edx - Movie Lens Project

Hesham Al Kayed

12 May 2019

1 Introduction

This Project is part of *edx - Data Science course*[1] , in which we want to build a recommendation model using the *MovieLens* dataset. *A recommender system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. They are primarily used in commercial applications.* [2]

In our case, we want to predict the rating a user would give to a given movie based on the provided dataset, in this project we are provided with a pre-wrangled version of *MovieLens* dataset which can be obtained using code in `Section.4`. to build our model this dataset will still need extra formatting as described in `Section 2.1`. Our first step to build our model is to understand data at hand; we need to explore our dataset, and find out the distribution of our predictors and how they affect movie ratings, `Section 2.2` explore the relation between between movie ratings and other predictors.

In `Section 3` we build a model based on the effect, or the average error each predictor contributed to the distance between the actual rating and the over all average. only the *user*, *movie* and *year* effect are considered, because of the limitations on the used PC, and as we will see ,only using these three parameters will achieve the required model performance.

2 Exploratory data analysis

2.1 Data Structer

Examining training set, we find that we have 9000055 rows and 6 columns, `Table.1` shows the header of our dataset,it is logical to assume that each user have one review only for each movie, below code confirms our assumption about the uniqueness of *usesrId + movieId* by showing no duplicates.

```
edx %>% group_by(userId, movieId) %>% summarise(N=n()) %>% filter(N >1)
```

Table 1: Training Set Header

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

title column holds the title of the movie and the year it was aired, separation of title and year will be more useful. *genres* is stored as a string which can be separated by "|", after separating genres we find 20 unique genres. *timestamp* should be changed to a readable format, and for the sake of simplicity only the year will be considered.

Table 2: Types and Missing Data

	userId	movieId	rating	timestamp	title	genres
Type	integer	double	double	integer	character	character
NA	0	0	0	0	0	0
Empty	0	0	0	0	0	0

`Table.2` shows that training data is complete with no missing or empty records.

[1] https://www.edx.org/professional-certificate/harvardx-data-science
[2] https://en.wikipedia.org/wiki/Recommender_system

Table 3: EDX Header After Formating

userId	movieId	rating	timestamp	title	genres	year
1	122	5	9710	Boomerang	c(“Comedy”, “Romance”)	1992
1	185	5	9710	Net, The	c(“Action”, “Crime”, “Thriller”)	1995
1	292	5	9710	Outbreak	c(“Action”, “Drama”, “Sci-Fi”, “Thriller”)	1995
1	316	5	9710	Stargate	c(“Action”, “Adventure”, “Sci-Fi”)	1994
1	329	5	9710	Star Trek: Generations	c(“Action”, “Adventure”, “Drama”, “Sci-Fi”)	1994
1	355	5	9710	Flintstones, The	c(“Children”, “Comedy”, “Fantasy”)	1994

2.2 Data Description

The purposes of this study is to build a model to predict movie ratings that will be given by a given user, so we will start there.

Table 4: Ratings Summary

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.5	3	4	3.512465	4	5

	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
%	0.95	3.84	1.18	7.9	3.7	23.57	8.8	28.76	5.85	15.45

- **Users tened to rate in whole numbers insted of fractions.**
- **Most ratings are above 2.5 .**
- **Highest rating used is 4.0 followed by 3.0 and 5.0.**

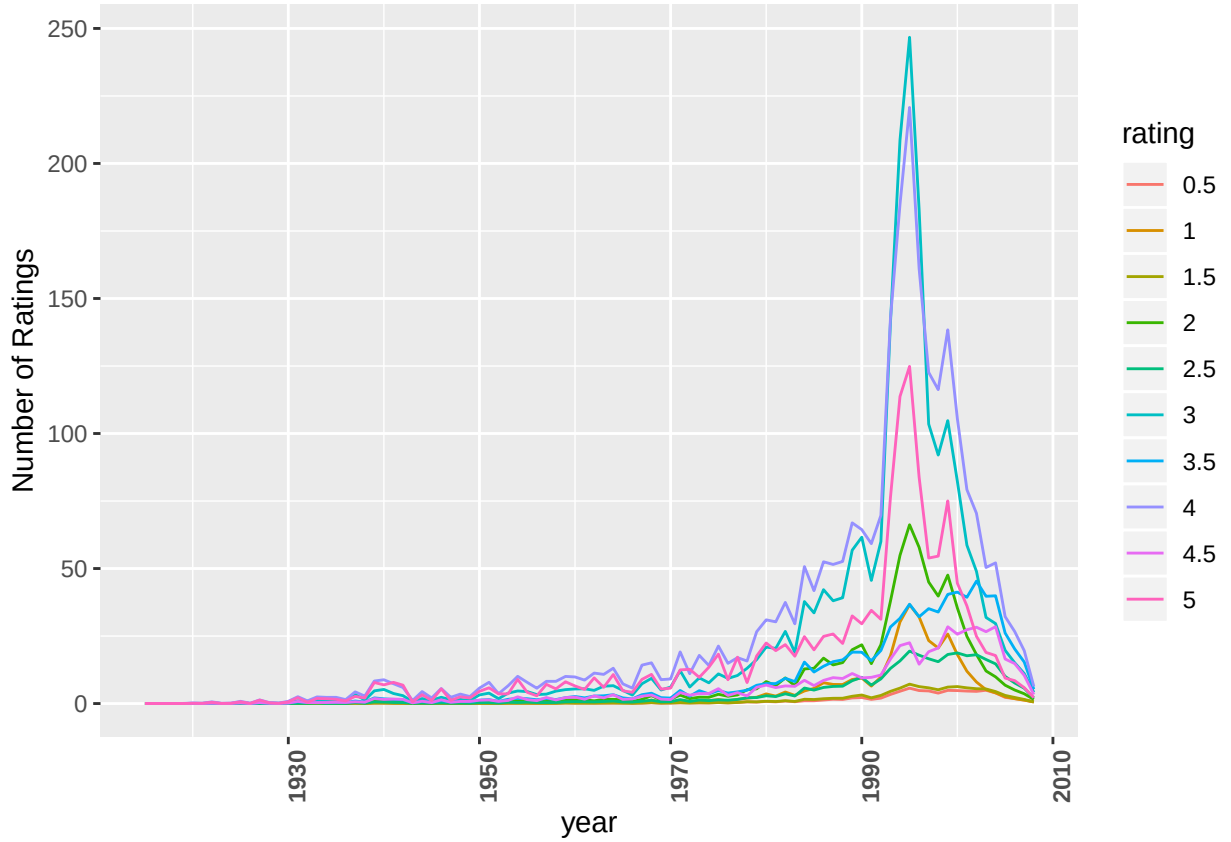

Figure 1: Numer of reviews per year

Looking at `Fig.1`, we can see that even after breaking *rating* description in `Table.4` into a year based summary, that the same pattern holds, which is most reviews averaged at **4.0** followed by **3.0** and **5.0**, which shows that in general people tends to give above average ratings. `Fig.2` verifies this assumption.

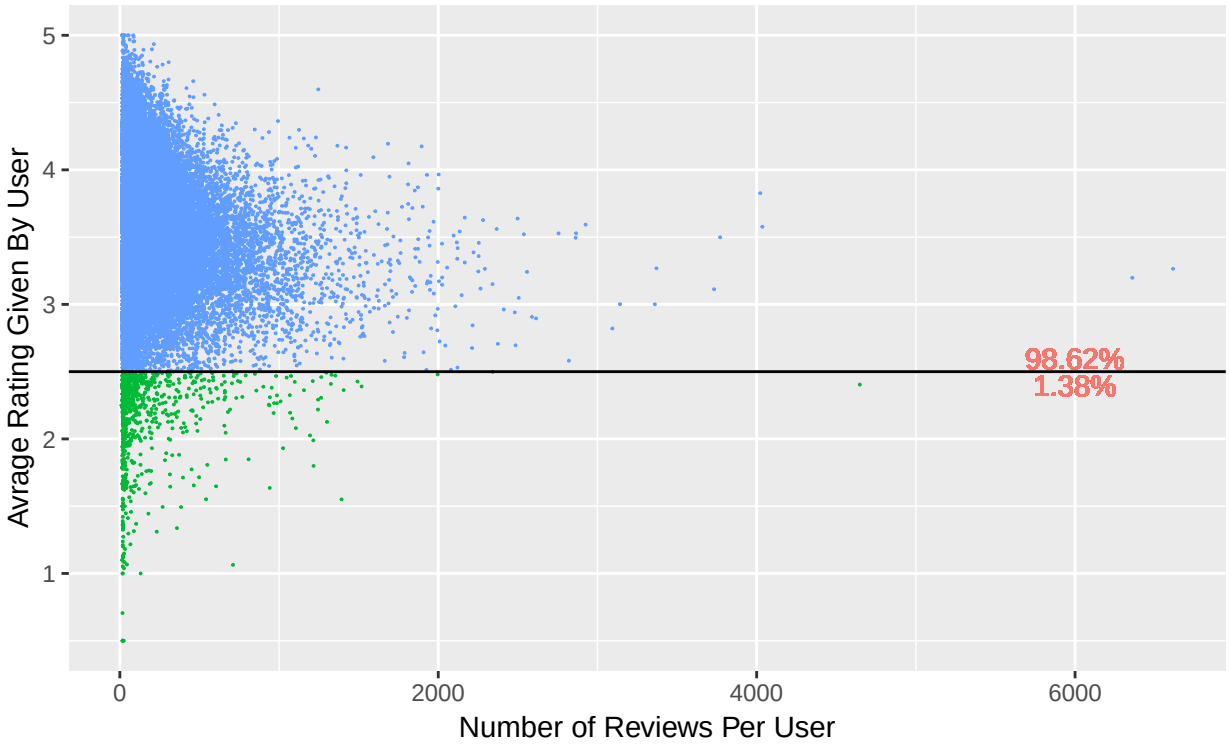

Figure 2: User Avrage Rating

Also we can see that some users gave an average rating above **4.5**, which means they like every movie they watch, and others tend to dislike every movie they like. looking at `Fig.3`, we can see that users who gave an average above **4.5** are more consistent with their ratings (*low standard deviation*) compared to low average users.

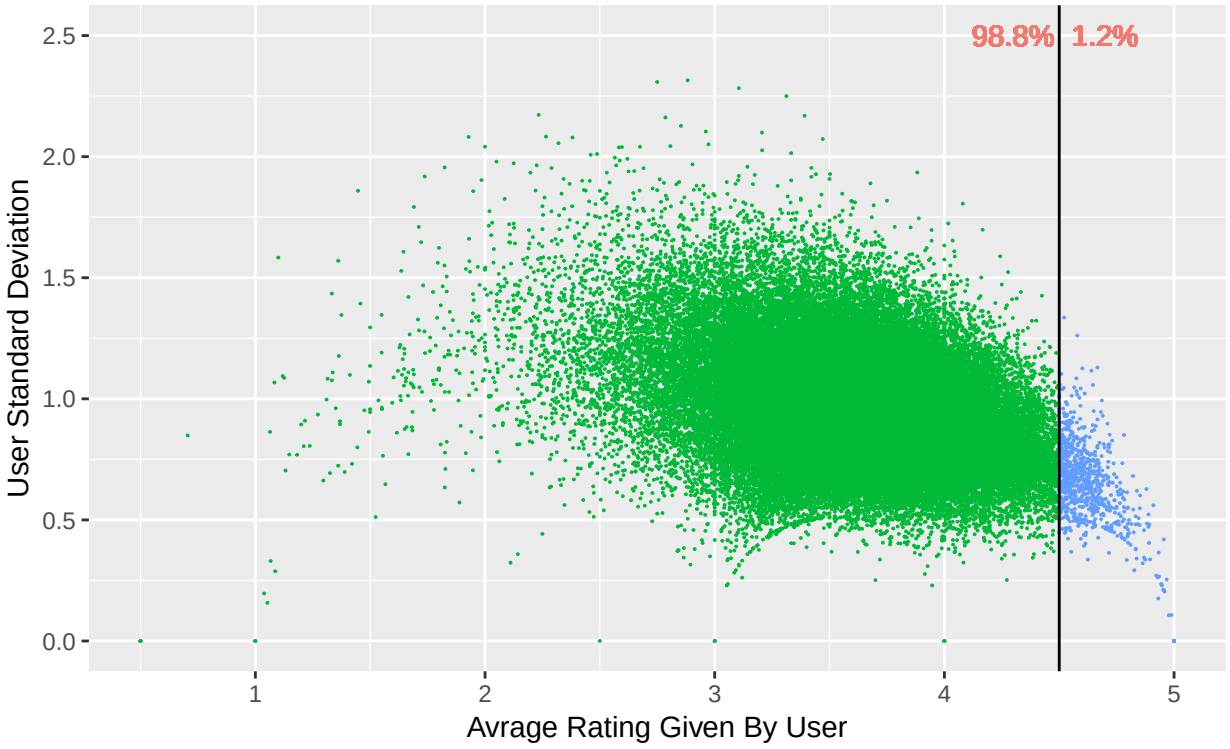

Figure 3: Rating Spread Per User

by examining reviews *timestamps*, we can see that all reviews was taken from 1995, 2009, looking at `Fig.4` we can see that most users tends to review the most recent movies. and not all users are interested in classical movies.

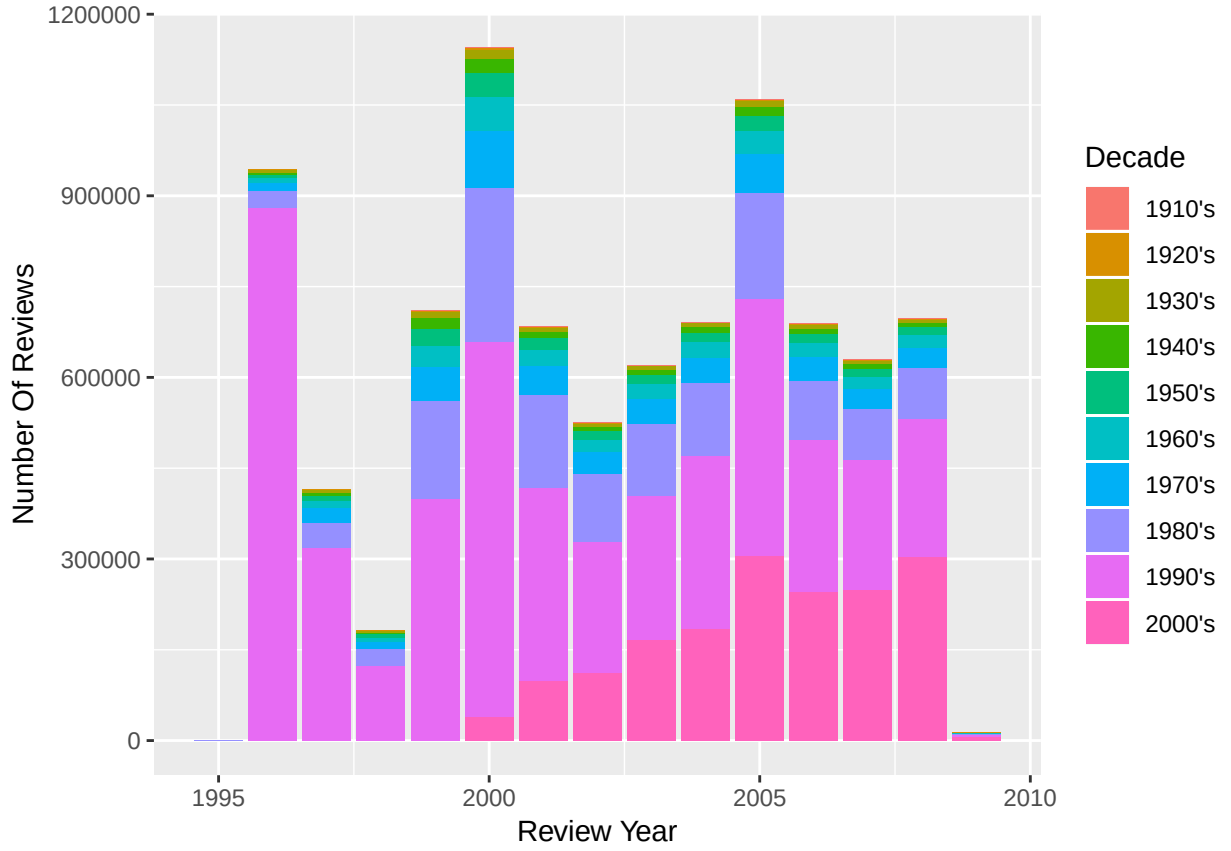

Figure 4: Decade Popularity

Examining available movies, we find that there are 10677 unique titles. from which 6titles have perfect score. but as we can see in `Table.6`, these movies have one or two reviews only, and if we filter for movies with a standard deviation less than **0.01** we get a range of Reviews per movie between 2, 4; this give us the indication that movies with low number of reviews will give unreliable results; filtering for movies with more than ten reviews, we get `table.7` which makes much more sense.

Table 6: Top Movies With The Highest Average

movieId	title	Number_Reviews	Avgerage_Rating	Standard_Deviation
3226	Hellhounds on My Trail	1	5	NA
33264	Satan's Tango (Sátántangó)	2	5	0
42783	Shadows of Forgotten Ancestors	1	5	NA
51209	Fighting Elegy (Kenka erejii)	1	5	NA
53355	Sun Alley (Sonnenallee)	1	5	NA
64275	Blue Light, The (Das Blaue Licht)	1	5	NA

Table 7: Top Movies With The Highest Average; N > 10

movieId	title	Number_Reviews	Avgerage_Rating	Standard_Deviation
318	Shawshank Redemption, The	28015	4.46	0.72
858	Godfather, The	17747	4.42	0.81
50	Usual Suspects, The	21648	4.37	0.76
527	Schindler's List	23193	4.36	0.81
904	Rear Window	7935	4.32	0.73
912	Casablanca	11232	4.32	0.82

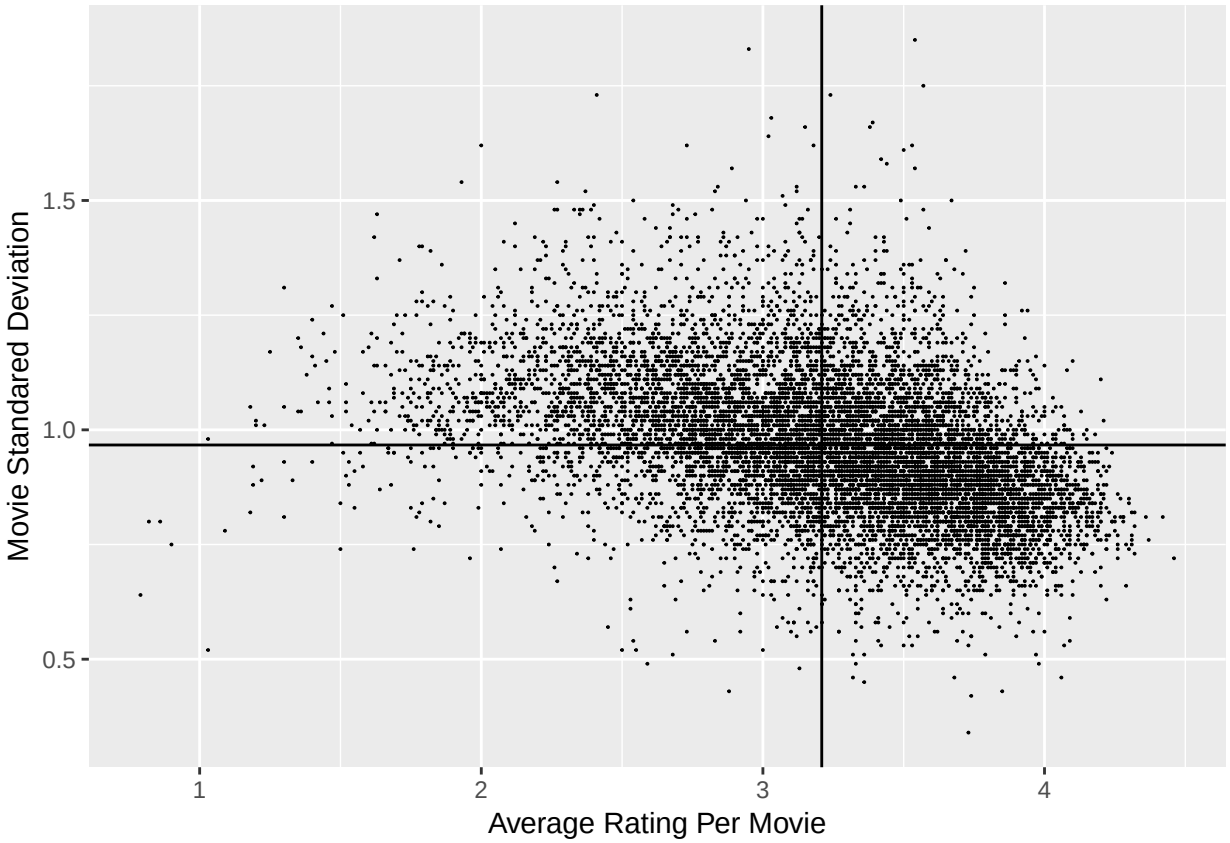

Figure 5: Rating Spread Per Movie

As we can see in `Fig.5`, movie ratings has an average standard deviation of 0.97, and a mean of 3.21. movies that are above average have a lower deviation compared to below average movies.

2.3 Conclusions

- Users tends to give ratings above the center of the used scale;**2.5**. at an average of **3.5**.
- In general, the most used rating is **4.0**, and this trend is followed regardless of the year.
- Users differ in their ratings, but it can be summarized that the more a user is consistent with their rating the higher the average rating they give, but the same does not apply for users with high deviation in their ratings.
- New movies are more popular than classical ones, still classical movies have their base of users.
- as in users, in general movies with low deviation have higher ratings.

3 Buliding Model

3.1 Formlation

I am going to build a model as described in edx - Data Science: Machine Learning - Recommendation Systems. we will set Y as the actual rating, μ as the over all average, ϵ as the error or distance from μ. which can be interpreted as, *all moveis should have a rating of* μ, *but for some effect* ϵ, *the rating deviates to* Y.

$$Y = \mu + \epsilon$$

from our data exploration we can break ϵ into *user effect* b_u, *movie effect* b_i and *year effect* b_y, which leaves us with.

$$Y = \mu + b_i + b_u + b_y$$

above model assumes that there are only three effects, which we know from experience that is not correct, to counter that we add ϵ as random error.

$$Y = \mu + b_i + b_u + b_y + \epsilon \Rightarrow Y = Y_{hat} + \epsilon$$

3.2 Calculating Effects

by including b_u and b_y in the random error parameter, we can assume that,

$$b_i + \epsilon = Y - \mu$$

so we can calculate b_i using below code,

```
# Calculationg over all rating avrage.
mu <-mean(edx$rating)
```

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

as we now have an estimation for b_i, we can estimate b_u and keep the year effect as part of our random error parameter,

$$b_u + \epsilon = Y - \mu - b_i$$

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

and the same goes for year effect b_y,

$$b_y + \epsilon = Y - \mu - b_i - b_u$$

```
# Calculating year effect
year_avg  <- edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(user_avg, by= "userId") %>%
  group_by(year) %>%
  summarise(b_y= mean(rating - mu - b_i - b_u))
```

Based on our assumption, we know that $Y_{hat} = \mu + b_i + b_u + b_y$; so we need to collect the above calculated effects and add them together, below function will facilitates that.

```
# takes df with the same structure as edx
# returns input df binded with b_i, b_u, b_y and predicted rating y_hat

PredictRating <- function(df){
  df %>%
    left_join(movie_avg, by = "movieId") %>% # collect movie effect
    left_join(user_avg, by="userId") %>%     # collect user effect
    left_join(year_avg, by = "year") %>%     # collect year effect
    mutate(y_hat = mu + b_i  + b_u + b_y,    # calculate y_hat
           y_hat = if_else(y_hat >  5 ,  5.0, y_hat), # make sure that our predictions are within range
           y_hat = if_else(y_hat < 0.5 , 0.5, y_hat))} # make sure that our predictions are within range
```

3.3 Model Performance

Model performance is measured in `RMSE` [3], where N is the number of records.

$$RMSE = \sqrt{\frac{\sum^{N}(Y - Y_{hat})^2}{N}}$$

```
edx %<>% PredictRating()
```

Table 8: Model Performance; Test Set

Included.Effects	RMSE	Improvement
mu	1.0603313	0.00
mu + b_i	0.9423475	11.13
mu + b_i + b_u	0.8567039	9.09
mu + b_i + b_u + b_y	0.8563777	0.04

RMSE score is under the required **0.9**.

apply the same formatting to the test set, then applying *predict function* we get.

```
validation%<>% PredictRating()
```

Table 9: Model Performance; Test Set

Included.Effects	RMSE	Improvement
mu	1.0612018	0.00
mu + b_i	0.9439087	11.05
mu + b_i + b_u	0.8653488	8.32
mu + b_i + b_u + b_y	0.8650043	0.04

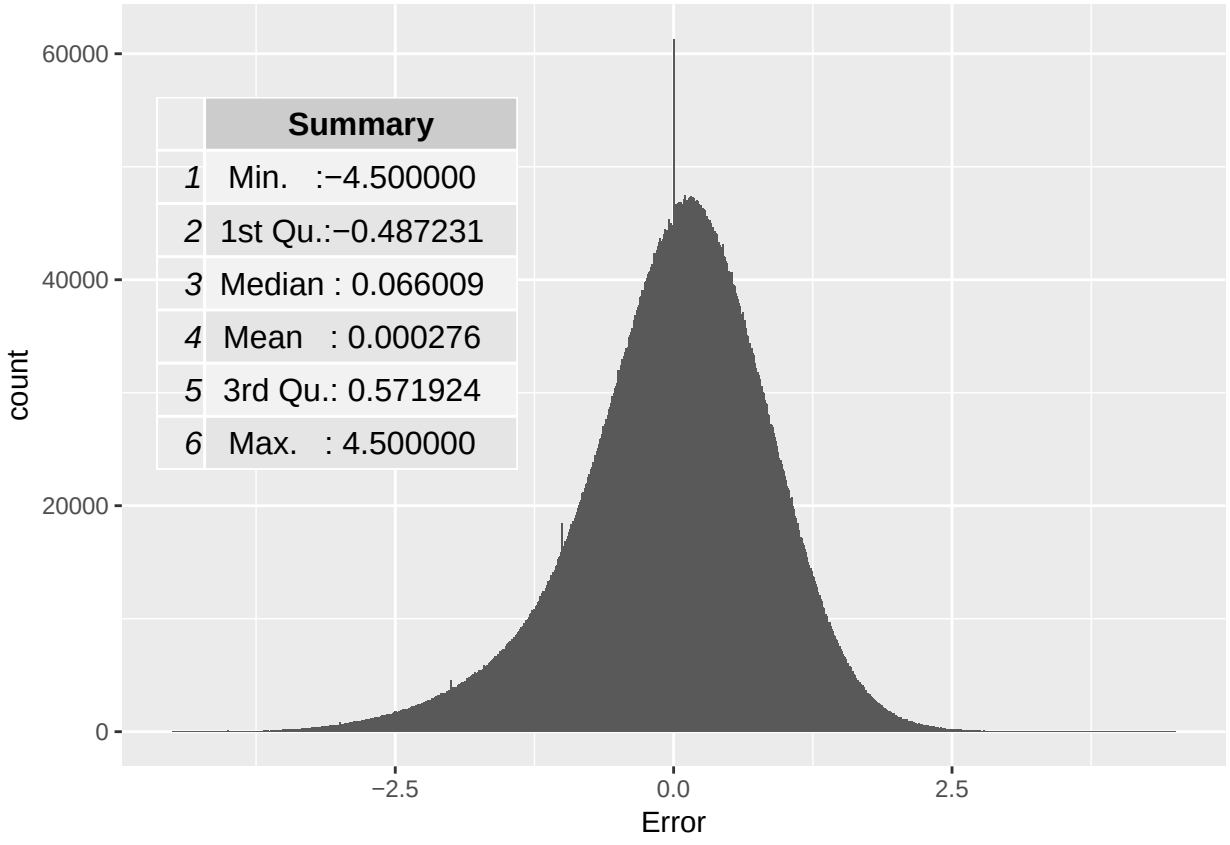

Figure 6: Error Distribution, training set

[3] https://en.wikipedia.org/wiki/Root-mean-square_deviation

4 Reference Code - Get Datasets

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################

# Note: this process could take a couple of minutes

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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data

set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```