MovieLens Recommendation System Project

Paul Henderson

2025-12-03

Introduction

This project builds a movie recommendation algorithm using the **MovieLens 10M dataset**. Following the course guidelines:

- All model development is performed using the **edx** dataset.
- The **final__holdout__test** dataset is used only once, at the end, to evaluate the final model.
- RMSE (Root Mean Squared Error) is used to compare models.

The goal is to construct increasingly effective models, evaluate their performance, and select a final approach for predicting movie ratings.

Data Preparation

```
edx <- readRDS("edx.rds")
final_holdout_test <- readRDS("final_holdout_test.rds")
```

We begin by examining the structure of **edx**, which contains:

- 9000055 rows
- 6 columns

Each row corresponds to a user–movie rating with metadata including timestamp, movie title, and genres.

Splitting edx into Training and Test Sets

Following best practices, we create an internal test set from edx for model evaluation.

```
set.seed(1)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)

edx_train <- edx[-test_index, ]
edx_test  <- edx[test_index, ] %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

RMSE <- function(true, pred){
  sqrt(mean((true - pred)^2))
}
```

Modeling Approach

We begin with simple baseline models and progressively introduce structure:

1. Global average model
2. Movie effect model
3. Movie + user effects model
4. Regularized movie + user effects model

Each model is evaluated using RMSE on the internal test set.

```
rmse_results <- tibble(method = character(), RMSE = numeric())

# Global mean
mu_hat <- mean(edx_train$rating)
pred_baseline <- rep(mu_hat, nrow(edx_test))
rmse_baseline <- RMSE(edx_test$rating, pred_baseline)

rmse_results <- bind_rows(rmse_results,
                          tibble(method = "Global mean", RMSE = rmse_baseline))
rmse_results
```

```
## # A tibble: 1 x 2
##   method       RMSE
##   <chr>       <dbl>
## 1 Global mean  1.06
```

Movie Effects Model

```r
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat), .groups = "drop")

pred_movie <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(pred = mu_hat + b_i) %>%
  pull(pred)

rmse_movie <- RMSE(edx_test$rating, pred_movie)

rmse_results <- bind_rows(rmse_results,
                          tibble(method = "Movie effect", RMSE = rmse_movie))
rmse_results
```

```
## # A tibble: 2 x 2
##   method        RMSE
##   <chr>        <dbl>
## 1 Global mean  1.06
## 2 Movie effect 0.944
```

Movie + User Effects Model

```r
user_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(u_j = mean(rating - mu_hat - b_i), .groups = "drop")

pred_movie_user <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu_hat + b_i + u_j) %>%
  pull(pred)

rmse_movie_user <- RMSE(edx_test$rating, pred_movie_user)

rmse_results <- bind_rows(rmse_results,
                          tibble(method = "Movie + user effects", RMSE = rmse_movie_user))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                RMSE
##   <chr>                <dbl>
## 1 Global mean          1.06
## 2 Movie effect         0.944
## 3 Movie + user effects 0.866
```

Regularized Movie + User Effects Model

To prevent overfitting, we introduce **regularization**, shrinking movie and user effects by lambda.

```r
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(lambda) {
  mu <- mean(edx_train$rating)

  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda), .groups = "drop")

  u_j <- edx_train %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(u_j = sum(rating - mu - b_i) / (n() + lambda), .groups = "drop")

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

  RMSE(edx_test$rating, preds)
})

lambda_best <- lambdas[which.min(rmses)]
rmse_reg <- min(rmses)

rmse_results <- bind_rows(rmse_results,
                          tibble(method = paste0("Regularized movie + user (lambda=", lambda_best, ")")
                                 RMSE = rmse_reg))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                               RMSE
##   <chr>                               <dbl>
## 1 Global mean                         1.06
## 2 Movie effect                        0.944
## 3 Movie + user effects                0.866
## 4 Regularized movie + user (lambda=5) 0.866
```

The optimal value found was **lambda = 5**, giving an RMSE of **0.865554**, the lowest among evaluated models.

Final Model Selection

The **regularized movie + user model** demonstrated the best performance and is therefore selected as the final algorithm.

The next step (run separately from this report) is to fit the final model on the full **edx** dataset, generate predictions for the **final__holdout__test** set, and compute the final RMSE. This procedure is implemented in the required `movielens_model.R` script. Running that script produces an RMSE of **0.8648** on the untouched holdout set.

Conclusion

The MovieLens 10M dataset is a great sandbox for testing recommendation models because the structure is clean and the scale is big enough to show real patterns. I started with the simplest possible approach, the global average rating, and then added complexity only when the data supported it. Bringing in movie effects, then user effects, and finally regularizing both pieces steadily tightened the predictions and kept the model from chasing noise.

The regularized movie + user effects model ended up performing the best, landing around 0.865 RMSE on the internal edx test set. Training that same model on the full edx data and evaluating it on the untouched final_holdout_test set gave an RMSE of 0.8648. The near-match between the two numbers is exactly what you want to see: it means the model generalizes instead of leaning on quirks of a particular split.

This kind of bias-based recommender isn't flashy, but it's reliable, fast, and easy to interpret. It also gives a sturdy foundation for anything more ambitious, like matrix factorization, time-based effects, or models that pull in movie metadata. For a baseline system, it gets the job done.
