

MovieLens Recommendation System Project

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Objective | 2 |
| 1.2 | Dataset | 3 |
| 2 | Methods and Analysis | 5 |
| 2.1 | Data Exploration and Cleaning | 5 |
| 2.1.1 | Overview | 5 |
| 2.1.2 | Genres Observations | 7 |
| 2.1.3 | Timestamp Observations | 9 |
| 2.1.4 | Title and Year | 11 |
| 2.1.5 | Data Cleaning | 12 |
| 2.2 | Models | 12 |
| 2.2.1 | The Naive Average Rating Movie Method | 12 |
| 2.2.2 | Movie Effect Model | 13 |
| 2.2.3 | Movie + User Effect Model | 14 |
| 2.2.4 | Movie + User + Year_Movie Effect Model | 16 |
| 2.2.5 | Movie + User + Year_Movie + Year rated Effects Model | 16 |
| 2.2.6 | Movie + User + Year_Movie + Year rated + Genres Effects Model | 17 |
| 2.2.7 | Regularized Movie + User Effect Model | 18 |
| 3 | Results | 21 |
| 4 | Conclusion | 22 |

1 Introduction

Recommendation Systems are popular algorithms that can help businesses make better propositions to their customers and thus increase their growth. They use ratings given by their users for products to predict what rating a certain user might give a product. Netflix for instance, uses a recommendation system to predict which movie a user might enjoy watching based on a ratings system.

This data science project is for the HarvardX PH125.9x Data Science: Capstone. We will create a recommendation system similar to the Netflix one using the MovieLens 10M dataset provided by GroupLens.

1.1 Objective

The goal of this project will be to create a model that predicts user ratings of a movie using different features values. To calculate the accuracy of our algorithm, we will use a loss function called the residual mean squared error or RMSE on a test set. The RMSE is defined as follow:

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

Where $y_{u,i}$ represents the rating for movie i by user u , and $\hat{y}_{u,i}$ denotes the prediction for movie i by user u .

```
#RMSE function  
RMSE <- function(true_ratings, predicted_ratings){  
  sqrt(mean((true_ratings - predicted_ratings)^2))  
}
```

Our aim we will be to develop an algorithm with a $RMSE < 0.86490$.

In order to create this algorithm, we will need to first separate the data into two different sets one for training and one for testing. Then we will explore and analyse the data, develop methods to predict ratings, compare the different methods using the results and finally conclude which one is the best algorithm.

1.2 Dataset

To download MovieLens 10M, we use the code provided `edx`. This code also adds libraries that we will be using to conduct analysis and splits the data into two different sets, one for training, the `edx` dataset, and one for testing, the validation dataset.

```
#####  
# Create edx set, validation set (final hold-out test set)  
#####  
  
# 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.r-project.org")  
if(!require(data.table)) install.packages("data.table", repos =  
                                           "http://cran.us.r-project.org")  
if(!require(lubridate)) install.packages("lubridate", repos =  
                                          "http://cran.us.r-project.org")  
if(!require(gridExtra)) install.packages("gridExtra", repos =  
                                          "http://cran.us.r-project.org")  
  
library(tidyverse)  
library(caret)  
library(data.table)  
library(lubridate)  
library(gridExtra)  
  
# MovieLens 10M dataset:  
# https://grouplens.org/datasets/movielens/10m/  
# http://files.grouplens.org/datasets/movielens/ml-10m.zip  
  
dl <- tempfile()  
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)  
  
ratings <- fread(text = gsub(":", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),  
                 col.names = c("userId", "movieId", "rating", "timestamp"))  
  
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\:", 3)  
colnames(movies) <- c("movieId", "title", "genres")  
  
# if using R 3.6 or earlier:  
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],  
                                           title = as.character(title),  
                                           genres = as.character(genres))  
  
# if using R 4.0 or later:  
  
movielens <- left_join(ratings, movies, by = "movieId")  
  
# Validation set will be 10% of MovieLens data  
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`  
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)  
edx <- movielens[-test_index,]
```

```
temp <- movielens[test_index,]  
  
# 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)
```

2 Methods and Analysis

2.1 Data Exploration and Cleaning

2.1.1 Overview

Let's first explore the data.

Here is what the first 6 data points look like:

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

```
dim(edx)
```

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

```
names(edx)
```

```
## [1] "userId"      "movieId"      "rating"      "timestamp" "title"      "genres"
```

We can see that there are 9000055 ratings and 6 columns: “userId”, “movieId”, “rating”, “timestamp”, “title” and “genres”. MovieId and userId represent distinctive Id numbers representing in the following order a particular movie and particular user. The ‘timestamp’ is the time in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970, when the movie was rated. The ‘title’ field is simply the name of the movie and the ‘genres’ define the genres of the movie. The ‘rating’ is a whole or half number between 0 and 5 where 0 means that the user hated the movie and 5 the user loved that movie.

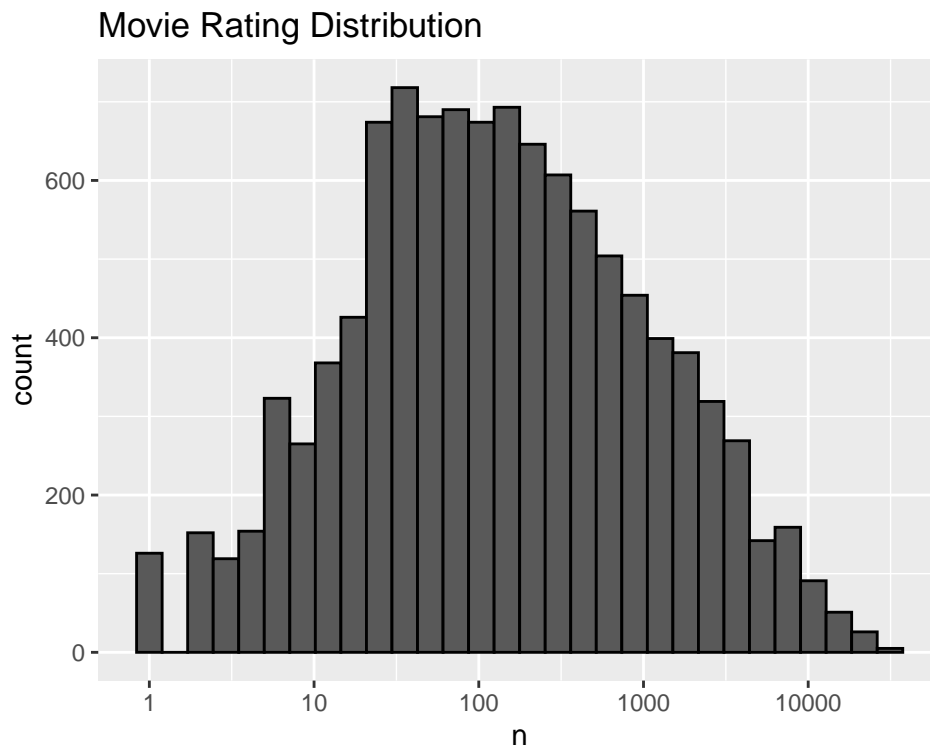
There are 69878 unique users and 10677 different movies:

```
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##      n_users n_movies
## 1      69878    10677
```

We can observe that not all movie are rated as often. Some are rated more and some less.

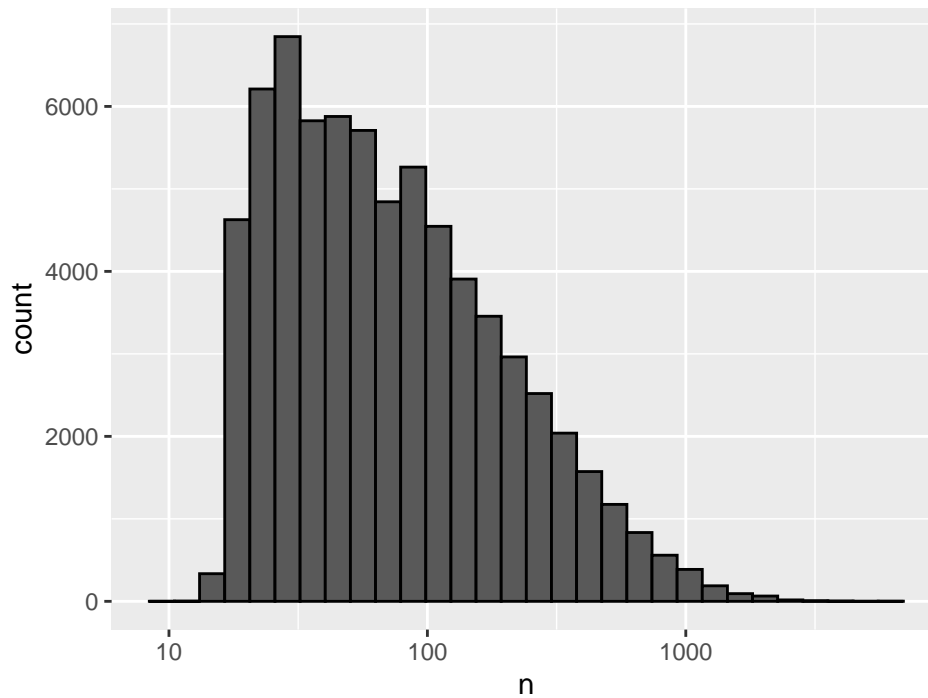
```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movie Rating Distribution")
```



Similarly we can observe that not all users rate at the same recurrence. Some rate often, some rate less.

```
edx %>%  
  dplyr::count(userId) %>%  
  ggplot(aes(n)) +  
  geom_histogram(bins = 30, color = "black") +  
  scale_x_log10() +  
  ggtitle("Users Rating Distribution")
```

Users Rating Distribution



The 5 most given ratings are:

```
## Selecting by count
## # A tibble: 5 x 2
##   rating count
##   <dbl> <int>
## 1     4 2588430
## 2     3 2121240
## 3     5 1390114
## 4    3.5 791624
## 5     2 711422
```

The top 10 most ranked movies are:

```
## # A tibble: 10 x 2
##   title count
##   <chr> <int>
## 1 Pulp Fiction (1994) 31362
## 2 Forrest Gump (1994) 31079
## 3 Silence of the Lambs, The (1991) 30382
## 4 Jurassic Park (1993) 29360
## 5 Shawshank Redemption, The (1994) 28015
## 6 Braveheart (1995) 26212
## 7 Fugitive, The (1993) 25998
## 8 Terminator 2: Judgment Day (1991) 25984
## 9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10 Apollo 13 (1995) 24284
```

2.1.2 Genres Observations

We can observe that every movie can be of different genres:

```
##               title               genres
## 1:      Boomerang (1992)      Comedy|Romance
## 2:      Net, The (1995)      Action|Crime|Thriller
## 3:      Outbreak (1995) Action|Drama|Sci-Fi|Thriller
## 4:      Stargate (1994)      Action|Adventure|Sci-Fi
## 5: Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi
## 6:      Flintstones, The (1994) Children|Comedy|Fantasy
```

There are exactly 20 different genres and 797 different combinations of genres.

```
popular_genres <- edx %>% group_by(genres) %>% summarize(count = n()) %>% arrange(desc(count))
different_genres <- popular_genres %>% select(genres) %>%
  separate_rows(genres, sep = "\\|") %>% unique()
number_different_genres <- nrow(different_genres)

different_combination <- edx %>% summarize(n_genre = n_distinct(genres)) %>% .$n_genre
genres <- data_frame(n_genres = number_different_genres, n_combination_genres = different_combination)

## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.

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

| n_genres | n_combination_genres |
|----------|----------------------|
| 20 | 797 |

Every movie has a genre, and if the genre is not defined it is marked as even if it is not listed “(no genres listed)”

```
#There is no NA genres
edx %>% filter(genres == NA)
```

```
## Empty data.table (0 rows and 6 cols): userId,movieId,rating,timestamp,title,genres
edx %>% select(title, genres) %>% filter(genres == "(no genres listed)") %>% slice(1:1)
```

```
##               title               genres
## 1: Pull My Daisy (1958) (no genres listed)
```

The top 10 genres are:

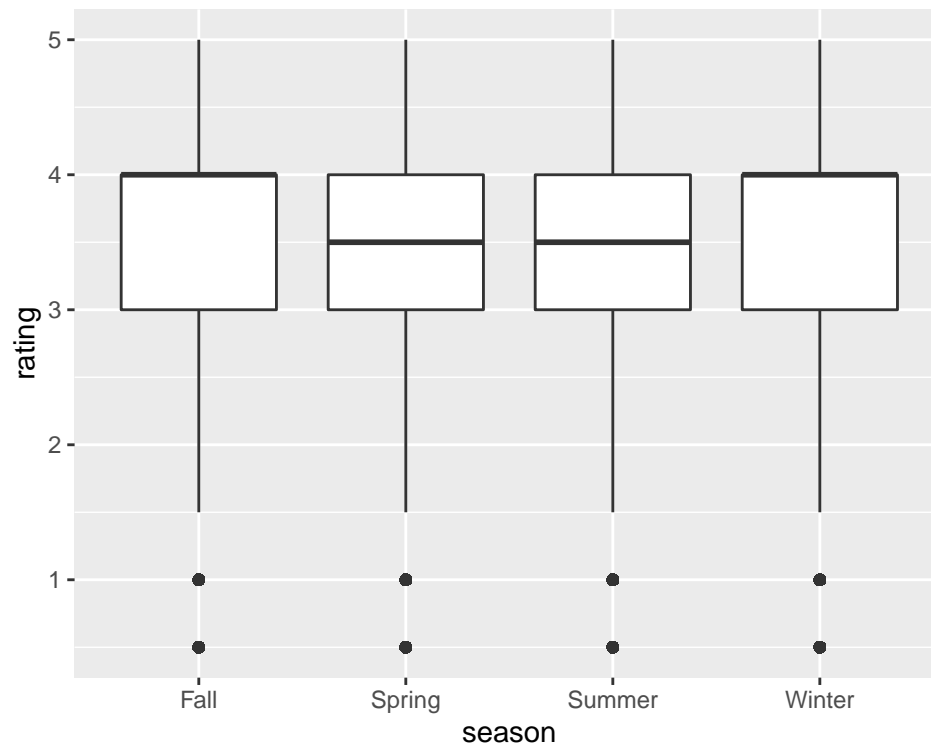
```
## # A tibble: 10 x 2
##   genres               count
##   <chr>              <int>
## 1 Drama              733296
## 2 Comedy             700889
## 3 Comedy|Romance     365468
## 4 Comedy|Drama       323637
## 5 Comedy|Drama|Romance 261425
## 6 Drama|Romance      259355
## 7 Action|Adventure|Sci-Fi 219938
## 8 Action|Adventure|Thriller 149091
## 9 Drama|Thriller      145373
## 10 Crime|Drama        137387
```


2.1.3 Timestamp Observations

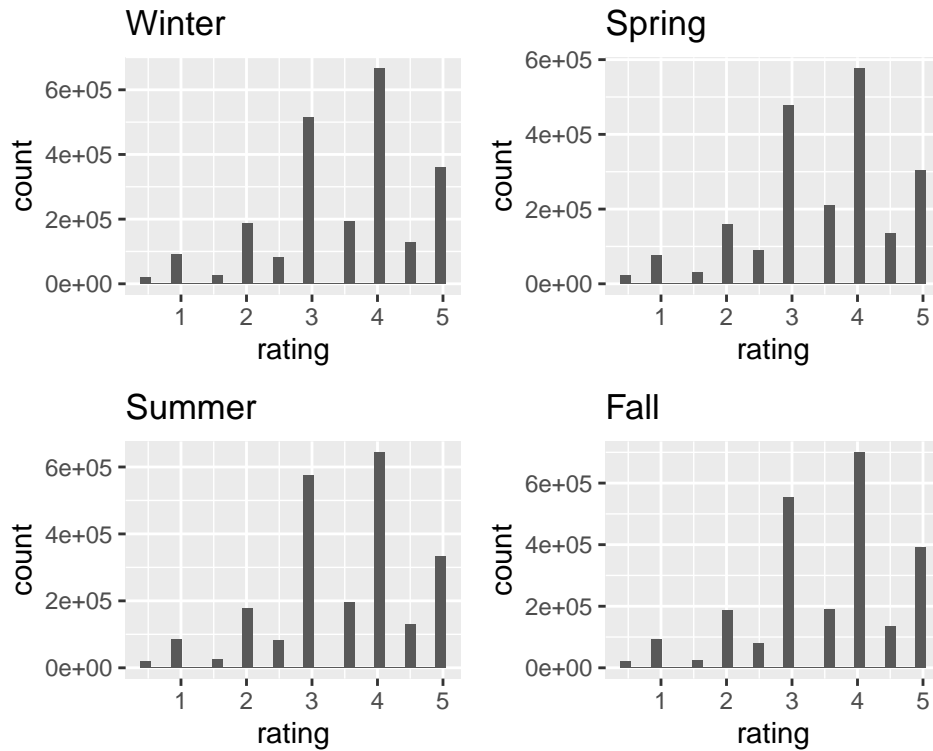
Firstly, let's convert the timestamp into a time that is more readable. We will create a new dataset from `edx` to add new fields. The field `time` represents the new time as a date time. From the 'time' field, we will extract the year, month and season, assuming that all the ratings were done in the northern hemisphere.

```
new_edx <- edx %>% mutate(time=as_datetime(timestamp))
new_edx <- new_edx %>% mutate(year=year(time), month=month(time))
new_edx <- new_edx %>% mutate(season=case_when(.$month%in%c(12,1,2)~"Winter",
                                              .$month%in%c(3,4,5)~"Spring",
                                              .$month%in%c(6,7,8)~"Summer",
                                              .$month%in%c(9,10,11)~"Fall"))
```

Let's now analyze the distribution of ratings per season and see if the season might influence the way a user rates a movie.



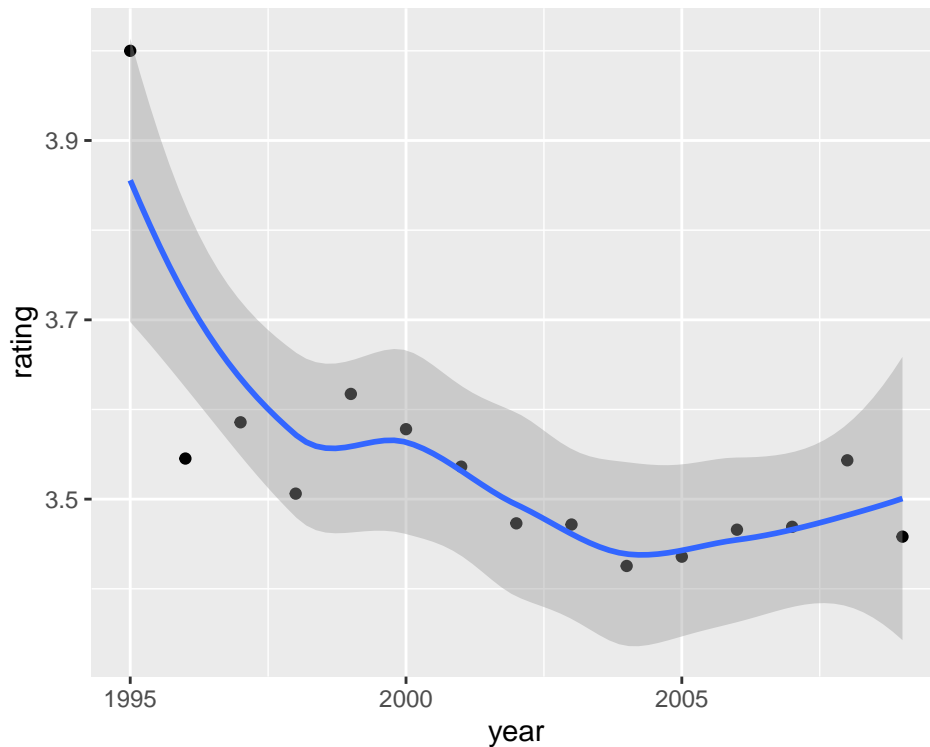
There does not seem to be not of variability between seasons but let's make a plot counting the number of different ratings per season:



We can observe that the season is not a good indicator in predicting the rating of a movie as it seems to be pretty constant every season. We will then ignore the season field.

Let's investigate if the year a movie was rated influences the ratings metric.

```
new_edx %>% group_by(year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year, rating)) + geom_point() + geom_smooth()
```



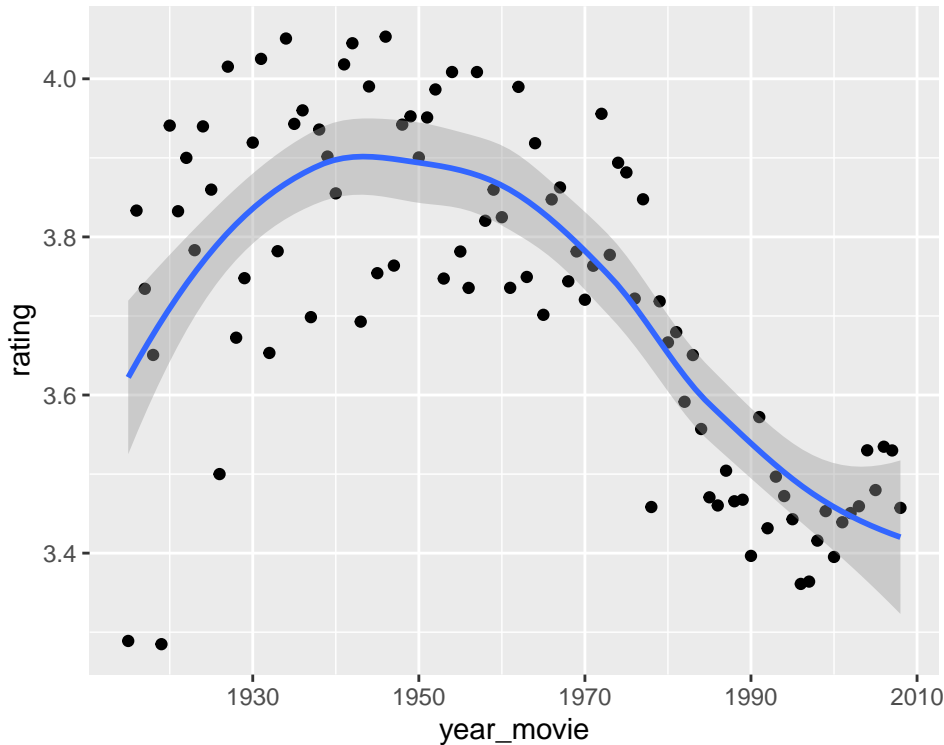
We can definitely see more variability that ranges between 4 and 3.4, we will use this metric to predict the rating of a movie.

2.1.4 Title and Year

Each movie title has in its title the year the movie came out. Let's extract that year and see if it has any influence on the rating of the movie.

```
new_edx<-new_edx%>%mutate(year_movie = substr(title,nchar(as.character(title))-4,nchar(as.character(title)))
#we first need to convert this as a date, so we will add jan 1st to the year. This will allow us to extract
new_edx<- new_edx %>% mutate(year_movie=paste(year_movie,"-01-01", sep = ""))
new_edx<- new_edx %>% mutate(year_movie=year(year_movie))
new_edx %>% group_by(year_movie) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(year_movie, rating)) + geom_point() + geom_smooth()

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can observe that the year the movie was released influences the rating so we will use it in our models.

2.1.5 Data Cleaning

Now let's only select the data we will use for our models from the new edx dataset and apply the same logic to the validation dataset.

```
#new edx dataframe
new_edx<- new_edx%>% select(-time,-timestamp,-season,-month)

#new validation dataframe
new_validation <- validation %>% mutate(time=as_datetime(timestamp))
new_validation <- new_validation %>% mutate(year=year(time))
new_validation<-new_validation %>%
  mutate(year_movie = substr(title,nchar(as.character(title))-4,nchar(as.character(title))-1))
new_validation<-new_validation%>%mutate(year_movie=paste(year_movie,"-01-01", sep = ""))
new_validation<-new_validation%>%mutate(year_movie=year(year_movie))
new_validation <- new_validation%>%select(-time,-timestamp)
```

2.2 Models

For the following models we will use the RMSE function that we defined earlier to calculate the accuracy of our model. We can interpret the RMSE similarly to a standard deviation: 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.

2.2.1 The Naive Average Rating Movie Method

Let's predict that the rating of a movie is similar for all movies regardless of the user. This model will assume the same rating for all movies and users with all the differences explained by random variation. The function

is defined as follow:

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

where $\epsilon_{u,i}$ is independent errors sampled from the same distribution centered at 0 and μ the “true” rating for all movies.

```
#calculating the average of all ratings
mu_hat <- mean(new_edx$rating)
mu_hat

## [1] 3.512465

#calculate the RMSE
naive_rmse <- RMSE(new_validation$rating, mu_hat)
naive_rmse

## [1] 1.061202

#Store the value of the RMSE in a table
rmse_results <- data_frame(method = "Just the average rating", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

| method | RMSE |
|-------------------------|----------|
| Just the average rating | 1.061202 |

We get an RMSE of 1.06 which means that we predict with an error of one star which is pretty significant. Let's see if we can do better.

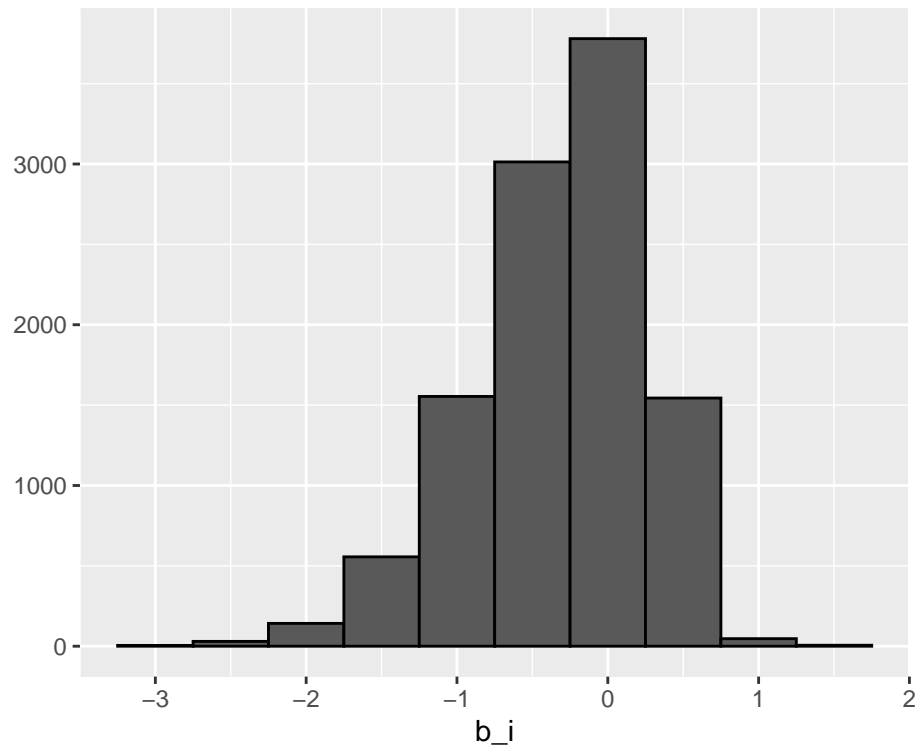
2.2.2 Movie Effect Model

As we have observed earlier, we know that some movies are rated higher than others. Let's include this in our model by adding a bias b_i to represent average ranking for movie i :

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

```
mu <- mean(new_edx$rating)
#caculate b_i
movie_avgs <- new_edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We can see that b_i varies between approximately -3 and 1.5.



Please note that μ was equal to roughly 3.5, which implies that a perfect star would be given when b_i is equal to 1.5.

Let's now evaluate our model:

```
#calculate b_i for new_validation
predicted_ratings <- mu + new_validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

#calculate the RMSE
model_1_rmse <- RMSE(predicted_ratings, new_validation$rating)
#store result
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Movie Effect Model",
    RMSE = model_1_rmse ))

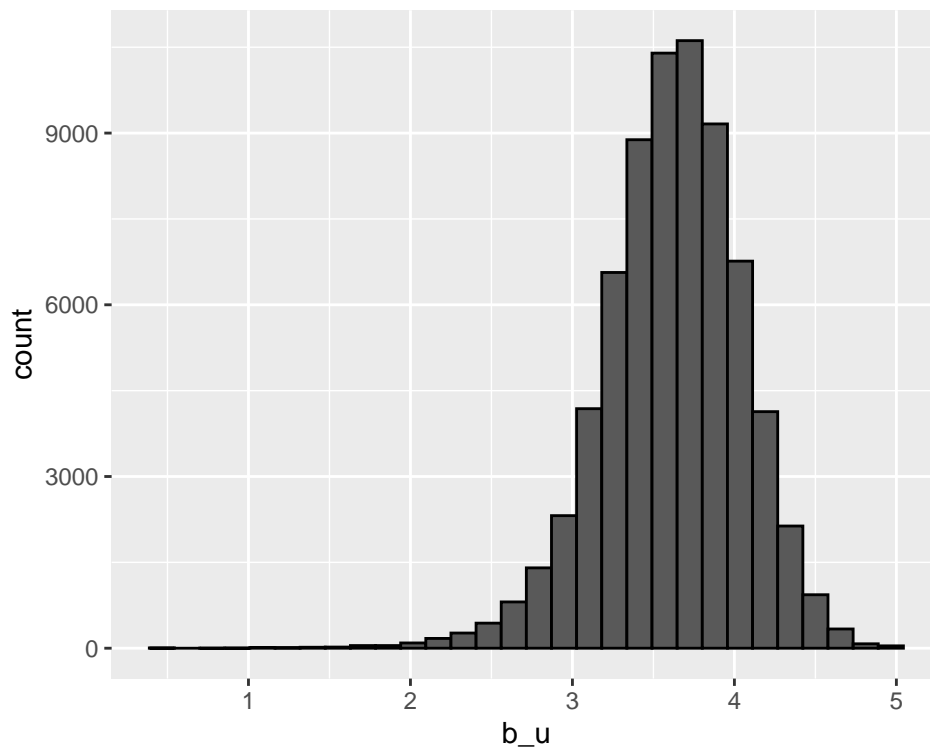
#show result
rmse_results %>% knitr::kable()
```

| method | RMSE |
|-------------------------|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |

Our RMSE is still pretty high, let's see if we can do better.

2.2.3 Movie + User Effect Model

If we compute the average rating for user u for those that have rated 100 or more movies, we can observe some variability too here:



Let's incorporate this finding and see if we can improve our model by adding a user specific effect b_u to represent average rating given by user u :

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

```
#calculating b_u for new_edx
user_avgs <- new_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#calculating b_u for new_validation
predicted_ratings <- new_validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

#calculating the RMSE for this model
model_2_rmse <- RMSE(predicted_ratings, new_validation$rating)

#Storing the RMSE for this model
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Movie + User Effects Model",
    RMSE = model_2_rmse ))
```

| method | RMSE |
|----------------------------|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effects Model | 0.8653488 |

We can observe that we have significantly improve our model and reduced our RMSE to 0.8653488.

2.2.4 Movie + User + Year_Movie Effect Model

Earlier we saw that the year a movie was released could influence the rating of a movie. Let's add it to the function where b_{iym} represents the average rating per year when movie i was released:

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

```
#calculating b_iym for new_edx
year_movie_avgs <- new_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(year_movie) %>%
  summarize(b_iym = mean(rating - mu - b_i- b_u))

#calculating b_iym for new_validation
predicted_ratings <- new_validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_movie_avgs, by='year_movie') %>%
  mutate(pred = mu + b_i + b_u +b_iym) %>%
  .$pred

#calculating the RMSE for this model
model_3_rmse <- RMSE(predicted_ratings, new_validation$rating)

#Storing the RMSE for this model
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Year_Movie Effects Model",
                                     RMSE = model_3_rmse ))
```

Our model has improved but not significantly.

| method | RMSE |
|---|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effects Model | 0.8653488 |
| Movie + User + Year_Movie Effects Model | 0.8650043 |

2.2.5 Movie + User + Year_Movie + Year rated Effects Model

Earlier we saw that the year a movie was rated could influence the rating of a movie. Let's add it to the function where b_{iuy} represents the average rating per year when movie i was rated by user u :

$$Y_{u,i} = \mu + b_i + b_u + b_{iym} + b_{iuy} + \epsilon_{u,i}$$

```
#calculating b_iuy for new_edx
year_avgs <- new_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_movie_avgs, by='year_movie') %>%
  group_by(year) %>%
  summarize(b_iuy = mean(rating - mu - b_i- b_u-b_iym))
```



```

#calculating b_iuy for new_validation
predicted_ratings <- new_validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_movie_avgs, by='year_movie') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_i + b_u +b_iym +b_iuy) %>%
  .$pred

#calculating the RMSE for this model
model_4_rmse <- RMSE(predicted_ratings, new_validation$rating)

#Storing the RMSE for this model
rmse_results <- bind_rows(rmse_results,
  data_frame(method = "Movie + User + Year_Movie
    + Year rated Effects Model",
    RMSE = model_4_rmse ))

```

Our model has improved but not significantly.

| method | RMSE |
|---|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effects Model | 0.8653488 |
| Movie + User + Year_Movie Effects Model | 0.8650043 |
| Movie + User + Year_Movie + Year rated Effects Model | 0.8649285 |

2.2.6 Movie + User + Year_Movie + Year rated + Genres Effects Model

Earlier we saw that the genres of a movie could influence the rating of a movie. Let's add it to the function where b_g represents the average rating per genres:

$$Y_{u,i} = \mu + b_i + b_u + b_{iym} + b_{iuy} + b_g + \epsilon_{u,i}$$

```

#calculating b_g for new_edx
genre_avgs <- new_edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_movie_avgs, by='year_movie') %>%
  left_join(year_avgs, by='year') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_iym - b_iuy))

#calculating b_g for new_validation
predicted_ratings <- new_validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_movie_avgs, by='year_movie') %>%
  left_join(year_avgs, by='year') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u +b_iym +b_iuy +b_g) %>%

```

```

.$pred

#calculating the RMSE for this model
model_5_rmse <- RMSE(predicted_ratings, new_validation$rating)
#Storing the RMSE for this model
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Year_Movie +
                                      Year rated + Genres Effects Model",
                                      RMSE = model_5_rmse ))

```

Our model has improved but not significantly.

| method | RMSE |
|--|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effects Model | 0.8653488 |
| Movie + User + Year_Movie Effects Model | 0.8650043 |
| Movie + User + Year_Movie + Year rated Effects Model | 0.8649285 |
| Movie + User + Year_Movie + Year rated + Genres Effects Model | 0.8645943 |

2.2.7 Regularized Movie + User Effect Model

We know that some movies are rated more or less than others and some users rate more or less often than others. In our last models, this was not incorporated and could strongly overestimate or underestimate a rating of a movie by user u . We will regularization technique to penalise that aspect. For this model we will try to minimize the following equation :

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

We will use cross validation to find λ , the tuning parameter using the following code:

```

#define the different lambdas
lambdas <- seq(0, 10, 0.25)

#cross validation algorithm
rmse <- sapply(lambdas, function(l){
  #average rating
  mu <- mean(new_edx$rating)

  #finding b_i for new_edx using l
  b_i <- new_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+1))

  #finding b_u for new_edx using l
  b_u <- new_edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+1))

  #Calculating the RMSE with l

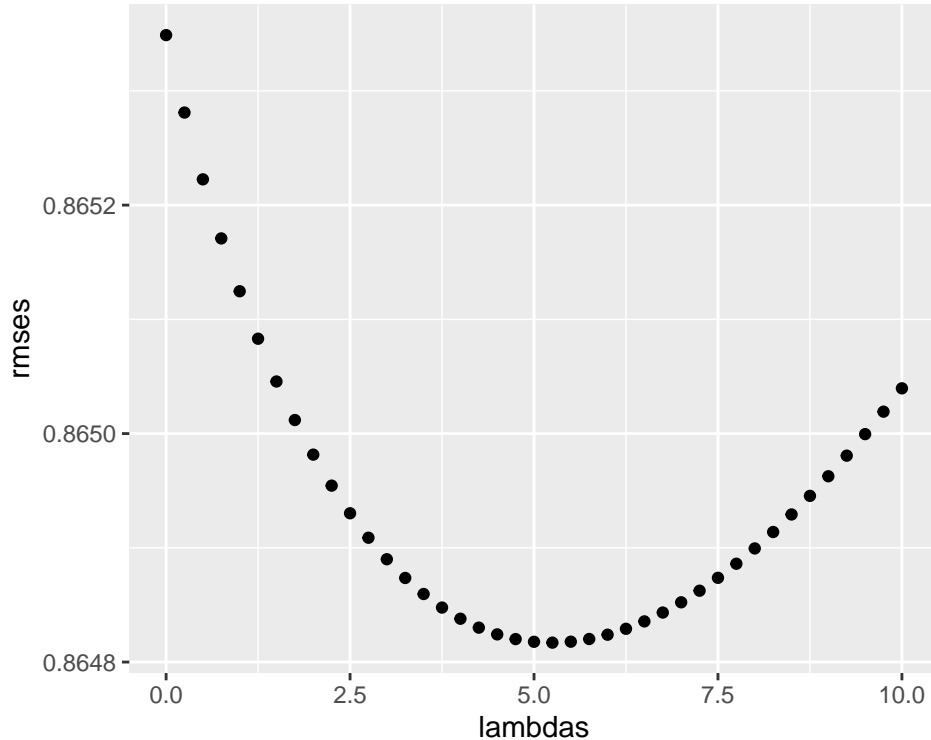
```

```

predicted_ratings <-
  new_validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
return(RMSE(predicted_ratings, new_validation$rating))
})

```

Here is a graph of all the different lambdas for our regularized model:



```

#find minimum lambda
lambda <- lambdas[which.min(rmses)]
lambda

## [1] 5.25

#pick the lambda that minimises the RMSE
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Regularized Movie + User Effect Model",
    RMSE = min(rmses)))

```

We can observe that our Regularized Movie + User Effect Model is more effective than our simple Movie + User Effects Model.

| method | RMSE |
|--|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effects Model | 0.8653488 |
| Movie + User + Year_Movie Effects Model | 0.8650043 |
| Movie + User + Year_Movie + Year rated Effects Model | 0.8649285 |

| method | RMSE |
|--|-----------|
| Movie + User + Year_Movie + Year rated + Genres Effects Model | 0.8645943 |
| Regularized Movie + User Effect Model | 0.8648170 |

3 Results

From our RMSE table:

| method | RMSE |
|--|-----------|
| Just the average rating | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effects Model | 0.8653488 |
| Movie + User + Year_Movie Effects Model | 0.8650043 |
| Movie + User + Year_Movie + Year rated Effects Model | 0.8649285 |
| Movie + User + Year_Movie + Year rated + Genres Effects Model | 0.8645943 |
| Regularized Movie + User Effect Model | 0.8648170 |

We can observe that the most effective model is the Movie + User + Year_Movie + Year rated + Genres Effects Model with an RMSE equal to 0.8645943 which is below our objective of 0.86490.

4 Conclusion

In conclusion, our most optimal model to predict the rating of a movie is the Movie + User + Year_Movie + Year rated + Genres Effects Model. We could use regularization on this model and add more effects to this model but this would make the calculation too long. We can observe that the more effects we add the smaller the RMSE becomes. However, the RMSE does not decrease more significantly and stays around approximately 0.86, which means that we need either more data or new methods if we want our RMSE to drop significantly. Hopefully, soon we will be able to use quantum computers which will allow us data scientist to create better models with more data.