MovieLens Recommendation System

Ivan Blanquez

Introduction / Executive Summary

This report analyzes the MovieLens 10M dataset to build a recommendation system that predicts movie ratings. The dataset contains ratings from users for movies along with features such as movie titles, genres, and user IDs. In this analysis, we will explore several methods for building a recommendation system, and we will evaluate these methods using the **Root Mean Squared Error (RMSE)** metric.

The process consists of:

1. Data loading and preprocessing.
2. Building several recommendation models.
3. Evaluating the performance of each model.
4. Reporting the results, including the final RMSE score on the test set.

Data Loading and Preprocessing

Data Sources

The **MovieLens 10M** dataset consists of two key files:

- **ratings.dat**: Contains user ratings for movies.
- **movies.dat**: Contains information about the movies, such as titles and genres.

The dataset was loaded from a local path `C:/RSTUDIO/MOVIE LENS` to avoid network download issues.

```
# Load necessary libraries
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 4.3.3
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Warning: package 'lubridate' was built under R version 4.3.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.0
## v purrr     1.0.2
```

```
## -- Conflicts --------------------------------------- tidyverse_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", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

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

```
if(!require(reshape2)) install.packages("reshape2", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: reshape2
```

```
## Warning: package 'reshape2' was built under R version 4.3.3
```

```
##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(tidyverse)
library(caret)
library(reshape2)

# Define the path to the MovieLens dataset
dataset_path <- "C:/RSTUDIO/MOVIE LENS"

# Load the data (adjust the path if needed)
ratings_file <- file.path(dataset_path, "ml-10M100K/ratings.dat")
movies_file <- file.path(dataset_path, "ml-10M100K/movies.dat")

# Read the ratings data and process it
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE), stringsAsFa
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))
```

```r
# Read the movies data and process it
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE), stringsAsFacto
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# Join ratings and movie data
movielens <- left_join(ratings, movies, by = "movieId")
```

Splitting the Data into Training and Testing Sets

Next, the data was split into a training set (edx) and a final hold-out test set (final_holdout_test) for model evaluation. The test set is reserved exclusively for evaluating the final model's performance.

```r
# Create training and hold-out test sets
set.seed(1, sample.kind="Rounding")
```

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

```r
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)
```

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

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

Building the Models

1. Basic Model: Mean Rating

The simplest recommendation model predicts the same rating for every movie, which is simply the average rating of all movies.

```r
# Build the basic recommendation model (Mean rating)
avg_rating <- mean(edx$rating)
predictions <- edx %>%
  mutate(predicted_rating = avg_rating)

# Calculate RMSE for the basic model
rmse_basic <- sqrt(mean((predictions$rating - predictions$predicted_rating)^2))

# Use sprintf to format the output and avoid LaTeX issues
sprintf("RMSE with basic model: %.4f", rmse_basic)
```

```
## [1] "RMSE with basic model: 1.0603"
```

2. Movie Bias Model

The next model incorporates a movie bias, which is the deviation from the average rating for each movie. The idea is that certain movies might be consistently rated higher or lower than others.

```
# Build model with movie bias
movie_bias <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - avg_rating))

predictions_movie_bias <- edx %>%
  left_join(movie_bias, by = "movieId") %>%
  mutate(predicted_rating = avg_rating + b_movie)

# Calculate RMSE with movie bias
rmse_movie_bias <- sqrt(mean((predictions_movie_bias$rating - predictions_movie_bias$predicted_rating)^

# Use sprintf to format the output and avoid LaTeX issues
sprintf("RMSE with movie bias: %.4f", rmse_movie_bias)
```

```
## [1] "RMSE with movie bias: 0.9423"
```

3. User and Movie Bias Model

The most advanced model incorporates both user and movie biases. This allows for personalized recommendations where the ratings are adjusted based on both the user's preferences and the movie's inherent bias.

```
# Build model with user and movie bias
user_bias <- edx %>%
  left_join(movie_bias, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - avg_rating - b_movie))

predictions_user_movie_bias <- edx %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(predicted_rating = avg_rating + b_movie + b_user)

# Calculate RMSE with user and movie bias model
rmse_user_movie_bias <- sqrt(mean((predictions_user_movie_bias$rating - predictions_user_movie_bias$pred

# Use sprintf to format the output and avoid LaTeX issues
sprintf("RMSE with user and movie bias: %.4f", rmse_user_movie_bias)
```

```
## [1] "RMSE with user and movie bias: 0.8567"
```

Final Evaluation

The final RMSE is calculated using the hold-out test set, which ensures that the final model is evaluated on data that was not used during training. This provides a fair evaluation of how the model performs on unseen data.

```r
# Final Evaluation: Use final_holdout_test set to evaluate final RMSE
final_predictions <- final_holdout_test %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(predicted_rating = avg_rating + b_movie + b_user)

final_rmse <- sqrt(mean((final_predictions$rating - final_predictions$predicted_rating)^2))

# Use sprintf to format the output and avoid LaTeX issues
sprintf("Final RMSE on holdout test set: %.4f", final_rmse)
```

```
## [1] "Final RMSE on holdout test set: 0.8653"
```

Conclusion

The models developed provide an increasingly accurate way to predict movie ratings. The User and Movie Bias Model showed the best performance, achieving an RMSE of [final RMSE] on the hold-out test set.

By incorporating both the inherent biases of movies and the preferences of users, we are able to provide more personalized and accurate predictions. This analysis demonstrates the potential of using simple recommendation techniques such as bias adjustments to improve the performance of recommendation systems.