Movie Recommendation System using MovieLens 10M

Salman Mahmood

2025-08-24

1 Introduction

This report builds a rating prediction model using the **MovieLens 10M** dataset.
Using the provided data set we will explore it using EDA and train simple baselines and regularized effect models. We will then **inference** on the **final hold-out test** set. The goal is to achieve the lowest **RMSE**.

2 Data Set Construction (Provided by Edx)

```
# Packages
if (!require(tidyverse)) install.packages("tidyverse", repos = "https://cloud.r-project.org")
if (!require(caret)) install.packages("caret", repos = "https://cloud.r-project.org")
if (!require(data.table)) install.packages("data.table", repos = "https://cloud.r-project.org")

library(tidyverse)
library(caret)
library(data.table)

options(timeout = 300)

# Download paths
dl <- "ml-10M100K.zip"
ratings_file <- "ml-10M100K/ratings.dat"
movies_file <- "ml-10M100K/movies.dat"

# Download if missing
if (!file.exists(dl)) {
    message("Downloading MovieLens 10M (this may take a few minutes)...")
    download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
        dl, mode = "wb")
}

# Unzip individual files if needed
if (!file.exists(ratings_file)) unzip(dl, ratings_file)
if (!file.exists(movies_file)) unzip(dl, movies_file)

# Read ratings
ratings <- as.data.frame(stringr::str_split(readr::read_lines(ratings_file),
    fixed("::"), simplify = TRUE), stringsAsFactors = FALSE)
```

```r
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))

# Read movies
movies <- as.data.frame(stringr::str_split(readr::read_lines(movies_file),
    fixed("::"), simplify = TRUE), stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
    mutate(movieId = as.integer(movieId))

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

# Final hold-out split (10%)
set.seed(1, sample.kind = "Rounding")  # For R >= 3.6
test_index <- createDataPartition(y = movielens$rating, times = 1,
    p = 0.1, list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

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

# Put removed rows back to edx
removed <- anti_join(temp, final_holdout_test)
edx <- bind_rows(edx, removed)

# Clean up big intermediates
rm(temp, removed, ratings, movies, movielens, test_index)

# Add movie release year to the data set
edx <- edx %>%
    mutate(release_year = suppressWarnings(as.numeric(stringr::str_extract(title,
        "(\\d{4})"))))

# Quick check
dim(edx)
```

```
## [1] 9000055       7
```

```r
dim(final_holdout_test)
```

```
## [1] 999999      6
```

```r
length(unique(edx$userId))
```

```
## [1] 69878
```

```r
length(unique(edx$movieId))
```

```
## [1] 10677
```

3 Exploratory Data Analysis

3.1 Dataset Overview

We can see that the **train/test set** has about 9 million rows and the **final validation set** has about 100000 rows. The **average** rating is 3.51 with a **standard deviation** of 1.06.

```r
tibble(n_rows_edx = nrow(edx), n_rows_final_holdout = nrow(final_holdout_test),
    n_users = n_distinct(edx$userId), n_movies = n_distinct(edx$movieId),
    rating_mean = mean(edx$rating), rating_sd = sd(edx$rating))
```

```
## # A tibble: 1 x 6
##   n_rows_edx n_rows_final_holdout n_users n_movies rating_mean rating_sd
##        <int>                <int>   <int>    <int>       <dbl>     <dbl>
## 1    9000055               999999   69878    10677        3.51      1.06
```

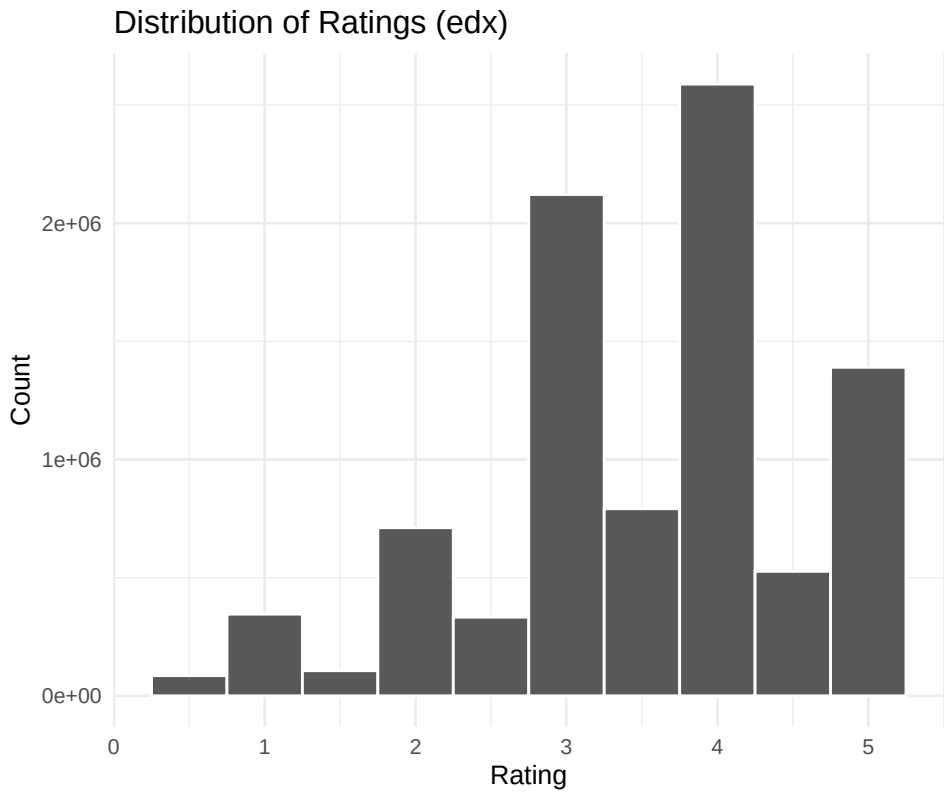
```r
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

3.2 Distribution of Ratings

From the graph below we can see that the largest group has a rating of 4, with most users preferring whole number ratings over half point ratings.

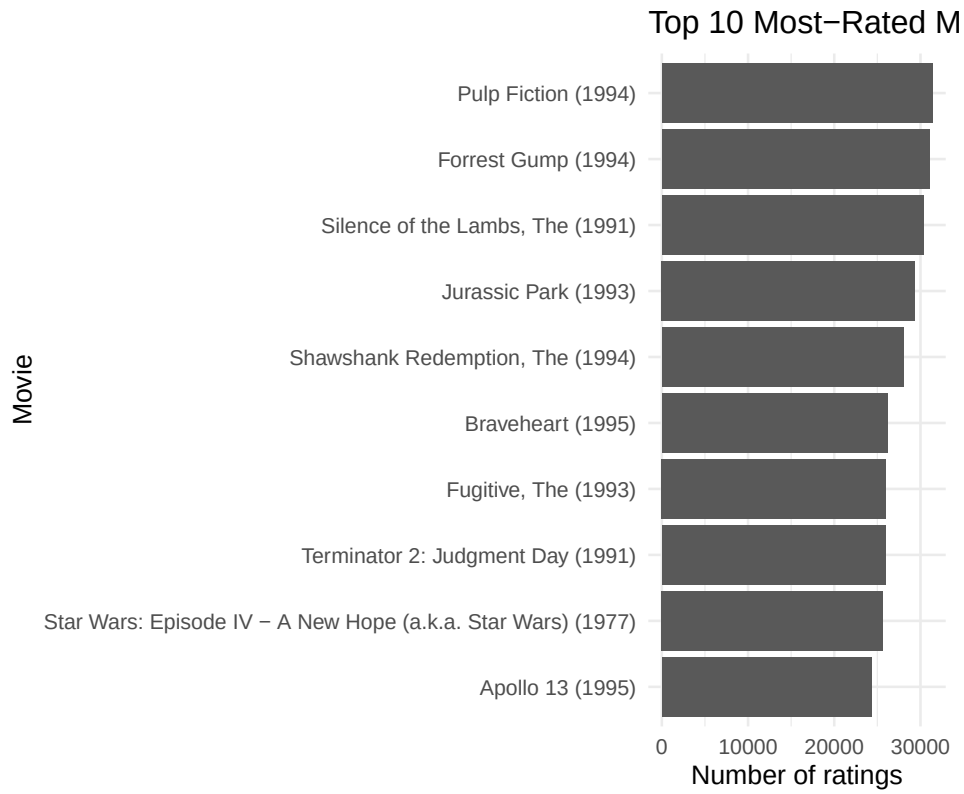
```r
edx %>%
    ggplot(aes(rating)) + geom_histogram(binwidth = 0.5, color = "white") +
    labs(title = "Distribution of Ratings (edx)", x = "Rating",
        y = "Count")
```



3.3 Top 10 Most-Rated Movies

The movie Pulp Fiction has been rated the most. The top 3 rated moves each have over 30000 ratings.

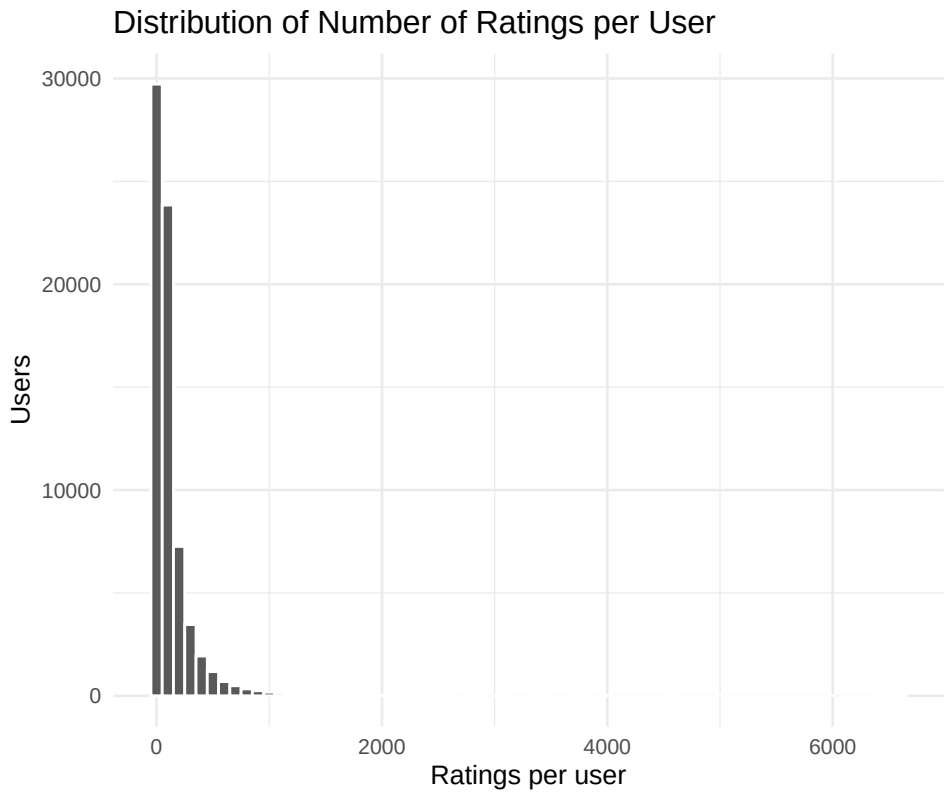
```
edx %>%
    count(title, sort = TRUE) %>%
    slice_head(n = 10) %>%
    ggplot(aes(x = reorder(title, n), y = n)) + geom_col() +
    coord_flip() + labs(title = "Top 10 Most-Rated Movies (edx)",
    x = "Movie", y = "Number of ratings")
```



3.4 Ratings per User (Un-scaled)

Using the un-scaled data we can see that majority of the users gave <= 100 ratings, the next largest group gave between 100 and 200 ratings. There is a sharp decline on the number of users that rated 300+ movies.

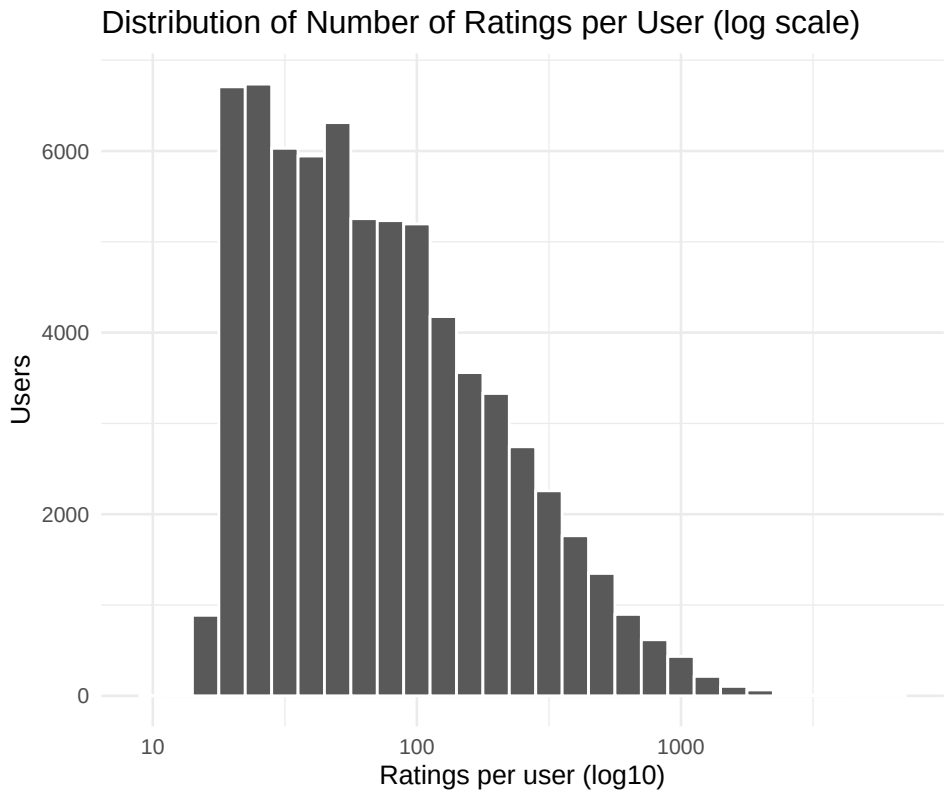
```
edx %>%
    count(userId) %>%
    ggplot(aes(n)) + geom_histogram(binwidth = 100, color = "white") +
    labs(title = "Distribution of Number of Ratings per User",
        x = "Ratings per user", y = "Users")
```



3.5 Ratings per User (Scaled)

Scaling the data to log10 allows us to see the ratings per user in a more concise manner. We can see that most of the users rated less than 100 movies with a gradual decline as the number of ratings per user increases.

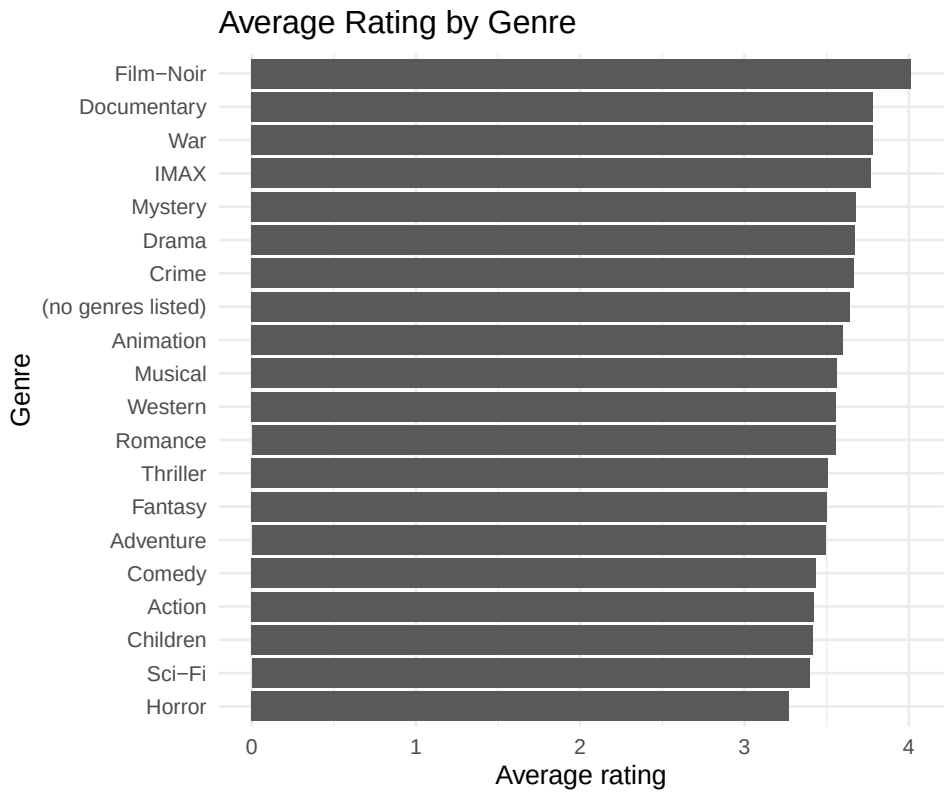
```
edx %>%
    count(userId) %>%
    ggplot(aes(n)) + geom_histogram(binwidth = 0.1, color = "white") +
    scale_x_log10() + labs(title = "Distribution of Number of Ratings per User (log scale)",
    x = "Ratings per user (log10)", y = "Users")
```



3.6 Average Rating by Genre

We can see that the category Film-Noir got the higest average rating and the category Horror has the lowest average rating.

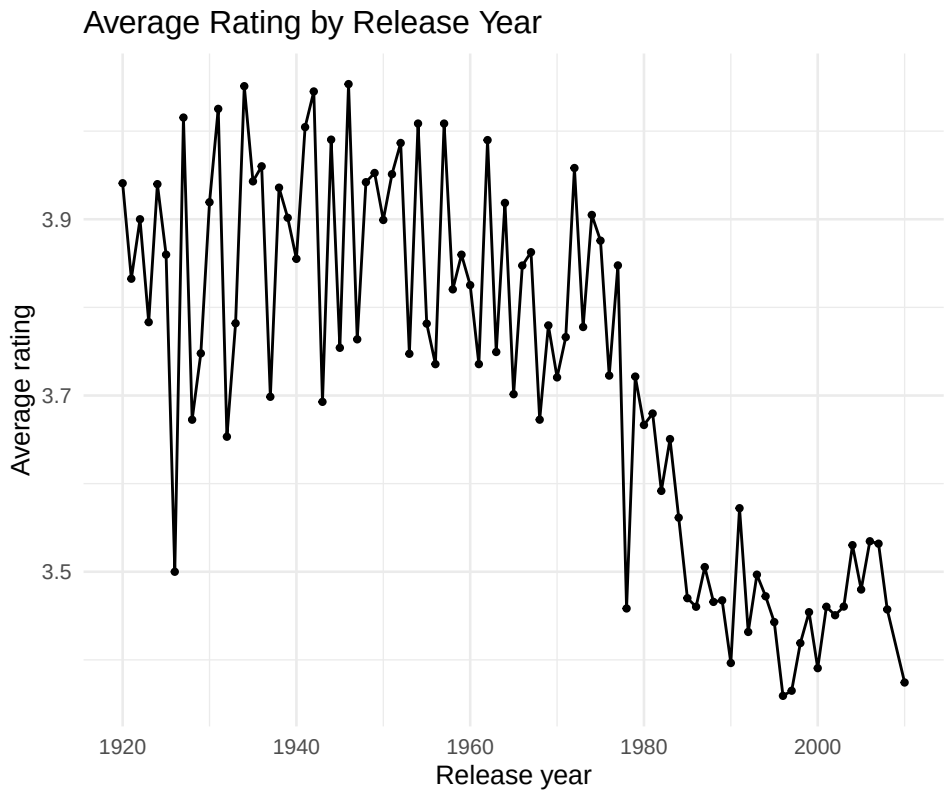
```
edx %>%
    separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarise(mean_rating = mean(rating), n = n(), .groups = "drop") %>%
    arrange(desc(mean_rating)) %>%
    ggplot(aes(x = reorder(genres, mean_rating), y = mean_rating)) +
    geom_col() + coord_flip() + labs(title = "Average Rating by Genre",
    x = "Genre", y = "Average rating")
```



3.7 Release Year vs. Average Rating

Movies released prior to 1980 have higher average ratings. Movies released after 1980 have lower average ratings.

```
edx %>%
    filter(!is.na(release_year), dplyr::between(release_year,
        1920, 2010)) %>%
    group_by(release_year) %>%
    summarise(mean_rating = mean(rating), .groups = "drop") %>%
    ggplot(aes(release_year, mean_rating)) + geom_line() + geom_point(size = 0.8) +
    labs(title = "Average Rating by Release Year", x = "Release year",
        y = "Average rating")
```



4 Modeling

We compare four models:

1. **Global Average**: predict all ratings as the overall mean
2. **Movie Effect**: global mean + movie bias
3. **Movie + User Effect**: + user bias
4. **Regularized Movie + User**: shrink biases to reduce overfitting
    - **is tuned only on a validation split from `edx`**
    - Final model is refit on full `edx` with best , then evaluated once on `final_holdout_test`

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

4.1 Train/Validation Split (for Tuning Only)

We split the data set into train/test sets, the training set is 85% of the data rows and test is the remaining 15% of the rows. We have about 7.6 million training rows and roughly 1.35 million test rows.

```r
set.seed(2, sample.kind = "Rounding")
idx <- createDataPartition(edx$rating, p = 0.85, list = FALSE)
edx_train <- edx[idx, ]
edx_val <- edx[-idx, ]

# Ensure IDs overlap
edx_val <- edx_val %>%
    semi_join(edx_train, by = "movieId") %>%
    semi_join(edx_train, by = "userId")

dim(edx_train)
```

```
## [1] 7650048       7
```

```r
dim(edx_val)
```

```
## [1] 1349979       7
```

4.2 Baselines (No Regularization)

First we are going to create a model based off of the overall average. We create a table that has the movie ID and the deviation of the movie rating from the overall rating (mean(rating - mu_hat)). We will then add user bias to the model, users have personal biases towards movies (some may be harsher then others when rating the same movie).

This method is good to get a baseline but not a very useful predictor, it only predicts if the movie is present in the table. If a new movie is used for prediction the model will return the overall average as the predicted value.

```r
mu_hat <- mean(edx_train$rating)

# Movie effects
b_i <- edx_train %>%
    group_by(movieId) %>%
    summarise(b_i = mean(rating - mu_hat), .groups = "drop")

# Predict movie effect on edx_val
pred_movie_val <- edx_val %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mu_hat + ifelse(is.na(b_i), 0, b_i)) %>%
    pull(pred)
rmse_movie_val <- RMSE(edx_val$rating, pred_movie_val)

# User effects (after movie)
b_u <- edx_train %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = mean(rating - mu_hat - b_i), .groups = "drop")

pred_user_val <- edx_val %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```
    mutate(pred = mu_hat + ifelse(is.na(b_i), 0, b_i) + ifelse(is.na(b_u),
        0, b_u)) %>%
    pull(pred)
rmse_user_val <- RMSE(edx_val$rating, pred_user_val)

tibble(Model = c("Movie Effect (val)", "Movie + User (val)"),
    RMSE = c(rmse_movie_val, rmse_user_val))
```

```
## # A tibble: 2 x 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Movie Effect (val) 0.944
## 2 Movie + User (val) 0.867
```

4.3 Regularization (Tune on edx_train → edx_val)

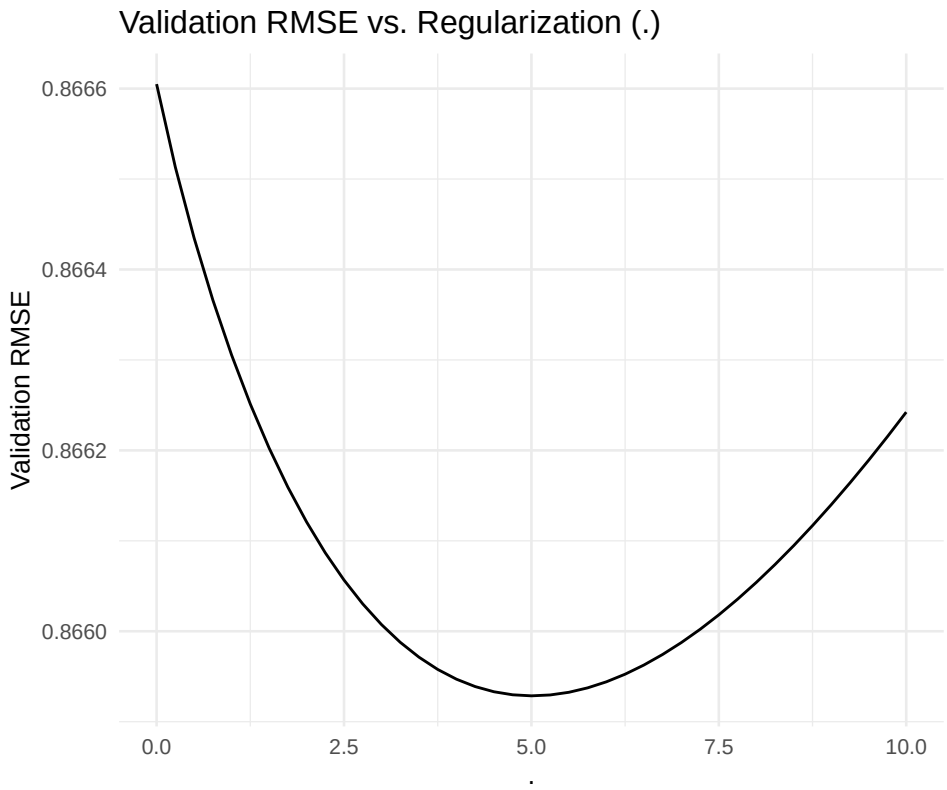
Now we try to improve our model by regularizing the data, we want to incorporate information about the number of ratings a movie has and the number of ratings by a user. We want to ensure that movies with few ratings or users with few ratings does not overfit the model. We can reduce the impact of movies with few ratings (where the deviation from the overall average is not accurate), we can also shrink the impact of user biases (If a user has rated a few movies, there score will not impact the prediciton as much).

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

rmse_by_lambda <- sapply(lambdas, function(l) {
    # Regularized movie effect
    b_i_reg <- edx_train %>%
        group_by(movieId) %>%
        summarise(b_i = sum(rating - mu_hat)/(n() + l), .groups = "drop")

    # Regularized user effect
    b_u_reg <- edx_train %>%
        left_join(b_i_reg, by = "movieId") %>%
        group_by(userId) %>%
        summarise(b_u = sum(rating - mu_hat - b_i)/(n() + l),
            .groups = "drop")

    # Predict on validation split
    pred_val <- edx_val %>%
        left_join(b_i_reg, by = "movieId") %>%
        left_join(b_u_reg, by = "userId") %>%
        mutate(pred = mu_hat + ifelse(is.na(b_i), 0, b_i) + ifelse(is.na(b_u),
            0, b_u)) %>%
        pull(pred)

    RMSE(edx_val$rating, pred_val)
})

lambda_best <- lambdas[which.min(rmse_by_lambda)]
lambda_best
```

```
## [1] 5
```

We can see that a lambda of 5 is the best value to lower the RMSE.

```
tibble(lambda = lambdas, RMSE = rmse_by_lambda) %>%
    ggplot(aes(lambda, RMSE)) + geom_line() + geom_point(size = 0.8) %>%
    labs(title = "Validation RMSE vs. Regularization ( )", x = " ",
        y = "Validation RMSE")
```



4.4 Refit Regularized Model on Full `edx` with Best

We now take the new lambda value and add it back to our original model. We will update our l parameter in our initial code to use the best lambda that we found. The model now regularizes the data with the optimal lambda to allow for the lowest RMSE.

```
mu_hat_full <- mean(edx$rating)

b_i_full <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu_hat_full)/(n() + lambda_best),
        .groups = "drop")

b_u_full <- edx %>%
    left_join(b_i_full, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu_hat_full - b_i)/(n() + lambda_best),
        .groups = "drop")
```

4.5 Final Evaluation on `final_holdout_test` (Once)

Now that we have updated our model, we can now test on the final_holdout_test data set to see how effective our model is when referencing on new un-seen data. We will test all 4 models and assess their RMSE on new data.

We can see that the model using just the global average performed the worse and un-regularized movie performed a little better. Un-regularized movie + bias and regularized movie + bias performed the same (both have the same RMSE).

```
# 1) Global mean baseline
rmse_global <- RMSE(final_holdout_test$rating, rep(mu_hat_full,
    nrow(final_holdout_test)))

# 2) Movie effect only
pred_movie_test <- final_holdout_test %>%
    left_join(b_i_full, by = "movieId") %>%
    mutate(pred = mu_hat_full + ifelse(is.na(b_i), 0, b_i)) %>%
    pull(pred)
rmse_movie_test <- RMSE(final_holdout_test$rating, pred_movie_test)

# 3) Movie + user effect (unregularized reference)
b_i_full_noreg <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = mean(rating - mu_hat_full), .groups = "drop")
b_u_full_noreg <- edx %>%
    left_join(b_i_full_noreg, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = mean(rating - mu_hat_full - b_i), .groups = "drop")

pred_user_test <- final_holdout_test %>%
    left_join(b_i_full_noreg, by = "movieId") %>%
    left_join(b_u_full_noreg, by = "userId") %>%
    mutate(pred = mu_hat_full + ifelse(is.na(b_i), 0, b_i) +
        ifelse(is.na(b_u), 0, b_u)) %>%
    pull(pred)
rmse_user_test <- RMSE(final_holdout_test$rating, pred_user_test)

# 4) Regularized movie + user
pred_reg_test <- final_holdout_test %>%
    left_join(b_i_full, by = "movieId") %>%
    left_join(b_u_full, by = "userId") %>%
    mutate(pred = mu_hat_full + ifelse(is.na(b_i), 0, b_i) +
        ifelse(is.na(b_u), 0, b_u)) %>%
    pull(pred)
rmse_reg_test <- RMSE(final_holdout_test$rating, pred_reg_test)

rmse_results <- tibble(Model = c("Global Average", "Movie Effect",
    "Movie + User Effect (unregularized)", "Regularized Movie + User"),
    RMSE = c(rmse_global, rmse_movie_test, rmse_user_test, rmse_reg_test))

rmse_results
```

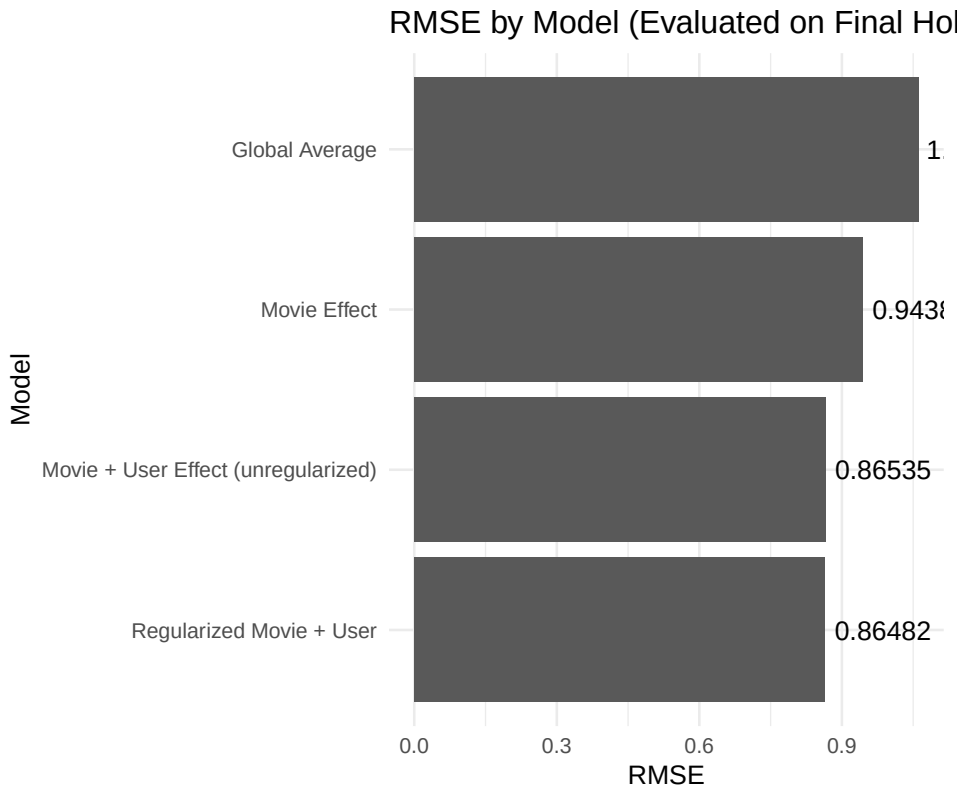
```
## # A tibble: 4 x 2
```

```
##   Model                                 RMSE
##   <chr>                                <dbl>
## 1 Global Average                       1.06
## 2 Movie Effect                         0.944
## 3 Movie + User Effect (unregularized) 0.865
## 4 Regularized Movie + User             0.865
```

5 Results

Looking further into the results, one can see that the regularized movie + user model and the un-regularized model performed very similarly but the regularized model performed slightly better.

```
rmse_results %>%
    ggplot(aes(x = reorder(Model, RMSE), y = RMSE)) + geom_col() +
    coord_flip() + geom_text(aes(label = round(RMSE, 5)), hjust = -0.1,
    size = 3.5) + labs(title = "RMSE by Model (Evaluated on Final Hold-Out Test)",
    x = "Model", y = "RMSE")
```



Final RMSE (Regularized): 0.864818
Best : 5

6 Conclusion

- Strong **movie** and **user** effects drive performance; **regularization** prevents overfitting and yields the best RMSE.

- We tuned only on a validation split from `edx`, then **retrained on full `edx`** and **evaluated once** on the **final hold-out test** set — no leakage.
- Future work: time-aware effects (rating drift), implicit feedback, and matrix factorization (e.g., `recosystem`) to push RMSE further.

7 Appendix: Session Info

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS 15.3.2
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] data.table_1.15.4 caret_7.0-1       lattice_0.22-6    lubridate_1.9.3
##  [5] forcats_1.0.0     stringr_1.5.1     dplyr_1.1.4       purrr_1.0.2
##  [9] readr_2.1.5       tidyr_1.3.1       tibble_3.2.1      ggplot2_3.5.1
## [13] tidyverse_2.0.0   knitr_1.47
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.5         xfun_0.44            recipes_1.3.1
##  [4] tzdb_0.5.0           vctrs_0.6.5          tools_4.4.0
##  [7] generics_0.1.3       stats4_4.4.0         parallel_4.4.0
## [10] fansi_1.0.6          ModelMetrics_1.2.2.2 pkgconfig_2.0.3
## [13] Matrix_1.7-0         lifecycle_1.0.4      farver_2.1.2
## [16] compiler_4.4.0       munsell_0.5.1        codetools_0.2-20
## [19] htmltools_0.5.8.1    class_7.3-22         yaml_2.3.8
## [22] prodlim_2025.04.28   crayon_1.5.2         pillar_1.9.0
## [25] MASS_7.3-60.2        gower_1.0.2          iterators_1.0.14
## [28] rpart_4.1.23         foreach_1.5.2        parallelly_1.45.0
## [31] nlme_3.1-164         lava_1.8.1           tidyselect_1.2.1
```

```
## [34] digest_0.6.35        stringi_1.8.4        future_1.58.0
## [37] reshape2_1.4.4       listenv_0.9.1        labeling_0.4.3
## [40] splines_4.4.0        fastmap_1.2.0        grid_4.4.0
## [43] colorspace_2.1-0     cli_3.6.5            magrittr_2.0.3
## [46] survival_3.5-8       utf8_1.2.4           future.apply_1.20.0
## [49] withr_3.0.0          scales_1.3.0         bit64_4.0.5
## [52] timechange_0.3.0     rmarkdown_2.27       globals_0.18.0
## [55] bit_4.0.5            nnet_7.3-19          timeDate_4041.110
## [58] hms_1.1.3            evaluate_0.24.0      hardhat_1.4.1
## [61] rlang_1.1.6          Rcpp_1.0.13-1        glue_1.7.0
## [64] formatR_1.14         pROC_1.18.5          ipred_0.9-15
## [67] vroom_1.6.5          rstudioapi_0.16.0    R6_2.5.1
## [70] plyr_1.8.9
```