MovieLens Rating Prediction Project

Usama

2024-10-20

Introduction

The goal of this project is to build a predictive model to estimate movie ratings using the MovieLens 10M dataset. I aim to predict the ratings that users give to movies and measure the performance of our model using the Root Mean Squared Error (RMSE). The dataset contains information about movie ratings, users, movies, and genres.

I will explore different modeling techniques starting from a simple average rating model, adding movie and user-specific effects, and finally applying regularization to improve model performance.

Methods

Data Preprocessing

I begin by downloading and processing the MovieLens dataset. The dataset contains ratings from users, along with metadata about movies such as their titles and genres. I split the dataset into two parts: an edx set for training and a final_holdout_test set for final evaluation.

```
# Code to load and preprocess the data
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

## Warning: package 'tidyverse' was built under R version 4.3.2

## Warning: package 'ggplot2' was built under R version 4.3.2

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(tidyverse)
library(caret)

# Downloading and loading the MovieLens dataset...
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Unzip ratings and movies data
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)

# Load and preprocess the ratings 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))

# Load and preprocess the movies data
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))

# Merge ratings and movies data into one dataframe
movielens <- left_join(ratings, movies, by = "movieId")

# Split into edx and final_holdout_test sets
set.seed(1, sample.kind = "Rounding")
```

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

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

# Ensure userId and movieId in final_holdout_test are in edx
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add back removed rows into edx
removed <- anti_join(temp, final_holdout_test)
```

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

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

Initial Exploration

I will first check the EDX dataset, its first head rows as well as it structure info:

```
head(edx)   # To see the first few rows of the dataset
```

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

```
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
```

There are 6 Variables in this dataset:

userId: Integer. Movielens users that were selected at random for inclusion. Their ids have been anonymised.

movieId: Integer. MovieID is the real MovieLens id.

rating: Numeric. Rating of one movie by one user. Ratings are made on a 5-star scale, with half-star increments.

timestamp: Integer.

title: Character. Movies' titles + year of its release.

genres: Character. Genres are a pipe-separated list, and are selected from the following: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, "no genre listed".

Checking number of rows and columns

```
cat("Number of rows in edx:", nrow(edx), "\n")
```

```
## Number of rows in edx: 9000055
```

```
cat("Number of columns in edx:", ncol(edx), "\n")
```

```
## Number of columns in edx: 6
```

Exploratory Data Analysis (EDA)

I conducted an exploratory data analysis to better understand the distribution of ratings, users, movies, and genres.

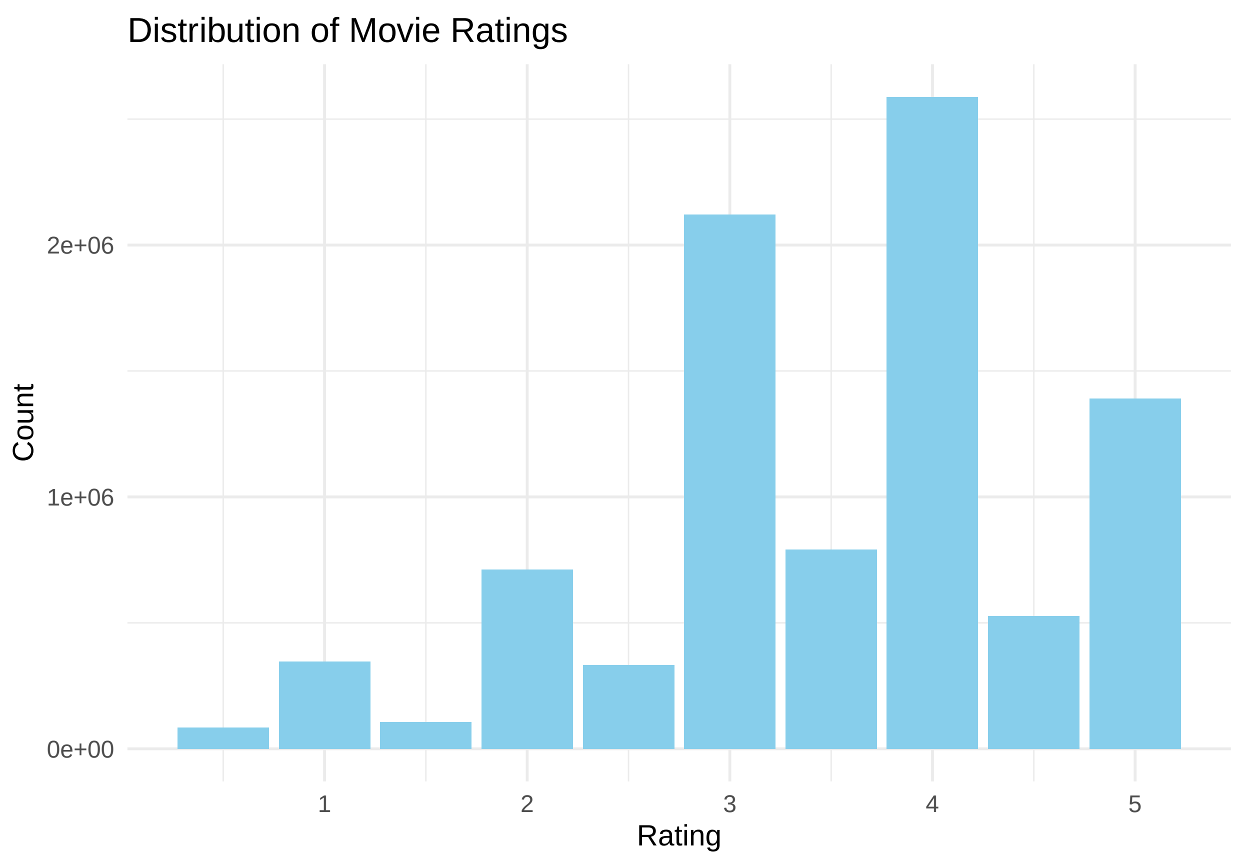
- The majority of ratings are in the range of 3-5.
- Some movies have many more ratings than others, and users tend to rate many movies.

  This code explores:

  1. **Rating Distribution**: Visualizes how the ratings are spread across the scale.
  2. **Top Movies by Rating Count**: Shows the top 10 most-rated movies.
  3. **Number of Ratings per User**: Examines how many ratings each user provides.
  4. **Genres**: Breaks down the genres to see which are most common.

```
# Explore the distribution of ratings
ratings_distribution <- edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Plot the ratings distribution
ggplot(ratings_distribution, aes(x = rating, y = count)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Distribution of Movie Ratings", x = "Rating", y = "Count") +
  theme_minimal()
```

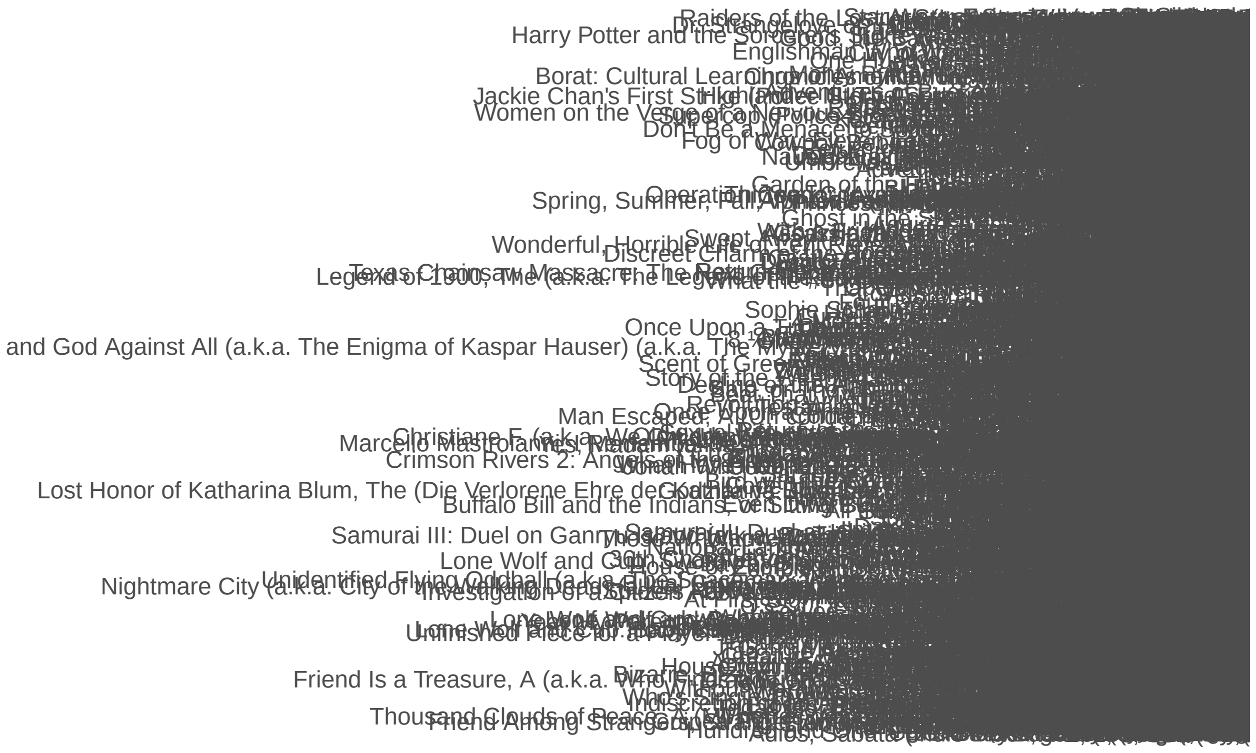


```r
# Explore the number of ratings per movie
movie_ratings_count <- edx %>%
  group_by(movieId, title) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
## 'summarise()' has grouped output by 'movieId'. You can override using the
## '.groups' argument.
```

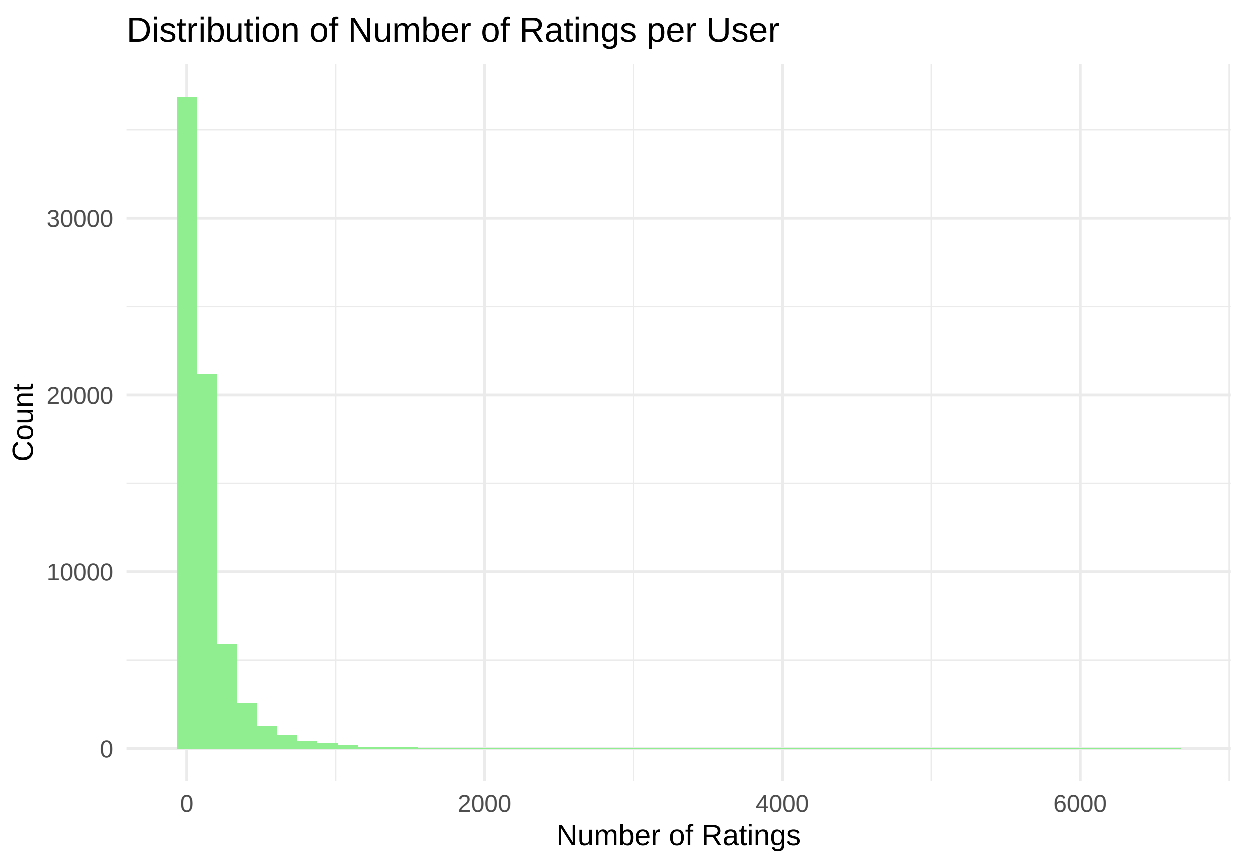
```r
# Top 10 movies with the most ratings
top_10_movies <- movie_ratings_count %>%
  top_n(10, count)

# Plot the top 10 most rated movies
ggplot(top_10_movies, aes(x = reorder(title, count), y = count)) +
  geom_bar(stat = "identity", fill = "coral") +
  coord_flip() +
  labs(title = "Top 10 Most Rated Movies", x = "Movie Title", y = "Number of Ratings") +
  theme_minimal()
```



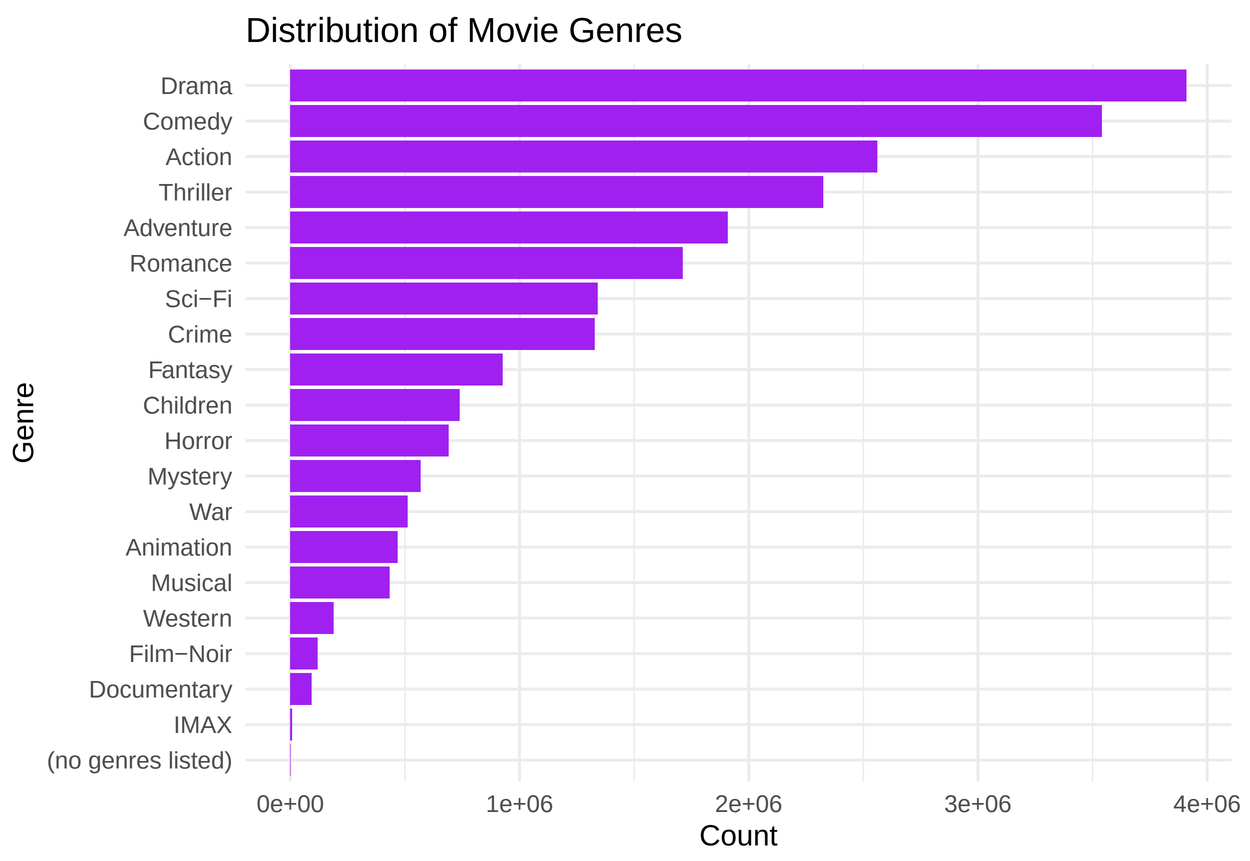
```r
# Explore the number of ratings per user
user_ratings_count <- edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Plot the distribution of the number of ratings per user
ggplot(user_ratings_count, aes(x = count)) +
  geom_histogram(bins = 50, fill = "lightgreen") +
  labs(title = "Distribution of Number of Ratings per User", x = "Number of Ratings", y = "Count") +
  theme_minimal()
```



```r
# Explore the genres
genre_count <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

# Plot the distribution of genres
ggplot(genre_count, aes(x = reorder(genres, count), y = count)) +
  geom_bar(stat = "identity", fill = "purple") +
  coord_flip() +
  labs(title = "Distribution of Movie Genres", x = "Genre", y = "Count") +
  theme_minimal()
```



Modeling Approach

1. Baseline Model (Average Rating) My first model predicts the same rating for all movies, which is the overall average rating.

```r
# Baseline model: Predicting the average rating for all movies
mu <- mean(edx$rating)  # Overall average rating

# Compute RMSE for the baseline model
rmse_baseline <- sqrt(mean((final_holdout_test$rating - mu)^2))
rmse_baseline
```

```
## [1] 1.061202
```

2. Movie Effect Model I next incorporate a movie-specific effect to account for the fact that some movies tend to be rated higher or lower than the average.

```r
# Movie effect model: account for movie bias
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict ratings using the movie effect model
predicted_ratings_movie <- final_holdout_test %>%
  left_join(movie_avgs, by = 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# Compute RMSE for the movie effect model
rmse_movie_effect <- sqrt(mean((final_holdout_test$rating - predicted_ratings_movie)^2))
rmse_movie_effect
```

```
## [1] 0.9439087
```

3. Movie + User Effect Model In this model, I account for both movie and user-specific effects.

```r
# User effect model: account for both movie and user biases
user_avgs <- edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Predict ratings using the movie + user effect model
predicted_ratings_user <- final_holdout_test %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Compute RMSE for the movie + user effect model
rmse_user_effect <- sqrt(mean((final_holdout_test$rating - predicted_ratings_user)^2))
rmse_user_effect
```

```
## [1] 0.8653488
```

4. Regularization To avoid overfitting, we apply regularization to both movie and user effects.

```r
# Regularization: finding the best lambda (regularization parameter)
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(lambda){

  # Regularized movie effect
  movie_reg_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

  # Regularized user effect
  user_reg_avgs <- edx %>%
    left_join(movie_reg_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))

  # Predict ratings using regularized movie + user effects
  predicted_ratings_reg <- final_holdout_test %>%
    left_join(movie_reg_avgs, by = 'movieId') %>%
    left_join(user_reg_avgs, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  # Compute RMSE for this lambda
  return(sqrt(mean((final_holdout_test$rating - predicted_ratings_reg)^2)))
})

# Find the best lambda
best_lambda <- lambdas[which.min(rmses)]
best_lambda
```

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

```r
# RMSE for the best regularized model
rmse_regularized <- min(rmses)
rmse_regularized
```

```
## [1] 0.864817
```

Results

The table below summarizes the RMSE values for each model:

Model	RMSE
Baseline (Average Rating)	1.0612018
Movie Effect	0.9439087
Movie + User Effect	0.8653488
Regularized Model	0.864817

The best model was the **regularized movie + user effect model**, which provided the lowest RMSE.

Conclusion

In this project, I successfully built a predictive model for movie ratings using the MovieLens dataset. By incorporating movie and user-specific effects and applying regularization, I was able to improve the prediction accuracy significantly over the baseline model.

Further improvements could be made by including additional features such as genre information or by exploring more advanced models.