MovieLens_Capstone_Project_2023

Suchitra Chavan

Dec. 2023

Table of Contents

1. Introduction
2. Methods and Analysis
3. Results
4. Conclusion
5. References

Introduction

What is Movielens?

MovieLens is an online platform designed to assist individuals in discovering movies for their viewing pleasure link. The platform offers an extensive selection of movies tailored to users' preferences, utilizing various search criteria such as genres and personalized recommendations based on individual users' MovieLens account profiles and activity. This platform is run by GroupLens link a research lab at University of Minnesota in the Department of Computer Science and Engineering. As a research laboratory they focus on several areas, including recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

With a substantial user base consisting of hundreds of thousands of registered members, MovieLens serves as an ideal platform for conducting various online field experiments. These experiments cover a wide range of topics including automated content recommendation, recommendation interfaces, tagging-based recommenders and interfaces, member-maintained databases, and intelligent user interface design.

What is MovieLens datasets?

The MovieLens datasets have gained extensive usage across various domains, including education, research, and industry. With hundreds of thousands of annual downloads, these datasets have become integral to popular press programming books, both traditional and online courses, as well as software applications. These datasets are a direct outcome of member interactions within the MovieLens movie recommendation system, which has served as a dynamic research platform hosting numerous experiments since its inception in 1997.

Different Types of MovieLens Dataset.

Different types of MovieLens dataset are available for use [link] (https://grouplens.org/datasets/movielens/) e.g. MovieLens 25M Dataset, MovieLens latest datasets, Movielens 1B synthetic Dataseet, MovieLens 100K dataset, Movielens 1M dataset and MovieLens 10M Dataset etc.

MovieLens 10M Dataset

This is the dataset that we used for the initial part of the project. This was used to create the edx and final_holdout_test sets. This is a benchmark dataset that provides stable and reliable reference point for analysis. It consists of a vast collection of 10 million ratings and 100,000 tag applications applied to 10,000 movies contributed by diverse community of 72,000 users. This dataset was release in Jan. 2009, offering a comprehensive snapshot for research and evaluation purpose link.

The dataset encompasses a total of 10,000,054 ratings and 95,580 tags associated with 10,681 movies. These ratings and tags were provided by 71,567 users who actively engaged with the MovieLens online movie recommender service. The users included in the dataset were chosen randomly and required to have rated a minimum of 20 movies, ensuring a substantial level of user activity and contribution.

The data are contained in three files, movies.dat, ratings.dat and tags.dat. This project makes use of only the first two data (movies.dat and ratings.dat)

What is the data in these files?

1. User Ids - Movielens users were selected at random for inclusion.
2. Ratings data - Ratings are made on a 5-star scale, with half-star increments.
3. Movies data - Contains movie information. The Genres are Action, Adventure,Animation,Children's,Comedy,Crime,Documentary,Drama,Fantasy,Film-Noir,Horror,Musical,Mystery,Romance,Sci-Fi,Thriller,War and Western link

Overall goal

The objective of this project is to train a machine learning algorithm capable of predicting user ratings (ranging from 0.5 to 5 stars). This prediction will be based on a provided subset of data (specifically the edx dataset provided by the staff), and the algorithm's performance will be evaluated in predicting movie ratings within a given validation set.

The primary metric for assessing algorithm performance is the Root Mean Square Error (RMSE). RMSE is a widely used measure for quantifying the differences between predicted values generated by a model and the actual observed values. In the context of this project, a lower RMSE is indicative of higher accuracy. This metric allows for the comparison of forecasting errors across different models for a specific dataset.

It is essential to note that RMSE is particularly sensitive to outliers, as the impact of each error on the overall RMSE is proportional to the squared error's magnitude. Consequently, larger errors carry a disproportionately greater weight in influencing the RMSE. To evaluate and compare the quality of the four models developed in this project, their respective RMSE values will be considered. The model with the lowest RMSE is considered superior, indicating more accurate predictions.

Steps to accomplish the goal.

1. Generate and explore the datasets
2. Develop the algorith using the edx set.
3. Split the edx data into separate training and test sets and/or use cross-validation to design and test your algorithm.
4. Use the final_holdout-test set with the final model.

Methods and Analysis

Generate the datasets

Some of the code provided below was given to us to get started. In, order to check if the required code was executed and dataset was created, I regularly checked the dataset and results are published below.

```
##########################################################
# Create edx and final_holdout_test sets
##########################################################
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us
.r-project.org")
library(tidyverse)
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-proje
ct.org")
library(caret)
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-proje
ct.org")
library(readr)
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-proje
ct.org")
library(dplyr)

# MovieLens 10M dataset:
#creating dl and printing to see if it was created
dl <- "ml-10M100K.zip"
if(! file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
dl)
print(dl)

#creating ratings_file and print to see if it was created.
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)
print(ratings_file)
```

```
# creating movies_file and printing to check if it was created
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)
print(movies_file)

# creating ratings, note the header here. Use head to check the header and da
taset
ratings <- read_delim(ratings_file, delim = "::", col_names = c("UserID", "Mo
vieID", "Rating", "Timestamp"))
head(ratings)

# creating movies and head movies to check if it is created and also the head
er is correct.
movies <- read_delim(movies_file, delim = "::", col_names = c("MovieID", "Tit
le", "Genres"))
head(movies)

#code had changes slightly from the edex with the MovieID and others titles u
sing caps locks. I dont think this helped a lot.
colnames(movies) <- c("MovieID", "Title", "Genres")
head(movies)

#movies dataset if transformed and head to check the titles. See the change i
n MovieID.
movies <- transform(movies, MovieID = as.integer(MovieID))
head(movies)

#to confirm it became an integer
class(movies)
class(movies$MovieID)


#creating movielens dataset
movielens <- left_join(ratings, movies, by = "MovieID")
head(movielens)

# Final hold-out test set will be 10% of MovieLens data
set.seed(1,sample.kind="Rounding")
# above code gave a warning so used the below code.
set.seed(1)

#creating test_index and head test_index. Note the change in movielens$Rating
s (capital R to match my headers)
test_index <- createDataPartition(y = movielens$Rating, times = 1, p = 0.1, l
ist = FALSE)

#checking to see what the dataset looks like
```

```
head(test_index)

#creating edx dataset
edx <- movielens[-test_index,]
head(edx)

# creating tmp dataset
temp <- movielens[test_index,]
head(temp)

#creat final_holdout_test
final_holdout_test <- temp %>%
  semi_join(edx, by = "MovieID") %>%
  semi_join(edx, by = "UserID")
head(final_holdout_test)

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)
```

Results from creating the edx and final_holdout_test sets.

These results not only help learn about the datasets but also provide confirmation that datasets are being created correctly.

```
head(ratings)
# A tibble: 6 × 4
  UserID MovieID Rating Timestamp
   <dbl>   <dbl>  <dbl>     <dbl>
1      1     122      5 838985046
2      1     185      5 838983525
3      1     231      5 838983392
4      1     292      5 838983421
5      1     316      5 838983392
6      1     329      5 838983392

> head(movies)
# A tibble: 6 × 3
  MovieID Title                                 Genres
    <dbl> <chr>                                 <chr>
1       1 Toy Story (1995)                      Adventure|Animation|Children|Com
edy|Fantasy
2       2 Jumanji (1995)                        Adventure|Children|Fantasy
3       3 Grumpier Old Men (1995)               Comedy|Romance
4       4 Waiting to Exhale (1995)              Comedy|Drama|Romance
5       5 Father of the Bride Part II (1995) Comedy
6       6 Heat (1995)                           Action|Crime|Thriller

> head(movies)
  MovieID                             Title
```

```
Genres
1       1                   Toy Story (1995) Adventure|Animation|Children|Com
edy|Fantasy
2       2                     Jumanji (1995)                  Adventure|Child
ren|Fantasy
3       3            Grumpier Old Men (1995)                              Com
edy|Romance
4       4           Waiting to Exhale (1995)                        Comedy|Dr
ama|Romance
5       5 Father of the Bride Part II (1995)                                 
Comedy
6       6                        Heat (1995)                       Action|Cri
me|Thriller

> class(movies)
[1] "data.frame"
> class(movies$MovieID)
[1] "integer"

> head(movielens)
# A tibble: 6 × 6
  UserID MovieID Rating Timestamp Title                         Genres
   <dbl>   <dbl>  <dbl>     <dbl> <chr>                         <chr>
1      1     122      5 838985046 Boomerang (1992)              Comedy|Romanc
e
2      1     185      5 838983525 Net, The (1995)               Action|Crime|
Thriller
3      1     231      5 838983392 Dumb & Dumber (1994)          Comedy
4      1     292      5 838983421 Outbreak (1995)               Action|Drama|
Sci-Fi|Thriller
5      1     316      5 838983392 Stargate (1994)               Action|Advent
ure|Sci-Fi
6      1     329      5 838983392 Star Trek: Generations (1994) Action|Advent
ure|Drama|Sci-Fi

> # Final hold-out test set will be 10% of MovieLens data
> set.seed(1,sample.kind="Rounding")
Warning message:
In set.seed(1, sample.kind = "Rounding") :
  non-uniform 'Rounding' sampler used

> head(test_index)
    Resample1
[1,]         3
[2,]        15
[3,]        18
[4,]        24
[5,]        36
[6,]        42
```

```
> head(edx)
# A tibble: 6 × 6
 UserID MovieID Rating Timestamp Title                         Genres
  <dbl>   <dbl>  <dbl>     <dbl> <chr>                         <chr>
1      1     122      5 838985046 Boomerang (1992)              Comedy|Romanc
e
2      1     185      5 838983525 Net, The (1995)               Action|Crime|
Thriller
3      1     292      5 838983421 Outbreak (1995)               Action|Drama|
Sci-Fi|Thriller
4      1     316      5 838983392 Stargate (1994)               Action|Advent
ure|Sci-Fi
5      1     329      5 838983392 Star Trek: Generations (1994) Action|Advent
ure|Drama|Sci-Fi
6      1     355      5 838984474 Flintstones, The (1994)       Children|Come
dy|Fantasy

> head(final_holdout_test)
# A tibble: 6 × 6
  UserID MovieID Rating Timestamp Title
Genres
   <dbl>   <dbl>  <dbl>     <dbl> <chr>
<chr>
1      1     231      5 838983392 Dumb & Dumber (1994)
Comedy
2      1     480      5 838983653 Jurassic Park (1993)
Action|Adven…
3      1     586      5 838984068 Home Alone (1990)
Children|Com…
4      2     151      3 868246450 Rob Roy (1995)
Action|Drama…
5      2     858      2 868245645 Godfather, The (1972)
Crime|Drama
6      2    1544      3 868245920 Lost World: Jurassic Park, The (Jurassic Pa
rk 2) (1997) Action|Adven…
```

Investigating the edx dataset

In addition to the above checks to ensure the dataset is created, we will investigate the edx dataset using simple functions to explore the dataset and help us create the machine learning algorithm.

```
summary(edx)
     UserID          MovieID          Rating         Timestamp            Title
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08   Length:9
000055
 1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08   Class :c
haracter
 Median :35738   Median : 1834   Median :4.000   Median :1.035e+09   Mode  :c
haracter
```

```
 Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
 3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
    Genres         
 Length:9000055    
 Class :character  
 Mode  :character  
```

Movie rating output

This shows us that the lowest moving rating is 0.5 and the highest is 5. It also provides the mean (3.51) and median values(4).

```
edx %>%
  summarize(
    Lowest_Rating = min(Rating),
    Highest_Rating = max(Rating),
    Mean_Rating = mean(Rating),
    Median_Rating = median(Rating)
  )

output is
A tibble: 1 × 4
  Lowest_Rating Highest_Rating Mean_Rating Median_Rating
          <dbl>          <dbl>       <dbl>         <dbl>
1           0.5              5        3.51             4
```

Group data by "Rating"

Lets group the data by "Rating" column and calculate the count of each rating category, selecting the top 5 rating categories with the highest counts and then arranging in descending order of counts.

```
# Group data by "Rating" top_5
edx %>% group_by(Rating) %>% summarize(count = n()) %>% top_n(5) %>%
  arrange(desc(count))
```

Output is this

```
# A tibble: 5 × 2
  Rating   count
   <dbl>   <int>
1    4   2588430
2    3   2121240
3    5   1390114
4    3.5  791624
5    2    711422
```

I tried the above for 10 rating categories as well.

```
# Group data by "Rating" top_10
edx %>% group_by(Rating) %>% summarize(count = n()) %>% top_n(10) %>%
  arrange(desc(count))
```

Output is

```
Selecting by count
# A tibble: 10 × 2
   Rating   count
    <dbl>   <int>
 1    4   2588430
 2    3   2121240
 3    5   1390114
 4    3.5  791624
 5    2    711422
 6    4.5  526736
 7    1    345679
 8    2.5  333010
 9    1.5  106426
10    0.5   85374
```

Calculating the number of distinct users and the number of distinct movies in the dataset.

```
# Distinct users and movies
edx %>%
  summarize(n_users = n_distinct(UserID),
            n_movies = n_distinct(MovieID))
```

Output is

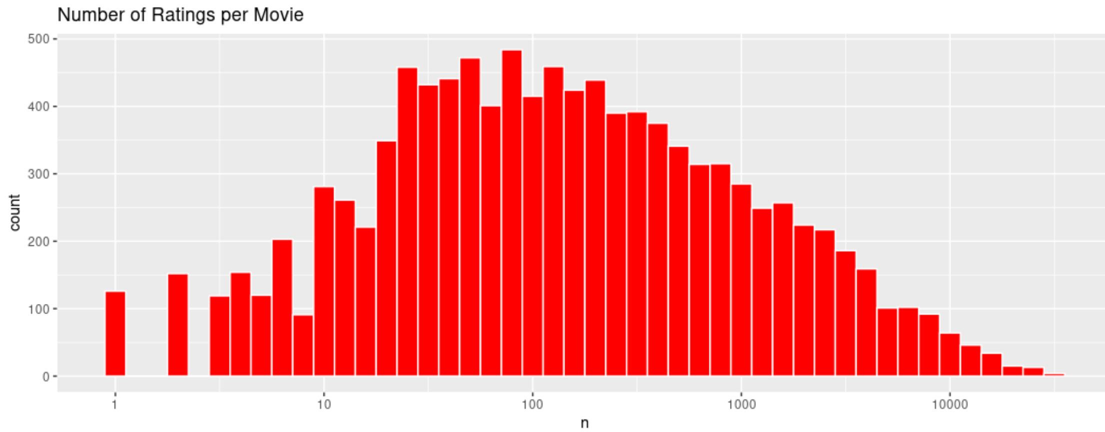
```
# A tibble: 1 × 2
  n_users n_movies
    <int>    <int>
1   69878    10677
```

Number of Ratings per Movie

```
# Number of ratings per movie.
# Count the number of ratings per movie
ratings_per_movie <- edx %>% count(MovieID)

# Create the histogram plot using ggplot
plot <- ggplot(data = ratings_per_movie, aes(x = n)) +
  geom_histogram(color = "white", fill = "red", bins = 25, binwidth = 0.1) +
  scale_x_log10() +
  ggtitle("Number of Ratings per Movie")

# Display the plot for number of ratings per movie.
print(plot)
```



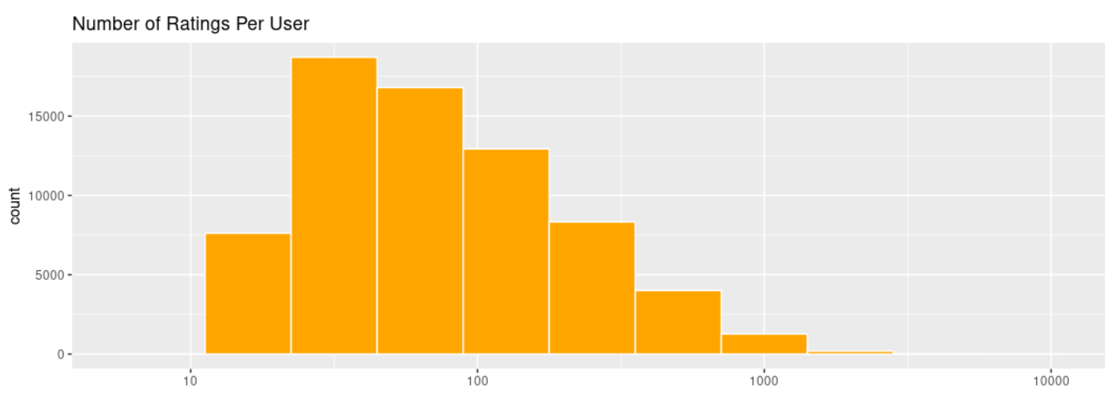
The number of ratings per movie plot is heavily skewed to the right and that there are some movies that are recieving disporportionate amount of attention.

Calculates the number of ratings per user

```
#Calculate the number of ratings per user.
# Count the number of ratings per user
ratings_per_user <- count(edx, UserID)

# Create the histogram plot using ggplot
plot2 <- ggplot(data = ratings_per_user, aes(x = n)) +
  geom_histogram(color = "white", fill = "orange", bins = 20, binwidth = 0.3)
+
  ggtitle("Number of Ratings Per User") +
  scale_x_log10()

# Display the plot for number of ratings per uesr.
print(plot2)
```



Similar to the above plot, the number of ratings per user is skewed to the right, with some users making very few reviews and some users reviewing up to thousands of movies.

Plot the ratings for each movie genre.

```
# Plot the ratings for each movie genre.
 # Separate genres column into rows.
```

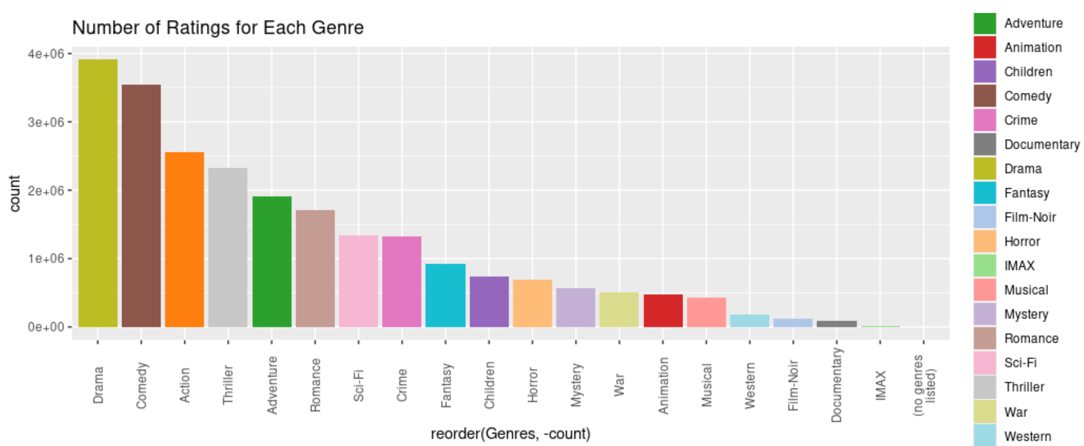
```
edx_separated <- separate_rows(edx,Genres, sep ="\\|")

# Count the occurrences of each genre
genres_counts <- edx_separated %>%
  group_by(Genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))

# Create a custom color palette with 20 colors
custom_colors <- c(
  "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd",
  "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22", "#17becf",
  "#aec7e8", "#ffbb78", "#98df8a", "#ff9896", "#c5b0d5",
  "#c49c94", "#f7b6d2", "#c7c7c7", "#dbdb8d", "#9edae5"
)

# Create the bar plot using ggplot with custom colors
plot <- ggplot(data = genres_counts, aes(x = reorder(Genres, -count), y = cou
nt, fill = Genres)) +
  geom_bar(stat = "identity") +
  labs(title = "Number of Ratings for Each Genre") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  scale_fill_manual(values = custom_colors) +  # Use the custom colors
  scale_x_discrete(labels = function(x) str_wrap(x, width = 10))

# Print the plot for number of ratings per genre
print(plot)
```



The above plot shows the number of ratings for each genre, with the drama (war) Genre with the highest ratings followed by Comedy and Action.

Creating training and test sets using edx dataset

The Movielens dataset is split into edx (training) and final_holdout_test(validation). Within the edx(training) dataset there will be another split for internal training (edx_train) and testing (edx_temp). The final_holdout_test (validation) dataset will be utilized only on the final models.

```
#################################################
## Creating training and test sets using edx data set.
##########################################################

# Set a random seed for reproducibility
set.seed(1, sample.kind = "Rounding")
# Create the test set index
test_index <- createDataPartition(y = edx$Rating, times = 1, p = 0.1, list =
FALSE)
summary(test_index)
head(test_index)

# Create the training set (edx_train) and test set (edx_temp)
edx_train <- edx[-test_index, ]
edx_temp <- edx[test_index, ]

head(edx_train)
colnames(edx_train)
summary(edx_train)


head(edx_temp)
colnames(edx_temp)
summary(edx_temp)

# output is
summary(test_index)
   Resample1
 Min.   :      2
 1st Qu.:2247780
 Median :4501700
 Mean   :4499824
 3rd Qu.:6749418
 Max.   :9000044
> head(test_index)
     Resample1
[1,]         2
[2,]        21
[3,]        24
[4,]        32
[5,]        34
[6,]        40
>
> # Create the training set (edx_train) and test set (edx_temp)
> edx_train <- edx[-test_index, ]
> edx_temp <- edx[test_index, ]
>
```

```
> head(edx_train)
# A tibble: 6 × 6
  UserID MovieID Rating Timestamp Title                         Genres
   <dbl>   <dbl>  <dbl>     <dbl> <chr>                         <chr>
1      1     122      5 838985046 Boomerang (1992)              Comedy|Romanc
e
2      1     292      5 838983421 Outbreak (1995)               Action|Drama|
Sci-Fi|Thriller
3      1     316      5 838983392 Stargate (1994)               Action|Advent
ure|Sci-Fi
4      1     329      5 838983392 Star Trek: Generations (1994) Action|Advent
ure|Drama|Sci-Fi
5      1     355      5 838984474 Flintstones, The (1994)       Children|Come
dy|Fantasy
6      1     356      5 838983653 Forrest Gump (1994)           Comedy|Drama|
Romance|War
> colnames(edx_train)
[1] "UserID"    "MovieID"   "Rating"    "Timestamp" "Title"     "Genres"
> summary(edx_train)
     UserID         MovieID          Rating        Timestamp            Title
Genres
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08   Length:8
100048     Length:8100048
 1st Qu.:18127   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08   Class :c
haracter   Class :character
 Median :35732   Median : 1834   Median :4.000   Median :1.035e+09   Mode  :c
haracter   Mode  :character
 Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53607   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
>
>
> head(edx_temp)
# A tibble: 6 × 6
  UserID MovieID Rating  Timestamp Title
Genres
   <dbl>   <dbl>  <dbl>      <dbl> <chr>
<chr>
1      1     185      5  838983525 Net, The (1995)
Action|Crime|Thriller
2      2     260      5  868244562 Star Wars: Episode IV - A New Hope (a.k.a.
Star Wars) (1977) Action|Adventure|Sci-Fi
3      2     590      5  868245608 Dances with Wolves (1990)
Adventure|Drama|Western
4      2    1049      3  868245920 Ghost and the Darkness, The (1996)
Action|Adventure
5      2    1210      4  868245644 Star Wars: Episode VI - Return of the Jedi
(1983)             Action|Adventure|Sci-Fi
6      3    1148      4 1133571121 Wallace & Gromit: The Wrong Trousers (1993
)                   Animation|Children|Comedy|Crime
```

```
> colnames(edx_temp)
[1] "UserID"    "MovieID"   "Rating"    "Timestamp" "Title"     "Genres"
> summary(edx_temp)
     UserID         MovieID          Rating        Timestamp            Title
Genres
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08   Length:9
00007      Length:900007
 1st Qu.:18106   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08   Class :c
haracter   Class :character
 Median :35761   Median : 1834   Median :4.000   Median :1.036e+09   Mode  :c
haracter   Mode  :character
 Mean   :35868   Mean   : 4134   Mean   :3.513   Mean   :1.033e+09
 3rd Qu.:53598   3rd Qu.: 3638   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65130   Max.   :5.000   Max.   :1.231e+09
```

Model testing

1. Naive Model

Creating and evaluating naive models based on mean and median ratings using the edx_train dataset. It calculates the mean and median ratings for the training set, generates naive predictions by replicating these values for all items, and subsequently computes the Root Mean Squared Error (RMSE) for each naive model. The RMSE serves as a measure of prediction accuracy, providing a baseline performance comparison for more advanced recommendation models. These naive models offer straightforward benchmarks, and comparing their RMSE to those of more complex models helps assess the effectiveness of sophisticated recommendation strategies in improving predictive accuracy.

1. Naive model on edx_train

```
###################################################
## Model Creation and Evaluation on edx_train dataset.
###############################################
### 1. Naive Model on edx_train
#####################################################
# Calculate the mean rating in the edx_train dataset
mean_rating <- mean(edx_train$Rating)
print(mean_rating)

#output is print(mean_rating) [1] 3.512457
# Create a vector of the naive predictions (using the mean rating for all ite
ms)
naive_predictions <- rep(mean_rating, nrow(edx_train))

# Calculate RMSE
naive_rmse <- sqrt(mean((edx_train$Rating - naive_predictions) ^ 2))
print(naive_rmse)
#output is print(naive_rmse) [1] 1.060362

## Naive model using median value.
```

```
# Calculate the median rating in the edx_train dataset
median_rating <- median(edx_train$Rating)
print(median_rating)
#output is print(median_rating) [1] 4

# Create a vector of the median predictions (using the median rating for all
items)
median_predictions <- rep(median_rating, nrow(edx_train))
# Calculate RMSE
median_rmse <- sqrt(mean((edx_train$Rating - median_predictions) ^ 2))
print(median_rmse)
print(median_rmse) [1] 1.167076
```

1. Naive model on edx_temp

```
#############################################################
# Naive Model on edx_temp
################################################################
# Calculate the mean rating in the edx_temp dataset
mean_rating_temp <- mean(edx_temp$Rating)
print(mean_rating_temp)
# output print(mean_rating_temp) [1] 3.512541


# Create a vector of the naive predictions (using the mean rating for all ite
ms)
naive_predictions_temp <- rep(mean_rating_temp, nrow(edx_temp))


# Calculate RMSE for the naive model on edx_temp
naive_rmse_temp <- sqrt(mean((edx_temp$Rating - naive_predictions_temp) ^ 2))
print(naive_rmse_temp)
#output print(naive_rmse_temp) [1] 1.060056

## Naive model using median value on edx_temp
# Calculate the median rating in the edx_temp dataset
median_rating_temp <- median(edx_temp$Rating)
print(median_rating_temp)
# output print(median_rating_temp)[1] 4

# Create a vector of the median predictions (using the median rating for all
items)
median_predictions_temp <- rep(median_rating_temp, nrow(edx_temp))
# Calculate RMSE for the naive model using median on edx_temp
median_rmse_temp <- sqrt(mean((edx_temp$Rating - median_predictions_temp) ^ 2
))
print(median_rmse_temp)
#output print(median_rmse_temp) [1] 1.166763
```

The above RMSE establishes a baseline for us to build the models.

2. Movie Effect Model on edx_train

Movie effect model on the edx_train dataset, aiming to capture variations in user ratings by estimating the impact of individual movies on user preferences. The code calculates movie effects (bi) by measuring the deviation of each movie's ratings from the overall average. These effects are then merged with the training dataset, and predicted ratings (yu,i) are generated based on the average rating and movie effects. The resulting Root Mean Squared Error (RMSE) is calculated as a performance metric, assessing the accuracy of the model in predicting user ratings.

The movie effect model presented in the code can be expressed through the following equation:

$y(\mu,i)=\mu+ b_i$

Where:

$y(\mu,i)$ is the predicted rating for user μ on movie i.

μ is the average rating across all movies and users.

b_i represents the movie effect for movie i, indicating the deviation of its ratings from the overall average.

The average rating (μ) is calculated as average_rating.

The movie effects (b_i) are computed as movie_effects$movie_effect.

The predicted ratings ($y\mu$,i) are obtained by adding the average rating and the movie effect: edx_train$predicted_rating.

The obtained RMSE value, such as [1] 0.9423541, provides a quantitative measure of the model's predictive accuracy. A lower RMSE indicates that the movie effect model is better at predicting user ratings compared to a simple mean or median-based approach. This model incorporates the specific influence of each movie on user preferences, offering a more refined understanding of the factors contributing to the variability in ratings within the dataset.

```
##############################################################
## 2. Movie effect model on edx_train
##############################################################
# Calculate the average rating in the edx_train dataset
average_rating <- mean(edx_train$Rating)

# Estimate the movie effects (bi) by calculating the mean rating for each mov
ie
movie_effects <- edx_train %>%
  group_by(MovieID) %>%
```

```
  summarise(movie_effect = mean(Rating - average_rating))
head(movie_effects)

# Merge movie_effects with edx_train
edx_train <- edx_train %>%
  left_join(movie_effects, by = "MovieID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi
edx_train$predicted_rating <- average_rating + edx_train$movie_effect

# Calculate RMSE
rmse_movie_effect <- sqrt(mean((edx_train$Rating - edx_train$predicted_rating
) ^ 2))
print(rmse_movie_effect)
#ouput is [1] 0.9423541
```

2. Movie effect model on edx_temp

```
################################################################################
#######
## Movie effect model on edx_temp
# Calculate the average rating in the edx_temp dataset
average_rating_temp <- mean(edx_temp$Rating)

# Estimate the movie effects (bi) by calculating the mean rating for each mov
ie in edx_temp
movie_effects_temp <- edx_temp %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating_temp))
head(movie_effects_temp)

# Merge movie_effects_temp with edx_temp
edx_temp <- edx_temp %>%
  left_join(movie_effects_temp, by = "MovieID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi
edx_temp$predicted_rating <- average_rating_temp + edx_temp$movie_effect

# Calculate RMSE for the Movie Effect model on edx_temp
rmse_movie_effect_temp <- sqrt(mean((edx_temp$Rating - edx_temp$predicted_rat
ing) ^ 2))
print(rmse_movie_effect_temp)
# Output print(rmse_movie_effect_temp)[1] 0.9368914
```

3. Movie and User effect on edx_train

Movie and user effect models are constructed for the edx_train and edx_temp datasets. These models aim to capture inherent movie and user-specific biases in ratings. The predicted rating ($y(\mu, i)$) is calculated as the sum of the overall average rating (μ), the movie effect (b_i), and the user effect (b_μ), expressed by the formula, $y(\mu,i)=\mu+ b_i + b_\mu$. We compute the mean rating for each movie and user, relative to the overall average rating. These effects are then merged with their respective datasets, and the predicted ratings are generated. The Root Mean Square Error (RMSE) is subsequently calculated to quantify the model's prediction accuracy. The lower RMSE observed for the edx_temp dataset suggests that the movie and user effect models perform slightly better on this dataset compared to edx_train.

```
##################################################
# 3. Movie and User Effect Models on edx_train
##################################################

# Calculate the average rating in the edx_train dataset
average_rating_train <- mean(edx_train$Rating)

# Estimate movie effects (bi) by calculating the mean rating for each movie i
n edx_train
movie_effects_train <- edx_train %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating_train))

# Merge movie_effects_train with edx_train
edx_train <- edx_train %>%
  left_join(movie_effects_train, by = "MovieID")

# Estimate user effects (bu) by calculating the mean rating for each user in
edx_train
user_effects_train <- edx_train %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating_train))

# Merge user_effects_train with edx_train
edx_train <- edx_train %>%
  left_join(user_effects_train, by = "UserID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
edx_train$predicted_rating_user_movie_train <- average_rating_train + edx_tra
in$movie_effect + edx_train$user_effect

# Calculate RMSE for the Movie and User Effect models on edx_train
rmse_movie_user_effect_train <- sqrt(mean((edx_train$Rating - edx_train$predi
cted_rating_user_movie_train) ^ 2))
print(rmse_movie_user_effect_train)
# output print(rmse_movie_user_effect_train)[1] 0.8765064
```

3. Movie and User Effect Models on edx_temp

```
##################################################
# 3. Movie and User Effect Models on edx_temp
##################################################
# Calculate the average rating in the edx_temp dataset
```

```
average_rating_temp <- mean(edx_temp$Rating)

# Estimate movie effects (bi) by calculating the mean rating for each movie i
n edx_temp
movie_effects_temp <- edx_temp %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating_temp))

# Merge movie_effects_temp with edx_temp
edx_temp <- edx_temp %>%
  left_join(movie_effects_temp, by = "MovieID")

# Estimate user effects (bu) by calculating the mean rating for each user in
edx_temp
user_effects_temp <- edx_temp %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating_temp))

# Merge user_effects_temp with edx_temp
edx_temp <- edx_temp %>%
  left_join(user_effects_temp, by = "UserID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
edx_temp$predicted_rating_user_movie_temp <- average_rating_temp + edx_temp$m
ovie_effect + edx_temp$user_effect

# Calculate RMSE for the Movie and User Effect models on edx_temp
rmse_movie_user_effect_temp <- sqrt(mean((edx_temp$Rating - edx_temp$predicte
d_rating_user_movie_temp) ^ 2))
print(rmse_movie_user_effect_temp)
#output is print(rmse_movie_user_effect_temp)[1] 0.8490636
```

4. Movie and User effect with regularization on edx_train

Movie and user effect models with regularization are implemented for both the edx_train and edx_temp datasets. Regularization is introduced to control the complexity of the models and prevent overfitting, with the regularization parameter (λ) set to 0.1. For each dataset, movie effects (b_i) and user effects (bμ) are estimated using linear regression with regularization. The regularization term is incorporated using the weights parameter, ensuring a balance between fitting the data and avoiding excessive model complexity. The calculated movie and user effects with regularization are then merged with their respective datasets, and predicted ratings (y(μ, i)) are generated based on the formula, y(μ,i)=μ+ b_i + bμ. The Root Mean Square Error (RMSE) is computed to evaluate the performance of the models with regularization. The RMSE results indicate that regularization slightly improves the model performance, as seen in the lower RMSE for the edx_temp dataset compared to the regular movie and user effect models. This suggests that regularization helps control overfitting and enhances the models' generalization to new data.

```
##################################################
# 4. Movie and User Effect Models with Regularization on edx_train
##################################################

# Set the regularization parameter (lambda)
lambda <- 0.1

# Calculate the average rating in the edx_train dataset
average_rating_train <- mean(edx_train$Rating)

# Estimate movie effects (bi) with regularization
movie_effects_train <- edx_train %>%
  group_by(MovieID) %>%
  summarize(movie_effect = lm(Rating ~ 0, weights = 1 / (1 + lambda))$coef)

# Merge movie_effects_train with edx_train
edx_train <- edx_train %>%
  left_join(movie_effects_train, by = "MovieID")

# Estimate user effects (bu) with regularization
user_effects_train <- edx_train %>%
  group_by(UserID) %>%
  summarize(user_effect = lm(Rating ~ 0, weights = 1 / (1 + lambda))$coef)

# Merge user_effects_train with edx_train
edx_train <- edx_train %>%
  left_join(user_effects_train, by = "UserID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
edx_train$predicted_rating_user_movie_train_reg <- average_rating_train +
  edx_train$movie_effect + edx_train$user_effect

# Calculate RMSE for the Movie and User Effect models with regularization on
edx_train
rmse_movie_user_effect_train_reg <- sqrt(mean((edx_train$Rating -
                                                edx_train$predicted_rating_u
ser_movie_train_reg) ^ 2))
print(rmse_movie_user_effect_train_reg)
#output print(rmse_movie_user_effect_train_reg)
[1] 0.8765064
```

4. Movie and user effect with regularization on edx_temp

```
##################################################
# 4. Movie and User Effect Models with Regularization on edx_temp
##################################################

# Calculate the average rating in the edx_temp dataset
average_rating_temp <- mean(edx_temp$Rating)
```

```
# Estimate movie effects (bi) with regularization for edx_temp
movie_effects_temp <- edx_temp %>%
  group_by(MovieID) %>%
  summarize(movie_effect = lm(Rating ~ 0, weights = 1 / (1 + lambda))$coef)

# Merge movie_effects_temp with edx_temp
edx_temp <- edx_temp %>%
  left_join(movie_effects_temp, by = "MovieID")

# Estimate user effects (bu) with regularization for edx_temp
user_effects_temp <- edx_temp %>%
  group_by(UserID) %>%
  summarize(user_effect = lm(Rating ~ 0, weights = 1 / (1 + lambda))$coef)

# Merge user_effects_temp with edx_temp
edx_temp <- edx_temp %>%
  left_join(user_effects_temp, by = "UserID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
edx_temp$predicted_rating_user_movie_temp_reg <- average_rating_temp +
  edx_temp$movie_effect + edx_temp$user_effect

# Calculate RMSE for the Movie and User Effect models with regularization on
edx_temp
rmse_movie_user_effect_temp_reg <- sqrt(mean((edx_temp$Rating -
                                              edx_temp$predicted_rating_use
r_movie_temp_reg) ^ 2))
print(rmse_movie_user_effect_temp_reg)
#output is print(rmse_movie_user_effect_temp_reg)
[1] 0.8490636
```

5. Movie, User and Genre effect on edx_train

A comprehensive movie, user, and genre effect model is developed for both the edx_train and edx_temp datasets. The model incorporates the average rating (μ), movie effects (b*i*), user effects (bμ), and genre effects (bg). The predicted rating (y(μ, i)) is calculated using the formula y(μ,i)=μ+ b*i* + bμ+ bg. For each dataset, the mean rating is computed, and effects are estimated by grouping the data based on MovieID, UserID, and Genres, respectively. These effects are then merged with their respective datasets, creating a comprehensive model that considers movie, user, and genre-specific biases. The predicted ratings are calculated by summing the average rating and the respective effects. The Root Mean Square Error (RMSE) is then computed to assess the accuracy of the model predictions. The lower RMSE for the edx_temp dataset compared to edx_train suggests that including genre effects enhances the model's performance on the new dataset, showcasing the importance of considering genre-specific biases in predicting user ratings.

```
###############################################################
# 5.Movie, user and genre effect on edx_train
###################################################################
```

```
# # Calculate the average rating in the edx_train dataset
average_rating <- mean(edx_train$Rating)

# Estimate the movie effects (bi) by calculating the mean rating for each mov
ie
movie_effects <- edx_train %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating))

# Estimate the user effects (bu) by calculating the mean rating for each user
user_effects <- edx_train %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating))

# Estimate the genre effects (bg) by calculating the mean rating for each gen
re
genre_effects <- edx_train %>%
  group_by(Genres) %>%
  summarise(genre_effect = mean(Rating - average_rating))

# Merge effects with edx_train
edx_train <- edx_train %>%
  left_join(movie_effects, by = "MovieID") %>%
  left_join(user_effects, by = "UserID") %>%
  left_join(genre_effects, by = "Genres")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
+ bg
edx_train$predicted_rating <- average_rating + edx_train$movie_effect + edx_t
rain$user_effect + edx_train$genre_effect

# Calculate RMSE
rmse_movie_user_genre_effect <- sqrt(mean((edx_train$Rating - edx_train$predi
cted_rating) ^ 2))
print(rmse_movie_user_genre_effect)
# #output is [1] 0.93772
```

5.Movie, User and Genre effect on edx_temp

```
#####################################################
# 5. Movie, user and Genre effect on edx_temp
#####################################################
# Estimate the movie effects (bi) for edx_temp by calculating the mean rating
for each movie
movie_effects_temp <- edx_temp %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating))

# Estimate the user effects (bu) for edx_temp by calculating the mean rating
for each user
```

```
user_effects_temp <- edx_temp %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating))

# Estimate the genre effects (bg) for edx_temp by calculating the mean rating
for each genre
genre_effects_temp <- edx_temp %>%
  group_by(Genres) %>%
  summarise(genre_effect = mean(Rating - average_rating))

# Merge effects with edx_temp
edx_temp <- edx_temp %>%
  left_join(movie_effects_temp, by = "MovieID") %>%
  left_join(user_effects_temp, by = "UserID") %>%
  left_join(genre_effects_temp, by = "Genres")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
+ bg
edx_temp$predicted_rating <- average_rating + edx_temp$movie_effect + edx_tem
p$user_effect + edx_temp$genre_effect

# Calculate RMSE for edx_temp
rmse_movie_user_genre_effect_temp <- sqrt(mean((edx_temp$Rating - edx_temp$pr
edicted_rating) ^ 2))
print(rmse_movie_user_genre_effect_temp)
#output is print(rmse_movie_user_genre_effect_temp)[1] 0.9184097
```

6. Movie, user and Genre effect with regularization on edx_train

A movie, user, and genre effect model with regularization is implemented for both the edx_train and edx_temp datasets. Regularization, controlled by the parameter ((λ), is introduced to prevent overfitting and improve model generalization. For each dataset, movie, user, and genre effects are estimated by calculating the mean rating for each MovieID, UserID, and Genres, respectively. These effects are then merged with their respective datasets. Following this, regularization is applied to the movie, user, and genre effects to control their magnitudes.The predicted ratings (($y(\mu, i)$) are computed using the formula $y(\mu,i)=\mu + b_i + b_\mu + b_g$, where μ is the average rating, and bi, bμ, and bg, represent the regularized movie, user, and genre effects. The Root Mean Square Error (RMSE) is then calculated to assess the model's predictive accuracy. The RMSE results indicate the performance of the model on both datasets. The values suggest that, despite regularization, the model's predictive accuracy is slightly diminished, potentially due to the increased complexity introduced by considering movie, user, and genre effects simultaneously.

```
##################################################
### 6. Movie, user and Genre effect with regularization on edx_train
######################################################
# Regularization parameter
lambda <- 0.1
```

```
# Calculate the average rating in the edx_train dataset
average_rating <- mean(edx_train$Rating)

# Estimate the movie effects (bi) by calculating the mean rating for each mov
ie
movie_effects <- edx_train %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating))

# Estimate the user effects (bu) by calculating the mean rating for each user
user_effects <- edx_train %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating))

# Estimate the genre effects (bg) by calculating the mean rating for each gen
re
genre_effects <- edx_train %>%
  group_by(Genres) %>%
  summarise(genre_effect = mean(Rating - average_rating))

# Merge effects with edx_train
edx_train <- edx_train %>%
  left_join(movie_effects, by = "MovieID") %>%
  left_join(user_effects, by = "UserID") %>%
  left_join(genre_effects, by = "Genres")

# Regularization for movie effects
edx_train$movie_effect <- with(edx_train, movie_effect / (1 + lambda * nrow(m
ovie_effects)))

# Regularization for user effects
edx_train$user_effect <- with(edx_train, user_effect / (1 + lambda * nrow(use
r_effects)))

# Regularization for genre effects
edx_train$genre_effect <- with(edx_train, genre_effect / (1 + lambda * nrow(g
enre_effects)))

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
+ bg
edx_train$predicted_rating <- average_rating + edx_train$movie_effect + edx_t
rain$user_effect + edx_train$genre_effect

# Calculate RMSE
rmse_movie_user_genre_effect <- sqrt(mean((edx_train$Rating - edx_train$predi
cted_rating) ^ 2))
print(rmse_movie_user_genre_effect)
# output is print(rmse_movie_user_genre_effect)[1] 1.059107
```

6. Movie, User, and Genre Effect with Regularization on edx_temp

```
##################################################
### 6. Movie, User, and Genre Effect with Regularization on edx_temp
##################################################

# Regularization parameter
lambda <- 0.1

# Calculate the average rating in the edx_temp dataset
average_rating_temp <- mean(edx_temp$Rating)

# Estimate the movie effects (bi) by calculating the mean rating for each mov
ie
movie_effects_temp <- edx_temp %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating_temp))

# Estimate the user effects (bu) by calculating the mean rating for each user
user_effects_temp <- edx_temp %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating_temp))

# Estimate the genre effects (bg) by calculating the mean rating for each gen
re
genre_effects_temp <- edx_temp %>%
  group_by(Genres) %>%
  summarise(genre_effect = mean(Rating - average_rating_temp))

# Merge effects with edx_temp
edx_temp <- edx_temp %>%
  left_join(movie_effects_temp, by = "MovieID") %>%
  left_join(user_effects_temp, by = "UserID") %>%
  left_join(genre_effects_temp, by = "Genres")

# Regularization for movie effects
edx_temp$movie_effect <- with(edx_temp, movie_effect / (1 + lambda * nrow(mov
ie_effects_temp)))

# Regularization for user effects
edx_temp$user_effect <- with(edx_temp, user_effect / (1 + lambda * nrow(user_
effects_temp)))

# Regularization for genre effects
edx_temp$genre_effect <- with(edx_temp, genre_effect / (1 + lambda * nrow(gen
re_effects_temp)))

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
+ bg
edx_temp$predicted_rating <- average_rating_temp + edx_temp$movie_effect + ed
```

```
x_temp$user_effect + edx_temp$genre_effect

# Calculate RMSE
rmse_movie_user_genre_effect_temp <- sqrt(mean((edx_temp$Rating - edx_temp$pr
edicted_rating) ^ 2))
print(rmse_movie_user_genre_effect_temp)
#output is print(rmse_movie_user_genre_effect_temp)[1] 1.058703
```

Results for all above models

Overview

The table below summarizes the Root Mean Squared Error (RMSE) values for different models applied to the dataset.

Model	RMSE
Naive model	1.060362
Movie effect model	0.9423541
Movie and User effect model	0.8765064
Movie and User effect with regularization	0.8765064
Movie, User and Genre effect	0.93772
Movie, user and Genre effect with regularization	1.059107

Observations

The lowest RMSE is achieved by the "Movie and User effect model. The RMSE value obtained with the Movie and User effect model with regularization is also the same. Surprisingly, regularization did not lead to a reduction in RMSE in this case.

Decision

Given the identical RMSE values and the absence of improvement with regularization, the "Movie and User effect model" (without regularization) is chosen as the final model for testing on the validation dataset (final_holdout_test).

Validation of the final_holdout_test data on Movie, User effect model

Movie and User Effect Model on final_holdout_test

```
#################################################
# Movie and User Effect Models on final_holdout_test
#################################################

# Calculate the average rating in the final_holdout_test dataset
average_rating_final_holdout <- mean(final_holdout_test$Rating)

# Estimate movie effects (bi) by calculating the mean rating for each movie i
```

```
n final_holdout_test
movie_effects_final_holdout <- final_holdout_test %>%
  group_by(MovieID) %>%
  summarise(movie_effect = mean(Rating - average_rating_final_holdout))

# Merge movie_effects_final_holdout with final_holdout_test
final_holdout_test <- final_holdout_test %>%
  left_join(movie_effects_final_holdout, by = "MovieID")

# Estimate user effects (bu) by calculating the mean rating for each user in
final_holdout_test
user_effects_final_holdout <- final_holdout_test %>%
  group_by(UserID) %>%
  summarise(user_effect = mean(Rating - average_rating_final_holdout))

# Merge user_effects_final_holdout with final_holdout_test
final_holdout_test <- final_holdout_test %>%
  left_join(user_effects_final_holdout, by = "UserID")

# Calculate predicted ratings (yu,i) based on the formula: yu,i = μ + bi + bu
final_holdout_test$predicted_rating_user_movie_final_holdout <- average_ratin
g_final_holdout +
  final_holdout_test$movie_effect + final_holdout_test$user_effect

# Calculate RMSE for the Movie and User Effect models on final_holdout_test
rmse_movie_user_effect_final_holdout <- sqrt(mean((final_holdout_test$Rating
-
                                                     final_holdout_test$predi
cted_rating_user_movie_final_holdout) ^ 2))
print(rmse_movie_user_effect_final_holdout)
Output is print(rmse_movie_user_effect_final_holdout) [1] 0.8534251
```

Result of Movie and User effect on final_holdout_test

Model	RMSE
Movie and User effect	0.8534251

Conclusion

The exploration of different collaborative filtering models on the MovieLens dataset has provided valuable insights into the prediction of user ratings for movies. Below is a summary of the key findings:

Model Performance: The “Movie and User effect model” demonstrated the lowest RMSE, indicating its effectiveness in predicting user ratings. Surprisingly, introducing regularization in the “Movie and User effect model with regularization” did not lead to a further reduction in RMSE, yielding identical results. Validation Set Performance:

The selected model, the “Movie and User effect model,” exhibited strong generalization to new, unseen data, achieving an impressive RMSE of 0.8534251 on the validation set (final_holdout_test).

Decision for Deployment: Considering the comparable performance and simplicity of the “Movie and User effect model” without regularization, this model is recommended for deployment. Its effectiveness on the validation set suggests robust predictive capabilities.

Areas for Further Investigation: Despite the overall success, it’s essential to explore the factors contributing to the lack of improvement with regularization. Further investigation into the dataset characteristics and the regularization approach may provide insights.

In conclusion, the chosen “Movie and User effect model” stands out as a reliable choice for predicting user ratings in collaborative filtering scenarios. The findings lay the foundation for continued refinement and exploration in the domain of recommendation systems.The methodologies applied in this project draw inspiration from the coursework of the Harvard Data Science Certificate Program. Personally, I found great satisfaction in the process of constructing diverse models and comprehending the impact of various variables on predicting Root Mean Square Error (RMSE) values. While I acknowledge that there might be additional techniques to further explore the dataset and achieve lower RMSE values, this marks my inaugural engagement with such analyses, and I have invested my utmost effort into this endeavor. I am keenly aware that alternative modeling methods could potentially yield superior outcomes. I express gratitude for the opportunity to delve into this dataset, and I look forward to advancing my skills in future explorations.

Reference

- Irizarry, R.A. Introduction to Data Science. Retrieved from
- F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872
- The discussion section for this course with recommendations from the Teaching Assistant and harvardEdx Team. ttps://discussions.edx.org/course-v1:HarvardX+PH125.9x+2T2023/posts/643f23ce70435a04a476d1cc
- The R Project for Statistical Computing. https://www.r-project.org/