Movie Recommendation System

Using Machine Learning to Predict Movie Ratings

Naman Agrawal

January 1, 2021

Abstract

The purpose of this project is to use different machine learning approaches to develop a movie recommendation system. The project will first provide an introduction to recommendation systems. This is followed by a description of the data set used to build the system. Some exploratory data analysis has also been done to provide a qualitative and quantitative explanation of the various characteristics of the data set. The project then illustrates the various machine learning models that will be used to compute the required predictions. A comparative analysis of these models has also been done. The project concludes with a brief overview of the results and highlights some future considerations.

Contents

Introduction

The last few decades have witnessed a massive expansion in digital information, giving rise to various Information Filtering (IF) systems. These systems can be used to filter and analyze vast quantities of data through specific algorithms. Recommendation systems are one class of these systems. They can be used to predict user preference for a given item based on user profile as well as the nature of that item. Recommendation systems are of three types: Content Based Recommendation Systems, Collaborative Filtering Recommendation Systems and Hybrid Recommendation Systems. This project will try to build one Hybrid Movie Recommendation System. Such a system would compare the preferences of similar users (collaborative filtering) as well as offer movies that share characteristics with other movies, that were given preference by the user (content-based). To build this system, a model-based approach will be followed, wherein a variety of intermediate-models will be developed based on different Machine Learning Techniques.

Collaborative-filtering approach will also be used to build a second model through the recosystem package. The assessment of all of these models will be done through their Root Mean Squared Error (RMSE).

The project will be based on the following methodology:

1. The data set is downloaded, and split into train set (edx) and test set (validation). The train set is used to build the system and the test set is used exclusively for validation.
2. The data sets are pre-processed to add on more parameters (or equivalently predictors). Some other parameters, which won't be utilized in the project, are also removed.
3. The features of the train set (edx) are analyzed individually to garner important information regarding their distribution, frequency, summary statistics, etc.
4. Models are constructed using the information regarding the features in the train set to predict the user ratings in the test set
5. A comparative analysis of the different models (based on their root mean squared errors in validation set) is done.

The project will use R for all of the above-mentioned tasks. The following libraries are required:

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(recosystem)
library(ggcorrplot)
library(ggridges)
library(tinytex)
library(kableExtra)
library(knitr)
library(gridExtra)
library(rmarkdown)
library(xfun)
```

Data Set Description

The project uses the 10M version of the MovieLens data set. The data set has been generated by the GroupLens Research Lab. The data can be downloaded and converted into a usable format by the following code:

```
#Creating a temporary file to store the downloaded data:
dl <- tempfile()

#Downloading the required file:
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)


#Extracting and reading the data:

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)

#Assigning the Column names:
colnames(movies) <- c("movieId", "title", "genres")

# Save the data set:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

#Final data set:
movielens <- left_join(ratings, movies, by = "movieId")
```

For further analysis, the data set will be split into train and test sets:

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Making sure that userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Adding the rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

#Removing the elements we don't require:
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

So, two data sets are created:

1. edx: This is the train set. It will be used for exploratory data analysis and to develop a variety of models. This data set will also be used for tuning various parameters through cross validation. It contains 90% of the observations of movielens (a total of 9,000,055 observations).
2. validation: This is the test set. It will be reserved exclusively for testing the final model. A final comparative analysis will also be presented based on the Root Mean Squared Error of the models on the validation set. It contains 10% of the observations of movielens (a total of 999,999 observations).

Before, further analysis, the data set is pre-processed as follows:

```
edx <- edx %>% mutate(month_rating = month(as_datetime(timestamp)),
                      year_release = as.numeric(str_sub(title, -5, -2)),
                      year_rating = year(as_datetime(timestamp)),
                      years_lapsed = year_rating - year_release) %>%
  select(-c("genres", "timestamp", "title"))


validation <- validation %>% mutate(month_rating = month(as_datetime(timestamp)),
                                    year_release = as.numeric(str_sub(title, -5, -2)),
                                    year_rating = year(as_datetime(timestamp)),
                                    years_lapsed = year_rating - year_release) %>%
  select(-c("genres", "timestamp", "title"))
```

Here, columns indicating year of release of movie (year_release), year of user rating (year_rating), month of user rating (month_rating), and years lapsed between release of movie and year of rating (years_lapsed) are calculated and added. Moreover, columns containing information about movie title, genres, and timestamp (after extracting the useful information) are removed from the data set. These features would not be used in the analysis or the development of models. They also occupy the memory of global environment and therefore increase the computational time. Hence, they are excluded.

The resultant data set edx has 7 attributes and 9,000,055 observations. The project shall now describe the different features of the edx data set:

Table 1: Features Information

Feature	Explanation
userId	Unique ID for the User
movieId	Unique ID for the Movie
rating	A rating between 0 and 5 for the movie
month_rating	Indicates the month of the rating: 1 indicates January and 12 indicates December
year_release	Indicates the year of release of the movie
year_rating	Indicates the year of the rating
years_lapsed	Shows the years lapsed between release of movie and year of rating (calculated as the difference between year_rating and year_release

The following code describes the summary statistics of the relevant features of the edx data set:

```
summary(edx)
```

```
##      userId         movieId          rating       month_rating
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   : 1.000
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.: 4.000
```

```
##  Median :35738   Median : 1834   Median :4.000   Median : 7.000
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   : 6.786
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:10.000
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :12.000
##   year_release   year_rating    years_lapsed
##  Min.   :1915   Min.   :1995   Min.   :-2.00
##  1st Qu.:1987   1st Qu.:2000   1st Qu.: 2.00
##  Median :1994   Median :2002   Median : 7.00
##  Mean   :1990   Mean   :2002   Mean   :11.98
##  3rd Qu.:1998   3rd Qu.:2005   3rd Qu.:16.00
##  Max.   :2008   Max.   :2009   Max.   :93.00
```

One surprising observation is that the years_lapsed is seen to have a negative value. This is likely due to entry errors (an error in the noted timestamp or a wrong year printed beside the movie title). The following code describes the structure of the edx data set:

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  7 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 ...
##  $ month_rating: num  8 8 8 8 8 8 8 8 8 8 ...
##  $ year_release: num  1992 1995 1995 1994 1994 ...
##  $ year_rating : num  1996 1996 1996 1996 1996 ...
##  $ years_lapsed: num  4 1 1 2 2 2 2 2 2 2 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The first 6 rows of the data set 'edx' have been displayed for viewing here.

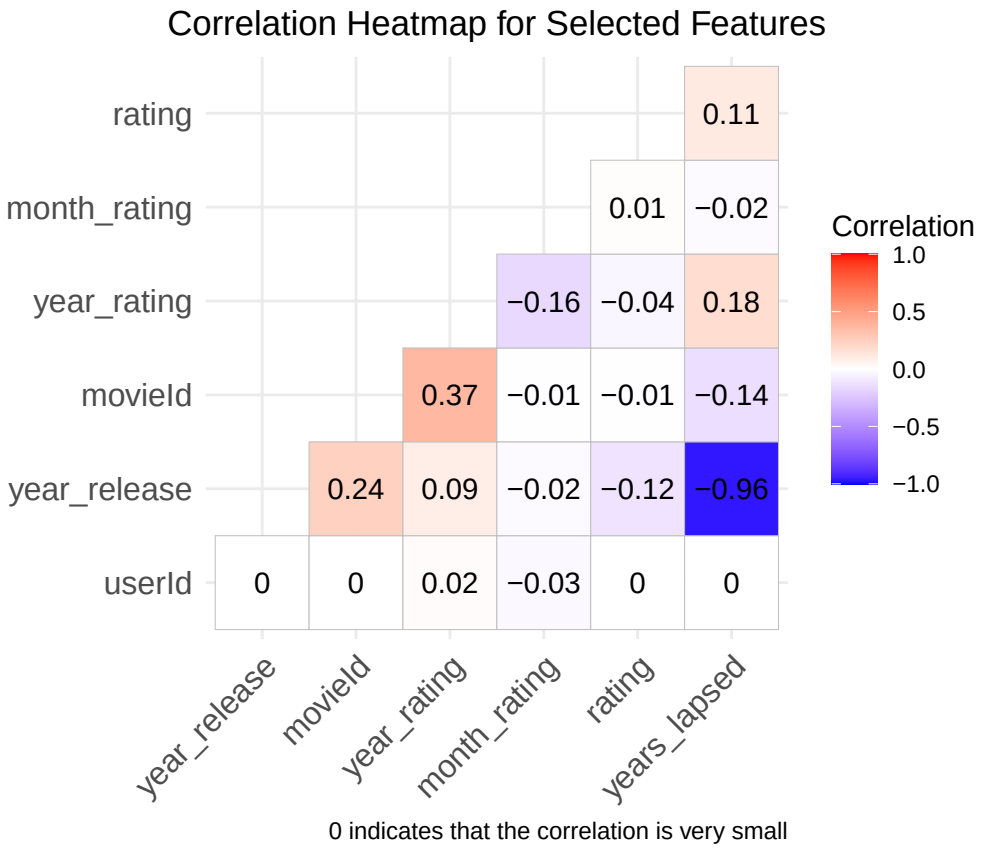
Table 2: First 6 Rows of the Data Set

userId	movieId	rating	month_rating	year_release	year_rating	years_lapsed
1	122	5	8	1992	1996	4
1	185	5	8	1995	1996	1
1	292	5	8	1995	1996	1
1	316	5	8	1994	1996	2
1	329	5	8	1994	1996	2
1	355	5	8	1994	1996	2

Exploratory Data Analysis

Here, different graphs and associated statistics are presented to familiarize the reader with the characteristics of the edx data set. A correlation heat map has been created for the numeric features of the data set:

```
ggcorrplot::ggcorrplot(cor(edx), hc.order = TRUE, type = "lower",
                       lab = TRUE, legend.title = "Correlation") +
  theme(plot.title = element_text(hjust = 0.5)) +
  labs(title="Correlation Heatmap for Selected Features",
       caption="0 indicates that the correlation is very small")
```



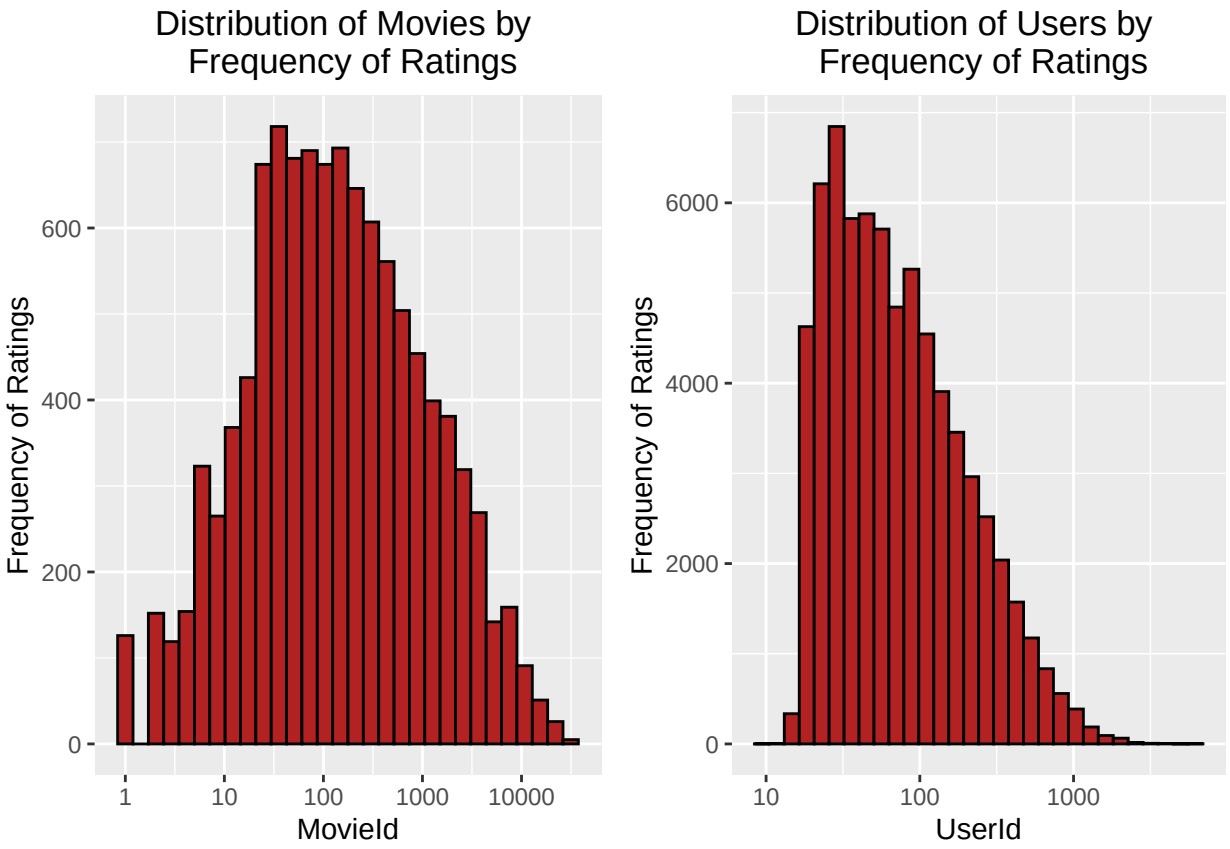
The correlation heat map shows the pearson correlation coefficients between different features. In most cases the correlation is negligible, except in obvious cases (for example: years_lapsed and year_release; year_release and movieId).

The number of unique movies and users in edx can be calculated using the following code:

```
edx %>%
  summarise(number_of_users = n_distinct(userId),
            nunmber_of_movies = n_distinct(movieId))
```

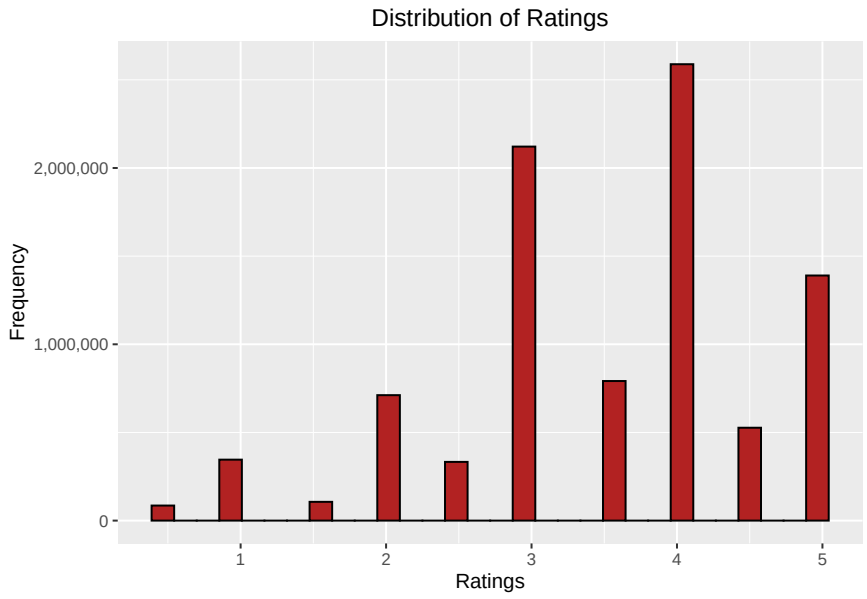
```
##   number_of_users nunmber_of_movies
## 1           69878             10677
```

The following graphs describe the distribution of movies and users by the frequency of ratings:

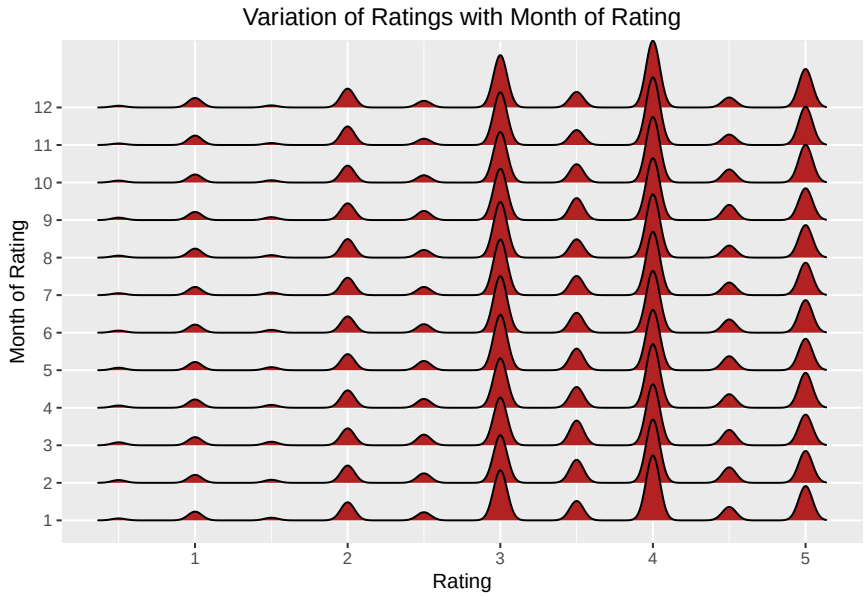


There are two important observations from the above plot. First, some movies get rated many times more than other movies. Second, some users are more active in rating movies than other users.

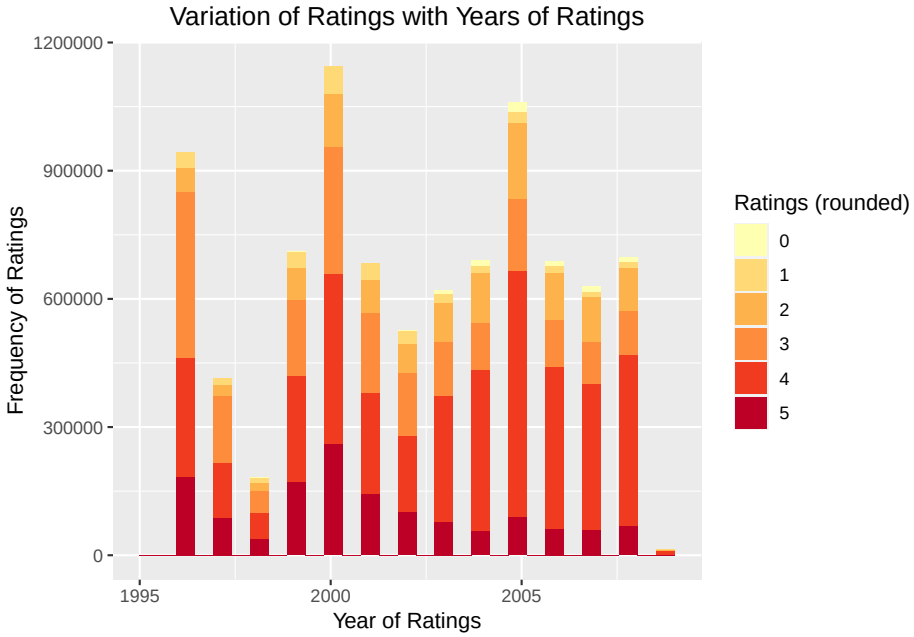
To get a visual idea of the distribution of different ratings, the following bar plot has been made:



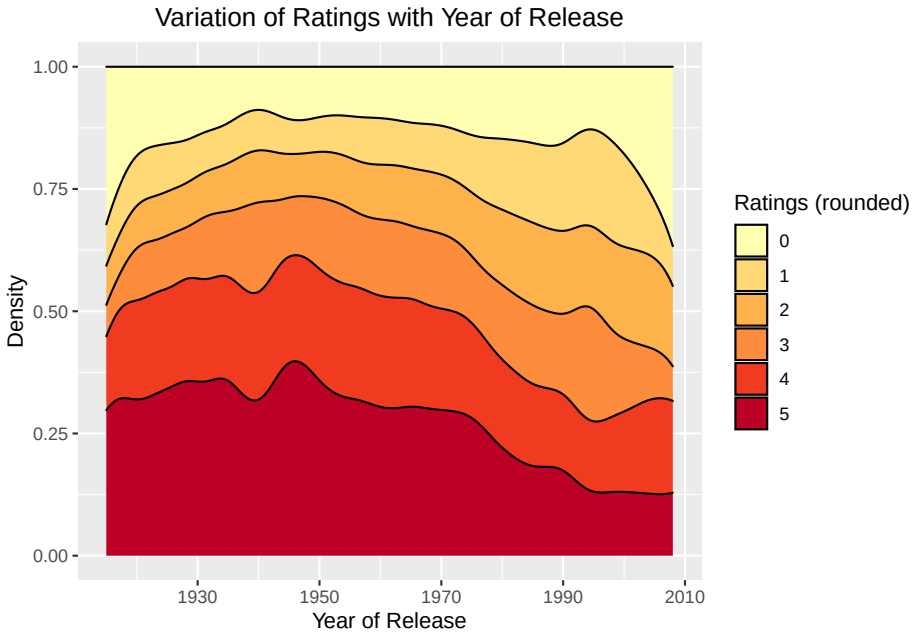
The bar plot shows that users have a higher tendency to rate in whole numbers. It also shows that in general users tend to give more 3 or 4 star ratings. To get a clear idea of the distribution of ratings (rounded) against the month of the rating, the following ridge plot has been made



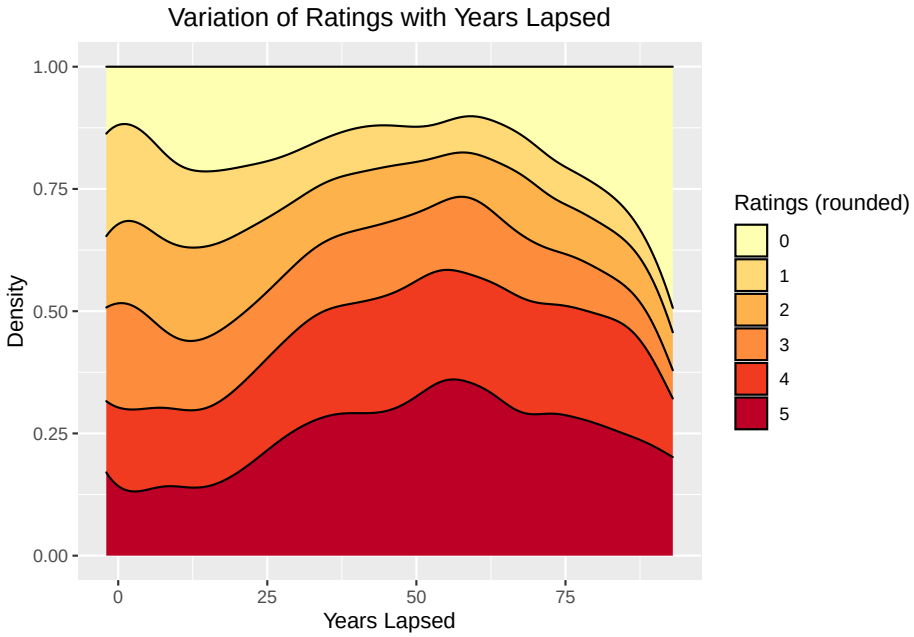
This plot shows that the ratings seems to show a very small variation with the month in which the rating was made. The following bar plot shows the variation of ratings (rounded) with the the year of rating:



The plot shows that the ratings vary irregularly with the year in which the rating was made. Most of the ratings were made in 2000. The following stacked density plot shows the variation of ratings (rounded) with the the year of release:



The plot shows that older movies tend to have a higher proportion of extreme ratings (5s and 0s(rounded)). The variation in other ratings is not very significant. The following stacked density plot shows the variation of ratings (rounded) with the the years lapsed between release and rating:



The plot shows that the extreme ratings (5s and 0s(rounded)) tend to come up more often when more years are lapsed between release and rating.

Machine Learning Models

Two main models will be developed in this project. These models will be tested on the validation set and evaluated on the basis of Root Mean Squared Error (RMSE) metric. RMSE is the loss function that quantifies the deviation of the predicted values (predicted ratings) from the actual values (actual ratings). It is mathematically expressed as:

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

Here N refers to the total number of user-movie combinations in the data set $\hat{y_{u,i}}$ refers to the predicted ratings for movie i, and user u, $y_{u,i}$ refers to the actual ratings (in validation set) for movie i, and user u. In R, the following function is defined to calculate the value of RMSE, whenever required:

```
true_ratings <- validation$rating
RMSE <- function(predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Now, the project will attempt to build the models used for prediction.

Hybrid Models

Here, two kinds of models will be described: Unregularized and Regularized models:

Unregularised models: In unregularized models, no additional error term is accounted to penalize large estimates with small sample sizes. The final model will be developed through a variety of intermediate models:

Intermediate Model 1: The simplest model will assume that all movies will be rated the same, regardless of the movie or the user. Such a model can be expressed as follows:

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

Where $y_{u,i}$ refers to actual ratings for movie i by user u in validation set, and μ refers to the mean ratings of all movies by all users in the train set. The model can be run using the following code:

```
#Defining mean ratings (mu):
mu <- mean(edx$rating)

#Applying the model on validation set:
predicted_ratings_model_1 <- validation %>%
  mutate(pred = mu ) %>%
  pull(pred)
```

The RMSE for the model can be calculated using the RMSE function, previously defined.

```
#Calculating the RMSE:
model_1_rmse <- RMSE(predicted_ratings_model_1)
model_1_rmse
```

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

The RMSE for this model is about 1.0612. This is quite high, and therefore not useful.

Intermediate Model 2: Intuitively, as well as evidently from the data set, it is obvious that some movies get rated better than others. To improvise upon the performance of the model, the movie effect can be included. The new model can be expressed as follows:

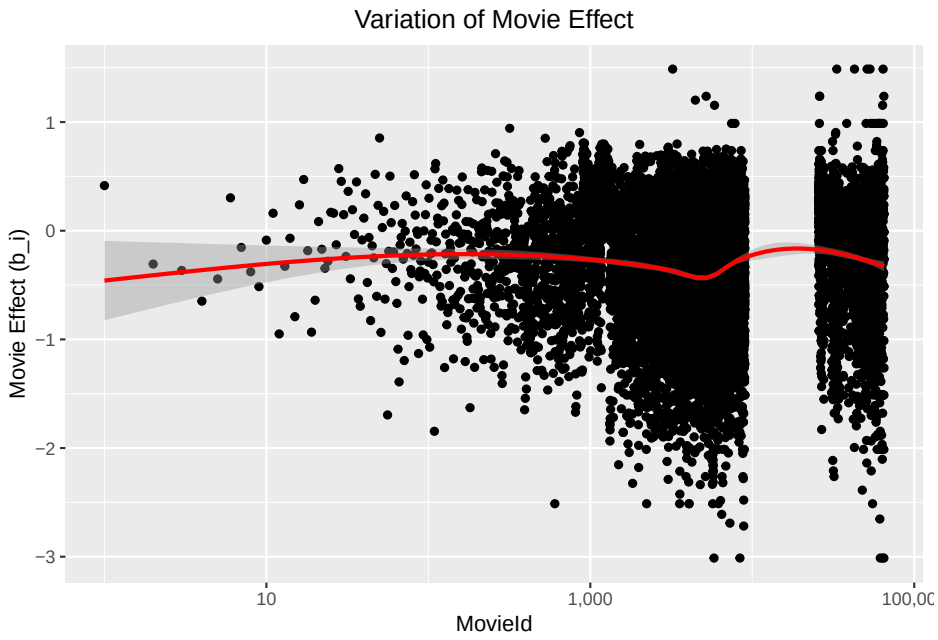
$$y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Here b_i refers to the effect on ratings due to movie i. The following code enables us to calculate and apply the movie effect:

```
#Defining movie effect:
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating-mu))

#Applying the model on validation set:
predicted_ratings_model_2 <- validation %>%
  left_join(movie_avg, by="movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
```

To show the variation of b_i with the movieId, the following scatter plot has been made:



The plot shows that the movie effect varies very irregularly with the movieId. The RMSE for the model can be calculated using the RMSE function, previously defined.

```r
#Calculating the RMSE:
model_2_rmse <- RMSE(predicted_ratings_model_2)
model_2_rmse
```

```
## [1] 0.9439087
```

This reduces the RMSE by about 11%.

Intermediate Model 3: Another way to improve the model's performance is to account for the user effect. The model can be expressed as follows:

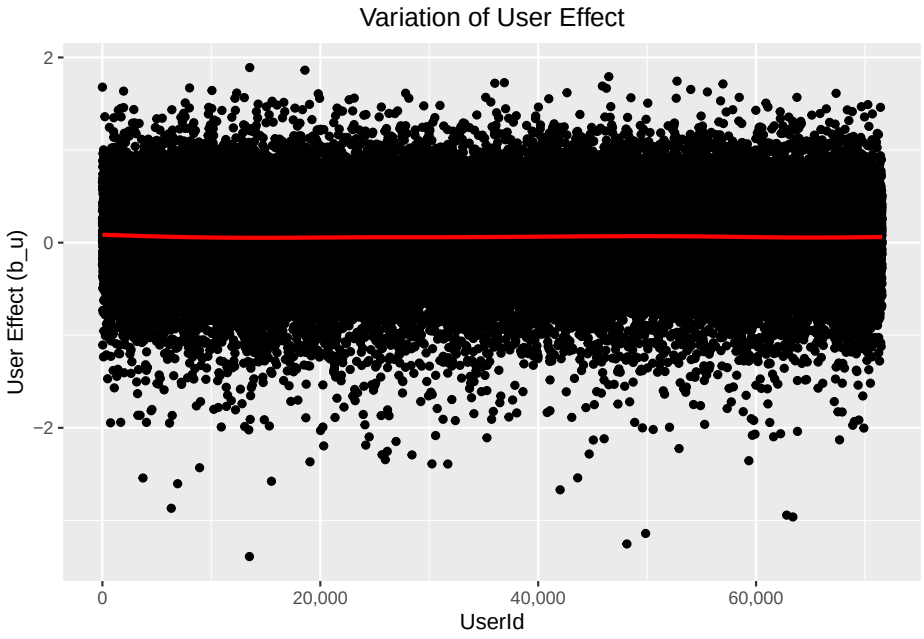
$$y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Here b_u refers to the effect on ratings due to user u. The following code calculates and applies the user effect:

```r
#Defining user effect:
user_avg <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating-mu-b_i))

#Applying the model on validation set:
predicted_ratings_model_3 <- validation %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  mutate(pred = mu + b_i + b_u ) %>%
  pull(pred)
```

To show the variation of b_u with the userId, the following scatter plot has been made:



The plot shows that user effect varies very irregularly with change in userId. The RMSE for the model can be calculated using the RMSE function, previously defined.

```
#Calculating the RMSE:
model_3_rmse <- RMSE(predicted_ratings_model_3)
model_3_rmse
```

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

This further reduces the RMSE by about 8.3%. Now the project will try to include several aspects related to time to account as much variability as possible.

Intermediate model 4: Here, the models are further updated to account for the month of rating effect (the effect on ratings due to the month in which the user made the ratings). The modified model is expressed as follows:

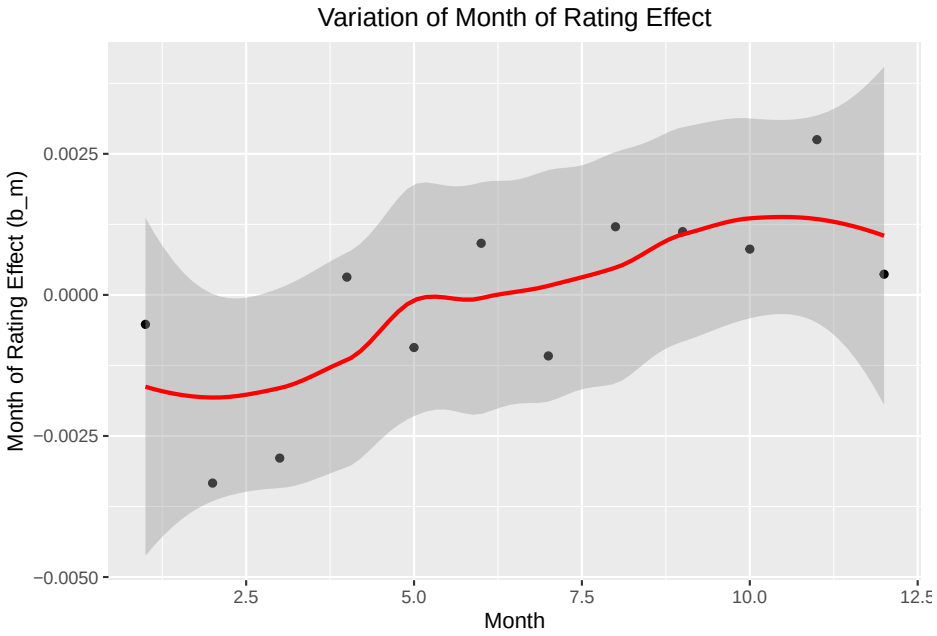
$$y_{u,i} = \mu + b_i + b_u + b_m + \epsilon_{u,i}$$

Here b_m refers to the effect on ratings due to the month of rating m. The following code calculates and applies the month of rating effect:

```
#Defining month of rating effect:
month_rating_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  group_by(month_rating) %>%
  summarise(b_m = mean(rating-mu-b_i-b_u))

#Applying the model on validation set:
predicted_ratings_model_4 <- validation %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  mutate(pred = mu + b_i + b_u + b_m ) %>%
  pull(pred)
```

To show the variation of b_m with the month of rating, the following scatter plot has been made:



The plot shows that in general as more months pass by, b_m increases i.e. the users are likely to give a higher rating. The RMSE for the model can be calculated using the RMSE function, previously defined.

```
#Calculating the RMSE:
model_4_rmse <- RMSE(predicted_ratings_model_4)
model_4_rmse
```

```
## [1] 0.8653459
```

This contributes to a very low reduction in RMSE (around 0.00033%).

Intermediate model 5: The models will now account for the year of release effect (the effect on ratings due to the year in which the movie was released). The modified model is expressed as follows:

$$y_{u,i} = \mu + b_i + b_u + b_m + b_y + \epsilon_{u,i}$$

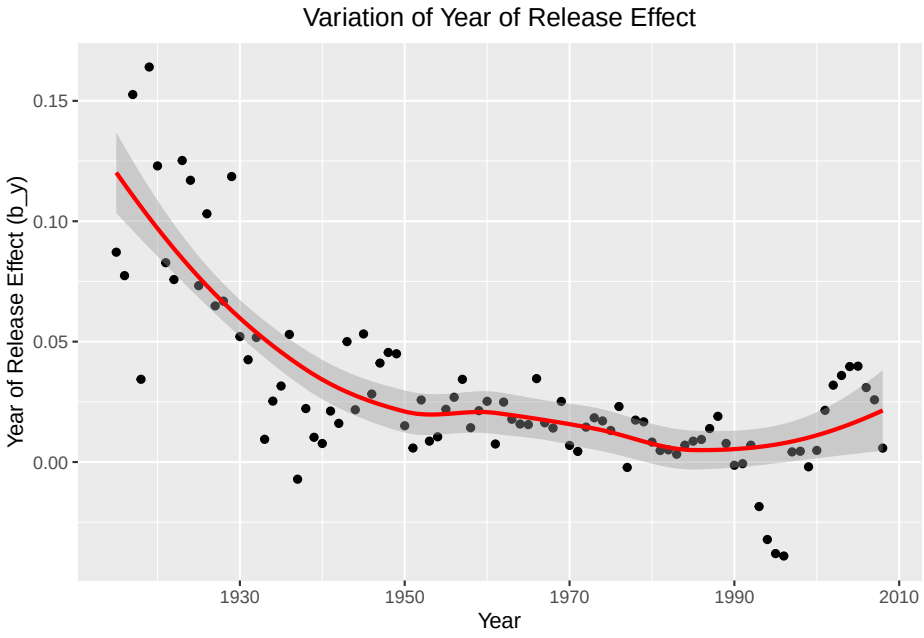
Here b_y refers to the effect on ratings due to the year of release y. The following code calculates and applies the year of release effect:

```
#Defining year of release effect:
year_release_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  group_by(year_release) %>%
  summarise(b_y = mean(rating-mu-b_i-b_u-b_m))

#Applying the model on validation set:
predicted_ratings_model_5 <- validation %>%
  left_join(movie_avg, by="movieId") %>%
```

```
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  mutate(pred = mu + b_i + b_u + b_m +b_y) %>%
  pull(pred)
```

To show the variation of b_y with the year of movie release, the following scatter plot has been made:



The plot shows that in general for older movies, b_y is higher i.e. the users are likely to give a higher rating. The RMSE for the model can be calculated using the RMSE function, previously defined.

```
#Calculating the RMSE:
model_5_rmse <- RMSE(predicted_ratings_model_5)
model_5_rmse
```

```
## [1] 0.8650001
```

The inclusion of this effect, thus reduces the RMSE by 0.04% to around 0.865.

Intermediate model 6: The models were previously updated to include month of rating effect. Now, the project shall try to quantify and determine the year of rating effect The modified model is expressed as follows:

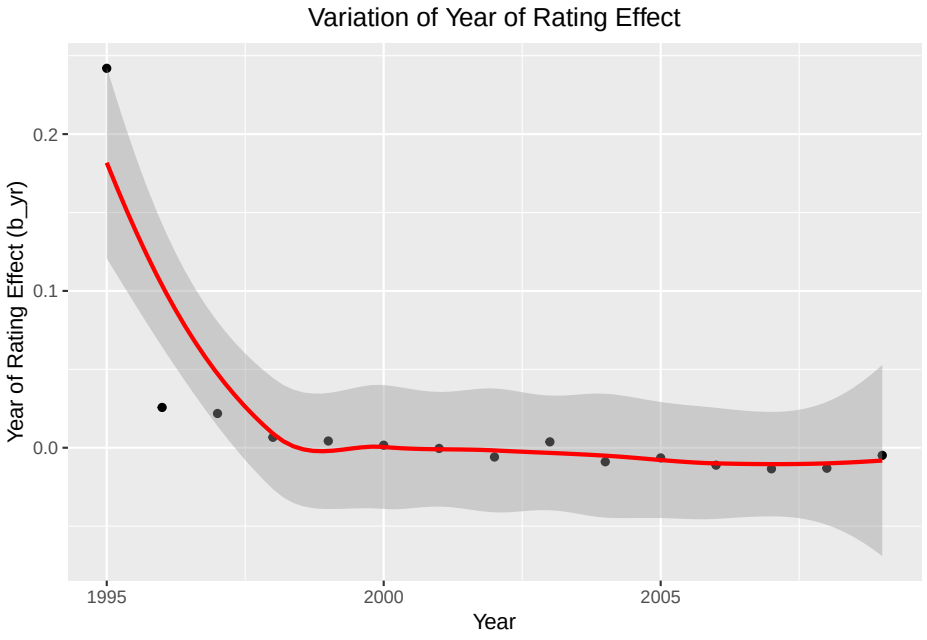
$$y_{u,i} = \mu + b_i + b_u + b_m + b_y + b_{yr} + \epsilon_{u,i}$$

Here b_{yr} refers to the effect on ratings due to the year of rating yr. The following code calculates and applies the year of rating effect:

```r
#Defining year of rating effect:
year_rating_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  group_by(year_rating) %>%
  summarise(b_yr = mean(rating-mu-b_i-b_u-b_m-b_y))

#Applying the model on validation set:
predicted_ratings_model_6 <- validation %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  left_join(year_rating_effect, by="year_rating") %>%
  mutate(pred = mu + b_i + b_u + b_m + b_y + b_yr) %>%
  pull(pred)
```

To show the variation of b_{yr} with the year of rating, the following scatter plot has been made:



The plot shows that the year of rating effect does not vary much with the year of rating (except in case of early years, during the late 19th century). The RMSE for the model can be calculated using the RMSE function, previously defined.

```r
#Calculating the RMSE:
model_6_rmse <- RMSE(predicted_ratings_model_6)
model_6_rmse
```

```
## [1] 0.8649263
```

The inclusion of this effect, thus reduces the RMSE by 0.008%.

Intermediate model 7: The models will be finally updated to include one additional effect: the years lapsed effect. This effect will account for the differences between year of rating and the year of release. The final unregularised model can be expressed as follows:

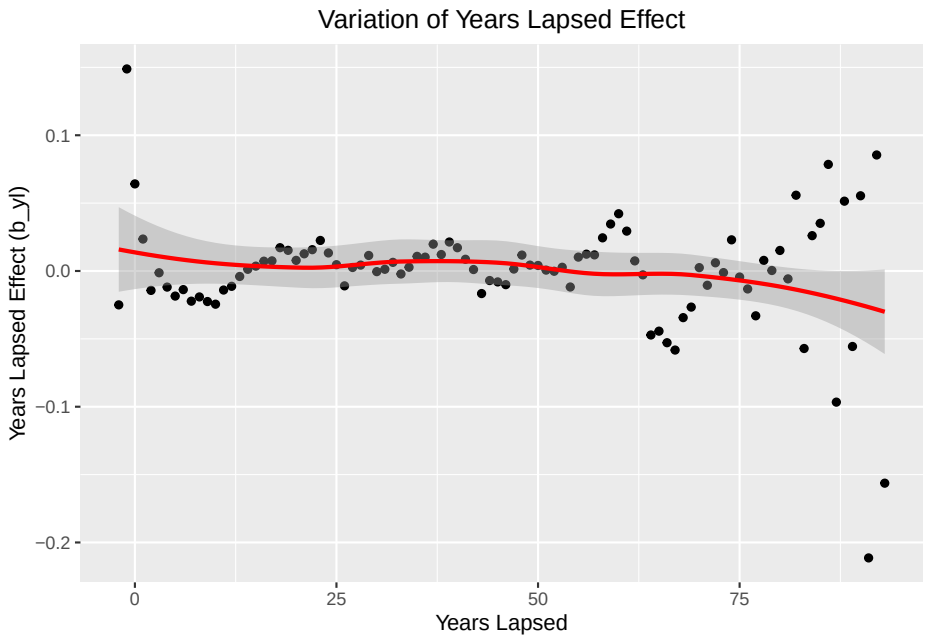
$$y_{u,i} = \mu + b_i + b_u + b_m + b_y + b_{yr} + b_{yl} + \epsilon_{u,i}$$

Here b_{yl} refers to the effect on ratings due to the years lapsed yl. The following code calculates and applies the years lapsed effect:

```
#Defining year lapsed effect:
years_lapsed_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  left_join(year_rating_effect, by="year_rating") %>%
  group_by(years_lapsed) %>%
  summarise(b_yl = mean(rating-mu-b_i-b_u-b_m-b_y-b_yr))

#Applying the model on validation set:
predicted_ratings_model_7 <- validation %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  left_join(year_rating_effect, by="year_rating") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  mutate(pred = mu + b_i + b_u + b_m + b_y + b_yr + b_yl) %>%
  pull(pred)
```

To show the variation of b_{yl} with the years lapsed, the following scatter plot has been made:



The plot shows that the years lapsed effect varies very irregularly with the years lapsed. However, for very large values of years lapse, the effect becomes very negative or very positive. This also supports the conclusion drawn from a previously plotted graph (Variation of Ratings with Years Lapsed) that extreme ratings (5s and 0s(rounded)) tend to come up more often when more years are lapsed between release and rating. The RMSE for the model can be calculated using the RMSE function, previously defined.

```
#Calculating the RMSE:
model_7_rmse <- RMSE(predicted_ratings_model_7)
model_7_rmse
```

```
## [1] 0.8646826
```

The inclusion of this effect, thus reduces the RMSE by 0.02%.

Regularised Models: Regularization is used to penalize large estimates that are formed using small sample sizes. Earlier (under unregularized approach), the final model developed was essentially trying to minimize:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_m - b_y - b_{yr} - {b_{yl}}^2)$$

But, under regularization, another set of terms is added for penalization. The final regularized model error term can be expressed as:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_m - b_y - b_{yr} - b_{yl})^2 + \lambda(\sum_{i}({b_i}^2) + \sum_{u}({b_u}^2) + \sum_{m}({b_m}^2) + \sum_{y}({b_y}^2) + \sum_{yr}({b_{yr}}^2) + \sum_{yl}({b_{yl}}^2))$$

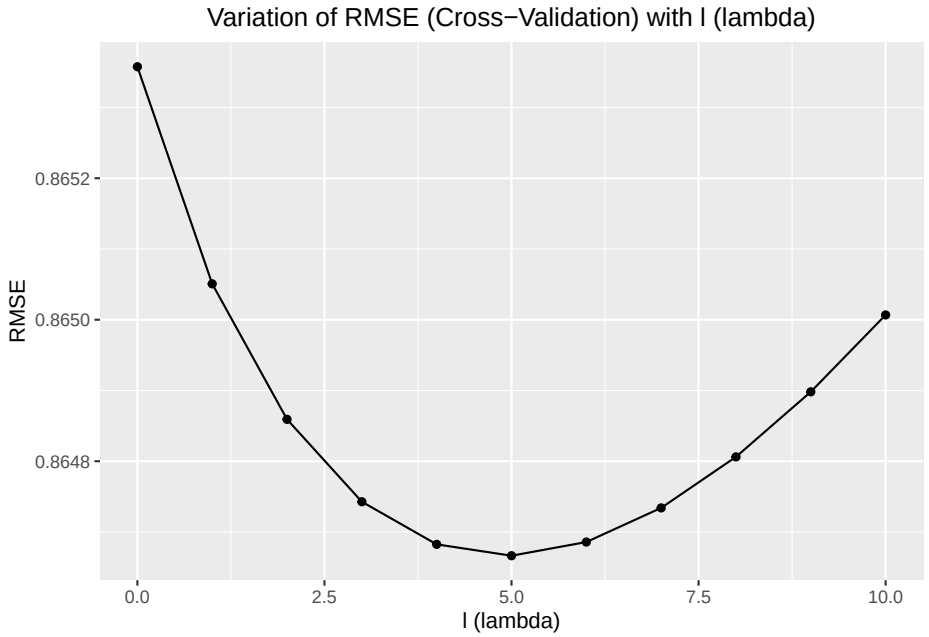
To minimize the above error, it is important to divide every term (in unregularized mode) by λ. To determine the value of λ, parameter tuning is used. For tuning, cross validation is implemented on edx set. The edx set is partitioned into edx train and edx test sets using the following codes:

```
set.seed(820, sample.kind = "Rounding")
#Creating the text index
edx_test_index <- createDataPartition(edx$rating, times=1, p=0.2, list = FALSE)

#Splitting appropriately
edx_train <- edx[-edx_test_index,] #used for training
edx_test <- edx[edx_test_index,] #used for tuning

#To ensure only those movies and users are present in edx_test
#for which we have observations in edx_train
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

It is important to note that the edx set itself is used for training and tuning because the validation set is unseen, and thus unavailable for tuning. The validation set will be used only for determining the value of the RMSE. Subsequently, an algorithm is implemented that calculates the RMSE on edx_test for different values of λ. The following plot shows the variation of RMSE with λ:



The value of λ that minimizes the RMSE is:

```
l[which.min(rmses)]
```

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

Now, this value of λ is used to develop a final regularized model on the entire edx set as follows:

```
#Implementing the required value of l and getting final RMSE on validation set:
l <- l[which.min(rmses)]

#Mean Ratings:
mu <- mean(edx$rating)

#Movie Effect:
movie_avg <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating-mu)/(n()+l))

#User Effect:
user_avg <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating-mu-b_i)/(n()+l))

#Month of Rating Effect:
month_rating_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  group_by(month_rating) %>%
  summarise(b_m = sum(rating-mu-b_i-b_u)/(n()+l))
```

```
#Year of Release Effect:
year_release_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  group_by(year_release) %>%
  summarise(b_y = sum(rating-mu-b_i-b_u-b_m)/(n()+l))

#Year of Rating Effect:
year_rating_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  group_by(year_rating) %>%
  summarise(b_yr = sum(rating-mu-b_i-b_u-b_m-b_y)/(n()+l))

#Years Lapsed Effect:
years_lapsed_effect <- edx %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  left_join(year_rating_effect, by="year_rating") %>%
  group_by(years_lapsed) %>%
  summarise(b_yl = sum(rating-mu-b_i-b_u-b_m-b_y-b_yr)/(n()+l))

#Applying the model on validation set:
predicted_ratings_regularization <- validation %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(year_release_effect, by="year_release") %>%
  left_join(year_rating_effect, by="year_rating") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  mutate(pred = mu + b_i + b_u + b_m + b_y + b_yr + b_yl) %>%
  pull(pred)
```

The RMSE on the final hybrid model can be calculated as follows:

```
#Calculating the RMSE:
rmse_regularization <- RMSE(predicted_ratings_regularization)
rmse_regularization
```

```
## [1] 0.8641502
```

Thus regularization reduces the RMSE by 0.06%

Collaborative Filtering Model

Here, another model is developed base on collaborative filtering. For this model, different user preferences for a movie will be compared to yield a predicted rating for the movie by a given user. This approach will exclude the characteristics of the movie itself (for example, the year in which the movie was released).

The model will use matrix factorization. Here, the user-item interaction matrix will be decomposed to as a product of two smaller matrices. Such a decomposition will allow the model to explain a greater amount of variability than any of the previously shown models. For this project, matrix factorization will be carried out using the the recosystem package in R. The following codes demonstrate the usage of the package to develop the required model.

```
set.seed(820, sample.kind = "Rounding")

# Converting the 'edx' and 'validation' sets to the recosystem input format
#(data_memory specifies a data set from R objects)
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))
validation_reco  <-  with(validation, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))

# Creating the model object (reco() is used for constructing a Recommender System Object)
r <-  recosystem::Reco()

# Training the required model (outputting the factorization matrices)
# nthreads: number of threads for parallel computing:
r$train(edx_reco, opts = c(nthread = 4))

# Calculating the prediction ratings (out_memory: Result should be returned as R objects)
predicted_ratings_reco <-  r$predict(validation_reco, out_memory())
```

Once the final predictions are made, the RMSE for the final Matrix-Factorization model can be calculated as follows:

```
#Calculating the RMSE:
rmse_reco <- RMSE(predicted_ratings_reco)
rmse_reco
```

```
## [1] 0.8324478
```

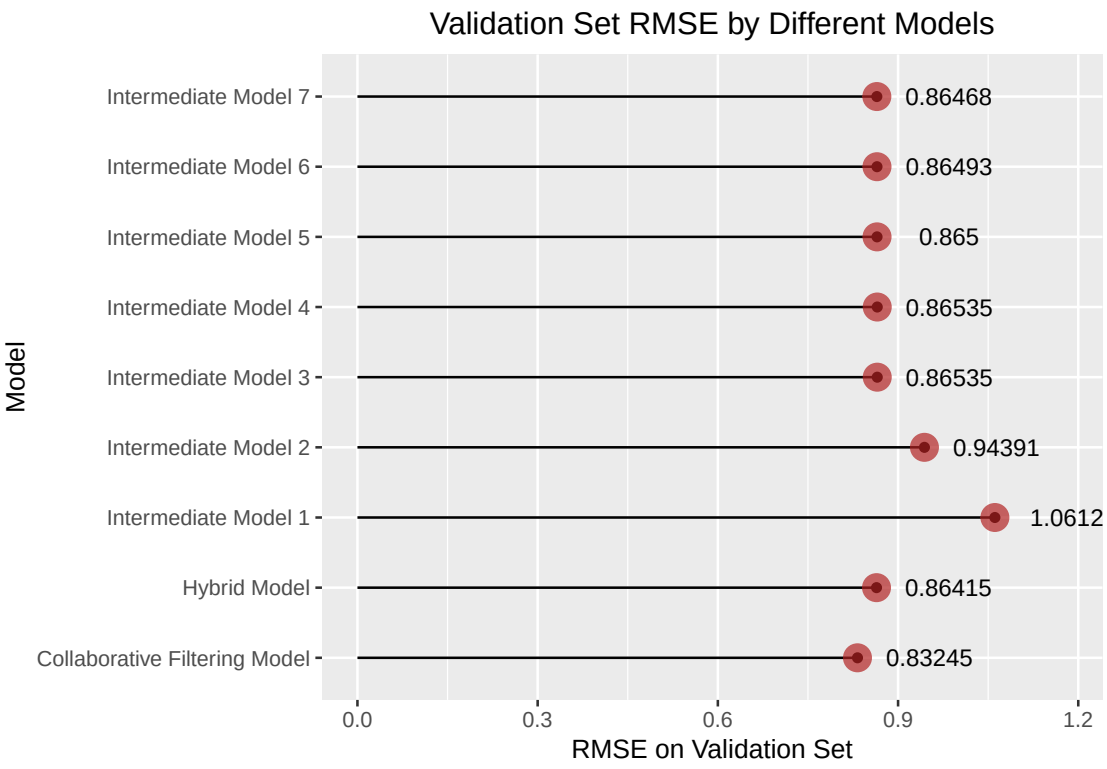
Thus, the RMSE in this model is around 3.6 % lesser than the RMSE produced in the hybrid-regularized model.

Result

After using a variety of final and intermediate models, an RMSE of 0.8324478 was achieved. This was made possible by implementing a collaborative-filtering matrix factorization based movie recommendation system using the recosystem package. The following table describes the results of the RMSE values of the models that have been developed so far.

Model	Method/Expression	RMSE
Intermediate Model 1	$y_{u,i} = \mu + \epsilon_{u,i}$	1.0612018
Intermediate Model 2	$y_{u,i} = \mu + b_i + \epsilon_{u,i}$	0.9439087
Intermediate Model 3	$y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$	0.8653488
Intermediate Model 4	$y_{u,i} = \mu + b_i + b_u + b_m + \epsilon_{u,i}$	0.8653459
Intermediate Model 5	$y_{u,i} = \mu + b_i + b_u + b_m + b_y + \epsilon_{u,i}$	0.8650001
Intermediate Model 6	$y_{u,i} = \mu + b_i + b_u + b_m + b_y + b_{yr} + \epsilon_{u,i}$	0.8649263
Intermediate Model 7	$y_{u,i} = \mu + b_i + b_u + b_m + b_y + b_{yr} + b_{yl} + \epsilon_{u,i}$	0.8646826
Hybrid Model	Regularization	0.8641502
Collaborative Filtering Model	Matrix Factorization	0.8324478

The following graph has been made to give a visual interpretation for the same:



Conclusion

Throughout the project, a variety of models have been used, giving different values of RMSE. Among the various models, the collaborative-filtering matrix factorization based movie recommendation system has given the lowest RMSE of 0.8324478 on the validation set.

Meanwhile, it is imperative to recognize the limitations of the project. Firstly, the computational time for some models, particularly regularization, are quite high (around 5-10 minutes). Secondly, a few important predictors (for example the genre effect) have not been accounted for. Lastly, the RMSE for the model is still about 0.8, which means that the model is able to predict user ratings with a large margin of error of about 0.8 stars.

Future Considerations

To improve upon the performance of the models, it is suggested that the models be constantly retrained on fresh data sets. In future, one could explore different machine learning techniques such as k-nearest neighbors, gradient boosted decision trees, etc. to further reduce the RMSE. In addition, adaptive sorting algorithms can be developed to exploit the properties of a data set to greatly reduce computational time. Finally, the inclusion of additional features such as movie genre, the actors in the movie, the movie duration, number of ratings done by users in one day, etc. can greatly enhance the performance of the movie recommendation systems.

References

1. 'Introduction to Data Science: Data Analysis and Prediction Algorithms with R' by Rafael A. Irizarry
2. 'R Markdown Cookbook' by Yihui Xie, Christophe Dervieux, and Emily Riederer
3. 'R Markdown: The Definitive Guide' by Yihui Xie, J. J. Allaire, and Garrett Grolemund