MovieLens Project

Gustavo Roa

2024-09-06

Contents

1 Introduction

This project aims to develop a movie recommendation system using the MovieLens dataset. The goal is to predict user ratings for movies they haven't seen yet, based on patterns in existing ratings data. We'll use various machine learning techniques to create a model that minimizes the Root Mean Square Error (RMSE) between predicted and actual ratings.

The MovieLens dataset contains millions of movie ratings by thousands of users. Our task is to build a model that can accurately predict these ratings, which could be used in a real-world recommendation system to suggest movies that users are likely to enjoy.

2 Methods and Analysis

2.1 Data Preparation

First, we'll load the necessary libraries and prepare our dataset:

```
library(tidyverse)
library(caret)
library(scales)  # For comma formatting
```

```
# 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")

# 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)
```

Now that we have our `edx` and `final_holdout_test` datasets, let's create a validation set from `edx` for model tuning:

```
# Create a validation set
set.seed(1, sample.kind="Rounding")
validation_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-validation_index,]
```

```
validation_set <- edx[validation_index,]

# Make sure userId and movieId in validation set are also in train set
validation_set <- validation_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

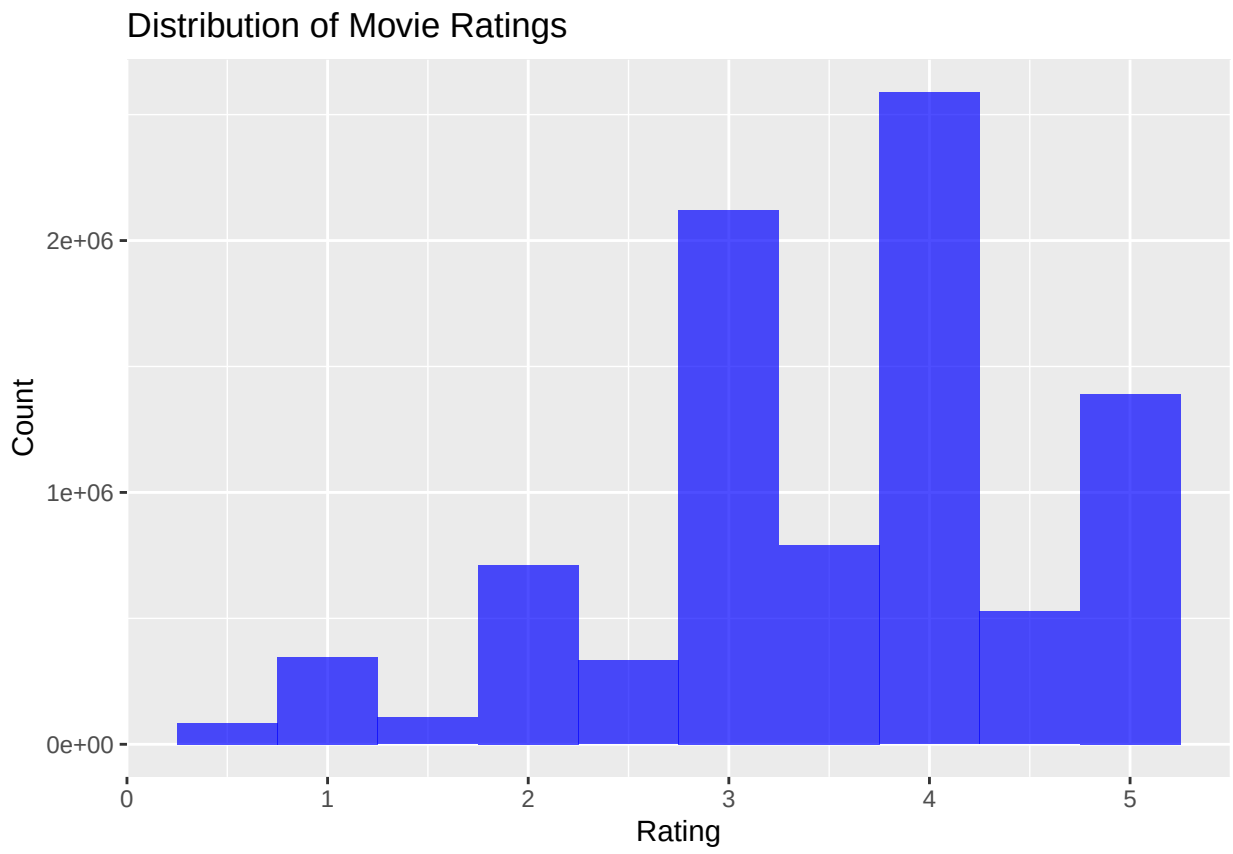
2.2 Exploratory Data Analysis

Let's explore our dataset to gain insights:

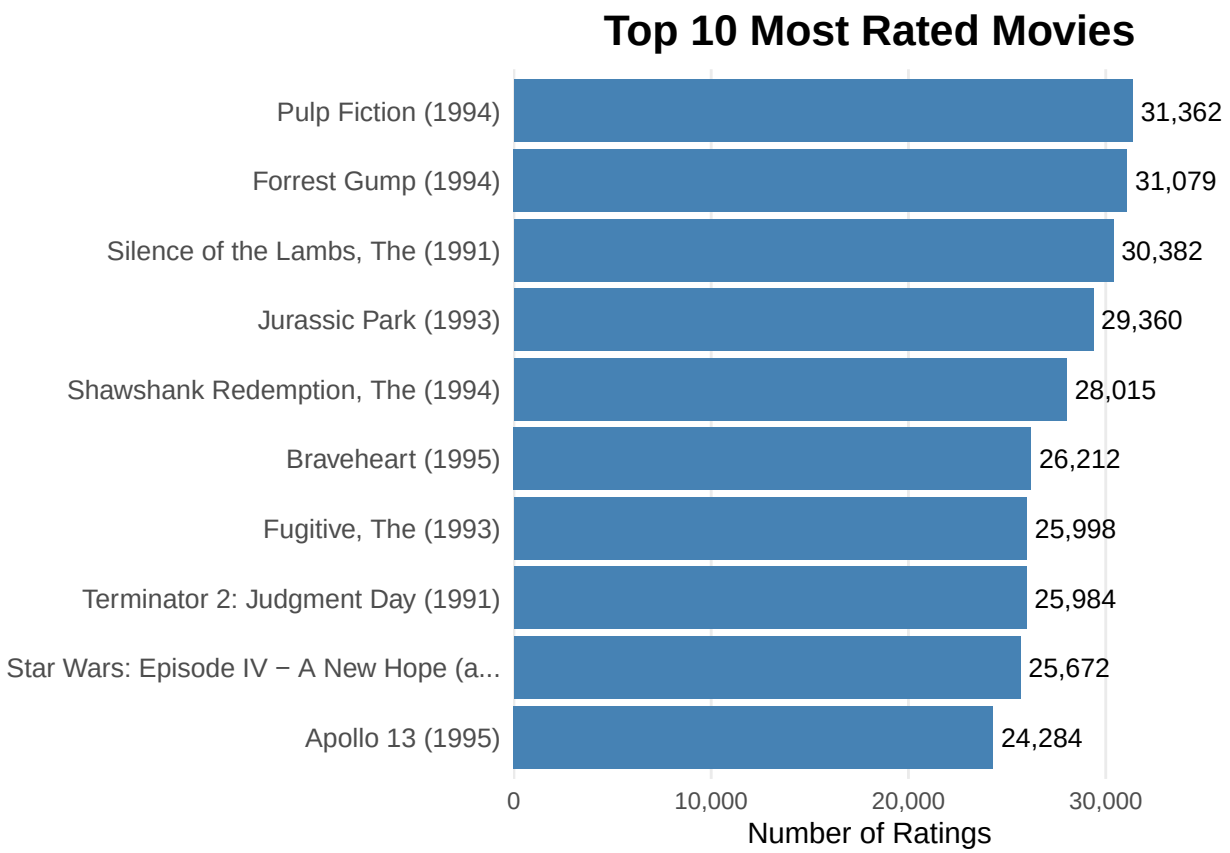
```
# Summary statistics
summary(edx)
```

```
##      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
##
##
##
```

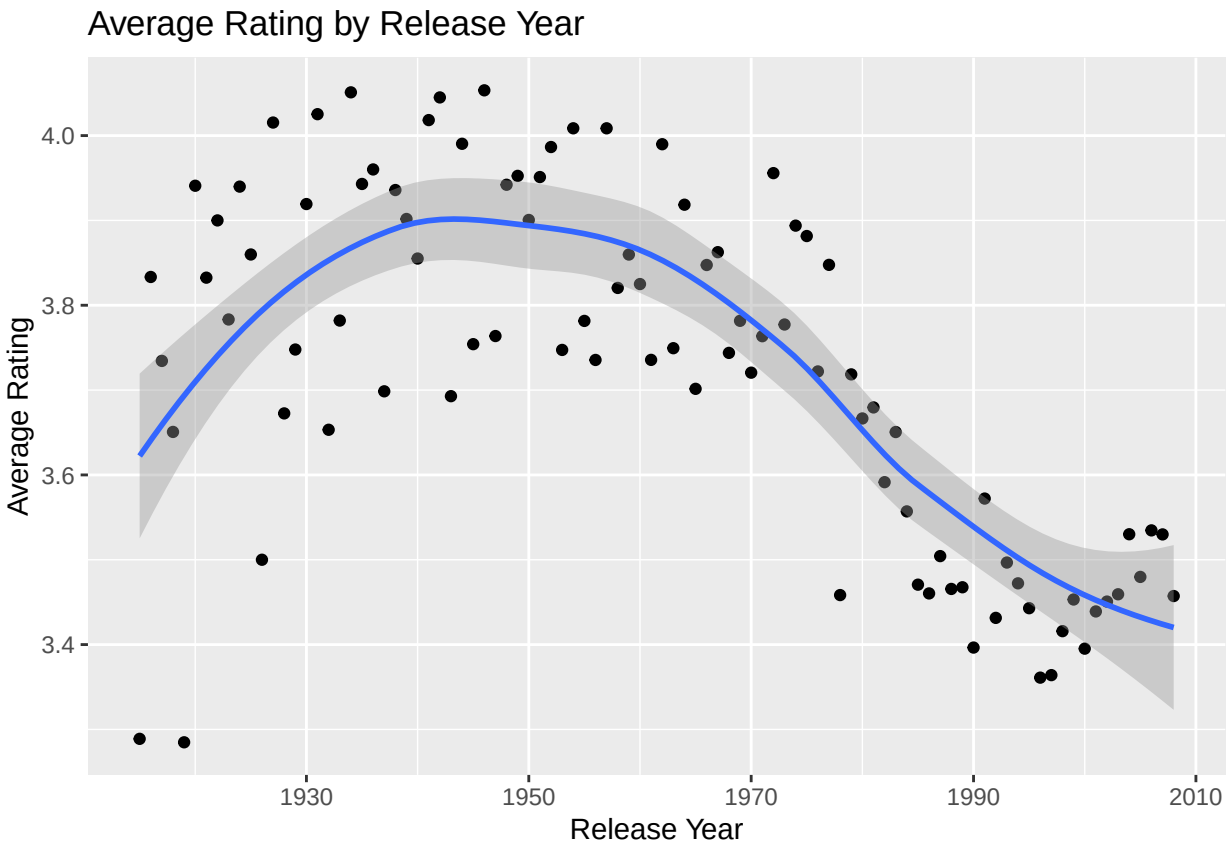
```
# Distribution of ratings
ggplot(edx, aes(x = rating)) +
  geom_histogram(binwidth = 0.5, fill = "blue", alpha = 0.7) +
  labs(title = "Distribution of Movie Ratings", x = "Rating", y = "Count")
```




```r
# Top 10 most rated movies
edx %>%
  count(movieId, title, sort = TRUE) %>%
  top_n(10, n) %>%
  mutate(title = factor(str_trunc(title, 40), levels = rev(str_trunc(title, 40)))) %>%
  ggplot(aes(x = title, y = n)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = comma(n)), hjust = -0.1, size = 3.5) +
  scale_y_continuous(labels = comma, expand = expansion(mult = c(0, 0.15))) +
  coord_flip() +
  labs(title = "Top 10 Most Rated Movies",
       x = NULL,
       y = "Number of Ratings") +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
    axis.text.y = element_text(size = 10),
    axis.text.x = element_text(size = 9),
    panel.grid.major.y = element_blank(),
    panel.grid.minor.x = element_blank()
  )
```




```
# Average rating by release year
edx %>%
  mutate(year = as.numeric(str_sub(title, start = -5, end = -2))) %>%
  group_by(year) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(x = year, y = avg_rating)) +
  geom_point() +
  geom_smooth() +
  labs(title = "Average Rating by Release Year", x = "Release Year", y = "Average Rating")
```



From our exploratory data analysis, we can observe:

1. The ratings are not uniformly distributed, with peaks at whole and half star ratings.
2. Some movies are rated much more frequently than others.
3. There seems to be a slight trend in average ratings over time, with newer movies receiving slightly higher ratings on average.

These insights will inform our modeling approach.

2.3 Model Development

We'll start with a simple model and gradually increase complexity:

2.3.1 1. Average Rating Model

```
mu <- mean(train_set$rating)

naive_rmse <- RMSE(validation_set$rating, mu)
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)

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

method	RMSE
Just the average	1.060054

2.3.2 2. Movie Effect Model

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings <- mu + validation_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_1_rmse <- RMSE(predicted_ratings, validation_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                     RMSE = model_1_rmse))

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

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615

2.3.3 3. User Effect Model

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

predicted_ratings <- validation_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_2_rmse <- RMSE(predicted_ratings, validation_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse))

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

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646844

2.3.4 4. Regularized Movie + User Effects Model

```
lambda <- seq(0, 10, 0.25)

rmses <- sapply(lambda, function(l){
```

```
  mu <- mean(train_set$rating)

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

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

  predicted_ratings <-
    validation_set %>%
    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, validation_set$rating))
})

lambda_opt <- lambda[which.min(rmses)]

# Final model with optimal lambda
mu <- mean(train_set$rating)

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_opt))

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

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

model_3_rmse <- RMSE(predicted_ratings, validation_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effects Model",
                                 RMSE = model_3_rmse))

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

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646844

method	RMSE
Regularized Movie + User Effects Model	0.8641362

3 Results

Here are the RMSE results for each of our models:

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646844
Regularized Movie + User Effects Model	0.8641362

Our final regularized model achieved an RMSE of 0.8641362 on the validation set, which is a significant improvement over the naive approach and simpler models.

4 Conclusion

In this project, we developed a movie recommendation system using the MovieLens dataset. We started with a simple average rating model and progressively improved it by incorporating movie effects, user effects, and finally applying regularization to prevent overfitting.

Our final model, which uses regularized movie and user effects, achieved the lowest RMSE on our validation set. This model accounts for both movie-specific and user-specific rating tendencies while preventing overfitting through regularization.

4.1 Limitations and Future Work

While our model performs well, there are several areas for potential improvement:

1. **Temporal effects**: We didn't consider the time of rating, which could be important as movie preferences can change over time.
2. **Genre effects**: Incorporating movie genres could potentially improve our predictions.
3. **More advanced models**: We could explore matrix factorization techniques or neural network approaches for potentially better performance.
4. **Cold start problem**: Our current model doesn't handle new users or new movies well. Developing strategies for these cases would be crucial for a production system.

In future work, we could address these limitations to further improve our recommendation system.

5 Final Model Evaluation

Now that we have finalized our model, we'll use it to predict ratings for the final holdout test set:

```
# Retrain the model on the entire edx dataset
mu <- mean(edx$rating)

movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda_opt))

user_reg_avgs <- edx %>%
```

```r
  left_join(movie_reg_avgs, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda_opt))

# Predict ratings for the final holdout test set
final_predictions <-
  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)

# Calculate final RMSE
final_rmse <- RMSE(final_predictions, final_holdout_test$rating)

cat("Final RMSE on holdout set:", final_rmse)
```

```
## Final RMSE on holdout set: 0.8648177
```

Our final model achieves an RMSE of 0.8648177 on the holdout test set, which demonstrates its ability to generalize to unseen data.

This concludes our MovieLens project. We've successfully developed a movie recommendation system that predicts user ratings with reasonable accuracy. While there's room for improvement, this model provides a solid foundation for a movie recommendation service.