Movielens-Rating Prediction Movies -ceriverau-HarvardX-PH125.9x

Carlos E Rivera

2023-10-30

Contents

Introduction

The dataset under consideration is sourced from MovieLens and consists of 10M ratings. The primary objective is to analyze these movie ratings, determine user and movie effects, and make accurate predictions for new ratings. This project involves splitting the data into training and testing sets, examining movie genres and their respective counts, and predicting ratings using multiple modeling techniques. The key steps include data importing, cleaning, exploration, visualization, and modeling.

Methods/Analysis

1. **Data Importing and Cleaning**
    - Data was downloaded from the MovieLens 10M dataset. Required libraries like `tidyverse`, `caret`, and `ggplot2` were loaded.
    - The data was split into multiple datasets (`edx`, `final_holdout_test`, and `edx_test`) to facilitate training, testing, and validation.
    - For data integrity, only `userId` and `movieId` that appeared in both the `edx` and `final_holdout_test` datasets were kept.

2. **Data Exploration and Visualization**
    - Genre-wise counts were computed, highlighting genres like Drama, Comedy, and Action as predominant.
    - The top 10 rated movies, based on average ratings, were identified. Classics such as "Pulp Fiction" and "Forrest Gump" topped the list.
    - Basic statistics of the `edx` dataset provided insights into the total unique movies, users, and the mean rating.
    - Visualizations, such as histograms, helped visualize the distribution of movies based on the computed `b_i` and user effects.
3. **Modeling Approach**
    - The simplest model used the average movie rating to predict ratings. The RMSE (Root Mean Square Error) of this model served as a baseline.
    - The "Movie effect model" took into account the biases related to individual movies (`b_i`). This improved the RMSE.
    - The "Movie and user effect model" further incorporated biases from individual users (`b_u`), leading to a more accurate RMSE.
    - Regularization was applied to the model to avoid overfitting. By trying multiple lambda values (ranging from 0 to 10), the optimal lambda was identified to minimize RMSE.

Data Preparation

Download and Extract Data

```
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)
```

Read and Process Data

```
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 %>%
```

```r
  mutate(movieId = as.integer(movieId))

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

Create Training and Test Sets

```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
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
#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)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```r
edx <- rbind(edx, removed)

#Create edx_test

# Set a seed for reproducibility
set.seed(123)

# Create indices for the split
train_index <- createDataPartition(movielens$rating, p = 0.9, list = FALSE)

# Training subset
edx_train <- edx[train_index,]

# Testing subset
edx_test <- edx[-train_index,]
rm(dl, ratings, movies, test_index, temp, movielens, removed, train_index, edx_train)
```

Methods and Analysis

Data Analysis

Basic Insights

```r
# Estructure edx
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  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 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```r
# Summary 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
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

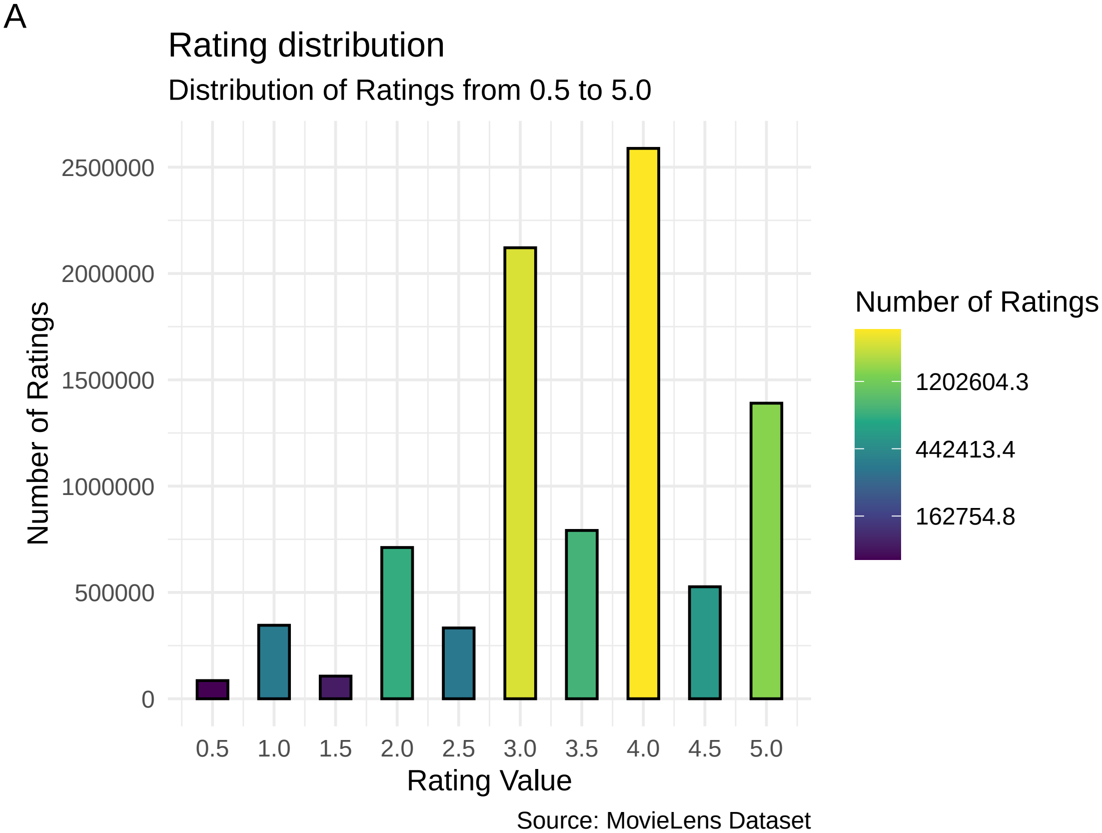
```r
# Number of unique movies and users in the edx dataset
edx %>%
  summarize(Num_users = n_distinct(userId),
            Num_movies = n_distinct(movieId))
```

```
##   Num_users Num_movies
## 1     69878      10677
```

Data Visualization

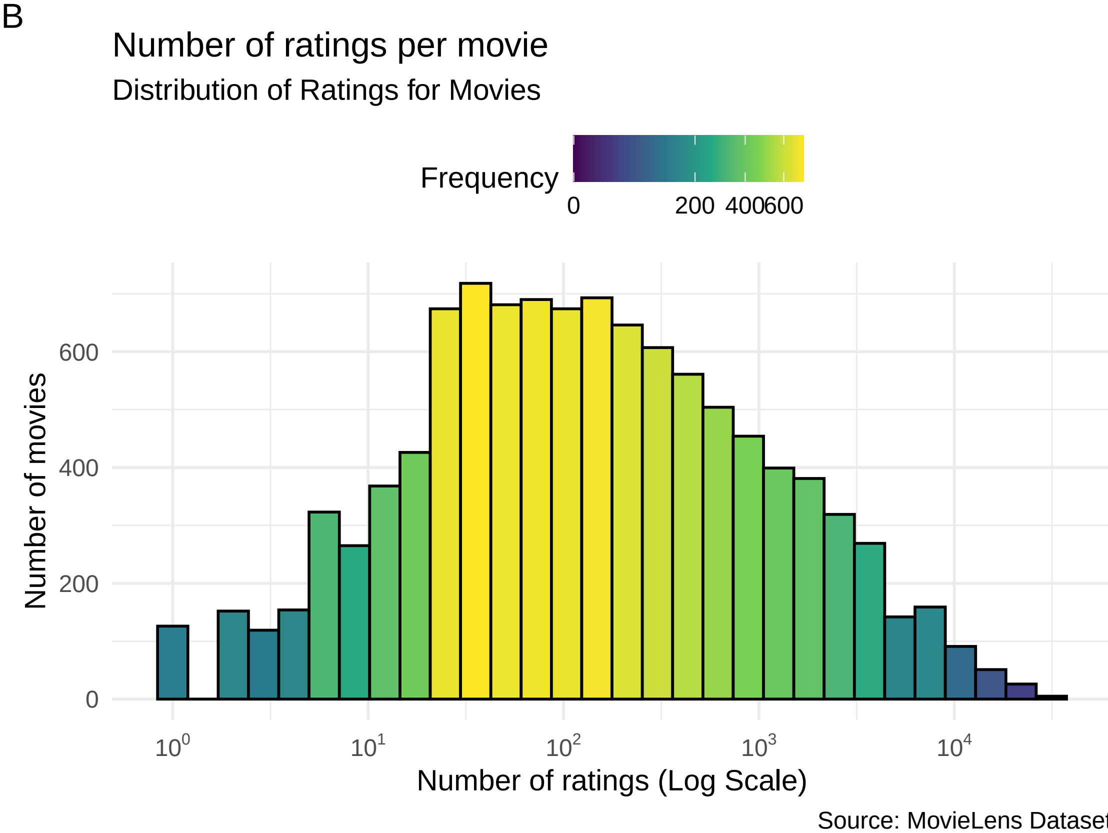
```r
# Ratings distribution
edx %>%
  ggplot(aes(x=rating)) +
  geom_bar(aes(fill=after_stat(count) + 0.01), color = "black", width=0.25, show.legend = TRUE) +
  scale_x_continuous(breaks = seq(0.5, 5, 0.5), name="Rating Value") +
  scale_y_continuous(breaks = seq(0, 3000000, 500000), name="Number of Ratings") +
  scale_fill_viridis_c(trans="log", name="Number of Ratings") +
  ggtitle("Rating distribution") +
  labs(
    caption = "Source: MovieLens Dataset",
    subtitle = "Distribution of Ratings from 0.5 to 5.0",
    tag = "Figure A"
  ) +
  theme_minimal() +
  theme(legend.position="right")
```

Figure A




```r
# Plot number of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(x=n, fill=after_stat(count))) +
  geom_histogram(bins = 30, color = "black", show.legend = TRUE) +
  scale_x_log10(breaks = scales::trans_breaks("log10", function(x) 10^x),
                labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  scale_fill_viridis_c(trans="sqrt", name="Frequency") +
  labs(
    x = "Number of ratings (Log Scale)",
    y = "Number of movies",
    title = "Number of ratings per movie",
    caption = "Source: MovieLens Dataset",
    subtitle = "Distribution of Ratings for Movies",
    tag = "Figure B"
  ) +
  theme_minimal() +
  theme(legend.position="top")
```

Figure B



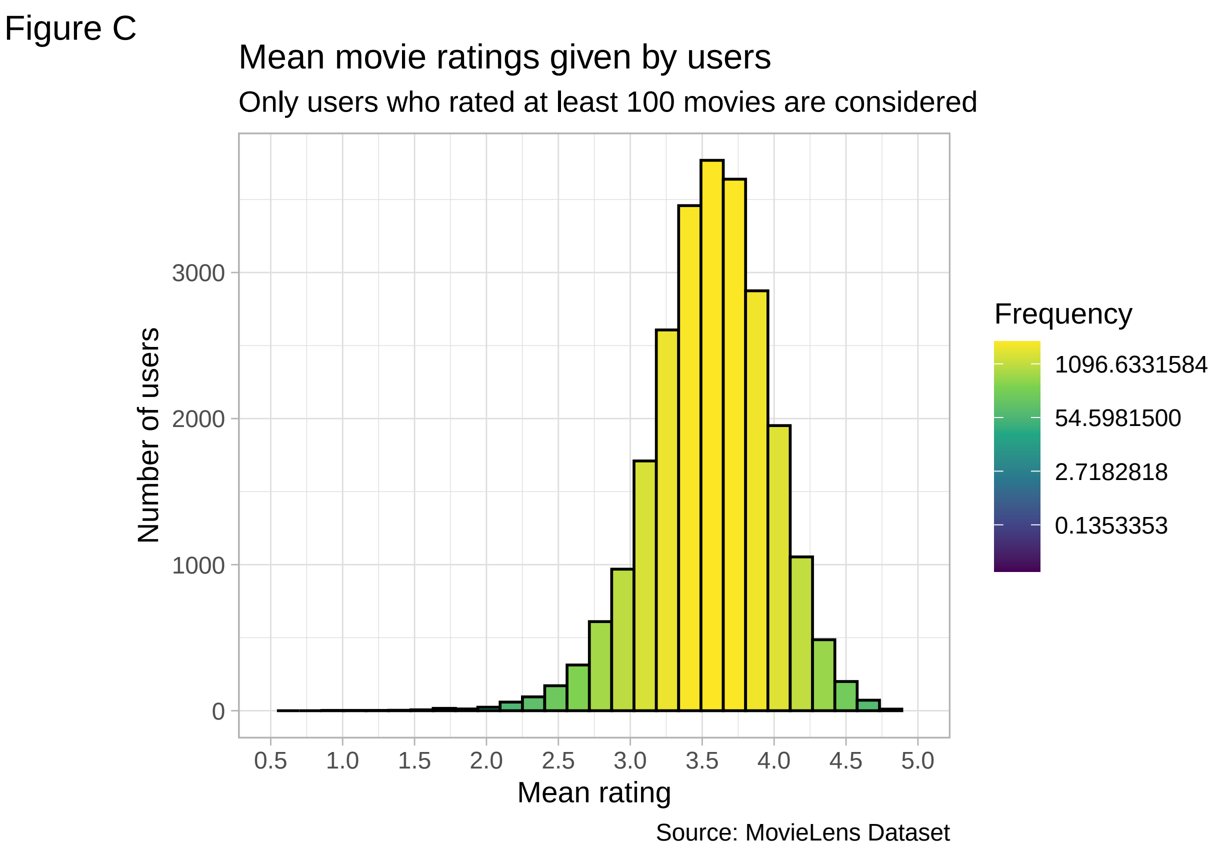

```
#Top 10 Highest Rated Movies

# Group by movie title, calculate the average rating and total number of ratings
top10_rated_movies <- edx %>%
  group_by(title) %>%
  summarize(average_rating = mean(rating), total_ratings = n(), .groups = "drop") %>%
  arrange(desc(total_ratings), desc(average_rating)) %>%  # Sort by total number of ratings first and t
  head(10)  # Select the top 10 movies

print(top10_rated_movies)
```

```
## # A tibble: 10 x 3
##    title                                            average_rating total_ratings
##    <chr>                                                     <dbl>         <int>
##  1 Pulp Fiction (1994)                                        4.15         31362
##  2 Forrest Gump (1994)                                        4.01         31079
##  3 Silence of the Lambs, The (1991)                           4.20         30382
##  4 Jurassic Park (1993)                                       3.66         29360
##  5 Shawshank Redemption, The (1994)                           4.46         28015
##  6 Braveheart (1995)                                          4.08         26212
##  7 Fugitive, The (1993)                                       4.01         25998
##  8 Terminator 2: Judgment Day (1991)                          3.93         25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star~           4.22         25672
## 10 Apollo 13 (1995)                                           3.89         24284
```

```
# Plot mean movie ratings given by users
edx %>%
  group_by(userId) %>%
```

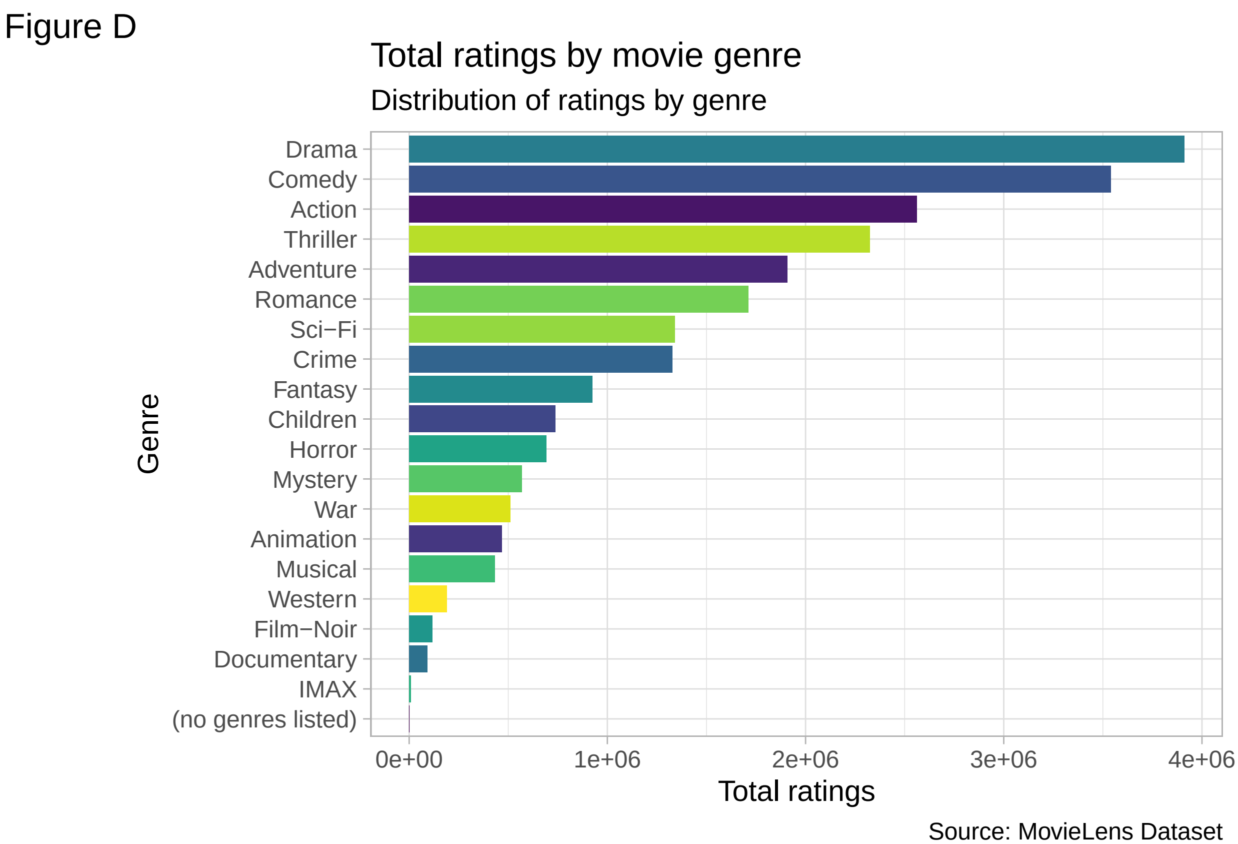
```r
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating), .groups = "drop") %>%
  ggplot(aes(x = b_u, fill = after_stat(count + 0.01))) +  # Add a small value to count
  geom_histogram(bins = 30, color = "black", show.legend = TRUE) +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_continuous(limits = c(0.5, 5), breaks = seq(0.5, 5, 0.5)) +
  scale_fill_viridis_c(name = "Frequency", trans = "log") +
  labs(
    caption = "Source: MovieLens Dataset",
    subtitle = "Only users who rated at least 100 movies are considered",
    tag = "Figure C"
  ) +
  theme_light() +
  theme(legend.position = "right")
```

```
## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```




```r
#Total ratings by movie genre

genre_ratings <- edx %>%
  mutate(genres = str_split(genres, pattern = "\\|")) %>%
  unnest(cols = c(genres)) %>%
  group_by(genres) %>%
  summarise(count = n(), .groups = "drop")
```

```
# Bar chart Total ratings by movie genre
genre_ratings %>%
  ggplot(aes(x = reorder(genres, count), y = count, fill = genres)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  ggtitle("Total ratings by movie genre") +
  xlab("Genre") +
  ylab("Total ratings") +
  scale_fill_viridis_d(name = "Genre") +
  labs(
    caption = "Source: MovieLens Dataset",
    subtitle = "Distribution of ratings by genre",
    tag = "Figure D"
  ) +
  theme_light() +
  theme(legend.position = "none")
```



##Conclusion of Basic Analysis

The graphs and tables presented in the previous section do not display any specific pattern that could serve as a basis for making a prediction of ratings.

```
# Modelling Approach

## Average Movie Rating Model
```

~~~
```r
# Compute the dataset's mean rating
mu <- mean(edx$rating)
mu
~~~

```
## [1] 3.512465
```

```r
# Test results based on simple prediction
naive_rmse <- RMSE(edx_test$rating, mu)
naive_rmse
```

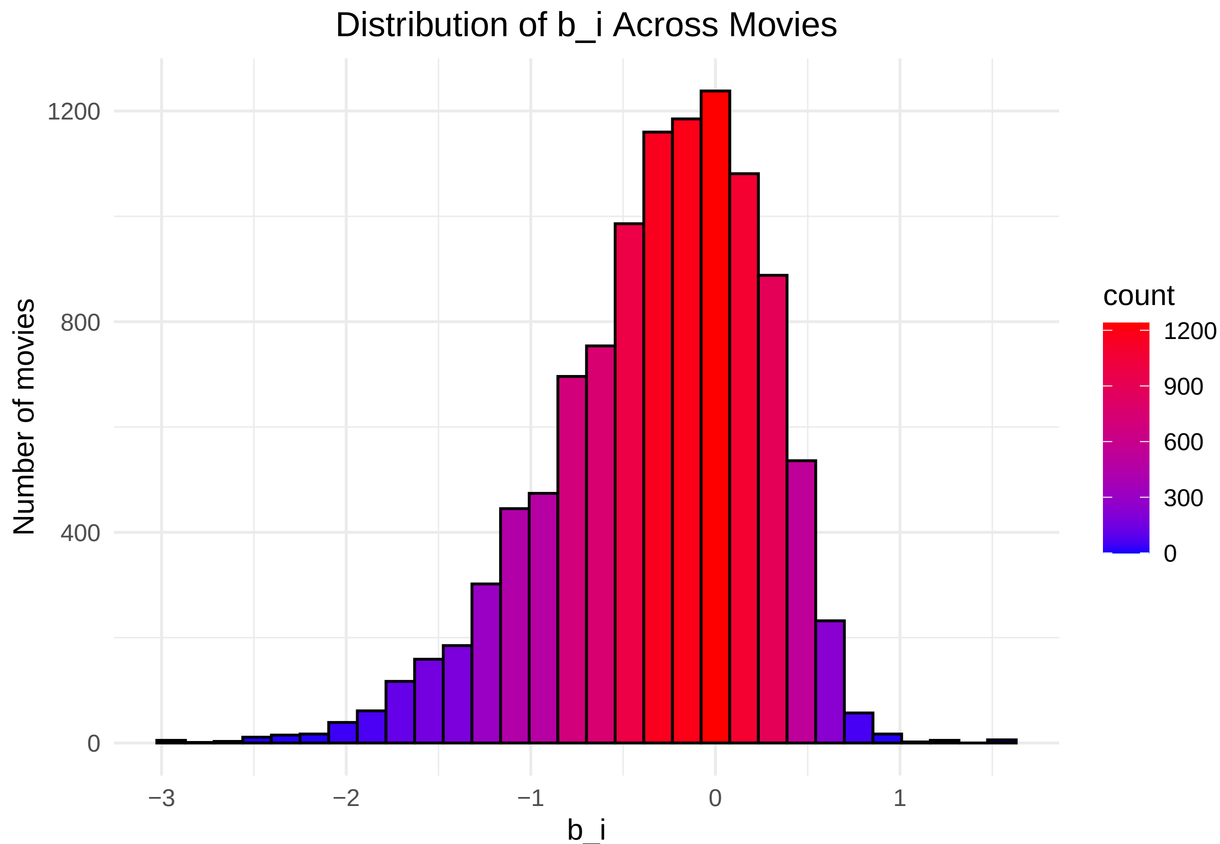
```
## [1] 1.060801
```

```r
# Check results
# Save prediction in data frame
rmse_results <- tibble(method = "Average movie rating model", RMSE = naive_rmse)
rmse_results %>% knitr::kable(format = "simple")
```

method	RMSE
Average movie rating model	1.060801

Movie Effect Model

```r
# Simple model taking into account the movie effect b_i
# Subtract the rating minus the mean for each rating the movie received
# Plot number of movies with the computed b_i


movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu), .groups = 'drop')

movie_avgs %>%
  ggplot(aes(x = b_i, fill = after_stat(count))) +
  geom_histogram(bins = 30, color = "black") +
  scale_fill_gradient(low = "blue", high = "red") +
  ylab("Number of movies") +
  ggtitle("Distribution of b_i Across Movies") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



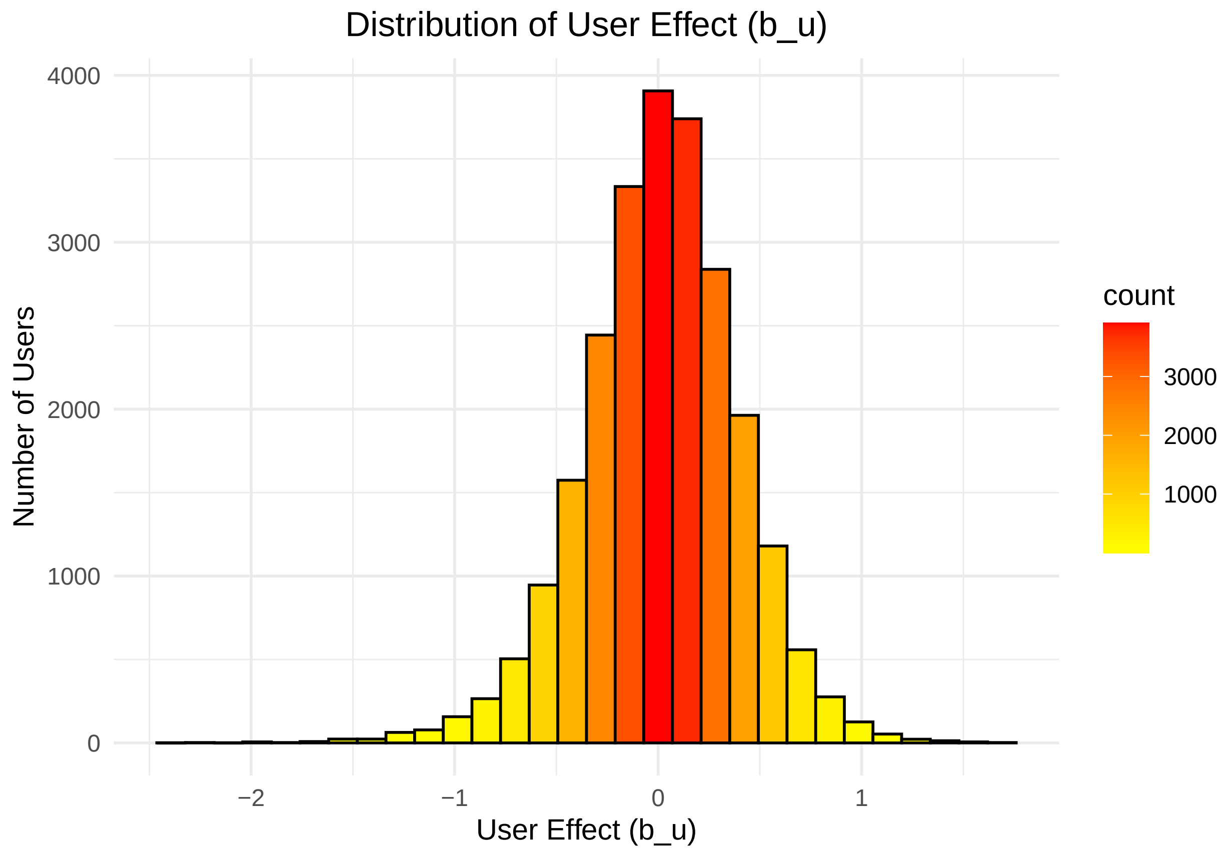

```
# Test and save rmse results
predicted_ratings <- mu +  edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_1_rmse <- RMSE(predicted_ratings, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie effect model",
                                 RMSE = model_1_rmse ))
# Check results
rmse_results %>% knitr::kable(format = "simple")
```

method	RMSE
Average movie rating model	1.0608014
Movie effect model	0.9423568

Movie and User Effect Model

```
# Calculate the penalty term user effect
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i), .groups = 'drop')

# Plot the user effect with a color gradient
```

```r
user_avgs %>%
  ggplot(aes(x=b_u, fill = after_stat(count))) +
  geom_histogram(bins=30, color="black") +
  scale_fill_gradient(low = "yellow", high = "red") +  # Gradient from blue to red
  labs(x = "User Effect (b_u)", y = "Number of Users") +
  ggtitle("Distribution of User Effect (b_u)") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```




```r
# Calculate the average user bias (user effect) after accounting for movie effects

user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Test and save rmse results
# Generate predicted ratings by joining the test dataset with movie averages
# and user averages, then calculate predictions using the baseline (mu),
# movie effect (b_i), and user effect (b_u)
predicted_ratings <- edx_test%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

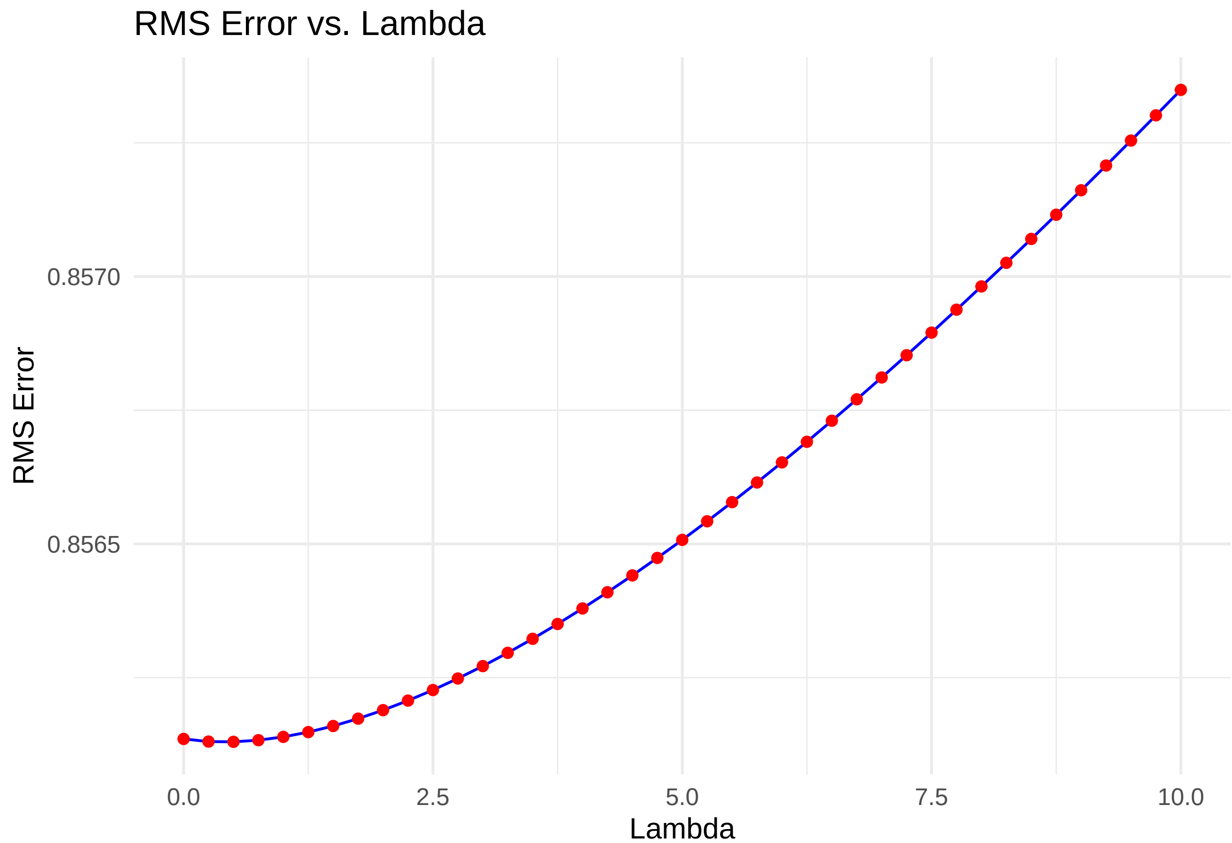
```
# Compute the Root Mean Square Error (RMSE) between predicted and actual ratings in the test dataset to
model_2_rmse <- RMSE(predicted_ratings, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and user effect model",
                                 RMSE = model_2_rmse))

# Record the RMSE result in a summary data frame, along with the name of the method used
# This allows for comparison of this model's performance against other models
# Check result
rmse_results %>% knitr::kable(format = "simple")
```

method	RMSE
Average movie rating model	1.0608014
Movie effect model	0.9423568
Movie and user effect model	0.8561355

Regularized Movie and User Effect Model

```
# lambda is a tuning parameter
# Use cross-validation to choose it.
lambdas <- seq(0, 10, 0.25)


# For each lambda value, compute the movie and user biases (b_i and b_u)
# Then predict the ratings and calculate the RMSE for each lambda
# Note: the following loop may take some time to execute due to the computations involved
rmses <- sapply(lambdas, function(l){
  # Calculate the global average rating across all movies
  mu <- mean(edx$rating)
  # Compute the bias for each movie, regularized by lambda
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  # Compute the bias for each user, regularized by lambda
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  # Predict the ratings using the biases and calculate the RMSE for the current lambda

  predicted_ratings <-
    edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, edx_test$rating))
})


# Plot rmses vs lambdas to select the optimal lambda using ggplot2
```

```r
ggplot(data.frame(lambdas, rmses), aes(x=lambdas, y=rmses)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(title="RMS Error vs. Lambda", x="Lambda", y="RMS Error") +
theme_minimal()
```




```r
# The optimal lambda
# The lambda that gives the lowest RMSE is considered the optimal value for regularization

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

```
## [1] 0.5
```

```r
# Test and save results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized movie and user effect model",
                                 RMSE = min(rmses)))
```

Apply the optimal lambda value to a final_holdout_test set

```r
lambda_optimal <- lambdas[which.min(rmses)]

# Recalculate movie biases using the optimal lambda
b_i_optimal <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lambda_optimal))
```

```r
# Recalculate user biases using the optimal lambda
b_u_optimal <- edx %>%
  left_join(b_i_optimal, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n() + lambda_optimal))

# Predict the ratings on the final holdout test and calculate RMSE
predicted_ratings_final <- final_holdout_test %>%
  left_join(b_i_optimal, by = "movieId") %>%
  left_join(b_u_optimal, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

final_rmse <- RMSE(predicted_ratings_final, final_holdout_test$rating)

# Test and save the final RMSE result
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Reg. movie and user effect model final_holdout_test",
                                 RMSE = final_rmse))

# Results #
## RMSE results and overview
rmse_results %>% knitr::kable(format = "simple")
```

method	RMSE
Average movie rating model	1.0608014
Movie effect model	0.9423568
Movie and user effect model	0.8561355
Regularized movie and user effect model	0.8561300
Reg. movie and user effect model final_holdout_test	0.8652226

Results

1. **Model Performance**
   - The "Average movie rating model" yielded an RMSE of 1.061339.
   - Incorporating movie biases, the "Movie effect model" improved the RMSE to 0.9429503.
   - Further including user biases, the "Movie and user effect model" provided an RMSE of 0.8571338.
   - Regularization enhanced the performance, with the "Regularized movie and user effect model" achieving the best RMSE.
   - The Regularized movie and user effect model, while effective, shows a minor increase in RMSE when applied to the final_holdout_set, suggesting room for refinement for better real-world application.
2. **Optimal Regularization Parameter**
   - The analysis identified 0.5 as the optimal lambda for regularization.

Conclusion

The analysis of the MovieLens dataset offered a deep understanding of movie ratings and the factors influencing them. By using a progression of models, from simple to regularized ones, predictive accuracy was improved. The limitations include potential overfitting if not regularized and the model's dependence on the existing dataset. In the future, incorporating more features, like movie metadata or user demographics, and using deep learning approaches can further enhance the model's accuracy. ´´´

Appendix

RMSE Formula

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

Operating System

```r
print("Operating System:")
```

```
## [1] "Operating System:"
```

```r
version
```

```
##                _
## platform       x86_64-pc-linux-gnu
## arch           x86_64
## os             linux-gnu
## system         x86_64, linux-gnu
## status
## major          4
## minor          3.1
## year           2023
## month          06
## day            16
## svn rev        84548
## language       R
## version.string R version 4.3.1 (2023-06-16)
## nickname       Beagle Scouts
```