Capstone Project Report

Building a recommender system using 10M version of Movielens dataset

Endri Raco

Contents

Dedication

This project and all my work is dedicated to victims of Albanian earthquake 26 November 2019.

Acknowledgement

I would like to express my special thanks of gratitude to Prof. Rafael Irizarry for the wonderful material and thorough explanations he provided during all courses. Also I want to thank my friends of this course who share the same interests for Data Science.

Introduction

Recommender systems are information filtering tools that aspire to predict the rating for users and items, predominantly from big data to recommend their likes. Movie recommendation systems provide a mechanism to assist users in classifying users with similar interests. This makes recommender systems essentially a central part of websites and e-commerce applications. This project focuses on building a movie recommendation system using data from 10M version of movielens dataset. Several Machine Learning techniques such as Matrix Factorization, Regularization etc will be used to produce evaluation

metrics such as root mean square error (RMSE) for the movie recommender system. For all project calculations is used the following PC:

```
print("Operating System:")
```

```
[1] "Operating System:"
```

```
version
```

```
               _
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
system         x86_64, mingw32
status
major          3
minor          6.1
year           2019
month          07
day            05
svn rev        76782
language       R
version.string R version 3.6.1 (2019-07-05)
nickname       Action of the Toes
```

Importing data

MovieLense dataset contains the ratings that the users give to movies. Code used in 'Importing data' section was previously provided by edX.

Let's start by checking if needed R packages for this project are installed. If not, code below will install them.

```
# required packages for our project
if(!require(kableExtra)) install.packages('kableExtra',
repos = 'http://cran.us.r-project.org')
if(!require(dataCompareR)) install.packages('dataCompareR',
repos = 'http://cran.us.r-project.org')
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')
```

Now we are ready for data downloading:

```
# 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)
```

Some adjustments of downloaded data to have **movielens** dataframe as a result

```
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')
```

When developing an algorithm, we usually have a dataset for which we know the outcomes, as we do with the heights: we know the sex of every student in our dataset. Therefore, to mimic the ultimate evaluation process, we typically split the data into two parts and act as if we don't know the outcome for one of these. We stop pretending we don't know the outcome to evaluate the algorithm, but only after we are done constructing it. We refer to the group for which we know the outcome, and use to develop the algorithm, as the training set. We refer to the group for which we pretend we don't know the outcome as the test set. A standard way of generating the training and test sets is by randomly splitting the data. The caret package includes the function **createDataPartition** that helps us generates indexes for randomly splitting the data into training and test sets:

```
# 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)
```

We use the result of the **createDataPartition** function call to define the training and test sets like this:

```
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

And some final adjustments before cleaning environment from unused elements.

```
# 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)
```

Now, we can save the result of steps above (**edx** and **validation** dataframes) as R objects, so we can reload the final version of the data into the session for further analysis without repeating the process.

```
# Save our data as R objects
save(edx, file = 'edx.RData')
save(validation, file = 'validation.RData')
```

Describing Data

We stored the data for our project in two data frames. Let's access these datasets using the **load** function:

```
# Load data
load('edx.RData')
load('validation.RData')
```

First, we make a check if our data format is indeed **data frame**:

```
# Check format
class(edx)
```

```
[1] "data.frame"
```

```
class(validation)
```

```
[1] "data.frame"
```

Now let's take a look in our data. We start by finding out more about the structure of our **edx**:

```
as_tibble(edx) %>%
slice(1:5) %>%
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

Now for **validation**:

```
as_tibble(validation) %>%
slice(1:5) %>%
knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	480	5	838983653	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
1	586	5	838984068	Home Alone (1990)	Children|Comedy
2	151	3	868246450	Rob Roy (1995)	Action|Drama|Romance|War
2	858	2	868245645	Godfather, The (1972)	Crime|Drama

We see that **edx** data frame has 9000055 rows and 6 variables, while **validation** data frame has 999999 rows and 6.

Now let's print features of both data frames **edx** and **validation together** to reassure ourselves that both contain the same features.

```
library(dataCompareR)
comp_edx_val <- rCompare(edx, validation)
comp_summ <- summary(comp_edx_val)
comp_summ[c('datasetSummary', 'ncolInAOnly', 'ncolInBOnly', 'ncolCommon', 'rowsInAOnly', '
```

```
$datasetSummary
```

```
  Dataset Name Number of Rows Number of Columns
1          edx        9000055                 6
2   validation         999999                 6

$ncolInAOnly
[1] 0

$ncolInBOnly
[1] 0

$ncolCommon
[1] 6

$rowsInAOnly
  indices_removed
1         2071691
2         6108796
3         8926402
4         1873609
5         4286454

$rowsInBOnly
[1] indices_removed
<0 rows> (or 0-length row.names)

$nrowCommon
[1] 999999
```

It is a good idea to check for dublicates so to create a general idea about number of distinct users, movies and genres.

```
# Distinct users, movies, genres
dist_col <- edx %>%
summarize(distinct_users = n_distinct(userId),
            distinct_movies = n_distinct(movieId),
            distinct_genres = n_distinct(genres))
knitr::kable(dist_col)
```

distinct_users	distinct_movies	distinct_genres
69878	10677	797

Data Wrangling

When we printed **edx** and **validation** data frames as tibbles we noticed that we can make some arrangements in **title**, **timestamp** and **genres** columns to bring our data in a tidy format.

We are going to perform these tasks:

- Most of the movies have their **premier year** added to their **titles**. We will extract debut years in a separate column.
- Column **genres** has to be categorized. We will change the class of **genres** to **factor**
- **Timestamp** needs to be converted to **rate_year**.

For the sake of analysis we will need **userId** and **movieId** converted from class **integer** to class **factor**.

```
tidydf <- function(df){
  df$genres <- as.factor(df$genres) #Convert genres to factor
  df$timestamp <- as.Date(as.POSIXct(df$timestamp, origin='1970-01-01'))
  #Convert timestamp
  names(df)[names(df) == 'timestamp'] <- 'rate_year' # Rename column timestamp to rate_yea
  df <- df %>%
    mutate(title = str_trim(title), rate_year = year(rate_year)) %>%  #Mutate title and ra
    extract(title, c('title', 'premier_year'), regex = '(.*)\\s\\((\\d+)\\)', convert = TR
#Separate title from year
return(df)
}
# Transform our dataframes
edx <- tidydf(edx)
validation <- tidydf(validation)
```

Now our data frames look like this:

```
as_tibble(edx)
```

```
as_tibble(validation)
```

Probably is a good idea in this step to check for NA values:

```
# Check edx dataframe for NA values
edx_na <- edx %>%
filter(is.na(title) | is.na(year))
glimpse(edx_na)
```

```
Observations: 0
Variables: 7
$ userId       <int>
$ movieId      <dbl>
$ rating       <dbl>
$ rate_year    <int>
$ title        <chr>
$ premier_year <int>
$ genres       <fct>
```

```r
# Check validation dataframe for NA values
validation_na <- validation %>%
filter(is.na(title) | is.na(year))
glimpse(validation_na)
```

```
Observations: 0
Variables: 7
$ userId       <int>
$ movieId      <dbl>
$ rating       <dbl>
$ rate_year    <int>
$ title        <chr>
$ premier_year <int>
$ genres       <fct>
```

Exploring Data

Ratings frequency

From this step to the development of our algorithm we will continue using **edx** dataframe. We will come back to **validation** dataframe to perform a final test of our algorithm, predict movie ratings in the validation set as if they were unknown.

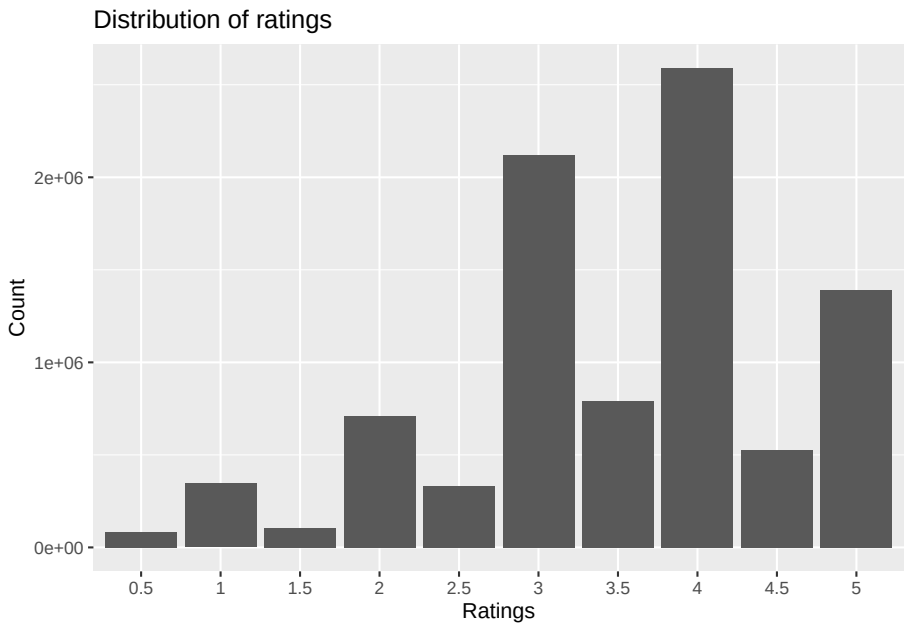
Now let's begin our exploration by looking at **rating** variable.

```r
# Check frequencies of ratings unique values
table_rating <- as.data.frame(table(edx$rating))
colnames(table_rating) <- c('Rating', 'Frequencies')
knitr::kable(table_rating)
```

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

Now, we will build a frequency plot of the ratings using **ggplot2**. For this step we need to convert **ratings** into categories:

```r
# Frequency plot of the ratings
table_rating %>% ggplot(aes(Rating, Frequencies)) +
geom_bar(stat = 'identity') +
labs(x='Ratings', y='Count') +
ggtitle('Distribution of ratings')
```



We notice from the figure that most of the ratings are above 2, and the most common is 4.

Most viewed movies

Now let's check 10 most viewed movies:

```r
# Top movies by number of views
tmovies <- edx %>% select(title) %>%
group_by(title) %>%
summarize(count=n()) %>%
arrange(desc(count))
# Print top_movies
knitr::kable(head(tmovies,10))
```

title	count
Pulp Fiction	31362
Forrest Gump	31079
Silence of the Lambs, The	30382
Jurassic Park	29360
Shawshank Redemption, The	28015
Braveheart	26212
Fugitive, The	26020
Terminator 2: Judgment Day	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	25672
Batman	24585

As we can see from the table, the most viewed movie is **Pulp Fiction** with ratings.

Ratings average

Our next task is to identify the top-rated movies by computing the **average score** for each of them.

```
# Top movies by rating average
rating_avg <- edx %>%
  select(title, rating) %>%
  group_by(title) %>%
  summarise(count = n(), avg = mean(rating), min = min(rating), max = max(rating)) %>%
  arrange(desc(avg))
# Print top_movies
knitr::kable(head(rating_avg,10))
```

title	count	avg	min	ma
Blue Light, The (Das Blaue Licht)	1	5.00	5.0	
Fighting Elegy (Kenka erejii)	1	5.00	5.0	
Hellhounds on My Trail	1	5.00	5.0	
Satan's Tango (SÃ¡tÃ¡ntangÃ³)	2	5.00	5.0	
Shadows of Forgotten Ancestors	1	5.00	5.0	
Sun Alley (Sonnenallee)	1	5.00	5.0	
Constantine's Sword	2	4.75	4.5	
Human Condition II, The (Ningen no joken II)	4	4.75	4.5	
Human Condition III, The (Ningen no joken III)	4	4.75	4.5	
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)	4	4.75	4.0	

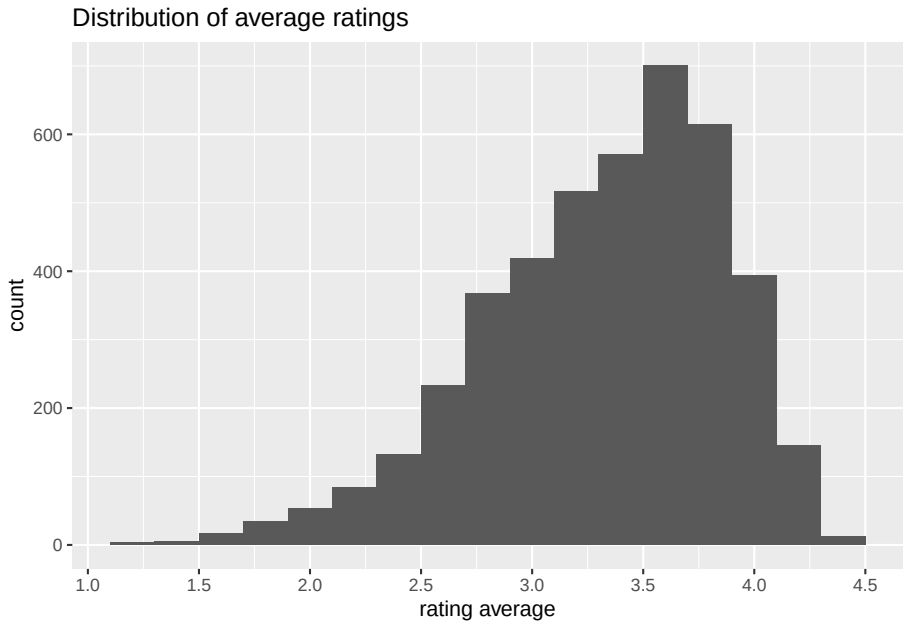
By looking at the table above we get the feel that something is wrong. We see that best average rating is for movies with a very low number of ratings (sometimes only one person has voted for this movie). Since this doesn't help getting our analysis right, we have to treat these records as outliers and exclude them from further analysis. We will remove the movies whose number of views is below a defined threshold, for instance, below 200:

```r
# Top movies by rating average
rating_avg_200 <- edx %>%
select(title, rating) %>%
group_by(title) %>%
summarise(count = n(), avg = mean(rating), min = min(rating), max = max(rating)) %>%
filter(count > 200) %>%
arrange(desc(avg))
# Print top_movies
knitr::kable(head(rating_avg_200,10))
```

title	count	avg	min	max
Shawshank Redemption, The	28015	4.455131	0.5	5
Godfather, The	17747	4.415366	0.5	5
Usual Suspects, The	21648	4.365854	0.5	5
Schindler's List	23193	4.363493	0.5	5
Casablanca	11232	4.320424	0.5	5
Rear Window	7935	4.318651	0.5	5
Sunset Blvd. (a.k.a. Sunset Boulevard)	2922	4.315880	0.5	5
Third Man, The	2967	4.311426	0.5	5
Double Indemnity	2154	4.310817	0.5	5
Paths of Glory	1571	4.308721	0.5	5

Let's built the chart:

```r
rating_avg_200 %>%
  ggplot(aes(x= avg, fill = count)) +
  geom_histogram( binwidth = 0.2) +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  labs(x='rating average', y='count') +
  ggtitle('Distribution of average ratings ')
```

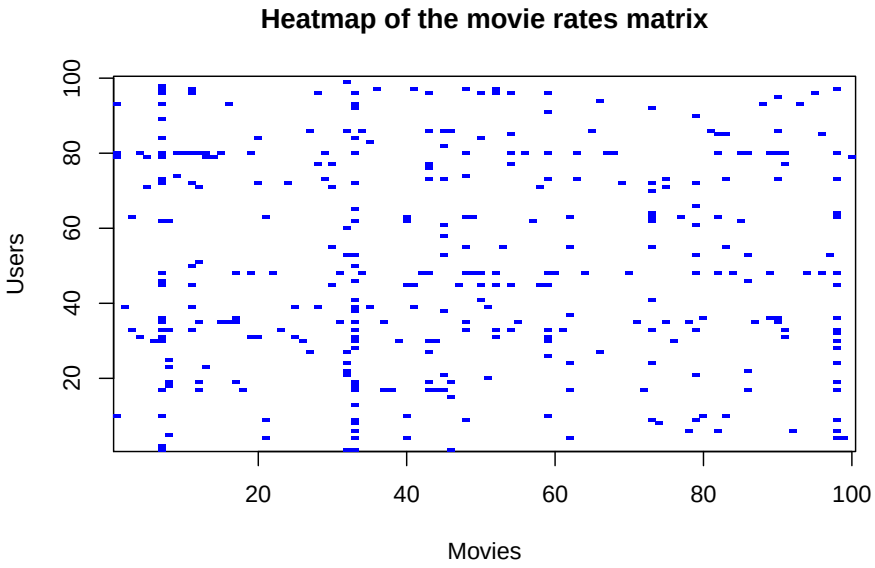


Largest number of rankings are between 3 and 4. The highest value it is around 4. It is interesting that no user has rated movies with 0s.

Data as matrix

As professor Rafael Irizarry mentioned in his book, we can think of our dataframe **edx** as a very large matrix, with users on the rows and movies on the columns. To see how sparse the matrix is, we have to convert **edx** to this format. The figure below shows the matrix for a random sample of 100 movies and 100 users with blue indicating a user/movie combination for which we have a rating

```
# We create a copy of existing edx
edx_copy <-edx
# Sample of 100 users
users <- sample(unique(edx_copy$userId), 100)
edx_copy %>% 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,., col = 'blue', xlab='Movies', ylab='Users', main = 'Heatmap of the
```

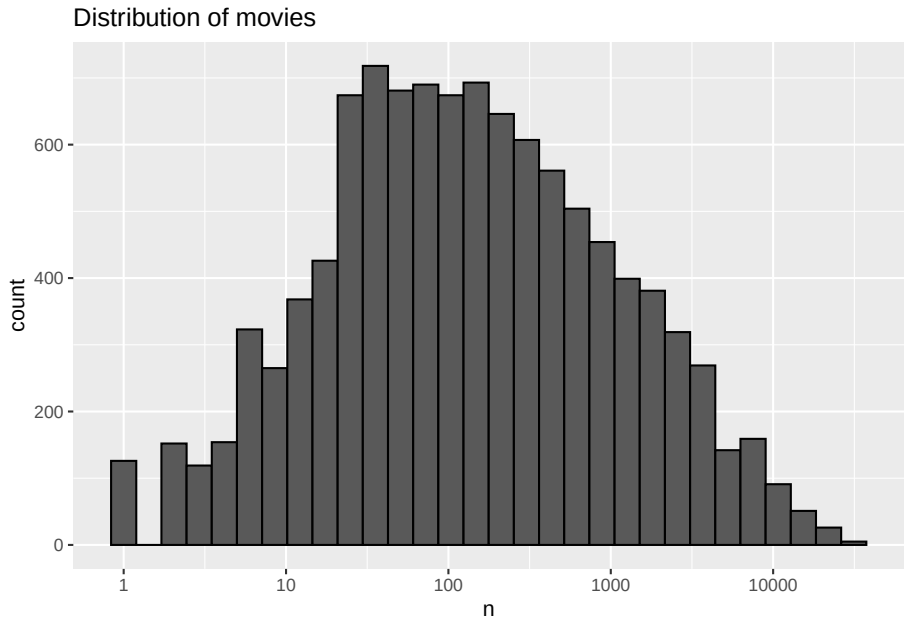


However, this chart is just displaying some random users and items. What if, instead, want to visualize only the users who have seen many movies and the movies that have been seen by many users?

To identify and select the most relevant users and movies, let's determine first:

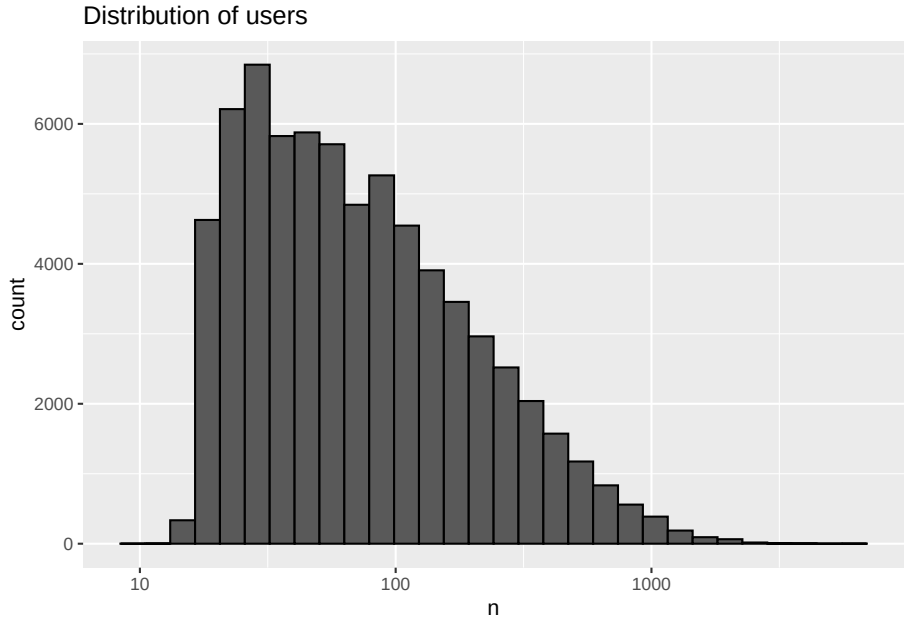
1. Some movies are rated more often than others. On the plot we can see distribution of movies based on user ratings

```
edx %>%  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = 'black') +
  scale_x_log10() +
  ggtitle('Distribution of movies')
```



2. Some users are more active than others at rating movies:

```
edx %>%  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = 'black') +
  scale_x_log10() +
  ggtitle('Distribution of users')
```



Building models

Following instruction provided in professor Rafael Irizarry course, I will start by building the simplest recommendation system:

Model 1 : Computing predicted ratings for all movies regardless of user

Our first model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

where $\varepsilon_{i,u}$ are the i.i.d errors centered at 0 and μ the **true** rating for all movies.

```
# Rating for all movies
mu_hat <- mean(edx$rating)
mu_hat
```

```
[1] 3.512465
```

If we want to assess the strength of fit, one method is to check how far off the model is for a typical case. That is, for some observations, the fitted value will be very close to the actual value, while for others it will not. The magnitude of a typical residual can give us a sense of generally how close our estimates are.Least squares fitting procedure guarantee that the mean of the residuals is zero. Thus, it makes more sense to compute **root mean squared error (RMSE)**. We will use function **RMSE** from Prof. Rafael A. Irizarry lectures.

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

```
# RMSE calculation
simple_model_rmse <- RMSE(validation$rating, mu_hat)
simple_model_rmse
```

```
[1] 1.061202
```

We see that our first evaluation for RMSE is 1.0612018 a little bit higher that 1. We will continue comparing different approaches to check if we can get a lower value for RMSE.

Below we create a results table to store all RMSE values we get in different approaches:

```
rmse_values <- tibble(method = 'Simple model RMSE', RMSE = simple_model_rmse)
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.061202

Model 2 : Computing predicted ratings for all movies based on movie effects

During data exploration, we noticed that some movies are just generally rated higher than others. We will add in our previously built simple model the term b_i to represent average ranking for movie i:

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

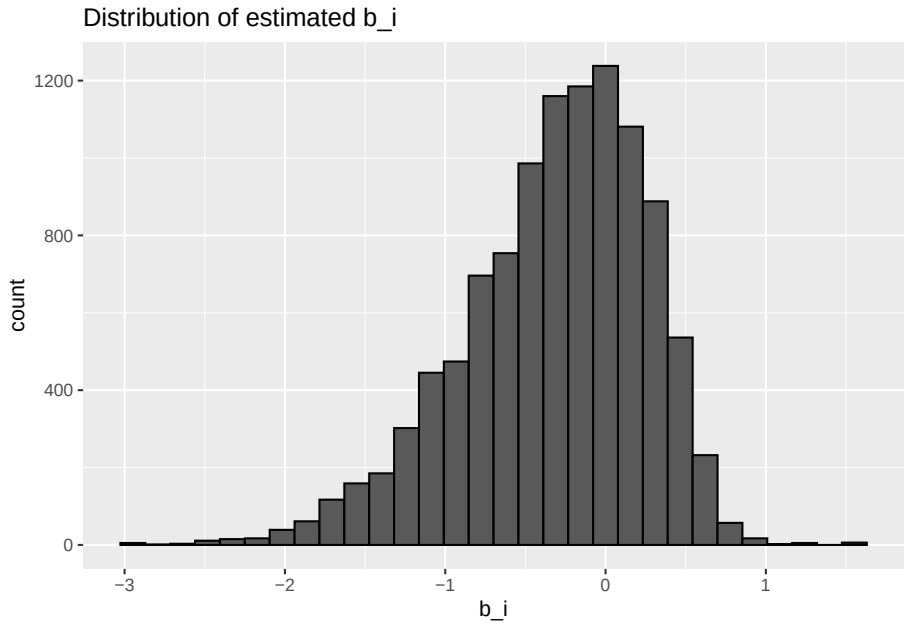
We will refer to the bs as **effects** or **bias**, movie-specific effect. We will estimate b_i-s using **least square method**. Function **lm** makes this possible, but it can be very slow so we will procced by taking Professor Rafael Irizarry's advice.

In this particular situation, we know that the least squares estimate $\hat{b}_i$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. So we can compute them this way:

```
#Compute the average of all ratings of the edx set
mu <- mean(edx$rating)
#Compute b_i
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

Let's plot now the estimated b_i-s distribution:

```
#Plot b_i distribution
movie_avgs %>%
ggplot(aes(b_i)) +
geom_histogram(bins = 30, color = 'black') +
ggtitle('Distribution of estimated b_i')
```



From the plot we can see that these estimates vary substantially.

```
# Predict bi
model_2_pred <- mu + validation %>%
left_join(movie_avgs, by='movieId') %>%
.$b_i
movie_effect_rmse <- RMSE(model_2_pred, validation$rating)
# Enter RMSE value in table
rmse_values <- bind_rows(rmse_values,
                         tibble(method='Movie Effect Model',
                                RMSE = movie_effect_rmse))
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.0612018
Movie Effect Model	0.9439087

Our model has improved with movie effect added. Let's try to get it better.

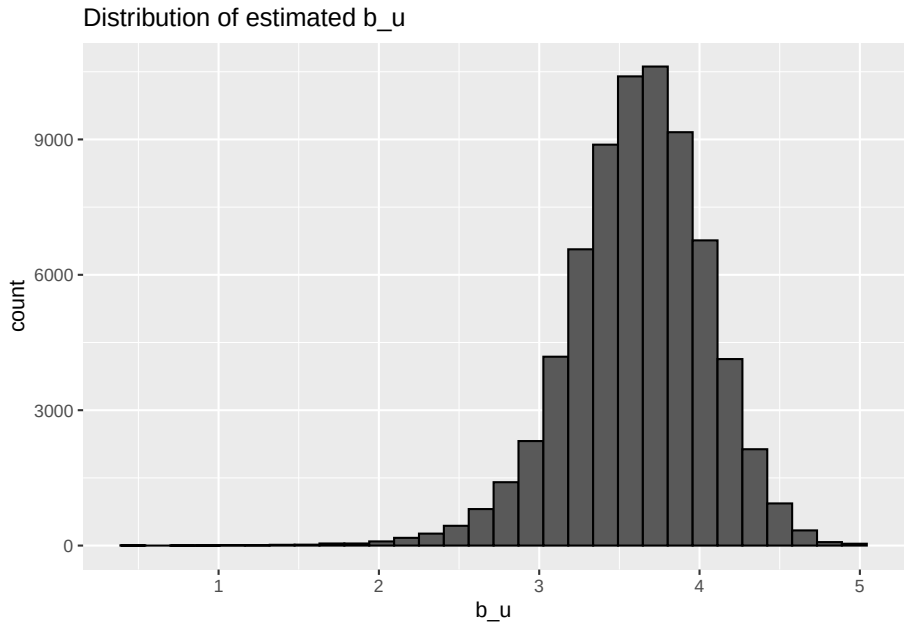
Model 3 : Computing predicted ratings for all movies based on movie and user effects

Another improvement to our model may be:

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

where b_u is a user-specific effect. Why should we think that adding user effect can lead to model improvement? Let's compute the average rating for user u for those that have rated over 100 movies and plot the estimated b_u-s distribution:

```
# Compute average rating for user u who rated more than 100 movies
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = 'black') +
  ggtitle('Distribution of estimated b_u')
```



We notice that there is substantial variability across users as well so we proceed with model fitting. Let's compute an approximation by computing $\hat{\mu}$ and $\hat{b}_i$ and estimating $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$:

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

Now let's see if RMSE improves:

```
# Predicted ratings
model3_pred <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
movie_user_effect <- RMSE(model3_pred, validation$rating)
rmse_values <- bind_rows(rmse_values,
                         tibble(method='Movie + User Effects Model',
                                RMSE = movie_user_effect))
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488

Our RMSE is lower now but still not at the target we are looking for.

Regularization

For any machine learning problem, essentially, we can divide data points into two components — pattern + stochastic noise. The goal of machine learning is to model the pattern and ignore the noise. Probably our algorithm is trying to fit the noise in addition to the pattern, so we are dealing with **overfitting**.

Maybe we can hold back and check if our algorithm is fitting the noise. Let's explore problems in our first model, using only movie effects b_i following Prof. Rafael Irizarry approach.

```
validation %>%
left_join(movie_avgs, by='movieId') %>%
mutate(residual = rating - (mu + b_i)) %>%
arrange(desc(abs(residual))) %>%
select(title, residual) %>% slice(1:10)
```

We notice some large predictions for many of movies in the above table. Let's look at the top 10 worst and best movies based on b_i. First, let's create a database that connects **movieId** to movie **title**:

```
# merged database
merge_db <- edx %>%
select(movieId, title) %>%
distinct()
```

Here are the 10 best movies according to our estimate and how often they were rated:

```
# top 10 best movies based on b_i
movie_avgs %>% left_join(merge_db, by="movieId") %>%
arrange(desc(b_i)) %>%
```

```
select(title, b_i) %>%
slice(1:10)
```

title	b_i
Hellhounds on My Trail	1.487535
Satan's Tango (SÃ¡tÃ¡ntangÃ³)	1.487535
Shadows of Forgotten Ancestors	1.487535
Fighting Elegy (Kenka erejii)	1.487535
Sun Alley (Sonnenallee)	1.487535
Blue Light, The (Das Blaue Licht)	1.487535
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)	1.237535
Human Condition II, The (Ningen no joken II)	1.237535
Human Condition III, The (Ningen no joken III)	1.237535
Constantine's Sword	1.237535

```
validation %>% count(movieId) %>%
left_join(movie_avgs) %>%
left_join(merge_db, by="movieId") %>%
arrange(desc(b_i)) %>%
select(title, b_i, n) %>%
slice(1:10)
```

title	b_i	n
Hellhounds on My Trail	1.4875348	1
More	1.2018205	1
Valerie and Her Week of Wonders (Valerie a tÃ½den divu)	0.9875348	1
Kansas City Confidential	0.9875348	1
Shawshank Redemption, The	0.9426660	3111
Red Desert, The (Deserto rosso, Il)	0.9042015	1
Godfather, The	0.9029008	2067
Man Who Planted Trees, The (Homme qui plantait des arbres, L')	0.8875348	2
Usual Suspects, The	0.8533885	2389
Schindler's List	0.8510281	2584

And here are the 10 worst. Let's look at how often they are rated.

```
# top 10 worse movies based on b_i
movie_avgs %>% left_join(merge_db, by="movieId") %>%
arrange(b_i) %>%
select(title, b_i) %>%
slice(1:10)
```

title	b_i
Besotted	-3.012465
Hi-Line, The	-3.012465
Accused (Anklaget)	-3.012465
Confessions of a Superhero	-3.012465
War of the Worlds 2: The Next Wave	-3.012465
SuperBabies: Baby Geniuses 2	-2.717822
Hip Hop Witch, Da	-2.691037
Disaster Movie	-2.653090
From Justin to Kelly	-2.610455
Criminals	-2.512465

```
knitr::kable(movie_avgs)
```

63350	-0.0124652
3370	-0.9588938
63393	-1.9670197
3371	-1.3249652
63433	-0.3893867
3372	-0.0827777
63476	-0.8019389
3373	-0.4943078
63446	-0.5124653
3374	-1.8249652
63478	-0.0124652
3375	-0.1261016
63468	-0.5124653
3376	-0.7566912
63479	-0.4707985
3377	-0.2321159
63481	-0.2624652
3378	-0.3961293
63515	-0.5124653
3379	-0.2253030
63540	-0.1033743
3380	-0.0950034
movie40	
63617	-0.0124652
3381	-0.4053795
63645	-0.6543045
3382	-0.3074858
63647	-0.4875348
3383	-0.3054811
63662	-0.4875348
3384	-0.0289439
63676	-0.4875348
3385	-0.4437933
63688	-0.0124652
3386	-0.3028194
63698	-0.2375348
3387	-0.8348739
63760	-0.4875348
3388	-0.3278732
63780	-0.0124652
3389	-0.9946804
63773	-0.7375348
3390	-0.0867405
63781	-0.5124653
3391	-0.7440052
63765	-0.0124652
3392	-0.9496597
63791	-0.5124653
3393	-0.3270035
63793	-0.0124652
3394	-0.0692957
63806	-0.0124652
3395	-0.7905812
63808	-0.1543045
3396	-0.2342259
63819	-0.0124652
3397	-0.4744473
63826	-0.4875348
3398	-0.4236346
63838	-0.0124652
3399	-0.9324982
63836	-0.6784719
3400	-0.0399990
63840	-0.4875348
3401	-0.0849078
63853	-0.2606081
3402	-0.4748282
63859	-0.0224485
3403	-0.3457203
63862	-0.2624652
3404	-0.2800843
63876	-0.6303010
3405	-0.9822079
63900	-0.2624652
3406	-0.1671817
63903	-0.2624652
3407	-0.4296432
64080	-0.8917755
3408	-0.6749662
64033	-0.0124652
3409	-0.4542268
64034	-0.2067500
3430	-0.4286300
64114	-0.5124653
3431	-0.2567123
64116	-0.0124652
3432	-0.3022278
64153	-0.0124653
3433	-0.4420978
64197	-0.3386650
3434	-0.1931892
64229	-0.4124653
3435	-0.0940748
64231	-0.0124653
3436	-0.4426906
64237	-0.5124653
3437	-0.6278498
64248	-0.0124653
3438	-0.0920597
64245	-0.0124653
3439	-0.8550245
64240	-0.3974358
3440	-0.7100209
64273	-0.2624652
3441	-0.3393586
64275	-0.4875348
3442	-0.4705229
64278	-0.4875348
3443	-0.0029909
64280	-0.7375348
3444	-0.4267886
64288	-0.0124652
3445	-0.1955047
64285	-0.0124652
3446	-0.9498881
64288	-0.5124653
3447	-0.5342748
64331	-0.2375348
3448	-0.6034229
64326	-0.5124652
3449	-0.2333304
64327	-0.5124652
3450	-0.8353885
64338	-0.0675248
3451	-0.9393421
64365	-0.5124653
3452	-0.4291730
64368	-0.1701319
3453	-0.6792417
64403	-0.0124652
3454	-0.6294245
64439	-0.2624652
3455	-0.3023827
64418	-0.9875348
3456	-0.6933920
64427	-0.2875348
3457	-0.4893333

```r
validation %>% count(movieId) %>%
left_join(movie_avgs) %>%
left_join(merge_db, by="movieId") %>%
arrange(b_i) %>%
select(title, b_i, n) %>%
slice(1:10)
```

title	b_i	n
Confessions of a Superhero	-3.012465	1
War of the Worlds 2: The Next Wave	-3.012465	1
SuperBabies: Baby Geniuses 2	-2.717822	5
Disaster Movie	-2.653090	8
From Justin to Kelly	-2.610455	17
Criminals	-2.512465	2
Mountain Eagle, The	-2.512465	2
When Time Ran Out... (a.k.a. The Day the World Ended)	-2.512465	2
PokÃ©mon Heroes	-2.483268	19
Roller Boogie	-2.479132	2

Penalized least squares

The general idea behind regularization is to constrain the **total variability** of the effect sizes. We make this possible by adding a **penalty term** to the least square equation. This **penalty term** gets larger when b_i are large. So we consider **penalty term** a parameter which we have to tune to get optimal results.

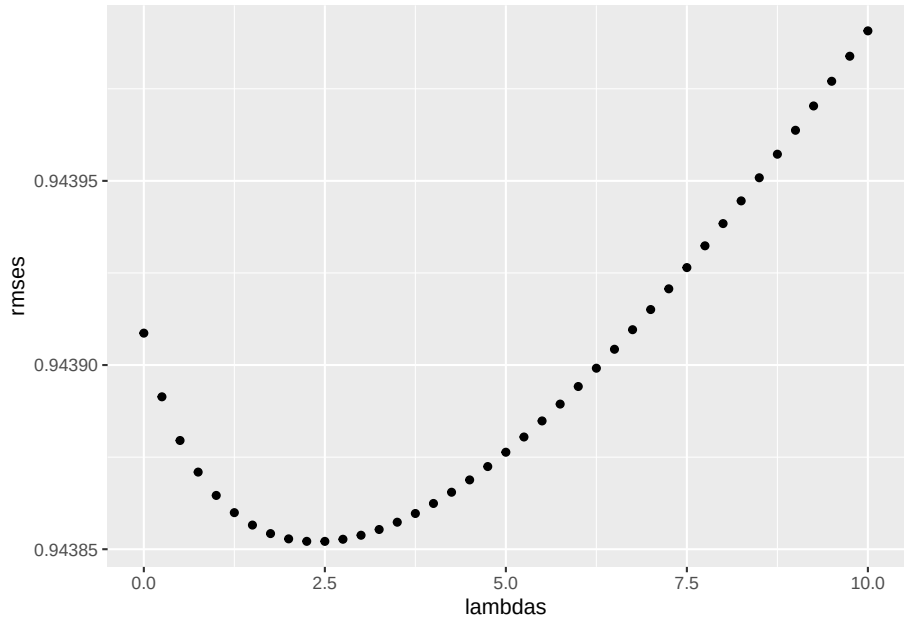
We will use λ is a tuning parameter. We can use cross-validation to choose it.

```r
lambdas <- seq(0, 10, 0.25)
mu <- mean(edx$rating)
just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
```

Now we plot RMSE values together with lambdas:

```r
# Plot lambdas and rmse
ggplot(data.frame(lambdas = lambdas, rmses = rmses ), aes(lambdas, rmses)) +
  geom_point()
lambdas[which.min(rmses)]
```

```
[1] 2.5
```



We can use regularization for the estimate user effects as well. We are minimizing:

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

The estimates that minimize this can be found similarly to what we did above. Here we use cross-validation to pick a λ:

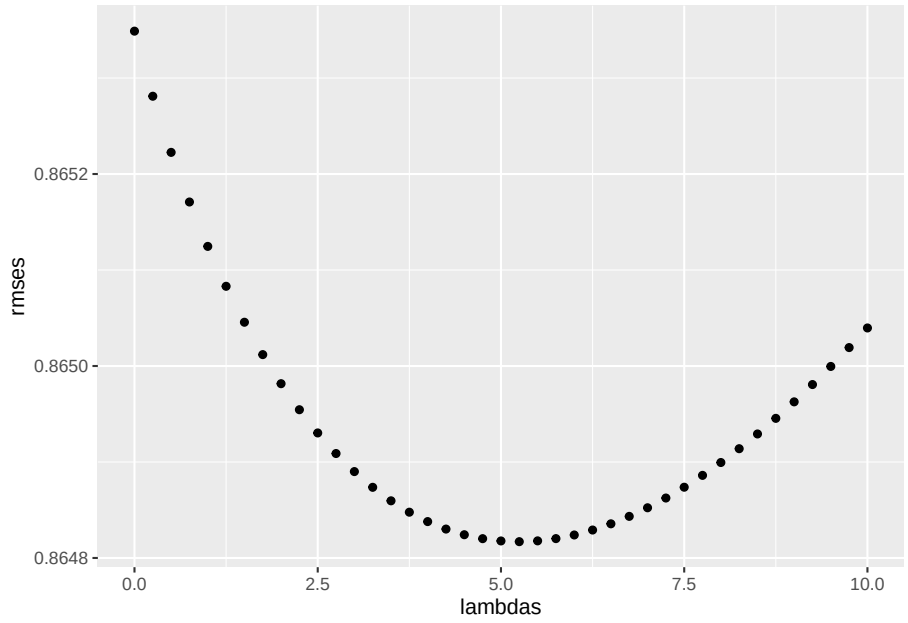
```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
mu <- mean(edx$rating)

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 - b_i - mu)/(n()+l))
    predicted_ratings <-
    validation %>%
    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, validation$rating))
})
```

```
ggplot(data.frame(lambdas = lambdas, rmses = rmses ), aes(lambdas, rmses)) +
geom_point()
```



For the full model, the optimal λ is:

```
# Value of lambda that minimizes  RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

```
[1] 5.25
```

method	RMSE
Simple model RMSE	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648170

Results and conclusions

Finally we print again table of RMSE values for all models we build during this work:

method	RMSE
Simple model RMSE	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648170

We can see that lowest value RMSE we could achive so far is which is lower than our starting goal (0.8775). **movieId** variable has a large impact on the **rmse** value. When we combined this impact with **usedId** effect, **rmse** value became smaller.

So our final model will be:

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