MovieLens Project

Ammar Yasser Mohamed Abdullah

2025-01-17

Introduction

Recommender systems are a crucial part of modern information filtering systems that are designed to predict the preferences or ratings of users against items such as movies, products, or services. In addition, these systems make personalized recommendations based on user-generated ratings to users for increasing their experience and engagement. The Netflix Prize, launched in 2006, showed that a major improvement was possible in recommendation algorithms since the winning team achieved an improvement of 10% in predictive accuracy. It derived inspiration from this, and this project tries to provide a movie recommendation system on the MovieLens dataset-10M, which includes approximately 10 million ratings from over 72,000 users for about 10,000 movies.

This project predicts individual movie ratings by in-depth analysis of the effects of movies, users, and genres. The dataset is split into training, testing, and validation sets to ensure proper model evaluation. Thereby, the size of data and sparsity may turn out to be so huge that traditional modeling through linear regression itself might turn out to be impractical; the least squares estimate is computed along with regularization for better prediction results. The performances of the model could be measured w.r.t. the **Root Mean Squared Error (RMSE)**-thereby attaining an error less than the target value for meeting project requirements or **0.86490**.

Methodology

Data Preparation

The MovieLens dataset, collected by GroupLens Research, was loaded and split into an edx set (90% of the data) and a final_holdout set (10% of the data). The edx set was further divided into training (90%) and testing (10%) sets to facilitate model development and evaluation. The final_holdout set was reserved for final model validation.

```r
if (!require(tidyverse)) install.packages("tidyverse")
```

```
## Loading required package: tidyverse

## -- Attaching core tidyverse packages ------------------------------------------------------------------
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.2
## -- 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
```

```
if (!require(caret)) install.packages("caret")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if (!require(raster)) install.packages("raster")
```

```
## Loading required package: raster
## Loading required package: sp
##
## Attaching package: 'raster'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(tidyverse)
library(caret)
library(raster)

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)
}

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)
```

```r
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
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
# 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)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```r
edx <- rbind(edx, removed)

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

Splitting Edx Data Step

```r
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
test_indices <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_data <- edx[-test_indices,]
test_data <- edx[test_indices,]

test_data <- test_data %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")

removed_data <- anti_join(edx[test_indices,], test_data)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```r
train_data <- rbind(train_data, removed_data)

rm(test_indices, removed_data)
```

Separating Genres

```r
train_genres <- train_data %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)

test_genres <- test_data %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)
```

Model Development

Several models were developed, each incorporating additional effects to improve prediction accuracy:

- **Naive Model**: Predicts the average rating for all movies and users, serving as a baseline.
- **Movie Effect Model**: Incorporates movie-specific biases to account for differences in movie popularity.
- **User Effect Model**: Adds user-specific biases to capture individual rating tendencies.
- **Genre Effect Model**: Includes genre-specific effects to account for differences in genre preferences.
- **Genre-User Interaction Model**: Introduces interaction effects between users and genres to capture personalized genre preferences.
- **Regularized Model**: Applies regularization to penalize overfitting, particularly for movies and genres with few ratings.

A function to Calculate RMSE

```r
calculate_rmse <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Baseline Naive Model

```r
mean_rating <- mean(train_data$rating)
naive_model_rmse <- calculate_rmse(test_data$rating, mean_rating)
```

Adding Movie Effects

```r
movie_effects <- train_data %>%
  group_by(movieId) %>%
  summarize(movie_effect = mean(rating - mean_rating))

predicted_ratings_movie_effect <- mean_rating + test_data %>%
  left_join(movie_effects, by = 'movieId') %>%
  pull(movie_effect)

predicted_ratings_movie_effect <- clamp(predicted_ratings_movie_effect, 0.5, 5)

movie_effect_rmse <- calculate_rmse(predicted_ratings_movie_effect, test_data$rating)
```

User Effect Addition

```
user_effects <- train_data %>%
  left_join(movie_effects, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(user_effect = mean(rating - mean_rating - movie_effect))

predicted_ratings_user_effect <- test_data %>%
  left_join(movie_effects, by = 'movieId') %>%
  left_join(user_effects, by = 'userId') %>%
  mutate(pred = mean_rating + movie_effect + user_effect) %>%
  pull(pred)

predicted_ratings_user_effect <- clamp(predicted_ratings_user_effect, 0.5, 5)

user_effect_rmse <- calculate_rmse(predicted_ratings_user_effect, test_data$rating)
```

Genre Effects Addition

```
genre_effects <- train_genres %>%
  left_join(movie_effects, by = 'movieId') %>%
  left_join(user_effects, by = 'userId') %>%
  group_by(genres) %>%
  summarize(genre_effect = mean(rating - mean_rating - movie_effect - user_effect))

predicted_ratings_genre_effect <- test_genres %>%
  left_join(movie_effects, by = 'movieId') %>%
  left_join(user_effects, by = 'userId') %>%
  left_join(genre_effects, by = 'genres') %>%
  mutate(pred = mean_rating + movie_effect + user_effect + genre_effect) %>%
  pull(pred)

predicted_ratings_genre_effect <- clamp(predicted_ratings_genre_effect, 0.5, 5)

genre_effect_rmse <- calculate_rmse(predicted_ratings_genre_effect, test_genres$rating)
```

Genre User Interaction

```
genre_user_interaction <- train_genres %>%
  left_join(movie_effects, by = 'movieId') %>%
```

```r
  left_join(user_effects, by = 'userId') %>%
  left_join(genre_effects, by = 'genres') %>%
  group_by(genres, userId) %>%
  summarize(genre_user_effect = mean(rating - mean_rating - movie_effect - user_effect - genre_effect))
```

```
## ‘summarise()‘ has grouped output by ’genres’. You can override using the
## ‘.groups‘ argument.
```

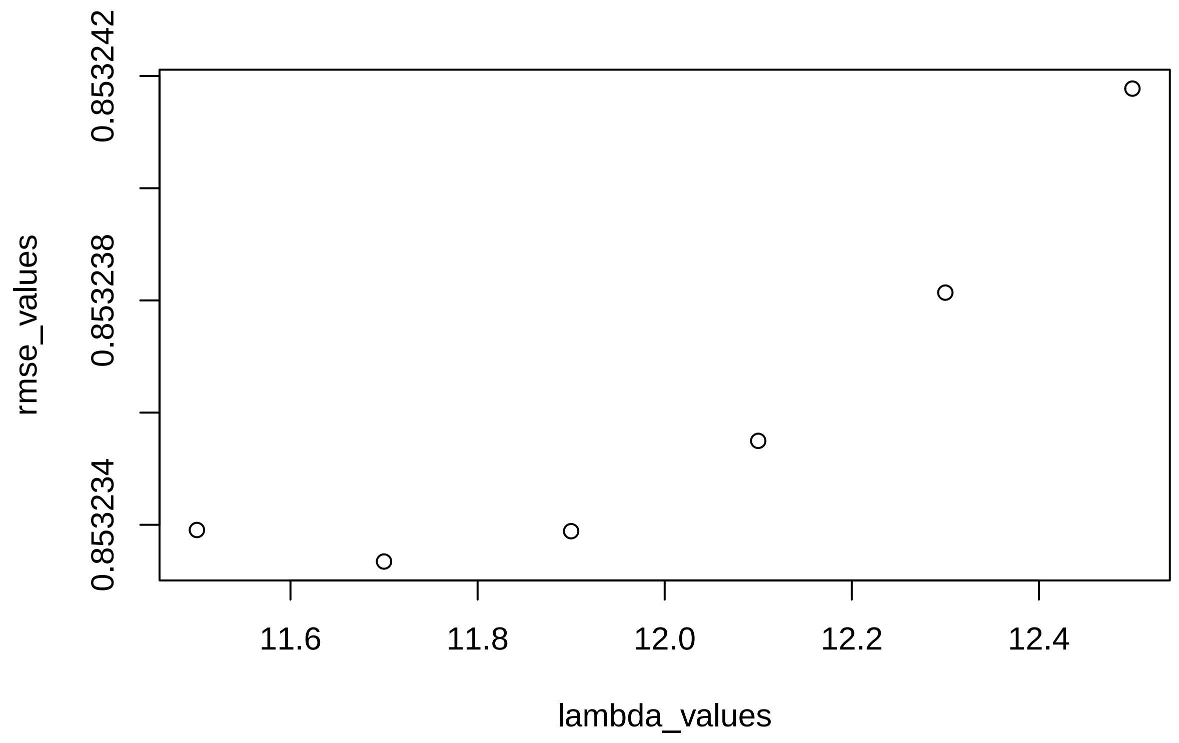
```r
predicted_ratings_genre_user_effect <- test_genres %>%
  left_join(movie_effects, by = 'movieId') %>%
  left_join(user_effects, by = 'userId') %>%
  left_join(genre_effects, by = 'genres') %>%
  left_join(genre_user_interaction, by = c("userId", "genres")) %>%
  mutate(genre_user_effect = ifelse(is.na(genre_user_effect), 0, genre_user_effect),
         pred = mean_rating + movie_effect + user_effect + genre_effect + genre_user_effect) %>%
  pull(pred)

predicted_ratings_genre_user_effect <- clamp(predicted_ratings_genre_user_effect, 0.5, 5)

genre_user_effect_rmse <- calculate_rmse(predicted_ratings_genre_user_effect, test_genres$rating)
```

Regularization Step

```r
regularization_function <- function(lambda) {
  mean_rating <- mean(train_data$rating)

  movie_effects_reg <- train_data %>%
    group_by(movieId) %>%
    summarize(movie_effect = sum(rating - mean_rating) / (n() + lambda))

  user_effects_reg <- train_data %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(user_effect = sum(rating - movie_effect - mean_rating) / (n() + lambda))

  genre_effects_reg <- train_genres %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    left_join(user_effects_reg, by = "userId") %>%
    group_by(genres) %>%
    summarize(genre_effect = sum(rating - mean_rating - movie_effect - user_effect) / (n() + lambda))

  genre_user_interaction_reg <- train_genres %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    left_join(user_effects_reg, by = "userId") %>%
    left_join(genre_effects_reg, by = "genres") %>%
    group_by(genres, userId) %>%
    summarize(genre_user_effect = sum(rating - mean_rating - movie_effect - user_effect - genre_effect)

  predicted_ratings_reg <- test_genres %>%
    left_join(movie_effects_reg, by = "movieId") %>%
    left_join(user_effects_reg, by = "userId") %>%
    left_join(genre_effects_reg, by = "genres") %>%
    left_join(genre_user_interaction_reg, by = c("userId", "genres")) %>%
    mutate(genre_user_effect = ifelse(is.na(genre_user_effect), 0, genre_user_effect),
           pred = mean_rating + movie_effect + user_effect + genre_effect + genre_user_effect) %>%
    pull(pred)

  predicted_ratings_reg <- clamp(predicted_ratings_reg, 0.5, 5)

  return(calculate_rmse(predicted_ratings_reg, test_genres$rating))
}

lambda_values <- seq(11.5, 12.5, 0.2)
rmse_values <- sapply(lambda_values, regularization_function)
```

```
## `summarise()` has grouped output by 'genres'. You can override using the `.groups` argument.
## `summarise()` has grouped output by 'genres'. You can override using the `.groups` argument.
## `summarise()` has grouped output by 'genres'. You can override using the `.groups` argument.
## `summarise()` has grouped output by 'genres'. You can override using the `.groups` argument.
## `summarise()` has grouped output by 'genres'. You can override using the `.groups` argument.
## `summarise()` has grouped output by 'genres'. You can override using the `.groups` argument.
```

```
plot(lambda_values, rmse_values)
```



```
optimal_lambda <- lambda_values[which.min(rmse_values)]
regularized_model_rmse <- min(rmse_values)
```

Evaluation Metric

The Root Mean Square Error (**RMSE**) was used to evaluate model performance. RMSE measures the difference between predicted and actual ratings, with lower values indicating better performance.

```
# Create a data frame to store all RMSE values
rmse_results <- data.frame(
  Model = c("Naive Model", "Movie Effect Model", "User Effect Model",
            "Genre Effect Model", "Genre-User Interaction Model", "Regularized Model"),
  RMSE = c(naive_model_rmse, movie_effect_rmse, user_effect_rmse,
           genre_effect_rmse, genre_user_effect_rmse, regularized_model_rmse)
)

# Print the RMSE table
rmse_results
```

```
##                          Model      RMSE
## 1                  Naive Model 1.0600537
## 2           Movie Effect Model 0.9429615
## 3            User Effect Model 0.8644818
## 4           Genre Effect Model 0.8625073
## 5 Genre-User Interaction Model 0.8662169
## 6            Regularized Model 0.8532333
```

Final Holdout Test Set Evaluation

```
# Separate genres in the final holdout test set
final_holdout_genres <- final_holdout_test %>%
  separate_rows(genres, sep = "\\|", convert = TRUE)

# Calculate effects using the optimal lambda (11.7)
mean_rating_final <- mean(train_data$rating)

movie_effects_final <- train_data %>%
  group_by(movieId) %>%
  summarize(movie_effect = sum(rating - mean_rating_final) / (n() + optimal_lambda))

user_effects_final <- train_data %>%
  left_join(movie_effects_final, by = "movieId") %>%
  group_by(userId) %>%
  summarize(user_effect = sum(rating - movie_effect - mean_rating_final) / (n() + optimal_lambda))

genre_effects_final <- train_genres %>%
  left_join(movie_effects_final, by = "movieId") %>%
  left_join(user_effects_final, by = "userId") %>%
  group_by(genres) %>%
  summarize(genre_effect = sum(rating - mean_rating_final - movie_effect - user_effect) / (n() + optima
genre_user_interaction_final <- train_genres %>%
  left_join(movie_effects_final, by = "movieId") %>%
  left_join(user_effects_final, by = "userId") %>%
  left_join(genre_effects_final, by = "genres") %>%
  group_by(genres, userId) %>%
  summarize(genre_user_effect = sum(rating - mean_rating_final - movie_effect - user_effect - genre_eff
```

```
## ‘summarise()‘ has grouped output by ’genres’. You can override using the
## ‘.groups‘ argument.
```

```
# Predict ratings on the final holdout test set
predicted_ratings_final <- final_holdout_genres %>%
  left_join(movie_effects_final, by = "movieId") %>%
  left_join(user_effects_final, by = "userId") %>%
  left_join(genre_effects_final, by = "genres") %>%
  left_join(genre_user_interaction_final, by = c("userId", "genres")) %>%
  mutate(genre_user_effect = ifelse(is.na(genre_user_effect), 0, genre_user_effect),
         pred = mean_rating_final + movie_effect + user_effect + genre_effect + genre_user_effect) %>%
  pull(pred)

# Clamp predictions to be between 0.5 and 5
predicted_ratings_final <- clamp(predicted_ratings_final, 0.5, 5)

# Calculate RMSE for the final holdout test set
final_holdout_rmse <- calculate_rmse(predicted_ratings_final, final_holdout_genres$rating)
final_holdout_rmse
```

```
## [1] 0.8539444
```

Results

The following table summarizes the RMSE values for each model:

Model	RMSE
Naive Model	1.0600537
Movie Effect Model	0.9429615
User Effect Model	0.8644818
Genre Effect Model	0.8625073
Genre-User Interaction Model	0.8662169
Regularized Model	0.8532333

The Regularized Model achieved the lowest RMSE of 0.8532333 on the test set. When applied to the final_holdout set, the model achieved an RMSE of 0.8539444, demonstrating strong generalization performance.

Conclusion

This project successfully developed a movie rating prediction system using the MovieLens dataset. By incorporating movie, user, and genre effects, as well as regularization, the final model achieved an RMSE of 0.8532333, significantly improving upon the baseline naive model. The inclusion of regularization helped mitigate overfitting, particularly for movies and genres with limited ratings.

Key findings include:

- The movie effect had the most significant impact on reducing RMSE, highlighting the importance of movie-specific biases.
- The user effect further improved predictions by accounting for individual rating tendencies.
- The genre effect and genre-user interaction provided additional improvements, though their impact was relatively smaller.
- Regularization proved essential for stabilizing predictions, especially for less frequently rated movies and genres.

For future work, additional features such as user demographics (e.g., age, gender) and movie metadata (e.g., release year, director, actors) could be incorporated to further enhance prediction accuracy.

Additional Notes

Execution Environment:

This project was executed using Google Colab to leverage its superior computational resources, including higher memory and processing power. The original work was done in a Jupyter Notebook (.ipynb), which was later converted to an R Markdown (.rmd) file for efficient submission.

The decision to use Google Colab was driven by the limitations of my local machine, which has only 8GB of RAM. Given the size of the MovieLens dataset (over 10 million ratings), local execution was impractical due to memory constraints.

Data Partitioning:

Throughout the project, the dataset was split into a training set (90%) and a testing set (10%). This partitioning ensures that the model is trained on a large portion of the data while reserving a smaller portion for evaluation. The testing set is used to assess the model's generalization performance.

Future Improvements:

Incorporating additional features such as user demographics (e.g., age, gender) and movie metadata (e.g., release year, director, actors) could further improve prediction accuracy.

Exploring advanced techniques like matrix factorization or deep learning could provide better results.