MovieLens Capstone Project

Sibashish Mohapatro/ Sibs M

2025-09-14

Introduction / Executive Summary

This report analyzes the MovieLens 10M dataset, which contains over 10 million movie ratings from users. The main goal was to build a model that can predict how a user would rate a particular movie - a common problem in recommendation systems (like those used by Netflix or Amazon or etc). The key steps in this project include loading and cleaning the data, exploring the dataset, building several predictive models, and evaluating their performance using Root Mean Squared Error (RMSE).

Methods / Analysis

Data Cleaning and Prep'ation

```
{r message=FALSE, warning=FALSE}
# Install and load required packages
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
library(tidyverse)
library(caret)

# Download and unzip the MovieLens 10M dataset
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Read and process the ratings data
ratings_file <- unzip(dl, "ml-10M100K/ratings.dat")
ratings_lines <- readLines(ratings_file)
ratings_lines <- gsub("::", "\t", ratings_lines)  # Replace '::' with tabs for easier reading
ratings <- read.table(text = ratings_lines, col.names = c("userId", "movieId", "rating", "timestamp"))

# Read and process the movies data
movies_file <- unzip(dl, "ml-10M100K/movies.dat")
movies_lines <- readLines(movies_file)
movies <- str_split_fixed(movies_lines, "\\::", 3)  # Split each line into 3 parts
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies)
movies$movieId <- as.numeric(as.character(movies$movieId))
movies$title <- as.character(movies$title)
movies$genres <- as.character(movies$genres)
```

```
# Merge ratings and movie information into one dataframe
movielens <- left_join(ratings, movies, by = "movieId")

# Create validation set (10% of data) and edx set (remaining 90%)
set.seed(1)  # For reproducibility
test_index <- createDataPartition(movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

# Ensure validation set only contains users and movies that exist in edx
validation <- temp %>% semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

# Clean up unnecessary objects
rm(dl, ratings, movies, test_index, temp, movielens, removed)
Data Exploration and Visualization
# Print basic dataset information
cat("Number of users:", length(unique(edx$userId)), "\n")
cat("Number of movies:", length(unique(edx$movieId)), "\n")
cat("Number of ratings:", nrow(edx), "\n")

# Plot distribution of ratings
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black") +
  labs(title = "Distribution of Ratings", x = "Rating", y = "Count")

# Show ratings per movie and per user (summary)
movie_counts <- edx %>% group_by(movieId) %>% summarize(count = n())
user_counts <- edx %>% group_by(userId) %>% summarize(count = n())
cat("Median ratings per movie:", median(movie_counts$count), "\n")
cat("Median ratings per user:", median(user_counts$count), "\n")
```

Insights: Most ratings are between 3 and 4 stars. Some movies and users have many more ratings than others. Modeling Approach

```
# Further split edx into training and test sets for model tuning (10% for testing)
set.seed(42)
edx_index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-edx_index, ]
edx_test  <- edx[edx_index, ]

# Function to calculate Root Mean Squared Error (RMSE)
RMSE <- function(true_ratings, predicted_ratings) {
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

The following models are to be built and compared: Mean Model: Predicting same average rating for all movies. Movie Effect Model: Adjusting for movies that are generally rated higher or lower. Movie + User Effect Model: Adjusting for users who rate higher or lower than average. Regularized Model: Adding regularization to avoid overfitting by smoothing movie and user effects.

Results

Model 1: Mean Model

```
# Model 1: Mean model (predicts the overall average rating)
mu <- mean(edx_train$rating)
mean_pred <- rep(mu, nrow(edx_test))
mean_rmse <- RMSE(edx_test$rating, mean_pred)
cat("Mean RMSE:", mean_rmse, "\n")
```

Model 2: Movie Effect Model

```
# Model 2: Movie effect model (accounts for movie-specific average deviations)
movie_avgs <- edx_train %>% group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
edx_test2 <- left_join(edx_test, movie_avgs, by = "movieId")
movie_pred <- mu + edx_test2$b_i
movie_rmse <- RMSE(edx_test2$rating, movie_pred)
cat("Movie effect RMSE:", movie_rmse, "\n")
```

Model 3: Movie + User Effect Model

```
# Model 3: Movie + User effect model (also accounts for user-specific deviations)
user_avgs <- edx_train %>% left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
edx_test3 <- left_join(edx_test2, user_avgs, by = "userId")
movie_user_pred <- mu + edx_test3$b_i + edx_test3$b_u
movie_user_rmse <- RMSE(edx_test3$rating, movie_user_pred)
cat("Movie + user effect RMSE:", movie_user_rmse, "\n")
```

Model 4: Regularized Model

```
# Model 4: Regularized model (adds regularization to avoid overfitting)
lambdas <- seq(0, 10, 0.25)  # Try different lambda values
rmse_vec <- c()
for (l in lambdas) {
  mu_l <- mean(edx_train$rating)
  # Regularized movie effect
  b_i_l <- edx_train %>% group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_l) / (n() + l))
  # Regularized user effect
  b_u_l <- edx_train %>% left_join(b_i_l, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_l - b_i) / (n() + l))
  # Predict ratings on test set
  edx_test_l <- left_join(edx_test, b_i_l, by = "movieId")
  edx_test_l <- left_join(edx_test_l, b_u_l, by = "userId")
  pred_l <- mu_l + edx_test_l$b_i + edx_test_l$b_u
  rmse_l <- RMSE(edx_test_l$rating, pred_l)
  rmse_vec <- c(rmse_vec, rmse_l)
}
```

```
# Choose the lambda with the lowest RMSE
best_lambda <- lambdas[which.min(rmse_vec)]
cat("Best lambda:", best_lambda, "\n")
cat("Reg. model RMSE:", min(rmse_vec), "\n")
```

Final Model and Validation

```
# Final model: retrain on all edx data and evaluate on validation set
mu_final <- mean(edx$rating)
b_i_final <- edx %>% group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_final) / (n() + best_lambda))
b_u_final <- edx %>% left_join(b_i_final, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_final - b_i) / (n() + best_lambda))

# Predict on validation set
validation2 <- left_join(validation, b_i_final, by = "movieId")
validation2 <- left_join(validation2, b_u_final, by = "userId")
final_pred <- mu_final + validation2$b_i + validation2$b_u
final_rmse <- RMSE(validation2$rating, final_pred)
cat("Final RMSE on validation:", final_rmse, "\n")
```

Results Table

```
# Create and print a summary table of results
results_mat <- data.frame(
  Method = c("Mean Model", "Movie Effect", "Movie + User Effect", "Regularized Model"),
  RMSE = c(mean_rmse, movie_rmse, movie_user_rmse, min(rmse_vec))
)
print(results_mat)
```

The regularized model performed best, achieving the lowest RMSE. Each successive model improved upon the last by incorporating more information about movies and users.

Conclusion In this project, I used the MovieLens 10M dataset to build models that predict movie ratings. By starting with simple models and adding more information about movies and users, I was able to improve prediction accuracy. The regularized model achieved the best performance. Limitations include not using additional features like genres or timestamps, and not trying more advanced machine learning models. In the future, these could be explored to further improve predictions. Overall, this project provided a good introduction to building and evaluating recommendation systems. “‘