Unveiling Movie Preferences:

Analyzing MovieLens Dataset with R

Efren B. Abantao

November 22, 2023

# Introduction:

MovieLens stands as a rich reservoir of film ratings and tag applications meticulously assembled by researchers at the University of Minnesota. Initially crafted to revolutionize personalized movie suggestions, this dataset has evolved into a treasure trove of cinematic preferences. With its extensive collection of over 25 million ratings and a million tag applications encompassing 62,000 movies and 162,000 users, MovieLens presents a fertile ground for exploring and understanding user behavior and preferences in the realm of cinema.

Delving into MovieLens isn't merely about numbers—it's an expedition into the intricate tapestry of human tastes and tendencies. Through the lens of artificial intelligence and data analysis, the vast expanse of MovieLens becomes more than a repository of ratings; it morphs into a playground for insights and revelations about how individuals engage with and perceive movies.

In the realm of AI analysis, MovieLens serves as a captivating case study. It represents an opportunity to wield sophisticated machine learning models, unlocking the potential to forecast movie preferences, recommend undiscovered gems, and comprehend the nuances of user choices. The dataset's sheer size and complexity beckon data scientists and AI enthusiasts to employ cutting-edge methodologies, refining algorithms to not just predict ratings but to decode the intricate fabric of cinematic tastes.

Unraveling MovieLens through the lens of AI analysis isn't just about crunching numbers—it's about deciphering human inclinations, aspirations, and the enigmatic realm of cinematic preferences through the remarkable prism of data science and artificial intelligence.

[My GitHub Repo](https://github.com/abantao61677/MovieLens_Capstone)

# Objective:

To conduct a comprehensive analysis of the MovieLens dataset using R programming, encompassing exploratory analysis, predictive modeling for rating predictions, and the creation of a recommendation system based on user preferences.

# Tools and Packages:

# Define necessary R packages

required\_packages <- c("tidyverse", "caret", "stringr", “recommenderlab”, “shiny”)

# Identify packages that aren't installed

missing\_packages <- required\_packages[!(required\_packages %in% installed.packages()[,"Package"])]

# Install missing packages, if any

if(length(missing\_packages)) {

install.packages(missing\_packages)

}

# Load required libraries

invisible(lapply(required\_packages, function(pkg) {

if(!require(pkg, character.only = TRUE, quietly = TRUE)) {

install.packages(pkg)

library(pkg, character.only = TRUE)

} else {

library(pkg, character.only = TRUE)

}

}))

# 1. Data Download and Extraction:

Acquire the MovieLens dataset, specifically the 10M subset, and load it into R and extract using the following codes.

```R  
# Set the working directory  
setwd("path/to/directory")

# Download the dataset  
url <- "<http://files.grouplens.org/datasets/movielens/ml-10m.zip>"  
download.file(url, "ml-10m.zip")

# Extract the dataset

unzip("ml-10m.zip")  
```

# 2. Data Cleaning and Preprocessing:

R is meant to clean and preprocess data from the MovieLens dataset, specifically the "movies.dat" and "ratings.dat" files using tidyverse and stringr packages broken-down according to the part it does:

* Reading Movies Data: It reads the "movies.dat" file using read.table. It specifies the separator as :: since it seems the file is using that as a delimiter. The columns are named as "movieId", "title", and "genres".

```R  
# Read movies data  
movies <- read.table("ml-10M100K/movies.dat", sep="::", header=FALSE, stringsAsFactors=FALSE,  
col.names=c("movieId", "title", "genres"))  
```

* Reading Ratings Data: Similarly, it reads the "ratings.dat" file using read.table. The columns are named as "userId", "movieId", "rating", and "timestamp".

```R  
# Read ratings data  
ratings <- read.table("ml-10M100K/ratings.dat", sep="::", header=FALSE, stringsAsFactors=FALSE,  
col.names=c("userId", "movieId", "rating", "timestamp"))  
```

* Identify Missing Values

```R

# Check for missing values in ratings dataset

any(is.na(ratings))

```

* Dealing with Missing Values

```R

# Impute missing values in the 'rating' column with mean

ratings$rating[is.na(ratings$rating)] <- mean(ratings$rating, na.rm = TRUE)

```

* Removing Missing Values

```R

# Remove rows with any missing values in ratings dataset

ratings <- ratings[complete.cases(ratings), ]

```

* Handling Missing Values

```R

# Create imputed dataset

imputed\_ratings <- mice(ratings)

```

* Cleaning and Preprocessing Movies Data: The mutate function is used here to clean the "genres" column in the movies dataset. It replaces the | separator within the "genres" column with a , using str\_replace\_all. Then, separate\_rows is applied to split the genres separated by commas into separate rows, effectively creating multiple rows for a movie if it falls under multiple genres.

```R  
# Clean and preprocess data  
movies <- movies %>%  
mutate(genres = str\_replace\_all(genres, "\\|", ", ")) %>%  
separate\_rows(genres, sep = ", ")  
```

* Cleaning Ratings Data: For the ratings dataset, it selects only the columns "userId", "movieId", and "rating", discarding the "timestamp" column.

```R  
# Clean ratings data  
ratings <- ratings %>%  
select(userId, movieId, rating)  
```

This code cleans the genre information in the movies dataset by transforming the data from a single row with pipe-separated genres to multiple rows for each genre associated with a movie. It also simplifies the ratings dataset by keeping only the necessary columns for analysis.

Further data analysis or modeling could be applied to these cleaned datasets for tasks such as recommendation systems or exploratory data analysis related to movie ratings and genres.

# 3. Data Visualization:

This part of the code groups the movies dataset (**movies**) by the "genres" column and counts the number of occurrences of each genre using **summarise** with **n()** function, which counts the number of rows for each group.

```R  
# Count movies by genre  
genre\_counts <- movies %>%  
 group\_by(genres) %>%  
 summarise(count = n())

```

The code below is using ggplot2, a powerful visualization package in R, to create a bar plot illustrating the count of movies for each genre. It's a helpful way to understand the distribution of movies across different genres in the dataset.

The reorder() function is used to order the genres by count, presenting the genres with the highest movie count on the left side of the plot.

Adjustments to colors, themes, or other plot aesthetics can be made as needed to enhance readability or match specific visualization preferences.

This plot provides a quick overview of the distribution of movies across genres, allowing for easy comparison and identification of the most prevalent genres within the dataset.

```R

# Plot count of movies by genre  
ggplot(data = genre\_counts, aes(x = reorder(genres, -count), y = count)) +  
geom\_bar(stat = "identity", fill = "blue") +  
theme(axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1)) +  
labs(x = "Genres", y = "Count", title = "Count of Movies by Genre")  
```

# 4. Collaborative Filtering using RecommenderLab:

To build a recommendation system, create recommender model from the installed recommenderlab package:

```R  
# Create a recommender model using collaborative filtering  
recommender\_model <- Recommender(data = ratings, method = "UBCF")

# UBCF stands for User-Based Collaborative Filtering.

# Get the top 5 movie recommendations for user 1  
recommendations <- predict(object = recommender\_model, newdata = ratings[1:5,], n = 5)

#Retrieving Top Recommendations for Specific Users:

#To get recommendations for a specific user (e.g., user 1):

user\_id <- 1

user\_ratings <- ratings[ratings$userId == user\_id, ]

recommendations\_user1 <- predict(object = recommender\_model, newdata = user\_ratings, n = 5)

# This code filters the ratings dataset for user 1's ratings and then generates recommendations for that specific user.

# To evaluate the model's performance using cross-validation (k = 5 folds) on the ratings dataset to assess the performance of the recommendation model:

evaluation <- evaluate(ratings, method = "cross-validation", k = 5)

# To create a model using item-based collaborative filtering ("IBCF"):

recommender\_model\_ibcf <- Recommender(data = ratings, method = "IBCF")

# These additional functionalities allow for more specific recommendations, model evaluation, and customization of the recommendation system to suit specific needs or improve its performance.  
```

# 5. Model Evaluation using RMSE:

To evaluate the model, you can calculate the Root Mean Squared Error (RMSE) between the predicted ratings and the actual ratings using the installed caret package:

```R  
# Split the data into training and test sets (80:20)  
set.seed(123, sample.kind = “Rounding”)  
trainIndex <- createDataPartition(ratings$rating, p = 0.8, list = FALSE)  
trainData <- ratings[trainIndex, ]  
testData <- ratings[-trainIndex, ]

# Train the model on the training data  
model <- train(rating ~ ., data = trainData, method = "knn", trControl = trainControl(method = "cv", number = 5))

# Create a grid of hyperparameters

grid <- expand.grid(k = seq(1, 30, by = 1))

# Get the best model

best\_model <- model$finalModel

# Make predictions on the test data  
predictions <- predict(best\_model, newdata = testData)

# Calculate RMSE  
rmse <- RMSE(predictions, testData$rating)  
```

# 6. Visualizing predicted ratings against the actual ratings:

#This code creates a scatterplot to visualize the relationship between predicted and actual ratings.

```R

plot(predictions, testData$rating)

```

#Finally, let’s create a Shiny app to provide interactive movie recommendations to users.   
```R  
# Define UI  
ui <- fluidPage(  
titlePanel("Movie Recommendation System"),  
sidebarLayout(  
sidebarPanel(

# Add UI components for user input (e.g., user ID, number of recommendations)  
),  
mainPanel(

# Add UI components for displaying movie recommendations  
)  
)  
)

# Define server  
server <- function(input, output) {

# Add server logic for generating movie recommendations  
}

# Run the shiny app  
shinyApp(ui = ui, server = server)  
```

# Conclusion:

This Data Science Capstone project aims to leverage R programming to explore the MovieLens dataset comprehensively. Through exploratory analysis, predictive modeling for rating predictions, and the creation of a recommendation system, the project seeks to derive actionable insights that can contribute to enhancing user experiences and recommendation algorithms in the realm of movie streaming platforms.