Movie Ratings Analysis

Cristian Alfaro

2023-12-06

`{r setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE)`

Introduction

This report presents a detailed analysis of movie ratings data. The analysis covers various aspects including data cleaning, exploratory data analysis, visualization, and predictive modeling. The dataset used for this analysis is from the MovieLens 10M dataset.

Setup

library(tidyverse) library(caret) library(data.table) library(ggplot2) library(ggrepel) library(dslabs) library(lubridate)

options(digits = 3) options(timeout = 120)

Data Loading and Preprocessing

In this section, we will load and preprocess the data from the MovieLens 10M dataset. This dataset comprises two main files: one containing ratings given by users to movies, and the other containing movie details like titles and genres. The preprocessing steps include reading the data, splitting strings, converting data types, and merging the ratings with the movie details.

Define file paths for the datasets

dl <- “ml-10M100K.zip” # File path for the zipped dataset ratings_file <- “ml-10M100K/ratings.dat” # File path for the ratings data movies_file <- “ml-10M100K/movies.dat” # File path for the movies data

Loading and preprocessing the ratings data

The data is split using “::” as the delimiter, and the columns are appropriately named.

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), # Converting userId to integer movieId = as.integer(movieId), # Converting movieId to integer rating =

as.numeric(rating), # Converting rating to numeric timestamp = as.integer(timestamp)) # Converting timestamp to integer

Loading and preprocessing the movies data

Similar to ratings, the movies data is split and column names are assigned.

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)) # Ensuring movieId is of integer type

Merging the ratings and movies data

This creates a single comprehensive dataset with both user ratings and movie details.

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

Splitting the data into training and testing sets

The data is partitioned such that 90% is used for training and 10% for testing.

set.seed(1) # Setting a seed for reproducibility test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE) edx <- movielens[-test_index,] temp <- movielens[test_index,]

Creating the final holdout test set

This step ensures that the test set only contains movies and users present in the training set.

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)

Cleaning up the environment

Removing intermediate variables to keep the workspace clean and efficient.

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

Exploratory Data Analysis

This section explores the structure and composition of the dataset to understand its characteristics and identify any peculiarities. We will examine the dataset's class, dimensions, unique values, identify any duplicates, and explore the distribution of genres.

```{r exploratory-analysis} # Checking the class and dimensions of the datasets class(edx) dim(edx)[1] dim(final_holdout_test)[1]

*Displaying the structure of the dataset*

str(edx)
```

Checking for unique values

This helps to understand the diversity in the dataset in terms of users, movies, and titles.

n_distinct(edxuserId¿n_distinct¿movieId) n_distinct(edx$title)

Identifying movies with duplicate titles

Movies sharing the same title but being distinct entities might indicate data anomalies.

duplicateMovieTitle <- edx %>% select(movieId, title) %>% unique() %>% group_by(title) %>% summarize(n=n()) %>% filter(n>1)

edx %>% filter(title==duplicateMovieTitle$title) %>% select(movieId, genres) %>% group_by(movieId, genres) %>% summarize(n=n()) %>% unique()

Extracting all unique genres

Understanding the diversity of movie genres in the dataset.

genres <- str_extract_all(unique(edx$genres), "[^|]+") %>% unlist() %>% unique() n_distinct(genres) # Count of unique genres n_distinct(edx$genres) # Count of unique genre combinations

Data cleaning: Converting timestamp to date format and extracting movie release year

This step prepares the dataset for more time-oriented analyses.

edx <- edx %>% mutate(reviewDate = round_date(as_datetime(timestamp), unit = “day”)) %>% mutate(title = str_trim(title)) %>% extract(title, c(“shortTitle”, “year”), regex = “^(.) \ (([0-9 \-])\)$”, remove = FALSE) %>% mutate(year = as.integer(year)) %>% select(-shortTitle)

Calculating the delay in years between the movie review date and the movie’s release year

edx <- edx %>% mutate(reviewDelay = year(reviewDate) - year)

Data Visualization

In this section, we visually explore different aspects of the dataset to gain insights into the distribution of ratings, average ratings per movie and user, and other interesting trends. Each plot is crafted to provide a clear understanding of specific characteristics of the data.

```{r data-visualization} # Visualization: Rating Distribution # This plot shows the distribution of ratings across all movies. edx %>% ggplot(aes(rating)) + geom_histogram(binwidth=0.5, color=“black”) + labs(title = “Ratings Distribution”, x = “Rating Value”, y = “Count”)
```

Visualization: Average Ratings for Movies

Depicts how the average ratings are distributed across different movies.

edx %>% group_by(movieId) %>% summarise(mean_rating = mean(rating)) %>% ggplot(aes(mean_rating)) + geom_histogram(bins=50, color="black") + labs(title = "Distribution of Average Ratings for Movies", x = "Mean Rating", y = "Movie Count")

Visualization: Average Ratings Given by Users

Shows the distribution of average ratings given by individual users.

edx %>% group_by(userId) %>% summarise(mean_rating = mean(rating)) %>% ggplot(aes(mean_rating)) + geom_histogram(bins=50, color="black") + labs(title = "Distribution of Average Ratings Given by Users", x = "Mean Rating", y = "User Count")

Visualization: Number of Ratings per User

Illustrates how many ratings each user has given, on a logarithmic scale for clarity.

edx %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(bins=50, color="black") + scale_x_log10() + labs(title = "Distribution of Number of Ratings per User", x = "Number of Ratings", y = "User Count")

Visualization: Average Rating by Genre

This plot shows the average rating for each genre, along with error bars.

genres <- str_extract_all(unique(edx$genres), "[^|]+") %>% unlist() %>% unique() indiv_genres <- as.data.frame(genres) names(indiv_genres) <- c("genre") indiv_genres$n <- sapply(genres, function(g) { nrow(edx[str_detect(edx$genres, g),]) }) indiv_genres$meanRating <- sapply(genres, function(g) { mean(edx[str_detect(edx$genres, g), "rating"]) }) indiv_genres$sd <- sapply(genres, function(g) { sd(edx[str_detect(edx$genres, g), "rating"]) }) indiv_genres$se <- indiv_genres*sd/sqrt*¿ n) indiv_genres <- indiv_genres %>% arrange(desc(n)) indiv_genres %>% filter(genre!="(no genres listed)") %>% mutate(genre = reorder(genre, meanRating)) %>% ggplot(aes(x = genre, y = meanRating, ymin=meanRating - 2*se, ymax=meanRating* + 2se)) + geom_point() + geom_errorbar() + theme(axis.text.x = element_text(angle = 90, hjust = 1)) + labs(title = "Average Rating by Genre", x = "Genre", y = "Average Rating")

Visualization: Average Rating for Movies Across Different Release Years

Shows how the average rating for movies varies across different release years.

edx %>% group_by(year) %>% summarise(rating = mean(rating)) %>% ggplot(aes(year, rating)) + geom_point() + geom_smooth(formula='y~x', method='loess', span = 0.15) + labs(title = "Average Rating for Movies Across Different Release Years", x = "Release Year", y = "Average Rating")

Visualization: Distribution of Ratings Across Different Release Years

Illustrates the number of ratings movies receive across different release years.

edx %>% group_by(year) %>% summarise(n = n()) %>% ggplot(aes(year, n)) + geom_point() + scale_y_log10() + labs(title = “Distribution of Ratings Across Different Release Years”, x = “Release Year”, y = “Rating Count”)

Visualization: Average Rating Trend Over Time Based on Review Dates

This plot shows the trend of average ratings over time.

edx %>% group_by(reviewDate) %>% summarize(mean_rating = mean(rating)) %>% ggplot(aes(reviewDate, mean_rating)) + geom_point() + geom_smooth(formula=‘y~x’, method=‘loess’, span = 0.15) + labs(title = “Average Rating Trend Over Time Based on Review Dates”, x = “Review Date”, y = “Average Rating”)

Visualization: Average Rating Variance with Review Delays

Demonstrates how the average rating of movies varies with different review delays.

edx %>% group_by(reviewDelay) %>% summarise(mean_rating = mean(rating)) %>% ggplot(aes(reviewDelay, mean_rating)) + geom_point() + labs(title = “Average Rating Variance with Review Delays”, x = “Review Delay”, y = “Average Rating”)

Visualization: Number of Ratings Distributed Across Different Review Delays

Shows the distribution of the number of ratings over different review delays.

edx %>% group_by(reviewDelay) %>% summarise(n = n()) %>% ggplot(aes(reviewDelay, n)) + geom_point() + scale_y_log10()+ labs(title = “Number of Ratings Distributed Across Different Review Delays”, x = “Review Delay”, y = “Rating Count”)

Data Modelling

This section outlines the process of setting up our predictive models. We start by defining a function to calculate the Root Mean Squared Error (RMSE), a common measure of prediction accuracy. Then, we prepare our data by creating training and testing datasets.

```{r model-setup} # Defining the RMSE function for model evaluation RMSE <- function(true_ratings, predicted_ratings) { sqrt(mean((true_ratings - predicted_ratings)^2, na.rm=TRUE)) }

*Setting a seed for reproducibility*

set.seed(1)
```

Creating a partition to split the data into training and testing sets

d.index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE) d.train <- edx[-d.index,] d.test <- edx[d.index,]

Ensuring d.train and d.test are data frames

if (!is.data.frame(d.train)) stop(“d.train is not a data frame.”) if (!is.data.frame(d.test)) stop(“d.test is not a data frame.”)

Baseline Model

Before creating complex models, it’s crucial to establish a baseline for comparison. Here, we use the average movie rating as a simple prediction model. This model serves as our benchmark.

```{r baseline-model} # Calculating the average rating in the training data mu_hat <- mean(d.train$rating)

*Applying the simple model to the test set*

*Here, we predict every rating as the average rating from the training set*

simple_predictions <- rep(mu_hat, nrow(d.test))

*Calculating the RMSE for the baseline model*

rmse.simple <- RMSE(d.test$rating, simple_predictions) rmse.simple

## Data Modelling

This section details the development and evaluation of a series of predictive models, each building upon the previous one to increase complexity and potentially improve accuracy. We start with a simple model and progress through various models, incorporating different effects like movie, user, genre, release year, and review delay.

```{r predictive-modeling} # Defining the RMSE function RMSE <- function(true_ratings, predicted_ratings) { sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE)) }

*Setting up data partitions*

set.seed(1) d.index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE) d.train <- edx[-d.index, ] d.test <- edx[d.index, ]

*Checking if the datasets are data frames*

if (!is.data.frame(d.train)) stop(“d.train is not a data frame.”) if (!is.data.frame(d.test)) stop(“d.test is not a data frame.”)

### *Baseline Model (M01): Simple Average*

mu_hat <- mean(d.train*rating¿rmse.simple<−RMSE¿*rating, mu_hat)

### *M02: Adding Movie Effect*

avg.movies <- d.train %>% group_by(movieId) %>% summarise(movie_effect = mean(rating - mu_hat)) predicted.movie_effect <- d.test %>% left_join(avg.movies, by = “movieId”) %>% mutate(pred = mu_hat + movie_effect) %>% pull(pred) rmse.movie <- RMSE(d.test$rating, predicted.movie_effect)

### *M03: Adding User Effect*

avg.users <- d.train %>% left_join(avg.movies, by = “movieId”) %>% group_by(userId) %>% summarise(user_effect = mean(rating - mu_hat - movie_effect)) predicted.user_effect <- d.test %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% mutate(pred = mu_hat + movie_effect + user_effect) %>% pull(pred) rmse.user <- RMSE(predicted.user_effect, d.test$rating)

### *M04: Adding Genre Combination Effect*

avg.genres <- d.train %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% group_by(genres) %>% summarise(genre_effect = mean(rating - mu_hat - movie_effect - user_effect)) predicted.genre_effect <- d.test %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% left_join(avg.genres, by = “genres”) %>% mutate(pred = mu_hat + movie_effect + user_effect + genre_effect) %>% pull(pred) rmse.genre <- RMSE(predicted.genre_effect, d.test$rating)

### *M05: Adding Movie Release Year Effect*

avg.years <- d.train %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% left_join(avg.genres, by = “genres”) %>% group_by(year) %>% summarise(release_year_effect = mean(rating - mu_hat - movie_effect - user_effect - genre_effect)) predicted.release_year_effect <- d.test %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% left_join(avg.genres, by = “genres”) %>% left_join(avg.years, by = “year”) %>% mutate(pred = mu_hat + movie_effect + user_effect + genre_effect + release_year_effect) %>% pull(pred) rmse.year <- RMSE(predicted.release_year_effect, d.test$rating)

### *M06: Adding Review Delay Effect*

avg.delays <- d.train %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% left_join(avg.genres, by = “genres”) %>% left_join(avg.years, by = “year”) %>% group_by(reviewDelay) %>% summarise(review_delay_effect = mean(rating - mu_hat - movie_effect - user_effect - genre_effect - release_year_effect)) predicted.review_delay_effect <- d.test %>% left_join(avg.movies, by = “movieId”) %>% left_join(avg.users, by = “userId”) %>% left_join(avg.genres, by = “genres”) %>% left_join(avg.years, by = “year”) %>% left_join(avg.delays, by = “reviewDelay”) %>%

mutate(pred = mu_hat + movie_effect + user_effect + genre_effect + release_year_effect + review_delay_effect) %>% pull(pred) rmse.delay <- RMSE(predicted.review_delay_effect, d.test$rating)

### *Model 07: Regularization*

Regularization is used to refine our predictive models by penalizing the complexity of the model. This is done by introducing a regularization parameter, lambda, to control overfitting. We iterate over a range of lambda values to find the optimal balance.

```{r model-07} # Generating a sequence of lambda values inc <- 0.05 lambdas <- seq(4, 6, inc)

## Calculating RMSE for each lambda

rmses <- sapply(lambdas, function(l){ # Regularized model calculations movie_effect <- d.train %>% group_by(movieId) %>% summarise(movie_effect = sum(rating - mu_hat)/(n()+l)) user_effect <- d.train %>% left_join(movie_effect, by="movieId") %>% group_by(userId) %>% summarise(user_effect = sum(rating - movie_effect - mu_hat)/(n()+l)) genre_effect <- d.train %>% left_join(movie_effect, by="movieId") %>% left_join(user_effect, by="userId") %>% group_by(genres) %>% summarise(genre_effect = sum(rating - movie_effect - user_effect - mu_hat)/(n()+l)) release_year_effect <- d.train %>% left_join(movie_effect, by="movieId") %>% left_join(user_effect, by="userId") %>% left_join(genre_effect, by="genres") %>% group_by(year) %>% summarise(release_year_effect = sum(rating - movie_effect - user_effect - genre_effect - mu_hat)/(n()+l)) review_delay_effect <- d.train %>% left_join(movie_effect, by="movieId") %>% left_join(user_effect, by="userId") %>% left_join(genre_effect, by="genres") %>% left_join(release_year_effect, by="year") %>% group_by(reviewDelay) %>% summarise(review_delay_effect = sum(rating - movie_effect - user_effect - genre_effect - release_year_effect - mu_hat)/(n()+l)) predicted_ratings <- d.test %>% left_join(movie_effect, by="movieId") %>% left_join(user_effect, by="userId") %>% left_join(genre_effect, by="genres") %>% left_join(release_year_effect, by = "year") %>% left_join(review_delay_effect, by = "reviewDelay") %>% mutate(pred = mu_hat + movie_effect + user_effect + genre_effect + release_year_effect + review_delay_effect) %>% pull(pred) return(RMSE(predicted_ratings, d.test$rating)) })

## Identifying the best lambda value

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

### *Visualizing Lambda vs RMSE*

lambda_rmse_data <- as.data.frame(lambdas) lambda_rmse_data$rmses <- rmses names(lambda_rmse_data) <- c("lambdas", "rmses")

ggplot(lambda_rmse_data, aes(lambdas, rmses)) + geom_point() + xlab("Lambda") + ylab("RMSE") + geom_label_repel(data = subset(lambda_rmse_data, lambdas == lambda),
```

aes(label = lambdas), color = 'blue', size = 3.5, box.padding = unit(0.35, "lines"), point.padding = unit(0.3, "lines"))

Model Comparison

Finally, we compare the RMSE of all the models to assess their predictive performance. This helps in determining which model or combination of effects provides the most accurate predictions.

```{r model-comparison} # Compiling RMSE results from all models rmse.results <- tibble( Model = c("Simple Average", "Movie Effect", "User Effect", "Genre Combination Effect", "Release Year Effect", "Review Delay Effect", "Regularized"), RMSE = c(rmse.simple, rmse.movie, rmse.user, rmse.genre, rmse.year, rmse.delay, rmse.regularized) ) rmse.results

## Data Cleaning on Final Holdout Test Set

Before applying our models to the `final_holdout_test` dataset, we perform some essential data cleaning steps. This includes converting timestamps to date format, extracting movie release years, and calculating the delay between the review date and the movie's release year.

```{r data-cleaning-final-holdout} # Converting timestamp to date format final_holdout_test <- final_holdout_test %>% mutate(reviewDate = round_date(as_datetime(timestamp), unit = "week"))

### *Removing release year from the title and extracting it as a separate column*

final_holdout_test <- final_holdout_test %>% mutate(title = str_trim(title)) %>% extract(title, c("shortTitle", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = FALSE) %>% mutate(year = as.integer(year)) %>% select(-shortTitle)

### *Calculating the delay between the movie review date and the movie's release year*

final_holdout_test <- final_holdout_test %>% mutate(reviewDelay = year(reviewDate) - year)

## Applying the Regularized Model on Final Holdout Test Set

Now, we apply the regularized model to the `final_holdout_test` dataset. This model incorporates the effects based on movies, users, genres, release years, and review delays. The goal is to evaluate the model's performance on this final set of data, providing a comprehensive understanding of its predictive capability.

```{r model-application-final-holdout} # Applying regularized model components to final_holdout_test movie_effect <- edx %>% group_by(movieId) %>% summarise(movie_effect = sum(rating - mu_hat)/(n() + lambda)) user_effect <- edx %>%
```

left_join(movie_effect, by = “movieId”) %>% group_by(userId) %>% summarise(user_effect = sum(rating - movie_effect - mu_hat)/(n() + lambda)) genre_effect <- edx %>% left_join(movie_effect, by = “movieId”) %>% left_join(user_effect, by = “userId”) %>% group_by(genres) %>% summarise(genre_effect = sum(rating - movie_effect - user_effect - mu_hat)/(n() + lambda)) release_year_effect <- edx %>% left_join(movie_effect, by = “movieId”) %>% left_join(user_effect, by = “userId”) %>% left_join(genre_effect, by = “genres”) %>% group_by(year) %>% summarise(release_year_effect = sum(rating - movie_effect - user_effect - genre_effect - mu_hat)/(n() + lambda)) review_delay_effect <- edx %>% left_join(movie_effect, by = “movieId”) %>% left_join(user_effect, by = “userId”) %>% left_join(genre_effect, by = “genres”) %>% left_join(release_year_effect, by = “year”) %>% group_by(reviewDelay) %>% summarise(review_delay_effect = sum(rating - movie_effect - user_effect - genre_effect - release_year_effect - mu_hat)/(n() + lambda))

Predicting ratings on final_holdout_test

predicted.final <- final_holdout_test %>% left_join(movie_effect, by = “movieId”) %>% left_join(user_effect, by = “userId”) %>% left_join(genre_effect, by = “genres”) %>% left_join(release_year_effect, by = “year”) %>% left_join(review_delay_effect, by = “reviewDelay”) %>% mutate(pred = mu_hat + movie_effect + user_effect + genre_effect + release_year_effect + review_delay_effect) %>% pull(pred)

Evaluating model performance on final_holdout_test

rmse.fht <- RMSE(final_holdout_test$rating, predicted.final)