Movielens

Jordy Veuger

6/09/2019

0.1 Movielens Capstone

1 1. Introduction

The purpose of this R project is to try to create a model that predicts what rating a user will give to a movie. Or in other words, it predicts the user preference for a certain movie. As validation of the model we will use RMSE (Root Mean Square Error).

The most famoust recommender event was originally a Netflix competetion with the following goal: Whoever improves our recommendation engine with 10% wins a million dollars.

2 1.1 The approach

The goal of the project is to establish an algorithm with a RMSE of 0.87750 or lower.

In order to reach this goal, the following steps were taken: Explored the data set Split the data set into a training set (9,000,055 observations) and a test set (999,999 observations). Apply different methods to improve the algorithm until it produced an RMSE of 0.87750 or lower. Test the algorithm on the Test (validation) set. Compare RMSEs between training and test set to verify the model.

3 2. The Dataset.

In the provided data set we find 10,000,054 observations of 6 variables. List of variables. userId: Identification number for each user. movieId: Identification number for each movie. timestamp: Date and time at which each rating was recorded. title: Title of each movie. genres: Movie genre which each movie is classified as. rating: Rating between 0 and 5 with intervals of 0.5 for each movie.

This is the code provided by edx.

```
## Loading required package: tidyverse
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

```
## -- Attaching packages ---------------------------------------------------------------------------------
## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

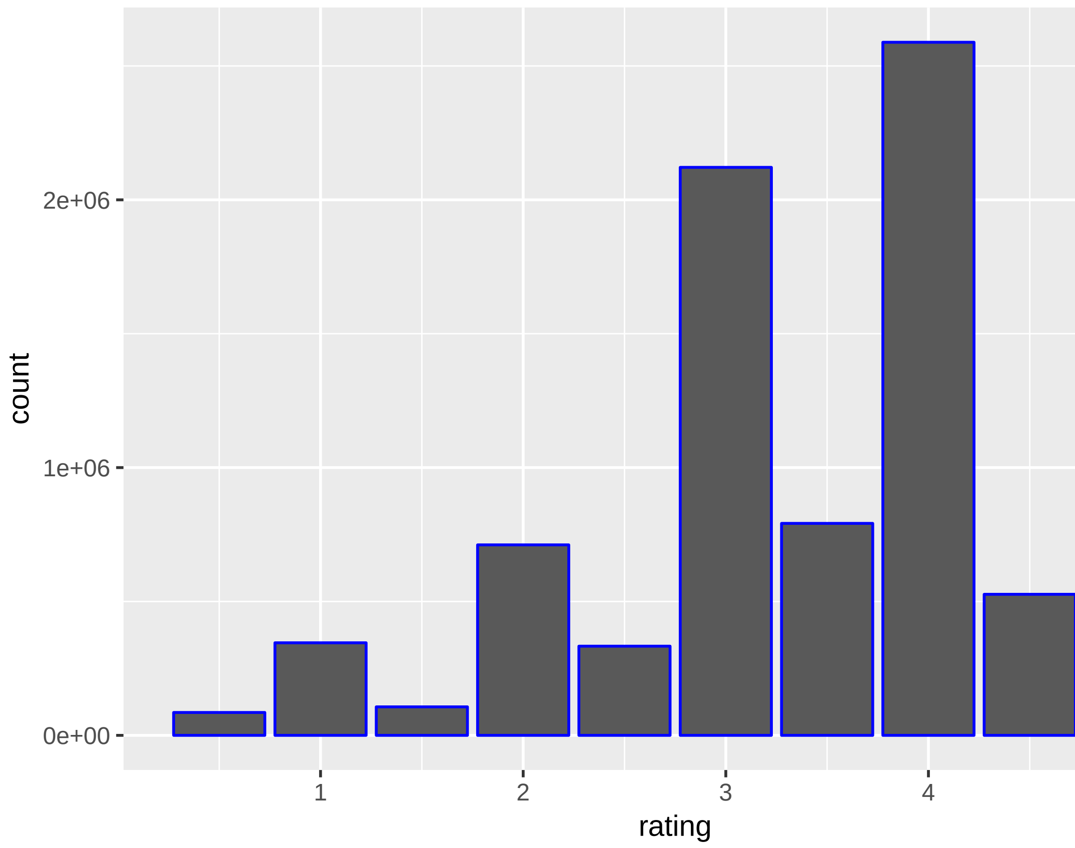
4 3. Exploratory Analysis.

Alot of the exploratory analysis has already been done in order to answer the questions on the Quiz in the course material. However I will try to visualize some of the data from the original data set to paint a better picture of what the set looks like and how it's distributed.

Printing the head of the data set to see what it looks like.

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```



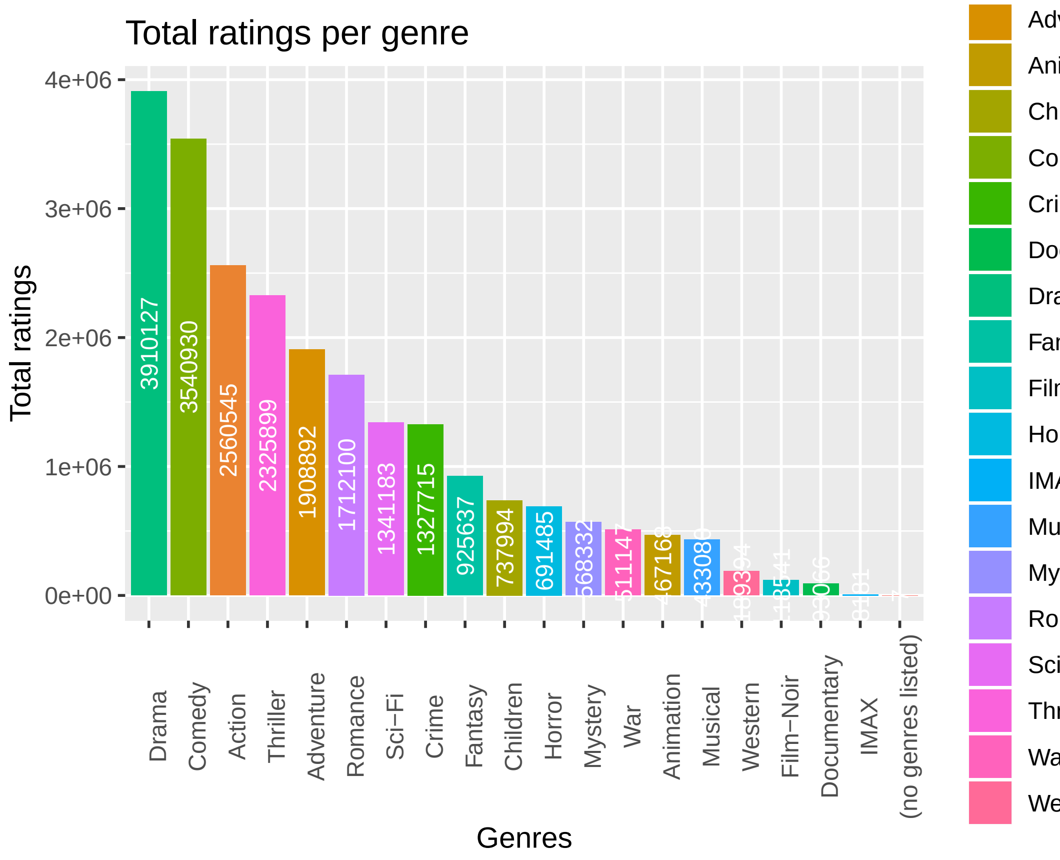
Plot the Distribution of ratings

```
## $x
## [1] "Rating"
##
## $y
## [1] "Frequency"
##
## $title
## [1] "Distribution of Ratings"
##
## attr(,"class")
## [1] "labels"
```

An overview of the metrics per genre shows us how the data set is distributed by genre.

```
edx_metrics_perGenre <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(Total_ratings_perGenre = n(), Avarage_ratings_perGenre = mean(rating),
            Total_movies_perGenre = n_distinct(movieId),Total_users_perGenre = n_distinct(userId))
edx_metrics_perGenre
```

```
## # A tibble: 20 x 5
##    genres Total_ratings_p~ Avarage_ratings~ Total_movies_pe~
##    <chr>             <int>            <dbl>            <int>
##  1 (no g~                7             3.64                1
##   2 Action         2560545              3.42              1473
##   3 Adven~         1908892              3.49              1025
##   4 Anima~          467168              3.60               286
##   5 Child~          737994              3.42               528
##   6 Comedy         3540930              3.44              3703
##   7 Crime          1327715              3.67              1117
##   8 Docum~           93066              3.78               481
##   9 Drama          3910127              3.67              5336
##  10 Fanta~          925637              3.50               543
##  11 Film-~          118541              4.01               148
##  12 Horror          691485              3.27              1013
##  13 IMAX              8181              3.77                29
##  14 Music~          433080              3.56               436
##  15 Myste~          568332              3.68               509
##  16 Roman~         1712100              3.55              1685
##  17 Sci-Fi         1341183              3.40               754
##  18 Thril~         2325899              3.51              1705
##  19 War             511147              3.78               510
##  20 Weste~          189394              3.56               275
## # ... with 1 more variable: Total_users_perGenre <int>
```



Plot the Total ratings per genre.

5 4. The model

Before we start we establish the baseline. By calculating the mean and RMSE of the training set we have a clear comparison to see how the model is performing.

```
# Baseline avarage rating of all movies
mu_base <- mean(edx$rating)
mu_base
```

```
## [1] 3.512465
```

```
# Baseline RMSE
RMSE_base <- RMSE(edx$rating, mu_base)
RMSE_base
```

```
## [1] 1.060331
```

```
# Creating tibble
RMSE_table <- tibble(Method = "Baseline", RMSE = RMSE_base)
RMSE_table
```

```
## # A tibble: 1 x 2
##   Method    RMSE
##   <chr>    <dbl>
## 1 Baseline  1.06
```

5.1 4.1 User and movie effect

The first step to improve the RMSE is to take into account the user (u_i) and movie (m_i) bias.

```
# Modelling user and movie effect
mu <- mean(edx$rating)
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarize(m_i = mean(rating - mu))
user_avg <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(u_i = mean(rating - mu - m_i))
predicted_ratings <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  mutate(pred = mu + m_i + u_i) %>% .$pred
model_RMSE <- RMSE(predicted_ratings, edx$rating)
model_RMSE
```

```
## [1] 0.8567039
```

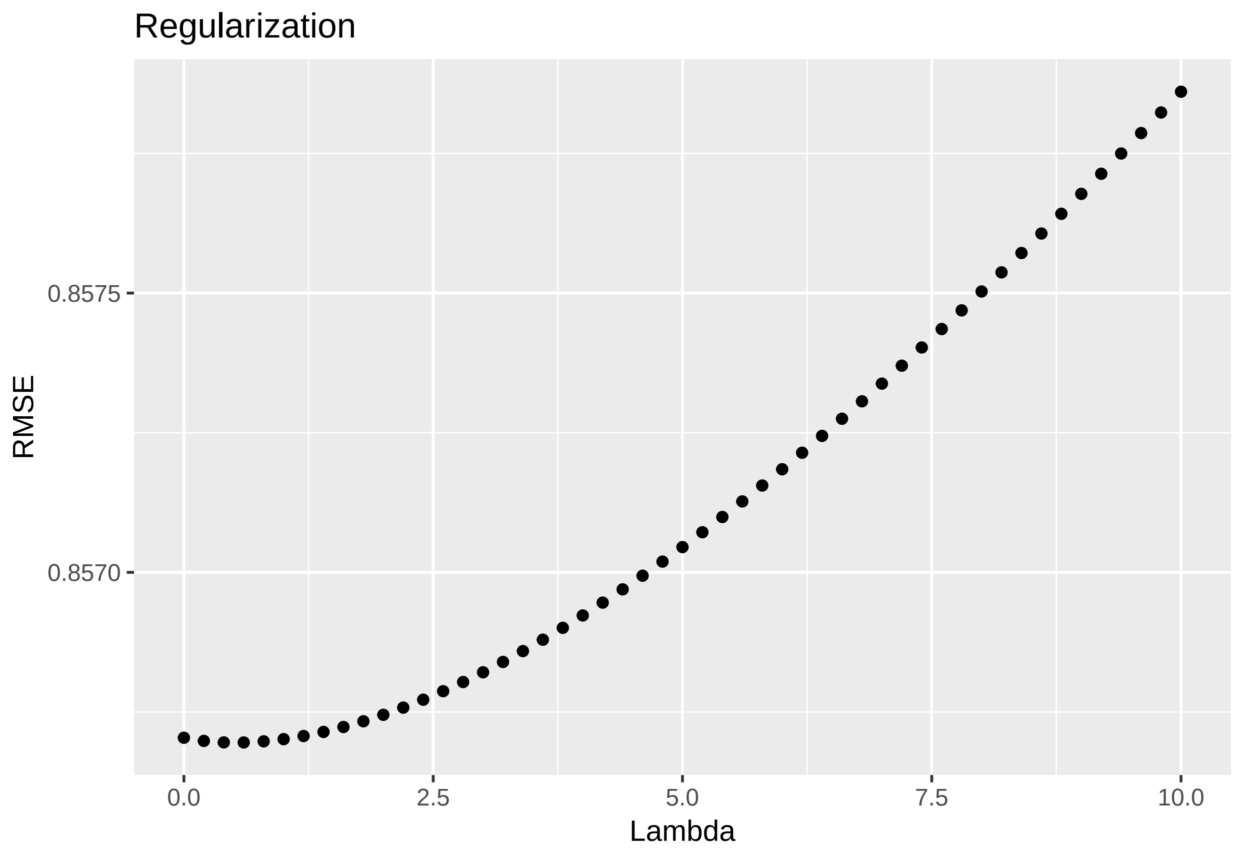
We already see a significant improvement over the baseline model. We´ll explore if we can further improve the model through regularization.

```
## # A tibble: 2 x 2
##   Method               RMSE
##   <chr>               <dbl>
## 1 Baseline            1.06
## 2 User & Movie Effect 0.857
```

5.2 4.2 Regularization

Regularization allows us to penalize large estimates formed by using small sample sizes. Therefore the model will be less likely to fit the noise of the training data.

```
# Select lambda
lambdas <- seq(0, 10, .2)
# Create function
RMSE_reg_fun <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

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

  u_i <- edx %>%
    left_join(m_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(u_i = sum(rating - m_i - mu)/(n()+l))

  predicted_ratings <- edx %>%
    left_join(m_i, by = "movieId") %>%
    left_join(u_i, by = "userId") %>%
    mutate(pred = mu + m_i + u_i) %>% .$pred

  return(RMSE(predicted_ratings, edx$rating))
})
```



```
## [1] 0.6
```

```
## [1] 0.8566954
```

Regularizing the model slightly improved the RMSE.

```
## # A tibble: 3 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Baseline                             1.06
## 2 User & Movie Effect                  0.857
## 3 User & Movie Effect + Regularization 0.857
```

5.3 4.3 Baseline validation

Before testing how our model performs on the validation set we have to establish the baseline.

```
# Baseline avarage rating of all movies
mu_base <- mean(validation$rating)
mu_base
```

```
## [1] 3.512033
```

```
# Baseline RMSE
RMSE_base <- RMSE(validation$rating, mu_base)
RMSE_base
```

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

```
## # A tibble: 4 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Baseline                             1.06
## 2 User & Movie Effect                  0.857
## 3 User & Movie Effect + Regularization 0.857
## 4 Baseline(validation)                 1.06
```

5.4 4.4 User and movie effect (validation)

Let´s test the model on the validation set.

```
# Modelling user and movie effect
mu <- mean(validation$rating)
movie_avg <- validation %>%
  group_by(movieId) %>%
  summarize(m_i = mean(rating - mu))
user_avg <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(u_i = mean(rating - mu - m_i))
predicted_ratings <- validation %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by = "userId") %>%
  mutate(pred = mu + m_i + u_i) %>% .$pred
model_RMSE <- RMSE(predicted_ratings, validation$rating)
model_RMSE
```

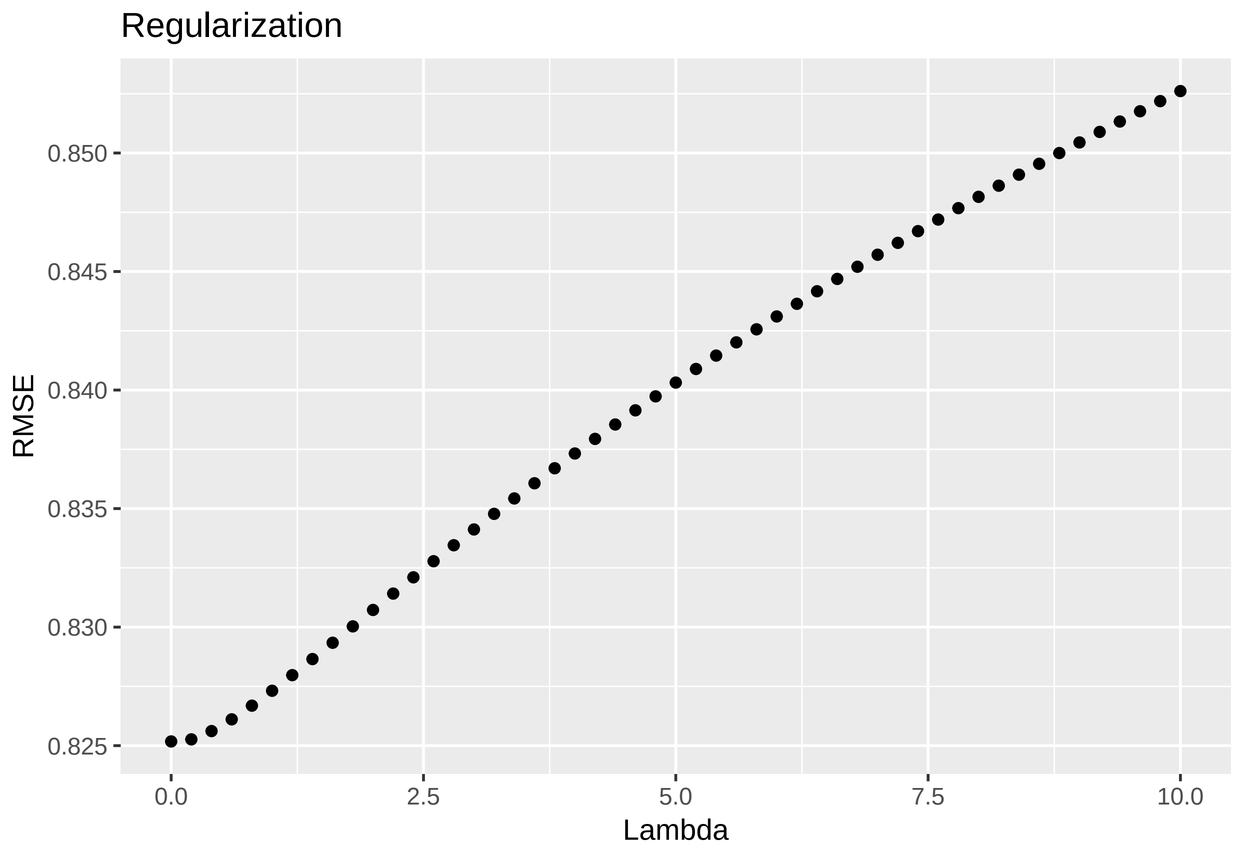
```
## [1] 0.825177
```

```
## # A tibble: 5 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Baseline                             1.06
## 2 User & Movie Effect                  0.857
## 3 User & Movie Effect + Regularization 0.857
## 4 Baseline(validation)                 1.06
## 5 User & Movie Effect (validation)     0.825
```

5.5 4.5 Regularization (validation)

Let’s see how the regularized model performs on the validation set.

```
# Select lambda
lambdas <- seq(0, 10, .2)
# Create function
RMSE_reg_fun <- sapply(lambdas, function(l){

  mu <- mean(validation$rating)

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

  u_i <- validation %>%
    left_join(m_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(u_i = sum(rating - m_i - mu)/(n()+l))

  predicted_ratings <- validation %>%
    left_join(m_i, by = "movieId") %>%
    left_join(u_i, by = "userId") %>%
    mutate(pred = mu + m_i + u_i) %>% .$pred

  return(RMSE(predicted_ratings, validation$rating))
})
```



```
## [1] 0
## [1] 0.825177
```

6 5. Results

```
## # A tibble: 6 x 2
##   Method                                            RMSE
##   <chr>                                            <dbl>
## 1 Baseline                                         1.06
## 2 User & Movie Effect                              0.857
## 3 User & Movie Effect + Regularization             0.857
## 4 Baseline(validation)                             1.06
## 5 User & Movie Effect (validation)                 0.825
## 6 User & Movie Effect + Regularization (validation) 0.825
```

6.1 5.1 Conclusion

From the results in the final table we find that user and movie ID variables had plenty of predicting power to create a model with a more optimal RMSE than our original goal (0.875). However I didn't expect the regularization to have such minimal impact on the model.