Rating Prediction of Movielens Report

Walaa Aburaad

17/11/2021

1. SUMMARY

MovieLens helps you find movies you will like. Rate movies to build a custom taste profile, then MovieLens recommends other movies for you to watch.

The version of movielens here is just a small subset of a much larger dataset with millions of ratings. We will create our own recommendation system using all the tools we have learned throughout the courses in this Data Science Program courses.

We will use the 10M version of the MovieLens dataset. We will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the test set. And this project try to generate a model with enough predictive power to know the rating that a user will give to a movie.

At this project, The movie rating predictions will be compared to the true ratings in the validation set (the final hold-out test set) using RMSE.

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

Where: N is the number of user-movie combinations $y_{u,i}$ is the true rating for movie (i) by user (u). $\hat{y}_{u,i}$ is the prediction rating.

The target is to reach RMSE less than 0.86490.

Code provided by the edx staff to download and create *edx* dataset.

```
#Load libraries
#Create edx set, validation set
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")
#Load the data
# 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(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)
```

1.1 Tha dataset edx

```
## [1] "The edx dataset has 9000061 rows and 6 columns."
```

As we see below, we have six variables **userId**, **movieId**, **rating**, **timestamp**, **titles** and **genres**.

```
str(edx)

## Classes 'data.table' and 'data.frame':   9000061 obs. of  6
variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392
838983392 838984474 838983653 838984885 838983707 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb &
Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy"
"Action|Drama|Sci-Fi|Thriller" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Number of unique movies and users in the edx dataset

```
edx %>%
  summarize(
    n_movies= n_distinct(edx$movieId),
    n_users=n_distinct(edx$userId),
    n_genres=n_distinct(edx$genres))

##   n_movies n_users n_genres
## 1    10677   69878      797
```

The dataset modified formats

The **userId** and **movieId** variables are `numeric` columns in the original data set. These characteristics are just *labels*, therefore they will be converted to *factor* type to be useful.

Both **movieId** and **title** variables give us the same exact information. They are the **unique identification code** to each film. Only the `movieId` colum will remain. It will be a *factor* too.

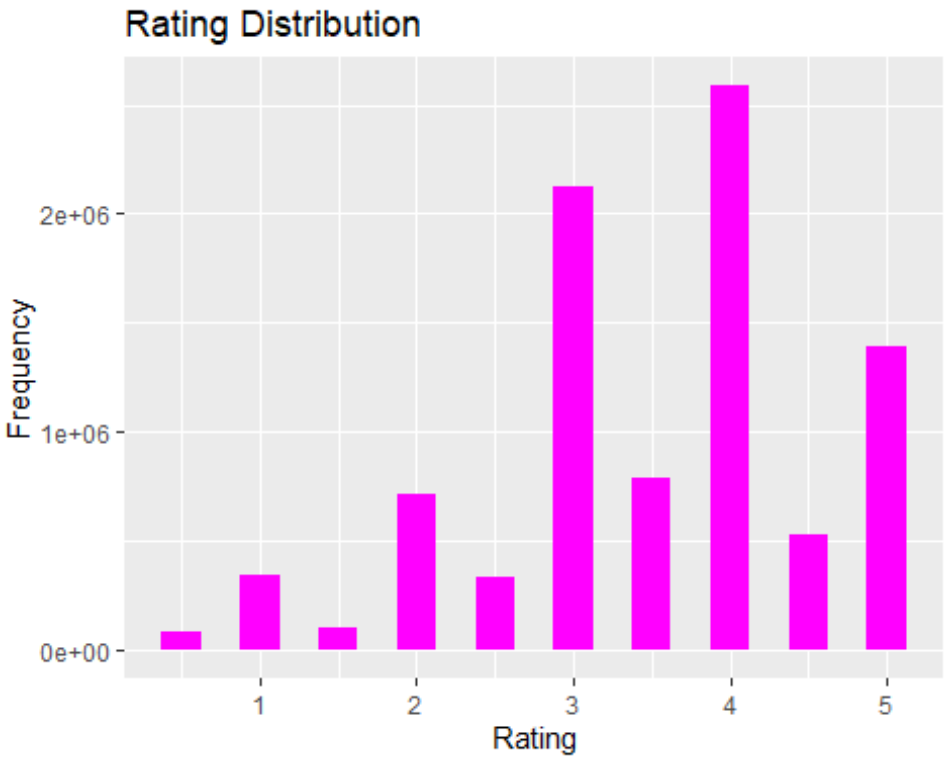
The **timestamp** variable is converted to *POSIXct* type, to be handle correctly as a `date` vector. The year is extracted to the **year** column and the **timestamp** column is dropped.

```
edx$userId <- as.factor(edx$userId)#converts 'userId' to factor
edx$movieId <- as.factor(edx$movieId)#converts 'movieId' to factor
edx$genres <- as.factor(edx$genres)#converts 'genres' to factor
edx$timestamp <- as.POSIXct(edx$timestamp, origin = "1970-01-
01")#converts 'timestamp' to POSIXct
edx <- edx %>%
  mutate(year = as.numeric(str_sub(title,-5,-2)))#extracts the release
year of the movie
edx <- edx %>%
  mutate(year_rate=year(timestamp))#extracts the year that the rate was
given by the user
```

1.2 Data Analysis

Rating Distribution

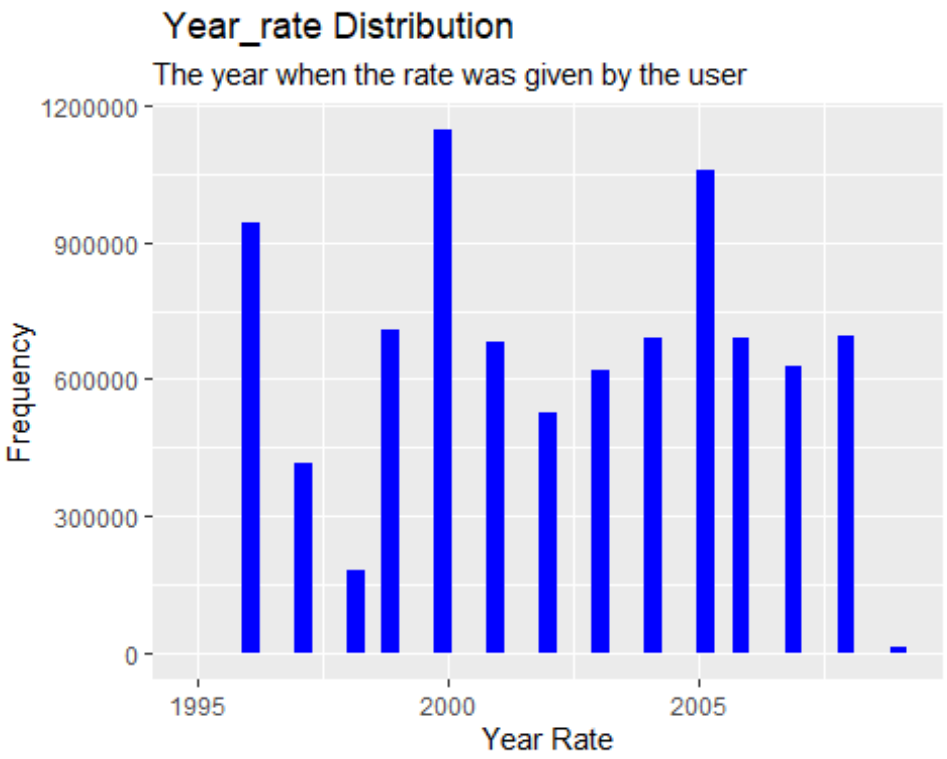
```
edx%>%
  ggplot(aes(rating))+
  geom_histogram(binwidth=0.25 , fill="magenta")+
  labs(title = "Rating Distribution",
       x="Rating",
       y="Frequency")
```



In general, we can notice that **4** and **3** ratings have the highest frequency than others. Also, half star ratings are less common than whole star ratings.(eg. there are fewer ratings of 3.5 than there are rating of 3 or 4, etc.)

Year_rate Distribution

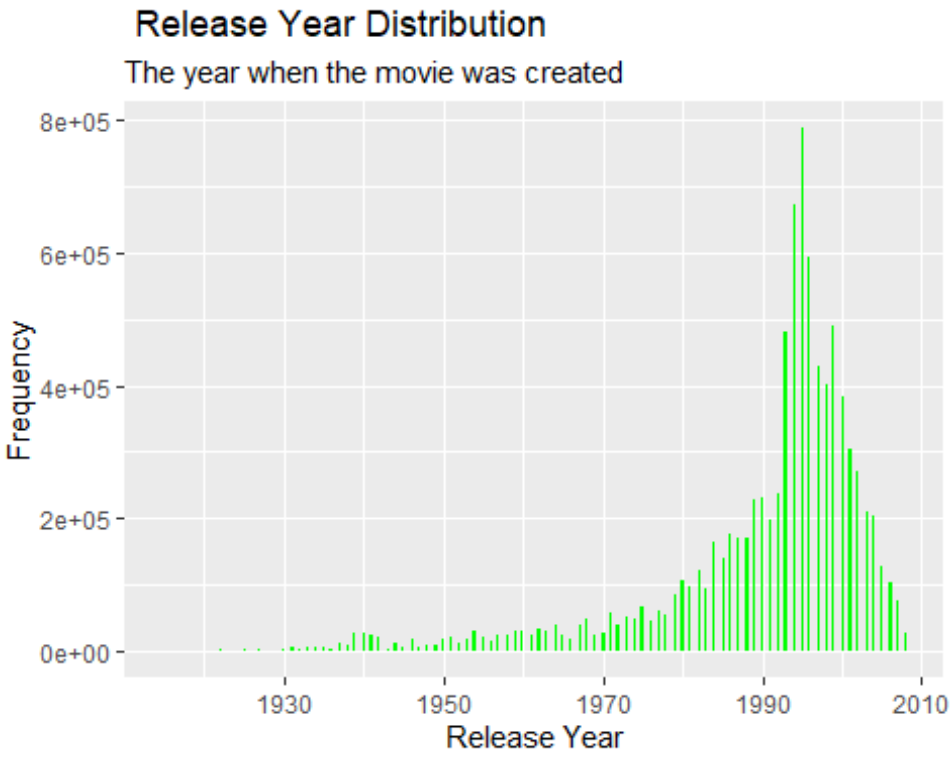
```
edx%>%
  ggplot(aes(year_rate))+
  geom_histogram(binwidth=0.35 , fill="blue")+
  labs(title = " Year_rate Distribution",
       subtitle = "The year when the rate was given by the user",
       x="Year Rate",
       y= "Frequency")
```



We can notice that the years of 1997, 1998 and 2009 have the lowest observations. Whereas the years of 1996, 2000 and 2005 have the highest frequency of observations.

Release_year Distribution

```
edx%>%
  ggplot(aes(year))+
  geom_histogram(binwidth=0.3 , fill="green")+
  labs(title = " Release Year Distribution",
       subtitle = "The year when the movie was created",
       x="Release Year",
       y= "Frequency")
```



In general, the movies which had been created between the years of (1990 and 2010) are the most liked movie by users. So they have been rating higher than the others.

The list of the highest five genres rating genres

Drama Movies are the most type of movies liked by users as see below:

```
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))%>%
  head(5)
```

```
## # A tibble: 5 x 2
##   genres      count
##   <chr>       <int>
## 1 Drama     3909401
## 2 Comedy    3541284
## 3 Action    2560649
## 4 Thriller  2325349
## 5 Adventure 1908692
```

The list of the first ten movies of highest movie rating

Pulp Fiction movie is the highest rating movie with 31336 count.

```
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count))%>%
  head(10)
```

```
## `summarise()` has grouped output by 'movieId'. You can override
using the `.groups` argument.
```

```
## # A tibble: 10 x 3
## # Groups:   movieId [10]
##    movieId title
count
##    <fct>   <chr>
<int>
##  1 296     Pulp Fiction (1994)
31336
##  2 356     Forrest Gump (1994)
31076
##  3 593     Silence of the Lambs, The (1991)
30280
##  4 480     Jurassic Park (1993)
29291
##  5 318     Shawshank Redemption, The (1994)
27988
##  6 110     Braveheart (1995)
26258
##  7 589     Terminator 2: Judgment Day (1991)
26115
##  8 457     Fugitive, The (1993)
26050
##  9 260     Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)
(1977) 25809
## 10 592     Batman (1989)
24343
```

2. DATA MODELING

2.1 Train and Test Set

We will create 'train' and 'test' set.

```
edx <- edx %>% select(userId, movieId, rating)
test_index <- createDataPartition(edx$rating, times = 1, p = .2, list =
F)
train <- edx[-test_index, ] # Create Train set
  test <- edx[test_index, ] # Create Test set
  test <- test %>% # The same movieId and usersId appears in both set.
    semi_join(train, by = "movieId") %>%
    semi_join(train, by = "userId")
```

2.2 Baseline Model

The mean of the movie rating effect model `rating` ($\hat{y}_i$):

$$\hat{y}_{u,i} = \mu + \varepsilon_{u,i}$$

where ε is independent errors.

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

## [1] 3.512492

#Check the test result
navie_rmse<-RMSE(test$rating, mu)
navie_rmse

## [1] 1.060006

#Save the result in table
rmse_results<- tibble(method="Mean of rating model",RMSE = navie_rmse)
rmse_results %>% knitr::kable(caption = "RMSE")
```

RMSE

method	RMSE
Mean of rating model	1.060006

2.3 The movie effect model

We are considering the *movie effect* (b_i) as predictors. Therefore, we are generating the next model to predict `rating` ($\hat{y}_i$):

$$\hat{y}_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

```
movie_mean<- train %>%
  group_by(movieId) %>%
  summarize(b_i= mean(rating-mu))

#Check the test result
movie_prediction <- test %>%
  right_join(movie_mean, by = "movieId")%>%
  mutate(prediction = mu + b_i)

movie_effect_rmse <- RMSE(test$rating, movie_prediction$prediction)
movie_effect_rmse

## [1] 0.9429084

#Save results in table
rmse_results <- rmse_results %>%
  add_row(method="Movie effect model", RMSE = movie_effect_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE

Mean of rating model	1.0600060
Movie effect model	0.9429084

2.4 The User and Movie effect model

We are considering the *user effect* (b_u) and the *movie effect* (b_i) as predictors. Therefore, we are generating the next model to predict `rating` ($\hat{y}_i$):

$$\hat{y}_{u,i} = \mu + b_u + b_i + \varepsilon_{u,i}$$

```
user_mean <- train %>%
  left_join(movie_mean, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-mu-b_i))

#Check the test result
user_prediction <- test %>%
  left_join(movie_mean, by = "movieId") %>%
  left_join(user_mean, by = "userId") %>%
  mutate(prediction = mu+b_i+b_u)

user_effect_rmse <- RMSE(test$rating, user_prediction$prediction)
user_effect_rmse

## [1] 0.8651144

#Save results in table
rmse_results <- rmse_results %>%
  add_row(method = "User & Movie effect model", RMSE =
user_effect_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Mean of rating model	1.0600060
Movie effect model	0.9429084
User & Movie effect model	0.8651144

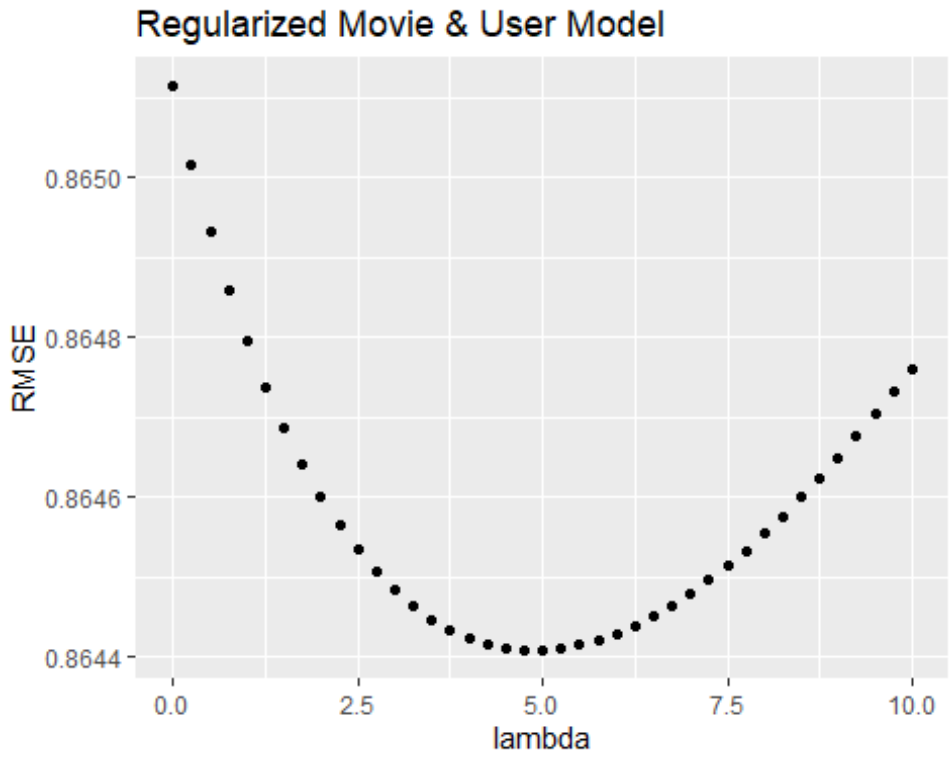
2.5 Regularized Movie and User effect model

The regularisation process will evaluate different values for λ, delivering to us the corresponding RMSE.

```
#We use cross-validation to pick the parameter lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(lambda){
  mu <- mean(train$rating)
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating-mu)/(n()+lambda))
  b_u <- train %>%
```

```
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating-b_i-mu)/(n()+lambda))
  predictions <- test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu+b_i+b_u) %>%
    .$pred
  RMSE(test$rating, predictions)
})

#Plot the lambdas vs RMSEs
qplot(lambdas, rmses,
      main = "Regularized Movie & User Model",
      xlab = "lambda", ylab = "RMSE")
```




```
"According to the plot, the best lambda is:"
```

```
## [1] "According to the plot, the best lambda is:"
```

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

```
## [1] 5
```

```
"And the min RMSE is:"
```

```
## [1] "And the min RMSE is:"
```

```
min(rmses)

## [1] 0.8644076

#Save results in table
rmse_results <- rmse_results %>%
  add_row(method = "Regularized Movie & User effect model", RMSE =
min(rmses))
rmse_results %>% knitr::kable()
```

method	RMSE
Mean of rating model	1.0600060
Movie effect model	0.9429084
User & Movie effect model	0.8651144
Regularized Movie & User effect model	0.8644076

3. RESULTS

As a result, the smallest RMES value was achieved by Regularized Movie and User effect model with RMSE of 0.8644076 and it is lower than 0.86490, so we achieved the target.

method	RMSE
Mean of rating model	1.0600060
Movie effect model	0.9429084
User & Movie effect model	0.8651144
Regularized Movie & User effect model	0.8644076

4. CONCLUSION

At the end, a movie recommendation system had been built using Movielens dataset. We trained a machine learning algorithms by following these steps, first: we start with a model that assumes the same rating for all movies and all users. Second: we improved it by adding a term (b_i) that represents the average rating for movie (i). Third: a further improvement for the model was made by adding (b_u), the user-specific effect. Finally, we used *Regularization* on *Movie* and *User* effect model which was the best model with the least RMSE of 0.8644076.