Movie Recommender System using Baseline Predictors

Saloua Daouki

2025-05-31

Contents

0.1 Introduction

This system recommends movies to users of a streaming platform based on collaborative filtering. We leverage the MovieLens dataset, which includes user ratings on movies, to build a simple recommender system using average and bias-adjusted baseline predictors.

0.2 Dataset Description

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

0.3 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.

```r
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
```

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

```r
# 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//Rtmp3PlhMf/file12c0f667a9e26.csv'
```

```r
# 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.
```

```r
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
```

```r
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
```

```r
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
```

```r
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
```

0.4 Global Average Rating

The global average rating from the training data is:

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

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

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

```r
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"
```

0.5 Baseline Predictor

We 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:

$$GlobalAvg + UserBias + ItemBias$$

```r
# 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
```

```r
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
```

```r
# 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>
```

0.6 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.

0.7 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.

0.8 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.

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