MovieLens Rating Prediction Project

Yap Kah Yong

2025-08-27

Introduction

In this project, a movie rating prediction algorithm is developed using the MovieLens 10M dataset. The goal is to predict user ratings for movies based on historical data. The model will be trained and validated using the `edx` dataset, and the final evaluation will be performed on the `final_holdout_test` set using **Root Mean Squared Error (RMSE)**.

I follow proper machine learning practices: - Do **not** use `final_holdout_test` during training or model selection. - Use a train/test split on `edx` to tune and evaluate the model. - Only apply the final model to `final_holdout_test` once, at the end.

The algorithm uses a **simple baseline model** with regularized user and item biases:

where: - u = global average rating - b_u = user bias (adjusted with regularization) - b_i = item (movie) bias (adjusted with regularization)

This model is fast, memory-efficient, and avoids the computational bottlenecks of matrix factorization or collaborative filtering.

```
# MovieLens 10M dataset
options(timeout = 120)
dl <- "ml-10M100K.zip"
ratings_file <- "ml-10M100K/ratings.dat"
movies_file <- "ml-10M100K/movies.dat"


# Download if not exists
if (!file.exists(dl)) {
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl,
                method = "curl",
                extra = "-k" )
}

if (!file.exists(ratings_file)) unzip(dl, ratings_file)
if (!file.exists(movies_file)) unzip(dl, movies_file)

# Read data
ratings <- read_delim(ratings_file, delim = "::",
                      col_names = c("userId", "movieId", "rating", "timestamp"),
                      show_col_types = FALSE)
movies <- read_delim(movies_file, delim = "::",
                     col_names = c("movieId", "title", "genres"),
                     show_col_types = FALSE)
```

```r
# Convert types
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Create edx and final_holdout_test
test_index <- createDataPartition(movielens$rating, p = 0.1, list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, final_holdout_test)
```

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

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

# Clean up
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

```r
# Split edx into train and test for model development
set.seed(5)
train_index <- createDataPartition(edx$rating, p = 0.8, list = FALSE)
train_set <- edx[train_index, ]
test_set <- edx[-train_index, ]
```

```r
# --- Enhanced Baseline Model ---
set.seed(5)

# Global mean
mu <- mean(edx$rating)

# Regularization parameter
lambda <- 20

# User bias: b_u = sum(rating - mu) / (lambda + n_u)
user_bias <- edx %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu) / (lambda + n()), .groups = 'drop')

# Movie bias: b_i = sum(rating - mu) / (lambda + n_i)
```

```r
movie_bias <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu) / (lambda + n()), .groups = 'drop')

# Predict on final_holdout_test
predictions <- final_holdout_test %>%
  left_join(user_bias, by = "userId") %>%
  left_join(movie_bias, by = "movieId") %>%
  mutate(
    b_u = ifelse(is.na(b_u), 0, b_u),    # Cold start: no bias → 0
    b_i = ifelse(is.na(b_i), 0, b_i),
    pred = mu + b_u + b_i,                # Final prediction
    pred = pmin(pmax(pred, 0.5), 5)       # Clamp to valid range
  )

# Calculate RMSE
rmse_final <- RMSE(predictions$pred, predictions$rating)
cat("Final Holdout Test RMSE:", round(rmse_final, 5), "\n")
```

```
## Final Holdout Test RMSE: 0.88061
```