Movie Recommender System using Baseline Predictors

Saloua Daouki

2025-05-31

Contents

Update Note for Resubmission (June 7, 2025):

Per your feedback, the following clarifications and improvements were added to this submission:

1. **Explanation of the baseline recommender model** (under "Understanding the Baseline Model")
2. **Clarification of user and item biases**
3. **Discussion of model improvements, including regularization** (as a part of the conclusion)
4. **Definition and interpretation of RMSE** (under "RMSE Interpretation")

Thank you for your guidance!

0.1 Introduction

This project builds a simple movie recommender system using the MovieLens dataset. I implement baseline predictors, which adjust predictions based on user and item effects relative to the global average rating. This model is a common starting point in collaborative filtering systems.

0.2 Understanding the Baseline Model

The **baseline recommender** estimates a user's rating for an item by starting with the **global average rating** and adjusting it using two bias terms:

- **User bias**: how much a particular user tends to rate higher or lower than average.
- **Item bias**: how much a particular movie tends to receive higher or lower ratings than average.

The **core idea** is that some users consistently rate things higher (or lower), and some movies are universally liked or disliked. Capturing these effects improves prediction accuracy over just using a global average.

Formally, the baseline prediction formula is:

$$\hat{r}_{ui} = \mu + b_u + b_i$$

Where:

- μ is the global average rating
- b_u is the bias for user u
- b_i is the bias for item i

This approach helps account for systematic rating patterns before applying more complex models.

0.3 Dataset Description

I use the MovieLens 100k dataset, which includes userId, movieId, and rating. Ratings range from 1 to 5 and are sparse across users and items.

0.4 Data Preparation

The dataset was loaded into R and split into training and test sets. A small subset of the data was used to verify calculations by hand. All analyses were performed in tidyverse.

```
library(googledrive)
library(readr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v purrr     1.0.2
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## -- 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 errors
```

```
# Specify the Drive file ID
file_id <- "1hwPfRD8x7QBQE5Vv4QyR7SLehfm-UH9d"

# Download the file to a temp location
temp_file <- tempfile(fileext = ".csv")
drive_download(as_id(file_id), path = temp_file, overwrite = TRUE)
```

```
## ! Using an auto-discovered, cached token.
##   To suppress this message, modify your code or options to clearly consent to
##   the use of a cached token.
##   See gargle's "Non-interactive auth" vignette for more details:
##   <https://gargle.r-lib.org/articles/non-interactive-auth.html>
## i The googledrive package is using a cached token for
##   'saloua.daouki@gmail.com'.
## File downloaded:
## * 'ratings_for_additional_users.csv' <id: 1hwPfRD8x7QBQE5Vv4QyR7SLehfm-UH9d>
## Saved locally as:
## * '/var/folders/vs/601rmjj90ynddctkwkpsbj7m0000gn/T//RtmpieCEFg/filec0e34847d45.csv'
```

```
# Read the CSV from temp file
ratings <- read_csv(temp_file)
```

```
## Rows: 4185688 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl  (3): userId, movieId, rating
```

```
## dttm (1): tstamp
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(ratings)
```

```
## # A tibble: 6 x 4
##   userId movieId rating tstamp
##    <dbl>   <dbl>  <dbl> <dttm>
## 1 393217       1    3.5 2023-01-25 19:45:46
## 2 393217       6    4   2023-02-07 21:17:19
## 3 393217      16    3.5 2023-01-25 19:50:40
## 4 393217      17    4.5 2023-01-25 19:49:45
## 5 393217      32    3   2023-01-25 19:46:01
## 6 393217      47    4   2023-01-25 15:44:44
```

```
set.seed(42)

# Add a random flag for train/test split
ratings_split <- ratings %>%
  group_by(userId) %>%
  mutate(split = sample(c("train", "test"), n(), replace = TRUE, prob = c(0.8, 0.2))) %>%
  ungroup()

train <- ratings_split %>% filter(split == "train")
test <- ratings_split %>% filter(split == "test")

head(ratings_split)
```

```
## # A tibble: 6 x 5
##   userId movieId rating tstamp              split
##    <dbl>   <dbl>  <dbl> <dttm>              <chr>
## 1 393217       1    3.5 2023-01-25 19:45:46 train
## 2 393217       6    4   2023-02-07 21:17:19 test
## 3 393217      16    3.5 2023-01-25 19:50:40 train
## 4 393217      17    4.5 2023-01-25 19:49:45 train
## 5 393217      32    3   2023-01-25 19:46:01 test
## 6 393217      47    4   2023-01-25 15:44:44 train
```

```
head(train)
```

```
## # A tibble: 6 x 5
##   userId movieId rating tstamp              split
##    <dbl>   <dbl>  <dbl> <dttm>              <chr>
## 1 393217       1    3.5 2023-01-25 19:45:46 train
## 2 393217      16    3.5 2023-01-25 19:50:40 train
## 3 393217      17    4.5 2023-01-25 19:49:45 train
## 4 393217      47    4   2023-01-25 15:44:44 train
## 5 393217     111    4   2023-01-25 17:14:10 train
## 6 393217     260    3   2023-01-25 17:10:01 train
```

```
head(test)
```

```
## # A tibble: 6 x 5
##   userId movieId rating tstamp              split
##    <dbl>   <dbl>  <dbl> <dttm>              <chr>
```

```
## 1 393217       6    4   2023-02-07 21:17:19 test
## 2 393217      32    3   2023-01-25 19:46:01 test
## 3 393217      50    4.5 2023-01-25 15:38:58 test
## 4 393217     377    3.5 2023-01-25 19:46:12 test
## 5 393217     541    4   2023-01-25 15:42:52 test
## 6 393217     778    2.5 2023-01-25 19:47:24 test
```

0.5 Global Average Rating

The global average rating from the training data is:

```
global_avg <- mean(train$rating, na.rm = TRUE)
print(global_avg)
```

```
## [1] 2.906781
```

Using this as a predictor for all unknown ratings, I calculated the RMSE on the test set:

```
library(Metrics)

# Predict global average for all test ratings
test$pred_global <- global_avg

# Compute RMSE
rmse_global <- rmse(test$rating, test$pred_global)
print(paste("Global Average RMSE:", round(rmse_global, 4)))
```

```
## [1] "Global Average RMSE: 1.7601"
```

0.6 Baseline Predictor

I calculated user and item biases based on deviations from the global average. These were merged with the test set, and the baseline predictor was calculated as:

$$GlobalAvg + UserBias + ItemBias$$

```
# User bias = avg user rating - global average
user_bias <- train %>%
  group_by(userId) %>%
  summarise(user_bias = mean(rating) - global_avg)

# Item bias = avg item rating - global average
item_bias <- train %>%
  group_by(movieId) %>%
  summarise(item_bias = mean(rating) - global_avg)

head(user_bias)
```

```
## # A tibble: 6 x 2
##   userId user_bias
##    <dbl>     <dbl>
## 1   1892     0.152
## 2   3114     0.387
## 3  12559     0.330
## 4  15893     0.260
## 5  22005     0.608
```

```
## 6  41965     0.755
```

```
head(item_bias)
```

```
## # A tibble: 6 x 2
##   movieId item_bias
##     <dbl>     <dbl>
## 1       1     0.759
## 2       2     0.409
## 3       3    -0.355
## 4       4    -1.09
## 5       5    -0.455
## 6       6     0.947
```

```
# Merge user and item bias into test set
test <- test %>%
  left_join(user_bias, by = "userId") %>%
  left_join(item_bias, by = "movieId")

# Replace missing biases with 0 (for cold-start users/items)
test$user_bias[is.na(test$user_bias)] <- 0
test$item_bias[is.na(test$item_bias)] <- 0

# Predict using baseline
test <- test %>%
  mutate(pred_baseline = global_avg + user_bias + item_bias)

# Cap predictions to valid rating range (e.g., 1 to 5)
test <- test %>%
  mutate(pred_baseline = pmin(5, pmax(1, pred_baseline)))

head(test)
```

```
## # A tibble: 6 x 9
##   userId movieId rating tstamp              split pred_global user_bias
##    <dbl>   <dbl>  <dbl> <dttm>              <chr>       <dbl>     <dbl>
## 1 393217       6    4   2023-02-07 21:17:19 test         2.91     0.917
## 2 393217      32    3   2023-01-25 19:46:01 test         2.91     0.917
## 3 393217      50    4.5 2023-01-25 15:38:58 test         2.91     0.917
## 4 393217     377    3.5 2023-01-25 19:46:12 test         2.91     0.917
## 5 393217     541    4   2023-01-25 15:42:52 test         2.91     0.917
## 6 393217     778    2.5 2023-01-25 19:47:24 test         2.91     0.917
## # i 2 more variables: item_bias <dbl>, pred_baseline <dbl>
```

0.7 RMSE for Baseline Predictor

After applying this model:

```
rmse_baseline <- rmse(test$rating, test$pred_baseline)
print(paste("Baseline Predictor RMSE:", round(rmse_baseline, 4)))
```

```
## [1] "Baseline Predictor RMSE: 1.3131"
```

This shows a significant improvement over the global average model.

0.8 RMSE Interpretation

The **Root Mean Squared Error (RMSE)** measures the average prediction error of the recommender system:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2}$$

Lower RMSE means better predictive performance. For example:

- A high RMSE (e.g., >1.0) suggests large discrepancies between predicted and actual ratings.
- A lower RMSE (e.g., 0.8 or below) suggests more accurate predictions.

In this project, the **baseline model** significantly reduced RMSE compared to the global average, meaning it captured meaningful structure in the data.

0.9 Summarize Results

```
results <- tibble(
  Model = c("Global Average", "Baseline Predictor"),
  RMSE = c(round(rmse_global, 4), round(rmse_baseline, 4))
)
print(results)
```

```
## # A tibble: 2 x 2
##   Model               RMSE
##   <chr>              <dbl>
## 1 Global Average      1.76
## 2 Baseline Predictor  1.31
```

The baseline predictor significantly reduces error by accounting for individual user and item biases. This result supports the effectiveness of incorporating basic personalization into recommender systems.

0.10 Conclusion

This analysis shows how a simple baseline recommender model can meaningfully outperform naive predictors by accounting for user and item effects. In practice, such models are useful starting points before applying more complex collaborative filtering or matrix factorization techniques.

A key enhancement to this model is **regularization**. In my current implementation, user and item biases are calculated as simple averages. However, users or movies with very few ratings can skew the bias estimates.

Regularization helps prevent **overfitting** by shrinking large biases when there's limited data.

Let:

- I_u: the set of items rated by user u
- U_i: the set of users who rated item i
- μ: the global average rating
- λ: the regularization parameter

The **regularized user bias** is given by:

$$b_u = \frac{\sum_{i \in I_u}(r_{ui} - \mu - b_i)}{\lambda + |I_u|}$$

Similarly, the **item bias** is:

$$b_i = \frac{\sum_{u \in U_i} (r_{ui} - \mu - b_u)}{\lambda + |U_i|}$$

These formulas are derived by minimizing a regularized squared error loss function, balancing model fit with complexity.

> *Note:* This formulation follows Koren & Bell (2009), who use $R(u)$ in place of I_u and $R(i)$ instead of U_i.

0.10.1 Reference

Koren, Y., & Bell, R. (2009). *Advances in Collaborative Filtering.* In F. Ricci, L. Rokach, B. Shapira, & P. Kantor (Eds.), **Recommender Systems Handbook**. Springer.
Available online via DataJobs

Other improvements include:

- Matrix factorization (e.g., SVD)
- Implicit feedback (e.g., views, clicks)
- Context-aware models

Note: All code used for data processing and analysis is available in this GitHub repository: Project1