HarvardX Data Science Program

Movielens project

Do Quang Anh

2023-03-14

Contents

1 Overview

This project is related to the MovieLens Project of the HervardX: PH125.9x Data Science: Capstone course. The present report start with a general idea of the project and by representing its objectif.

Then the given dataset will be prepared and setup. An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict movie ratings until a final model. Results will be explained. Finally the report ends with some concluding remarks.

1.1 Introduction

Recommendation systems use ratings that users have given to items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give to a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

The same could be done for other items, as movies for instance in our case. Recommendation systems are one of the most used models in machine learning algorithms. In fact the success of Netflix is said to be based on its strong recommendation system. The Netflix prize (open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films), in fact, represent the high importance of algorithm for products recommendation system.

For this project we will focus on create a movie recommendation system using the 10M version of MovieLens dataset, collected by GroupLens Research.

1.2 Aim of the project

The aim in this project is to train a machine learning algorithm that predicts user ratings (from 0.5 to 5 stars) using the inputs of a provided subset (edx dataset provided by the staff) to predict movie ratings in a provided validation set.

The value used to evaluate algorithm performance is the Root Mean Square Error, or RMSE. RMSE is one of the most used measure of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers. Four models that will be developed will be compared using their resulting RMSE in order to assess their quality. The evaluation criteria for this algorithm is a RMSE expected to be lower than 0.8775. The function that computes the RMSE for vectors of ratings and their corresponding predictors will be the following:

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

Finally, the best resulting model will be used to predict the movie ratings.

1.3 Dataset

The MovieLens dataset is automatically downloaded

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

2 Methods/Analysis

2.1 Setting the Data

First the data is imported and partitioned. Normally, the code below would be used.

```r
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# 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")

library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

```r
# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

saveRDS(edx, file = "edx.rds")
saveRDS(final_holdout_test, file = "validation.rds")
```

```r
#Load the Data
edx <- readRDS("edx.rds", refhook = NULL)
final_holdout_test <- readRDS("validation.rds", refhook = NULL)
```

2.2 Dataset Dimenions

Our training set is edx and test set is validation. Together, there are 10,000,054 records

Dataset	Number of Rows	Number of Columns
edx	9,000,055	6
final_holdout_test	999,999	6

After generating codes provided in the project overview, we can see that the edX dataset is made of 6 features for a total of about 9,000,055 observations.The validation set which represents 10% of the 10M Movielens dataset contains the same features , but with a total of 999,999 occurences. we made sure that userId and movieId in edx set are also in validation set.

Each row represents a rating given by one user to one movie. The column "rating" is the outcome we want to predict, y. Taking into account both datasets, here are the features and their characteristics:

quantitative features

-userId : discrete, Unique ID for the user.

-movieId: discrete, Unique ID for the movie.

-timestamp : discrete , Date and time the rating was given.

qualitative features

-title: nominal , movie title (not unique)

-genres: nominal, genres associated with the movie.

outcome,y

-rating : continuous, a rating between 0 and 5 for the movie.

Check missing value in edx. There are no missing values in any column.

	x
userId	FALSE
movieId	FALSE
rating	FALSE
timestamp	FALSE
title	FALSE
genres	FALSE

A preview of the data structure is shown below from the first few rows in 'edx'.

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838,985,046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838,983,525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838,983,421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838,983,392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838,983,392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838,984,474	Flintstones, The (1994)	Children|Comedy|Fantasy

Rating is the dependent/target variable - the value we are tring to predict.

2.3 Data Classes

We can see that userId, movieId and rating are not factors, despite each having a smaller number of unique values. Furthermore, timestamp (the timestamp of the rating) is not useful as an integer.

	x
userId	integer
movieId	integer
rating	numeric
timestamp	integer
title	character
genres	character

2.4 Modifying data frames

```r
#Create new data base on final_holdout_test
validation_1 <- final_holdout_test
validation <- final_holdout_test %>% select(-rating)

#add a new column called "year" to two data frames
edx <- edx %>% mutate(year = as.numeric(str_sub(title,-5,-2)))
validation  <- validation %>% mutate(year = as.numeric(str_sub(title,-5,-2)))

#split edx into multiple rows based on the values in the "genres" column
split_genres_edx  <- edx  %>% separate_rows(genres, sep = "\\|")
genre_count <- split_genres_edx %>% group_by(genres) %>% summarise(number = n()) %>% arrange(desc(numbe
split_genres_validation <- validation  %>% mutate(year = as.numeric(str_sub(validation$title,-5,-2))) %
split_genres_validation_1 <- validation_1 %>% mutate(year = as.numeric(str_sub(validation_1$title,-5,-2
```

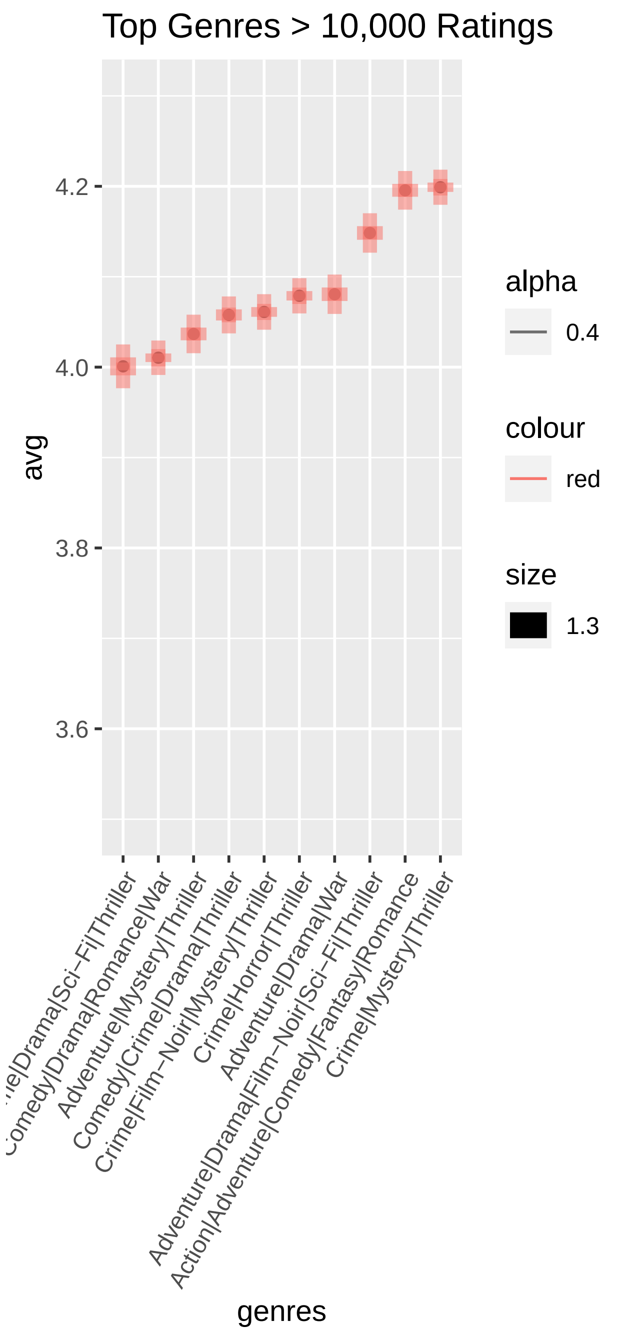
2.5 Genres

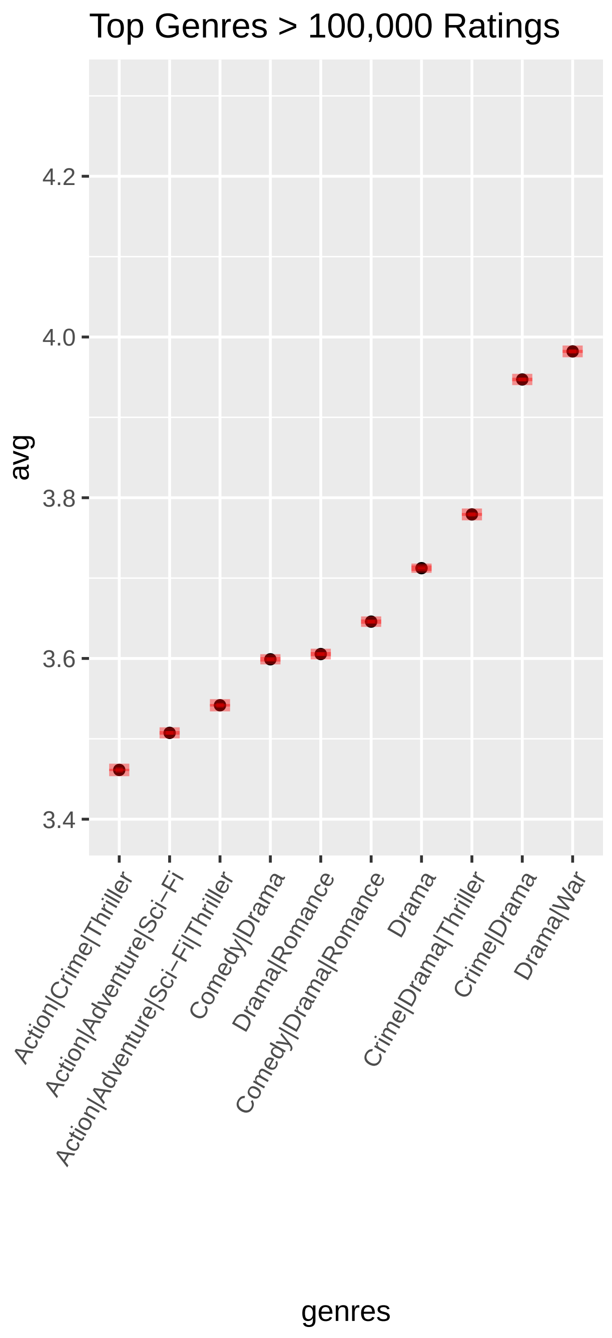
While there are only 20 unique genres, a film may be classified into multiple genres, up to seven at once. There are 797 of these unique combinations. Here are some of the biggest combinations.

	genres	genreCount
27166	Action|Adventure|Comedy|Drama|Fantasy|Horror|Sci-Fi|Thriller	7
754	Adventure|Animation|Children|Comedy|Crime|Fantasy|Mystery	6
32149	Adventure|Animation|Children|Comedy|Drama|Fantasy|Mystery	6
39662	Adventure|Animation|Children|Comedy|Fantasy|Musical|Romance	6

2.5.1 Best-Rated Genres

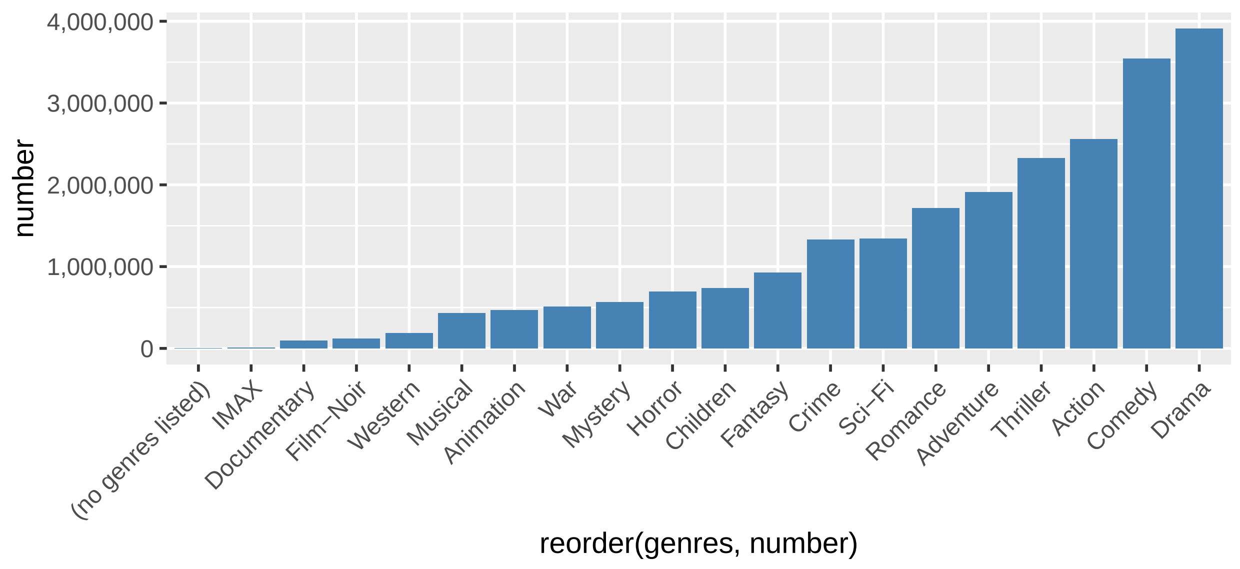
Which genre combinations have the best average ratings? We'll look only at the top 10 genres from all genres that have over **10,0000** ratings. Each genre will have an error bar with the standard error of the rating.



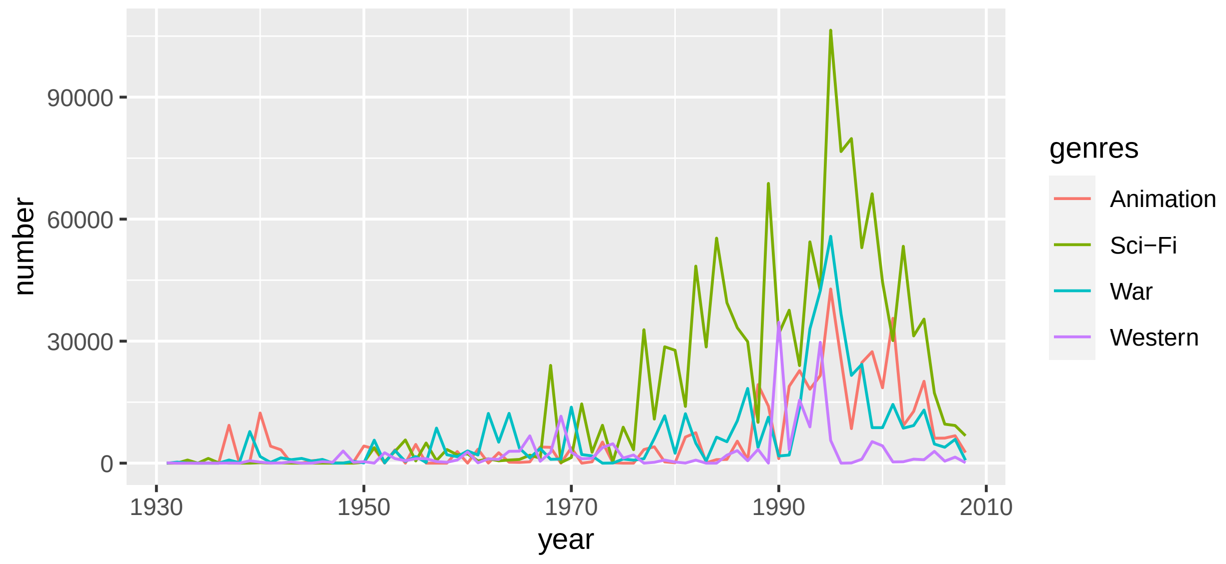


2.5.2 Genre prevalence

We may later decide to split these genre combinations up for better predictive value. Let's look at the individual prevalence of ratings each genre.

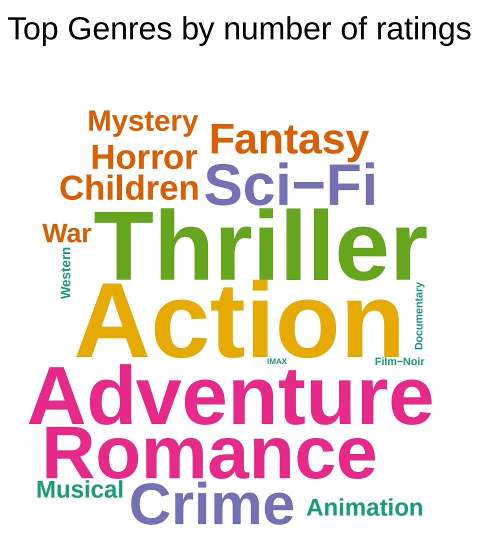


2.5.3 Genres popularity per year.



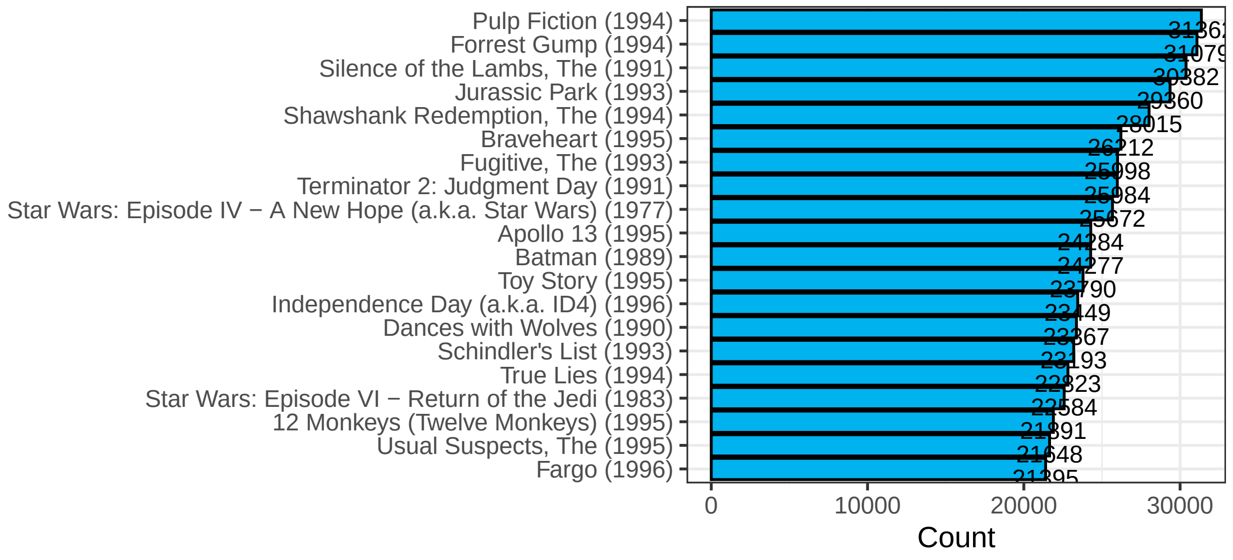
As expected, we can see some users are prolific, rating many movies. Likewise, some movies are very commonly rated.

2.5.4 creating a word cloud that shows the top genres by number of ratings



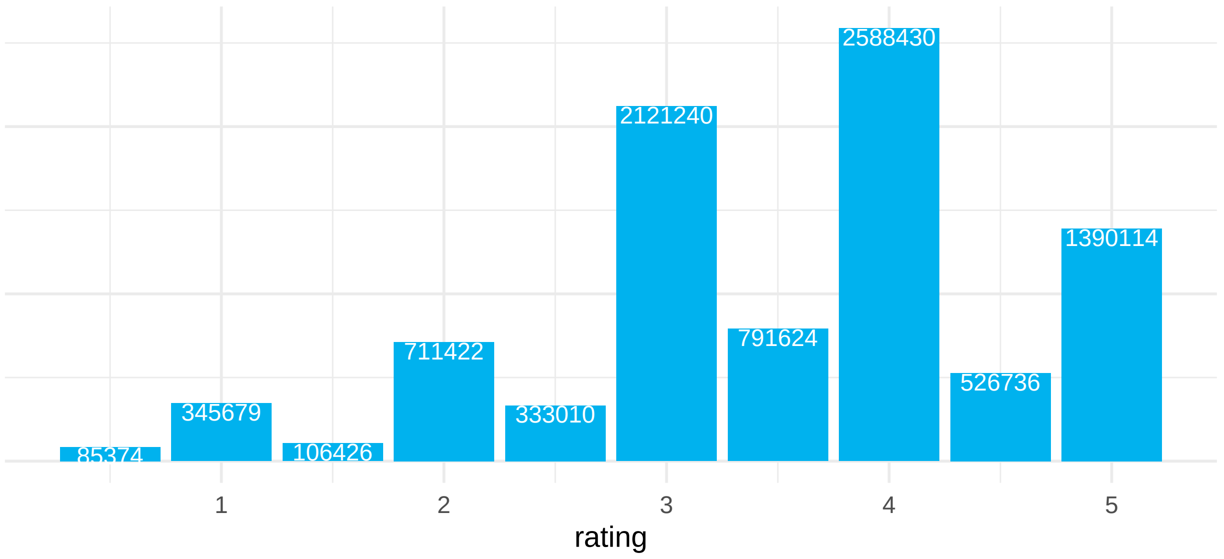
2.6 Reviews

Which films have the most popular(most rating)?

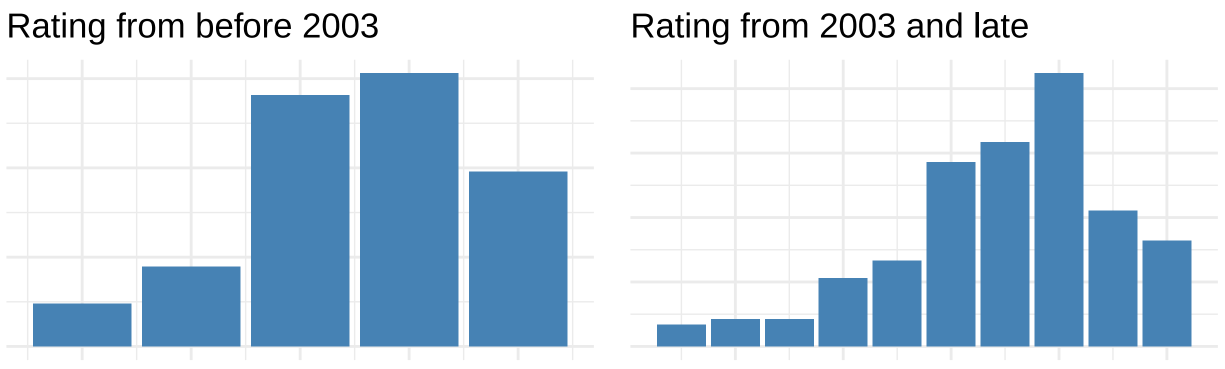


2.6.1 Rating Frequency

Half-star ratings are given out less frequently than full-stars. 4-star and 3-star ratings are the most common, followed by 5-star ratings.

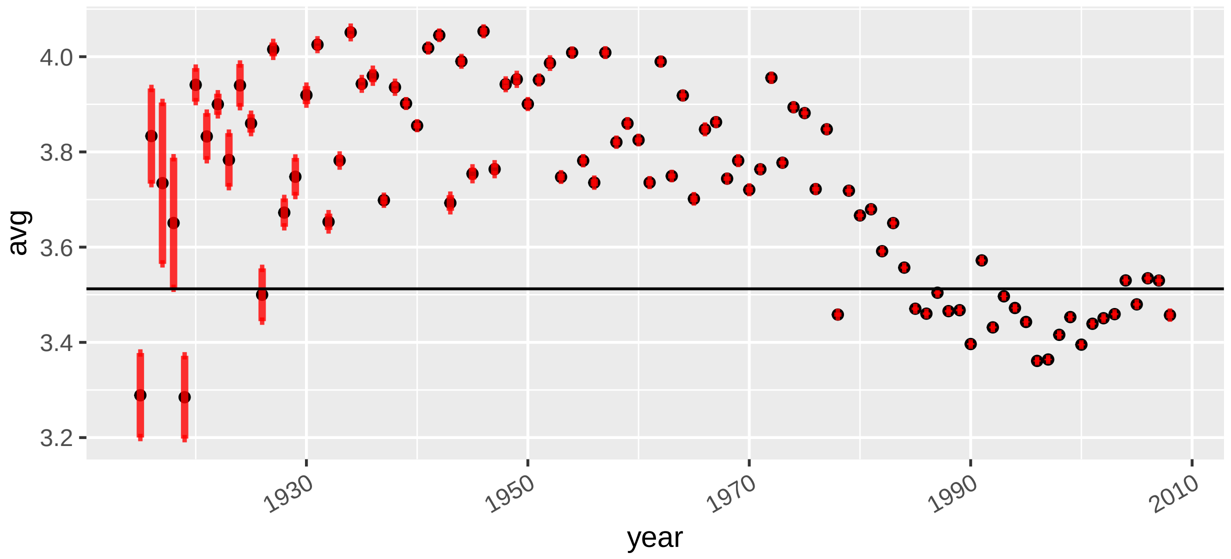


Part of the reason for fewer half-star ratings is likely that half-stars weren't used in the *Movielens* rating system prior to 2003. Here are the distributions of the ratings before and after half stars were introduced.

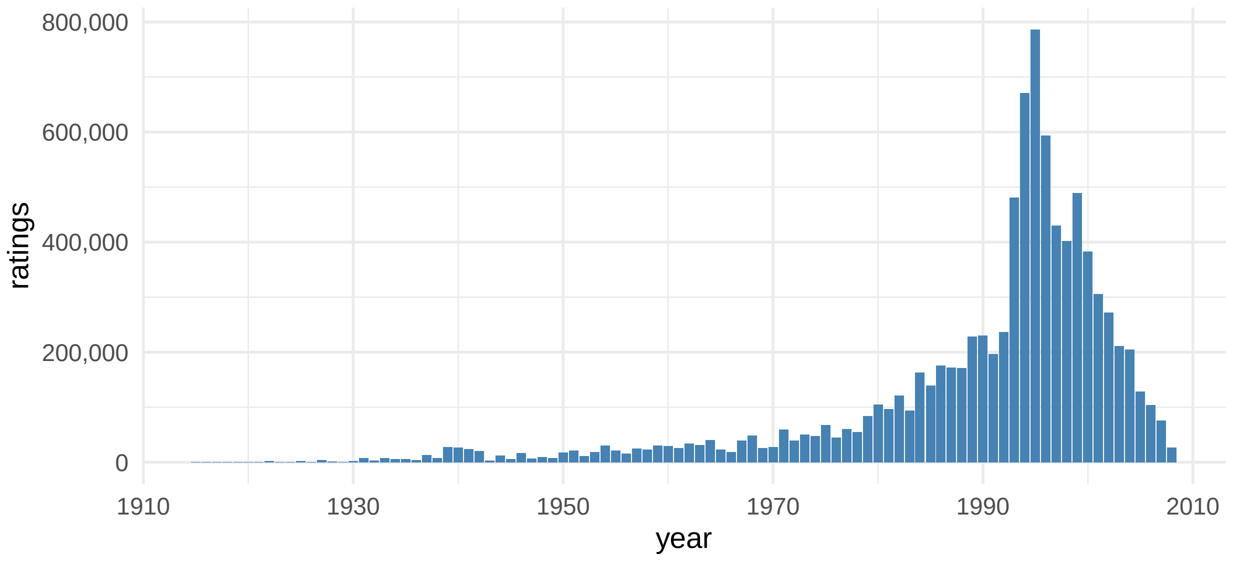


2.6.2 Average Rating By Year of Release

Earlier films have broader variability in their ratings, and films over 30 years old tend to be rated more highly on average. (The *Golden Age of Hollywood* is commonly held to have spanned from the late 1920s to the early 1960s).



Note: The chart above features standard error bars, and includes a reference line for the overall mean of all ratings More modern films have a tighter variability, and are generally located closer to the overall mean of 3.51. This is almost certainly due to the fact that older films have fewer ratings, and modern films have many more.


800,000
600,000
400,000
200,000
0
ratings
1910
1930
1950
1970
1990
2010
year

3 Results

The quality of the model will be assessed by the RMSE (the lower the better).

3.1 Summary Statistics

```r
edx %>%
summarize(
  First_Quart = quantile(rating,0.25),
  Mean_Rating = mean(rating) ,
  Median_Rating = median(rating),
  Third_Quart = quantile(rating,0.75)
  )  %>% niceKable
```

First_Quart	Mean_Rating	Median_Rating	Third_Quart
3	3.512465	4	4

3.2 Naive Prediction

We will make our first prediction, by simply using the mean of all ratings (3.51) as the estimate for every unknown rating. This will be the baseline against which model improvements will be evaluated.

```r
# Naive Model -- mean only
naive_rmse <- RMSE(validation_1$rating,overall_mean)
## Test results based on simple prediction
## Check results
rmse_results <- data_frame(method = "Using mean only", RMSE = naive_rmse)
```

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

```r
rmse_results %>%niceKable
```

method	RMSE
Using mean only	1.061202

```r
## Save prediction in data frame
```

3.2.1 Penalty Term- Movie Effect

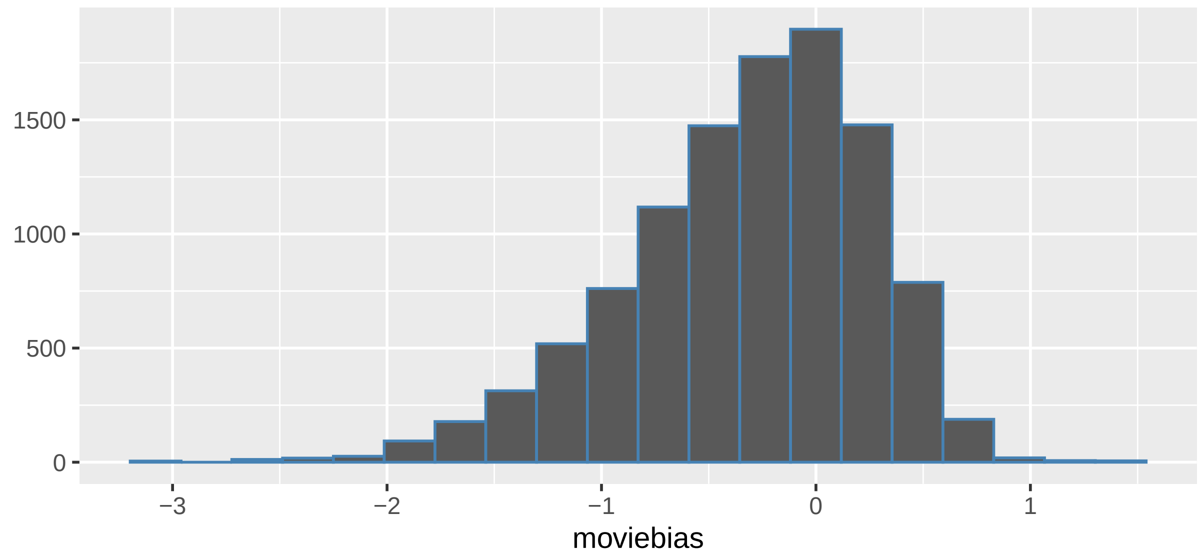
A bias term will be calculated for each movie to determine how much better/worse a given film is from the overall average, based on the average of all ratings for that film only. We will subtract the *overallMean* (‘r overall_mean’) from the *movieMean* for each film to determine the bias term, as below:

$$Bias_{movie} = Mean_{movie} - Mean_{overall}$$

More positively rated films will have a positive bias value, while negatively rated films will have a neagative bias value. Does this improve the model?

```
#Compute the mean rating for each movie
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(moviebias = mean(rating - overall_mean))
#Create a histogram of the movie biases
movie_avgs %>% qplot(moviebias, geom ="histogram", bins = 20, data = ., color = I("steelblue"))
```

```
## Warning: ‘qplot()‘ was deprecated in ggplot2 3.4.0.
```



```
# Use the movie biases to predict the ratings in the validation set
predicted_ratings_movie_norm <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = overall_mean + moviebias)

#Calculate the RMSE for the movie effect model
model_1_rmse <- RMSE(validation_1$rating,predicted_ratings_movie_norm$pred)

#Add the RMSE to the results table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results %>%niceKable
```

method	RMSE
Using mean only	1.0612018
Movie Effect Model	0.9439087

3.2.2 Penalty Term- User Effect

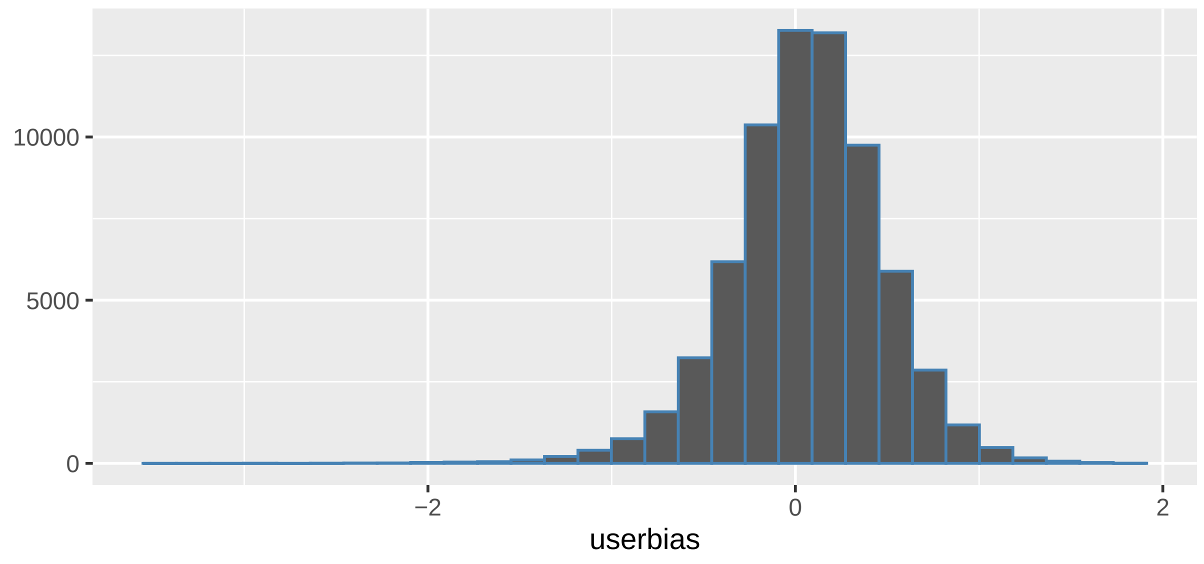
Similarly users can also affect the ratings either positivley(by giving higher ratings) or negatively (i.e.lower ratings). It is understood that users may have a tendency to rate movies higher or lower than the overall mean. Let’s add this into the model. First we’ll calculate the bias for each user:

$$Bias_{user} = Mean_{user} - Mean_{overall}$$

Then we'll combine the bias of a user, with the bias of a film and add both to the overall mean (3.51) for a combined bias rating for each unique combination of a user rating for a given film.

$UserMovieBias = overallMean + movieBias + userBias$

```
#Computes the average rating given by each user and the user bias of each user by subtracting the overa
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(userbias = mean(rating - overall_mean - moviebias))
#Generates a histogram of user biases.
user_avgs %>% qplot(userbias, geom ="histogram", bins = 30, data = ., color = I("steelblue"))
```



```
# Use test set,join movie averages & user averages
# Prediction equals the mean with user effect user bias & movie effect movies bias
predicted_ratings_user_norm <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = overall_mean + moviebias + userbias)

# test and save rmse results
model_2_rmse <- RMSE(validation_1$rating,predicted_ratings_user_norm$pred)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and User Effect Model",
                                     RMSE = model_2_rmse ))
#Displays the table of RMSE results
rmse_results  %>% niceKable
```

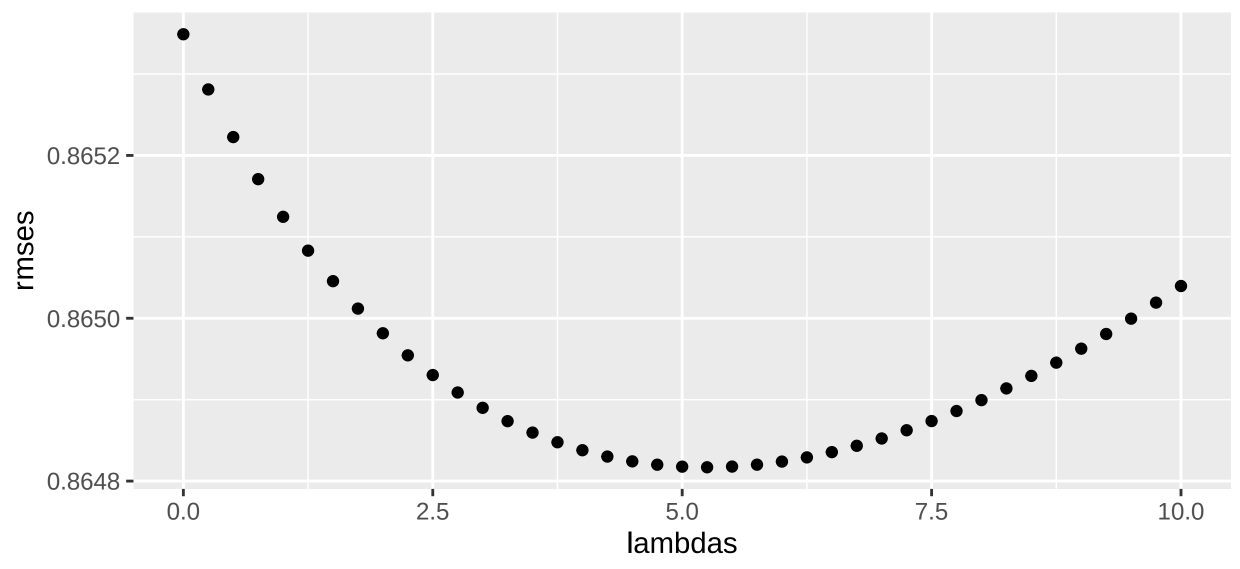
3.2.3 Regularized Movie and User Effect Model

This model implements the concept of regularization to account for the effect of low ratings' numbers for movies and users. The previous sections demonstrated that few movies were rated only once and that some

method	RMSE
Using mean only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488

users only rated few movies.Hence this can strongly influence the prediction. Regularization is a method used to reduce the effect of overfitting.

```r
# lambda is a tuning parameter
# Use cross-validation to choose it.
lambdas <- seq(0, 10, 0.25)
# For each lambda, compute the regularized estimates of the movie bias and user bias, followed by ratin
# note:the below code could take some time
rmses <- sapply(lambdas, function(l){

  overall_mean  <- mean(edx$rating)
  # Compute the regularized estimates of the movie bias.
  reg_movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(regmoviebias = sum(rating - overall_mean )/(n()+l))

  # Compute the regularized estimates of the user bias.
  reg_user_avgs <- edx %>%
    left_join(reg_movie_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(reguserbias = sum(rating - regmoviebias - overall_mean )/(n()+l))

  # Predict ratings using the regularized estimates of the movie bias and user bias.
  predicted_ratings <- validation %>%
    left_join(reg_movie_avgs, by = "movieId") %>%
    left_join(reg_user_avgs, by = "userId") %>%
    mutate(pred = overall_mean + regmoviebias + reguserbias) %>%
    .$pred

  return(RMSE(validation_1$rating,predicted_ratings))
})
# Plot rmses vs lambdas to select the optimal lambda.
qplot(lambdas, rmses)
```



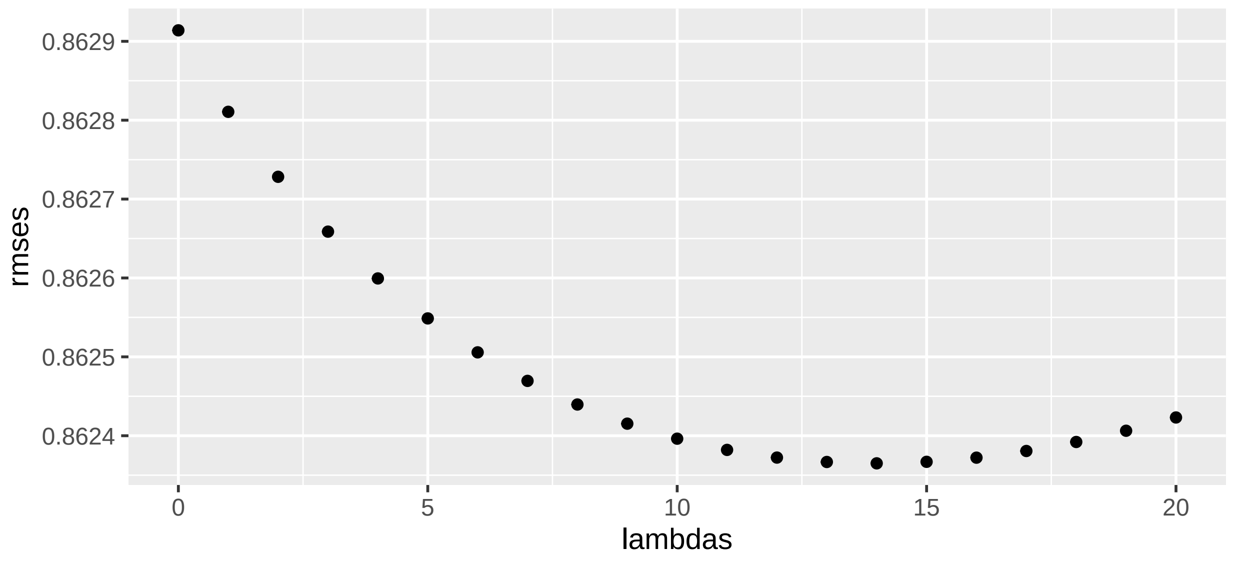
```
# Select the optimal lambda.
lambda <- lambdas[which.min(rmses)]

# Compute regularized estimates of movie bias using lambda
movie_avgs_reg <- edx %>%
  group_by(movieId) %>%
  summarize(regmoviebias = sum(rating - overall_mean)/(n()+lambda), n_i = n())

# Compute regularized estimates of user bias using lambda
user_avgs_reg <- edx %>%
  left_join(movie_avgs_reg, by='movieId') %>%
  group_by(userId) %>%
  summarize(reguserbias = sum(rating - overall_mean - regmoviebias)/(n()+lambda), n_u = n())

# Predict ratings using the regularized estimates of the movie bias and user bias.
predicted_ratings_reg <- validation %>%
  left_join(movie_avgs_reg, by='movieId') %>%
  left_join(user_avgs_reg, by='userId') %>%
  mutate(pred = overall_mean + regmoviebias + reguserbias) %>%
  .$pred

# Compute and save the RMSE for the regularized movie and user effect model.
model_3_rmse <- RMSE(validation_1$rating,predicted_ratings_reg)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie and User Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% niceKable
```

method	RMSE
Using mean only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170

3.2.4 Regularized With All Effects Model

The approach utilized in the above model is implemented below with the added genres and release year effects.

```r
# year bias and genres bias represent the year & genre effects, respectively
# create a sequence of lambda values from 0 to 20 in increments of 1
lambdas <- seq(0, 20, 1)

# This function fits a regularized model with movie, user, year, and genre effects using the given lamb
# and computes the RMSE of the predicted ratings on the validation set
# the resulting RMSE values are stored in the 'rmses' vector
# Note: the below code could take some time
rmses <- sapply(lambdas, function(l){

  # compute the overall mean rating across all movies and users in the training set
  overall_mean  <- mean(edx$rating)

  # compute regularized movie biases for each movie in the training set
  reg_movie_avgs <- split_genres_edx %>%
    group_by(movieId) %>%
    summarize(regmoviebias = sum(rating - overall_mean)/(n()+l))

  # compute regularized user biases for each user in the training set
  # these biases are adjusted for the regularized movie biases computed earlier
  reg_user_avgs <- split_genres_edx %>%
    left_join(reg_movie_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(reguserbias = sum(rating - regmoviebias - overall_mean)/(n()+l))

  # compute regularized year biases for each year in the training set
  # these biases are adjusted for the regularized movie and user biases computed earlier
  reg_year_avgs <- split_genres_edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    group_by(year) %>%
    summarize(regyearbias = sum(rating - overall_mean - regmoviebias - reguserbias)/(n()+lambda), n_y =

  # compute regularized genre biases for each genre in the training set
  # these biases are adjusted for the regularized movie, user, and year biases computed earlier
  reg_genres_avgs <- split_genres_edx %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    left_join(reg_year_avgs, by = 'year') %>%
    group_by(genres) %>%
    summarize(reggenresbias = sum(rating - overall_mean - regmoviebias - reguserbias - regyearbias)/(n(

  # compute predicted ratings for the validation set using the regularized biases computed earlier
  # these predicted ratings are stored in the 'predicted_ratings' vector
  predicted_ratings <- split_genres_validation %>%
    left_join(reg_movie_avgs, by='movieId') %>%
    left_join(reg_user_avgs, by='userId') %>%
    left_join(reg_year_avgs, by = 'year') %>%
    left_join(reg_genres_avgs, by = 'genres') %>%
    mutate(pred = overall_mean + regmoviebias + reguserbias + regyearbias + reggenresbias) %>%
    .$pred

  # return the RMSE of the predicted ratings on the validation set
  return(RMSE(split_genres_validation_1$rating,predicted_ratings))
})

# Compute new predictions using the optimal lambda
# Test and save results
qplot(lambdas, rmses)
```



```
#select the lambda that results in the minimum RMSE
lambda_2 <- lambdas[which.min(rmses)]

#calculate regularized movie bias and number of ratings per movie
movie_reg_avgs_2 <- split_genres_edx %>%
  group_by(movieId) %>%
  summarize(regmoviebias = sum(rating - overall_mean)/(n()+lambda_2), n_i = n())

#calculate regularized user bias and number of ratings per user
user_reg_avgs_2 <- split_genres_edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  group_by(userId) %>%
  summarize(reguserbias = sum(rating - overall_mean - regmoviebias)/(n()+lambda_2), n_u = n())

#calculate regularized year bias and number of ratings per year
year_reg_avgs <- split_genres_edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  left_join(user_reg_avgs_2, by='userId') %>%
  group_by(year) %>%
  summarize(regyearbias = sum(rating - overall_mean - regmoviebias - reguserbias)/(n()+lambda_2), n_y =

#calculate regularized genre bias and number of ratings per genre
genre_reg_avgs <- split_genres_edx %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  left_join(user_reg_avgs_2, by='userId') %>%
  left_join(year_reg_avgs, by = 'year') %>%
  group_by(genres) %>%
  summarize(reggenresbias = sum(rating - overall_mean - regmoviebias - reguserbias - regyearbias)/(n()+l

#calculate predicted ratings using the regularized biases
predicted_ratings <- split_genres_validation %>%
  left_join(movie_reg_avgs_2, by='movieId') %>%
  left_join(user_reg_avgs_2, by='userId') %>%
  left_join(year_reg_avgs, by = 'year') %>%
  left_join(genre_reg_avgs, by = 'genres') %>%
  mutate(pred = overall_mean + regmoviebias + reguserbias + regyearbias + reggenresbias) %>%
  .$pred

#calculate RMSE of the model on the validation set
model_4_rmse <- RMSE(split_genres_validation_1$rating,predicted_ratings)

#add results to the table of RMSEs
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Reg Movie, User, Year, and Genre Effect Model",
                                     RMSE = model_4_rmse ))
rmse_results %>% niceKable
```

method	RMSE
Using mean only	1.0612018
Movie Effect Model	0.9439087
Movie and User Effect Model	0.8653488
Regularized Movie and User Effect Model	0.8648170
Reg Movie, User, Year, and Genre Effect Model	0.8623650

4 Conclusion

10 Million ratings were analyzed from the MovieLens 10M dataset, and bias factors were applied to account for the effects unique to individual users and movies. We regularized those effects to shrink smaller sample sizes towards the global average, and finally limited the resulting values to a range beween 0.5 and 5.

The final RMSE achieved in the model was:

Method	RMSE
Final RMSE	0.862365

BETTER.

4.1 Future Considerations

The model may be improved in the future by adjusting or changing the modeling approach altogether. Below are some of these opportunities that were not included in the final model. Some opportunities are mutually exclusive, as they would rely on entirely different methodologies.

4.1.1 Matrix Factorization

We could consider using matrix factorization to build a model.

We would create a matrix with movies in rows and users in columns (or vice versa), with the intersection being a given users's rating for a given film. Here is an example with the most commonly-rated films, with ratings given by the first 7 users.

	1	2	3	4	6	7
Apollo 13 (1995)	NA	NA	NA	5	NA	NA
Batman (1989)	NA	NA	NA	5	NA	NA
Braveheart (1995)	NA	5	4.5	5	NA	NA
Forrest Gump (1994)	5	NA	NA	NA	NA	NA
Fugitive, The (1993)	NA	NA	NA	NA	5	NA
Jurassic Park (1993)	NA	NA	NA	5	NA	NA
Silence of the Lambs, The (1991)	NA	NA	NA	NA	NA	3
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	NA	5	NA	NA	NA	3
Terminator 2: Judgment Day (1991)	5	NA	NA	5	NA	NA

Since each user doesn't rate every film, this is what's known as a 'sparse' matrix, with many *"NA"* values.

4.1.2 Rounding

We could perhaps further improve the performance of the model by rounding our predictions to the nearest integer, only when predicting ratings from 2003 and earlier.

4.1.3 Genre Effects

A future model may be improved by considering genre effects, especially user+genre effects.

To achive this, we could use *one-hot encoding* to transform our pipe separated "blob" of genres into individual boolean columns that indicate whether a film falls into that genre.

This would permit us to calculate genre biases (dramas tend to be more highly rated than comedies). A more advanced application could perhaps consider user-genre biases, accounting for the fact that certain users prefer certain genres.

	title	year	genres_Action	genres_Adventure
1	Boomerang (1992)	1,992	0	0
2	Net, The (1995)	1,995	1	0
4	Outbreak (1995)	1,995	1	0
5	Stargate (1994)	1,994	1	1
6	Star Trek: Generations (1994)	1,994	1	1

4.1.4 Rating Delay

Movies that are rated shortly after they were released seem to have a different bias than movies that were rated long after they were released. In our correlation plot, we observed that `review_delay` was positively correlated with `rating`.

There are many causal factors (e.g. nostalgia, recency bias, selection bias, true decline in quality over time) that could explain this correlation so additional exploration would be needed to understand this relationship and its potential predictive utility.

4.1.5 Machine learning techniques

Furthermore, to develop the final model, we didn't use any CPU or RAM-intensive machine learning models. For this author, such an undertaking proved infeasible due to technological limitations in handling the size of this dataset.

5 References

1. Rafael A. Irizarry (2021), Introduction to Data Science: https://rafalab.github.io/dsbook/
2. Model Fitting and Recommendation Systems Overview: https://learning.edx.org/course/course-v1:HarvardX+PH125.8x+2T2021/block-v1:HarvardX+PH125.8x+2T2021+type@sequential+block@568d02a4412440489621921b87e7536f/block-v1:HarvardX+PH125.8x+2T2021+type@vertical+block@d7b4b0783dd443d1bd14504fe2333c24