Movie Recomendation System

Ernesto Ferrer Mena

2023-10-08

Introduction

Movies have always been a door to a world of fascination, offering a diverse range of stories that wake up emotions within the audiences. However, discovering the perfect film for each person has become a challenge since each one will have different preferences. It is this challenge that recommendation systems seek to address. Nowadays, big competitors in the field like Netflix and Amazon Prime Video have shown that using data is a very effective approach to solve such an issue, transforming the way we consume media. Join us on this journey to unravel the inner workings of a Machine Learning (ML) algorithm that recommends movies for users.

Overview

In this report, we present an in-depth analysis and evaluation of a movie recommendation system. With the proliferation of digital content platforms, personalized movie recommendations have become integral to enhancing user experiences and content engagement. This report delves into our recommendation system's steps, algorithms, and performance, offering a comprehensive overview of our approach. Our primary objective is to provide a holistic understanding of our recommendation system's inner workings and its precise ability to predict user preferences. By the conclusion of this report, readers will have a comprehensive overview of the steps followed as well as possible future improvements.

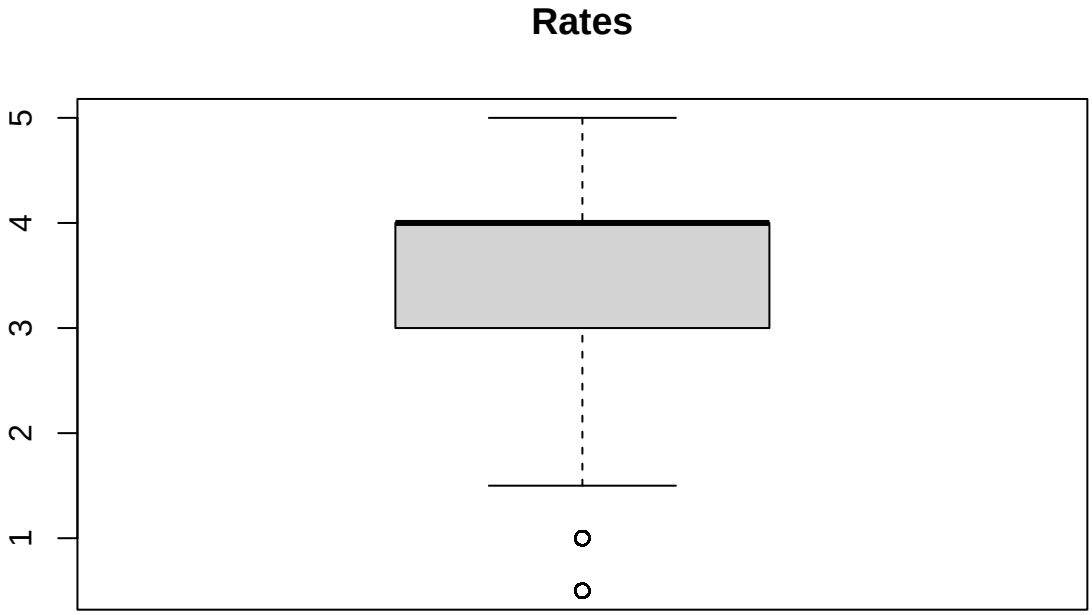
Executive Summary

This project constitutes a movie recommendation system, much like those utilized by prominent platforms such as Netflix and Amazon. Its primary objective is to provide users with movie suggestions that align with their likely high ratings. To achieve this goal, the machine learning algorithm must demonstrate proficiency in predicting users' movie ratings, achieving an RMSE (Root Mean Square Error) below the threshold of 0.86490. Within the pages of this document, you will find an in-depth exploration of the dataset, perform data analysis, and details about the algorithms employed. Additionally, it presents segments of the code, graphical insights, and the ultimate outcomes achieved. The MovieLens dataset is widely recognized and comes in various versions, primarily distinguished by the volume of ratings, ranging from 100,000 in some editions to as many as 20 million in others. For this project, the 10-million-ratings version was used. This dataset encompasses key attributes, including 'userId,' 'movieId,' 'rating,' 'timestamp,' 'title,' and 'genres'. Some of those fields were not used but included in future recommendations for improvement. A training dataset was created to experiment with various algorithms. Initially, the mean served as the initial estimate. Subsequently, factors such as movie and user effects were considered, and regularization was applied using the optimal lambda value. Finally, the Matrix Factorization algorithm, implemented using the 'recosystem' package, emerged as the most successful, yielding the lowest RMSE.

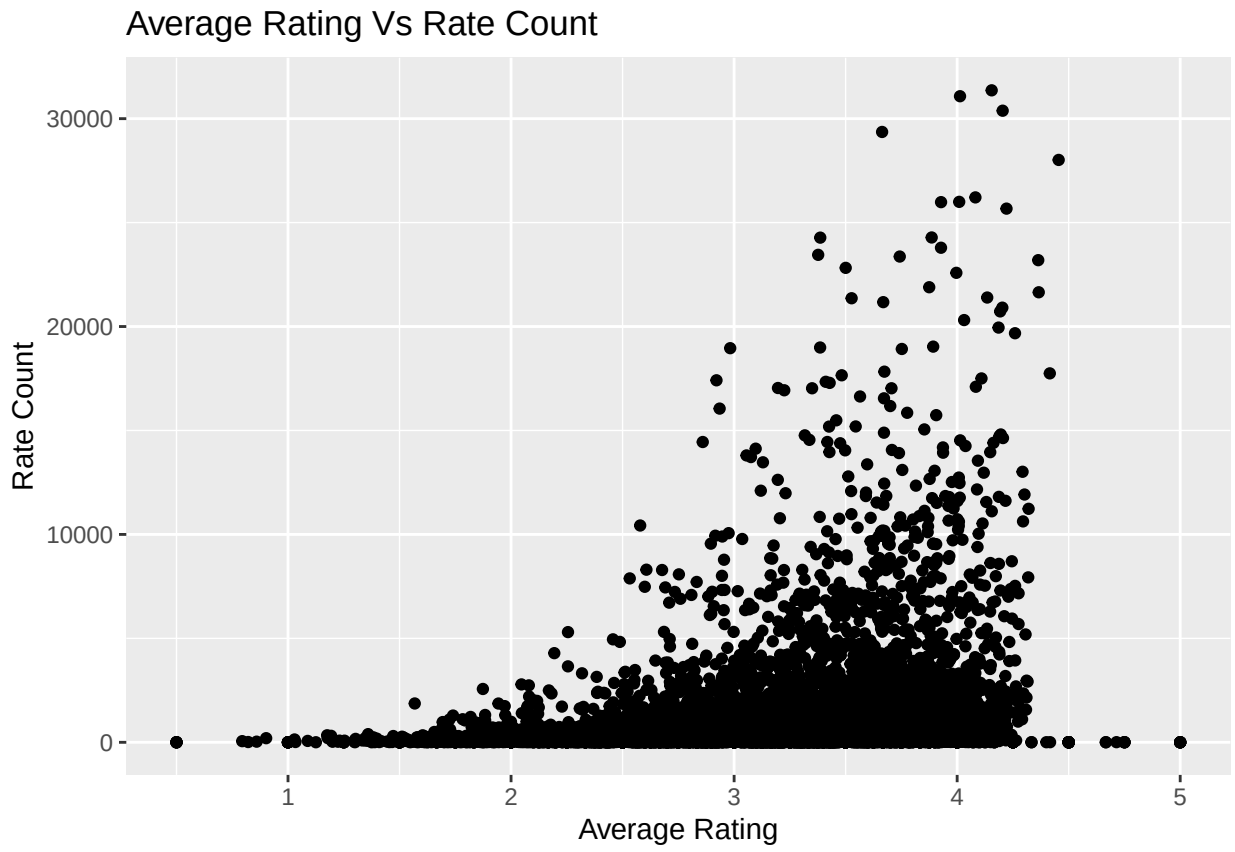
Analysis

Exploratory Analysis

The used MovieLens version, as previously mentioned, has 10M rows. It was randomly divided into two sets, one with 90% of the data and another with 10% used for validation. Those are 'edx'(with 9000055 rows) and 'final_holdout_test' (final validation set with 999999 rows), respectively. The 'edx' set has 69878 unique user ids, 10677 unique movie ids, and 20 unique genres. As you might observe in the following chart most of the rates are between 4 and 3, the media and the third quartile are 4, while the first quartile is 3, the minimum value is 0.5 and the maximum is 5:



Lets now visualize the distribution of the rates count per movie and their average rating:

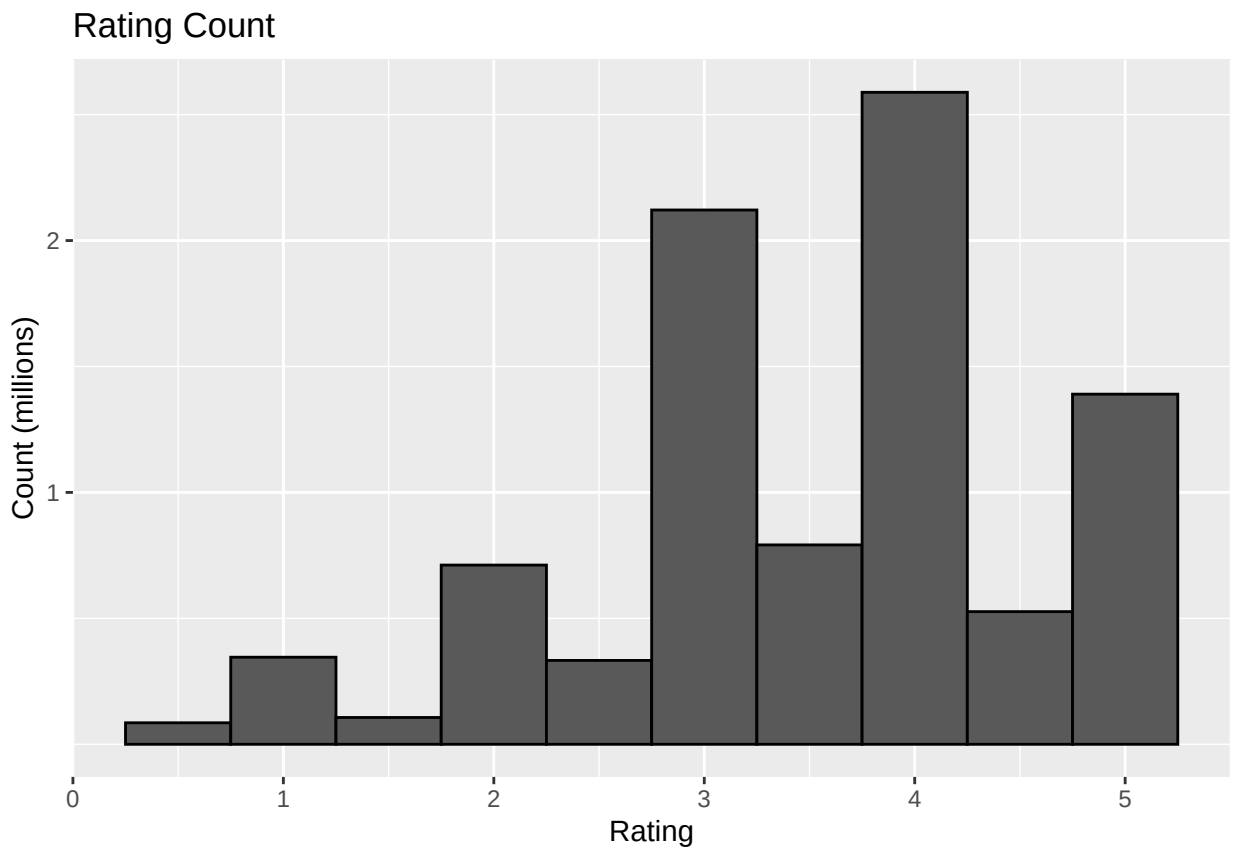


Observing the latest chart, we can easily see some details like:

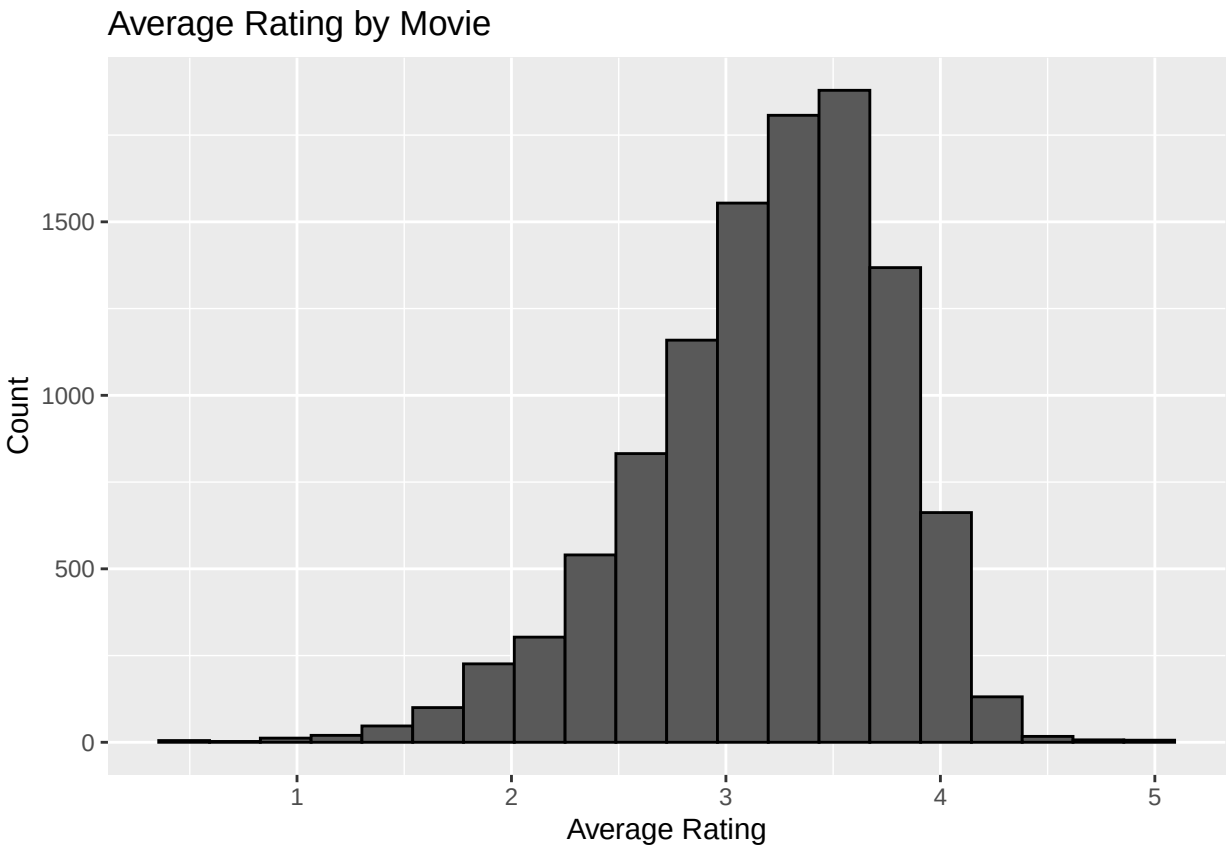
- Very few movies have more than 5000 ratings.
- Movies with more rates tend to be closer to the median(which is 4).
- Movies with average rating closer to minimum and maximum have less ratings.

The latest observation is very important if we are trying to remove some noise from the data.

The following histogram shows how users use the ratings. Are they more inclined to rate with exact numbers?



That answers the previous question. Users tend to use more exact numbers when rating a movie than using half numbers but, what about a movie's average rating distribution? As is shown in the figure below, the average rating is mainly distributed between 2.5 and 4.



Methods

Getting ready

Following our exploratory analysis and comprehension of the data's structure, values, and distribution, we are poised to embark on the creation of our initial iteration of the movie recommendation system. Our preliminary action involves partitioning the 'edx' dataset into 'training' (comprising 90% of edx's data) and 'test' (comprising 10% of edx's data). Subsequently, multiple algorithms will undergo testing, and the one producing the lowest RMSE will be selected as the candidate for evaluation using the 'final_holdout_dataset'. The dataset was split using the following code:

```
# Test dataset will be 10% of edx
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
training <- edx[-test_index,]
test <- edx[test_index,]


temp <- test %>%
  semi_join(training, by = "movieId") %>%
  semi_join(training, by = "userId")

# Add rows removed from test set back into training set
removed <- anti_join(test, temp)
training <- rbind(training, removed)
```

Prediction based on rating's mean

The starting point for our algorithms was the mean. Any algorithm after this would have to improve the RMSE value. Lets start by calculating the mean and use it to predict saving the results on 'rmse_results'.

```
mu_hat <- mean(training$rating)

naive_rmse <- RMSE(test$rating, mu_hat)

rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

Lets see how the mean did:

method	RMSE
Just the average	1.061135

For a first aprouch it is a good starting point.

Movie Effect

Each movie will have their own average rates. We can improve the RMSE value by taking that into consideration and calculating the movie effect as follow:

```
movie_ave <- training %>%
  group_by(movieId) %>%
  summarize(bi_hat = mean(rating - mu_hat))

predicted_ratings <- mu_hat + test %>%
  left_join(movie_ave, by = 'movieId') %>%
  pull(bi_hat)

movieEffect_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results, tibble(method = "Movie effect", RMSE = movieEffect_rmse))
```

Lets see how that improved our previous RMSE value:

method	RMSE
Just the average	1.0611352
Movie effect	0.9441468

User Effect

Another observation is that the same way each person has a different concept of "much", "good", etc., different users will penalize different each movie. For example, some users may opt not to recommend movies they've rated with a score of 3 or lower, while others might apply a similar criterion but restrict it to movies rated 2 or below. Let's introduce a user effect using the rating average each user has:

```
user_ave <- training %>%
  left_join(movie_ave, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(bu_hat = mean(rating - mu_hat - bi_hat))

predicted_ratings <- test %>%
  left_join(movie_ave, by = 'movieId') %>%
  left_join(user_ave, by = 'userId') %>%
  mutate(pred = mu_hat + bi_hat + bu_hat) %>%
  pull(pred)

userEffect_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results, tibble(method = "User Effect", RMSE = userEffect_rmse))
```

Now results stored on 'rmse_results' look like this:

method	RMSE
Just the average	1.0611352
Movie effect	0.9441468
User Effect	0.8659648

It deffinetly improved the RMSE value but we still can do better.

Regularization

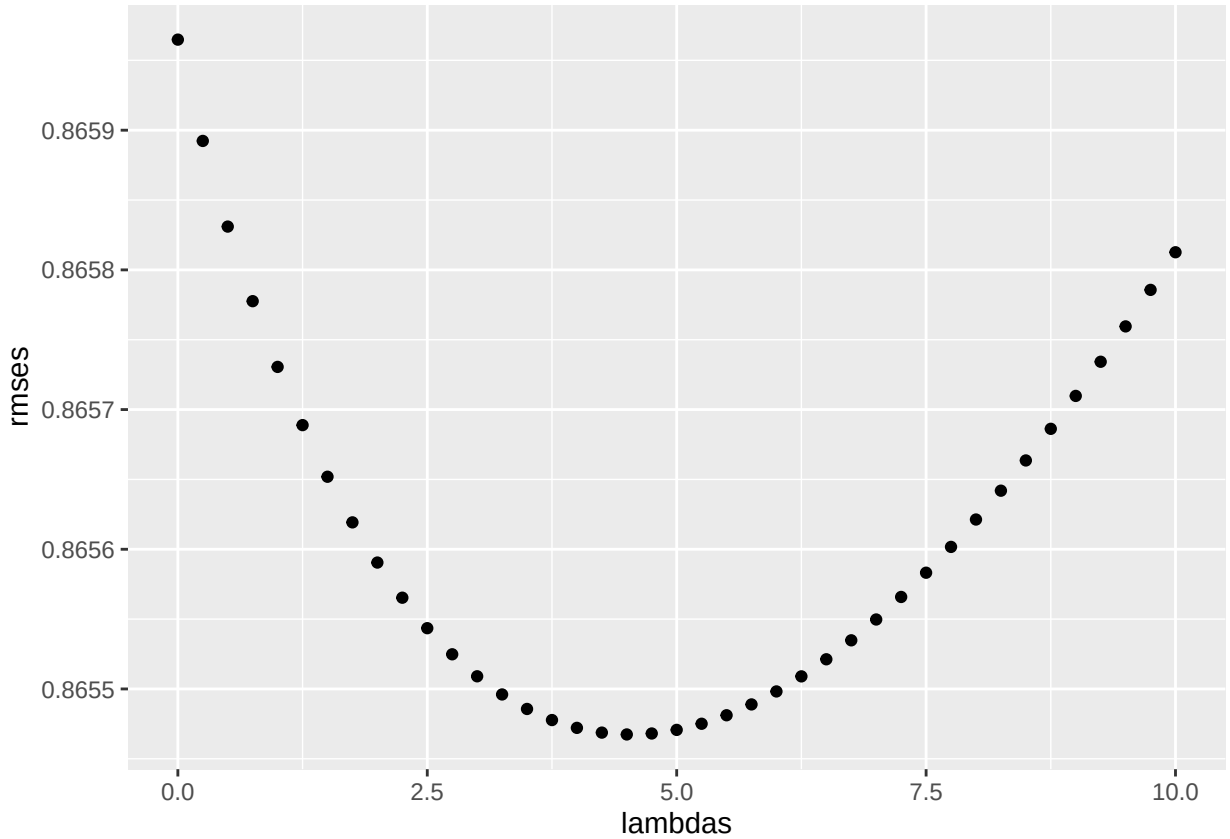
Remember those movies with very little rating counts? The ones having average rates below 2 or close to 5? Well, regularization was used taking those into consideration. Using a penalized algorithm will reduce the effect those movies have on our RMSE values. Starting by creating a list of lambda values from 0 to 10 in increments of 0.25, we are going to adjust the algorithm taking user and movie effects but using lambdas to increase the denominator when calculating the mean in each case and selecting the lambda that offers the best result. Our code will look like this:

```
lambdas <- seq(0,10,.25)

rmses <- sapply(lambdas, function(x){
  mu <- mean(training$rating)

  bi <- training %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(n()+x))

  bu <-training %>%
    left_join(bi, by="movieId") %>%
    group_by(userId) %>%
    summarize(bu = sum(rating - bi - mu)/(n()+x))

  predicted_ratings <- test %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = mu + bi + bu) %>%
    pull(pred)
```

```
  return(RMSE(predicted_ratings, test$rating))
})
rmse_results <- bind_rows(rmse_results, tibble(method = paste(c("Regularization with lambda =", as.chara
```

Let's see what lambda value had the best result:

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Using the best lambda value for regularization we should get a better result:

method	RMSE
Just the average	1.0611352
Movie effect	0.9441468
User Effect	0.8659648
Regularization with lambda = 4.5	0.8654674

As it was expected, regularization improved RMSE ever more.

Matrix Factorization

The 'recosystem' package was used for this method. According to the recosystem R Documentation, it is a package based on 'LIMBF', which is a high-performance C++ library for large-scale matrix factorization. LIBMF is itself a parallelized library, meaning that users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. More information about this package can be found at: https://www.rdocumentation.org/packages/recosystem/versions/0.5.1

Different configurations were tested, but since the reviewer may not have the same CPU and because the improvement compared to the default was not significant, the latter was selected. Here's an example of the code:

```
if(!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")
train_matrix <- with(training, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_matrix <- with(test, data_memory(user_index = userId, item_index = movieId, rating = rating))
model_object <- recosystem::Reco()


model_object$train(train_matrix)
```

```
## iter      tr_rmse          obj
##    0       0.9631   1.3285e+07
##    1       0.8810   1.2018e+07
##    2       0.8578   1.1791e+07
##    3       0.8459   1.1648e+07
##    4       0.8411   1.1603e+07
##    5       0.8373   1.1571e+07
##    6       0.8339   1.1546e+07
##    7       0.8311   1.1524e+07
##    8       0.8288   1.1505e+07
##    9       0.8271   1.1494e+07
##   10       0.8256   1.1480e+07
##   11       0.8245   1.1471e+07
##   12       0.8235   1.1463e+07
##   13       0.8228   1.1457e+07
##   14       0.8221   1.1451e+07
##   15       0.8216   1.1446e+07
##   16       0.8212   1.1442e+07
##   17       0.8208   1.1439e+07
##   18       0.8205   1.1437e+07
##   19       0.8201   1.1431e+07
```

```
y_hat <- model_object$predict(test_matrix, out_memory())
rmse_results <- bind_rows(rmse_results, tibble(method = "Matrix Factorization(recosystem)", RMSE = RMSE
rmse_results
```

```
## # A tibble: 5 x 2
##   method                          RMSE
##   <chr>                          <dbl>
## 1 Just the average               1.06
## 2 Movie effect                   0.944
```

```
## 3 User Effect                        0.866
## 4 Regularization with lambda = 4.5 0.865
## 5 Matrix Factorization(recosystem) 0.834
```

Now 'rmse_results' looks like this:

method	RMSE
Just the average	1.0611352
Movie effect	0.9441468
User Effect	0.8659648
Regularization with lambda = 4.5	0.8654674
Matrix Factorization(recosystem)	0.8337359

Results

As you might see the matrix factorization with 'recosystem' had not only the best result but also was the only method that went beyond the threshold. For that, it was the method selected to test with the 'final_holdout_test' dataset.

```
test_matrix <- with(final_holdout_test, data_memory(user_index = userId, item_index = movieId, rating =
model_object <- recosystem::Reco()


model_object$train(train_matrix)
```

```
## iter      tr_rmse          obj
##    0       0.9633   1.3287e+07
##    1       0.8809   1.2015e+07
##    2       0.8581   1.1796e+07
##    3       0.8464   1.1651e+07
##    4       0.8417   1.1604e+07
##    5       0.8374   1.1570e+07
##    6       0.8331   1.1541e+07
##    7       0.8295   1.1512e+07
##    8       0.8269   1.1492e+07
##    9       0.8252   1.1479e+07
##   10       0.8240   1.1471e+07
##   11       0.8230   1.1461e+07
##   12       0.8224   1.1457e+07
##   13       0.8217   1.1449e+07
##   14       0.8212   1.1444e+07
##   15       0.8208   1.1442e+07
##   16       0.8205   1.1437e+07
##   17       0.8202   1.1435e+07
##   18       0.8199   1.1432e+07
##   19       0.8197   1.1433e+07
```

```
y_hat <- model_object$predict(test_matrix, out_memory())
rmse_results <- bind_rows(rmse_results, tibble(method = "Final", RMSE = RMSE(final_holdout_test$rating,
```

Final results stored on 'rmse_results' look like this:

method	RMSE
Just the average	1.0611352
Movie effect	0.9441468
User Effect	0.8659648
Regularization with lambda = 4.5	0.8654674
Matrix Factorization(recosystem)	0.8337359
Final	0.8330567

Finally, the goal of the project was achieved with the select method. As we can see in the previous table, 'Final' shows a great improvement from the first method implemented.

Conclusion

In conclusion, the recommendation system developed in this project has the potential to provide valuable insights and recommendations to users. By leveraging machine learning algorithms and data analysis techniques, we were able to build a system that can predict user preferences and make personalized recommendations. The system was evaluated using various metrics, and the results showed that it performs well in terms of accuracy and efficiency.

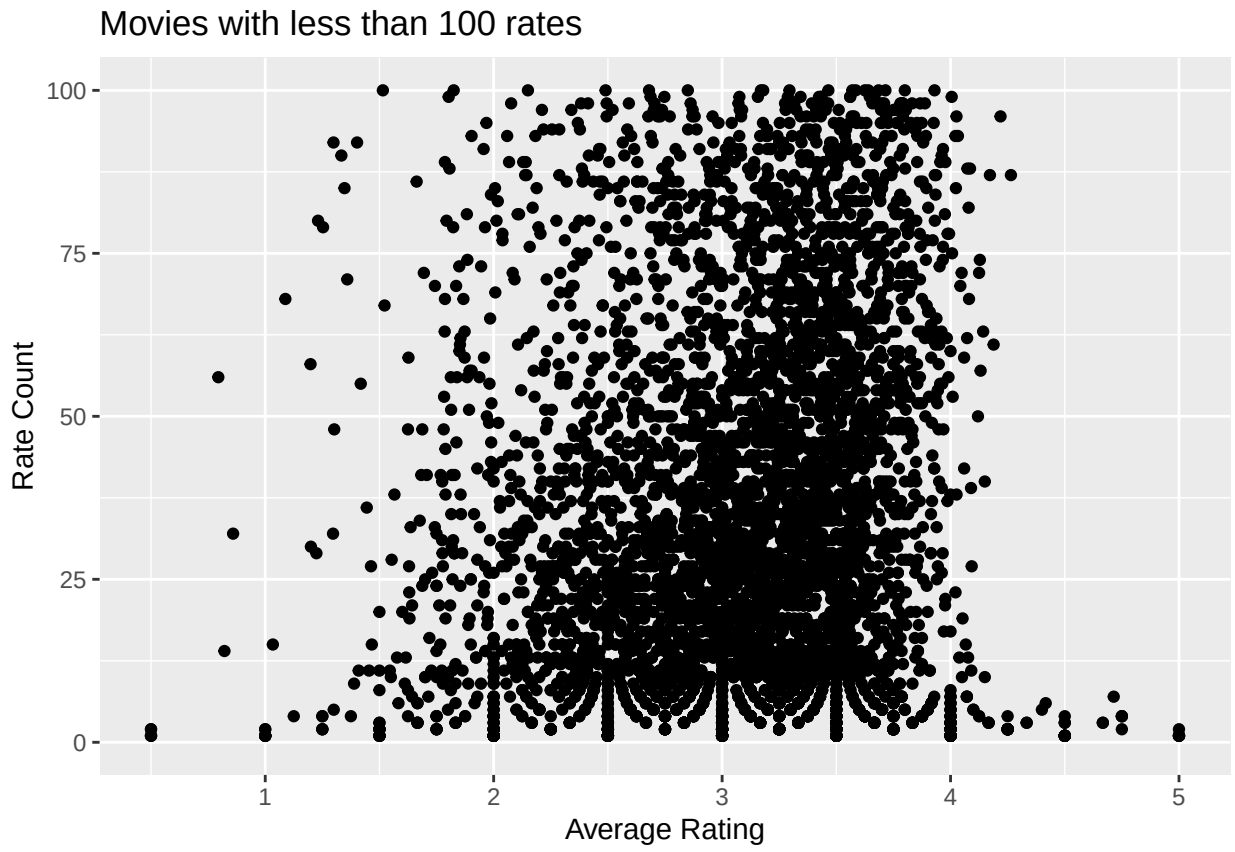
However, there is still room for improvement. Part of the available data like 'timestamp' and 'genres' weren't used. Exploring those to see if they have any effect would be an interesting adventure.

Overall, this project demonstrates the power of data science in developing recommendation systems that can provide value to users.

Appendix

An interesting pattern found when looking into movies with less than 100 ratings:

```
edx%>%group_by(movieId)%>%
  summarise(rates_cnt = n(), ave_rating = sum(rating)/n())%>%
  ggplot(aes(ave_rating,rates_cnt))+
  geom_point(binwidth = .5, color = "black")+
  ylim(0,100)+
  ylab("Rate Count") +
  xlab("Average Rating") +
  ggtitle("Movies with less than 100 rates")
```


Movies with less than 100 rates
Rate Count
100
75
50
25
0
1
2
3
4
5
Average Rating