MovieLens Recommendation System

Dong Hyun Kang

2025-04-17

```r
if (!require(tidyverse)) install.packages("tidyverse")
if (!require(caret)) install.packages("caret")
if (!require(recosystem)) install.packages("recosystem")
if (!require(lubridate)) install.packages("lubridate")
if (!require(future.apply)) install.packages("future.apply")

library(tidyverse)
library(caret)
library(recosystem)
library(lubridate)
library(future.apply)
```

Introduction

This report analyzes a subset of the MovieLens 10M dataset Harper and Konstan (2015) and aims to build a recommendation system that minimizes RMSE on a final hold-out test set. We explore several collaborative filtering models, including regularized bias models and matrix factorization via recosystem package Qiu (2023), and compare their performance.

```r
# Data Loading & Preprocessing
# Data Loading & Preprocessing (DO NOT MODIFY)
# Download and extract the dataset
# Provided by the course and should not be modified
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)
```

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

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

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

# Create edx and final_holdout_test sets (as per instructions)
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Exploratory Data Analysis (EDA)

```r
# EDA with Correlation Analysis
# Convert UNIX timestamp to POSIXct date format for time-based analysis
edx <- edx %>% mutate(rating_date = as_datetime(timestamp))

# Summary statistics for overall understanding of dataset
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres           rating_date
```

```
##  Length:9000055     Length:9000055     Min.   :1995-01-09 11:46:49.00
##  Class :character   Class :character   1st Qu.:2000-01-01 23:11:23.00
##  Mode  :character   Mode  :character   Median :2002-10-24 21:11:58.00
##                                        Mean   :2002-09-21 13:45:07.38
##                                        3rd Qu.:2005-09-15 02:21:21.00
##                                        Max.   :2009-01-05 05:02:16.00
```

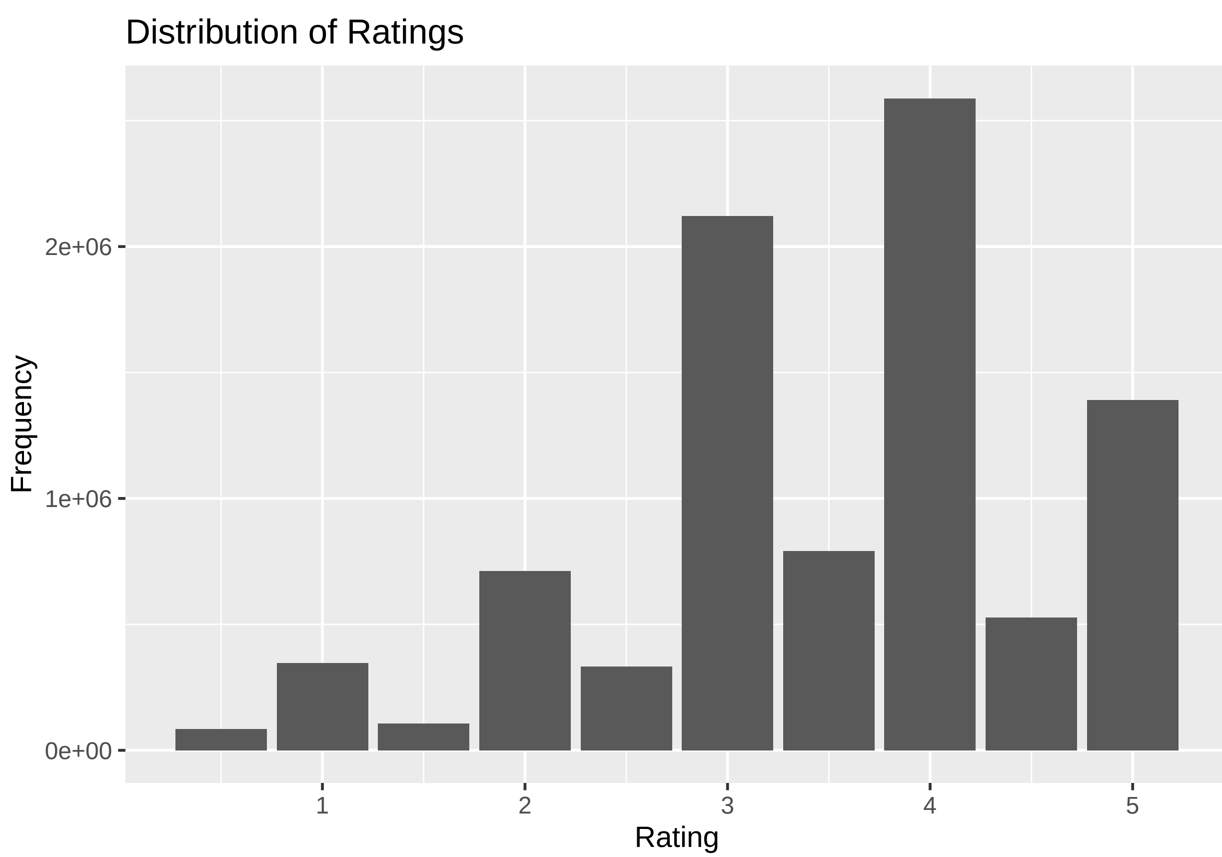
```r
cat("Number of unique users:", n_distinct(edx$userId), "
")
```

```
## Number of unique users: 69878
```

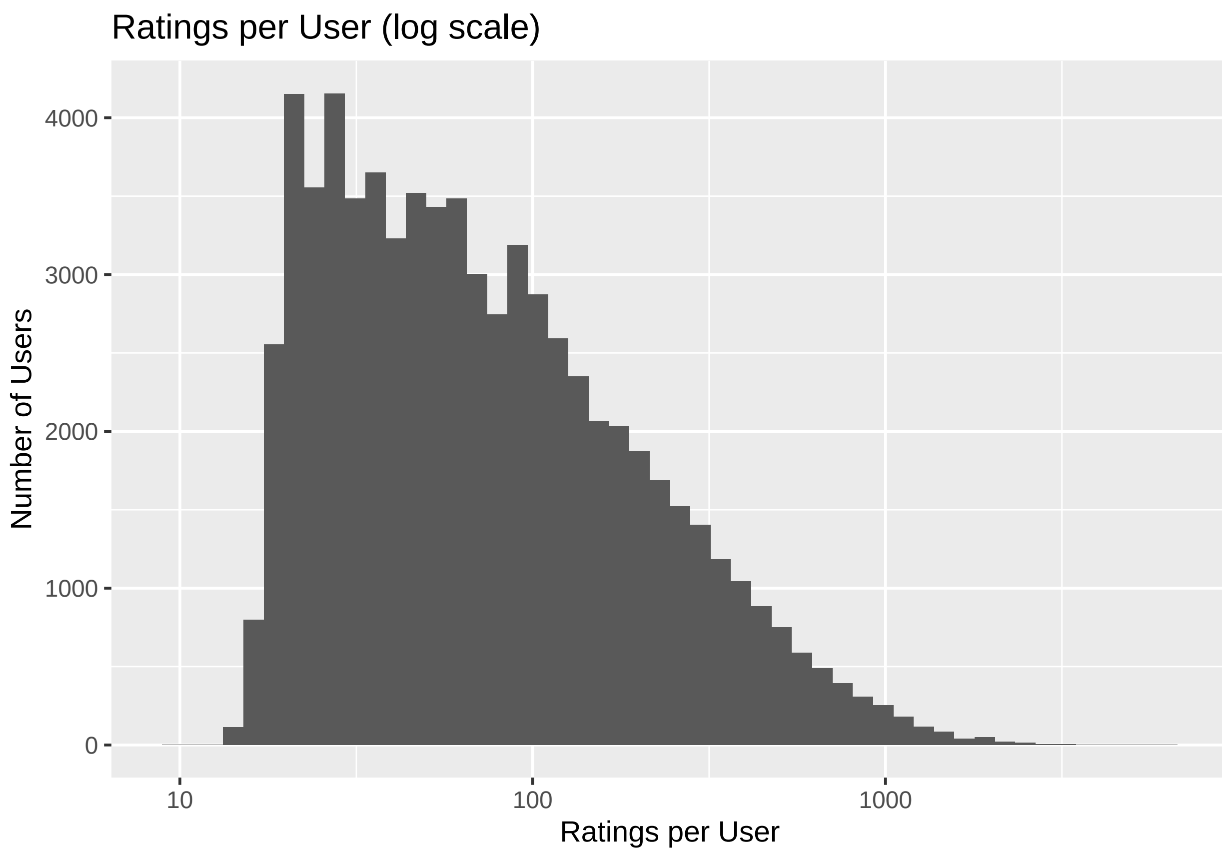
```r
cat("Number of unique movies:", n_distinct(edx$movieId), "
")
```

```
## Number of unique movies: 10677
```

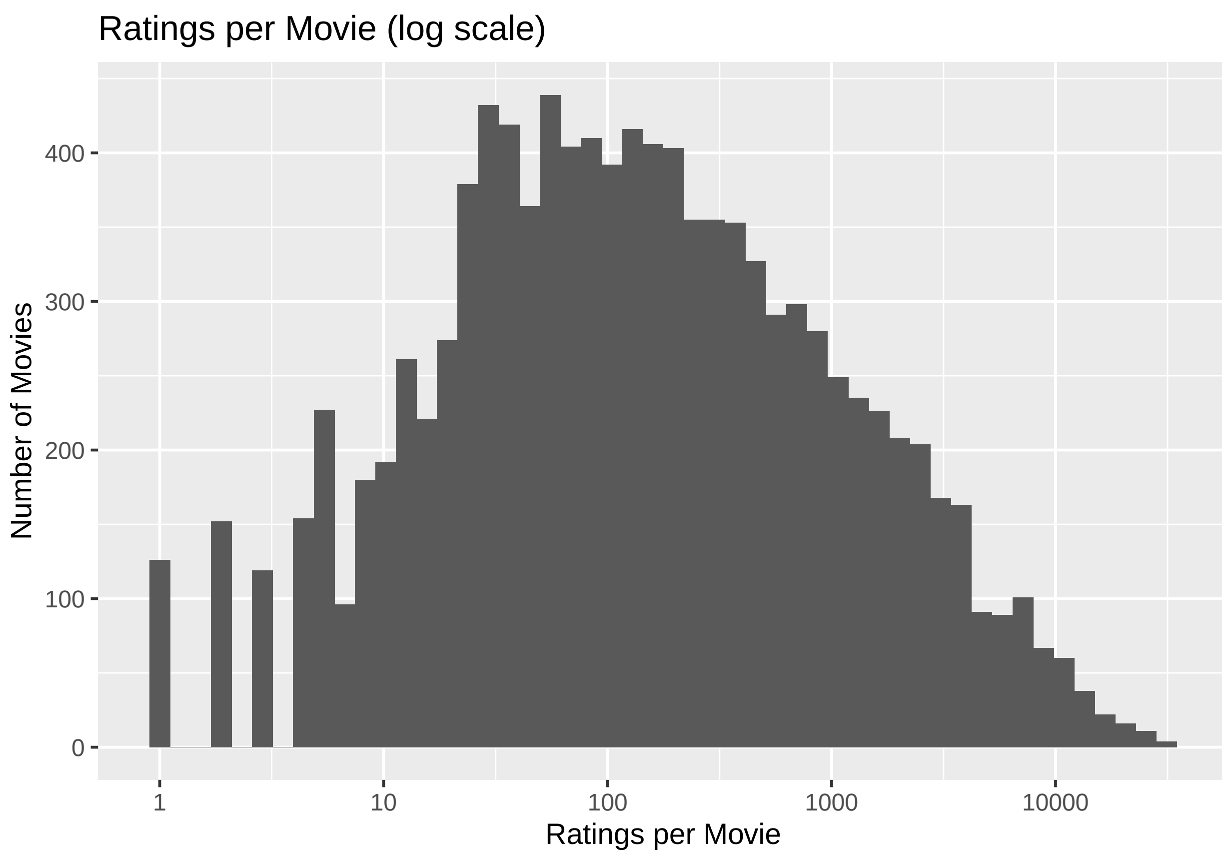
```r
# Distribution of ratings (e.g., is it left-skewed? Right-skewed?)
edx %>% ggplot(aes(x = rating)) +
  geom_bar() +
  labs(title = "Distribution of Ratings", x = "Rating", y = "Frequency")
```



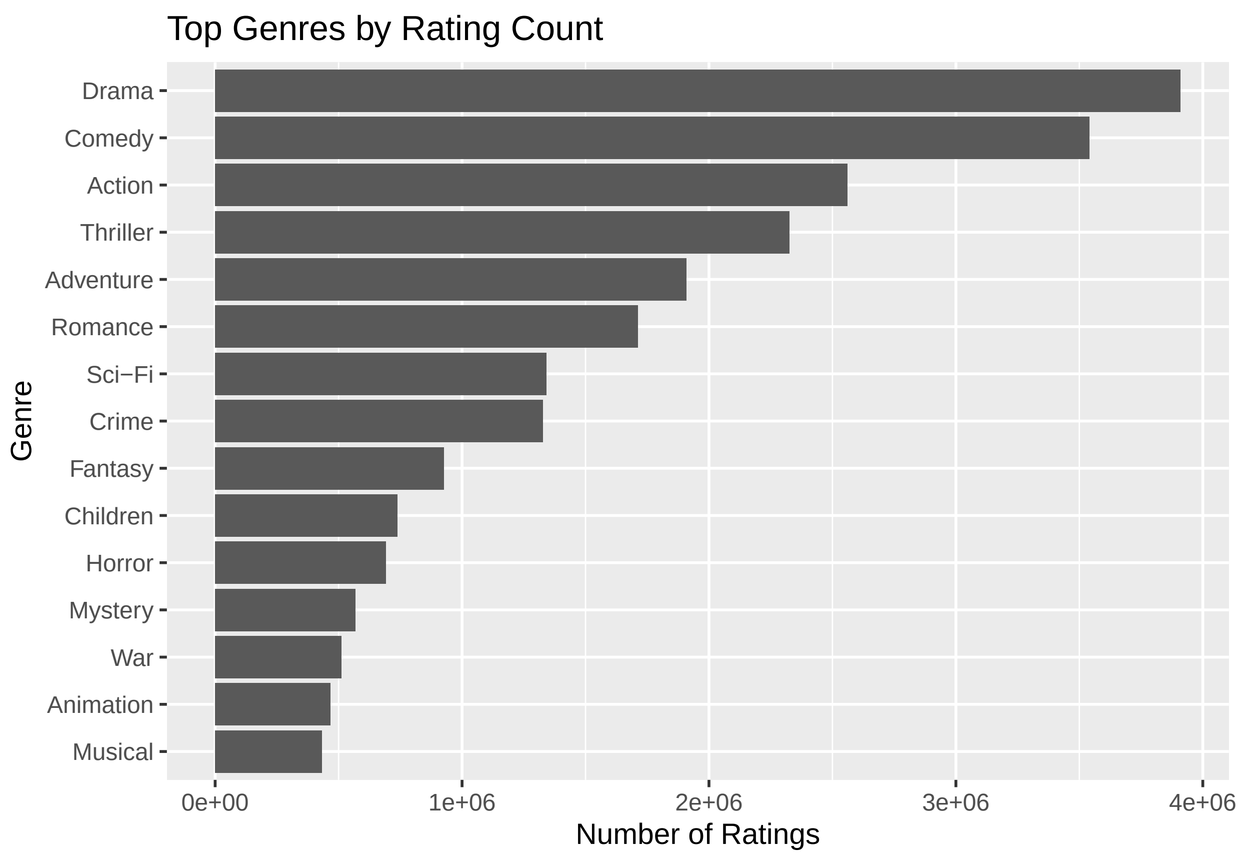
```r
# Number of ratings per user to identify active vs. infrequent users
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50) +
  scale_x_log10() +
  labs(title = "Ratings per User (log scale)", x = "Ratings per User", y = "Number of Users")
```



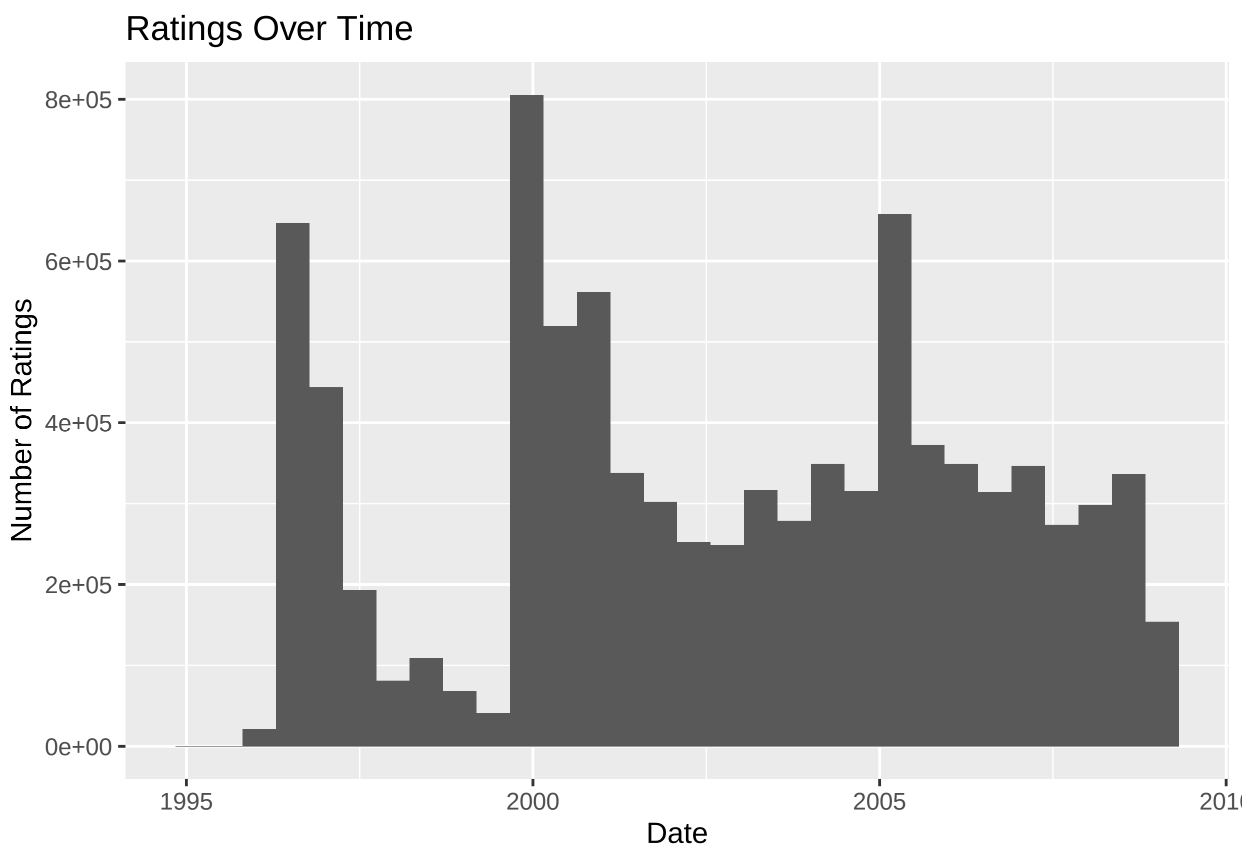

```r
# Number of ratings per movie to identify popular vs. obscure movies
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 50) +
  scale_x_log10() +
  labs(title = "Ratings per Movie (log scale)", x = "Ratings per Movie", y = "Number of Movies")
```




```r
# Top 15 most frequently rated genres
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  count(genres, sort = TRUE) %>%
  top_n(15) %>%
  ggplot(aes(x = reorder(genres, n), y = n)) +
  geom_col() +
  coord_flip() +
  labs(title = "Top Genres by Rating Count", x = "Genre", y = "Number of Ratings")
```




```r
# Rating frequency over time (to examine time-based trends)
edx %>% ggplot(aes(rating_date)) +
  geom_histogram(bins = 30) +
  labs(title = "Ratings Over Time", x = "Date", y = "Number of Ratings")
```




```
# Numeric correlation between timestamp and rating
cor(as.numeric(edx$timestamp), edx$rating, use = "complete.obs")
```

```
## [1] -0.03473968
```

```
# Correlation between average user rating and individual ratings
user_avg <- edx %>% group_by(userId) %>% summarize(user_mean = mean(rating), user_count = n())
edx_user <- edx %>% left_join(user_avg, by = "userId")
cor(edx_user$user_mean, edx_user$rating)
```

```
## [1] 0.4038697
```

```
# Correlation between average movie rating and individual ratings
movie_avg <- edx %>% group_by(movieId) %>% summarize(movie_mean = mean(rating), movie_count = n())
edx_movie <- edx %>% left_join(movie_avg, by = "movieId")
cor(edx_movie$movie_mean, edx_movie$rating)
```

```
## [1] 0.4584323
```

```
# Split edx into Train/Test for modeling
split_by_user <- function(user_df, p = 0.8) {
  n <- nrow(user_df)
  if (n == 1) {
```

```
    list(train = user_df, test = NULL)
  } else {
    idx <- createDataPartition(1:n, p = p, list = FALSE)
    list(train = user_df[idx, ], test = user_df[-idx, ])
  }
}

splits <- edx %>% group_by(userId) %>% group_split() %>% map(split_by_user)
edx_train <- map_dfr(splits, "train")
edx_test <- map_dfr(splits, "test") %>% filter(!is.null(.))

edx_test <- edx_test %>% filter(movieId %in% edx_train$movieId, userId %in% edx_train$userId)
```

Modeling

```
# Modeling

mu <- mean(edx_train$rating)  # Global average

# Naive Model
rmse_baseline <- RMSE(edx_test$rating, mu)
print(rmse_baseline)
```

```
## [1] 1.059152
```

```
# Movie Effect Model
b_i <- edx_train %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
pred_movie <- edx_test %>% left_join(b_i, by = "movieId") %>% mutate(pred = mu + b_i)
rmse_movie <- RMSE(pred_movie$rating, pred_movie$pred)
print(rmse_movie)
```

```
## [1] 0.9410797
```

```
# Movie + User Effect Model
b_u <- edx_train %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>% summarize(b_u = mean(rating
pred_user_movie <- edx_test %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mu
rmse_user_movie <- RMSE(pred_user_movie$rating, pred_user_movie$pred)
print(rmse_user_movie)
```

```
## [1] 0.86296
```

```
# Regularized Model
lambdas <- seq(0, 10, 0.25)
# Increase limit for large objects used in parallel processing
options(future.globals.maxSize = 2 * 1024^3)
plan(multisession, workers = parallel::detectCores() - 1)
rmse_results <- future_sapply(lambdas, function(lambda) {
  b_i <- edx_train %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n() + lambda))
```

```r
  b_u <- edx_train %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>% summarize(b_u = sum(ratin
  pred <- edx_test %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate(pred
  RMSE(edx_test$rating, pred)
})
plan(sequential)
best_lambda <- lambdas[which.min(rmse_results)]
print(best_lambda)
```

```
## [1] 4.75
```

```r
b_i <- edx_train %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu)/(n() + best_lambda))
b_u <- edx_train %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>% summarize(b_u = sum(rating
pred_reg <- edx_test %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate(pre
rmse_regularized <- RMSE(pred_reg$rating, pred_reg$pred)
print(rmse_regularized)
```

```
## [1] 0.8624198
```

```r
# Manual + MF Residual (Hybrid Model)
edx_train_resid <- edx_train %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mu
train_resid <- data_memory(user_index = edx_train_resid$userId, item_index = edx_train_resid$movieId, ra

r_hybrid <- Reco()
opts_hybrid <- r_hybrid$tune(train_resid, opts = list(dim = c(10, 20, 30), costp_l2 = c(0.01, 0.1), cost
r_hybrid$train(train_resid, opts = c(opts_hybrid$min, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.8603   5.6545e+06
##    1       0.8374   5.2573e+06
##    2       0.8204   5.1090e+06
##    3       0.8042   4.9837e+06
##    4       0.7883   4.8643e+06
##    5       0.7742   4.7604e+06
##    6       0.7620   4.6727e+06
##    7       0.7517   4.6000e+06
##    8       0.7429   4.5408e+06
##    9       0.7353   4.4914e+06
##   10       0.7287   4.4495e+06
##   11       0.7227   4.4097e+06
##   12       0.7176   4.3777e+06
##   13       0.7129   4.3494e+06
##   14       0.7088   4.3242e+06
##   15       0.7050   4.3017e+06
##   16       0.7016   4.2805e+06
##   17       0.6985   4.2624e+06
##   18       0.6956   4.2455e+06
##   19       0.6930   4.2303e+06
```

```r
test_mf_hybrid <- data_memory(user_index = edx_test$userId, item_index = edx_test$movieId)
pred_resid <- r_hybrid$predict(test_mf_hybrid, out_memory())
pred_hybrid <- edx_test %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>% mutate
rmse_hybrid <- RMSE(pred_hybrid$rating, pred_hybrid$pred)
print(rmse_hybrid)
```

```
## [1] 0.792098
```

```r
# Pure MF Model
train_data <- data_memory(user_index = edx_train$userId, item_index = edx_train$movieId, rating = edx_t
test_data <- data_memory(user_index = edx_test$userId, item_index = edx_test$movieId)

r_mf <- Reco()
opts_mf <- r_mf$tune(train_data, opts = list(dim = c(10, 20, 30), costp_l2 = c(0.01, 0.1), costq_l2 = c
r_mf$train(train_data, opts = c(opts_mf$min, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9925   1.0164e+07
##    1       0.8784   8.1948e+06
##    2       0.8464   7.6073e+06
##    3       0.8242   7.2524e+06
##    4       0.8073   6.9994e+06
##    5       0.7946   6.8170e+06
##    6       0.7841   6.6797e+06
##    7       0.7750   6.5681e+06
##    8       0.7671   6.4760e+06
##    9       0.7601   6.3974e+06
##   10       0.7541   6.3318e+06
##   11       0.7487   6.2750e+06
##   12       0.7437   6.2243e+06
##   13       0.7393   6.1834e+06
##   14       0.7352   6.1422e+06
##   15       0.7315   6.1109e+06
##   16       0.7281   6.0796e+06
##   17       0.7250   6.0502e+06
##   18       0.7221   6.0258e+06
##   19       0.7194   6.0024e+06
```

```r
pred_mf <- r_mf$predict(test_data, out_memory())
rmse_mf <- RMSE(edx_test$rating, pred_mf)
print(rmse_mf)
```

```
## [1] 0.786785
```

Results

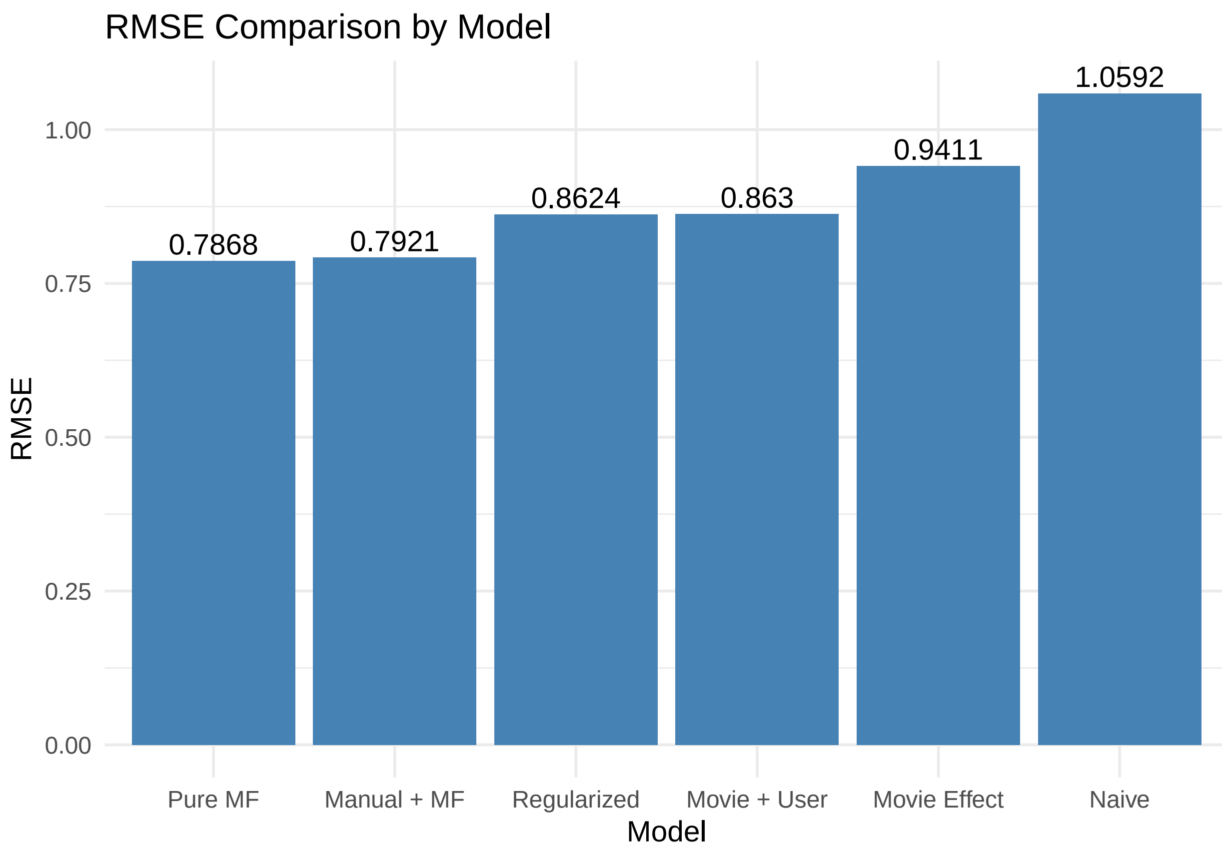
```r
# Compare all RMSEs
rmse_df <- data.frame(
  Model = c("Naive", "Movie Effect", "Movie + User", "Regularized", "Manual + MF", "Pure MF"),
  RMSE = c(rmse_baseline, rmse_movie, rmse_user_movie, rmse_regularized, rmse_hybrid, rmse_mf)
)

print(rmse_df)
```

```
##            Model      RMSE
```

```
## 1        Naive 1.0591516
## 2 Movie Effect 0.9410797
## 3 Movie + User 0.8629600
## 4  Regularized 0.8624198
## 5  Manual + MF 0.7920980
## 6      Pure MF 0.7867850
```

```r
ggplot(rmse_df, aes(x = reorder(Model, RMSE), y = RMSE)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = round(RMSE, 4)), vjust = -0.3) +
  labs(title = "RMSE Comparison by Model", x = "Model", y = "RMSE") +
  theme_minimal()
```



Final Model Evaluation

```r
# Retrain Pure MF Model on full edx and testout on final_holdout_test
train_data_final <- data_memory(user_index = edx$userId, item_index = edx$movieId, rating = edx$rating)
r_final <- Reco()
opts_final <- r_final$tune(train_data_final, opts = list(dim = c(10, 20, 30), costp_l2 = c(0.01, 0.1),
r_final$train(train_data_final, opts = c(opts_final$min, niter = 20))
```

```
## iter      tr_rmse          obj
```

```
##    0        0.9731   1.2043e+07
##    1        0.8732   9.8941e+06
##    2        0.8393   9.1872e+06
##    3        0.8170   8.7545e+06
##    4        0.8009   8.4697e+06
##    5        0.7888   8.2704e+06
##    6        0.7789   8.1168e+06
##    7        0.7708   7.9948e+06
##    8        0.7640   7.8958e+06
##    9        0.7582   7.8198e+06
##   10        0.7531   7.7538e+06
##   11        0.7486   7.6948e+06
##   12        0.7446   7.6458e+06
##   13        0.7410   7.6046e+06
##   14        0.7376   7.5662e+06
##   15        0.7346   7.5319e+06
##   16        0.7318   7.5024e+06
##   17        0.7292   7.4751e+06
##   18        0.7268   7.4487e+06
##   19        0.7245   7.4265e+06
```

```r
final_data <- data_memory(user_index = final_holdout_test$userId, item_index = final_holdout_test$moviel
pred_final <- r_final$predict(final_data, out_memory())
rmse_final <- RMSE(final_holdout_test$rating, pred_final)
print(paste("Final RMSE on final_holdout_test:", round(rmse_final, 5)))
```

```
## [1] "Final RMSE on final_holdout_test: 0.7829"
```

Conclusion

We found that matrix factorization achieved the best performance, although combining manual bias terms and MF residuals also gave strong results. Future work may explore time-aware models or hybrid systems incorporating content-based filtering.

References

Harper, F. Maxwell, and Joseph A Konstan. 2015. "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5 (4): 1–19.

Qiu, Yixuan. 2023. *Recosystem: Recommender System Using Matrix Factorization.* https://CRAN.R-project.org/package=recosystem.