MovieLens

Omkar Oka

6/15/2020

Contents

1 Executive summary: **2**
- 1.1 **Evaluation Model:** . . . 2
- 1.2 **Data Preparation:** . . . 2
- 1.3 **DataSet:** . . . 4

2 Analysis Section: **5**
- 2.1 **Data format:** . . . 5
- 2.2 **Exploratory Analysis:** . . . 5

3 The Model: **12**
- 3.1 **Train and Test set:** . . . 12
- 3.2 **Baseline model:** . . . 12
- 3.3 **User and Movie effect Model:** . . . 13
- 3.4 **User and Movie effect Model on *validation* data:** . . . 14
- 3.5 **Regularisation:** . . . 15
- 3.6 **RMSE_Curve_Regularisation_Test:** . . . 16
- 3.7 **Regularisation on *validation* data set:** . . . 17
- 3.8 **RMSE_Curve_Regularisation_Validation:** . . . 18

4 Results: **19**

5 Conclusion: **19**

6 Limitations and Future Work: **19**

1 Executive summary:

Recommendation is the act of generating a suitable suggestion based on a user's shopping pattern and their reviews for certain products , music or movies either on a shopping or a streaming platform. Making the right recommendation for the next product, music or movie increases user retention and satisfaction, leading to sales and profit growth.

The most famous project that tried to achieve that goal was the **The Netflix Prize** (October 2006). This project was an open competition to predict user ratings for films, based on previous ratings without any other information about the users or films. The goal was to make the company's recommendation engine 10% more accurate.

In this project we try to generate a model that can predict the rating that a user will give to a movie based on their preference and previous interactions, in terms of genres and the overall effects of the movie. We will be using the MovieLens data set pre-processed by the GroupLens research lab in the University of Minnesota.

Further we will conduct an exploratory analysis of the data and based on some characteristics of the data we can decide the features that will be used to create a Machine Learning algorithm. This document explores step by step approach, process, techniques and methods that were used to handle the data and to create the predictive model.

The final section shows the results of the previous process and then, the conclusion of the project with a future road map. Our goal is to achieve an RMSE of less than 0.86490.

1.1 Evaluation Model:

The evaluation of machine learning algorithm consists comparing the predicted value with the actual outcome. The loss function measures the difference between both values. We will be using the Root Mean Squared Error method for evaluating the performance of our model.

The Root Mean Squared Error, RMSE, is the square root of the MSE. It is the typical metric to evaluate recommendation systems, and is defined by the formula:

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

where N is the number of ratings, $y_{u,i}$ is the rating of movie i by user u and $\hat{y}_{u,i}$ is the prediction of movie i by user u.

RMSE penalizes large deviations from the mean and is appropriate in cases that small errors are not relevant. Contrary to other methods, the error has the same unit as the measurement.

1.2 Data Preparation:

In this section we download and prepare the dataset to be used in the analysis. We split the data set in two parts, the training set called edx and the evaluation set called validation with 90% and 10% of the original dataset respectively.

Then, we split the edx set in two parts, the train set and test set with 90% and 10% of edx set respectively. The model is created and trained in the train set and tested in the test set until the RMSE target is achieved, then finally we train the model again in the entire edx set and validate in the validation set. The name of this method is cross-validation.

Extracting data publicly available from GroupLens:

```
 #Create test and validation sets
 #Create edx set, validation set, and submission file
options(warn=-1) #Suppress Warnings
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")
if(!require(anytime)) install.packages("anytime", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(corrr)) install.packages("corrr", repos = "http://cran.us.r-project.org")
#Loading Libraries:
library("stringr")
library("tidyverse")
library("caret")
library("anytime")
library("lubridate")
library("corrr")

# 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), g
movielens <- left_join(ratings, movies, by = "movieId")

# 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)
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)
```

1.3 DataSet:

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

The data set contains 9000055 observations of 6 variables.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The Data Description is as follows:
- `userId`: Unique identification number given to each user. `numeric` variable
- `movieId`: Unique identification number given to each movie. `numeric` variable.
- `timestamp`: Code that contains date and time in what the rating was given by the user to the specific movie. `integer` variable.
- `title`: Title of the movie. `character` variable.
- `genres`: Motion-picture category associated to the film. `character` variable.
- `rating`: Rating given by the user to the movie. From 0 to 5 *stars* in steps of 0.5. `numeric` variable.

2 Analysis Section:

We will be performing necessary transformations, data preparation and exploratory data analysis as part of this section.

2.1 Data format:

The `userId` and `movieId` variables are `numeric` columns in the original data set. However it does not make sense. The `userId`=2 is not two times the `userId`=1 and the same effect happens with the `movieId` variable. 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. We could say that these pair of variables have 100% correlation! Only the `movieId` column will be used and will be used as a `factor`. It optimizes the memory (RAM) usage.

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.

```
##    userId movieId rating           timestamp                  title year
## 1:      1     122      5 1996-08-02 16:54:06              Boomerang 1992
## 2:      1     185      5 1996-08-02 16:28:45               Net, The 1995
## 3:      1     292      5 1996-08-02 16:27:01               Outbreak 1995
## 4:      1     316      5 1996-08-02 16:26:32               Stargate 1994
## 5:      1     329      5 1996-08-02 16:26:32 Star Trek: Generations 1994
## 6:      1     355      5 1996-08-02 16:44:34       Flintstones, The 1994
##                          genres year_rate
## 1:                Comedy|Romance      1996
## 2:         Action|Crime|Thriller      1996
## 3:  Action|Drama|Sci-Fi|Thriller      1996
## 4:       Action|Adventure|Sci-Fi      1996
## 5: Action|Adventure|Drama|Sci-Fi      1996
## 6:       Children|Comedy|Fantasy      1996
```

2.2 Exploratory Analysis:

The target is to create a model capable of predicting the variable `rating`.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

As we can see from the distribution of the ratings the minimum rating is 0.5 whereas the first Quantile starts at 3.0 and the mean is 3.5 with the third Quantile at 4.0 which means that most of our ratings are in between 3 and 4 up to 4.5

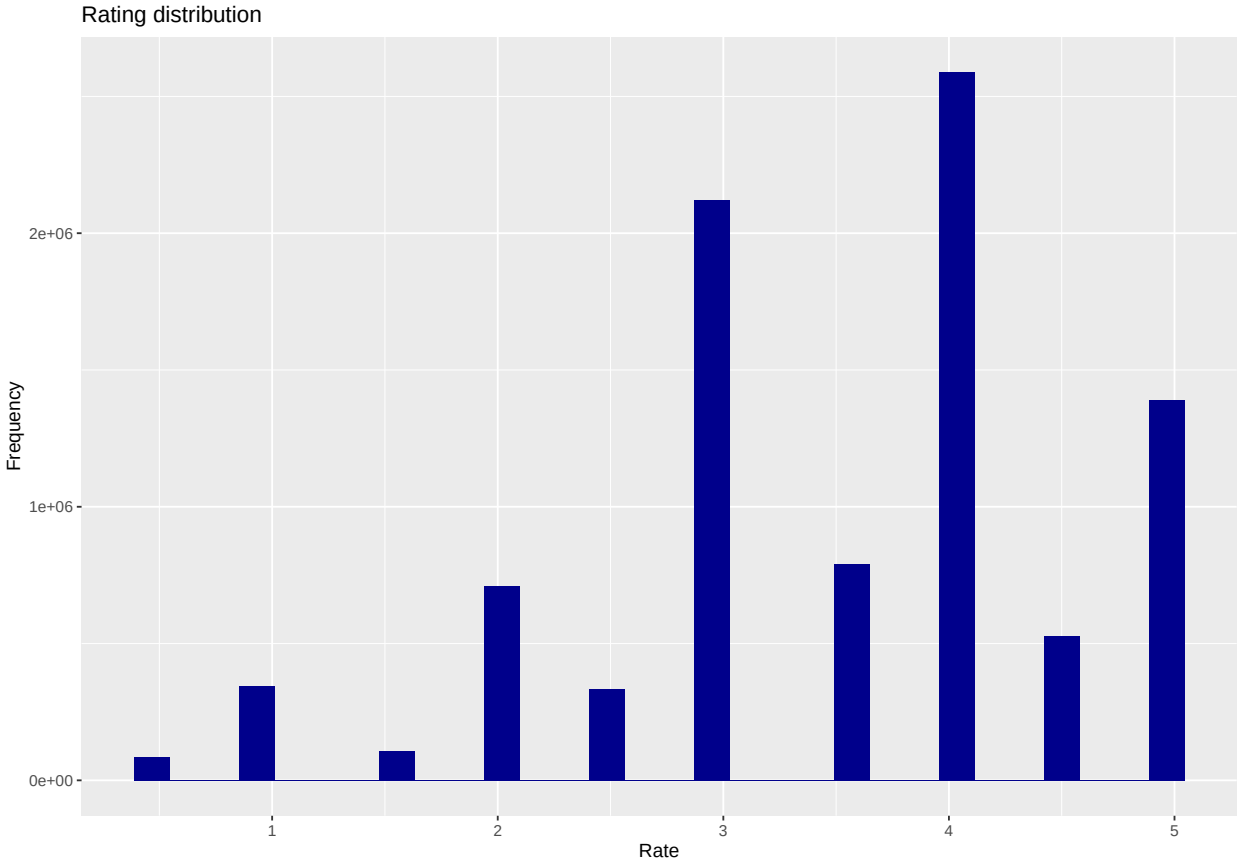
Users have the option to choose a rating value from 0.5 to 5.0, totaling 10 possible values. This is an unusual scale, so all the movies get a rounded value rating, as shown in the chart below.

We can convert these ratings into factors and design the Machine Learning Algorithm as a classification model. But, since we have a target RMSE that we need to beat, its better to create a regression model since RMSE is not usually performed on a classification model.

Overall Distribution of ratings:

Table 1: Rating_Distribution

rating	n	Percentage
4.0	2588430	28.76%
3.0	2121240	23.57%
5.0	1390114	15.45%
3.5	791624	8.8%
2.0	711422	7.9%
4.5	526736	5.85%
1.0	345679	3.84%
2.5	333010	3.7%
1.5	106426	1.18%
0.5	85374	0.95%



We can see that our `rating` variable has a left-skewed distribution. It's interesting that there are more *good* ratings than *bad* ratings. That could be explained by the fact that people want to recommend a film when they like it, but we could just assume that, since there is no data available with this data set to prove this theory.

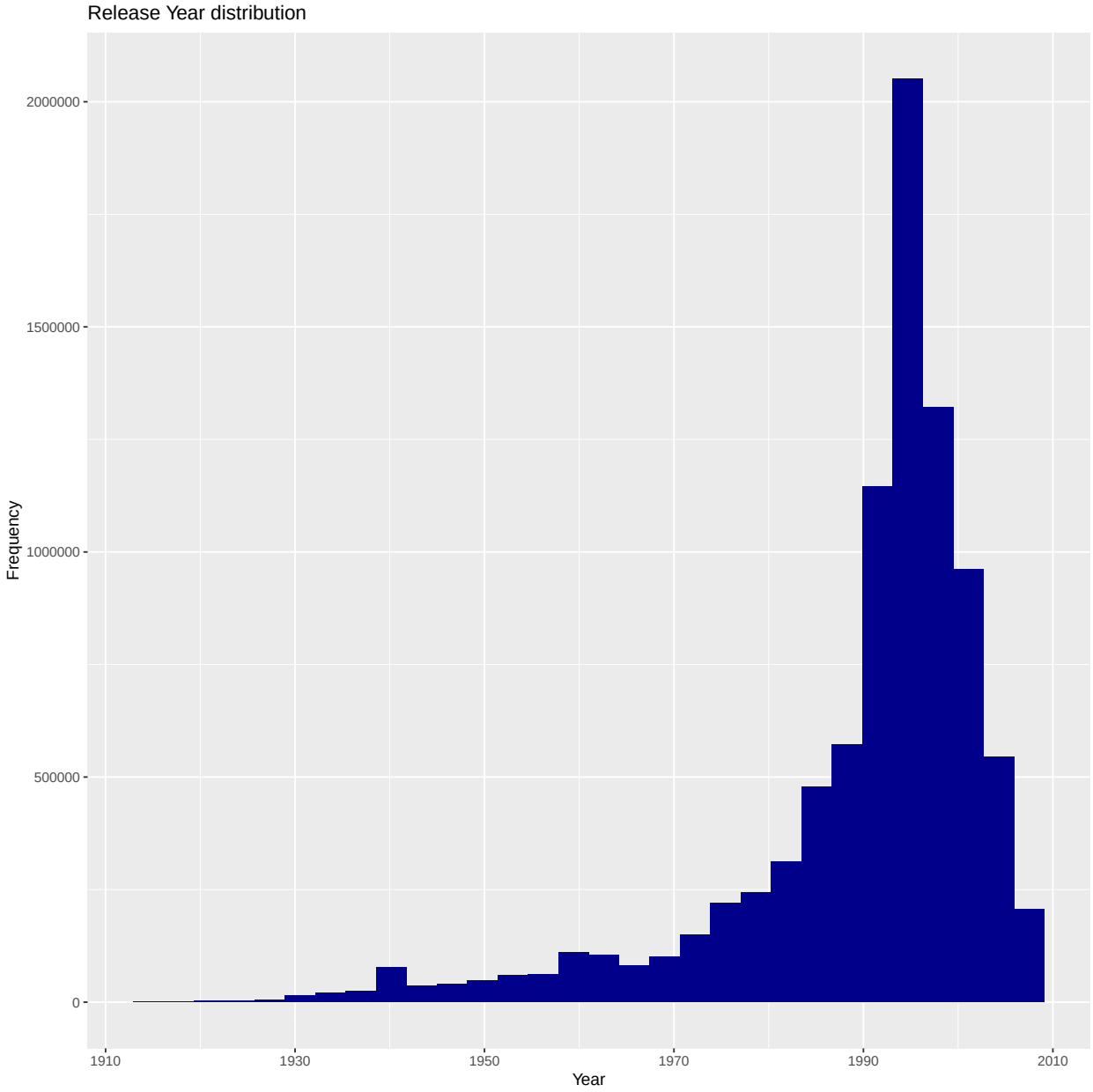
Important Data Players:

Table 2: Unique_Numbers

Unique_Users	Unique_Movies	Unique_Genres
69878	10677	797

We observe that there are 69878 unique users given ratings to 10677 different films. It's good to remember that the *unique genres* were counted as `factor` with no previous separation so, `Drama` and `Comedy|Drama` are counted as 2 different genres.

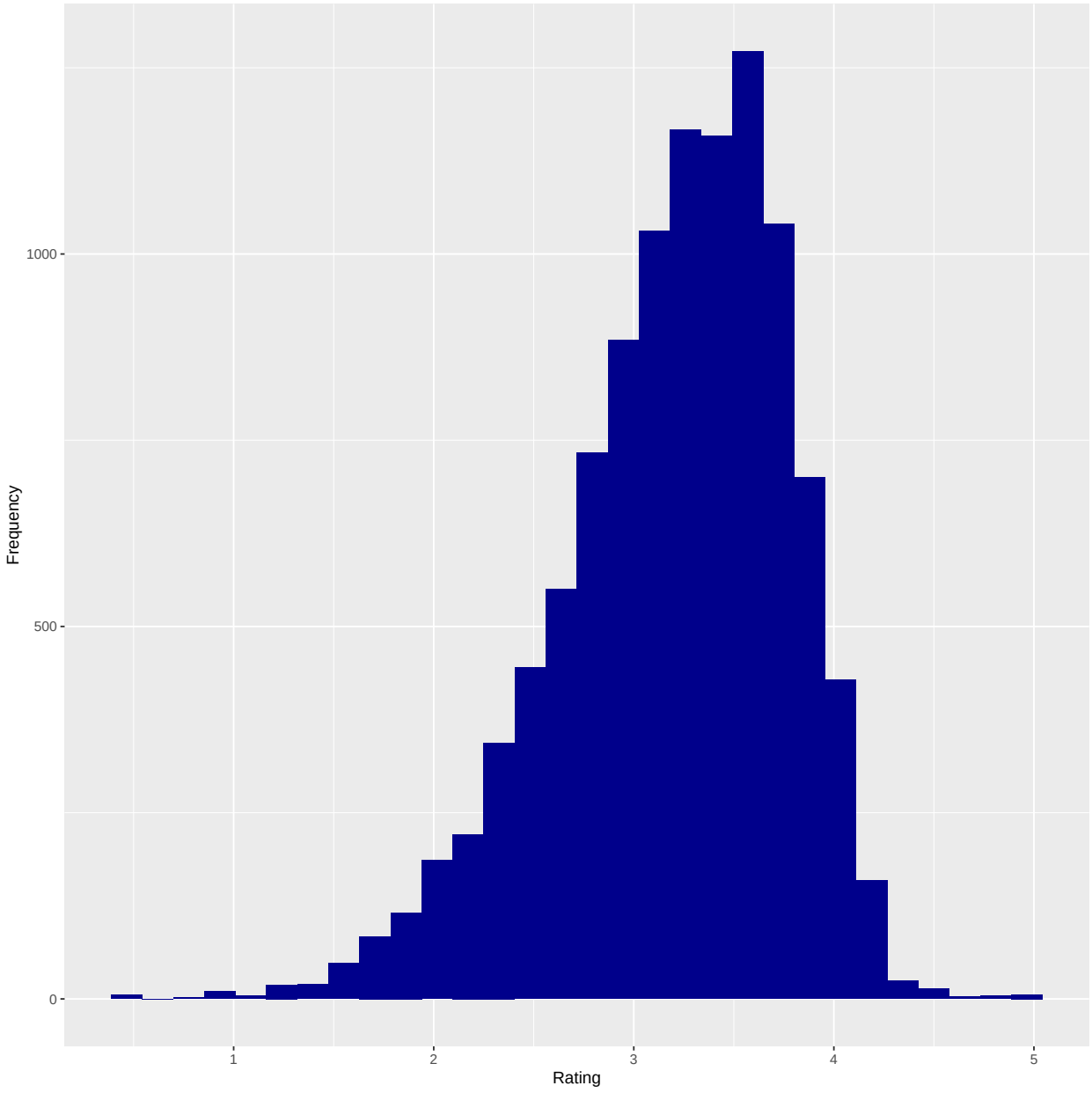
Rates by Release Year:



The frequency of ratings by *release year* of the films has a clear left skewed distribution. The most of those

year are between 1990 and 2009. Primary reason for this could be that since there is a clear age group that actively assign ratings and their preference is usually influenced by the era of the movies.

Average Movie Rating Distribution:



This is also a slightly left skewed distribution, since the average rating for most of the movies ranges from 2.5 to 4.0 stars.

Genres and Ratings:

Table 3: Genres_vs_Ratings

genres	mean	median	n	Percentage
Animation|IMAX|Sci-Fi	4.714286	5.0	7	0%
Drama|Film-Noir|Romance	4.304115	4.5	2989	0.03%
Action|Crime|Drama|IMAX	4.297068	4.5	2353	0.03%
Animation|Children|Comedy|Crime	4.275429	4.5	7167	0.08%
Film-Noir|Mystery	4.239479	4.0	5988	0.07%
Crime|Film-Noir|Mystery	4.216803	4.0	4029	0.04%
Film-Noir|Romance|Thriller	4.216470	4.0	2453	0.03%
Crime|Film-Noir|Thriller	4.210157	4.0	4844	0.05%
Crime|Mystery|Thriller	4.198981	4.0	26892	0.3%
Action|Adventure|Comedy|Fantasy|Romance	4.195557	4.0	14809	0.16%
Crime|Thriller|War	4.171273	4.0	4595	0.05%
Film-Noir|Mystery|Thriller	4.164298	4.0	4011	0.04%
Adventure|Drama|Film-Noir|Sci-Fi|Thriller	4.148384	4.0	13957	0.16%
Adventure|Animation|Children|Comedy|Sci-Fi	4.147917	4.0	3529	0.04%
Adventure|Comedy|Romance|War	4.127992	4.0	5223	0.06%

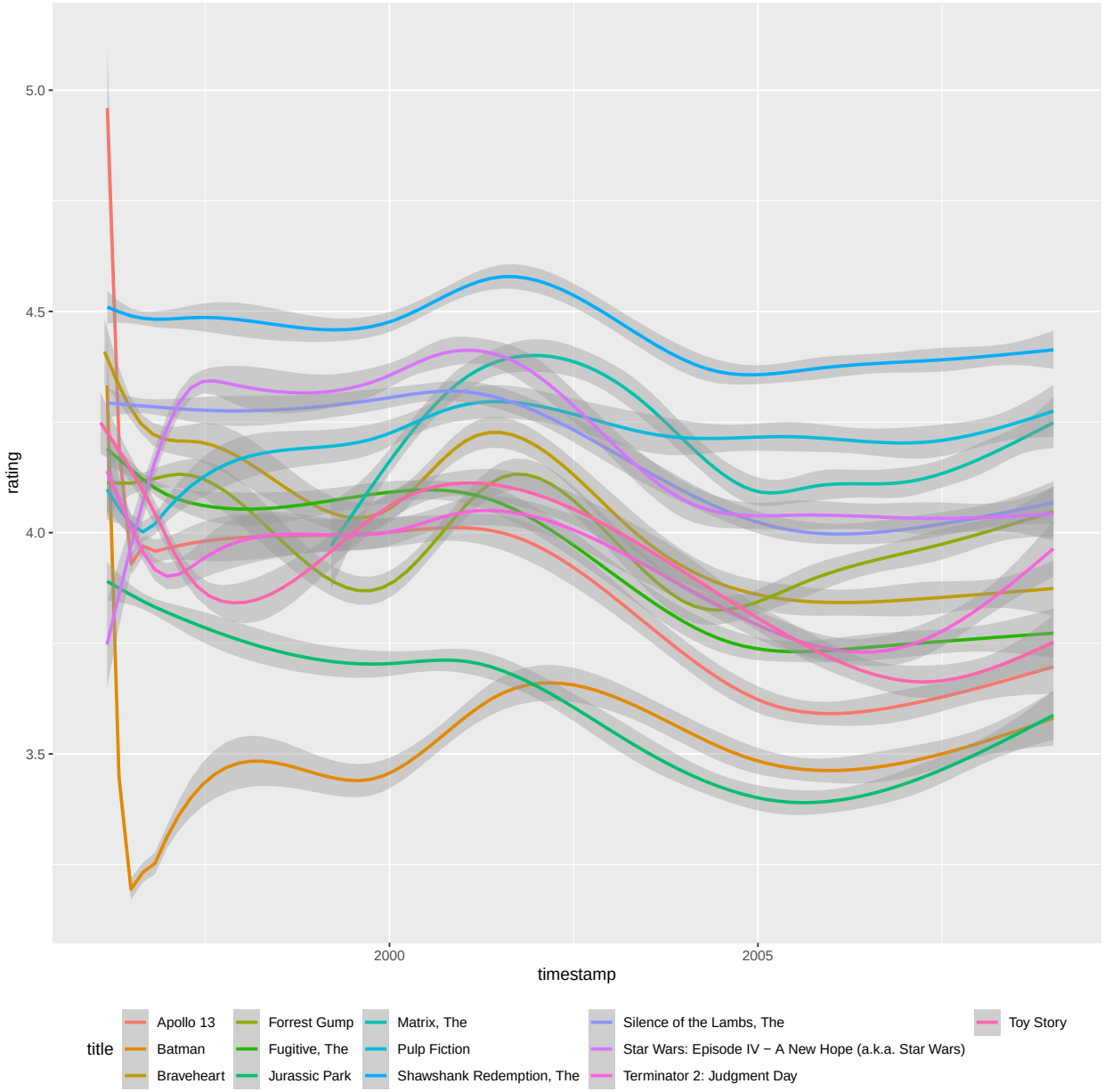
The top 10 **genres** listed above have the highest **mean**. The first place "Animation|IMAX|Sci-Fi" is clearly not significant, because of the only 7 observations. This *genre* will be eliminated below. **Drama** is present in the 2nd and 3rd place.

Table 4: Top_15_Genres

genres	mean	median	n
Drama	4.304115	4.5	2989
Film-Noir	4.304115	4.5	2989
Romance	4.304115	4.5	2989
Action	4.297068	4.5	2353
Crime	4.297068	4.5	2353
Drama	4.297068	4.5	2353
IMAX	4.297068	4.5	2353
Animation	4.275429	4.5	7167
Children	4.275429	4.5	7167
Comedy	4.275429	4.5	7167
Crime	4.275429	4.5	7167
Film-Noir	4.239479	4.0	5988
Mystery	4.239479	4.0	5988
Crime	4.216803	4.0	4029
Film-Noir	4.216803	4.0	4029

The genres associated with the highest mean are "Drama", "Film-Noir" and "Romance". The **difference** with the *second section* entry numbers from 4 till 7 in that ranking is almost zero.

Ratings over the Years:



Here are a few of the movies with good overall average ratings rated by a significant number of users over the years. There seems to be a definite time effect on the ratings which could be related to the Actors or the similarity of the movies with the current ones playing in a theater at the time. The time effect, actor influence or any other effect are out of the scope of this document but it still shows that there is a premise to explore further on these aspects.

Correlation Matrix:

Table 5: Feature_VS_Rating_Correlation

rowname	userId	movieId	rating	genres
userId	NA	0.0060875	0.0023208	-0.0030424
movieId	0.0060875	NA	-0.0282943	-0.0014997
rating	0.0023208	-0.0282943	NA	0.0577063
genres	-0.0030424	-0.0014997	0.0577063	NA

Most of the Machine Learning algorithms prefer a comparable correlation within the features and between the features and the target(rating in this case). Since, as we can see the `UserId`, `MovieId` and `Genres` have a weak correlation with `Rating` any kind of supervised Machine Learning algorithm will not have much effect on the rating predictions and therefore will not be able to provide useful recommendations.

Hence we will be sticking to a simple form of Linear Regression for Predicting ratings, which is covered in the next section.

3 The Model:

Creating a recommendation system involves the identification of the most important features that helps to predict the rating any given user will give to any movie. We start building a very simple model, which is just the mean of the observed values. Then, the user and movie effects are included in the linear model, improving the RMSE. Finally, the user and movie effects receive regularization parameter that penalizes samples with few ratings.

3.1 Train and Test set:

First of all, we create `train` and `test` sets.

```
edx <- edx %>% select(userId, movieId, rating)
set.seed(1, sample.kind="Rounding")
#set.seed(1) #for R version 3.5 and below
test_index <- createDataPartition(edx$rating, times = 1, p = .1, list = FALSE)
# Create the index

train <- edx[-test_index, ] # Create Train set
temp <- edx[test_index, ] # Create Test set

# Make sure userId and movieId in test set are also in train set
test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test)
train <- rbind(train, removed)

rm(test_index, temp, removed)
```

```
dim(train)
```

```
## [1] 8100065       3
```

```
dim(test)
```

```
## [1] 899990      3
```

3.2 Baseline model:

The most basic model is generated when we consider the most common rating from the *train* set to be predicted into the test set. This is the **baseline** model.

```
## [1] 1.060054
```

Now, we have the baseline RMSE to be *beaten* by our model.

Table 6: RMSEs

Method	RMSE
Baseline	1.060054

We can observe that the RMSE of the most basic model is 1.0600537. It's bigger than 1! In this context, this is a very bad model.

3.3 User and Movie effect Model:

The next step is to improve our model and get a better RMSE by introducing *user effect* (u_i) and the *movie effect* (m_i) as predictors. As seen in our exploratory analysis above, there is a definite impact of *user effect* with certain users assigning higher than usual or lower than usual ratings based on the their preference which usually varies from `0.5` to `0.7` in either direction.
Movie effect implies that there are certain movies that generally get rated higher than average based on their popularity. The movie effect can be calculated as the mean of the difference between the observed rating y and the mean μ.

Movie Effect:

$$m_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

User Effect:

$$u_i = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - \hat{b}_i - \hat{\mu})$$

Therefore, we are generating the next model to predict `rating` ($\hat{y}_i$):

$$\hat{y}_i = u_i + m_i + \varepsilon$$

While calculating ratings as continuous values we can assume that there will be some predicted ratings that might be less than zero or greater than 5. We will be applying a balancing condition to counteract these values. We can call this the error(ε) effect and it has been adjusted in the code already.

RMSE obtained by calculating the *user effect* and *movie effect*:

```
## [1] 0.8244714
```

Table 7: RMSE

Method	RMSE
Baseline	1.0600537
User & Movie Effect	0.8244714

We've obtained a better RMSE. Now it is time to test the predictions on Validation data.

3.4 User and Movie effect Model on *validation* data:

First of all, the *validation* data set needs to be handled the same was as the *train* data set was handled.

```
validation <- validation %>% select(userId, movieId, rating)
validation$userId <- as.factor(validation$userId)
validation$movieId <- as.factor(validation$movieId)
validation <- validation[complete.cases(validation), ]
```

Now, we are ready to predict on the validation data set.

```
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(m_i = mean(rating - mu))
user_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(u_i = mean(rating - mu - m_i))
predicted_ratings <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + m_i + u_i) %>% mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5, 5, pred))) %>% .

val_RMSE <- RMSE(predicted_ratings, validation$rating, na.rm = T)
```

Table 8: RMSE

Method	RMSE
Baseline	1.0600537
User & Movie Effect	0.8244714
User & Movie Effect on validation	0.8651772

We can see above that this RMSE is higher than the RMSE on the test set. This is highly probable, given that this was unseen data. The good thing is that the difference is just 0.0407059. Now, let's see if *regularisation* will give us better results.

3.5 Regularisation:

The linear model provides a good estimation for the ratings, but doesn't consider that many movies have very few number of ratings, and some users rate very few movies. This means that the sample size is very small for these movies and these users. Statistically, this leads to large estimated error.

The estimated value can be improved adding a factor that penalizes small sample sizes and have have little or no impact otherwise. Thus, estimated movie and user effects can be calculated with these formulas:

Movie Effect on Regularisation:

$$m_i = \frac{1}{N_i + \lambda} \sum_{i=1}^{N_i} (y_{u,i} - \hat{\mu})$$

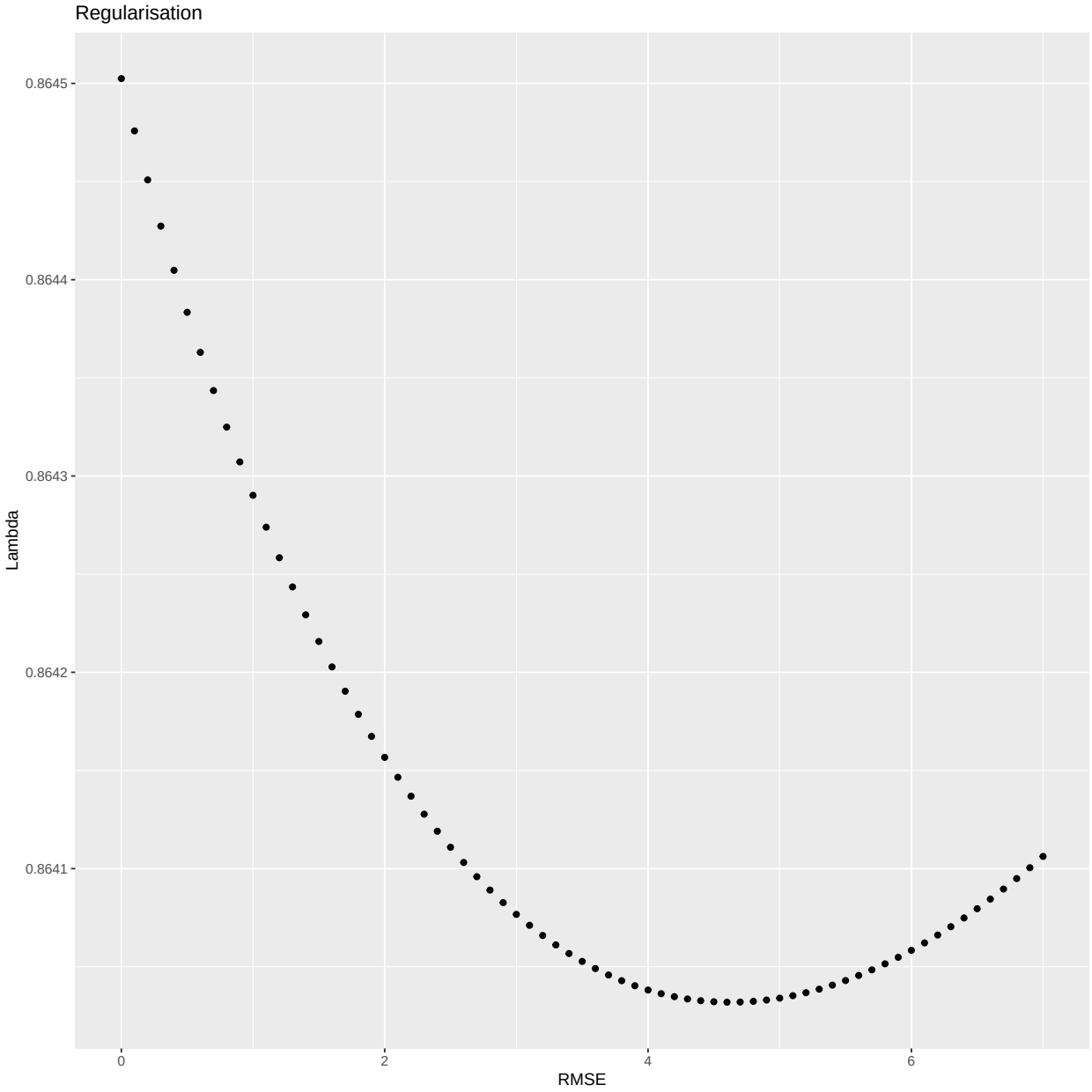
User Effect on Regularisation:

$$u_i = \frac{1}{N_u + \lambda} \sum_{i=1}^{N_i} (y_{u,i} - m_i - \hat{\mu})$$

For values of N smaller than or similar to λ, m_i and u_i is smaller than the original values, whereas for values of N much larger than λ, m_i and u_i change very little.

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

3.6 RMSE_Curve_Regularisation_Test:



Optimum λ for Test: 4.6
RMSE for Test data: 0.864032

Table 9: RMSEs

Method	RMSE
Baseline	1.0600537
User & Movie Effect	0.8244714
User & Movie Effect on validation	0.8651772
User & Movie Effect Regularisation	0.8640320

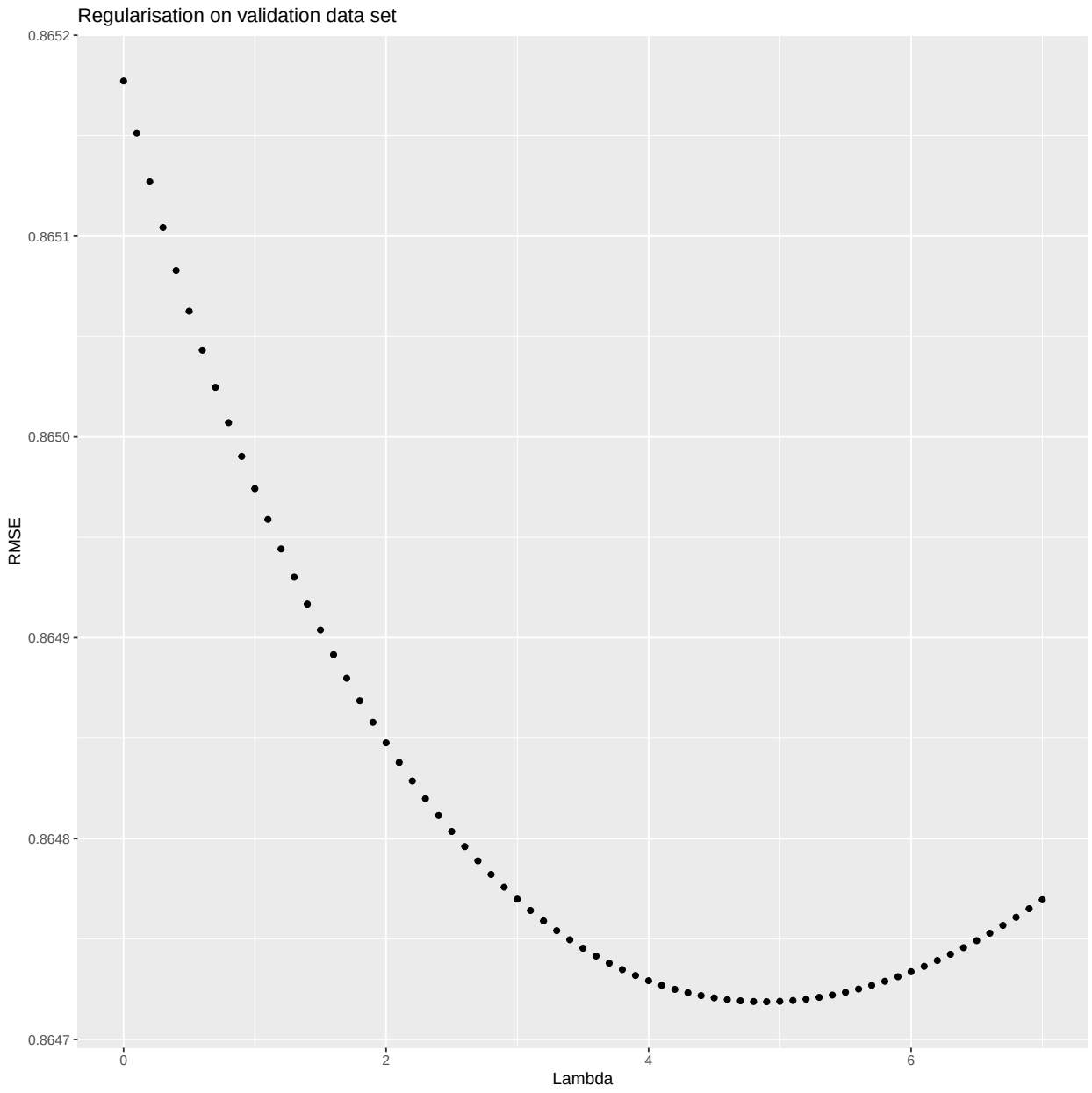
The *regularisation* gives us a higher RMSE than the first "User & Movie Effect" model. This is unexpected but since we have shrunk the population of the original data set, it is kind of expected.

3.7 Regularisation on *validation* data set:

It is time to see the effect of *regularisation* on the validation data set.

```
RMSE_function_val_reg <- sapply(lambda_values, 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_val_reg <- validation %>%
    left_join(m_i, by = "movieId") %>%
    left_join(u_i, by = "userId") %>%
    mutate(pred = mu + m_i + u_i) %>% mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5, 5, pred))) %>%

  return(RMSE(predicted_val_reg, validation$rating, na.rm = T))
})
```

3.8 RMSE_Curve_Regularisation_Validation:



Optimum λ for Validation: 4.9

Final RMSE on Regularization: 0.8647188

4 Results:

Table 10: RMSEs

Method	RMSE
Baseline	1.0600537
User & Movie Effect	0.8244714
User & Movie Effect on validation	0.8651772
User & Movie Effect Regularisation	0.8640320
User & Movie Effect Reg. on validation	0.8647188

We can observe that better RMSE is obtained from the *User & Movie Effect* model. However, this RMSE is *only* obtained on the *test* set. Considering that we must trust more in the performance of the model when we predict from unseen data, we can say that the RMSE that results from the *User & Movie Effect with Regularisation on validation* (the last line in the table above) is our definitive model. This RMSE is obtained when λ=4.9 which has achieved **RMSE equal to** 0.8647188, successfully passing the target of 0.8649.

5 Conclusion:

We started collecting and preparing the dataset for analysis, then we explored the information seeking for insights that might help during model build. Next, we created a random model that predicts the rating based on the probability distribution of each rating. This model gives the worst result.

We started the linear model with a very simple model which is just the mean of the observed ratings. From there, we added movie and user effects, that models the user behavior and movie distribution. With regularization we added a penalty value for the movies and users with few number of ratings.

The variables `userId` and `movieId` have sufficient predictive power and could make better recommendations about movie to specific users of the streaming service. Therefore, the user could decide to spend more time using the service.

The RMSE equal to 0.8647188 is pretty acceptable considering that we have few predictors, but both *User* and *Movie* effects are powerful enough to predict the `rating` that will be given to a movie, by a specific user.

6 Limitations and Future Work:

The model works only for existing users, movies and rating values, so the algorithm must run every time a new user or movie is included, or when the rating changes. This is not an issue for small client base and a few movies, but may become a concern for large data sets. The model should consider these changes and update the predictions as information changes.

There is definitely a time delta effect on the average rating for a movie, based on either Actors associated to the movie or new movies that have similar genres and similar story lines that could inspire more users to rate older movies and hence can bring a significant change to the predicting model.

Only two predictors are used, the movie and user information, not considering other features. Modern recommendation system models use many predictors, such as genres, bookmarks, play lists, history, etc.

From a future perspective we can look at using Matrix Factorization with other ensemble methods to create a much more powerful predicting algorithm and develop a penalty based system for the whole algorithm to improve its performance over time.