a much number larger than 10 million. This implies that not every user rated every movie. The main problem with the data in recommender systems is data sparsity. For example, it is almost impossible to have data where all users rank all items. So in reality only a small fraction of users rank items as a result we get a very sparse matrix when we build recommendation systems. Let's display our sparse matrix for top 5 movies and 20 random users.

```
top_movies <- edx %>%
  dplyr::count(movieId) %>%
  top_n(5) %>%
  pull(movieId)
```

```
## Selecting by n
```

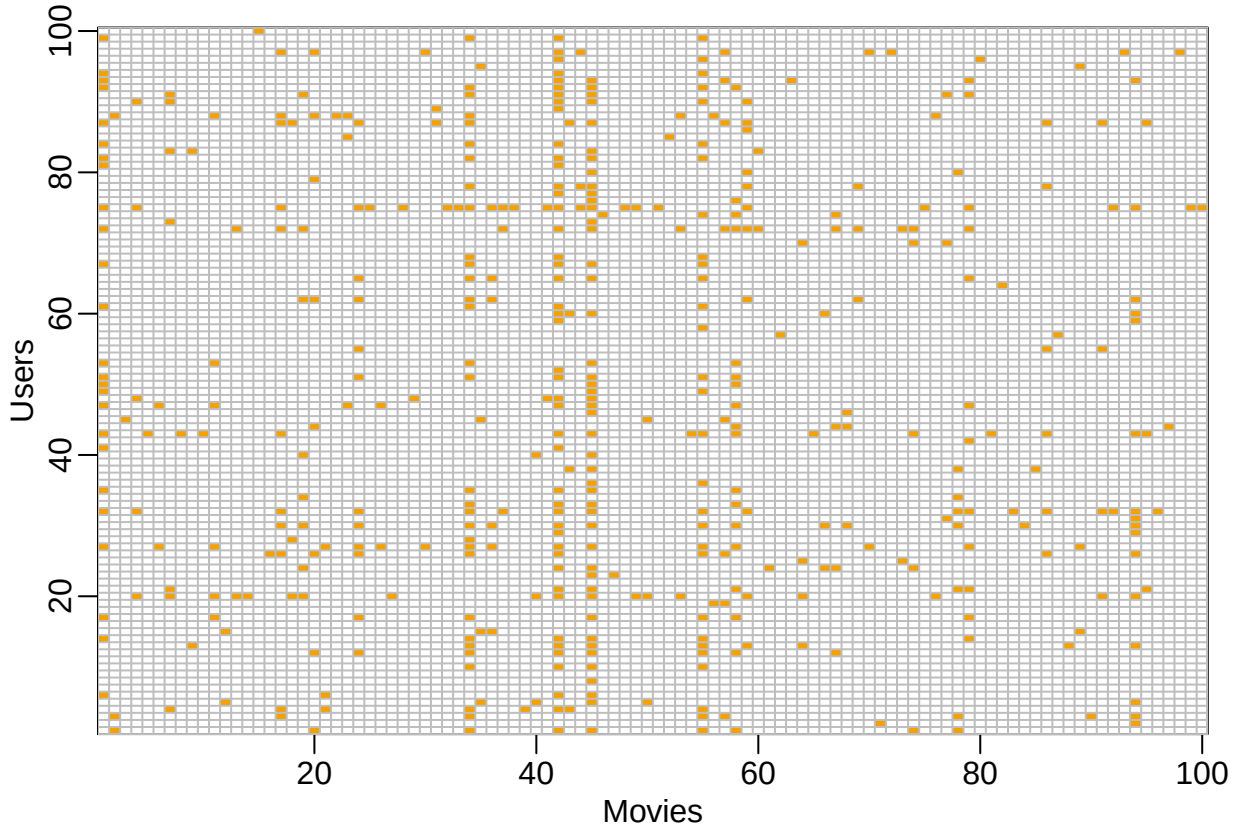
```
random_rankings <- edx %>%
  filter(userId %in% sample(unique(edx$userId), 20)) %>%
  filter(movieId %in% top_movies) %>%
  select(userId, title, rating) %>%
  spread(title, rating)

random_rankings %>% knitr::kable()
```

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)
1349	4.0	NA	5	5.0	4.0
3440	1.0	4.5	NA	3.5	NA
25904	4.0	4.0	NA	NA	NA
28838	NA	NA	2	NA	NA
30188	NA	NA	5	5.0	3.5
31098	NA	NA	NA	4.5	NA
36545	NA	4.0	5	NA	4.0
38659	NA	NA	NA	4.5	NA
49154	4.0	1.5	5	4.5	4.5
51100	3.0	3.0	5	NA	4.0
55903	4.5	NA	NA	NA	3.0
64170	3.5	3.0	5	4.0	NA
68989	4.5	NA	NA	NA	NA

We also can visualize the sparse matrix using random sample of 100 movies and 100 users. Orange mark

indicate that the user provided a rating for a particular movie. All empty cells indicate that there is no data.

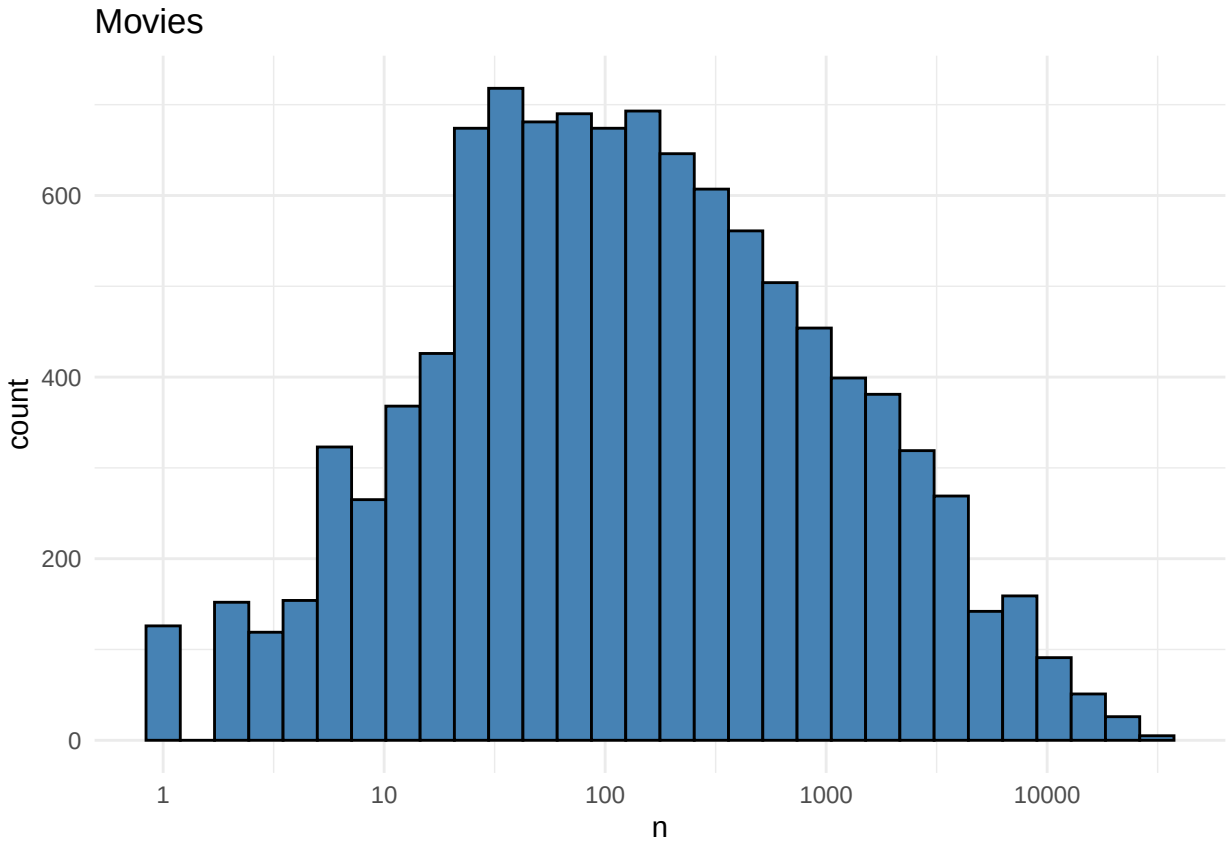
```
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>% as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")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



If we want to predict the rating of movie i using user data u we can look at the problem in two ways. As we see different users rated different movies and if we have a big enough data set we can find users similar to user u who rated movie i and use this data to provide a rating for movie i or we can find similarities between movies and use this data to rate movie i. That is why it is very important to have large enough data for the appropriate model accuracy.

Now we can look at distributions of some data features to better understand our data.

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, fill="steelblue", color = "black") +
  scale_x_log10() +
  theme_minimal() +
  ggtitle("Movies")
```



As we can see some movies were rated more often than others. There are clearly some outliers, some movies were rated a few times where others were rated over 10000 times. There are top 20 movies that were rated only once and have the highest ratings, these movies will create noise and impact RSME negatively, therefore regularization and penalty terms will be used to mitigate these outliers.

```
edx %>%
  group_by(title) %>%
  summarise(count = n(), rating = mean(rating)) %>%
  filter(count < 2) %>%
  arrange(desc(rating)) %>%
  slice(1:20) %>%
  knitr::kable()
```

title	count	rating
Blue Light, The (Das Blaue Licht) (1932)	1	5.0
Fighting Elegy (Kenka erejii) (1966)	1	5.0
Hellhounds on My Trail (1999)	1	5.0
Shadows of Forgotten Ancestors (1964)	1	5.0
Sun Alley (Sonnenallee) (1999)	1	5.0
Bad Blood (Mauvais sang) (1986)	1	4.5
Demon Lover Diary (1980)	1	4.5
Kansas City Confidential (1952)	1	4.5
Ladrones (2007)	1	4.5
Man Named Pearl, A (2006)	1	4.5
Mickey (2003)	1	4.5
Please Vote for Me (2007)	1	4.5

title	count	rating
Testament of Orpheus, The (Testament d'Orphée) (1960)	1	4.5
Tokyo! (2008)	1	4.5
Valerie and Her Week of Wonders (Valerie a týden divu) (1970)	1	4.5
Bellissima (1951)	1	4.0
David Holzman's Diary (1967)	1	4.0
Deadly Companions, The (1961)	1	4.0
Family Game, The (Kazoku gêmu) (1983)	1	4.0
Fists in the Pocket (I Pugni in tasca) (1965)	1	4.0

There are top 20 movies that were rated over 10000 times and have highest ratings. All these movies are either classic or blockbuster movies.

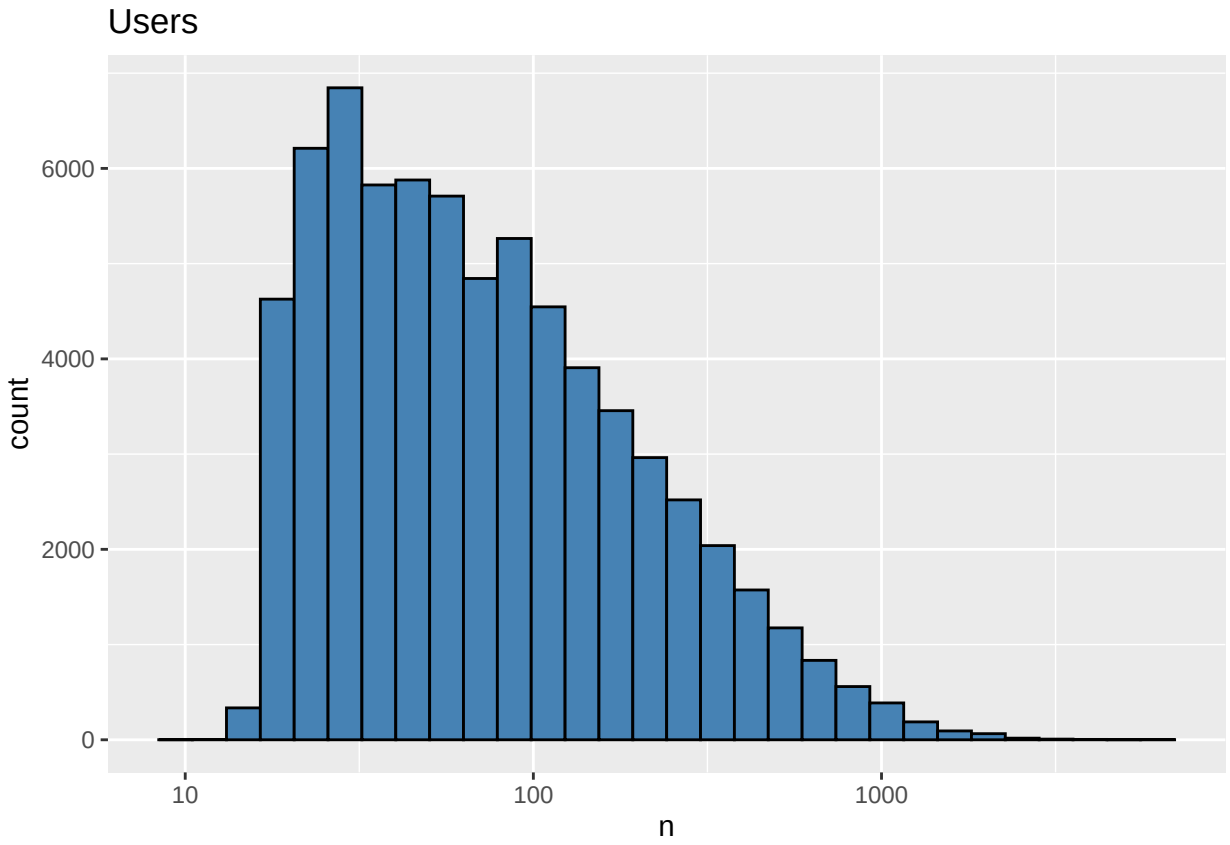
```
edx %>%
  group_by(title) %>%
  summarise(count = n(), rating = mean(rating)) %>%
  filter(count > 10000) %>%
  arrange(desc(rating)) %>%
  slice(1:20) %>%
  knitr::kable()
```

title	count	rating
Shawshank Redemption, The (1994)	28015	4.455131
Godfather, The (1972)	17747	4.415366
Usual Suspects, The (1995)	21648	4.365854
Schindler's List (1993)	23193	4.363493
Casablanca (1942)	11232	4.320424
Godfather: Part II, The (1974)	11920	4.301971
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	10627	4.295333
One Flew Over the Cuckoo's Nest (1975)	13014	4.293261
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)	19678	4.259401
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672	4.221311
Memento (2000)	11618	4.216862
Monty Python and the Holy Grail (1975)	14635	4.205535
Silence of the Lambs, The (1991)	30382	4.204101
Matrix, The (1999)	20908	4.202578
Princess Bride, The (1987)	14809	4.195557
Star Wars: Episode V - The Empire Strikes Back (1980)	20729	4.192918
Fight Club (1999)	14732	4.189825
Goodfellas (1990)	11807	4.187219
American Beauty (1999)	19950	4.185664
Lord of the Rings: The Fellowship of the Ring, The (2001)	14406	4.162884

By look at userId distribution we can conclude that some users were more active than others at rating movies, and the data is skewed right by some users who rated over 1000 movies.

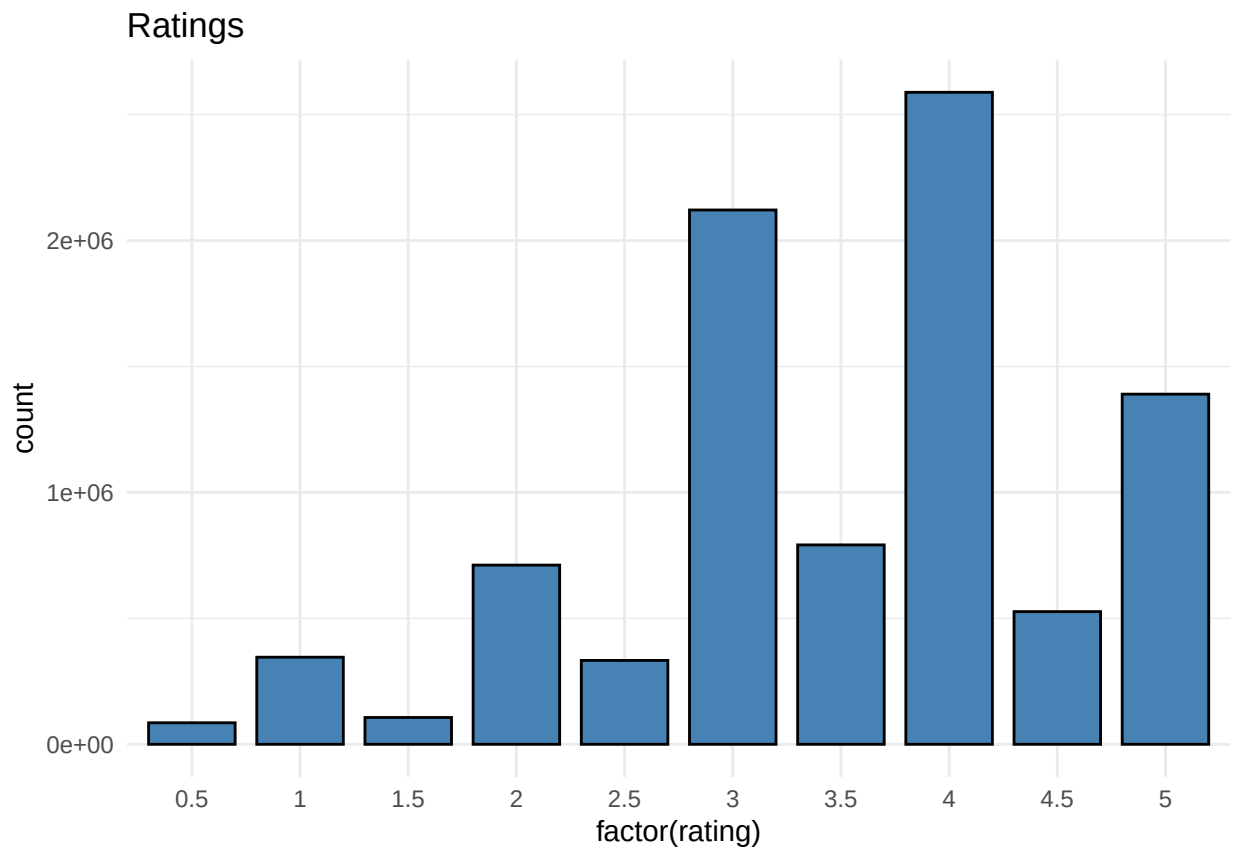
```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, fill="steelblue", color = "black") +
```

```
  scale_x_log10() +
  ggtitle("Users")
```

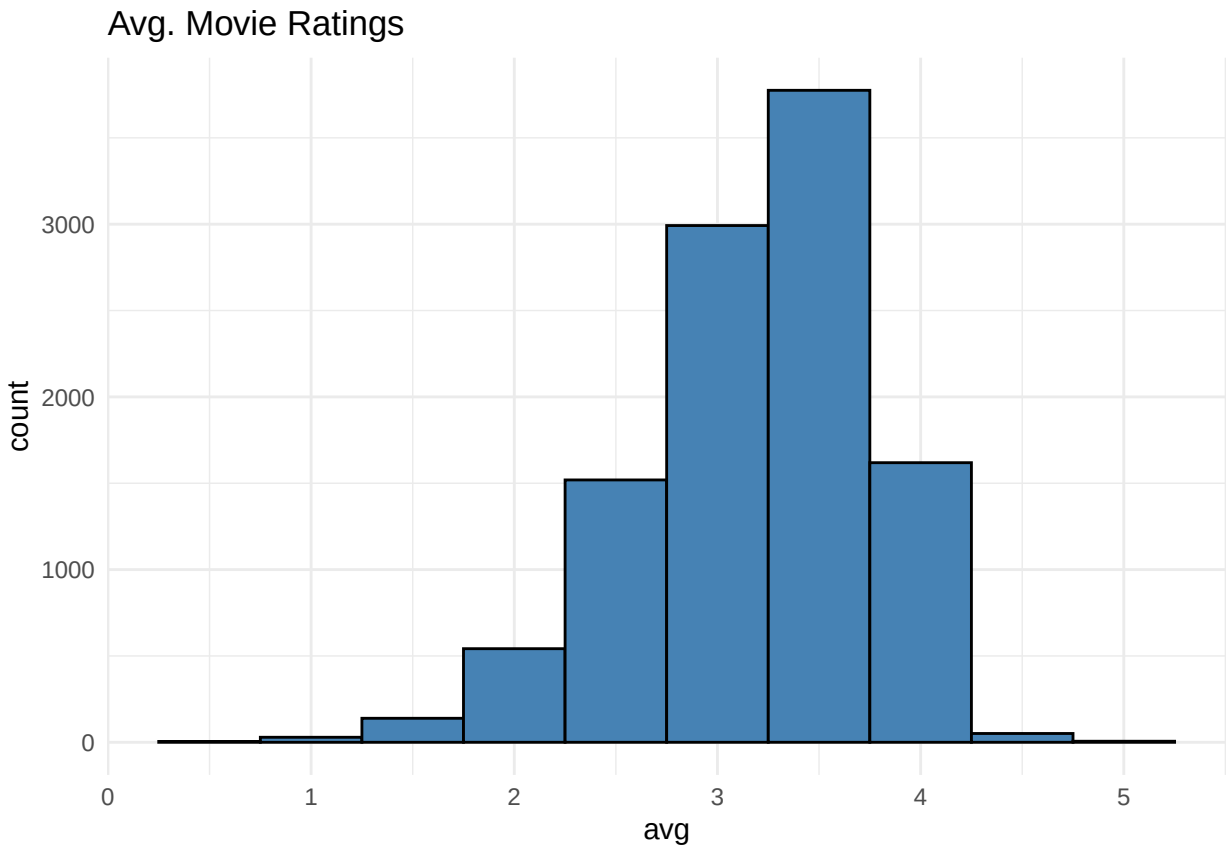


We can see that majority of ratings were 3 and 4 and there were very few 0.5 and 1.5 ratings.

```
ggplot(edx, aes(x = factor(rating))) +
  geom_bar(width=0.8, fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Ratings")
```

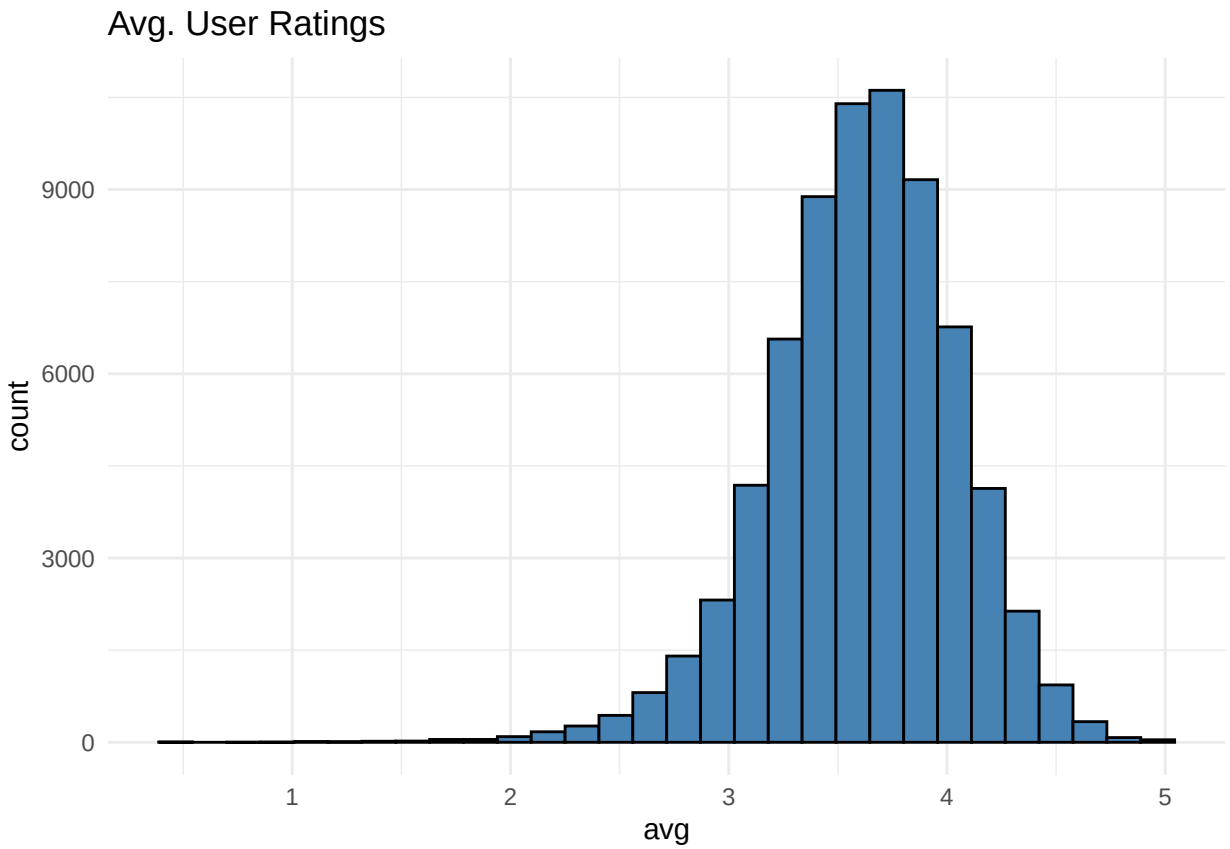



```
# plot average rating by movie
edx %>%
  group_by(movieId) %>%
  summarise(avg = mean(rating)) %>%
  ggplot(aes(x=avg)) +
  geom_histogram(bins = 10, fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Avg. Movie Ratings")
```



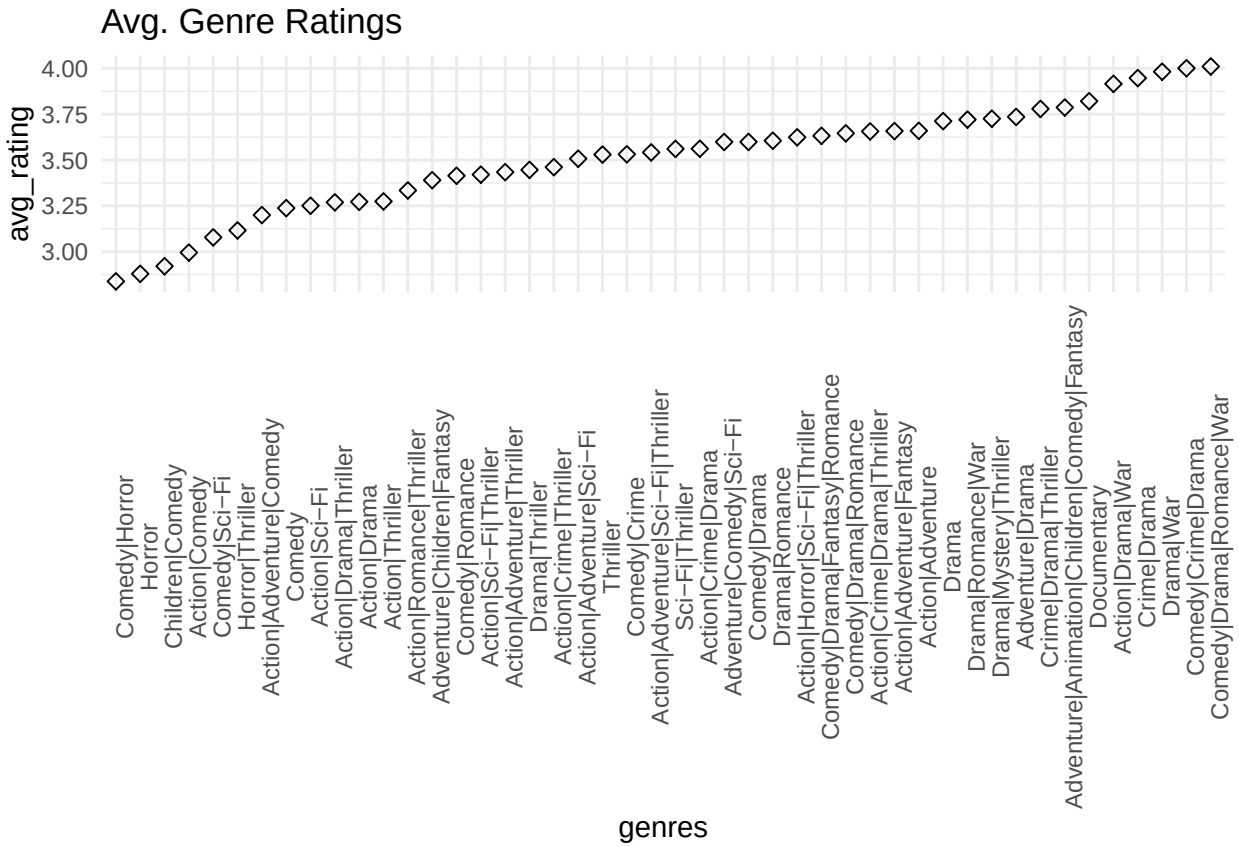
From this graph, it can be observed that the average movie ratings have almost normal distribution but there are a few outliers with very low and very high rankings.

```
# plot average rating by users
edx %>%
  group_by(userId) %>%
  summarise(avg = mean(rating)) %>%
  ggplot(aes(x=avg)) +
  geom_histogram(bins = 30, fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Avg. User Ratings")
```



This graph shows left skewed normal distribution which can indicate that some users only rate movies when they don't like them. However, the opposite is also true, there were some users that gave only the highest ratings.

```
# plot average rating by genres
edx %>%
  group_by(genres) %>%
  summarise(count=n(), avg_rating = mean(rating)) %>%
  filter(count >= 35000) %>%
  mutate(genres= reorder(genres, avg_rating)) %>%
  ggplot(aes(x=genres, y=avg_rating)) +
  geom_point(size = 2, shape = 23) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90)) +
  ggtitle("Avg. Genre Ratings")
```



By plotting the average rating by movie genre it can be seen that movie ratings depend on genre quite a lot. For example, Comedy/Horror movies on average will have lower rating than Comedy/Crime/Drama movies.

Loss function

We used residual mean squared error (RMSE) similarly to what was used in the Netflix challenge to determine how well each model performed. We can define $y_{u,i}$ as the rating for movie i by user u and therefore we can denote out prediction with $\hat{y}_{u,i}$. The RMSE is then defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

with N being the number of user/movie combinations and the sum occurring over all these combinations.

We can assume that the RMSE is quite similar to the standard deviation. If RMSE is larger than 1 which means that our typical error is larger than one star and the model prediction is not very accurate.

Let's create a function that will compute the RMSE between ground truth ratings and their corresponding predictors:

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

Model 1 - Base Model (Naive)

We can start by building the simplest recommendation system by predicting the same rating for all movies regardless of other data features. The difference in predictions can be explained by random variation in the independent variable.

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

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

```
## [1] 3.512482
```

```
rmse_base <- RMSE(test$rating, mu)
rmse_base
```

```
## [1] 1.059904
```

The base model yielded RMSE = 1.059904 which is quite high and will be improved by using other features of the data set. Let's create a result table with this base(naive) model.

```
rmse_results <- data.frame(method = "Base(Naive) Model", RMSE = rmse_base)
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.059904

Model 2 - Movie Effect

From the data exploration we know that movies should have an impact on the model since not all movies were rated equally and some were rated higher than others. So, we enhance our previous model by adding a term b_i that represents the average ranking for movie i. Our new model can be denoted as:

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

where $Y_{u,i}$ is the predicted rating, mu the mean rating for all movies, b_i is the bias for each movie i and$epsilon_{u,i}$ is the independent error.

We can't use *lm()* function to calculate least squares for each movie since it's slow and our data set quite big with over 10,000 movie titles, however, we know that b_i is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. So, they can be computed as follows:

```
# Find b_i bias for each movie
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
# Predict movie ratings
predicted_ratings <- mu + test %>%
  left_join(b_i, by='movieId') %>%
  pull(b_i)
# Find RMSE
rmse_movie <- RMSE(predicted_ratings, test$rating)
rmse_movie
```

```
## [1] 0.9437429
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effects Model",
                                     RMSE = rmse_movie))
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.0599043
Movie Effects Model	0.9437429

The RMSE has improved from 1.059904 to 0.9437429 using b_i.

Model 3 - User Effect

This model takes into consideration that not all users rates all movies, as we know there were more active users and users who rank only movies they didn't like or liked a lot. Similarly to the previous model, we will introduce user bias denoted by b_u. This term is based on the average rating of the individual user and the difference with the average rating. We would like to see if adding b_u term can improve the results of the previous model. As a result, our model can be denoted as:

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

As before using of *lm()* function can be very slow with our data set, therefore instead b_u can be estimated as the average of $y_{u,i} - \hat{\mu} - \hat{b_i}$

```
# Find b_u bias for each user
b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/n())

# Predict movie ratings
predicted_ratings <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Find RMSE
```

```
rmse_user <- RMSE(predicted_ratings, test$rating)
rmse_user
```

```
## [1] 0.865932
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and User Effects Model",
                                     RMSE = rmse_user))
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.0599043
Movie Effects Model	0.9437429
Movie and User Effects Model	0.8659320

The RMSE has improved further from 0.9437429 to 0.8659320.

Model 4 - Genre Effect

From data exploration, we know that different movie genre has an effect on average ratings and we can use this to further improve the RMSE. We can add movie genre b_k bias to the model and as result, our model can be denoted as:

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

The average residual can be calculated as: $y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_k}$

```
# Find b_u bias for each user
b_k <- train %>%
  left_join(b_u, by="userId") %>%
  left_join(b_i, by = "movieId") %>%
  group_by(genres) %>%
  summarise(b_k = (sum(rating - b_i - b_u - mu))/(n()))

# Predicting movie ratings on test set
predicted_ratings <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_k, by="genres") %>%
  mutate(b_k=replace_na(b_k,0)) %>%
  mutate(pred = mu + b_i + b_u + b_k) %>%
  pull(pred)


# Find RMSE
rmse_genre <- RMSE(predicted_ratings, test$rating)
rmse_genre
```

```
## [1] 0.8655941
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie, User and Genre Effects Model",
                                     RMSE = rmse_genre))
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.0599043
Movie Effects Model	0.9437429
Movie and User Effects Model	0.8659320
Movie, User and Genre Effects Model	0.8655941

The RMSE has improved further from 0.8659320 to 0.8655941.

Model 5 - Regularizing Movie and User Effects

Regularization minimizes the total variability of the data and helps to deal with outliers and data skewness that our data set has. Therefore regularization is used to avoid overfilling and can significantly improve the RMSE.

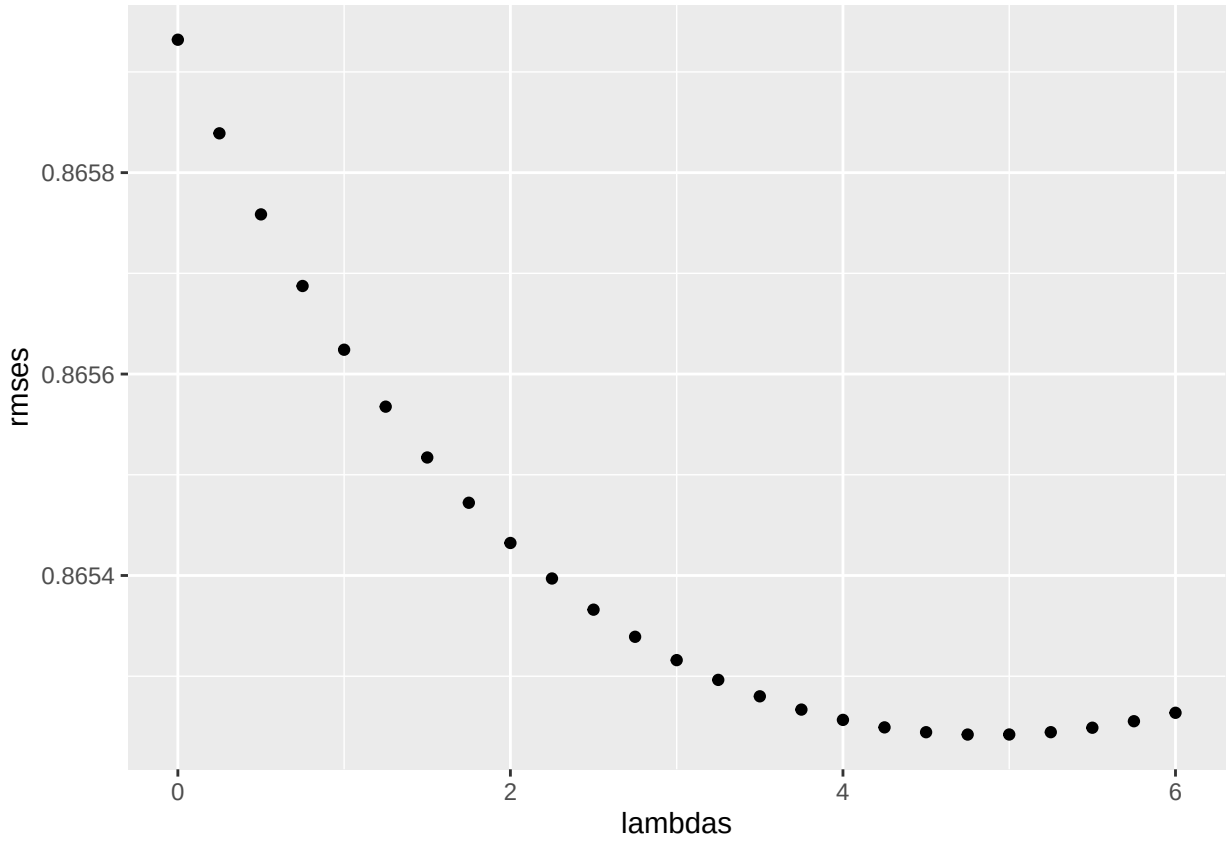
We can use regularization with our previous Movie and User Effects Model by adding λ parameter that minimizes the impact of obscure movies and users, as a result, improving the RMSE and our new model can be denoted as:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

λ is tuning parameter therefore we can use cross-validation to select the correct one.

```
# lambda sequence from 0 to 6 with 0.1 step.
lambdas <- seq(0, 6, 0.25)
# sapply takes values from lambdas and uses them in the function to
# find all possible RMSE values.
rmses <- sapply(lambdas, function(l){

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test$rating))
```

```
})

qplot(lambdas, rmses)
```




```
 # find the minimum value of lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

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

We can see that the lambda that minimizes RMSE is 4.75.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie and User Effects Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.0599043
Movie Effects Model	0.9437429
Movie and User Effects Model	0.8659320
Movie, User and Genre Effects Model	0.8655941
Regularized Movie and User Effects Model	0.8652421

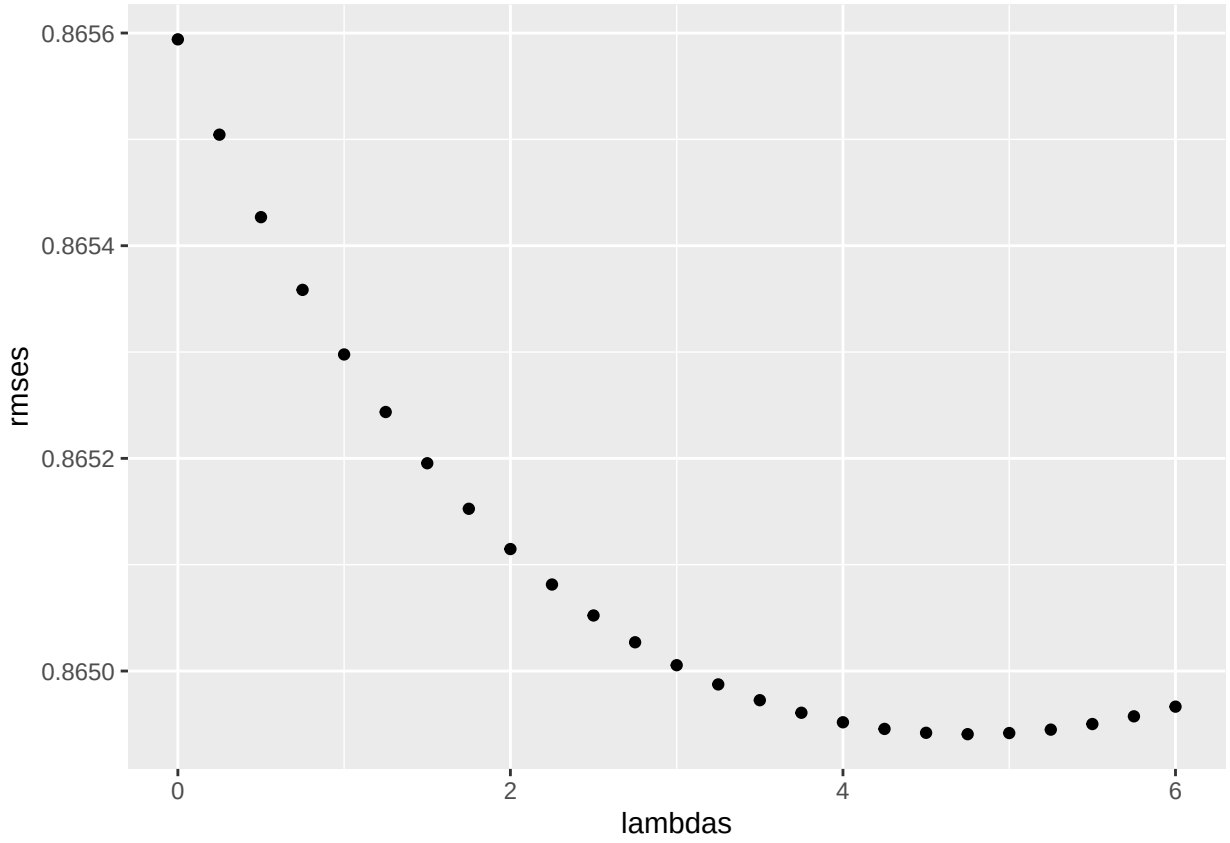
The RMSE improved from 0.8655941 to 0.8652418

Model 6 - Regularizing Movie, User and Genre Effects

This time we are going to improve the Movie, User and Genre Effects Model with regularization using a similar approach as before and our new model can be denoted as:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_i b_k^2\right)$$

```
# lambda sequence from 0 to 6 with 0.25 step.
lambdas <- seq(0, 6, 0.25)
# sapply takes values from lambdas and uses them in the function to
# find all possible RMSE values.
rmses <- sapply(lambdas, function(l){

  b_i <- train %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n()+l))

  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu)/(n()+l))

  b_k <- train %>%
    left_join(b_u, by="userId") %>%
    left_join(b_i, by = "movieId") %>%
    group_by(genres) %>%
    summarise(b_k = (sum(rating - b_i - b_u - mu))/(n()+l))

  predicted_ratings <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_k, by="genres") %>%
    mutate(b_k=replace_na(b_k,0)) %>%
    mutate(pred = mu + b_i + b_u + b_k) %>%
    pull(pred)

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

qplot(lambdas, rmses)
```




```
 # find the minimum value of lambda
lambda <- lambdas[which.min(rmses)]
lambda
```

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

We can see that the lambda that minimizes RMSE is 4.75.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie, User and Genre Effects Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.0599043
Movie Effects Model	0.9437429
Movie and User Effects Model	0.8659320
Movie, User and Genre Effects Model	0.8655941
Regularized Movie and User Effects Model	0.8652421
Regularized Movie, User and Genre Effects Model	0.8649406

The RMSE improved from 0.8652421 to 0.8649406.

Now, let's evaluate the best model with the validation data set

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

b_i <- validation %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu_evl)/(n()+lambda))

b_u <- validation %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - b_i - mu_evl)/(n()+lambda))

b_k <- validation %>%
  left_join(b_u, by="userId") %>%
  left_join(b_i, by = "movieId") %>%
  group_by(genres) %>%
  summarise(b_k = (sum(rating - b_i - b_u - mu_evl))/(n()+lambda))

predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_k, by = "genres") %>%
    mutate(b_k=replace_na(b_k,0)) %>%
    mutate(pred = mu + b_i + b_u + b_k) %>%
    pull(pred)

model_val <- RMSE(validation$rating, predicted_ratings)

model_val
```

```
## [1] 0.8391834
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie, User and Genre Effects Model on Validati
                                     RMSE = model_val))
rmse_results %>% knitr::kable()
```

method	RMSE
Base(Naive) Model	1.0599043
Movie Effects Model	0.9437429
Movie and User Effects Model	0.8659320
Movie, User and Genre Effects Model	0.8655941
Regularized Movie and User Effects Model	0.8652421
Regularized Movie, User and Genre Effects Model	0.8649406
Regularized Movie, User and Genre Effects Model on Validation Set	0.8391834

Our best model yielded the RMSE of 0.8391834 on the validation set.

Results

By going from the simplest (base) model to the more complex we can observe the improvements in the RMSE and the Regularized Movie, User and Genre Effects Model provided the lowest RMSE score with 0.8391834.

Conclusion

The main purpose of the project was the design of the recommendation system to predict movie ratings. At first glance, the 10M MovieLens dataset looks quite simple however, it contains a lot of potential for data exploration, analysis and visualization. It is a perfect option to practice data science skills and learning new concepts. The final model tested on the validation data set yielded the RMSE score of 0.8391834 which it lower than 0.86490.

Future work

There are many other methods that we could use to solve the problem, for example, matrix factorization or adding the time feature that can improve the RMSE further. More complex algorithms can be applied, like bagging and boosting, to improve the performance further.

References

Irizarry, R. A. (2020). Chapter 34.7 Recommendation systems. In Introduction to data science: Data analysis and prediction algorithms with R. Boca Raton: CRC Press.

Irizarry, R. A. (2020). Chapter 34.9 Regularization. In Introduction to data science: Data analysis and prediction algorithms with R. Boca Raton: CRC Press.