Capstone Project - Movie Recommendation System

Francois Herman

7/26/2020

Introduction

Nowadays movies are at the reach of everyone, whether it's going to the movie theatre or watching them on Netflix. Thousands of movies are produced each year, some are judged "good", others "bad": we all have different opinions. But what if a computer could predict your rating of a movie via machine learning. This is the goal of this Capstone Project. This project is based on a real-life Netflix case, when in 2006, the company held an open-source contest to improve their recommendation system.

We are going to study 10 million movie ratings from the 'movielens' dataset, and try and predict their ratings, using different factors, such as the movie itself, the user grading the movie, as well as the year it was produced in. The difficulty of the task comes from the different biases that are encountered when looking at the data: people don't rate movies the same way, certain movies are rated better than others, people mostly watch the most famous and therefore best rated "old" movies … These are only a few examples to illustrate the complexity of predicting a rating of a certain individual on a certain movie.

We will first explain how the dataset was created, explore the data, run different data analysis models, and finally conclude with the results from the best model.

Data Creation

The data we will study is from the 'movielens' dataset. We will not be using the entire dataset, but instead, a smaller portion containing 10 million ratings: https://grouplens.org/datasets/movielens/10m/ (https://grouplens.org/datasets/movielens/10m/)

The code to create the data was given in the instructions of the course.

```r
##########################################################
# 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.us.r-project.org")
if(!require(data.table)) install.packages("data.table", 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 <- 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 4.0 or later
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

Two datasets are then created:

1. A training dataset : “edx” containing 90% of the data.
2. A test dataset : “validation” containing 10% of the data.

We will therefore train our data on the “edx” dataset and then test our different models on the “validation” dataset.

```
# 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 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)
```

We then open different libraries that might be useful for our project.

```
library(dslabs)
library(tidyverse)
library(lubridate)
```

Finally we will convert the variable ‘timestamp’ into a comprehensive date/time format, and create a new variable ‘year’ by extracting it from the ‘title’ variable where it was initially set:

```
# Changing the format of the timestamp
edx <- mutate(edx, date = as_datetime(timestamp))
validation <- mutate(validation, date = as_datetime(timestamp))

#Extracting the year to be its own column
edx <- edx %>% mutate(year=as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(year=as.numeric(str_sub(title,-5,-2)))
```

Data Exploration

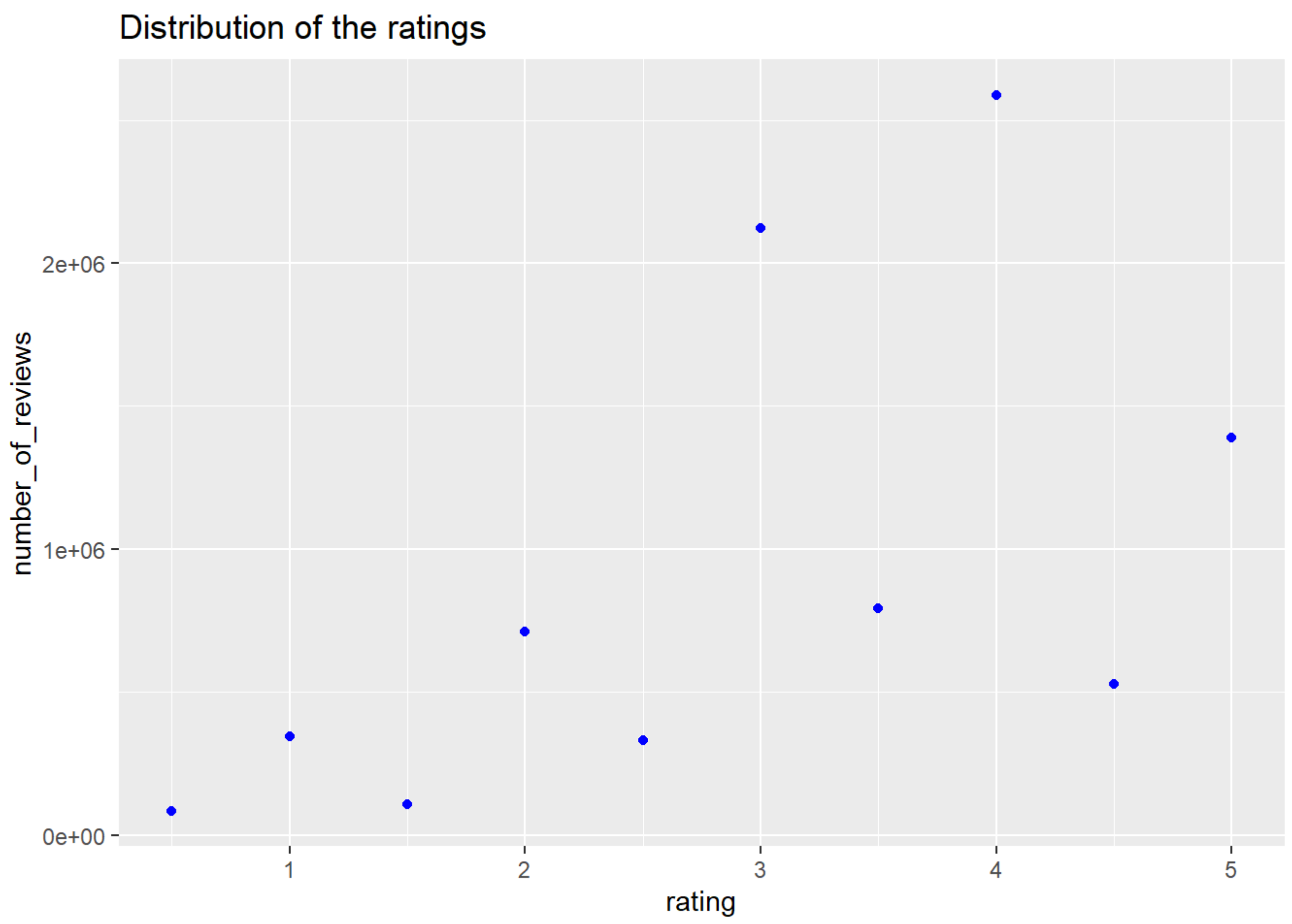
Before starting to build the recommendation system, we need to study and understand the data we are working with. Here is a brief summary of the ‘edx’ dataset and its variables:

```
# Summary of the 'edx' dataset
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               date
##  Length:9000055     Length:9000055     Min.   :1995-01-09 11:46:49
##  Class :character   Class :character   1st Qu.:2000-01-01 23:11:23
##  Mode  :character   Mode  :character   Median :2002-10-24 21:11:58
##                                        Mean   :2002-09-21 13:45:07
##                                        3rd Qu.:2005-09-15 02:21:21
##                                        Max.   :2009-01-05 05:02:16
##       year
##  Min.   :1915
##  1st Qu.:1987
##  Median :1994
##  Mean   :1990
##  3rd Qu.:1998
##  Max.   :2008
```

Here is a deeper understanding of the distribution of the movie ratings in the dataset:

```
#Distribution of the movie ratings
edx%>%group_by(rating)%>%
  summarise(number_of_reviews=n())%>%
  ggplot(aes(rating, number_of_reviews))+
  geom_point(color= "blue")+
  ggtitle("Distribution of the ratings")
```



There is a total of 9 million ratings in the ‘edx’ dataset. However, here are the numbers of unique movies being rated, and unique users rating movies:

```
# Number of unique movies in the 'edx' dataset
edx %>%
  select(movieId)%>%
  unique()%>%
  nrow()
```

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

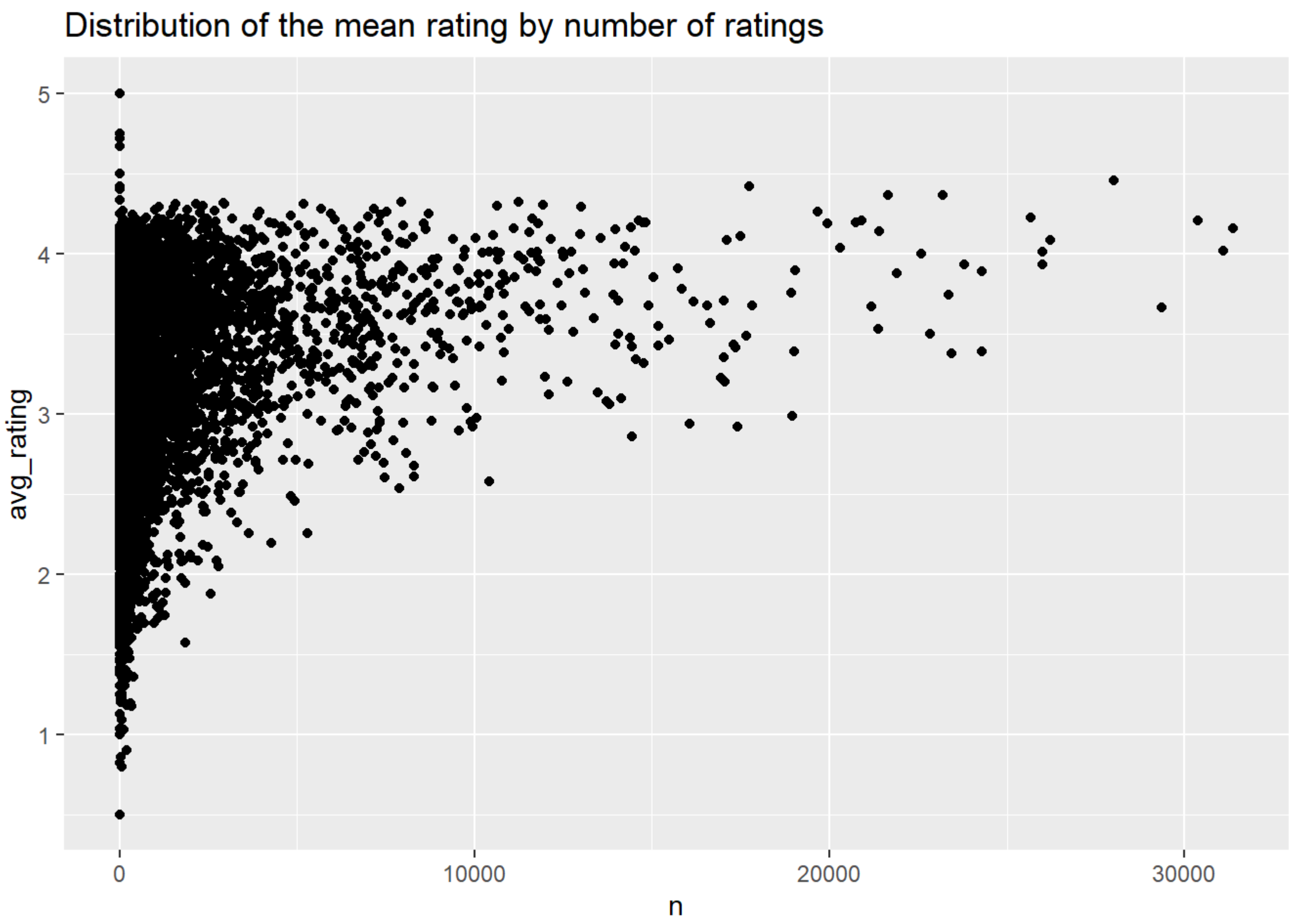
```
#Number of unique users in the 'edx' dataset
edx %>%
  select(userId)%>%
  unique()%>%
  nrow()
```

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

After looking at the basic statistics of our dataset, we are going to dive deeper in the different variables and look at the different issues we are going to encounter while doing our data analysis.

We are going to start off by looking at the distribution of the mean rating of a movie per number of ratings it has received :

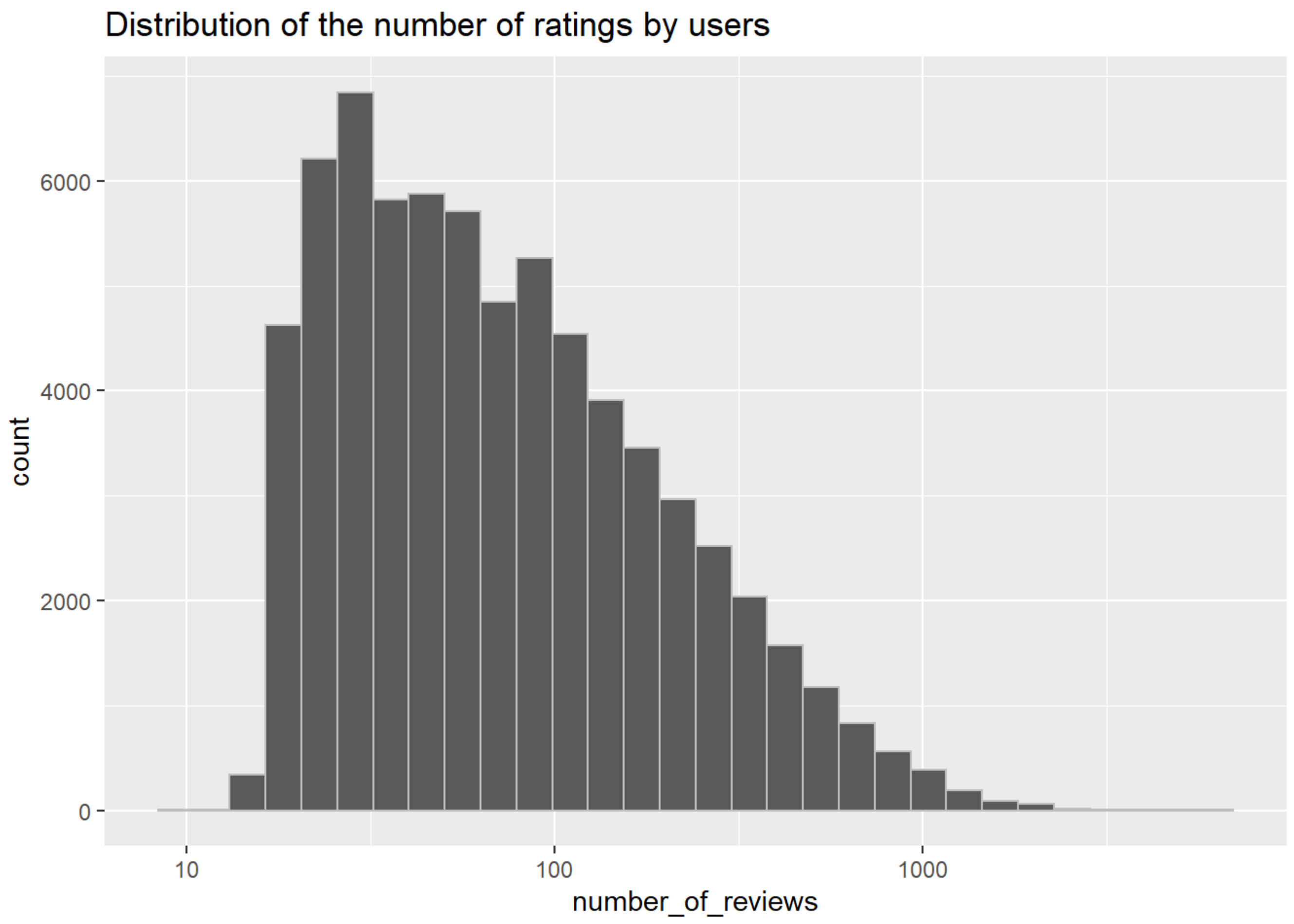
```
# Showing the effect of the number of ratings on the average rating of a movie
edx%>%group_by(movieId)%>%
  summarise(n=n(), avg_rating=mean(rating))%>%
  ggplot(aes(n,avg_rating))+
  geom_point()+
  ggtitle("Distribution of the mean rating by number of ratings")
```



With this plot we can see that the distribution of famous movies (ones that have received many ratings) is stable above the mean rating. However, this is not the case for movies that have not been rated many times. Some have mean ratings of 0.5 while others have mean ratings of 5.0. This is due to the fact that they have only been rated once. These should not have the same weight as other movies which have been rated a lot more.

The second variable we are going to look at is the distribution of number of ratings per users:

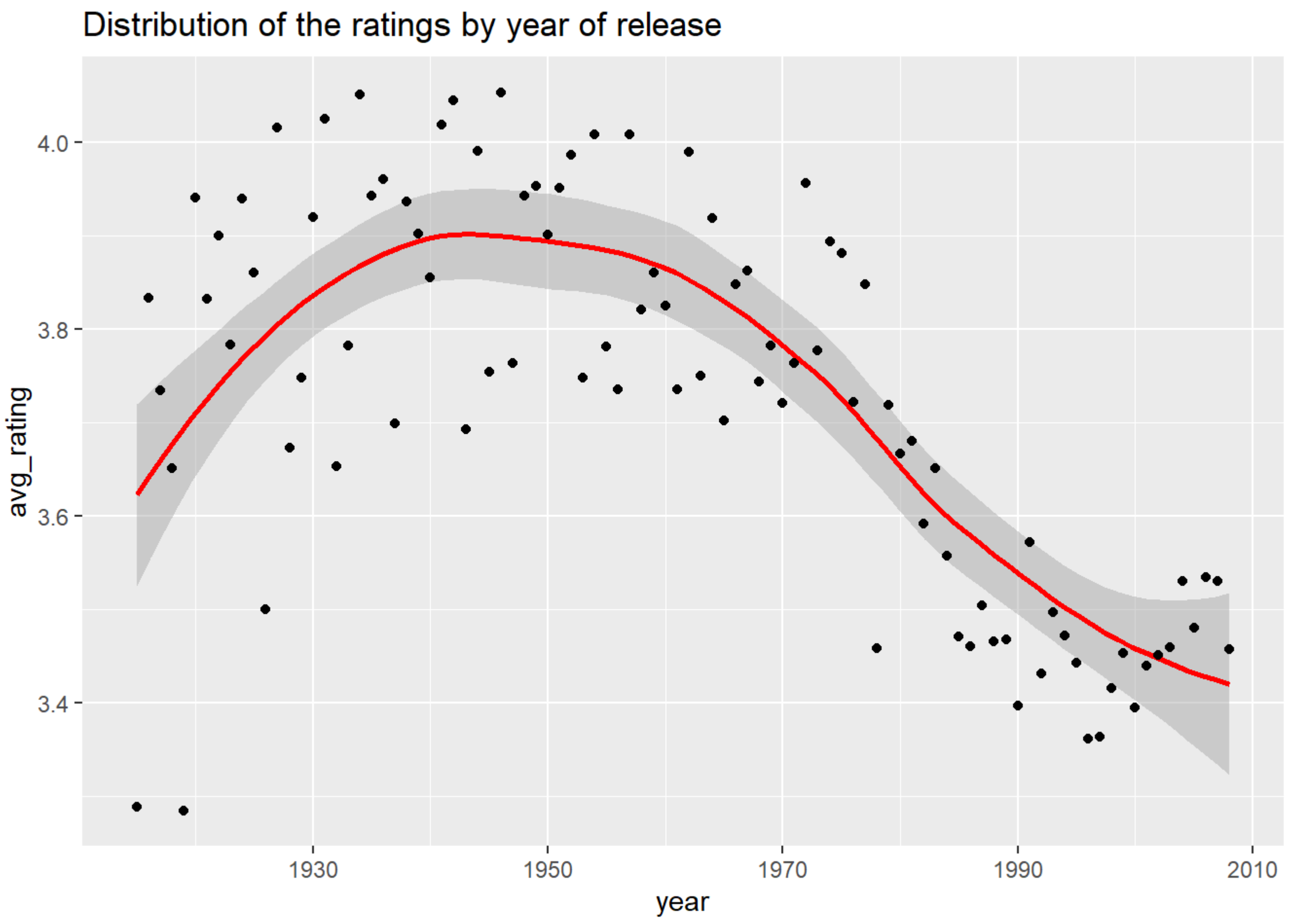
```
# Showing the number of ratings per user
edx%>%group_by(userId)%>%
  summarise(number_of_reviews=n())%>%
  ggplot(aes(number_of_reviews))+
  geom_histogram(color= "grey")+
  scale_x_log10()+
  ggtitle("Distribution of the number of ratings by users")
```



As we can clearly see from the histogram, some users have rated thousands of movies, while others have rated only a few dozens. We will have to keep this in mind when doing further analysis.

Finally, the third variable we are going to take a look at is the 'year' variable that we have previously extracted from the 'title' variable :

```
# Showing the effect of the year on the average rating of a movie
edx%>%group_by(year)%>%
  summarise(avg_rating=mean(rating))%>%
  ggplot(aes(year,avg_rating))+
  geom_smooth(color="red")+
  geom_point()+
  ggtitle("Distribution of the ratings by year of release")
```



As we can see older movies have a better average rating than recent movies. One explanation could be that people only watch ‘old’ movies because they know they are good movies, on the other hand, all the newer movies are watched and rated because there is no track record of their success.

All of the issues explained in this sections, will be important ideas we will have to keep in mind when conducting our analyses.

Methods / Analysis

The goal of the project is to be able to predict as accuratly as possible the rating of a certain movie by a certain user. Throughout this section we will utilize different models to try and reach our goal. To measure the success of our different models, we will compute their Root-mean-square Deviation (RMSE), which is the average distance between our prediction and the actual rating. All of the studied models are trained on the ‘edx’ dataset and tested on the ‘validation’ dataset.

Overall Mean Model

The first method we are going to use is the “Overall mean model”. We are going to predict the overall mean rating to every single rating and see what is the RMSE. This is the most basic model, but a good starting point in terms of RMSE.

```
#Methods/Analysis
#Overall Mean Model
overall <- mean(edx$rating)
overall_mean_rmse <- RMSE(validation$rating, overall)
```

model	RMSE
Giving the overall mean rating to all ratings	1.061202

Movie Effect Model

Every movie is rated differently: this is the movie effect. We are going to compute this effect by taking the difference between the rating and the overall mean rating. The average for all ratings gives us the movie bias b_i.

```
# Movie effect Model

movie_norm_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating-overall))

pred_ratings_movie <- validation %>%
  left_join(movie_norm_avgs, by="movieId")%>%
  mutate(prediction = overall+b_i)

movie_model_rmse <- RMSE(validation$rating, pred_ratings_movie$prediction)
```

model	RMSE
Giving the overall mean rating to all ratings	1.0612018
Movie Effect Model	0.9439087

We can see an improvement of t=our model’s RMSE when adding the movie effect.

Movie and User Effects Model

We are going to continue building on our previous model by taking into account the User Effect: not all users grade the same way. Some are really nice, others really harsh.

The way the User Effect is computed, is by substracting the overall mean rating as well as movie bias “b_i” from the actual rating. Taking the avarage for all ratings, gives us the user bias b_u.

```
# Movie and User effect Models

user_norm_avgs <- edx %>%
  left_join(movie_norm_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-overall-b_i))

pred_ratings_user <- validation %>%
  left_join(movie_norm_avgs, by="movieId")%>%
  left_join(user_norm_avgs, by="userId")%>%
  mutate(prediction=overall + b_i + b_u)

user_model_rmse <- RMSE(validation$rating, pred_ratings_user$prediction)
```

model	RMSE
Giving the overall mean rating to all ratings	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488

Our RMSE has improved when taking into account the User Effect.

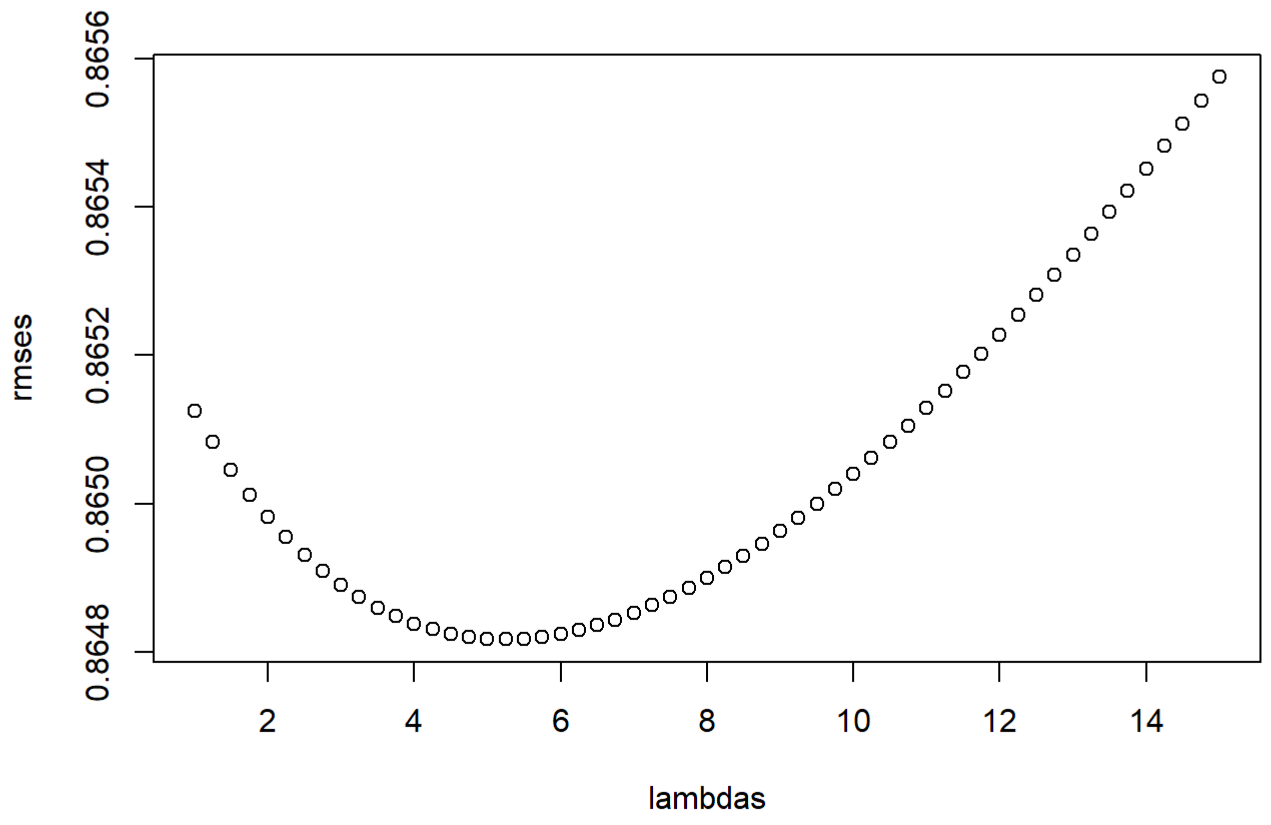
Regularization of the Movie and User Effects

An important issue we have not tackled yet is the weight each movie and each user should get when conducting our analyses. As we have seen when exploring the data, some movies have been graded more times than others and some users have graded more movies than others.

Some movies have only been rated once or twice, while some users only have a couple dozen ratings. These are noisy estimates that should not hold as much weight as other estimates. Because the RMSE is sensitive to these noisy estimates we must put a penalty term: lambda.

As you can see in the code below the penalty term lambda is incorporated within the b_i and b_u calculation formulas. We are going to find the best lambda by running the code below:

```
#Regularization of Movie and User Effects

lambdas <- seq(1, 15, 0.25)

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

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - overall)/(n()+l))

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - overall)/(n()+l))

  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = overall + b_i + b_u) %>%
    .$pred
  return(RMSE(validation$rating, predicted_ratings))
})
plot(lambdas, rmses)
```



```
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

```
## [1] 5.25
```

Now that we have found the penalty term that minimizes our RMSE, we can run our prediction model using this lambda:

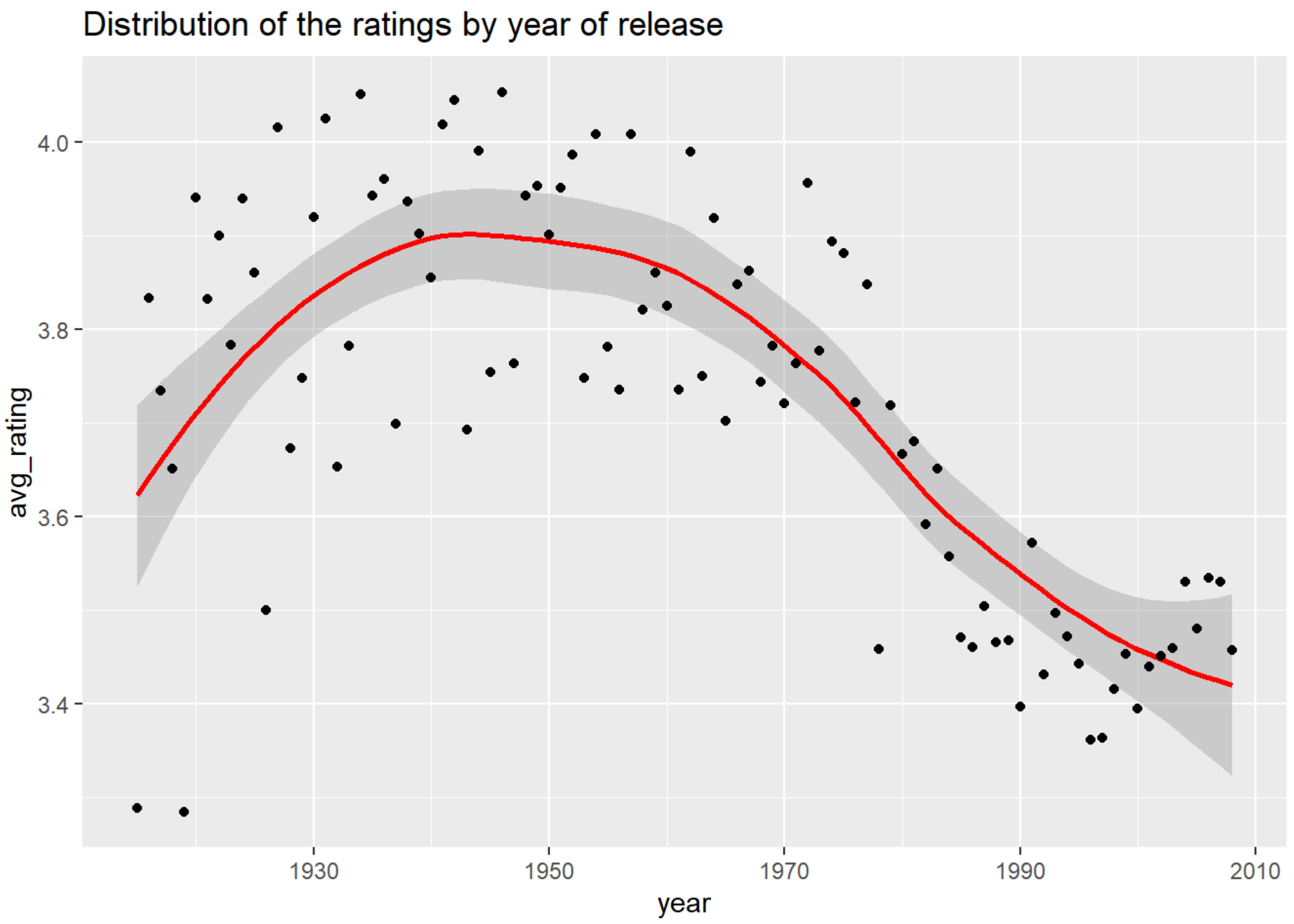
```
#Compute predictions with regularization for movie and user bias
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - overall)/(n()+lambda))

user_reg_avgs <- edx %>%
  left_join(movie_reg_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - overall)/(n()+lambda))

predicted_ratings_reg <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by = "userId") %>%
  mutate(pred = overall + b_i + b_u) %>%
  .$pred

reg_model_rmse <-RMSE(validation$rating, predicted_ratings_reg)
```

model	RMSE
Giving the overall mean rating to all ratings	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170

Regularization has improved our RMSE.

Regularization with Movie, User and Year effects

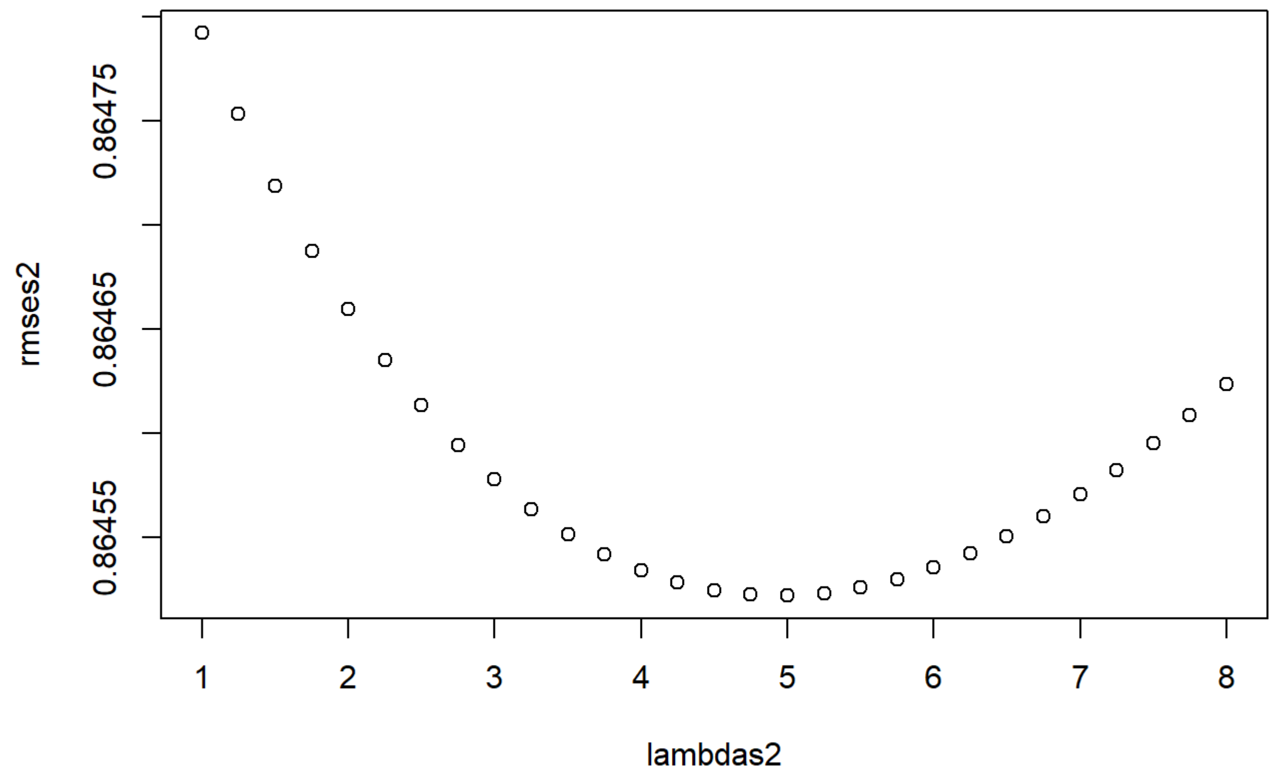
Finally to improve our improve we are going to take into account the year effect.



As we have seen with this plot, older movies tend to get higher ratings than newer movies. We are going to account for this by creating a year bias b_y.

We will incorporate it within our regularized model, and therefore get a different penalty term for this model: ‘lambda_final’

```
#Regularization with Movie, User and Year effects
lambdas2 <- seq(1, 8, 0.25)

rmses2 <- sapply(lambdas2, function(l){

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - overall)/(n()+l))

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - overall)/(n()+l))

  b_y <- edx%>%
    left_join(b_i, by="movieId")%>%
    left_join(b_u, by="userId")%>%
    group_by(year)%>%
    summarize(b_y = sum(rating - b_i - b_u - overall)/(n()+l))

  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year")%>%
    mutate(pred = overall + b_i + b_u + b_y) %>%
    .$pred
  return(RMSE(validation$rating, predicted_ratings))
})
plot(lambdas2, rmses2)
```



```
lambda_final <- lambdas2[which.min(rmses2)]
print(lambda_final)
```

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

Now that we have our new lambda we will rerun our prediction model with the year bias in it.

```
#Calculating Final RMSE with regularization

movie_reg2_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - overall)/(n()+lambda_final))

user_reg2_avgs <- edx %>%
  left_join(movie_reg2_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - overall)/(n()+lambda_final))

year_reg2_avgs <- edx%>%
  left_join(movie_reg2_avgs, by = "movieId") %>%
  left_join(user_reg2_avgs, by = "userId") %>%
  group_by(year)%>%
  summarize(b_y = sum(rating - b_i - b_u - overall)/(n()+lambda_final))

predicted_ratings_reg2 <- validation %>%
  left_join(movie_reg2_avgs, by = "movieId") %>%
  left_join(user_reg2_avgs, by = "userId") %>%
  left_join(year_reg2_avgs, by="year")%>%
  mutate(pred = overall + b_i + b_u + b_y) %>%
  .$pred

reg2_model_rmse <-RMSE(validation$rating, predicted_ratings_reg2)
```

```
## [1] 0.8645218
```

model	RMSE
Giving the overall mean rating to all ratings	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170
Regularized Movie, User and Year Effect Model	0.8645218

Our final RMSE is 0.8645218.

Results

After conducting our analyses, here are the multiple models we have built as well as their respective RMSE’s.

model	RMSE
Giving the overall mean rating to all ratings	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488

model	RMSE
Regularized Movie and User Effect Model	0.8648170
Regularized Movie, User and Year Effect Model	0.8645218

Our final model is the “Regularized Movie, User, and Year Effects Model”, and our final RMSE is 0.8645218.

Conclusion

Our final movie recommendation system takes into account each different movie, each different user as well as the year the movie was produced. Along with those variables we have used regularization to penalize noisy estimates that would increase our RMSE. Our final RMSE is 0.8645218.

Limitations: Predicting ratings is a really complex process that involves dozens of different variables, that would make our algorithm really complex. However, if we wanted to improve our RMSE even more the next variable we could have studied is the ‘genre’ variable as it does have an impact on movie ratings.