Movie Recommendation System HarvardX

Raheem Bukhsh

2020-10-16

Contents

1 Introduction:

This is the capstone as well as the 9th and last course for the "Data Science Professional Certificate" offered by HarvardX through Edx. This project is basically a test of all the knowledge gained through previous 8 courses. In this project we are required to build a movie recommendation system with different machine learning algorithms and then to calculate RMSE for each of these models to find the best model with minimum RMSE. The train set named **"edx"** will be used to train the model and test set **"validation"** will be used to test the trained model for RMSE.
The output for the project is required in the form of 3 documents:
1. a report in the form of an Rmd file
2. a report in the form of a PDF document knit from relevant Rmd file
3. an R script that generates predicted movie ratings and calculates RMSE

1.1 Project Overview:

The specific movie recommendation system for this project is motivated from the competition held by Netflix, an online DVD-rental and video streaming service provider.
The Netflix Prize was an open competition for the best machine algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users or the films being identified except by numbers assigned for the contest.

1.2 Recommendation System:

Movie recommendation system or a general recommendation system is a widely used machine learning phenomenon that is meant to predict the likelihood of rating a particular user would give to a particuar product. Recommendation systems are commonly incorporated by companies providing online music & video

services, online shopping stores, social media platforms and many more. Preferences for recommendation systems cover a lot of products like movies, music videos, physical products, real estate, restaurants and news articles etc.
There are many types of recommendation system but three are most commonly used.

- Collaborative filtering
- Content based filtering
- Hybrid recommendation system

1.2.1 Collaborative filtering:

Collaborative filtering methods are based on collecting and analyzing a large amount of information on users' behaviors, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself. Many algorithms have been used in measuring user similarity or item similarity in recommender systems. For example, the k-nearest neighbor (k-NN) approach and the Pearson Correlation.

1.2.2 Content based filtering:

Content-based filtering methods are based on a description of the item and a profile of the user's preference. In a content-based recommendation system, keywords are used to describe the items; beside, a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

1.2.3 Hybrid recommendation system:

Recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective in some cases. Hybrid approaches can be implemented in several ways, by making content-based and collaborative-based predictions separately and then combining them, by adding content-based capabilities to a collaborative-based approach (and vice versa), or by unifying the approaches into one model. Several studies empirically compare the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches. These methods can also be used to overcome some of the common problems in recommendation systems such as cold start and the sparsity problem.

Netflix is a good example of a hybrid system. They make recommendations by comparing the watching and searching habits of similar users (i.e. collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering).

1.3 MovieLens data set:

Since we do not have access to the original Netflix data so MovieLens data set by MovieLens Group will be used for this movie recommendation system.MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.
The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period

from September 19th,1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this data set.
We will use the 10M version of the MovieLens data set to make the computation a little easier.

2 Data analysis:

2.1 Data Preperation:

The 10M version of MovieLens data set is used to make the computations a little easier.
Now we have to download and prepare the data set for exploratory analysis. Data will be partioned in to edx(Train) set and validation(Test) set. Due to some problem in retrieving data from the given code on edx website the following link was provided by the Edx HarvardX staff to download the edx and validation data sets:

`https://drive.google.com/drive/folders/1IZcBBX0OmL9wu9AdzMBFUG8GoPbGQ38D?usp=sharing`

The two mentioned data sets in the google drive are manually downloaded and read using following commands:

```
edx <- readRDS("C:\\Users\\Raheem\\Downloads\\edx.rds")
validation <- readRDS("C:\\Users\\Raheem\\Downloads\\validation.rds")
```

Upload the following packages and liabraries:

```
library(tidyverse)
library(tidyr)
library(dplyr)
library(lubridate)
```

2.2 Exploratory data analysis:

Exploring data is the best way to get familiar with the data set and to prepare in the best way possible to devise a machine learning algorithm for movie recommendation system.

- **Dimensions of `edx` (training) data set:**

```
dim(edx)
```

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

Edx data set has 9000055 observations and 6 variables.

- **Structure of `edx` (training) data set:**

```
str(edx)
```

```
## '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
```

Edx is a data frame with 6 variables namely,

1 | userId | Integer
2 | movieId | Number
3 | rating | Number
4 | timestamp| Integer
5 | title | Character
6 | genres | Character

- **Structure of `validation` (test) data set:**
  As the machine learning algorithm trained on edx will be tested on the validation set. Therefore it is necessary to test the structure of `validation` (test) data set too for uniformity of the results.

```
str(validation)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)"
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
```

validation has the same structure as edx therefore a viable test data set with 6 variables namely,

1 | userId | Integer
2 | movieId | Number
3 | rating | Number
4 | timestamp| Integer
5 | title | Character
6 | genres | Character

edx (training) data set now will only be explored for data analysis and to be used to develope algorithm.

- **Summary of edx:**

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
```

```
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The edx data set has data in tidy format with each row having a single observation for 6 different variables. The user who rates movie has details in `userId` column and movie to be rated has relevant information in `movieId` and `title` columns. The `rating` column is the outcome sought by this whole project . The date at which a specific movie is rated by a user is provided in `timestamp` column which has data measured in seconds. Each movie falls in one or more categories described in `genres` column. * All exploratory data analysis will be relative to variables according to `rating` factor as `rating` is the desired outcome for this project and algorithm.

2.2.1 movieId:

- **Unique movies in Edx:**

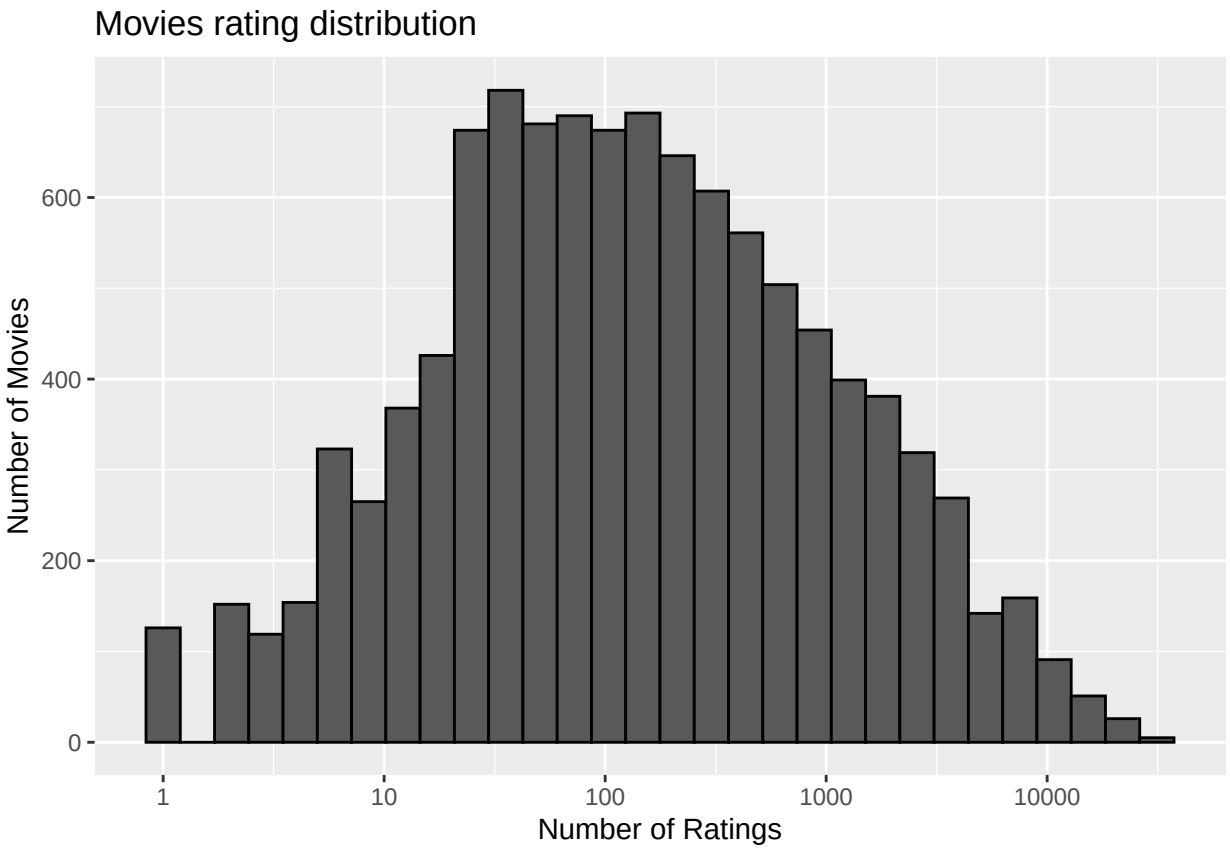
```
n_distinct(edx$movieId)
```

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

There are 10677 unique movies in `edx` data set.

- **Movies rating distribution:**

```
edx %>%
  group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies rating distribution") +
  xlab("Number of Ratings") +
  ylab("Number of Movies")
```



Some movies are rated more than the others as blockbusters are watched by the millions compared to niche movies watched by a few.

2.2.2 userId:

- **Unique users in Edx:**

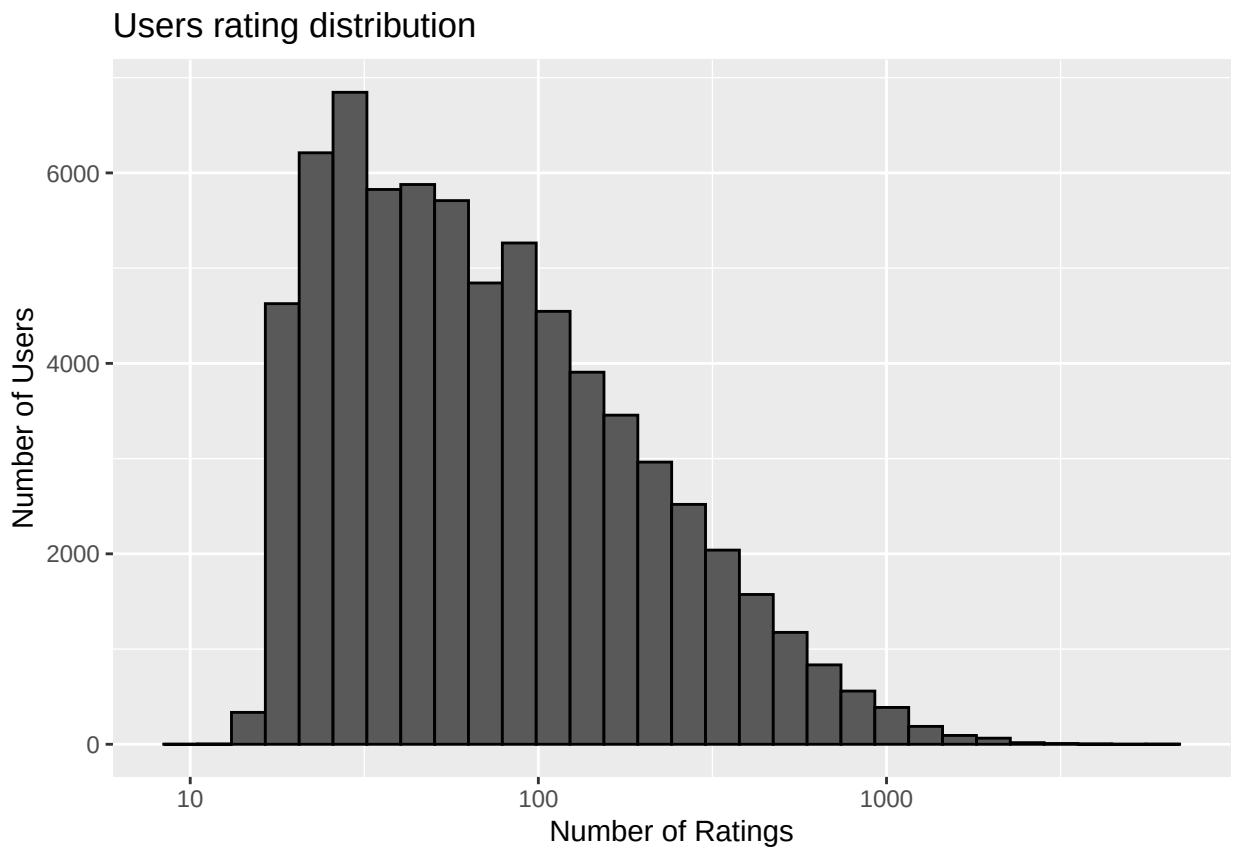
```
n_distinct(edx$userId)
```

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

There are 69878 unique users who have rated movies in `edx` data set. The number of unique movies is `10677` and unique users `69878` is not same therefore it means that not every user user has rated a movie and some unique users have rated multiple movies.

- **Users rating distribution:**

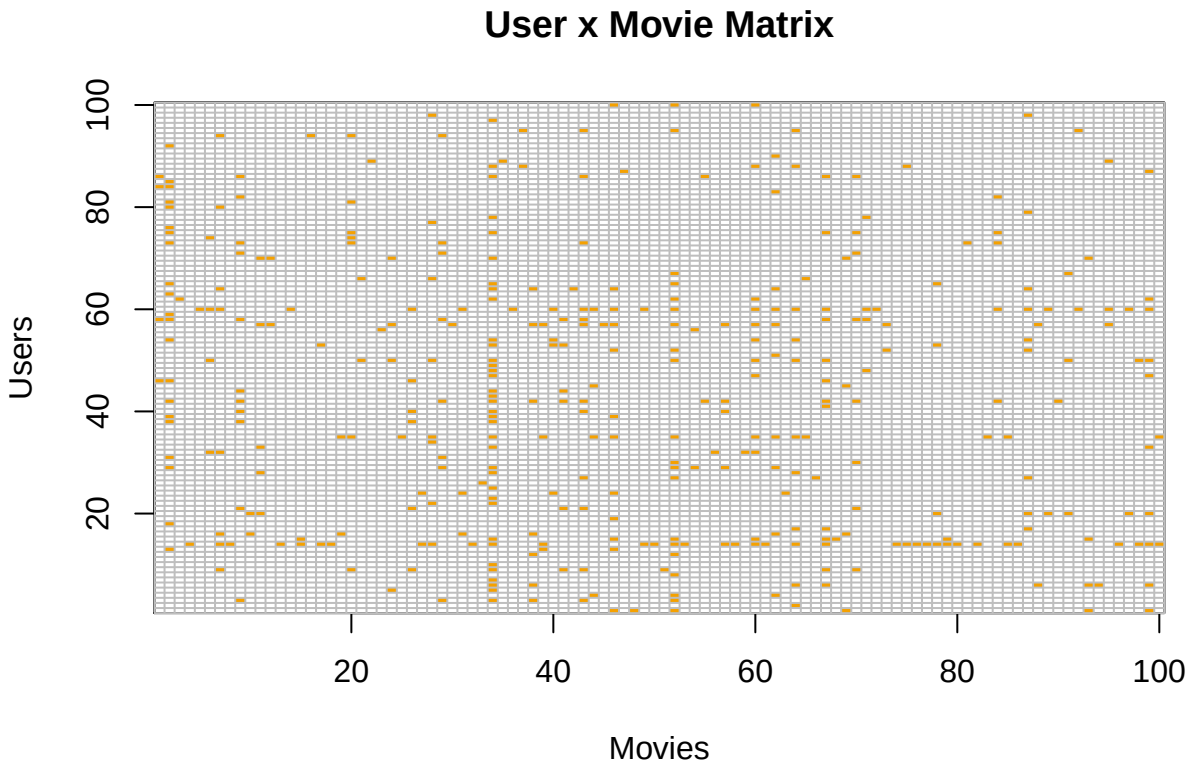
```
edx %>%
  group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users rating distribution") +
  xlab("Number of Ratings") +
  ylab("Number of Users")
```



The distribution is right skewed and it is clearly evident that some users are more active than others in rating the movies. Some users have rated over a thousand movies and some only a handful.

- **Sparcity of user X movie matrix:**
  Check the sparcity of User and Movie matrix at a sample of 100 users.

```
users <- sample(unique(edx$userId), 100)
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users") %>%
  abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
title("User x Movie Matrix")
```



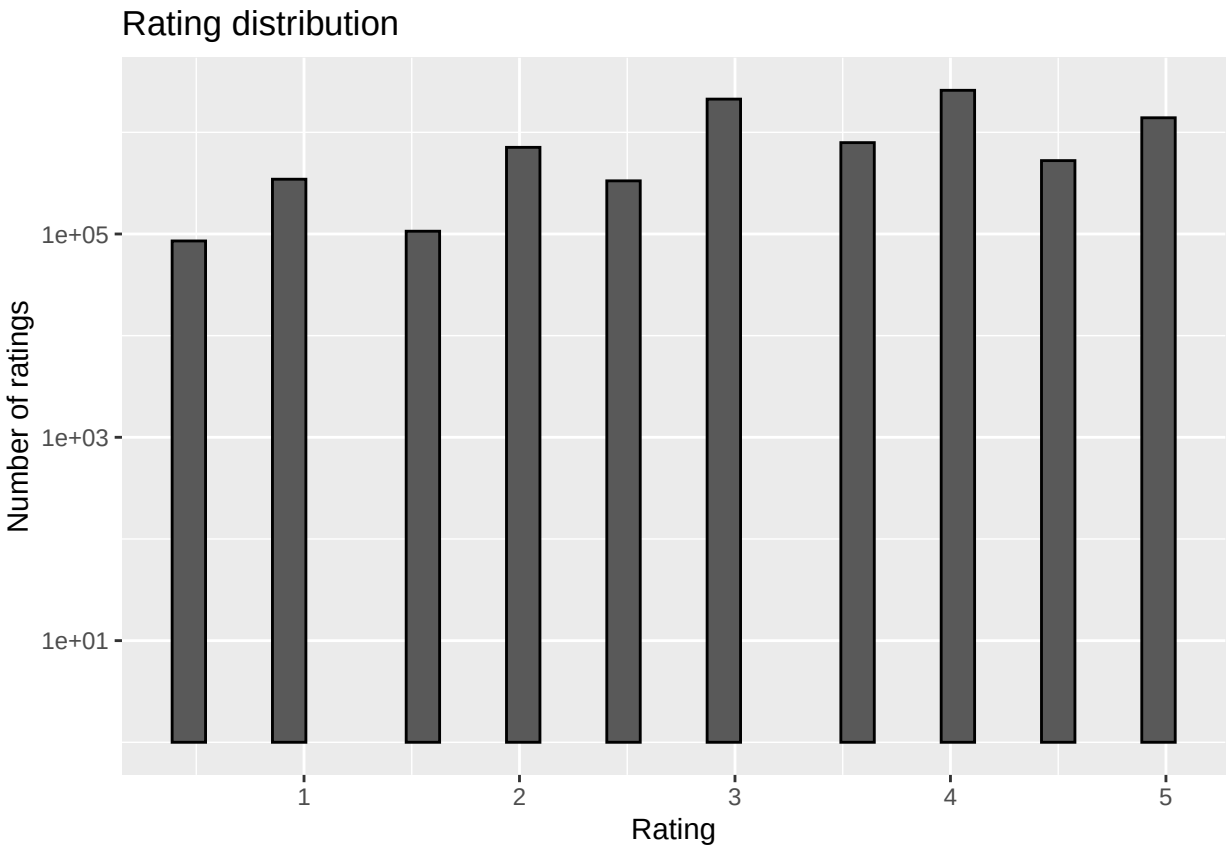
To see how sparce the entire matrix is above is the matrix of 100 movies and 100 users with yellow indicating a user movie combination for which we have a rating whereas blank spaces are combination with no rating provided by a user to a movie. clearly some movies have ratings from multiple users and some movies do not have any.

2.2.3 rating:

Users can provide the any rating choosen from 10 values ranging from 0.5 to 5.

- **Rating distribution:**

```
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(bins = 30, color = "black") +
  scale_y_log10() +
  ggtitle("Rating distribution") +
  xlab("Rating") +
  ylab("Number of ratings")
```



The rating of 4 is the most common rating followed by 3 and 5.

- **Each rating count:**

```
edx %>% group_by(rating) %>% summarize(count = n())  %>%
  arrange(desc(count))
```

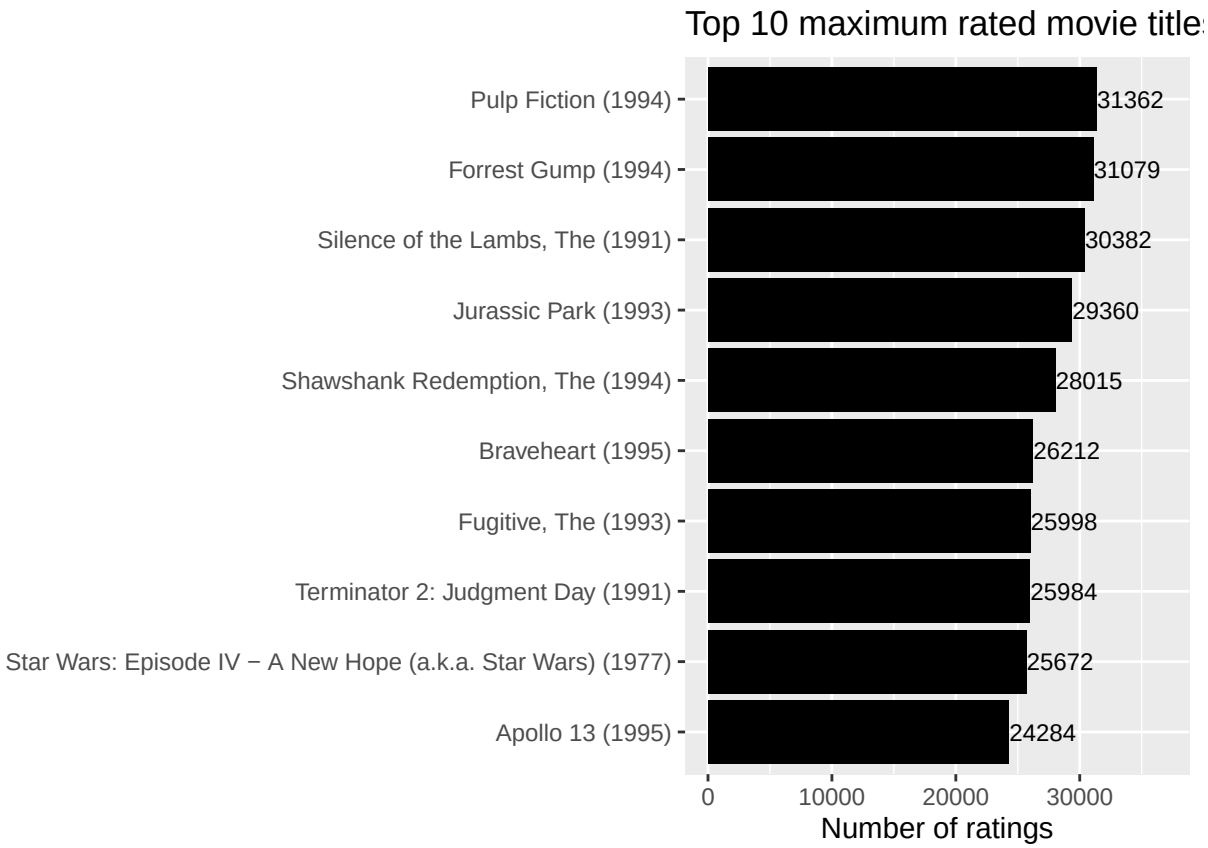
rating	count
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422
4.5	526736
1.0	345679
2.5	333010
1.5	106426
0.5	85374

Each individual rating count verifies the results of rating histogram with 4 being the most common rating for movies.

2.2.4 Title:

- **Maximum rated movie titles:**

```
max_r_titles <- edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  top_n(10, count) %>%
  arrange(desc(count))
max_r_titles %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="black") + coord_flip(y=c(0, 37000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.01, size=3) +
  labs(title="Top 10 maximum rated movie titles")
```



Pulp Fiction is the highest rated movie followed by Forrest Gump and Silence of the Lambs respectively.

2.2.5 timestamp:

- **Rating Period:**

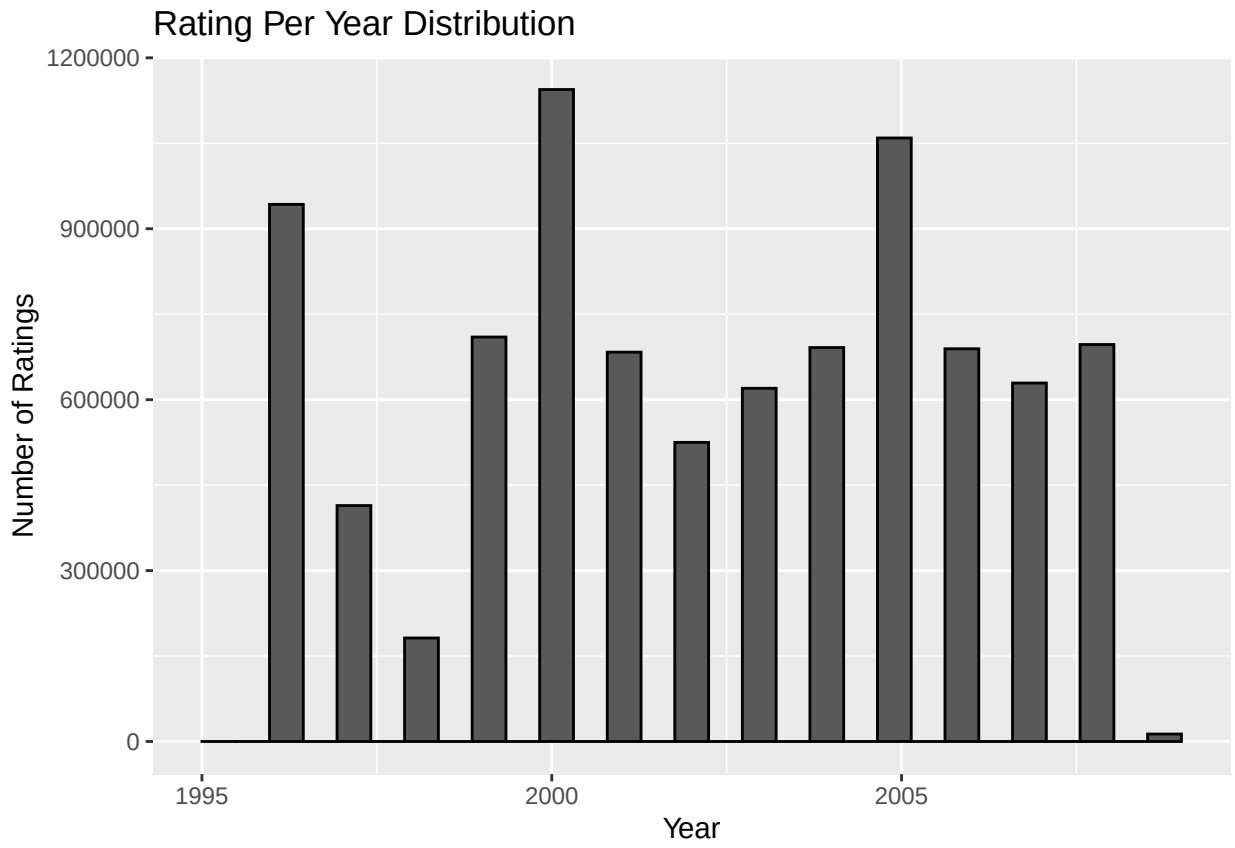
```
tibble(`Start Date` = date(as_datetime(min(edx$timestamp))),
       `End Date` = date(as_datetime(max(edx$timestamp)))) %>%
  mutate(Period = duration(max(edx$timestamp)-min(edx$timestamp)))
```

Start Date	End Date	Period
1995-01-09	2009-01-05	441479727s (~13.99 years)

The ratings were collected from a period between 1995-01-09 and 2009-01-05 for a period of 14 years approximately.

- **Rating Per Year Distribution:**

```
edx %>% mutate(year = year(as_datetime(timestamp))) %>%
  ggplot(aes(x=year)) +
  geom_histogram(bins = 30, color = "black") +
  ggtitle("Rating Per Year Distribution") +
  xlab("Year") +
  ylab("Number of Ratings")
```

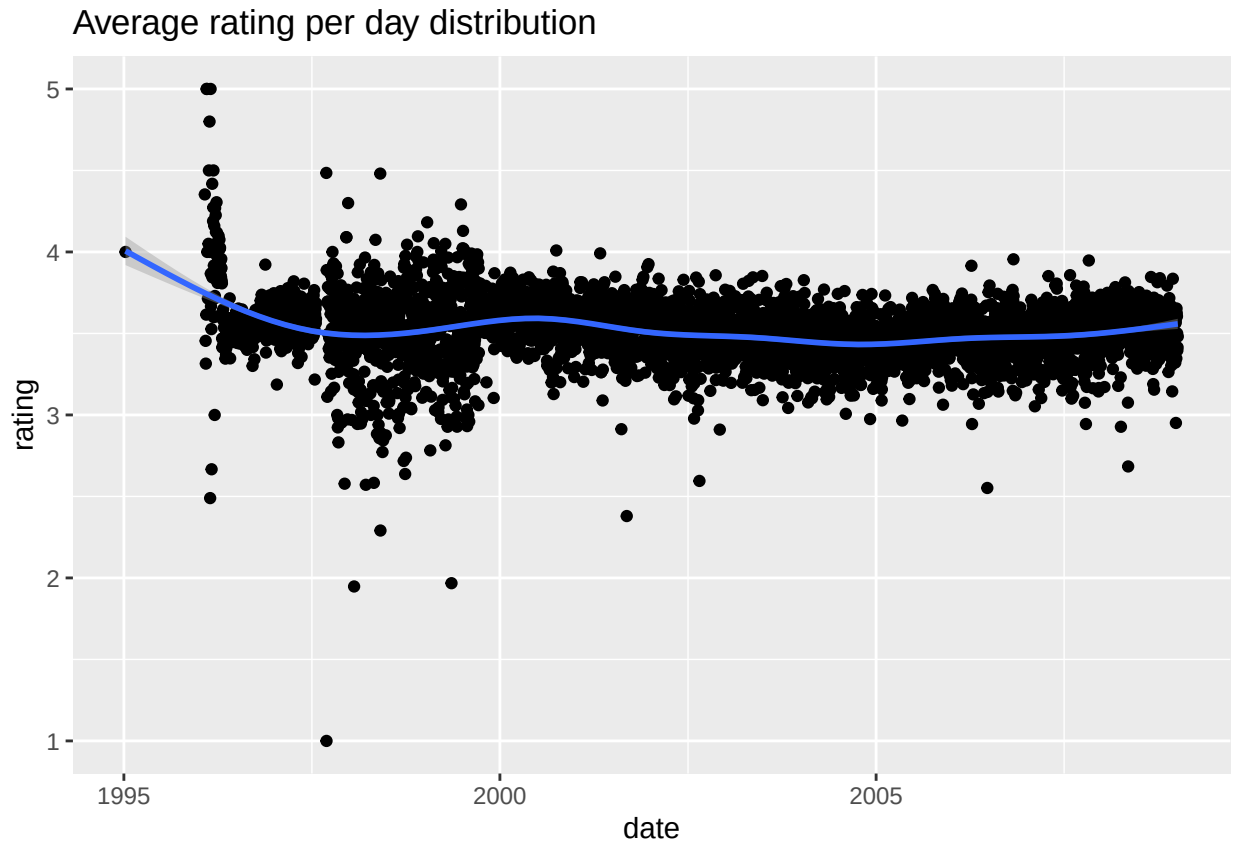


In year 2000 maximum number of people rated the movies followed by the year 2005.

- **Average rating over time distribution:**

```
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "day")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
```

```
  geom_point() +
  geom_smooth() +
  ggtitle("Average rating per day distribution")
```



The daily rating pattern of movies over the years confirms the mean rating of 3.512 as stated in summary statistics of edx data set.

2.2.6 genres:

- **Unique Genres:**

```
n_distinct(edx$genres)
```

```
## [1] 797
```

There are 797 different genres against movies these are either single category or a combination of more than one categories.

- **Highest rated genres:**

```
edx %>%
 group_by(genres) %>%
 summarize(count = n()) %>%
 top_n(10, count) %>%
 arrange(desc(count))
```

genres	count
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355
Action|Adventure|Sci-Fi	219938
Action|Adventure|Thriller	149091
Drama|Thriller	145373
Crime|Drama	137387

- **Genres of most rated movies:**

```
most_r_genres <- edx %>%
   group_by(genres, title) %>%
   summarize(count = n()) %>%
   arrange(desc(count))
head(most_r_genres)
```

genres	title	count
Comedy|Crime|Drama	Pulp Fiction (1994)	31362
Comedy|Drama|Romance|War	Forrest Gump (1994)	31079
Crime|Horror|Thriller	Silence of the Lambs, The (1991)	30382
Action|Adventure|Sci-Fi|Thriller	Jurassic Park (1993)	29360
Drama	Shawshank Redemption, The (1994)	28015
Action|Drama|War	Braveheart (1995)	26212

It clearly shows that movies are categorized in multiple genres and drama is the most rated genres among most rated movies.

3 Machine Learning methods

There are a number of methods to predict a movie rating and these methods are measured for their efficiency with various metrices.

3.1 Evaluation (Loss function):

3.1.1 Mean squared error-MSE:

The mean squared error or MSE meaures the average of the average squared difference or distance between the estimated values and the actual value. MSE is a risk function as it measures the expected value of the squared error loss. THe MSE is a measure of the quality of an estimator, it is always non-negative and vlues closer to zero are considered better.It is expressed as follows:

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

Where $y_{u,i}$ is the rating for the movie i by user u for N number of observations and sum occuring over all these combinations. Whereas $\hat{y}_{u,i}$ is the predicted rating.

3.1.2 Root mean squared error-RMSE:

The Root Mean Squared Error is the typical error loss, used to evaluate the recommendation system. It is the under root of MSE and is defined as:

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

Where $y_{u,i}$ is the rating for the movie i by user u for N number of observations and sum occuring over all these combinations. Whereas $\hat{y}_{u,i}$ is the predicted rating.RMSE can be interpreted similarly to a standard deviation as it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.
In Netflix challange the **RMSE** was used to evaluate the machine learning models to decide the winner and we will also use **RMSE** to evaluate models.

3.2 Linear Model:

3.2.1 Average model:

A simplest possible recommendation system predicts the same rating for all movies regardless of user. It assumes that all users will give the same rating to all movies and movie to movi variation is random error. It is the average of all the observed ratings. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like:

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

where $\epsilon_{u,i}$ is independent error samples from the sample distribution centered at 0 and μ the "true" rating for all movies. The estimate that minimizes the RMSE is the least squares estimate of
μ and, in this case, is the average of all ratings.

3.2.2 Movie effect:

some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. We can augment our previous model by adding the term b_i to represent average ranking for movie i:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Statistics textbooks refer to the b's as effects. However, in the Netflix challenge papers, they refer to them as "bias", thus the b notation. The movie effect is the mean of difference between actual rating y and the mean $\hat{\mu}$ and calculated as:

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

3.2.3 User effect:

There is substantial variability across users as well, some users are very cranky and others love every movie. This implies that a further improvement to our model may be:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is a user specific effect. User effect is that some users are cranky thus rate good movies very low compared to the users who rate movies very high.
The user effect is computed as an approximation by computing $\hat{\mu}$ and $\hat{b}_i$ and estimatimating as:

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

3.2.4 Time effect:

We observed in exploratory data analysis that during some days movies are higher when compared to others so with date effect b_t the model will look like:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_t + \epsilon_{u,i}$$

The time effect is not part of project scope and will be ignored further. However timestamp was converted to the number of day wise units in analysis and it can be calculated as:

$$\hat{b}_t = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

3.3 Regularization:

Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes.The linear model does not account for the fact that some movies have very few ratings as low as 1 and also some users rate very few movies as low as 1. It means that the small sample size of these movies and users can lead to large estimated errors. To mitigate this effect and to improve estimate of prediction a factor is added that penalizes the small sample sizes and have little or no effect otherwise.
A penalty term is added for large values of b_i and b_u to the sum of squares equation that we minimise. A more accurate estimate of b_i and b_u will treat them symmetrically, by solving the least squares problem,

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

The first term is the mean squared error and the second is a penalty term that gets larger when many b's are large.The values of b's that minimize this equation are given by:

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

,

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

,

where n_i is a number of ratings b for movie i by user u.

The larger λ is, the more we shrink. λ is a tuning parameter, so we can use cross-validation to choose it. We should be using full cross-validation on just the training set, without using the test set until the final assessment.
When n_i is very large, it will provide with a stable estimate and λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when ni is small, then the estimate b^i( ) is shrunken towards 0 .The larger λ, the more we shrink.

3.4 Matrix factorization:

Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix $R_{m\times n}$ into the product of two lower dimensionality rectangular matrices $P_{k\times m}$ and $Q_{k\times n}$ so that:

$$R \approx P'Q$$

Our earlier models fail to account for an important source of variation related to the fact that groups of movies and groups of users have similar rating patterns. We can observe these patterns by studying the residuals and converting our data into a matrix where each user gets a row and each movie gets a column:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

,

where $y_{u,i}$ is the entry in row u and column i.

We can factorize the matrix of residuals r into a vector p and vector q, $r_{u,i} \approx p_u q_i$, allowing more of variance using a model like:

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

As our data is complicated, we can use two factors to explain the structure and two sets of coefficients to descride users:

$$Y_{u,i} = \mu + b_i + b_u + p_{u,1} q_{1,i} + p_{u,2} q_{2,i} + \epsilon_{i,j}$$

4 Machine Learning Models Application:

In this section the machine learning models for a recommendation system are coded and evaluated for results.

4.1 Loss Function:

RMSE as explained earlier is the typical loss function used to evaluate a recommendation system.The model with minimum Loss is selected as most feasible recommendation system.

```
# Define Root Mean Suare Error (RMSE)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

4.2 Linear model:

The linear model to build a model is stated as:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

4.2.1 Average model:

The basic model predicts the average rating for all the movies and is stated as:

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

```
#4.2.1  Basic Linear Model
mu <- mean(edx$rating)

# RMSE Basic Linear Model for training set
RMSE_E1 <- RMSE(edx$rating, mu)

# #RMSE Basic Linear Model for test set
RMSE_V1 <- RMSE(validation$rating, mu)
RMSE_V1
```

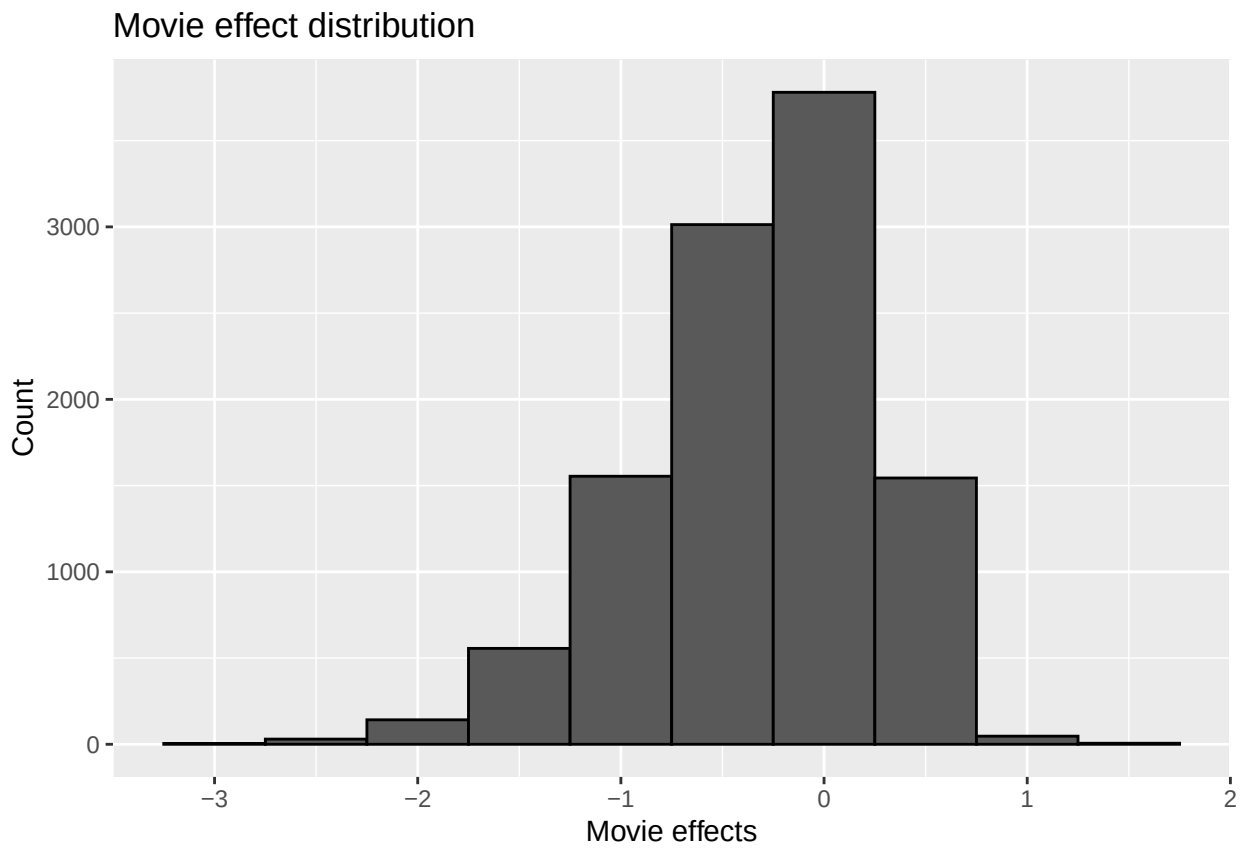
```
## [1] 1.061202
```

4.2.2 Movie Effect:

When movie effect is added to basic linear model it appears as:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

```
# Movie effect
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
#Movie effect distribution
qplot(b_i, data = movie_avgs, bins = 10, color = I("black")) +
  ggtitle("Movie effect distribution") +
  xlab("Movie effects") +
  ylab("Count")
```




```
#Predicted ratings with Movie Effect
predicted_ratings1 <- mu + validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)
#RMSE for Test set
RMSE_V2 <- RMSE(validation$rating, predicted_ratings1)
RMSE_V2
```

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

With movie effect there is a substential decrease in RMSE from 1.06 to 0.94.

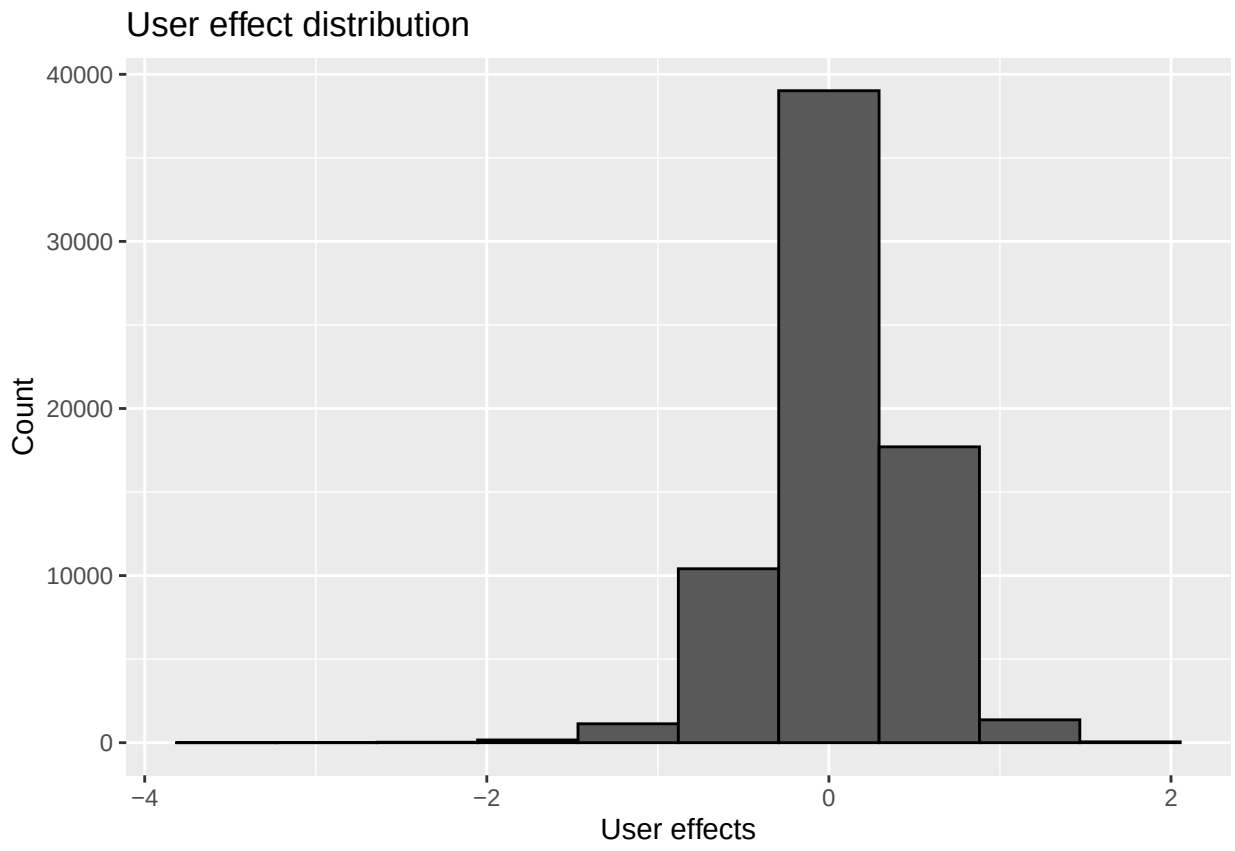
4.2.3 Movie + User Effect:

With user effect the formula for linear model is:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

```
# Movie+User Effect
user_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-mu-b_i))
#user effect distribution
```

```
qplot(b_u, data = user_avgs, bins = 10, color = I("black")) +
  ggtitle("User effect distribution") +
  xlab("User effects") +
  ylab("Count")
```




```
#Predicted rating with Movie + User Effect
predicted_ratings2 <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
#RMSE for test set
RMSE_V3 <- RMSE(validation$rating, predicted_ratings2)
RMSE_V3
```

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

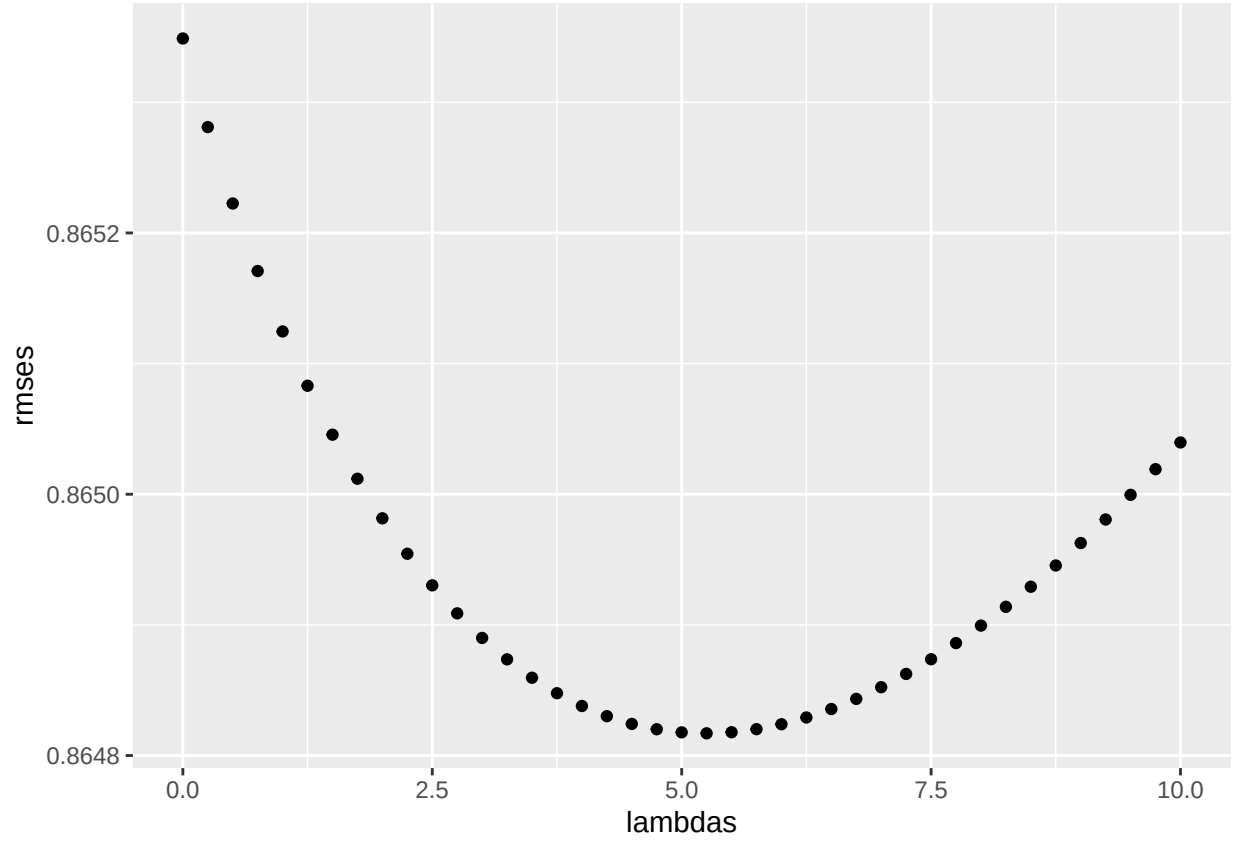
There is further improvement in RMSE after user effect as now it has been reduced to 0.8653488.

4.2.4 Regularization:

Now regularize the movie and user effects by adding a penalty factor λ, which is tunning parameter. Values of λ are defined and regularization is function is used to calculate the value which minimizes the RMSE.

```r
# Regularization
# Selecting the lambda a tunning parameter using cross-validation
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l) {
  mu <- mean(edx$rating)
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
  predicted_ratings_3 <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u ) %>%
    pull(pred)

  return(RMSE(validation$rating, predicted_ratings_3))
  })
qplot(lambdas, rmses)
```




```r
#For the full model, the optimal    is:
lambda <- lambdas[which.min(rmses)]
```

```
# Calculate the regularized RMSE on test set
RMSE_V4 <- min(rmses)
RMSE_V4
```

```
## [1] 0.864817
```

There is slight improvement in RMSE with regularization effect as it has been further reduced to 0.864817.

4.3 Matrix factorization:

Data may have different formats and types of storage, for example the input data set may be saved in a file or stored as R objects, and users may want the output results to be directly written into file or to be returned as R objects for further processing. In `recosystem`, we use two classes, `DataSource` and `Output`, to handle data input and output in a unified way.

```
#Install recosystem package for matrix factorization

library(recosystem)

# Matrix Factorization
set.seed(123)

# The data file for training and testing sets needs to be arranged in sparse form,
# each line in the file contains three numbers, "user_index", "item_index","rating".
# An object of class DataSource specifies the source of a data set,
# like data_memory(): Specifies a data set from R objects.
edx_4_mf <- with(edx, data_memory(user_index = userId,
                                  item_index = movieId,
                                  rating = rating))
validation_4_mf <- with(validation, data_memory(user_index = userId,
                                                item_index = movieId,
                                                rating = rating))

# Create a model object (a Reference Class object in R) by calling Reco().
r <- Reco()

# call the $tune() method to select best tuning parameters along a set of candidate values.
opts <- r$tune(edx_4_mf, opts = list(dim = c(10,20,30),
                                     lrate = c(0.1, 0.2),
                                     costp_l1 = 0,
                                     costq_l1 = 0,
                                     nthread = 1,
                                     niter = 10))
opts
```

```
## $min
## $min$dim
## [1] 30
##
## $min$costp_l1
## [1] 0
```

```
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.1
##
## $min$loss_fun
## [1] 0.7976821
##
##
## $res
##    dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
## 1   10        0     0.01        0     0.01   0.1 0.8252502
## 2   20        0     0.01        0     0.01   0.1 0.8065425
## 3   30        0     0.01        0     0.01   0.1 0.8152417
## 4   10        0     0.10        0     0.01   0.1 0.8279835
## 5   20        0     0.10        0     0.01   0.1 0.8016494
## 6   30        0     0.10        0     0.01   0.1 0.8012159
## 7   10        0     0.01        0     0.10   0.1 0.8283596
## 8   20        0     0.01        0     0.10   0.1 0.8015051
## 9   30        0     0.01        0     0.10   0.1 0.7976821
## 10  10        0     0.10        0     0.10   0.1 0.8371552
## 11  20        0     0.10        0     0.10   0.1 0.8314625
## 12  30        0     0.10        0     0.10   0.1 0.8273676
## 13  10        0     0.01        0     0.01   0.2 0.8163115
## 14  20        0     0.01        0     0.01   0.2 0.9159790
## 15  30        0     0.01        0     0.01   0.2 0.9260501
## 16  10        0     0.10        0     0.01   0.2 0.8169519
## 17  20        0     0.10        0     0.01   0.2 0.8595452
## 18  30        0     0.10        0     0.01   0.2 0.9104400
## 19  10        0     0.01        0     0.10   0.2 0.8219798
## 20  20        0     0.01        0     0.10   0.2 0.8001062
## 21  30        0     0.01        0     0.10   0.2 0.7991088
## 22  10        0     0.10        0     0.10   0.2 0.8410042
## 23  20        0     0.10        0     0.10   0.2 0.8263035
## 24  30        0     0.10        0     0.10   0.2 0.8258646
```

```
# Train the model by calling the $train() method.
# A number of parameters can be set inside the function,
# possibly coming from the result of $tune().
r$train(edx_4_mf, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9727   1.2036e+07
##    1       0.8718   9.8853e+06
##    2       0.8383   9.1699e+06
##    3       0.8168   8.7508e+06
```

```
##    4       0.8013   8.4722e+06
##    5       0.7898   8.2766e+06
##    6       0.7803   8.1277e+06
##    7       0.7723   8.0079e+06
##    8       0.7655   7.9099e+06
##    9       0.7595   7.8312e+06
##   10       0.7542   7.7637e+06
##   11       0.7496   7.7041e+06
##   12       0.7454   7.6548e+06
##   13       0.7416   7.6090e+06
##   14       0.7381   7.5718e+06
##   15       0.7350   7.5362e+06
##   16       0.7321   7.5047e+06
##   17       0.7294   7.4759e+06
##   18       0.7270   7.4530e+06
##   19       0.7248   7.4309e+06
```

```
# Use the $predict() method to compute predicted values.
# An object of class Output describes how the result should be output,
# out_memory(): Result should be returned as R objects

prediction_4_mf <- r$predict(validation_4_mf, out_memory())

RMSE_V5 <- RMSE(validation$rating, prediction_4_mf)
```

```
RMSE_V5
```

```
## [1] 0.7826517
```

5 Results:

5.1 RMSE of different models on Test Set:

```
Results <- tibble(Model = "Linear (Average Method)",
                  RMSE = RMSE_V1)
Results <- bind_rows(Results,
                     tibble( Model = "Linear (Average + Movie Effects)",
                             RMSE = RMSE_V2))
Results <- bind_rows(Results,
                     tibble( Model = "Linear (Average + Movie Effects + User Effects)",
                             RMSE = RMSE_V3))
Results <- bind_rows(Results,
                     tibble( Model = "Linear Regularized(Average + Movie Effects + User Effects)",
                             RMSE = RMSE_V4))
Results <- bind_rows(Results,
                     tibble( Model = "Matrix Factorization",
                             RMSE = RMSE_V5))
```

```
Results
```

Model	RMSE
Linear (Average Method)	1.0612018
Linear (Average + Movie Effects)	0.9439087
Linear (Average + Movie Effects + User Effects)	0.8653488
Linear Regularized(Average + Movie Effects + User Effects)	0.8648170
Matrix Factorization	0.7826517

5.2 Evaluation of Results:

In this MovieLens project we have examined the various recommender system algorithms to predict movie ratings for the 10M version of the Movielens data. While training the various machine learning models on the provided training set `edx` and testing it on the test set `validation` the models performance was evaluated through RMSE (Root Mean Squared Error) method. From the `Results` tibble it is visible that RMSE for Linear Model with `Average Method` was 1.0612 which further reduced to 0.9439 when `Movie Effects` were incorporated and further to 0.8653 with addition of `User Effects`. However the lowest RMSE of 0.86481 for linear model was attained with Regularized Linear Model.
The Matrix factorization method by `recosystem` produced the lowest RMSE of 0.7823178. It is the same method which was used and made popular by the prize winning team of Netflix Challange.

5.3 Future Works:

We only considered linear model and collaborative filtering represented by its sub class Matrix Factorization for recommendation system. There is another category called Content based filtering used for building recommendation systems. `recommenderlab` is a popular r package to develope a content based filtering movie recommendation system and it can be considered for this project purposes.

6 Conclusion:

This project is 9th also the last one in the series of courses for Data Science Professional Certificate by HarvardX. To accomplish this project the data was provided through through the code was provided on the official page of Data Science: Capstone but due to some errors the code failed to retrieve the data and therefore a link was provided to download the training set `edx` and test set `validation`. After downloading the required data first it was checked for the consistency and accuracy. Later the `edx` was explored through various data exploration techniques to get complete understanding of the data. Then various statestical learning methods were applied at the `edx` set to train data and then they were tested on `validation` set. The method used to evaluate these models was RMSE (Residual Mean Square Error) and the relevant RMSE was calculated for each of these models on `validation` set. The matrix factorization model produced the lowest RMSE. Thus it is the recommended model for Movie Recommendation System.

6.1 Limitations:

Various machine learning models require extensive computation and are difficult to measure on common household computers as they require much more virtual memory and computing speed and are thus excluded.
For linear model in this project the `timestamp` variable has been ignored.
Only two predictors Movie and User are considered to build recommendation system.
As model only considers existing users, movies and rating values, therefore when ever a new user rates movie

the new changes must be considered and incorporated.
There is no recommendation for a new user or for users what don't rate a movie.

6.2 References:

www.edx.org/professional-certificate/harvardx-data-science
https://github.com/rafalab/dsbook
https://cran.r-project.org/web/packages/recosystem/index.html
https://www.netflixprize.com