Movie Recommendation System

Dmitrii Krotov

2024-06-02

Contents

Introduction

Project Overview

This report details the development of a movie recommendation system using the MovieLens dataset. The original data was obtained from the MovieLens 10M dataset, which is widely used in recommendation system research. The data can be accessed from this page.

The primary goal of this project is to build a recommendation system that predicts movie ratings based on historical user ratings. By evaluating various models, we aim to determine the best approach for predicting user preferences.

The key steps performed in this project include:

- Data loading and cleaning
- Exploratory data analysis (EDA)
- Model development, including baseline and advanced models
- Model evaluation using cross-validation and a final holdout test set

By following these steps, we aim to develop an accurate and reliable movie recommendation system that can predict user ratings for movies they have not yet seen, enhancing their movie-watching experience. The performance of the models was evaluated using Root Mean Squared Error (RMSE).

Dataset Description

The MovieLens dataset was obtained from the GroupLens research lab at the University of Minnesota. It contains millions of ratings for thousands of movies provided by a large number of users. Specifically, the dataset includes:

- 10 million ratings
- 72 000 users
- 10 000 movies

The dataset is structured to facilitate research in collaborative filtering, an essential technique in recommendation systems. Its size and complexity make it an ideal choice for developing and testing recommendation algorithms. The data includes user IDs, movie IDs, ratings, and timestamps, along with movie titles and genres.

The importance of the MovieLens dataset in recommendation system research lies in its comprehensiveness and the breadth of user-movie interactions it captures, which allows for robust modeling and validation of various recommendation techniques.

Evaluation Metric

To evaluate the performance of the recommendation models, we used Root Mean Squared Error (RMSE). RMSE is a standard metric for assessing the accuracy of predictions in regression tasks, including recommendation systems. It measures the average magnitude of the errors between predicted and actual ratings, providing a clear indication of the model's prediction accuracy.

RMSE was chosen for this project because it is straightforward to interpret and penalizes larger errors more than smaller ones, making it suitable for capturing the nuances of user preferences in movie ratings. By minimizing RMSE, we aim to develop a model that provides precise and reliable recommendations.

1 Methods and Analysis

We will use the following libraries to assist with data manipulation, visualization, and model building:

```r
library(tidyverse)
library(caret)
library(recosystem)
```

1.1 Data Loading and Cleaning

To begin, we loaded the MovieLens dataset, which consists of user ratings for various movies. The dataset was then split into two parts: the `edx` set, used for developing and training the models, and the `final_holdout_test` set, used for the final evaluation of the model's performance.

```r
options(timeout = 120)

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

# Unzip the dataset if the ratings file does not exist
ratings_file <- "ml-10M100K/ratings.dat"
if (!file.exists(ratings_file)) {
  unzip(dl, files = ratings_file)
}

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

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))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1,
                                  list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Data cleaning procedures were applied to handle any missing or anomalous values, ensuring consistency and reliability in our analysis.

```
# Check for missing values
missing_values <- sapply(edx, function(x) sum(is.na(x)))
missing_values
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```
# Check for duplicates
num_duplicates <- sum(duplicated(edx))
num_duplicates
```

```
## [1] 0
```

```r
# Data Cleaning
edx <- edx %>%
  mutate(movieId = as.integer(movieId),
         rating = as.numeric(rating)) %>%
  drop_na() %>%
  distinct()

# Verify changes
dim(edx)
```

```
## [1] 9000055       6
```

```r
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
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```r
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <int> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

1.2 Exploratory Data Analysis (EDA)

Before performing EDA, we reduced the dataset to focus on active users and popular movies. Specifically, we included only users who have rated at least 50 movies and movies that have been rated at least 50 times. This reduction ensures that our analysis and subsequent models are based on meaningful and substantial user interactions.

```r
# Check the number of unique users and unique movies
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

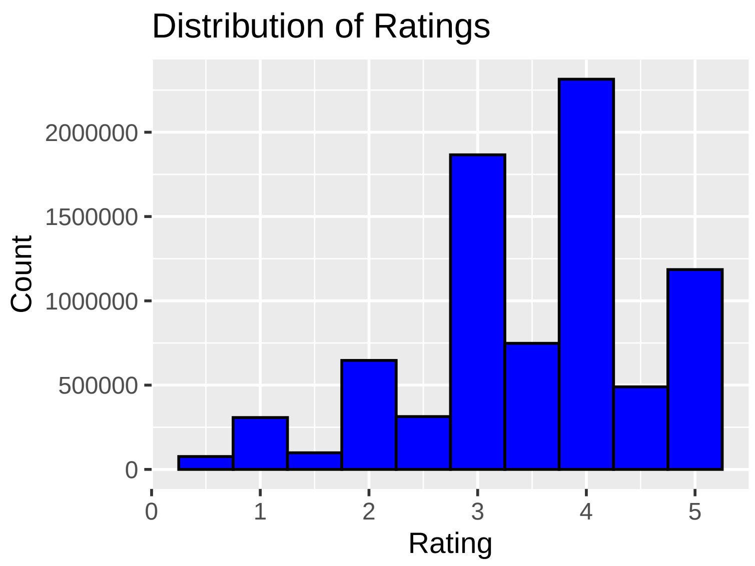
```
##   n_users n_movies
## 1   69878    10677
```

```r
# Filter users who have rated at least 50 movies
user_ratings <- edx %>%
  group_by(userId) %>%
  summarize(count = n()) %>%
  filter(count >= 50)

# Filter movies that have been rated at least 50 times
movie_ratings <- edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count >= 50)

# Reduce the edx dataset
edx_filtered <- edx %>%
  semi_join(user_ratings, by = "userId") %>%
  semi_join(movie_ratings, by = "movieId")

# Verify the reduction
edx_filtered %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   40661     7081
```

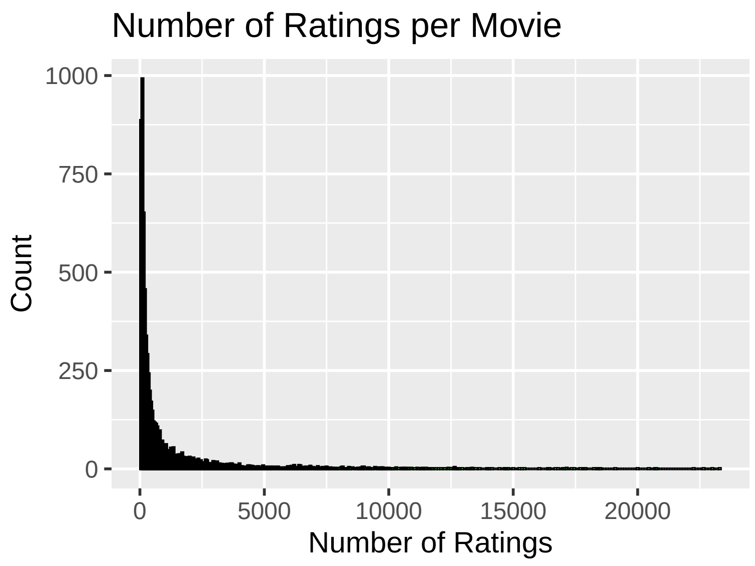
1.2.1 Distribution of Ratings

We visualized and summarized the distribution of movie ratings to understand the overall rating behavior of users. The histogram shows that the majority of ratings fall between 3 and 5 stars, with a significant peak at 4 stars. This suggests that users tend to rate movies favorably, with relatively few ratings below 3 stars. This could indicate a tendency for users to watch and rate movies they expect to enjoy or a general positive bias in rating behavior. This positive bias in ratings is a common phenomenon in user-generated content and should be considered when developing recommendation algorithms.



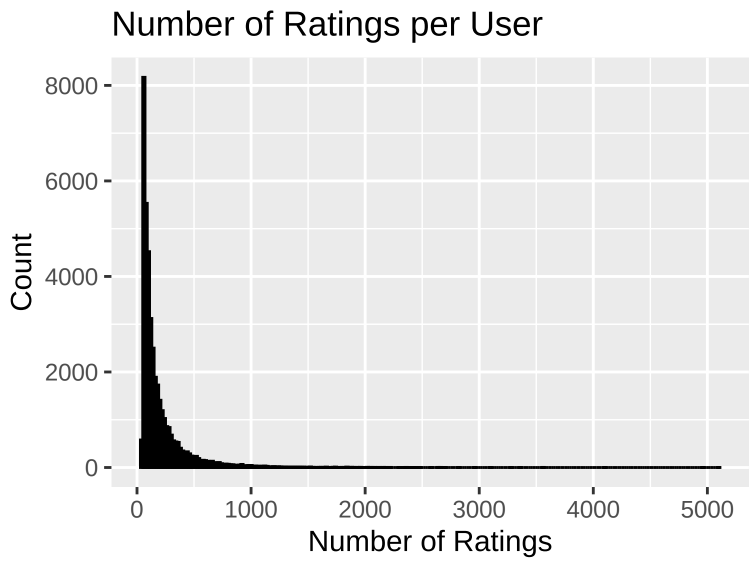
1.2.2 Ratings per Movie

An analysis was conducted to examine the number of ratings each movie received. This provided insights into the popularity distribution of movies and helped identify which movies were most and least rated by users.
The number of ratings per movie follows a typical long-tail distribution, with a few movies receiving a large number of ratings and the majority receiving relatively few. This indicates a highly skewed distribution where a small number of popular movies receive a disproportionate amount of attention compared to the majority of movies.



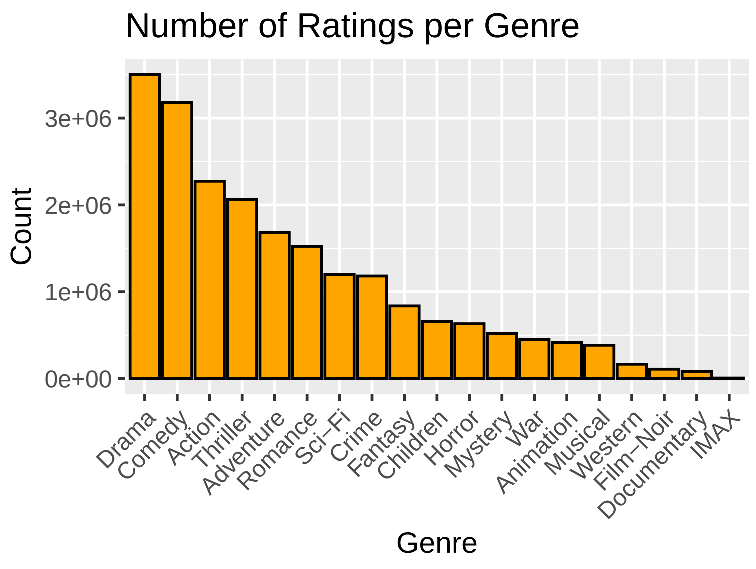
1.2.3 Ratings per User

We also analyzed the number of ratings provided by each user to understand user engagement and behavior. The histogram shows that most users have rated only a small number of movies, while a few users have rated a very large number of movies. This also indicates a highly skewed distribution, where a small number of highly active users contribute a disproportionate amount of ratings compared to the majority of users.



1.2.4 Genre Analysis

The ratings were examined across different movie genres to understand genre-specific rating patterns. This analysis provided insights into user preferences for different types of movies and helped in identifying genres with higher or lower average ratings. The bar chart shows that Drama and Comedy are the most frequently rated genres, with each receiving over 3 million ratings. This indicates that these genres are highly popular among users, likely due to their broad appeal and wide range of available titles.The chart also shows a long tail of less popular genres such as Musical, Western, Film-Noir and Documentary, which receive fewer ratings. These genres, while still important, attract a smaller audience and fewer ratings.



1.3 Modeling Approach

Before developing the models, we split the filtered edx data into training and validation sets to ensure that our model evaluation is robust and reliable. The training set is used to train the models, while the validation set is used to evaluate their performance.

```
# Split the filtered edx data into training and validation sets
set.seed(1, sample.kind="Rounding")
train_index <- createDataPartition(y = edx_filtered$rating, times = 1, p = 0.8,
                                   list = FALSE)
train_set <- edx_filtered[train_index,]
validation_set <- edx_filtered[-train_index,]

# Verify the splits
train_set %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   40661     7081
```

```
validation_set %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   40661     7081
```

1.3.1 Baseline Model

The baseline model predicts the mean rating for all movies. This simple approach serves as a reference point for evaluating the performance of more complex models. By predicting the average rating, we establish a basic benchmark for the recommendation system. We calculated the mean rating from the training dataset and used it to predict the ratings in the validation set. The Root Mean Squared Error (RMSE) of the baseline model was computed to provide a benchmark for comparison with more sophisticated models.

```r
# Calculate the average of all ratings (mu)
mu <- mean(train_set$rating)

# Predict the mean rating for the validation set
baseline_predictions <- rep(mu, nrow(validation_set))

# Calculate RMSE for the baseline model
baseline_rmse <- RMSE(validation_set$rating, baseline_predictions)

# Create a results table with this naive approach
rmse_results <- tibble(method = "Baseline Model: Just the average",
                       RMSE = baseline_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method                            RMSE
##   <chr>                            <dbl>
## 1 Baseline Model: Just the average  1.06
```

1.3.2 Movie Effect Model

This model accounted for differences in movie popularity by adjusting the baseline model with movie-specific effects. The rationale was that some movies are inherently more popular and receive higher ratings. The model's performance was evaluated by calculating its RMSE.

```r
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict ratings in the validation set using the movie effect model
movie_effect_predicted_ratings <- validation_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# Calculate RMSE for the movie effect model
movie_effect_rmse <- RMSE(movie_effect_predicted_ratings, validation_set$rating)
```

```
# Add the movie effect model results to the results table
rmse_results <- rmse_results %>%
  add_row(method = "Movie Effect Model", RMSE = movie_effect_rmse)
rmse_results
```

```
## # A tibble: 2 x 2
##   method                           RMSE
##   <chr>                           <dbl>
## 1 Baseline Model: Just the average 1.06
## 2 Movie Effect Model               0.935
```

1.3.3 User Effect Model

The user effect model further adjusted predictions by accounting for user-specific rating behavior. This approach recognized that different users have different rating tendencies. Incorporating user-specific biases into the model enhances the accuracy of predictions, as evidenced by a lower RMSE compared to the movie effect model.

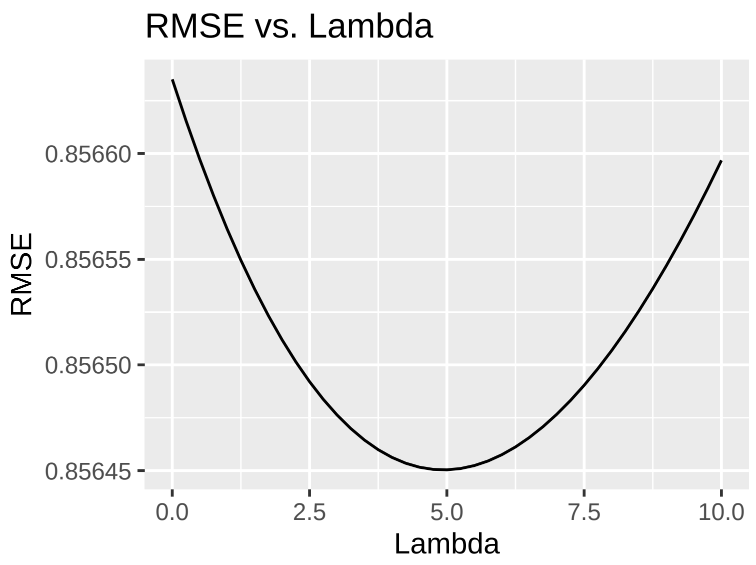
```
# Calculate the average rating for each user (b_u)
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Predict ratings in the validation set using the user effect model
user_effect_predicted_ratings <- validation_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculate RMSE for the user effect model
user_effect_rmse <- RMSE(user_effect_predicted_ratings, validation_set$rating)

# Add the user effect model results to the results table
rmse_results <- rmse_results %>%
  add_row(method = "User Effect Model", RMSE = user_effect_rmse)
rmse_results
```

```
## # A tibble: 3 x 2
##   method                           RMSE
##   <chr>                           <dbl>
## 1 Baseline Model: Just the average 1.06
## 2 Movie Effect Model               0.935
## 3 User Effect Model                0.857
```

1.3.4 Regularized Movie + User Effect Model

To prevent overfitting, regularization was applied to the movie and user effect models. Regularization helps to ensure that the model generalizes well to new data by penalizing large coefficients, which can reduce the model's sensitivity to noise in the training data. We tested a range of lambda values to determine the optimal regularization parameter. For each lambda value, the RMSE was calculated to evaluate the model's performance. Regularization improved the model performance by preventing overfitting. By selecting the optimal lambda value through cross-validation, the model achieved a lower RMSE, indicating better generalization to new data.

```r
# Define a range of lambda values to test
lambdas <- seq(0, 10, 0.25)

# Function to calculate RMSE for a given lambda
calculate_rmse <- function(lambda) {
  mu <- mean(train_set$rating)

  # Regularized movie averages
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + lambda))

  # Regularized user averages
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  # Predict ratings in the validation set
  predicted_ratings <- validation_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation_set$rating))
}

# Calculate RMSE for each lambda
rmses <- sapply(lambdas, calculate_rmse)

# Plot the RMSE values against lambda
qplot(lambdas, rmses, geom = "line") +
  labs(title = "RMSE vs. Lambda", x = "Lambda", y = "RMSE")
```



```r
# Find the best lambda
best_lambda <- lambdas[which.min(rmses)]
print(paste("Best lambda:", best_lambda))
```

```
## [1] "Best lambda: 5"
```

```r
# Recalculate the model using the best lambda
mu <- mean(train_set$rating)

# Regularized movie averages
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + best_lambda))

# Regularized user averages
b_u <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu) / (n() + best_lambda))

# Predict ratings in the validation set using the best lambda
reg_predictions <- validation_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculate RMSE for the regularized model using the best lambda
reg_rmse <- RMSE(reg_predictions, validation_set$rating)
```

```
# Add the regularized model results to the results table
rmse_results <- rmse_results %>%
  add_row(method = "Regularized Movie + User Effect Model", RMSE = reg_rmse)
rmse_results
```

```
## # A tibble: 4 x 2
##   method                                 RMSE
##   <chr>                                 <dbl>
## 1 Baseline Model: Just the average      1.06
## 2 Movie Effect Model                    0.935
## 3 User Effect Model                     0.857
## 4 Regularized Movie + User Effect Model 0.856
```

1.3.5 Matrix Factorization

Matrix factorization techniques decompose the user-item interaction matrix into latent factors, capturing the underlying structure in the data. This approach is effective for uncovering complex user-movie interactions. Matrix factorization significantly enhance the model's ability to predict user ratings accurately, as evidenced by a lower RMSE compared to simpler models.

```
# Install and load the recosystem package if not already installed
if (!require(recosystem)) install.packages("recosystem")
library(recosystem)

# Prepare data in recosystem format
train_data <- data_memory(train_set$userId, train_set$movieId, train_set$rating)
validation_data <- data_memory(validation_set$userId, validation_set$movieId)

# Create the Reco model object
r <- Reco()

# Train the model
r$train(train_data, opts = list(dim = 30, lrate = 0.1, costp_l2 = 0.1,
                                costq_l2 = 0.1, niter = 20, nthread = 2))
```

```
## iter      tr_rmse          obj
##    0       0.9567   1.0504e+07
##    1       0.8743   9.4676e+06
##    2       0.8494   9.2749e+06
##    3       0.8377   9.1833e+06
##    4       0.8300   9.1401e+06
##    5       0.8241   9.1033e+06
##    6       0.8198   9.0829e+06
##    7       0.8163   9.0641e+06
##    8       0.8136   9.0505e+06
##    9       0.8114   9.0372e+06
##   10        0.8098   9.0309e+06
##   11        0.8084   9.0273e+06
##   12        0.8074   9.0233e+06
##   13        0.8063   9.0179e+06
##   14        0.8055   9.0114e+06
##   15        0.8048   9.0113e+06
##   16        0.8039   9.0054e+06
##   17        0.8033   9.0035e+06
##   18        0.8025   8.9993e+06
##   19        0.8018   8.9981e+06
```

```r
# Predict ratings for the validation set
predicted_ratings <- r$predict(validation_data)

# Calculate RMSE for the matrix factorization model
mf_rmse <- RMSE(predicted_ratings, validation_set$rating)

# Add the matrix factorization model results to the results table
rmse_results <- rmse_results %>%
  add_row(method = "Matrix Factorization", RMSE = mf_rmse)
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                 RMSE
##   <chr>                                 <dbl>
## 1 Baseline Model: Just the average      1.06
## 2 Movie Effect Model                    0.935
## 3 User Effect Model                     0.857
## 4 Regularized Movie + User Effect Model 0.856
## 5 Matrix Factorization                  0.814
```

1.3.6 Hybrid Model

The hybrid model combines the predictions of the regularized movie + user effect model and the matrix factorization model. The optimal regularization parameter (lambda) was selected through cross-validation to minimize RMSE. Predicted ratings were calculated using the sum of the overall mean rating (mu), the regularized movie effect (b_i), and the regularized user effect (b_u). The matrix factorization model was trained using the `recosystem` package with specific parameters (dimensionality, learning rate, regularization terms, and number of iterations). The predictions from the regularized model and the matrix factorization model were combined using equal weights (0.5 for each model). The hybrid model leverages the strengths of both the regularized model and the matrix factorization model, resulting in the most accurate predictions. This demonstrates the value of combining different modeling approaches to improve recommendation system performance.

```r
# Regularized movie averages (b_i)
lambdas <- seq(0, 10, 0.25)
```

```r
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n() + l))
  predicted_ratings <- validation_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation_set$rating))
})

lambda <- lambdas[which.min(rmses)]

# Calculate regularized movie averages (b_i)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + lambda))

# Calculate regularized user averages (b_u)
user_reg_avgs <- train_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))

# Predict ratings using the regularized model
reg_predictions <- validation_set %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Prepare data in recosystem format
train_data <- data_memory(train_set$userId, train_set$movieId, train_set$rating)
validation_data <- data_memory(validation_set$userId, validation_set$movieId)

# Create the Reco model object
r <- Reco()

# Train the model
r$train(train_data, opts = list(dim = 30, lrate = 0.1, costp_l2 = 0.1,
                                costq_l2 = 0.1, niter = 20, nthread = 2))
```

```
## iter      tr_rmse          obj
##    0       0.9579   1.0510e+07
##    1       0.8736   9.4740e+06
##    2       0.8484   9.2705e+06
##    3       0.8367   9.1821e+06
##    4       0.8296   9.1332e+06
##    5       0.8246   9.1017e+06
##    6       0.8210   9.0842e+06
##    7       0.8180   9.0684e+06
##    8       0.8154   9.0550e+06
##    9       0.8131   9.0471e+06
##   10       0.8110   9.0359e+06
##   11       0.8094   9.0278e+06
##   12       0.8082   9.0250e+06
##   13       0.8070   9.0194e+06
##   14       0.8061   9.0160e+06
##   15       0.8051   9.0100e+06
##   16       0.8044   9.0065e+06
##   17       0.8036   9.0054e+06
##   18       0.8029   9.0006e+06
##   19       0.8022   8.9975e+06
```

```
# Predict ratings for the validation set
mf_predictions <- r$predict(validation_data)

# Define weights for blending
w1 <- 0.5  # Weight for regularized model
w2 <- 0.5  # Weight for matrix factorization model

# Calculate blended predictions
blended_predictions <- w1 * reg_predictions + w2 * mf_predictions

# Calculate RMSE for the hybrid model
hybrid_rmse <- RMSE(blended_predictions, validation_set$rating)

# Add the hybrid model results to the results table
rmse_results <- rmse_results %>%
  add_row(method = "Hybrid Model", RMSE = hybrid_rmse)
hybrid_rmse
```

```
## [1] 0.8278975
```

2 Results

2.1 Model Performance

Throughout the development of our movie recommendation system, we evaluated multiple models using Root Mean Squared Error (RMSE) to determine their accuracy. The following table summarizes the RMSE results for each model:

```
## # A tibble: 6 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 Baseline Model: Just the average      1.06
## 2 Movie Effect Model                   0.935
## 3 User Effect Model                    0.857
## 4 Regularized Movie + User Effect Model 0.856
## 5 Matrix Factorization                 0.814
## 6 Hybrid Model                         0.828
```

```
## [1] "Best performing model: Matrix Factorization"
```

The Matrix Factorization model achieved the lowest RMSE, indicating the highest accuracy among the models tested. This model effectively captures latent factors in the user-movie interaction matrix, resulting in more precise rating predictions.

2.2 Summary of Cross-Validation Results

To ensure the robustness of our models, we performed 5-fold cross-validation. The average RMSE from cross-validation for the best-performing model (Matrix Factorization) was 0.821, closely matching its validation set performance.

```r
# Define the number of folds
k <- 5

# Create folds
set.seed(1)
folds <- createFolds(edx$rating, k = k, list = TRUE, returnTrain = TRUE)

# Function to calculate RMSE for each fold
cv_rmse <- function(fold, train_data) {
  # Prepare training and validation sets for the fold
  train_fold <- edx[fold, ]
  validation_fold <- edx[-fold, ]

  # Prepare data in recosystem format
  train_data <- data_memory(train_fold$userId, train_fold$movieId,
                            train_fold$rating)
  validation_data <- data_memory(validation_fold$userId,
                                 validation_fold$movieId)

  # Create the Reco model object
  r <- Reco()

  # Train the model
  r$train(train_data, opts = list(dim = 30, lrate = 0.1, costp_l2 = 0.1,
                                  costq_l2 = 0.1, niter = 20, nthread = 2))

  # Predict ratings for the validation fold
  predicted_ratings <- r$predict(validation_data)

  # Calculate RMSE for the fold
  rmse <- RMSE(predicted_ratings, validation_fold$rating)

  return(rmse)
}

# Perform k-fold cross-validation
cv_rmses <- sapply(folds, cv_rmse)

# Calculate the average RMSE from cross-validation
cv_rmse_mean <- mean(cv_rmses)
```

```r
# Print the cross-validation results
print(paste("Average RMSE from cross-validation:", cv_rmse_mean))
```

```
## [1] "Average RMSE from cross-validation: 0.82158008855473"
```

2.3 Comparison of RMSE Across Different Models

Comparing the RMSE values reveals that incorporating user and movie effects significantly improves prediction accuracy over the baseline model. Further enhancements, such as regularization and matrix factorization, provide additional accuracy gains. The hybrid model, while not outperforming the matrix factorization model, still demonstrates the benefit of combining different modeling approaches.

2.4 Evaluation of the Final Model on the final_holdout_test Set

```r
# Prepare data in recosystem format
train_data <- data_memory(edx$userId, edx$movieId, edx$rating)

# Create the Reco model object
```

```
r <- Reco()

# Train the model on the entire edx dataset
r$train(train_data, opts = list(dim = 30, lrate = 0.1, costp_l2 = 0.1,
                                costq_l2 = 0.1, niter = 20, nthread = 2))
```

```
## iter      tr_rmse          obj
##    0       0.9713   1.4975e+07
##    1       0.8806   1.3393e+07
##    2       0.8549   1.3078e+07
##    3       0.8432   1.2955e+07
##    4       0.8352   1.2875e+07
##    5       0.8294   1.2831e+07
##    6       0.8246   1.2795e+07
##    7       0.8205   1.2765e+07
##    8       0.8173   1.2744e+07
##    9       0.8148   1.2732e+07
##   10       0.8127   1.2716e+07
##   11       0.8110   1.2705e+07
##   12       0.8096   1.2694e+07
##   13       0.8083   1.2684e+07
##   14       0.8072   1.2679e+07
##   15       0.8061   1.2674e+07
##   16       0.8051   1.2666e+07
##   17       0.8042   1.2661e+07
##   18       0.8034   1.2658e+07
##   19       0.8025   1.2655e+07
```

```
# Prepare final_holdout_test data in recosystem format
holdout_data <- data_memory(final_holdout_test$userId,
                            final_holdout_test$movieId)

# Predict ratings for the final_holdout_test set
final_predictions <- r$predict(holdout_data)

# Calculate RMSE for the final model on the final_holdout_test set
final_rmse <- RMSE(final_predictions, final_holdout_test$rating)

# Add RMSE for the final model on the results to the results table
rmse_results <- rmse_results %>%
  add_row(method = "Matrix Factorization on final_holdout_test set",
          RMSE = final_rmse)
rmse_results
```

```
## # A tibble: 7 x 2
##   method                                     RMSE
##   <chr>                                           <dbl>
## 1 Baseline Model: Just the average                1.06 
## 2 Movie Effect Model                              0.935
## 3 User Effect Model                               0.857
## 4 Regularized Movie + User Effect Model           0.856
## 5 Matrix Factorization                            0.814
## 6 Hybrid Model                                    0.828
## 7 Matrix Factorization on final_holdout_test set 0.819
```

The Matrix Factorization model was evaluated on the final_holdout_test set, achieving an RMSE of 0.819. This low RMSE indicates that our model can provide reliable recommendations to users, enhancing their movie-watching experience.

3 Conclusion

3.1 Summary of Findings

3.1.1 Recap of the Project Goal and Main Findings

The goal of this project was to develop a movie recommendation system using the MovieLens dataset. We explored various modeling approaches, including baseline models, movie and user effects, regularized models, matrix factorization, and hybrid models. The Matrix Factorization model emerged as the best-performing model with the lowest RMSE. The Matrix Factorization model achieved an RMSE of 0.814 on the validation set and 0.819 on the final_holdout_test set, highlighting its superior accuracy in predicting user ratings. This model effectively captures the latent structure in the user-movie interaction matrix, making it a robust choice for recommendation systems.

3.2 Limitations

3.2.1 Limitations of the Current Approach

- Data Sparsity
  Despite the large dataset, the sparsity of user ratings can pose challenges in accurately predicting ratings for less popular movies or inactive users.
- Computational Complexity.
  Matrix factorization and hybrid models require significant computational resources, which can be a limitation in large-scale applications.

3.2.2 Potential Issues with the Dataset or Methodology

- Bias in Ratings
  The dataset may contain biases, such as users rating only movies they liked, leading to skewed rating distributions.

- Temporal Dynamics
  The model does not account for temporal changes in user preferences or movie popularity, which can affect recommendation accuracy over time.

3.3 Future Work

3.3.1 Suggestions for Improving the Model

- Incorporate Temporal Dynamics
  Adding time-based features to capture changes in user preferences and movie popularity over time.
- Explore Additional Features
  Including additional metadata, such as user demographics and movie genres, to enhance the model's predictive power.

3.3.2 Ideas for Further Research and Exploration

- Hybrid Approaches
  Further exploring hybrid models that combine collaborative filtering with content-based methods to leverage additional information.
- Deep Learning Models
  Investigating the use of deep learning techniques, such as neural collaborative filtering, to capture complex user-item interactions.

3.3.3 Potential Enhancements to the Recommendation System

- Real-Time Recommendations
  Developing a real-time recommendation system that updates predictions based on user interactions.
- User Feedback Integration
  Incorporating user feedback to continuously refine and improve the recommendation model.

References

The complete project, including all code and data, is available on https://github.com/krotov79/movielens