MovieLens Recommendation System

Pratik Khadse

10/25/2020

Contents

1 Overview

During the last few decades, with the rise of YouTube, Amazon, Netflix and many other such web services, recommendation systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommendation systems are today unavoidable in our daily online journeys.

In a very general way, recommendation systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy or anything else depending on industries). They are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. As a proof of the importance of recommendation systems, we can mention that, a few years ago, Netflix organized a challenges (the "Netflix prize") where the goal was to produce a recommendation system that performs better than its own algorithm with a prize of 1 million dollars to win.

In this project, we try to build a recommendation system, by predicting the ratings a user will give to a movie. Consisting of 10M records, we use the Movielens dataset for our model. A regularized model is employed, selecting the factors which gives the least error in prediction.

The dataset downloaded is first split it into training and validation set, named as "edx" and "validation", with a 90/10 proportion as directed by edx content (problem statement). We check the "edx" set for understanding the effect of different parameters on our target variable. Some Data Cleaning and Preparation steps are performed to obtain additional variables deemed useful for our analysis. These steps are performed on both the "edx" and "validation" set, making sure the said changes maintain the principle of having training and validation sets (Data is only structured differently, additional rows of data are not added). Additional variables are derived, with usefulness of the same confirmed by performing Exploratory Data Analysis on "edx" set. Appropriate loss function (Root Mean Squared Error) is employed to compare models, desired value of which should be below 0.86490 as per our problem statement.

This report takes you through the various stages of project development, explaining various steps of model building and logical reasoning behind the same.

1.1 About Dataset

GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set. The full data set contains 26,000,000 ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users.

We use a smaller dataset with 10M ratings for different movies, consisting of information regarding movie title, genres, unique ids for different users and movies, along with the rating. The following link is used to download the same: http://files.grouplens.org/datasets/movielens/ml-10m.zip

2 Methodology

2.1 Literature Review: Regularized Model

Recommendation systems are complicated machine learning challenges. It is due to the fact that for every cell, essentially the entire matrix can be used as predictor.

For our model, we use estimates for what we call "bias" or also called "Effects". Eg: we start with a simple model which uses mean rating as prediction. This is shown as

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

The Root Mean Squared Error tells us how our predictions vary from the actual target variable values. This loss function is obtained using equation:

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

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

We can improve our prediction earlier by adding a term b_i, which represents average rating for movie i. This is because we know some movies have a higher rating than others.

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

To make further improvements, we can add a term for user effects. We include a term, b_u, which is the user-specific effect. So now if a cranky user- this is a negative b_u- rates a great movie, which will have a positive b_i, the effects counter each other, and we may be able to correctly predict that this user gave a great movie a three rather than a five, which will happen. And that should improve our predictions.

The regularized model constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. To obtain estimates for b's, we minimize the following equation:

$$Lossfunction = \frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda\sum_i b_i^2$$

The first term gives the MSE or Mean Squared Error, the second is a penalty term which gets larger. We obtain the values of b by taking derivatives, arriving at the equation:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})$$

where n_i is number of ratings b for movie i

The same can be extended to other effects as we add them. Example, using movie and user effects, the equation would look as follows:

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

Similarly, the final equation used for making predictions is thus given by,

$$Y_{u,i} = \mu + \underbrace{b_i}_{\text{Movie effect}} + \underbrace{b_u}_{\text{User effect}} + \underbrace{b_{yrate}}_{\text{Year of Movie Rating effect}} + \underbrace{b_{yrel}}_{\text{Year of Movie Release effect}} + \underbrace{b_g}_{\text{Genre effect}} + \epsilon_{u,i}$$

with biases/effects derived by minimizing equation,

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

Where,

i represents movieId

u represents userId

$yrate$ is the year when with movie i was rated by user u

$yrel$ is the year when movie i was released

and

$$b_g = \sum_{k=1}^{K} x_{u,i}^k b_k$$

with $x_{u,i}^k = 1$ if $g_{u,i}$ (genre for movie i rated by user u) belongs to genre k.

We now proceed with building our model to make predictions.

2.2 Importing Libraries and Downloading data

Required libraries are first imported, following which we obtain the "edx" and "validation" sets for training and validation respectively.

2.2.1 Libraries

```
options(digits = 5)

library(tidyverse)
library(caret)
library(data.table)
library(tidyr)
library(lubridate)

#Or use the following if packages are not installed earlier

# 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(tidyr)) install.packages("tidyr",
#                                      repos = "http://cran.us.r-project.org")
# if(!require(lubridate)) install.packages("lubridate",
#                                          repos = "http://cran.us.r-project.org")
```

2.2.2 Create Edx and Validation Sets

```
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

#Note: this takes couple of minutes. This code is provided by Edx community

# Download files
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

## Create dataframe
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies_temp <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies_temp) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies_temp) %>%
  mutate(movieId = as.numeric(movieId),
         title = as.character(title),
         genres = as.character(genres))
```

```r
movielens <- left_join(ratings, movies, by = "movieId")
head(movielens) %>%
  knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

```r
# 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 removed rows from validation set back to edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

#remove unneeded objects
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

2.3 Sanity checks

Here we do a basic sanity check on the training data, checking number of rows, columns and some summary statistics. This is a useful part of Data Understanding.

```r
#rows and columns
dim(edx)
```

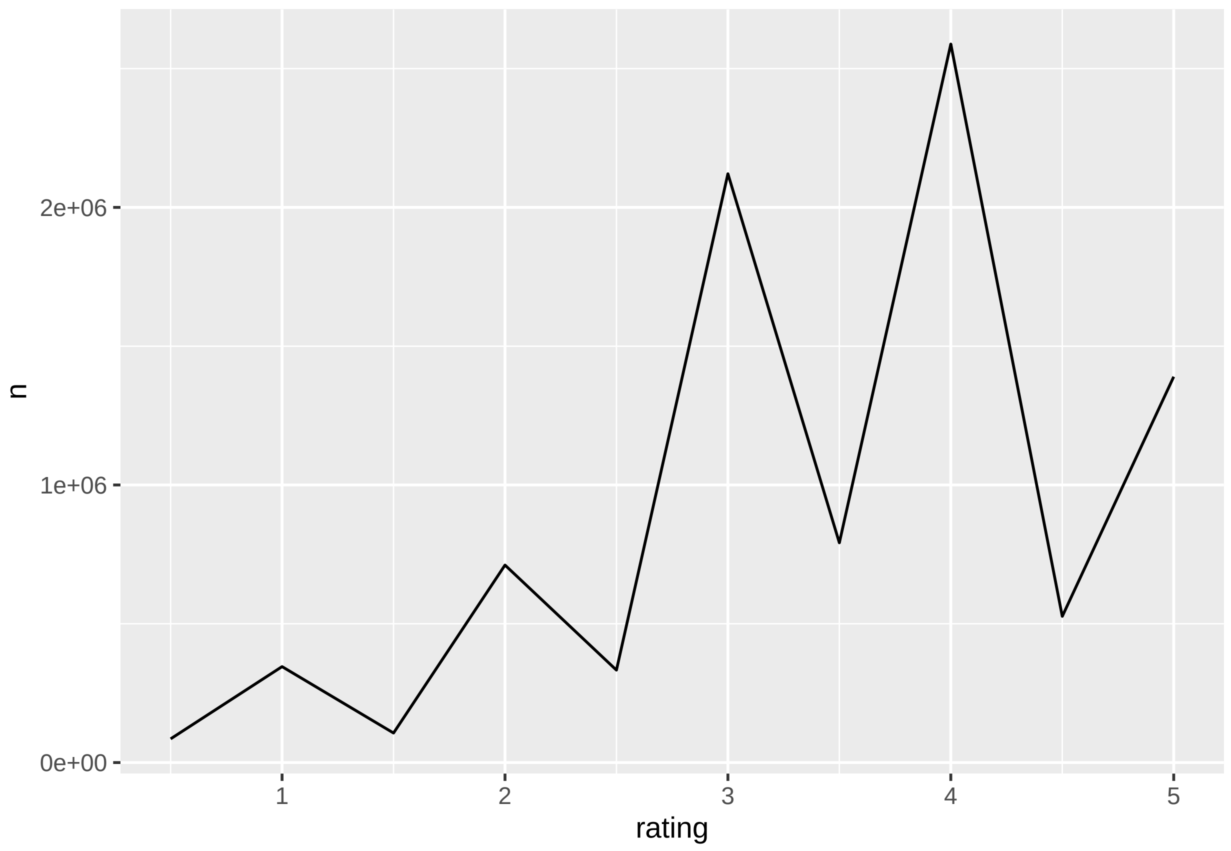
```
## [1] 9000055       6
```

```r
#summary statistics
summary(edx) %>%
  knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.50	Min. :7.90e+08	Length:9000055	Length:9000055
1st Qu.:18124	1st Qu.: 648	1st Qu.:3.00	1st Qu.:9.47e+08	Class :character	Class :character
Median :35738	Median : 1834	Median :4.00	Median :1.04e+09	Mode :character	Mode :character
Mean :35870	Mean : 4122	Mean :3.51	Mean :1.03e+09	NA	NA
3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.00	3rd Qu.:1.13e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.00	Max. :1.23e+09	NA	NA

A plot is made further to see how count of ratings is distributed.

```r
#Half star vs full star ratings
edx %>%
  group_by(rating) %>%
  summarize(n = n()) %>%
  ggplot(aes(x = rating, y = n)) +
  geom_line()
```



Thus, we see high number of full star ratings than half star. This is later also confirmed in our Exploratory Data Analysis for genre.

2.4 Data Cleaning and Preparation

The edx dataset can be seen consisting of following variables:-

1. userId: unique ID for each user who rated one or more movies
2. movieId: unique ID for each movie that received ratings from one or more users.
3. rating: a rating given on a scale of 1 to 5.
4. timestamp: containing information about when the movie was rated
5. title: The title of the movie, along with the year in which the movie was released.
6. genres: the movie genres associated to a movie

For our analysis, certain variables shall be derived for better performance. Year of movie release, year of movie rating and columns for individual genre form useful information that are included in our model. These are obtained as shown below.

2.4.1 Obtain YearOfRating from timestamp

```
edx <- edx %>%
  mutate(YearOfRating = year(as_datetime(timestamp)))

validation <- validation %>%
  mutate(YearOfRating = year(as_datetime(timestamp)))
```

The changes are also made in validation set, while respecting the principle behind training and testing sets, i.e., already present information is structured differently, without adding new information to validation set.

2.4.2 Obtain YearOfRelease from title

```
# for edx
edx <- edx %>%
  mutate(temp = str_extract(edx$title, "\\(\\d{4}\\)")) %>%
  mutate(YearOfRelease = as.numeric(str_extract(temp, "\\d{4}"))) %>%
  select(-temp)%>%
  select(userId, movieId, title, YearOfRelease, YearOfRating, genres, rating)

# for validation

validation <- validation %>%
  mutate(temp = str_extract(title, "\\(\\d{4}\\)")) %>%
  mutate(YearOfRelease = as.numeric(str_extract(temp, "\\d{4}"))) %>%
  select(-temp) %>%
  select(userId, movieId, title, YearOfRelease, YearOfRating, genres, rating)
```

2.4.3 Generate single rows for every genre value

Alongside generating individual columns for different genre, we generate a separate training (edx1) set with genres in different rows. This is useful for EDA, grouping by genre, which will be seen further.

```
# for edx
edx1 <- edx %>%
  separate_rows(., genres, sep = "\\|")
```

2.4.4 Create Individual columns for genres

```
#Convert genres to individual columns
#Note we have to consider all factors of genres,
#so there is no need to exclude one, for the dummy variable trap is inapplicable here
genre <- unique(edx1$genres)

# for edx
for (element in genre){
  edx <- edx %>%
    mutate(!!element :=
             ifelse(str_detect(edx$genres, element), 1, 0))
  ## We use Dynamic Variable here

}

edx <- edx %>%
  select(-genres)

# for validation
for (element in genre){
  validation <- validation %>%
    mutate(!!element :=
             ifelse(str_detect(validation$genres, element), 1, 0))
  ## We use Dynamic Variable here

}

validation <- validation %>%
  select(-genres)
```

2.4.5 Rename genre columns for better accessibility

We do some cleaning, renaming to replace spaces and hyphens with underscores, and removing spaces for better accessibility later.

```
#replace -, spaces with _ and remove ()

# for edx
colnames(edx)[str_detect(names(edx), pattern = "-")] <-
  str_replace_all(colnames(edx)[str_detect(names(edx), pattern = "-")],
              pattern = "-",
              replacement = "_")

colnames(edx)[str_detect(names(edx), pattern = "\\(")] <-
  str_replace_all(colnames(edx)[str_detect(names(edx), pattern = "\\(")],
              pattern = "\\s",
```

```r
                replacement = "_")

colnames(edx)[str_detect(names(edx), pattern = "\\(")] <-
  str_replace_all(colnames(edx)[str_detect(names(edx), pattern = "\\(")],
                pattern = "\\(|\\)",
                replacement = "")

# for validation

colnames(validation)[str_detect(names(validation), pattern = "-")] <-
  str_replace_all(colnames(validation)[str_detect(names(validation),
                                                  pattern = "-")],
                pattern = "-",
                replacement = "_")

colnames(validation)[str_detect(names(validation), pattern = "\\(")] <-
  str_replace_all(colnames(validation)[str_detect(names(validation),
                                                  pattern = "\\(")],
                pattern = "\\s",
                replacement = "_")

colnames(validation)[str_detect(names(validation), pattern = "\\(")] <-
  str_replace_all(colnames(validation)[str_detect(names(validation),
                                                  pattern = "\\(")],
                pattern = "\\(|\\)",
                replacement = "")
```

After completing all data cleaning and preparation operations above, our "edx" and "validation" set look as follows.

```r
head(edx) %>%
  knitr::kable(format = "latex", booktabs = TRUE) %>%
          kableExtra::kable_styling(latex_options = "scale_down")
```

userId	movieId	title	YearOfRelease	YearOfRating	rating	Comedy	Romance	Action	Crime	Thriller	Drama	Sci_Fi	Adventure	Children	Fantasy	War	Animation	Musical	Western	Mystery	Film_Noir	Horror	Documentary	IMAX	no_genres_listed
1	122	Boomerang (1992)	1992	1996	5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	185	Net, The (1995)	1995	1996	5	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	292	Outbreak (1995)	1995	1996	5	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	316	Stargate (1994)	1994	1996	5	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	329	Star Trek: Generations (1994)	1994	1996	5	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	355	Flintstones, The (1994)	1994	1996	5	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

```r
head(validation) %>%
  knitr::kable(format = "latex", booktabs = TRUE) %>%
          kableExtra::kable_styling(latex_options = "scale_down")
```

userId	movieId	title	YearOfRelease	YearOfRating	rating	Comedy	Romance	Action	Crime	Thriller	Drama	Sci_Fi	Adventure	Children	Fantasy	War	Animation	Musical	Western	Mystery	Film_Noir	Horror	Documentary	IMAX	no_genres_listed
1	231	Dumb & Dumber (1994)	1994	1996	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	480	Jurassic Park (1993)	1993	1996	5	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	586	Home Alone (1990)	1990	1996	5	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	151	Rob Roy (1995)	1995	1997	3	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
2	858	Godfather, The (1972)	1972	1997	2	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1544	Lost World: Jurassic Park, The (Jurassic Park 2) (1997)	1997	1997	3	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0

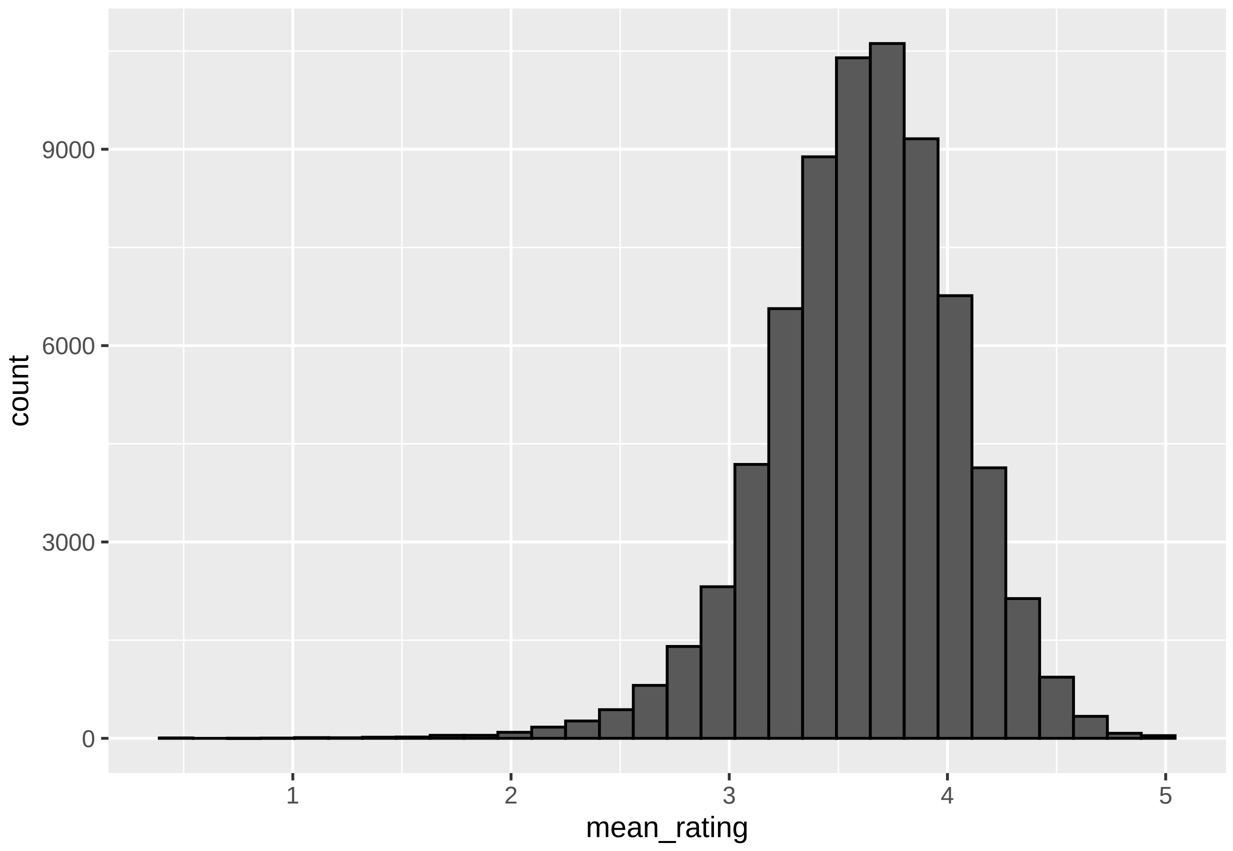
We now advance to Understanding relations between variables.

2.5 Exploratory Data Analysis: Check patterns in variables

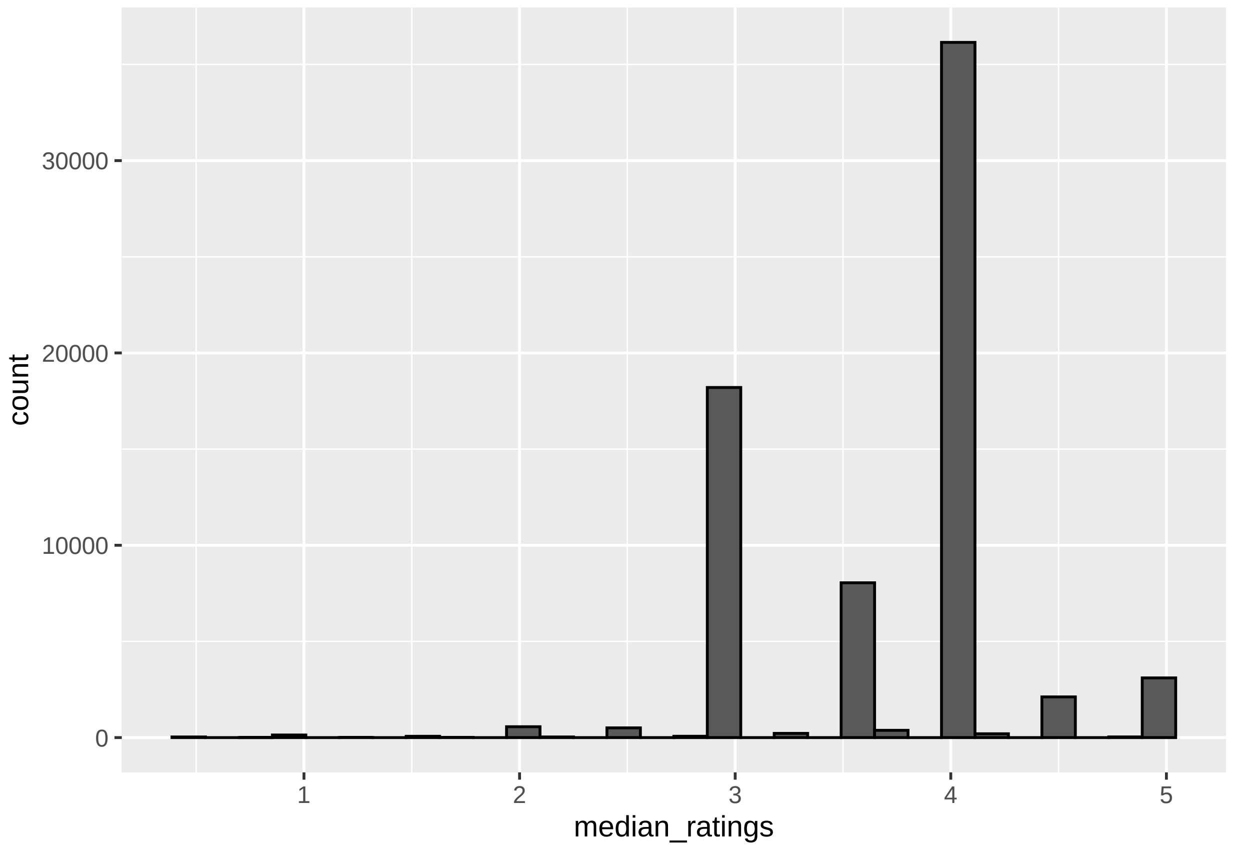
In order to understand the impact of above specified variables on the target variable "rating", we perform Exploratory Data Analysis.

2.5.1 UserId

```
# Plotting mean ratings of users: we see the distribution is left skewed
# Also considerable variation is seen, which makes it an important variable

edx %>%
  group_by(userId) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(x = mean_rating)) +
  geom_histogram(color = "black")
```



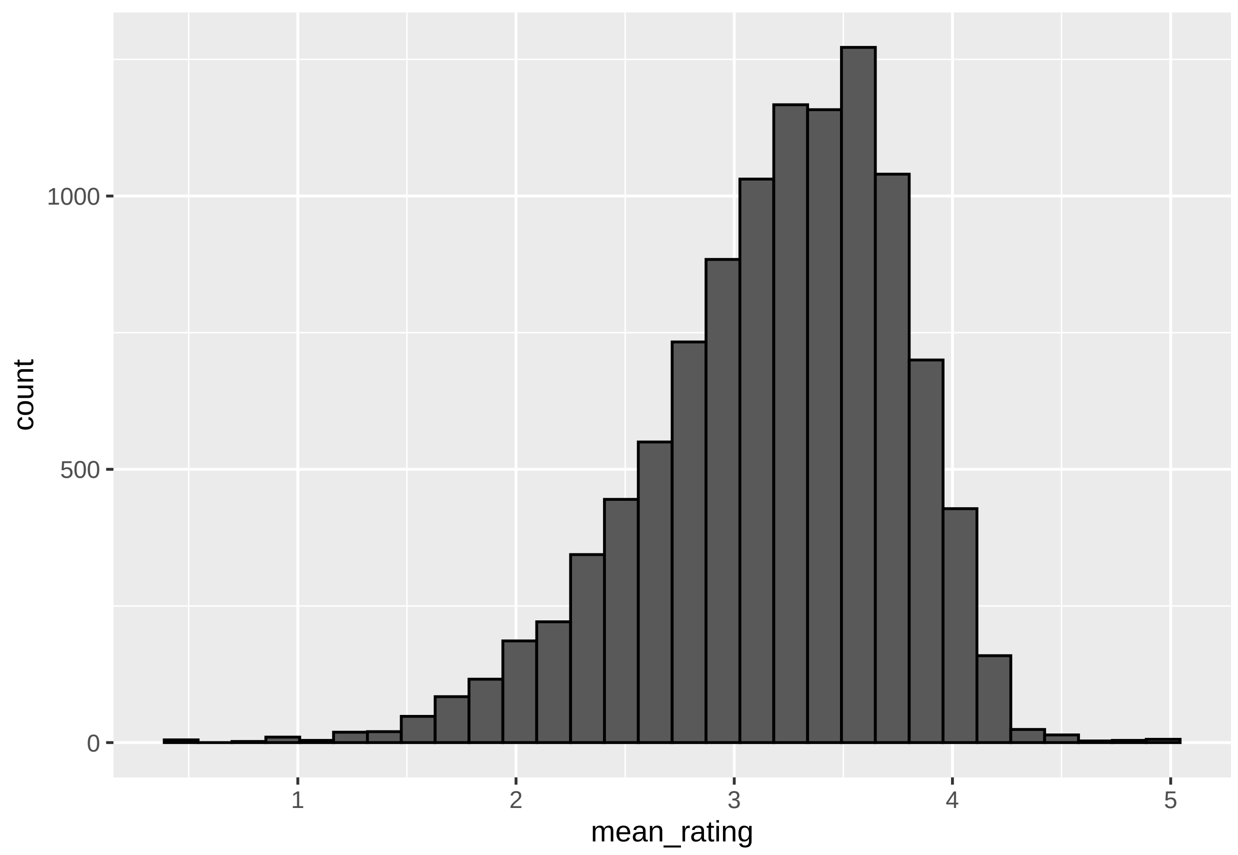
```
# Plotting median of ratings: we see that most median values lie between 3 and 4


edx %>%
  group_by(userId) %>%
  summarize(median_ratings = median(rating)) %>%
  ggplot(aes(x = median_ratings)) +
  geom_histogram(color = "black")
```



From the above graphs, we notice that mean of ratings gives us more information (since it displays greater variation) when plotted against userId. This can also be seen in the graphs for other variables.

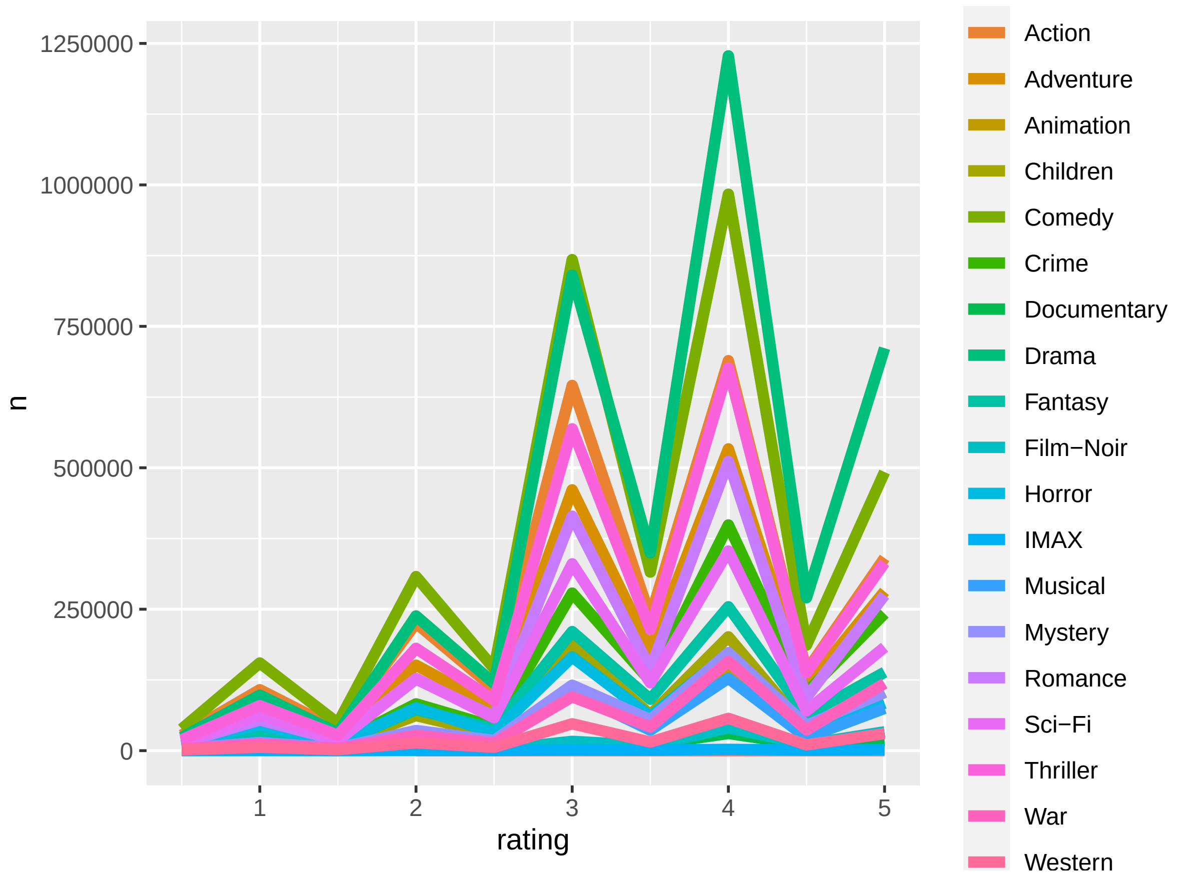
2.5.2 MovieId

```
# Mean for movies
edx %>%
  group_by(movieId) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(x = mean_rating)) +
  geom_histogram(color = "black")
```



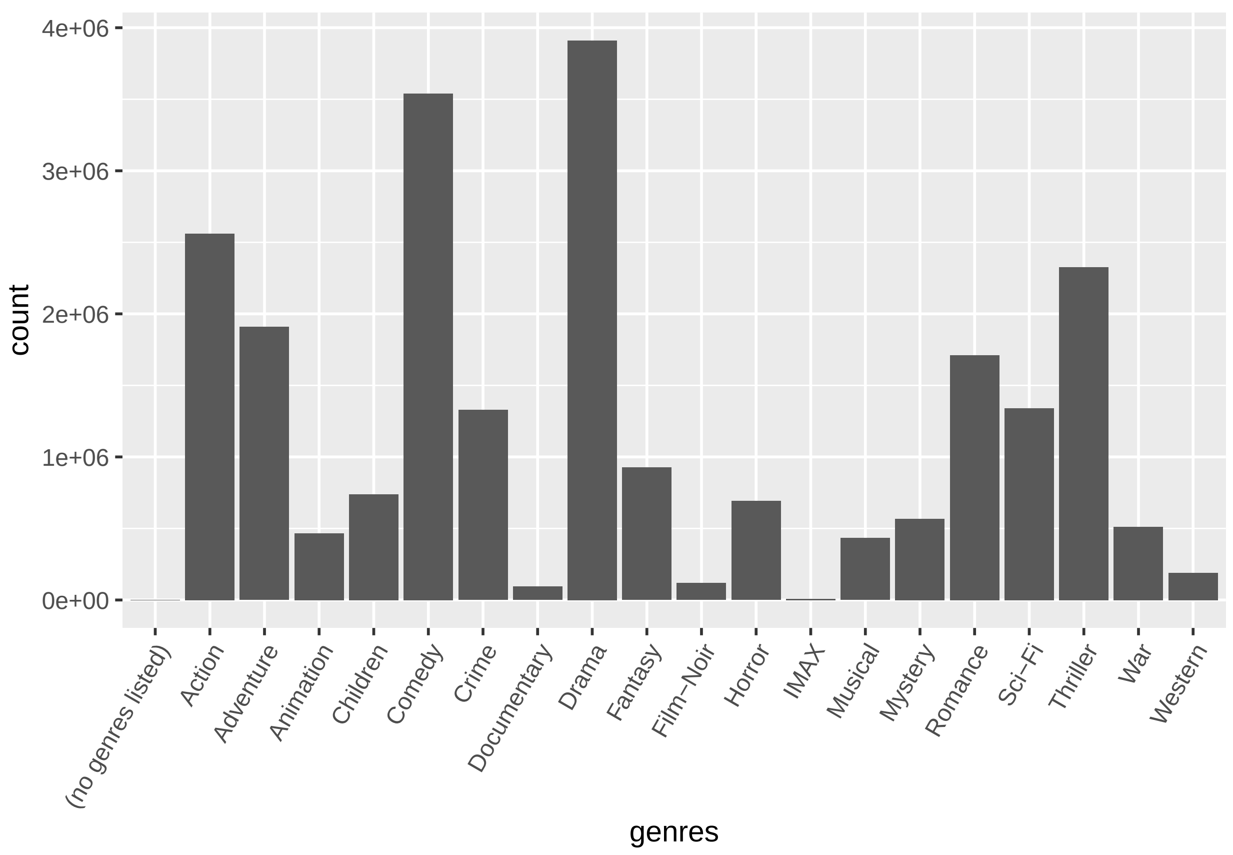
2.5.3 Genres

```
# Plot count of no of ratings for genres.
# We see that it follows the general trend, spikes for integer rating,
# lower count for decimal ratings
edx1 %>%
  group_by(genres, rating) %>%
  summarize(n = n()) %>%
  ggplot(aes(x = rating, y = n, color = genres)) +
  geom_line(size = 2)
```



Plotting the count of no of ratings for genres, we see that it conforms to the earlier stated trend of integer ratings accounting for a greater proportion. We also plot count of ratings for each genre, to see the proportion of different genres in our dataset.

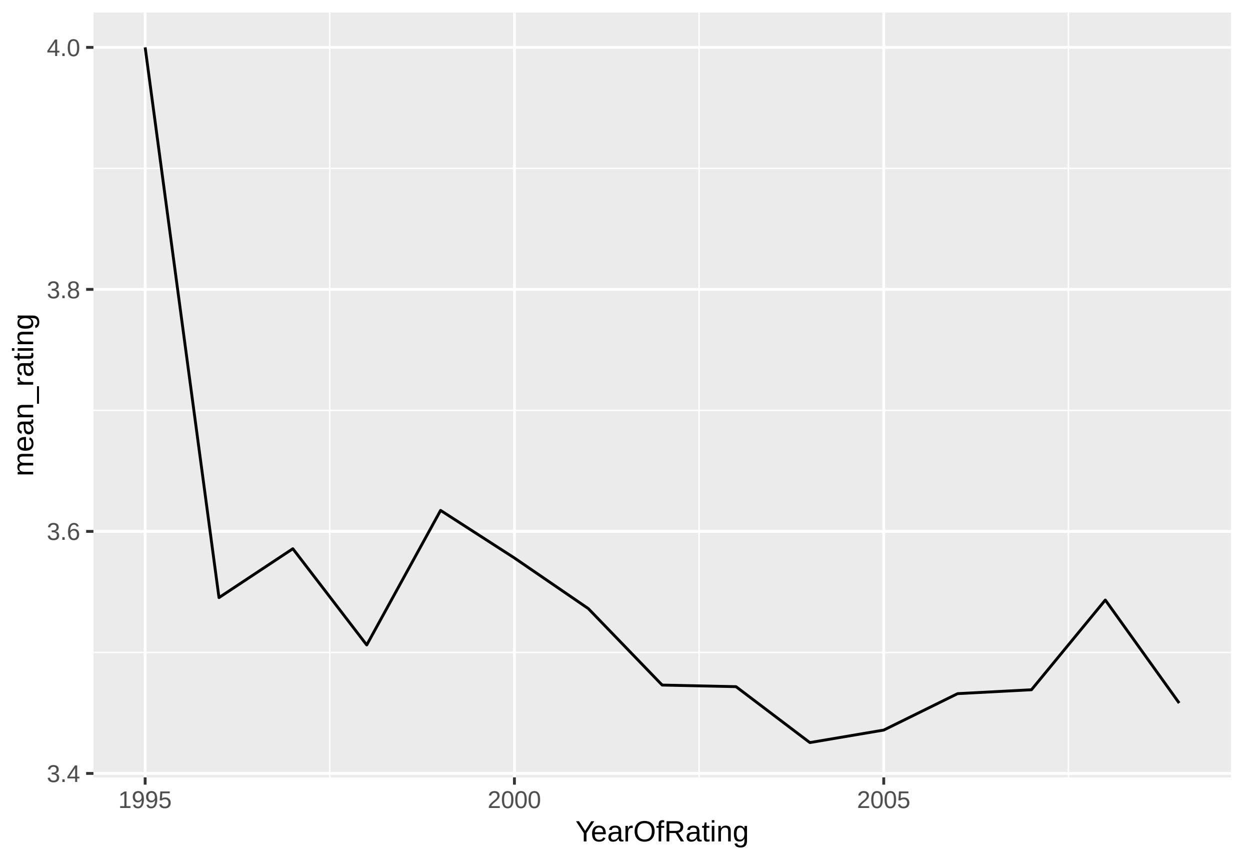
```
## Also plotting ratings received for each genre
edx1 %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = genres, y = count)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



We can see many user watched action, comedy and drama than rest of the genres. For our model, we have included all the genres, though we can remove those that account for a small number. This wont have much impact on our model.

2.5.4 YearOfRating

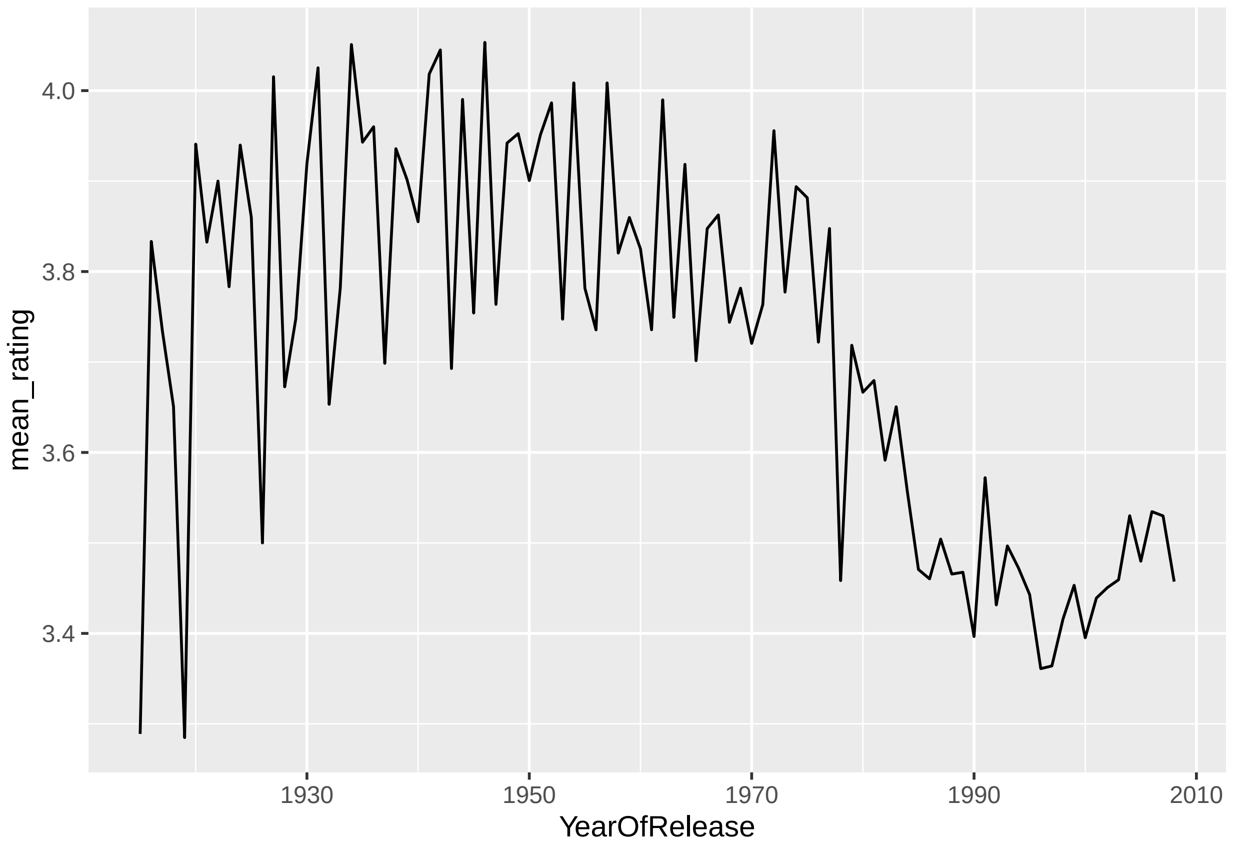
```
# Plot mean ratings
edx %>%
  group_by(YearOfRating) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(x = YearOfRating, y = mean_rating)) +
  geom_line()
```



We see that mean ratings fell as we moved ahead in time. This can thus be an important variable (effect) in our analysis.

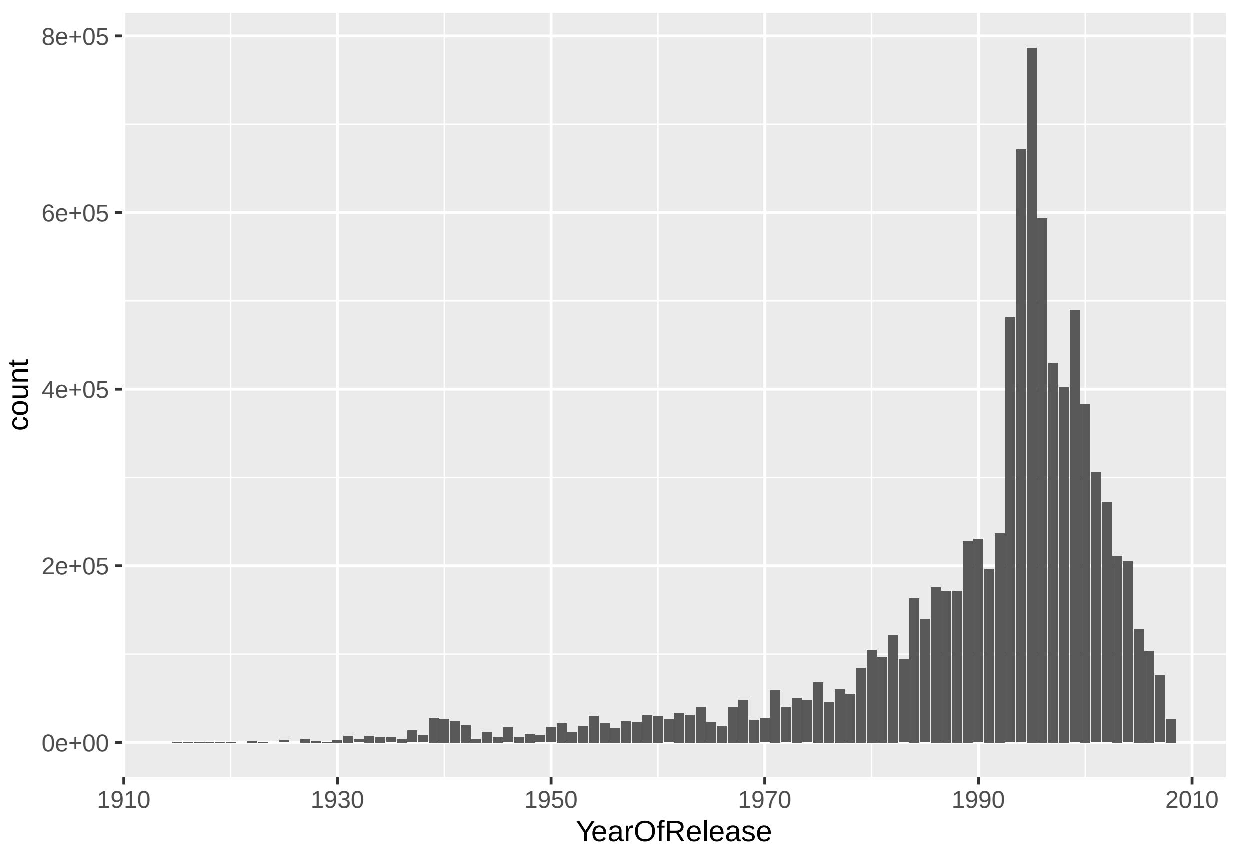
2.5.5 YearOfRelease

```
# Plot mean ratings
edx %>%
  group_by(YearOfRelease) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(x = YearOfRelease, y = mean_rating)) +
  geom_line()
```



We also plot count of movies, released in an year.

```
#Plot count of movies released in an year
edx %>%
  group_by(YearOfRelease) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = YearOfRelease, y = count)) +
  geom_bar(stat = "identity")
```



We see a good amount of variation when plotting mean_rating for years of movie release. The count of movies released also show significant difference for different years. This variable accounting for greater variation can justifiably be an important factor for our model development.

2.5.6 Unique values and observations of EDA

```
## Check unique values
length(unique(edx$movieId))
```

```
## [1] 10677
```

```
length(unique(edx$userId))
```

```
## [1] 69878
```

```
length(unique(edx$YearOfRating))
```

```
## [1] 15
```

```
length(genre)
```

```
## [1] 20
```

```r
length(unique(edx$YearOfRelease))
```

```
## [1] 94
```

Observations

The significance of a variable can be roughly determined by the variation it's graphs show. If the variation is less, then we can justifiably say that the variable in question is of less significance in the overall model, that is, the impact will be lesser.

We see that unique movieIds and userIds are a lot more in number than other variables. Looking at the variation accounted in mean ratings graphs of the two, we can say that these will have a significant impact on our model.

While that is the case for above two stated variables, the unique values present for other variables are considerably less. Though we see decent variation in the graphs of these variables, the impact of these variables will not be great due to lesser unique values. They will account for lesser variations as confirmed by the model results later.

2.6 Split Dataset for Finding Parameters

Once we are done with Data Cleaning and Preparation, we split the "edx" data further to get an intermediate dataset in 90/10 ratio. This is useful to obtain optimum values for certain parameters involved in our regularized model, Lambda value to be precise. We define our Loss function to be Root Mean Squared Error.

```r
# Split dataset
set.seed(1, sample.kind = "Rounding")
test_index_cv <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)

train_set <- edx %>% slice(-test_index_cv)
temp <- edx %>% slice(test_index_cv)

# Make sure all movies and users in test set are present in train set
test_set_cv <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set_cv)
train_set <- rbind(train_set, removed)

# Define function for RMSE used in optimizing parameters
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

2.7 Data Modeling

We are now ready to build our model. 6 models are build each one accounting for additional variation due to a variable.

We first build this model using train_set and predict ratings of test_set_cv, defined in the previous section, repeating the process for different values of lambda corresponding to a combination of variables. The lambda which gives the lowest error (loss function) in this intermediate set is selected, and the corresponding RMSE noted.

The chosen optimum value of lambda for the final model shall be used later to build model using entire "edx" set, recording the values of our loss function obtained after predicting ratings for validation set. Note that we don't use any parameter for our first model, one of the reasons we call it "Naive Model".

2.7.1 Naive Model

```r
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(test_set_cv$rating, mu_hat) %>%
  round(., digits = 5)
```

The Naive model only uses mean value to predict ratings. We see that the RMSE is very high for our first model developed on the intermediate test set named test_set_cv. We now record this RMSE with the method name in the results table as shown below. This table shall be updated as we proceed to our final model.

```r
#Create Results dataframe


results <- data.frame(method = "Only Average", RMSE = naive_rmse)

results %>%
  knitr::kable()
```

method	RMSE
Only Average	1.06

2.7.2 Regularized Movie Effect

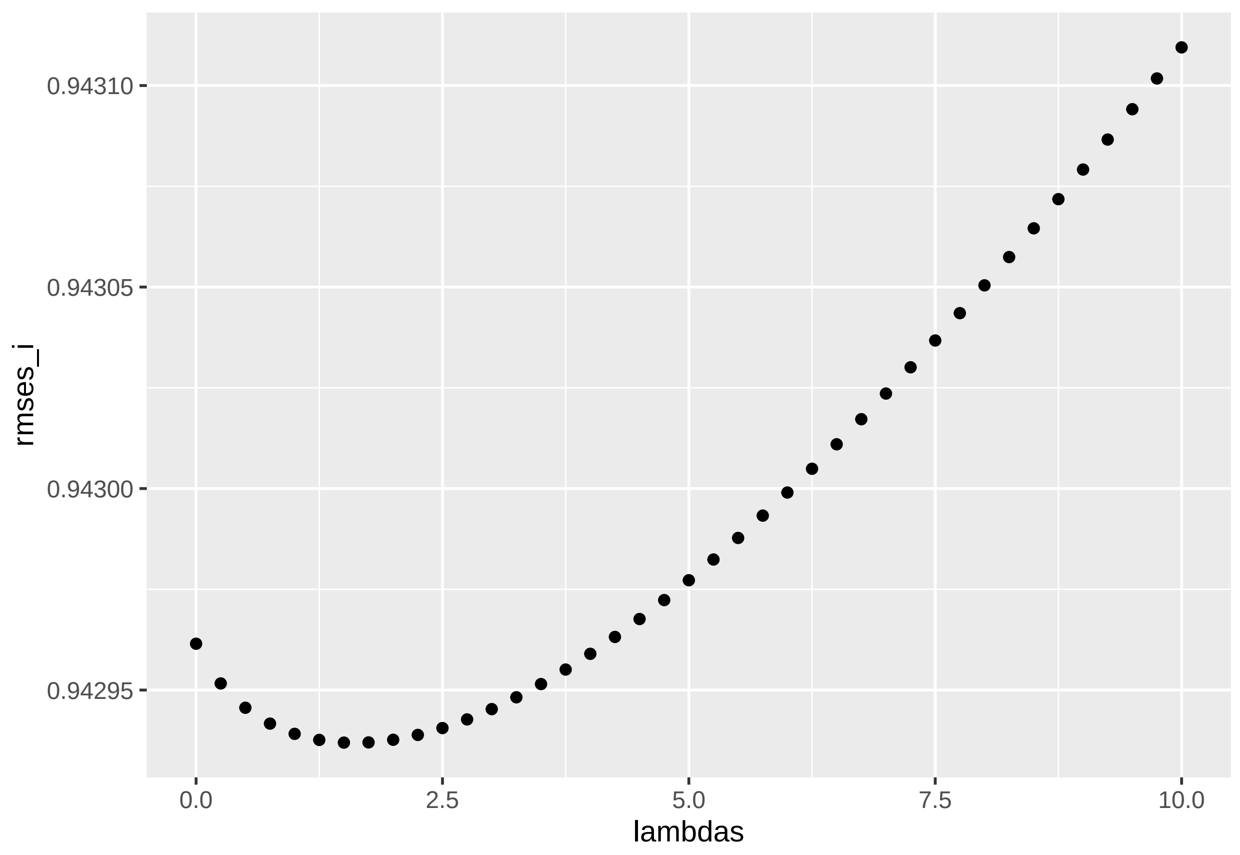
We now add Movie effect to our model.

```r
# Set lambdas
lambdas <- seq(0,10, 0.25)

# find lambda using intermediate test set: lowest rmse
rmses_i <- sapply(lambdas, function(l){

  # get b_i
  b_i <- train_set %>%
    select(movieId, rating) %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l))

  # predict
  predicted_ratings <- test_set_cv %>%
    select(movieId) %>%
    left_join(b_i, by = "movieId") %>%
    mutate(predicted = mu_hat + b_i) %>%
    pull(predicted)

  return(RMSE(test_set_cv$rating, predicted_ratings))
})

# Check lambda giving lowest rmse value
lambda_i <- lambdas[which.min(rmses_i)]
lambda_i
```

```
## [1] 1.5
```

For every value of lambda going from 0 to 10, in steps of 0.25, we find the effect labeled b_i. This is added to mu_hat for our final prediction, and RMSE is calculated. We select the lambda which gives lowest RMSE. This is confirmed with the help of the following plot

```r
#Confirm with plot
qplot(x = lambdas, y = rmses_i)
```



The RMSE corresponding to the chosen value of lambda is noted.

```r
# Update results dataframe

results <- rbind(results, data.frame(method = "Regularized Movie Effect",
                                     RMSE = min(rmses_i)))

results %>% knitr::kable()
```

method	RMSE
Only Average	1.06005
Regularized Movie Effect	0.94294

```r
## We see that RMSE has reduced.
```

From the results table, we see that RMSE is reduced significantly. Thus, greater variation is accounted for.

Similarly, we perform this operation for other effects/biases. The results table is updated for every intermediate model. The final model incorporating all variable biases is thus obtained.

2.7.3 Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect

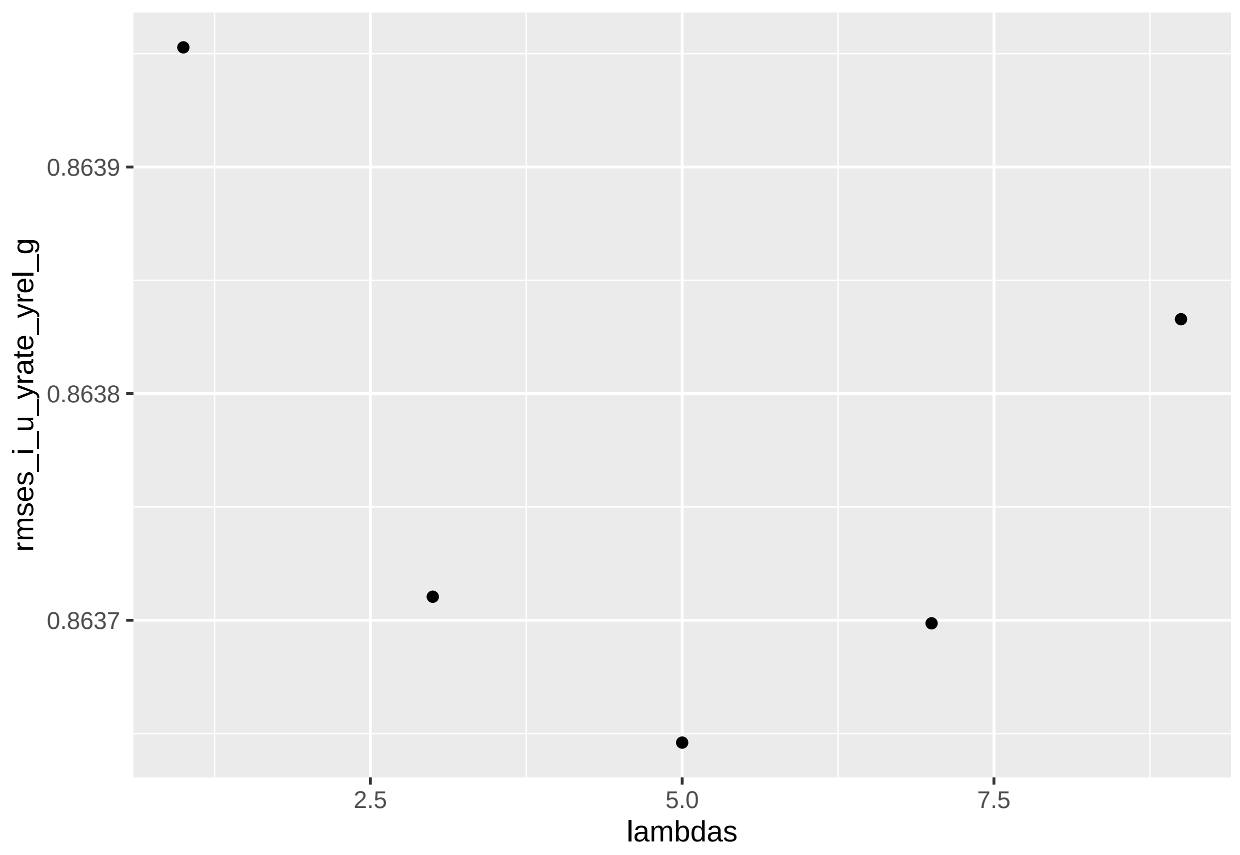
```r
# Initiate lambdas
lambdas <- seq(1,10, 2)
#we know that optimum lambda will be around 5,
# so we use bigger steps to process faster.
# Also, we check our final result with the graph to confirm

# find lambda using intermediate test set: lowest rmse

rmses_i_u_yrate_yrel_g <- sapply(lambdas, function(l){

  b_g <- list()
  b_g_x <- vector(mode = "character")


  #get b_i
  b_i <- train_set %>%
    select(movieId, rating) %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu_hat)/(n() + l)) %>%
    round(., digits = 5)

  #get b_u
  b_u <- train_set %>%
    select(movieId, userId, rating) %>%
    left_join(b_i, by = "movieId") %>%
    select(-movieId) %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu_hat - b_i)/(n() + l)) %>%
    round(., digits = 5)

  #get b_yr
  b_yrate <- train_set %>%
    select(movieId, userId, rating, YearOfRating) %>%
    left_join(b_i, by = "movieId") %>%
    select(-movieId) %>%
    left_join(b_u, by = "userId") %>%
    select(-userId) %>%
    group_by(YearOfRating) %>%
    summarise(b_yrate = sum(rating - mu_hat - b_i - b_u)/(n() + l)) %>%
    round(., digits = 5)

  b_yrel <- train_set %>%
    select(movieId, userId, rating, YearOfRating, YearOfRelease) %>%
    left_join(b_i, by = "movieId") %>%
    select(-movieId) %>%
    left_join(b_u, by = "userId") %>%
    select(-userId) %>%
    left_join(b_yrate, by = "YearOfRating") %>%
    select(-YearOfRating) %>%
    group_by(YearOfRelease) %>%
    summarize(b_yrel = sum(rating - mu_hat - b_i - b_u - b_yrate)/(n() + l)) %>%
    round(., digits = 5)

  for (i in 1:length(genre)){
  b_g_x[i] <- paste("b_g", i, sep = "_")
  df_temp <- train_set %>%
    select(movieId, userId, YearOfRating, YearOfRelease, rating, genre[i]) %>%
    left_join(b_i, by = "movieId") %>%
    select(-movieId) %>%
    left_join(b_u, by = "userId") %>%
    select(-userId) %>%
    left_join(b_yrate, by = "YearOfRating") %>%
    select(-YearOfRating) %>%
    left_join(b_yrel, by = "YearOfRelease") %>%
    select(-YearOfRelease) %>%
    mutate(b_g_effect = 0)

  k <- i

  while(k != 1){
    df_temp <- df_temp %>%
      cbind(., select(train_set, genre[k-1])) %>%
      left_join(b_g[[k-1]], by = genre[k-1]) %>%
      select(-genre[k-1]) %>%
      mutate_at(vars(b_g_x[k-1]), ~replace(., is.na(.), 0)) %>%
      mutate(b_g_effect = b_g_effect + (!!as.name(b_g_x[k-1]))) %>%
      select(-b_g_x[k-1]) %>%
      round(., digits = 5)
    k <- k-1
  }

  if(i != 1){
  b_g[[i]] <- df_temp %>%
    group_by_(genre[i]) %>%
    summarise(!!b_g_x[i] := sum(rating - mu_hat - b_i - b_u -
                                  b_yrate - b_yrel - b_g_effect)/(n() + 1)) %>%
    filter((!!as.name(genre[i])) == 1) %>%
    round(., digits = 5)
  } else {
    b_g[[i]] <- df_temp %>%
      group_by_(genre[i]) %>%
      summarise(!!b_g_x[i] := sum(rating - mu_hat - b_i - b_u
                                  - b_yrate - b_yrel)/(n()+ 1)) %>%
      filter((!!as.name(genre[i])) == 1) %>%
      round(., digits = 5)
  }
  }

  rm(df_temp)

  df_temp_test <- test_set_cv %>%
    select(movieId, userId, YearOfRating, YearOfRelease, rating) %>%
    left_join(b_i, by = "movieId") %>%
    select(-movieId) %>%
    left_join(b_u, by = "userId") %>%
    select(-userId) %>%
    left_join(b_yrate, by = "YearOfRating") %>%
    select(-YearOfRating) %>%
```

```r
    left_join(b_yrel, by = "YearOfRelease") %>%
    select(-YearOfRelease) %>%
    mutate(b_g = 0)

  for(i in 1: length(genre)){
    df_temp_test <- df_temp_test %>%
      cbind(., select(test_set_cv, genre[i])) %>%
      left_join(b_g[[i]], by = genre[i]) %>%
      select(-genre[i]) %>%
      mutate_at(vars(b_g_x[i]), ~replace(., is.na(.), 0)) %>%
      mutate(b_g = b_g + (!!as.name(b_g_x[i]))) %>%
      round(.,digits = 5) %>%
      select(-b_g_x[i])
  }

  predicted_ratings <- df_temp_test %>%
    mutate(pred = mu_hat + b_i + b_u + b_yrate + b_yrel + b_g) %>%
    round(.,digits = 5) %>%
    pull(pred)




return(RMSE(test_set_cv$rating, predicted_ratings))
})


#check lambda which gives lowest rmse
lambda_i_u_yrate_yrel_g <- lambdas[which.min(rmses_i_u_yrate_yrel_g)]
lambda_i_u_yrate_yrel_g
```

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

Note, that for our final model, we use bigger steps for lambda, taking only odd numbers from 1 to 10. Repeating the exercise of adding effects, we see the value for lambda giving lowest RMSE converges towards 5 after a few models. We thus lower the values of lambda to test, for better computational speed. The best value of lambda is further confirmed with the following plot, showing the chosen value to be justifiably suitable.

```r
#Confirm with plot
qplot(x = lambdas, y = rmses_i_u_yrate_yrel_g)
```



```
#Update results


results <- results %>%
  rbind(data.frame(method =
                     "Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect",
                   RMSE = min(rmses_i_u_yrate_yrel_g)))

results %>%
  knitr::kable()
```

method	RMSE
Only Average	1.06005
Regularized Movie Effect	0.94294
Regularized Movie+User Effect	0.86414
Regularized Movie+User+YearOfRating Effect	0.86413
Regularized Movie+User+YearOfRating+YearOfRelease Effect	0.86378
Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect	0.86365

From the above table, it is confirmed that the RMSE value for the 6th entry, accounting for effects due to variables viz. movieId, userId, YearOfRating, YearOfRelease and different genres, is the least. We thus utilize the lambda found and train our final model.

The "edx" set is used for training and results are obtained on the "validation" set.

```r
#Predict for validation set
rm(rmses_i_u_yrate_yrel_g, train_set, test_set_cv)

mu <- mean(edx$rating)

b_i <- edx %>%
  select(movieId, rating) %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/
            (n() + lambda_i_u_yrate_yrel_g)) %>%
  round(., digits = 5)

b_u <- edx %>%
  select(movieId, userId, rating) %>%
  left_join(b_i, by = "movieId") %>%
  select(-movieId) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/
            (n() + lambda_i_u_yrate_yrel_g)) %>%
  round(., digits = 5)

b_yrate <- edx %>%
  select(movieId, userId, YearOfRating, rating) %>%
  left_join(b_i, by = "movieId") %>%
  select(-movieId) %>%
  left_join(b_u, by = "userId") %>%
  select(-userId) %>%
  group_by(YearOfRating) %>%
  summarize(b_yrate = sum(rating - mu - b_i - b_u)/
            (n() + lambda_i_u_yrate_yrel_g)) %>%
  round(., digits = 5)

b_yrel <- edx %>%
  select(movieId, userId, YearOfRating, YearOfRelease, rating) %>%
  left_join(b_i, by = "movieId") %>%
  select(-movieId) %>%
  left_join(b_u, by = "userId") %>%
  select(-userId) %>%
  left_join(b_yrate, by = "YearOfRating") %>%
  select(-YearOfRating) %>%
  group_by(YearOfRelease) %>%
  summarize(b_yrel = sum(rating - mu - b_i - b_u - b_yrate)/
            (n() + lambda_i_u_yrate_yrel_g)) %>%
  round(., digits = 5)

b_g <- list()
b_g_x <- vector(mode = "character")


for (i in 1:length(genre)){
  b_g_x[i] <- paste("b_g", i, sep = "_")

  df_temp <- edx %>%
    select(movieId, userId, YearOfRating, YearOfRelease, rating, genre[i]) %>%
    left_join(b_i, by = "movieId") %>%
    select(-movieId) %>%
    left_join(b_u, by = "userId") %>%
    select(-userId) %>%
    left_join(b_yrate, by = "YearOfRating") %>%
    select(-YearOfRating) %>%
    left_join(b_yrel, by = "YearOfRelease") %>%
    select(-YearOfRelease) %>%
    mutate(b_g_effect = 0)

  k <- i

  while(k != 1){
    df_temp <- df_temp %>%
      cbind(., select(edx, genre[k-1])) %>%
      left_join(b_g[[k-1]], by = genre[k-1]) %>%
      select(-genre[k-1]) %>%
      mutate_at(vars(b_g_x[k-1]), ~replace(., is.na(.), 0)) %>%
      mutate(b_g_effect = b_g_effect + (!!as.name(b_g_x[k-1]))) %>%
      select(-b_g_x[k-1]) %>%
      round(., digits = 5)
    k <- k-1
  }

  if(i != 1){
  b_g[[i]] <- df_temp %>%
    group_by_(genre[i]) %>%
    summarise(!!b_g_x[i] :=
                sum(rating - mu - b_i - b_u -
                      b_yrate - b_yrel - b_g_effect)/
                (n() + lambda_i_u_yrate_yrel_g)) %>%
    filter((!!as.name(genre[i])) == 1) %>%
    round(., digits = 5)
  } else {
    b_g[[i]] <- df_temp %>%
      group_by_(genre[i]) %>%
      summarise(!!b_g_x[i] :=
                  sum(rating - mu - b_i - b_u - b_yrate - b_yrel)/
                  (n()+ lambda_i_u_yrate_yrel_g)) %>%
      filter((!!as.name(genre[i])) == 1) %>%
      round(., digits = 5)
  }
  }



 df_temp_val <- validation %>%
   select(movieId, userId, YearOfRating, YearOfRelease, rating) %>%
   left_join(b_i, by = "movieId") %>%
   select(-movieId) %>%
   left_join(b_u, by = "userId") %>%
   select(-userId) %>%
   left_join(b_yrate, by = "YearOfRating") %>%
   select(-YearOfRating) %>%
   left_join(b_yrel, by = "YearOfRelease") %>%
   select(-YearOfRelease) %>%
   mutate(b_g = 0)

  for(i in 1: length(genre)){
    df_temp_val <- df_temp_val %>%
      cbind(., select(validation, genre[i])) %>%
      left_join(b_g[[i]], by = genre[i]) %>%
      select(-genre[i]) %>%
      mutate_at(vars(b_g_x[i]), ~replace(., is.na(.), 0)) %>%
      mutate(b_g = b_g + (!!as.name(b_g_x[i]))) %>%
      round(.,digits  = 5) %>%
      select(-b_g_x[i])
  }

  movie_user_YearOfRating_YearOfRelease_genre_effect <- df_temp_val %>%
    mutate(pred = mu + b_i + b_u + b_yrate + b_yrel + b_g) %>%
    pull(pred)

results_final <- data.frame(method =
                     "Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect",
                   RMSE = RMSE(validation$rating,
                               movie_user_YearOfRating_YearOfRelease_genre_effect))
```

```
## the final RMSE is as follows
results_final %>%
  knitr::kable()
```

method	RMSE
Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect	0.86434

We thus obtain a desired RMSE value (below 0.86490).

3 Result

Observing the Loss Function values in the “results” table, we see that the best model is when all the variables, that is, movieId, userId, year of movie rating, year of movie release and genre are included.

```r
#Results obtained through intermediate sets
results %>% knitr::kable()
```

method	RMSE
Only Average	1.06005
Regularized Movie Effect	0.94294
Regularized Movie+User Effect	0.86414
Regularized Movie+User+YearOfRating Effect	0.86413
Regularized Movie+User+YearOfRating+YearOfRelease Effect	0.86378
Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect	0.86365

For comparison purpose, we check the RMSE that the first model gives using “edx” and “validation” set.

```r
#Results obtained using "edx" and "validation" sets

mu <- mean(edx$rating)

results_final <- results_final %>%
  rbind(data.frame(method = "Only average",
                   RMSE = RMSE(validation$rating, mu)))

results_final %>%
  arrange(desc(RMSE)) %>%
  knitr::kable()
```

method	RMSE
Only average	1.06120
Regularized Movie+User+YearOfRating+YearOfRelease+Genre Effect	0.86434

A substantial reduction in RMSE, from 1.06120 to 0.86434 , when using “edx” and “validation” sets is observed (results_final table). The desired RMSE value is also obtained.

Looking at the “results” table, we observe that the reduction is more when effects for variables movieId, userId and year of movie release are accounted for. This tells us that these account for the maximum variation among all other variables we employed.

We have thus found the optimal value for parameter lambda as 5 and successfully built a decent recommendation system using a regularized model.

4 Conclusion

The regularized model utilized shows great promise, with RMSE 0.86434 (utilizing effects due to all variables), well below the 0.86490 desired.

The significant reduction from Naive Model obtained is nevertheless subject to few factors that are observed in the analysis

In particular, the decrease in RMSE is different for different variables. Major reduction can only be seen if the said variable account for greater variation. This can be found to be dependent on unique values present for a variable in our regularized model. MovieId, userId and year of movie release have the most unique values present, and thus lead to the greatest reduction in loss function compared to other variables, as seen in the "results" table, modeled using intermediate "train" and "test" sets (and can be verified to follow the same pattern when using "edx" and "validation" sets to model)

This observation brings us to an important limitation of our model. While we used all the variables available to us for getting final RMSE, the computation time and memory usage has been ignored for the same. An efficient model taking into account these elements will be more beneficial in practical use.

The future models can thus be built with greater focus to variable importance. Some Machine Learning techniques like Random Forest are known to generate Feature Importance and similar techniques can thus be employed, provided the RMSE is close to, if not greater than, that obtained in our regularized model.

The given recommendation systems utilizing all information of the dataset is therefore an able model, providing further scope and insights to build on for practical application.