Final MovieLens Report

Ing. José Ramón Riesgo Escovar

May 2020

Introduction to the Report

This report is one of the final projects for the course **HarvardX: PH125.9x Data Science: Capstone** the goal is to demonstrate the acquire skills within R in the field of Data Science, in order to solve real world problems.

The goal is to generate a recommendation model that is able to predict the "rating" and the "potential" evaluation a user will give to a specific movie.

Classification Systems are based on analyzing the actual ratings previous users made to the movies, allowing them to predict their future rating of others.

In October 2006, Netflix offered to the data science community a challenge where anyone that can improve the recommendation algorithm by 10% will win a million dollars; on September 2009, the winner was announced. Taking into consideration the data analysis strategies used by the winners, this paper will propose a model based on a similar algorithm which generates movies´ratings.

Due that the actual information of Netflix is not available, we will use the information from MovieLens. This company was created by GroupLens Research, responsable of building data sets from the movies that were collected over various periods of time; for this specific study, the sample will be 10M data set.

We are going to load all the information, first the 90% of it will be used to generate a training set called **edx**. The remaining 10% will be defined as the final testing set, known as **validation**; which later on will be execute against the training group to define the "best" model.

As said, the first is to upload the **edx** information, generating the following summary:

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

As seen in the table, the average rating "stars" is **3.512**; showing a minimum of .5 and maximum of 5; being this range the one consider in the propose model.

The objective of the model is that the ratings generated by the "best" produce model against the **validation** data, represents a root-mean-square-error (RMSE) smaller than $<$ **.86490**.

For the calculation of the RMSE, the formula will be:

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

In R language, the code for doing so is:

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

The next step is the division of the data set **edx** in a training and test groups with the following characteristics: - Setting the seed to 1, so always you have the same result - Dividing the **edx** data, where 80% is considered training and 20% test set. - Defining the training set as **train_set** and test set as **test_set** with the following code:

```
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Before running the calculation, it is necessary to revise there are no users, movies, and genres in the test group that are not present in the training set. For doing so, the following code should be use:

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "genres")
```

Methods and Analysis:

In this section, there will be shown several models; starting with the most general one and concluding with a more precise, meaning this last one is the "best" possible model to achieve the goal mention above.

FIRST MODEL

For the first and more simple model, there will be use the average values of the training set with the following formula having μ as the average of the ratings in the set and a standard error $\epsilon_{u,i}$:

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

Assuming same ratings for all movies and all users.

Calculating it with the following code:

```
mu_adv <- mean(train_set$rating)
```

Within the variable **mu_adv** there will be kept the mean of the ratings or μ.

The value of **mu_adv** is **3.5124**

Executing the RMSE function and comparing the generated **mu_adv** with the actual ratings of the test set:

```
mu_model <- RMSE( mu_adv,test_set$rating)
```

The actual result of the RMSE of the first Algorithm is: **1.0599**

The results of each model are captured in the table of "results of the RMSE" of each model, named the table **rmse_results**

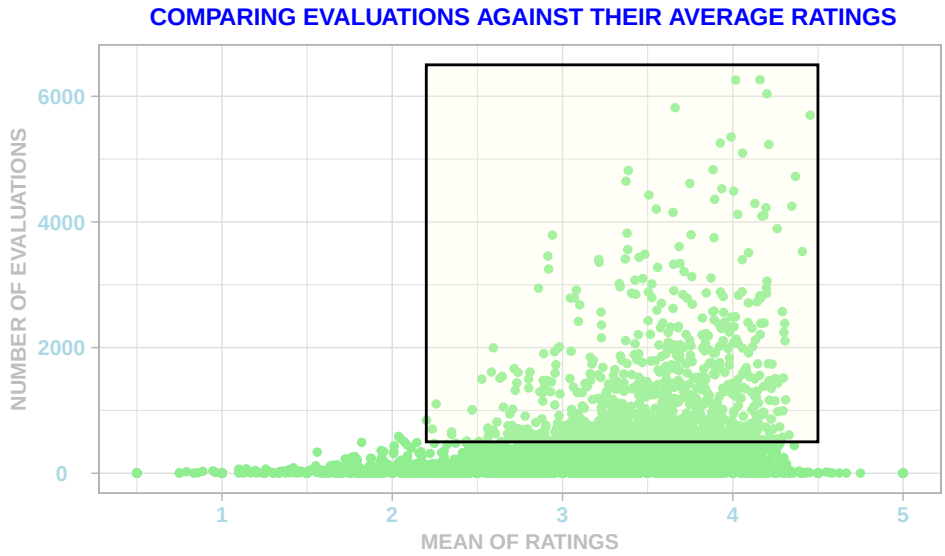
```
rmse_results <- data.frame(METHOD = "Simple Average Model", RMSE = mu_model )
```

The result of the first algorithm is:

METHOD	RMSE
Simple Average Model	1.059912

SECOND MODEL

The first model was very basic, now exploring more the data with a chart, it is shown that some movies get higher ratings than others:



There is a clear tendency highlighted in a rectangle, meaning that a higher number of evaluations tend to get a higher average in the ratings

Based on this fact, the previous model can be augment by adding an average ranting per movie; transforming the formula to this one:

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

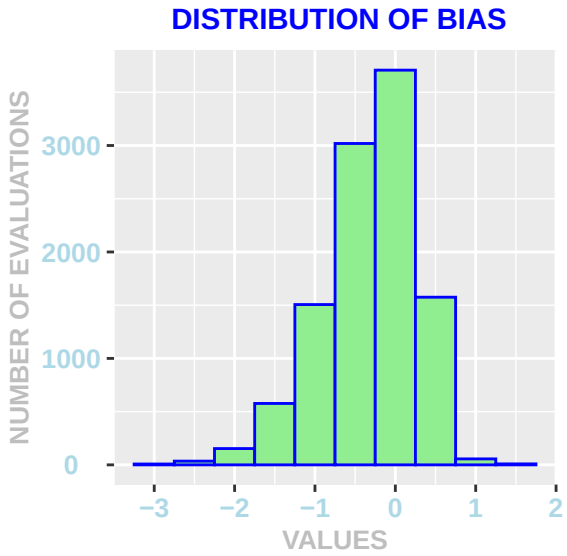
Using the least square,to calculate the b_i that will be the average of $Y_{u,i}$ minus the overall mean (**mu_adv**) for each movie i.

For this estimation, it is use the following code:

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_adv))
```

These b_i are also refered as the "BIAS".

In the following table is shown the distribution of the BIAS



These varies substantially because some movies are good, other average and some bad.

Using these bias there will be calculated the ratings using the following code:

```
movie_model <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(prediction = mu_adv + b_i) %>%
  pull(prediction)
```

With these new ratings generated b_i the model, will be calculate the RMSE against the test set

```
movie_result <- RMSE( movie_model,test_set$rating)
```

These are the results of the enhance model with the "Movies Effect":

```
movie_result
```

```
## [1] 0.9438759
```

Including the model results in our table of **rmse_results**

These is the updated table with the two results:

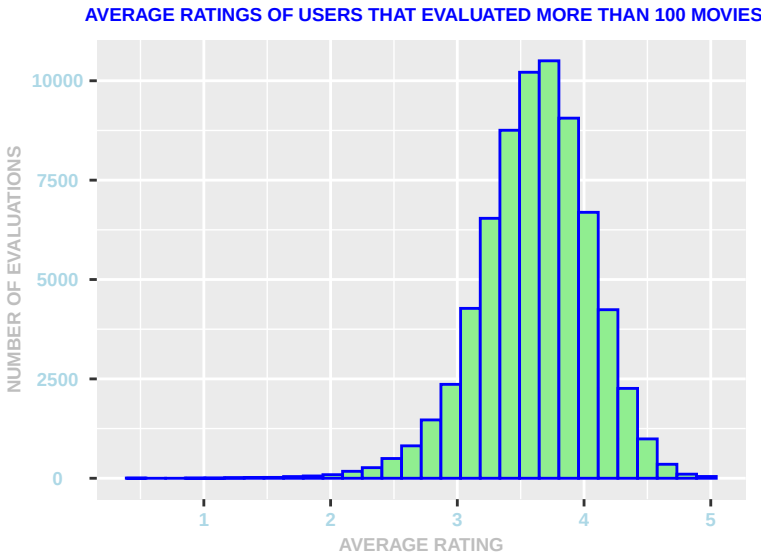
METHOD	RMSE
Simple Average Model	1.0599118
Movie-Based Model	0.9438759

THIRD MODEL

Another optimization to the model is to consider if the amount of evaluations the users made have any influence in the overall accuracy of the ratings.

Let's explore these effect for users that have rated more than 100 movies in a chart with their average evaluations.

The following table shows this relationship:



Now there will be calculate this effect adding the user evaluation in the model.

The following formula will be used adding the effect of the users

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

Generating the effect of the users b_u

The following code calculates the effect of the users:

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_adv - b_i))
```

With this user Adjustment, there will be generate the new adjusted model using the following code:

```
movie_user_model <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_adv + b_i + b_u) %>%
  pull(pred)
```

With the new estimation it is calculate the RMSE for the model with this code:

```
movie_user_result <- RMSE(movie_user_model, test_set$rating)
```

The following is the result of this model that takes now into account the movies as well as the user rating adjustment:

```
movie_user_result
```

```
## [1] 0.8662815
```

Include the new model results in our table of **rmse_results**

This is the updated **rmse_results** table:

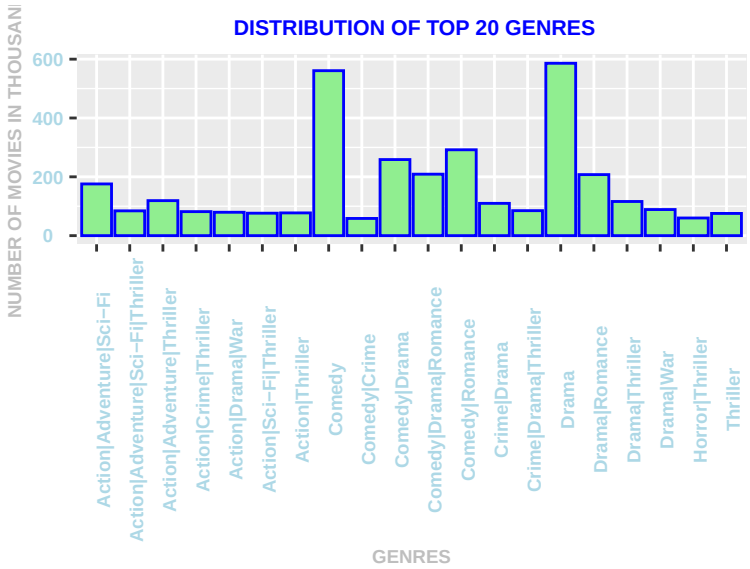
METHOD	RMSE
Simple Average Model	1.0599118
Movie-Based Model	0.9438759
Movie+User Model	0.8662815

FOURTH MODEL

Another optimization to the model is to consider if the genres have any influence in the overall accuracy of the ratings.

We show a table of the top 20 genres and the amount of movies within that genre:

```
## Selecting by num_movies
```



It looks like there is enough potential influence to enhance our model

Let's explore these effect for genres have by adding a b_g to the model

Now the effect of adding the genres evaluation will be calculate.

The following formula will be used adding the effect of the genres

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

Generating the effect of the genres b_g

The following code calculates the effect of the Genres:

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by= 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_adv - b_i - b_u))
```

With this user Adjustment there will be generate the new adjusted model using the following code:

```
movie_user_genres_model <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu_adv + b_i + b_u + b_g) %>%
  pull(pred)
```

With the new estimation it will calculate the RMSE for the model with this code:

```
movie_user_genres_result <- RMSE(movie_user_genres_model, test_set$rating)
```

The following is the result of this model that takes now into account the movies as well as the user and the genres in the rating adjustment:

```
movie_user_genres_result
```

```
## [1] 0.8659344
```

Including the new model results in our table of **rmse_results**

This is the updated **rmse_results** table:

METHOD	RMSE
Simple Average Model	1.0599118
Movie-Based Model	0.9438759
Movie+User Model	0.8662815
Movie+User+Genres Model	0.8659344

FIFTH MODEL

Now applying the concept of Regularization to the 3 elements of our previous model.

Let's explore these effect Regularization in b_i , b_u, b_g to the previous model

The following formula will be used adding the effect of Regularization

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

The following code calculates the effect Regularization in b_i, b_u and b_g, it will be calculated with Lambdas from 0 to 10

```
lambdas <- seq(0, 10, 0.25)
# Compute the predicted ratings on test_set dataset using different values of lambda
rmses <- sapply(lambdas, function(lambda) {
  # Calculate the average by movie
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_adv) / (n() + lambda))
  # Calculate the average by user
  b_u <- train_set %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_adv) / (n() + lambda))
  # Calculate the average by genre
  b_g <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - mu_adv - b_u) / (n() + lambda))

  # Compute the predicted ratings on testing data dataset
  predicted_ratings <- test_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    mutate(pred = mu_adv + b_i + b_u + b_g) %>%
    pull(pred)

  # Predict the RMSE on the testing set
  return(RMSE(predicted_ratings,test_set$rating))
})

# Get the lambda value that minimize the RMSE
min_lambda <- lambdas[which.min(rmses)]

# Get the "best" minimize RMSE function
reg_movie_user_genres_result <- min(rmses)
reg_movie_user_genres_result
```

```
## [1] 0.8652471
```

Including the new model results in our table of **rmse_results**

This is the updated **rmse_results** table:

METHOD	RMSE
Simple Average Model	1.0599118
Movie-Based Model	0.9438759
Movie+User Model	0.8662815
Movie+User+Genres Model	0.8659344
Reg Movie+User+Genres Model	0.8652471

Final Validation of the "Best Model:

Uploading the **validation** to calculate the actual precision with "best" model to validate the objective to have < than **.86490** in the RSME function.

```
load(file = "rda/validation.rda")
```

FINAL BEST MODEL EVALUATION

This is the actual code for the edx set with the validation set as the test set

```
lambdas <- seq(0, 10, 0.25)
# Compute the final ratings on validation dataset using different values of lambda
# and the whole edx data set

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

  # Calculate the average by movie
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_adv) / (n() + lambda))

  # Calculate the average by user
  b_u <- edx %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_adv) / (n() + lambda))

  # Calculate the average by genre
  b_g <- edx %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - mu_adv - b_u) / (n() + lambda))

  # Compute the predicted ratings on validation dataset
  predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    mutate(pred = mu_adv + b_i + b_u + b_g) %>%
    pull(pred)

  # Predict the RMSE on the validation set
  return(RMSE(predicted_ratings,validation$rating))
})

# Get the lambda value that minimize the RMSE
min_lambda <- lambdas[which.min(rmses)]

# Predict the RMSE on the validation set
min_lambda
```

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

```
final_result <- min(rmses)
```

```r
# Display the result of the RMSE function
final_result
```

```
## [1] 0.8644501
```

Conclusion:

This report show very clearly how we can add more relevant predictors and combined them with a final regularization effect and achieve a very reasonable result obtaining **.8644501** tha is below the target objective of **.86490** in the RSME function.

Possible future enhancements will be to also wotk with the timestamp information and add another predictor for year month so continue looking for enhancements in the model and gain more precision.