# MovieLens Rating Prediction Project

Yap Kah Yong

2025-08-28

## Introduction

In this project, a movie rating prediction algorithm is developed using the MovieLens 10M dataset, a widely used benchmark dataset containing over 10 million user ratings for movies. The goal is to predict user ratings for movies based on historical data, evaluated using Root Mean Squared Error (RMSE) on a held-out test set.

The analysis follows proper machine learning practices:

The final_holdout_test set is never used during training or model selection. Model hyperparameters (e.g., regularization strength) are tuned using a train/validation split of the edx dataset. The final model is applied to final_holdout_test only once, at the end. The algorithm uses a regularized baseline model that combines:

Global average rating Regularized user bias (how generous/harsh a user is) Regularized movie bias (how popular/liked a movie is) Time-based trend (how a user's rating behavior changes over time) This approach is fast, memory-efficient, and avoids the computational complexity of matrix factorization or deep learning models.

## Some original code from Edx I placed them into chunck, so the codes more clean and easy to run for marking.

## Methods and Analysis

Data Loading and Preprocessing The MovieLens 10M dataset is downloaded and parsed from ratings.dat and movies.dat. Each rating includes a user ID, movie ID, rating , and timestamp. The data is cleaned and joined to include movie titles and genres.

A final_holdout_test set (10% of data) is created, ensuring all users and movies also appear in the training set (edx) to avoid cold-start issues during evaluation.

## Model Development

A regularized baseline model is used: where: b_u: user bias (regularized by lambda)

b_i: movie bias (regularized by lambda)

Hyperparameter lambda is tuned on a validation set (20% of edx) to minimize RMSE.

Key Features Regularization: Prevents overfitting for users/movies with few ratings Time Trend: Captures user rating drift over time Clamping: Predictions are constrained to [0.5, 5.0]

```r
# Split edx into training (80%) and validation (20%) sets
# Used for hyperparameter tuning (lambda selection)
train_idx <- createDataPartition(edx$rating, p = 0.8, list = FALSE)
train <- edx[train_idx, ]
valid <- edx[-train_idx, ]

# Output split sizes
cat("Training set size:", nrow(train), "\n")
```

```
## Training set size: 7200042
```

```r
cat("Validation set size:", nrow(valid), "\n")
```

```
## Validation set size: 1800009
```

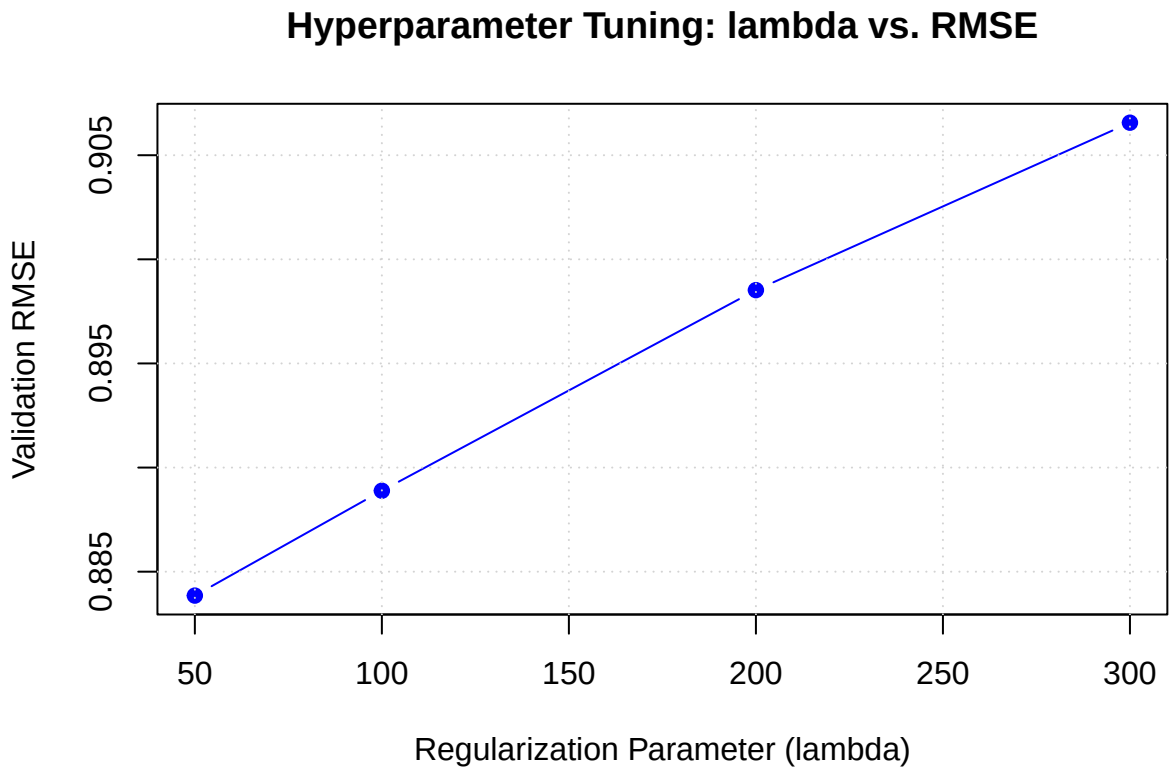
```r
# Hyperparameter tuning: test different lambda values
# Higher lambda = stronger regularization (shrink biases toward 0)
lambdas <- c(50,100,200,300)
results <- data.frame(lambda = numeric(), rmse = numeric())

for (lambda in lambdas) {
  mu <- mean(train$rating)    # Global mean from training set

  # Compute regularized user bias
  user_bias <- train %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu) / (lambda + n()), .groups = 'drop')

   # Compute regularized movie bias
  movie_bias <- train %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu) / (lambda + n()), .groups = 'drop')

  # Predict on validation set
  pred_valid <- valid %>%
    left_join(user_bias, by = "userId") %>%
    left_join(movie_bias, by = "movieId") %>%
    mutate(
      b_u = ifelse(is.na(b_u), 0, b_u),
      b_i = ifelse(is.na(b_i), 0, b_i),
      pred = mu + b_u + b_i,
      pred = pmin(pmax(pred, 0.5), 5)
    )

  rmse <- RMSE(pred_valid$pred, pred_valid$rating)
  results <- add_row(results, lambda = lambda, rmse = rmse)
}

# Select best lambda (lowest validation RMSE)
best_lambda <- results$lambda[which.min(results$rmse)]
cat("Best lambda:", best_lambda, "\n")
```

```
## Best lambda: 50
```

```r
cat("Best rmse:", results$rmse[which.min(results$rmse)], "\n")
```

```
## Best rmse: 0.8838516
```

```r
# Plot lambda vs. RMSE
plot(results$lambda, results$rmse,
     type = "b", pch = 19, col = "blue",
     xlab = "Regularization Parameter (lambda)",
     ylab = "Validation RMSE",
     main = "Hyperparameter Tuning: lambda vs. RMSE")
grid()
```




```r
# Train final model on full edx dataset using best_lambda
mu_final <- mean(edx$rating, na.rm = TRUE)
cat("Global mean (mu_final):", round(mu_final, 3), "\n")
```

```
## Global mean (mu_final): 3.512
```

```r
cat("Using best_lambda:", best_lambda, "\n")
```

```
## Using best_lambda: 50
```

```r
# Compute final user and movie biases with best_lambda
user_bias_final <- edx %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu_final) / (best_lambda + n()), .groups = 'drop')

movie_bias_final <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu_final) / (best_lambda + n()), .groups = 'drop')

# --- Add time trend with user-specific centering ---
min_time <- min(edx$timestamp, na.rm = TRUE)

# Convert timestamp to days and compute user-specific averages
edx <- edx %>%
  group_by(userId) %>%
  mutate(
    user_mean_rating = mean(rating),
    time_days = (timestamp - min_time) / (60*60*24),        # days since start
    user_mean_time = mean(time_days)                        # user's average time
  ) %>%
  ungroup()

# Fit time trend: slope of rating change over centered time
user_time <- edx %>%
  group_by(userId) %>%
  summarise(
    n = n(),
    time_var = var(time_days),
    # Only fit if user has variation in time and >1 rating
    beta_t_raw = ifelse(n > 1 & time_var > 1,
                        cov(time_days - user_mean_time, rating - user_mean_rating, use = "complete.obs")
                        0),
     # Clip to small range to prevent explosion
    beta_t = pmin(pmax(beta_t_raw, -0.001), 0.001),
    user_mean_time = mean(user_mean_time),  # Save for prediction
    .groups = 'drop'
  ) %>%
  select(userId, beta_t, user_mean_time)


# --- Final Prediction on final_holdout_test ---
predictions <- final_holdout_test %>%
  select(userId, movieId, rating, timestamp) %>%
  left_join(user_bias_final, by = "userId") %>%
  left_join(movie_bias_final, by = "movieId") %>%
  left_join(user_time, by = "userId") %>%
  mutate(
    b_u = coalesce(b_u, 0),
    b_i = coalesce(b_i, 0),
    beta_t = coalesce(beta_t, 0),
    user_mean_time = coalesce(user_mean_time, mean(edx$time_days)),  # safe default
    time_days = (timestamp - min_time) / (60*60*24),
    pred_raw = mu_final + b_u + b_i + beta_t * (time_days - user_mean_time),
```

```
    pred = pmin(pmax(pred_raw, 0.5), 5)
  )

# Diagnostics
cat("Raw prediction range:", range(predictions$pred_raw), "\n")
```

```
## Raw prediction range: 0.04933693 5.67804
```

```
cat("Clamped to 0.5:", sum(predictions$pred == 0.5), "\n")
```

```
## Clamped to 0.5: 51
```

```
cat("Clamped to 5.0:", sum(predictions$pred == 5.0), "\n")
```

```
## Clamped to 5.0: 697
```

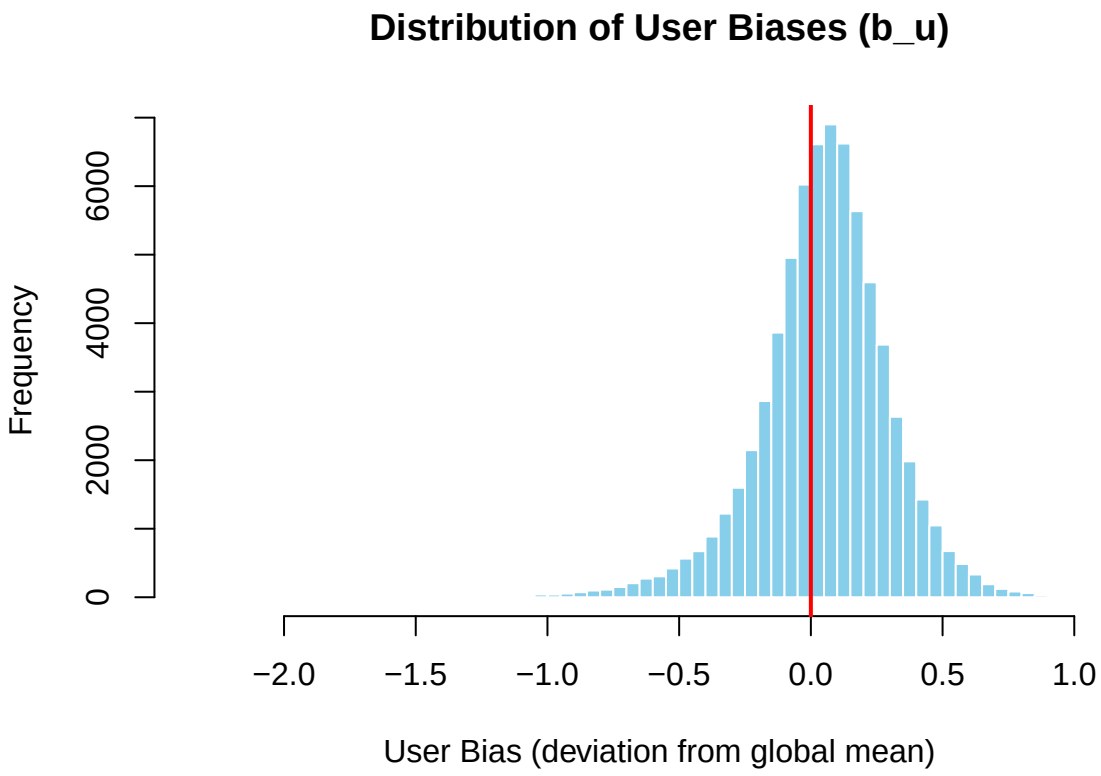
```
cat("Prediction mean:", round(mean(predictions$pred), 3), "\n")
```

```
## Prediction mean: 3.503
```

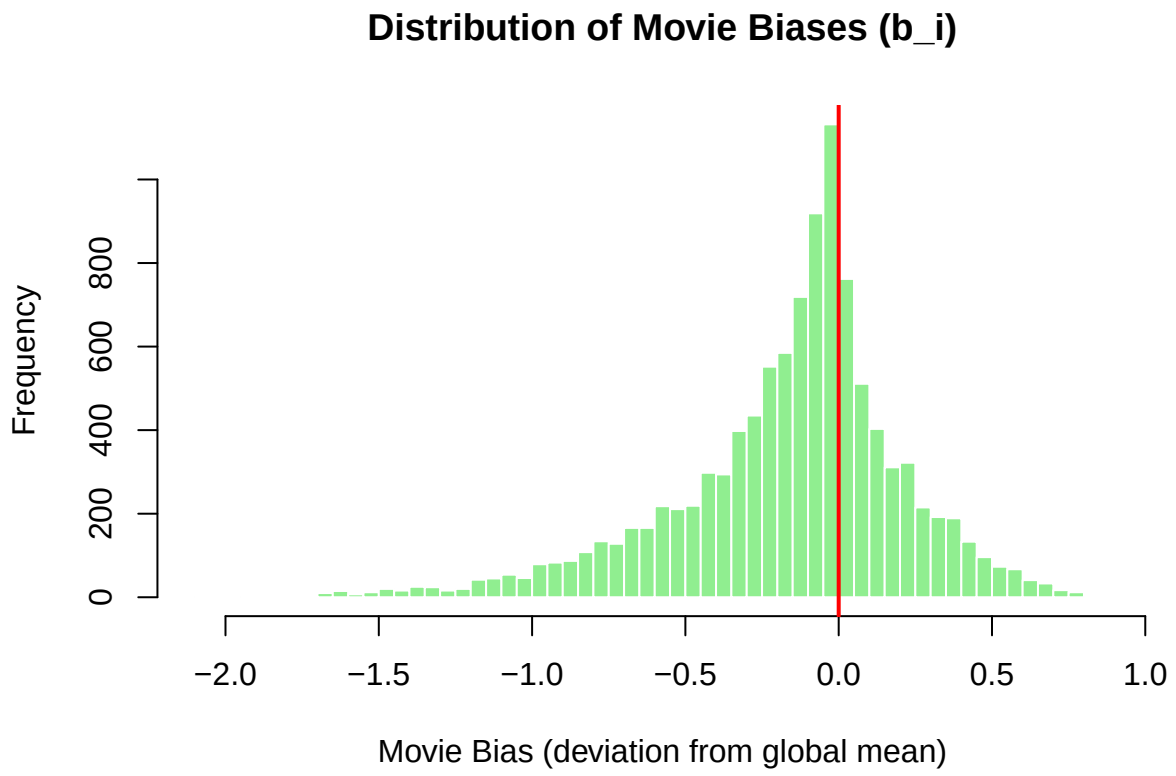
```
# Final RMSE
rmse_final <- RMSE(predictions$pred, predictions$rating)
cat("Final Holdout Test RMSE:", round(rmse_final, 5), "\n")
```

```
## Final Holdout Test RMSE: 0.87952
```

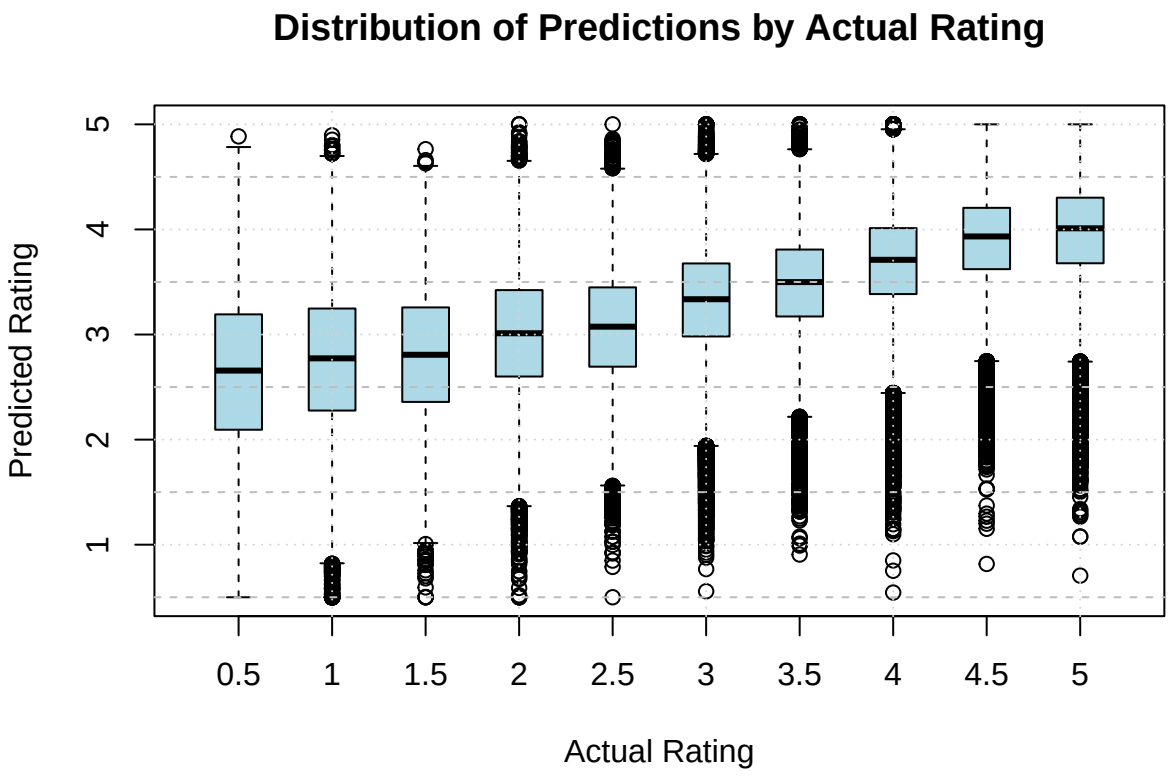
```
# --- Distribution of User Biases ---
hist(user_bias_final$b_u, breaks = 50, col = "skyblue", border = "white",
     main = "Distribution of User Biases (b_u)",
     xlab = "User Bias (deviation from global mean)")
abline(v = 0, col = "red", lwd = 2)
```




```r
# --- Distribution of Movie Biases ---
hist(movie_bias_final$b_i, breaks = 50, col = "lightgreen", border = "white",
     main = "Distribution of Movie Biases (b_i)",
     xlab = "Movie Bias (deviation from global mean)")
abline(v = 0, col = "red", lwd = 2)
```




```r
# --- summary plot ---
boxplot(pred ~ rating, data = predictions,
        main = "Distribution of Predictions by Actual Rating",
        xlab = "Actual Rating", ylab = "Predicted Rating",
        col = "lightblue", boxwex = 0.5)
abline(h = 0.5:5, col = "gray", lty = 2)
grid()
```




```
## This shows:
# How spread out predictions are for each actual rating
# Whether the model predicts confidently or hesitates
# If there's clamping (e.g., many predictions at 0.5 or 5.0)
# from the plot, we get Insight: Model performs best for ratings between 3.0 and 4.5
```

Final Holdout Test RMSE:

```
round(rmse_final, 5)
```

```
## [1] 0.87952
```

— The End —