MOVIELENS PROJECT

Pulapa Sai Prasanth

24/02/2021

Introduction section

The MovieLens dataset is a database with over 10 million ratings for over 10,000 movies by more than 72,000 users. The dataset includes the identification of the user and the movie, as well as the rating, genre, and the timestamp. No demographic information is included.

The goal of this project is to predict movie ratings. To do that, the dataset was classified into two: the train and validation set. The validation set is 10% of the original data and is not used in the construction of the model.

Due to the large size of the dataset, usual data wrangling (for example, the *lm* model) was not possible because of memory allocation. As the dataset is very sparse, we included regularization in the model.

The goal of this project is to predict movie ratings.

In this project, the aim is to create a model of movie rating with the movielens data provided. The challenge is to create the model with a RMSE < 0.86490.

First, Let's download all the data sets, Libraries and packages. Then we will create a data partition of the movielens ratings, which 90 % of that partition will be the training set (edx) and 10 % of that partition will be the test set (Validation).

```
############################################################
# Create edx set, validation set, and submission file
############################################################
# Note: this first code chunk was provided by the course
# Note: this process could take a couple of minutes

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

## Loading required package: tidyverse

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

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.6     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- 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

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

## Loading required package: data.table

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

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

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

# 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 3.6 or earlier
movies <- as.data.frame(movies) %>% mutate(movieId =
```

```
as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(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")

### DATA PARTITION

# Validation set will be 10% of MovieLens data

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

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

# if using R 3.5 or earlier, use `set.seed(1)` instead
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)

# Clean up memory by deleting unsused objects and performing a garbage
collection
rm(dl, ratings, movies, test_index, temp, movielens, removed)
gc()

##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2346304 125.4   15160933  809.7  22280564 1190.0
## Vcells 74233774 566.4  222295398 1696.0 208939749 1594.1
```

DATA ANALYSIS AND EXPLORATION

Let's have a quick overview of the data , here we looking at all the variables to identify key variable for our model prediction.

```
dim(edx)
```

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

```
head(edx)
```

```
##    userId movieId rating timestamp title genres
## 1:      1     122      5 838985046  <NA>   <NA>
## 2:      1     185      5 838983525  <NA>   <NA>
## 3:      1     292      5 838983421  <NA>   <NA>
## 4:      1     316      5 838983392  <NA>   <NA>
## 5:      1     329      5 838983392  <NA>   <NA>
## 6:      1     355      5 838984474  <NA>   <NA>
```

Analysis section

As explained before, due to the size of the dataset, modeling the data using a function like *lm* is not appropriate. Now Let's count all the movies in the data set and have an overview and tendencies.

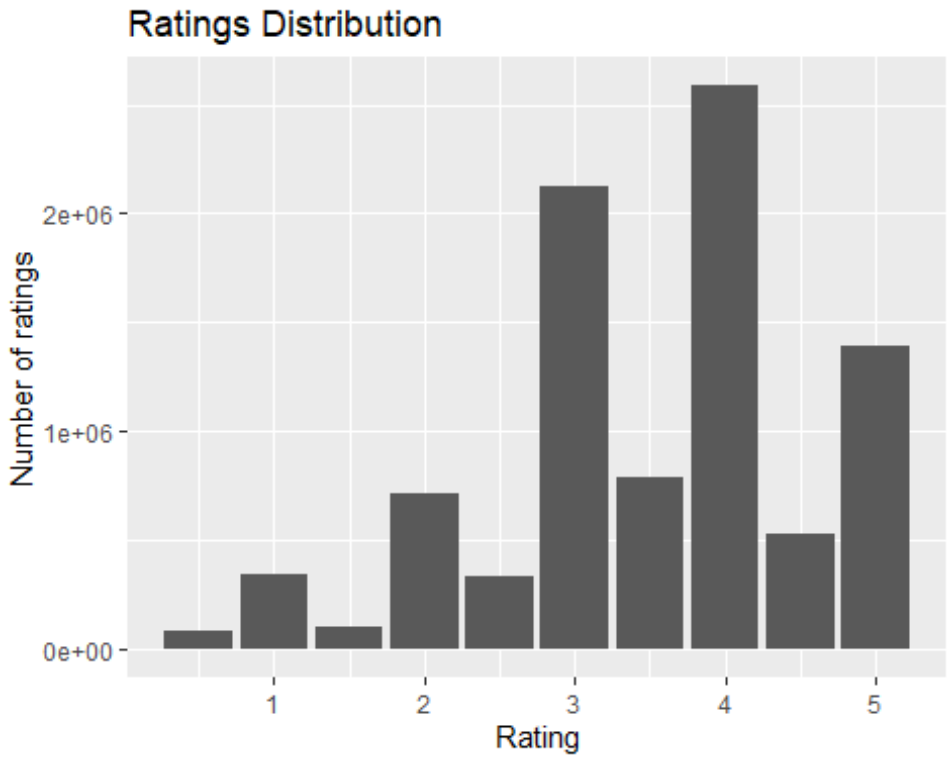
```
# let's group all movies by movieID
edx_movies <- edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
# let's have an overview distribution of movies in the data set
summary(edx_movies$count)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    30.0   122.0   842.9   565.0 31362.0
```

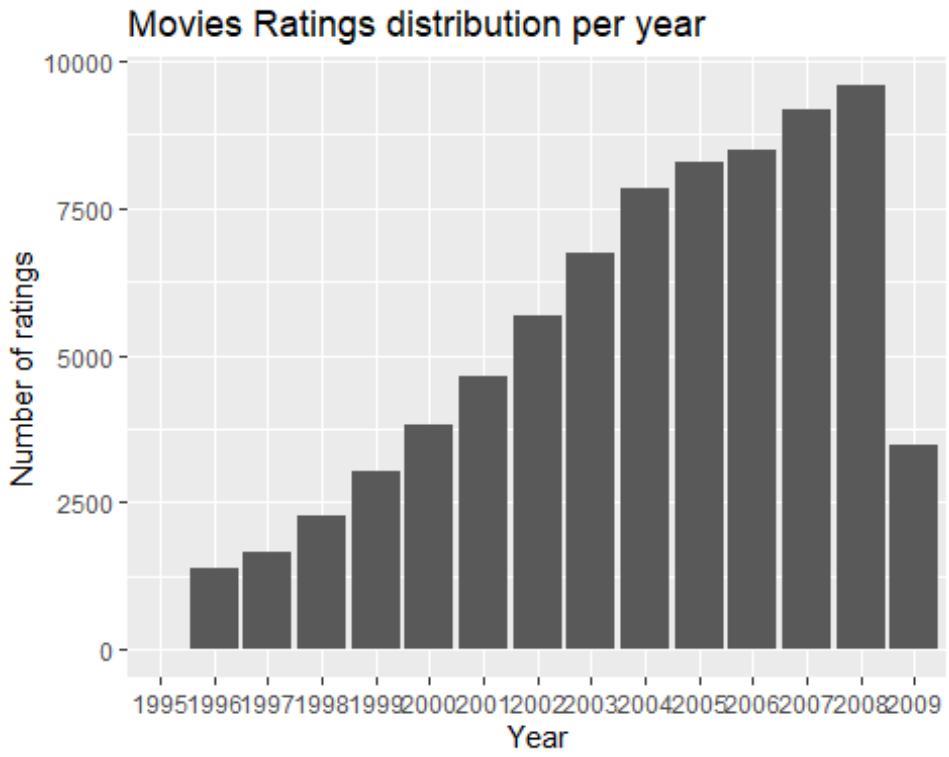
Here we can see that one movie was rated 31362 times. Also 122 movies represent half of the ratings Movies was rated 842.9 in averages

Now, let's visualize the ratings distribution in the data set and also the movie rating distribution per year.

```
# ratings Distribution
ggplot(data = edx, aes(x = rating)) +
  geom_bar() +
  labs(title = "Ratings Distribution", x = "Rating", y = "Number of ratings")
```




```
# Movie Ratings distribution  per year
movies_year <- edx %>%
  transform(timestamp = format(as.POSIXlt(timestamp, origin = "1970-01-01"),
"%Y")) %>%
  select(timestamp, movieId) %>%
  group_by(timestamp) %>%
  summarise(count = n_distinct(movieId))
ggplot(data = movies_year, aes(x = timestamp, y = count)) +
  geom_bar(stat = "identity") +
  labs(title = "Movies Ratings distribution per year", x = "Year", y =
"Number of ratings")
```



-FINDING THE MODEL We are starting with a model , assuming that all movies in the trainig set have equal ratings. Then the formula for that model will be : Ymu,i= u + ϵ u,i Here u represent the average rating for all movies and users in edx , and ϵ represent all errors (in this model we are minimizing ϵ). Now we can compute the average ratings on edx (u), test it into the validation set and predict the RMSE.

```
 # we calculate the overall average rating on the training dataset
u <- mean(edx$rating)
# Here is the formula of RMSE
RMSE <- function(true_ratings = NULL, predicted_ratings = NULL) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
# Calculate RMSE using validation ratings

  RMSE(validation$rating, u)
```

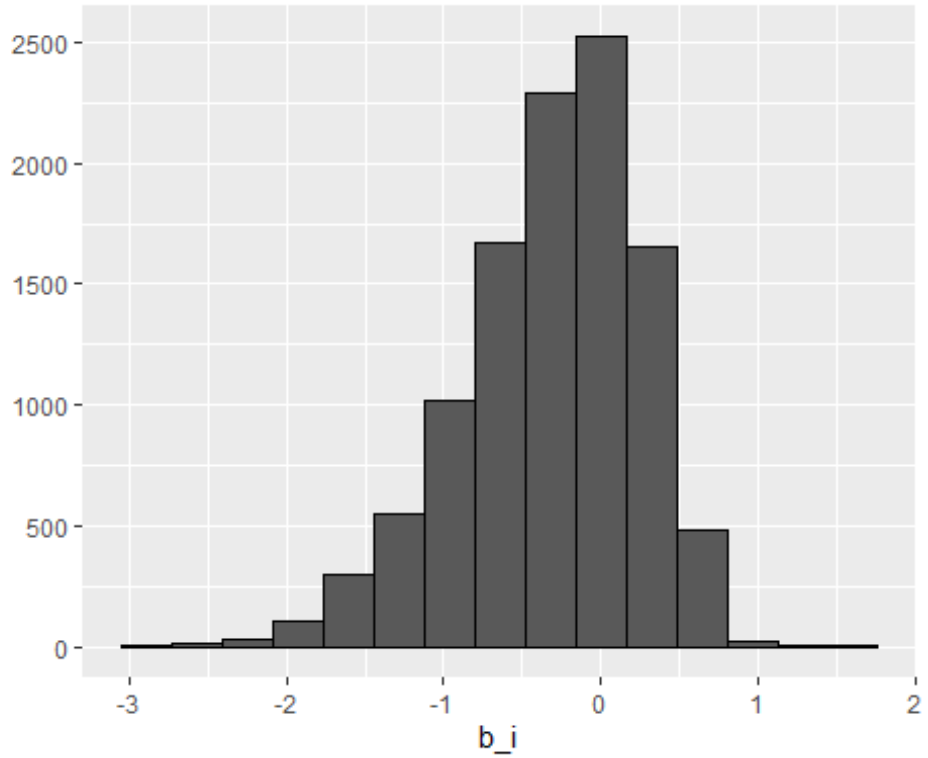
```
## [1] 1.061202
```

This model give us a RMSE of 1.06.

MOVIE EFFECT

WE can Optimize our model by including the movie effect. lets call bi the average rating of movie i. To calculate the new model we can use the formula : Yu, i = u + bi + ϵui if rearange the formula and isolate bi we will have : bi = Yu,i - u. This means we can calculate bi by substracting the overall average of each movie rating with the overall average rating of all movies.

```r
# caluclate b_i for each movie and let's compare it  with the overall average
u on training dataset
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - u))
#  Lets add b_i into the validaation set and lets predict all unknown ratings
with u and b_i
predicted_ratings <- validation %>%
  left_join(b_i, by='movieId') %>%
  mutate(pred = u + b_i) %>%
  pull(pred)
#  calculate RMSE of movie ranking effect
RMSE(validation$rating, predicted_ratings) # 0.94 still not good enough

## [1] 0.9439087

# plot the distribution of b_i's
qplot(b_i, data = b_i, bins = 15, color = I("black"))
```



Besides the movie effect, we also assume that some users rate movies higher than others, so the next model considers both the movie and the user effect. We estimate the user effect as the average of the ratings per user.

MOVIE EFFECT AND USER EFFECT

We can include the user effect (bu) into the model to optimize it. Yu, i = μ + bi + bu + ϵμ, i we can then compute bu based on the formula above

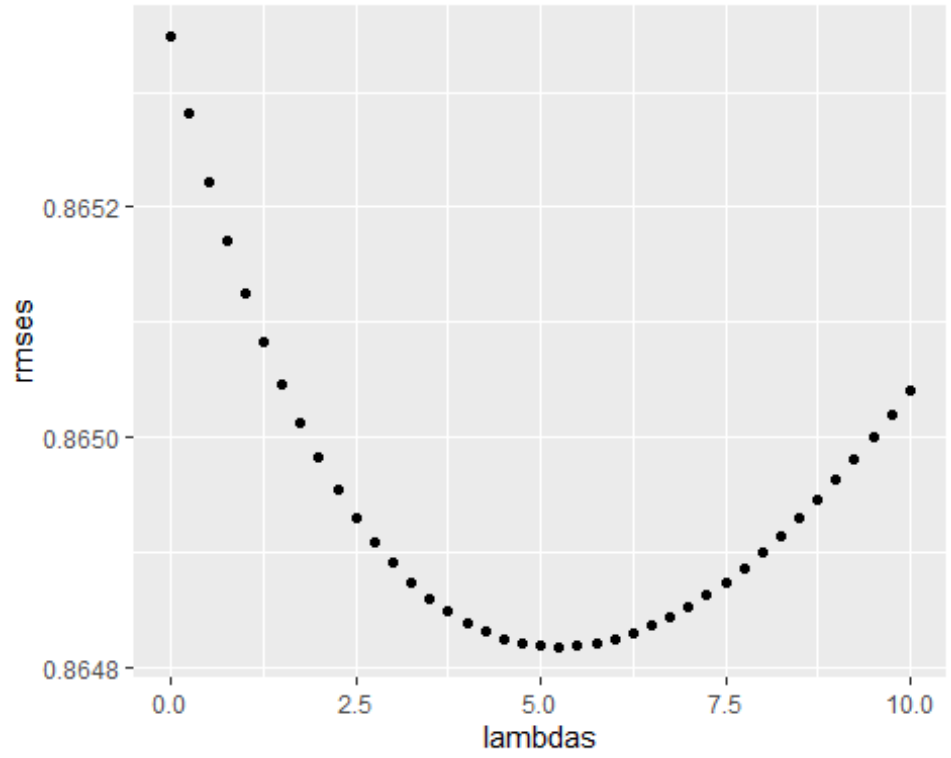
```r
# lets train the model with movie effect (b_i) and users effect (b_u)
# let's find b_u
b_u <- edx %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - u - b_i))
# predict new ratings with movie and user bias
predicted_ratings <- validation %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = u + b_i + b_u) %>%
  pull(pred)
# calculate RMSE of movie ranking effect
RMSE(predicted_ratings, validation$rating) # 0.8653488 we getting close
```

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

LETS TRAIN THE MODEL WITH THE BEST REGULARIZATION FACTOR LAMBDA

```r
# lets optimized movie and user effect method with the best regularization
factor (lamba)

  # let's determine the best lambda from a sequence

  lambdas <- seq(from=0, to=10, by=0.25 )

  # output RMSE of each lambda, repeat earlier steps (with regularization)

  rmses <- sapply (lambdas, function(l) {

    # calculate average rating across training data
    u <- mean(edx$rating)

    # compute regularized movie bias term
    b_i <- edx %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - u)/(n()+l))

    # compute regularize user bias term
    b_u <- edx %>%
      left_join(b_i, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - u)/(n()+l))

    # compute predictions on validation set based on these above terms
    predicted_ratings <- validation %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = u + b_i + b_u) %>%
      pull(pred)
```

```
    # output RMSE of these predictions
    return(RMSE(predicted_ratings, validation$rating))
  })
  # quick plot of RMSE vs lambdas
  qplot(lambdas,rmses)
```




```
  # print minimum RMSE
  min(rmses)
```

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

Results section

Final model with the best fitted lambda

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

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - u)/(n()+lam))
  # compute regularize user bias term
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - u)/(n()+lam))

  # compute predictions on validation set based on these above terms
```

```r
  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = u + b_i + b_u) %>%
    pull(pred)

  # Let's find the RMSE based on the above terms
  RMSE(predicted_ratings, validation$rating)
```

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

Conclusion section

This project's goal was to predict movie ratings from a database with over 10 million evaluations. To do that, we considered the impact of movies, users and genres to the ratings. We divided the dataset into train and validation to avoid redundancy.

As the dataset was large, usual data wrangling was not possible in most computers due to memory allocation.

It would have been interesting to have more information about the users (e.g. age and gender) and the movies (e.g. actors, director and language) to try to improve the model.