Project Submission: Movie Lens

HarvardX PH125.9x Data Science: Capstone

Todd Blackwell

04/04/2024

Table of Contents

Project Overview

For this project, we will be creating a movie recommendation system from the MovieLens dataset using all the tools that have been taught throughout the Harvard Data Science courses. All students are instructed to use the 10M version of the MovieLens dataset to make the computation a little easier.

We will download the MovieLens data and run code provided by the course to generate the datasets.

We will train machine learning algorithms using the inputs in one subset to predict movie ratings in the validation set.

This project will be assessed by peer grading.

Honor Code

My participation in this course is governed by the terms of the edX Honor Code. I understand that to receive full marks on this project, I may not simply copy code from other courses in the course series and be done with this analysis. My work on this project needs to build on code that is already provided. Instances where I have copied code, test or ideas from other projects are given proper citation.

Executive Summary

In this machine learning project, we applied various methods to predict moving ratings on the MovieLens edx datset and evaluated their performance using Root Mean Squared Error (RMSE) as the metric. Here is a summary of the methods applied along with their resulting RMSE scores:

1. Just the Average Model: This simple baseline model, which predicts all ratings as the average rating, yielded an RMSE of 1.06045, serving as our initial reference point.

2. Movie Effect Model: Incorporating movie-specific effects, this model achieved an improved RMSE of 0.94393, indicating that considering individual movie characteristics enhances prediction accuracy.

3. User Effect Model: By accounting for user-specific effects, this model further reduced the RMSE to 0.86625, demonstrating the importance of user preferences in rating predictions.

4. Genre Effect Model: Leveraging genre-specific effects led to a comparable RMSE of 0.86591, suggesting that genre information contributes to predicting movie ratings.

5. Regularized Movie Effect Model: Introducing regularization to the movie effect model slightly improved performance, with an RMSE of 0.94386, indicating limited impact.

6. Regularized Movie + User Effect Model: Combining regularization with user-specific effects resulted in an RMSE of 0.86552, maintaining the same level of prediction accuracy as the non-regularized counterpart.

7. Regularized Movie + User + Genre Effect Model: Extending regularization to incorporate genre-specific effects yielded the lowest regularized RMSE of 0.86523, yet not a significant improvement over the non-regularized model.

8. Matrix Factorization: Employing matrix factorization techniques resulted in an initial RMSE of 0.97250 at the start of training, progressively improving to 0.72540 by the end of training, with a mean RMSE of 0.77768, which was the best of all models tested.

9. Final Holdout Test: Finally, the Matrix Factorization model underwent evaluation on a holdout test set, resulting in an RMSE of 0.78306, thereby confirming its predictive performance.

Summary of Results

method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179
Regularized Movie Effect Model	0.9438669
Regularized Movie + User Effect Model	0.8655286
Regularized Movie + User + Genre Effect Model	0.8652370
Matrix Factorization - Start	0.9725000
Matrix Factorization - End	0.7254000
Matrix Factorization - Mean	0.7776800
Final Holdout Test	0.7830654

In summary, our best working algorithm was the Matrix Factorization method, which delivered a mean training RSMSE of 0.77768 and a predicted RMSE also of 0.78306. This meets the course requirements of a predicted RMSE <0.86490.

Project Setup

The following code is used to set up the project, creating edx and final holdout datasets, and creating train and test sets.

Create edx and final holdout sets

We will use the following code to generate the datasets. We will develop algorithms using the edx set. For a final test of the final algorithm, we will predict movie ratings in the final_holdout_test set as if they were unknown. RMSE will be used to evaluate how close the predictions are to the true values in the final_holdout_test set.

Important: The final_holdout_test data should NOT be used for training, developing, or selecting the algorithm and it should ONLY be used for evaluating the RMSE of the final algorithm. The final_holdout_test set should only be used at the end of the project with the final model. It may not be used to test the RMSE of multiple models during model development. We will split the edx data into separate training and test sets and/or use cross-validation to design and test the algorithm.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes
#Load necessary packages
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us
.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-proje
ct.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-proje
ct.org")
if(!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-proje
ct.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-p
roject.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-proje
ct.org")
if(!require(magrittr)) install.packages("magrittr", repos = "http://cran.us.r
-project.org")
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.
us.r-project.org")

library(tidyverse)
library(caret)
library(dplyr)
library(tidyr)
library(ggplot2)
library(knitr)
library(magrittr)
```

```r
library(recosystem)


# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip",
dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), sim
plify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simpl
ify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, l
ist = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

```r
# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

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

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

Getting to know the dataset

We will run the following to get a better understanding of our data.

```r
#############################################################
# Getting to know the data
#############################################################
#//Questions adapted from https://learning.edx.org/course/course-v1:HarvardX+
PH125.9x+3T2023
#//Code is my own (author)

#How many rows and columns are there in the edx dataset?

num_rows <- nrow(edx)
num_rows

## [1] 9000055

num_columns <- ncol(edx)
num_columns

## [1] 6

#How many 0 through 5's were given as ratings in the edx dataset?
num_zeros <- sum(edx$rating == 0)
num_zeros

## [1] 0

num_ones <- sum(edx$rating == 1)
num_ones

## [1] 345679

num_twos <- sum(edx$rating == 2)
num_twos

## [1] 711422

num_threes <- sum(edx$rating == 3)
num_threes
```

```
## [1] 2121240
```

```
num_fours <- sum(edx$rating == 4)
num_fours
```

```
## [1] 2588430
```

```
num_fives <- sum(edx$rating == 5)
num_fives
```

```
## [1] 1390114
```

```
#How many different movies are in the edx dataset?
num_unique_movies <- length(unique(edx$movieId))
num_unique_movies
```

```
## [1] 10677
```

```
#How many different users are in the edx dataset?
num_unique_users <- length(unique(edx$userId))
num_unique_users
```

```
## [1] 69878
```

```
#How many movie ratings are in each of the following genres in the edx datase
t?
num_drama_ratings <- sum(grepl("Drama", edx$genres))
num_drama_ratings
```

```
## [1] 3910127
```

```
num_comedy_ratings <- sum(grepl("Comedy", edx$genres))
num_comedy_ratings
```

```
## [1] 3540930
```

```
num_thriller_ratings <- sum(grepl("Thriller", edx$genres))
num_thriller_ratings
```

```
## [1] 2325899
```

```
num_romance_ratings <- sum(grepl("Romance", edx$genres))
num_romance_ratings
```

```
## [1] 1712100
```

```
#Which movie has the greatest number of ratings?
ratings_per_movie <- table(edx$title)
movie_with_most_ratings <- names(ratings_per_movie)[which.max(ratings_per_mov
ie)]
movie_with_most_ratings
```

```
## [1] "Pulp Fiction (1994)"
```

```r
#What are the five most given ratings in order from most to least?
ratings_count <- table(edx$rating)
sorted_ratings <- sort(ratings_count, decreasing = TRUE)
five_most_given_ratings <- names(sorted_ratings)[1:5]
five_most_given_ratings
```

```
## [1] "4"   "3"   "5"   "3.5" "2"
```

```r
#Which is more common - half star or whole star ratings?

# Convert ratings to numeric
ratings_numeric <- as.numeric(edx$rating)

# Extract the decimal parts of the ratings
decimal_parts <- ratings_numeric %% 1

# Count the number of ratings with non-zero decimal parts (indicating half-st
ar ratings)
num_half_star_ratings <- sum(decimal_parts != 0)

# Print the count of half-star ratings
print(num_half_star_ratings)
```

```
## [1] 1843170
```

```r
# Count the total number of ratings
total_ratings <- length(ratings_numeric)

# Calculate the number of whole-star ratings
num_whole_star_ratings <- total_ratings - num_half_star_ratings

# Print the count of whole-star ratings
print(num_whole_star_ratings)
```

```
## [1] 7156885
```

Using tools that we have been taught throughout the courses, we now have a better understanding of the dataset we will be working with.

Create Training and Test Sets

We will now create training and test sets to eventually evaluate how well a trained model can respond to new, unseen data. The training set will be used to teach, and the test set is used to assess. We will not use the final_holdout_test data for training, developing, or selecting the algorithm. By doing this, we ensure that the evaluation of the model is unbiased and provides a realistic estimate of how well it will perform in practice.

```r
#############################################################
# Create training and testing sets
#############################################################
```

```r
#set seed for reproducing
set.seed(1)

#split row indices based on userID.
indexes <- split(1:nrow(edx), edx$userId)

#sample 20% of the indices, then unlist the indices, sort them, and store the
m in test
test_ind <- sapply(indexes, function(ind) sample(ind, ceiling(length(ind)*.2)
)) |>
  unlist(use.names = TRUE) |> sort()

#extract rows in test_ind to create the test set
test_set <- edx[test_ind,]

#create the training set by excluding test rows
train_set <- edx[-test_ind,]

#Remove movies that are not in both test and train sets
test_set <- test_set |>
  semi_join(train_set, by = "movieId")
train_set <- train_set |>
  semi_join(test_set, by = "movieId")

#Use pivot_wider to make a matrix with users in rows and movies in columns
y <- select(train_set, movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating)
rnames <- y$userId
y <- as.matrix(y[,-1])
rownames(y) <- rnames
```

Basic Models

In machine learning, we try different models and techniques with the goal of achieving a low RMSE because a lower RMSE indicates better predictive performance of the model.

Trying different models and techniques allows us to select the one that not only fits the training data well but also generalizes well to new data, resulting in a low RMSE on both the training and test datasets.

Different models have different strengths and weaknesses, and they may perform differently depending on the nature of the data and the problem at hand. By trying a variety of models, we can identify the one that best captures the underlying patterns in the data, leading to the lowest RMSE.

Model selection is often an iterative process that involves tuning hyperparameters and selecting the most appropriate features. By experimenting with different models and

tuning their parameters, we aim to optimize the model's performance and minimize the RMSE.

Just the Average Model

A "just the average" model, also known as a naive or baseline model, is one of the simplest predictive models used in machine learning. It predicts the average of the target variable for all instances in the dataset, regardless of the input features.

The "just the average" model serves as a baseline against which more sophisticated machine learning models can be compared.

```
############################################################
# Just the Average Model
############################################################
#//Code for this model adapted from https://learning.edx.org/course/course-v1
:HarvardX+PH125.8x+3T2023/home
#//Good faith efforts were made to make the code my own (the authors)


#Compute RMSE for vectors of ratings and their corresponding predictors
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#Predict the same rating for all movies regardless of user
mu <- mean(y, na.rm = TRUE)
mu

## [1] 3.512338

#Predict all unknown ratings
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse

## [1] 1.060454

#Create a results table, with our first result called "Just the average"
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

method	RMSE
Just the average	1.060455

The RMSE for the “Just the Average” model is 1.06045, meaning that on average, the predictions made by this model deviate from the actual values by approximately 1.06 units. Since the “Just the Average” model is a simple baseline model that does not consider any features or patterns in the data, this RMSE will be used as a baseline reference point for gauging the effectiveness of other more complex models.

Movie Effect Model

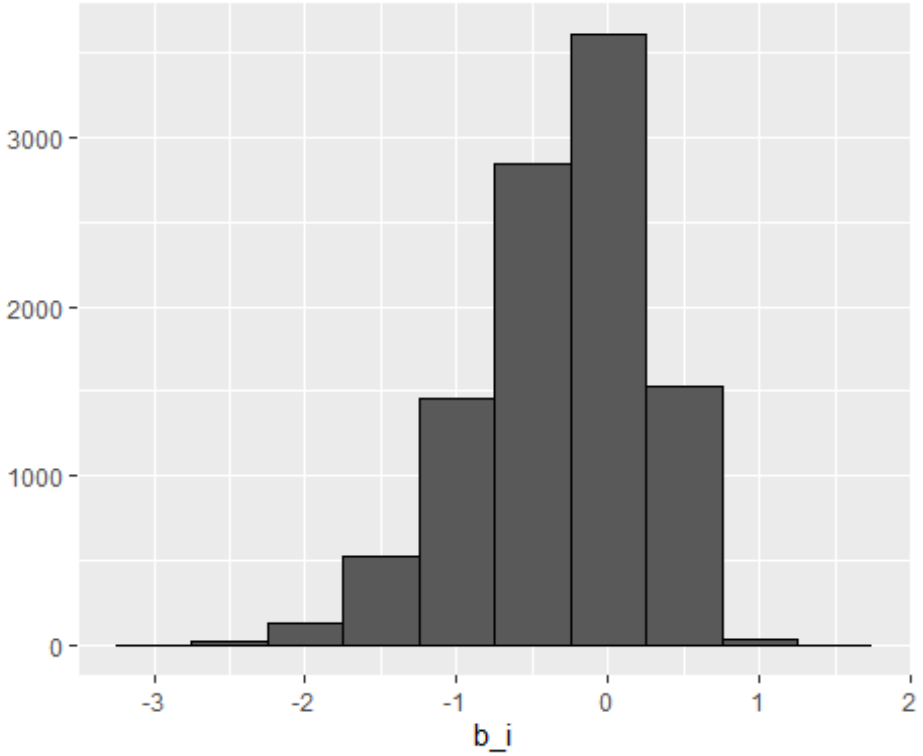
Based on personal preferences, consumers of movies rate some movies higher than others. But are different movies actually rated differently? Let's confirm this with data.

b_i in this instance will represent the bias in movies.

```
############################################################
#Movie Effect Model
############################################################

#Compute the least squares estimate
b_i <- colMeans(y - mu, na.rm = TRUE)

#View the estimates in a plot
qplot(b_i, bins = 10, color = I("black"))
```




```
#create a data frame with three columns, movieID, mu and b_i
fit_movies <- data.frame(movieId = as.integer(colnames(y)),
                         mu = mu, b_i = b_i)

#perform left join between test_set and fit_movies, matching on movieID.
movie_effect_result <- left_join(test_set, fit_movies, by = "movieId") |>

#add a new column prediction to the data frame
mutate(pred = mu + b_i) |>
```

```
#summarize the RMSE
summarize(rmse = RMSE(rating, pred))

#place the RMSE into a Vector
Movie_Effect_RMSE <- movie_effect_result$rmse


#Confirm vector result = summarize result
Movie_Effect_RMSE
```

```
## [1] 0.943936
```

```
#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = Movie_Effect_RMSE ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060455
Movie Effect Model	0.943936

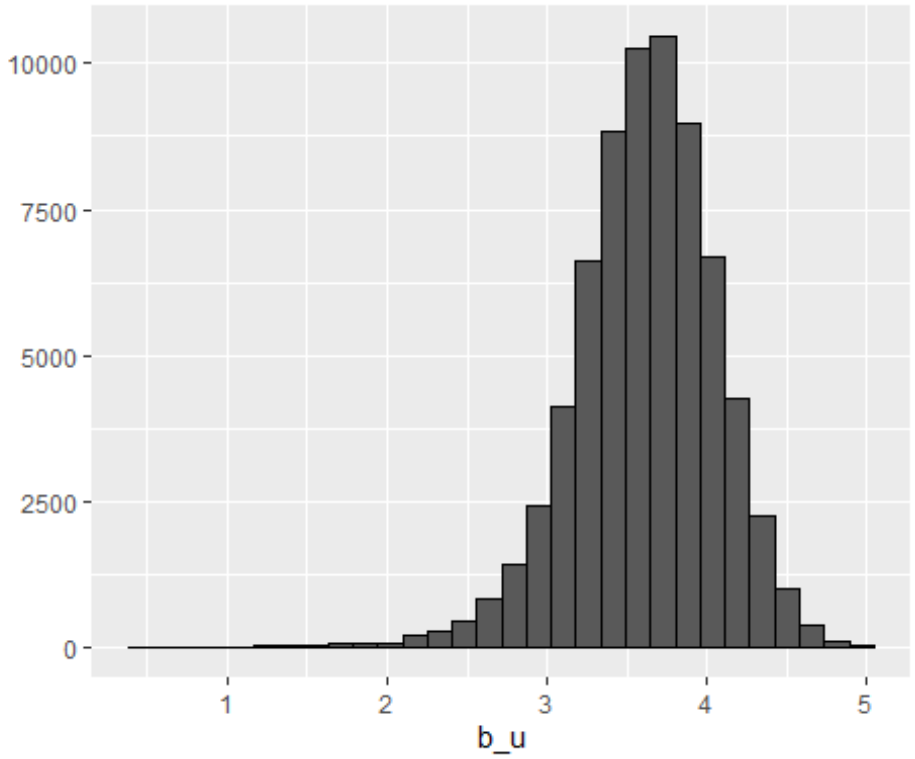
Our Movie Bias RMSE of 0.94393 shows significant improvement compared to the baseline model of 1.06. This tells us that movie bias exists in ratings, and we should consider it in other models.

User Effect Model

We believe that users may also have an effect on movie ratings. For example, the user's mood (happy or sad) could lead to a biased rating of a movie. Therefore, we will test the user effect with b_u representing user bias.

```
############################################################
#User Effect Model
############################################################

#Compute and plot average rating for those that have rated 100 or more movies
b_u <- rowMeans(y, na.rm = TRUE)
qplot(b_u, bins = 30, color = I("black"))
```




```r
#Compute b_u as a user-specific effect on ratings
b_u <- rowMeans(sweep(y - mu, 2, b_i), na.rm = TRUE)

#Construct a predictor and see if RMSE improves
fit_users <- data.frame(userId = as.integer(rownames(y)), b_u = b_u)

user_effect_result <- left_join(test_set, fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  mutate(pred = mu + b_i + b_u) |>
  summarize(rmse = RMSE(rating, pred))

#place the RMSE into a Vector
User_Effect_RMSE <- user_effect_result$rmse

#Confirm vector result = summarize result
User_Effect_RMSE
```

```
## [1] 0.8662513
```

```r
#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User Effect Model",
                                 RMSE = User_Effect_RMSE ))

rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513

The User Bias RMSE is 0.86625, which is better than the previous models of just the average and movie bias. This suggests that user bias has an impact on ratings, and therefore we will include it in other models.

Genre Effect Model

Some genres have significantly more ratings that others as shown in the table below:

Table: Genre Counts

all genres	Freq
Action	2560545
Adventure	1908892
Animation	467168
Children	737994
Comedy	3540930
Crime	1327715
Documentary	93066
Drama	3910127
Fantasy	925637
Film-Noir	118541
Horror	691485
IMAX	8181
Musical	433080
Mystery	568332
Romance	1712100
Sci-Fi	1341183
Thriller	2325899
War	511147
Western	189394

Therefore, we believe that there may be genre bias in the data, and we will test this bias in the model, where b_g represents genre bias.

```
############################################################
#Genre Effect Model
############################################################


# Compute b_g as a genre-specific effect on ratings
fit_genres <- train_set %>% left_join(fit_movies, by = "movieId") %>%
  left_join(fit_users, by='userId') %>%
  group_by(genres) %>% summarize(b_g = mean(rating - mu - b_i - b_u))


# Construct a predictor for genre testing
predict_test_with_genre <- test_set %>% left_join(fit_movies, by='movieId') %
>%
  left_join(fit_users, by='userId') %>%
  left_join(fit_genres, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)


#Generate Genre Effect RMSE
Genre_Effect_RMSE <- RMSE(predict_test_with_genre, test_set$rating)

#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Genre Effect Model",
                                     RMSE = Genre_Effect_RMSE ))

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179

The Genre Effect RMSE of 0.86591 is a very slight improvement versus the User Effect RMSE and a lower RMSE compared to the Movie Effect model.

Regularized Models

Regularized machine learning models are used to prevent over-fitting predictive models. Over fitting occurs when a model learns the training data too well, capturing noise or randomness rather than the underlying patterns. Regularization methods add a penalty

term to the model's function, which encourages the model to learn simpler patterns and avoid overly complex solutions, which leads to better performance on unseen data.

We will perform regularization first on movie bias, then on movie + user bias and finally on movie + user + genre bias to see if we can improve our RMSE.

Regularized Movie Effect

```
############################################################
#Regularized Movie Effect Model
############################################################

#Calculate regularization terms for each movie in the training set

#set regularization parameter to 3
lambda <- 3

#calculate mean rating of all movies in the training set
mu <- mean(train_set$rating)

#group rows of training set by movieID column & calculate summary statistics
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

#Calculate predicted ratings based on regularization
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

regularized_movie_result <- RMSE(predicted_ratings, test_set$rating)

#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = regularized_movie_result ))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179
Regularized Movie Effect Model	0.9438669

The RMSE of this regularized movie effect model is not significantly different from that of the unregularized movie effect model, suggesting that regularization may not be providing

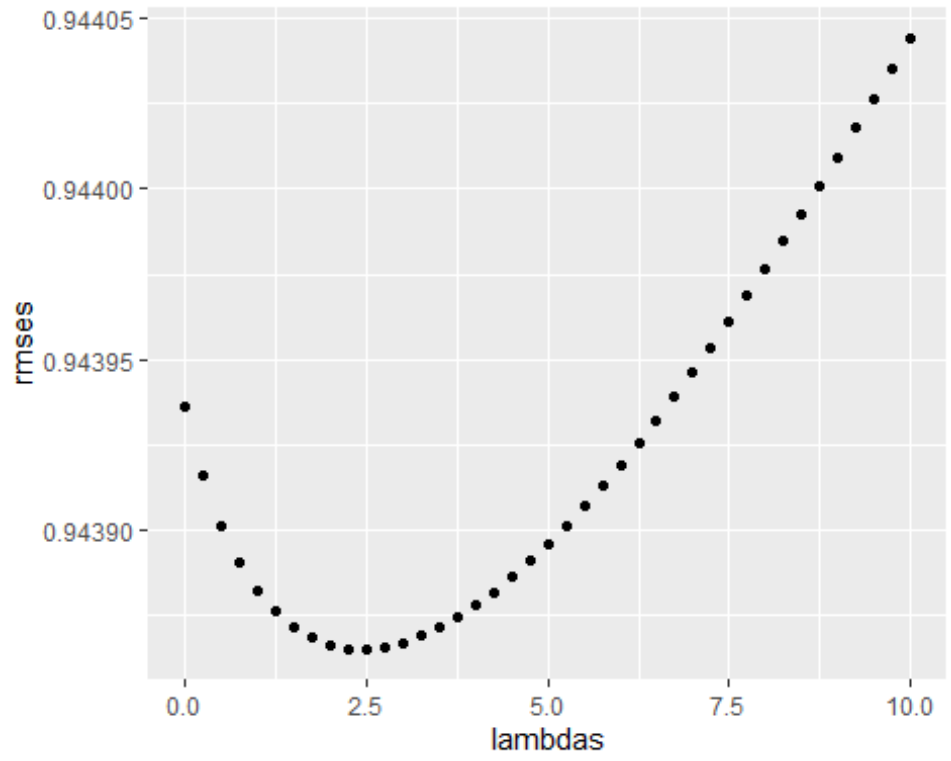
substantial benefits in predictive performance . Therefore, we will now try regularization by combining both the movie effect and the user effect models.

Regularized Movie + User Effect Model

```
########################################
#Regularized Move + User Effect Model
########################################

#Create regularization parameters for collaborative filtering
lambdas <- seq(0, 10, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

#Select the optimal value of lambda by computing RMSE for different lambdas
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```

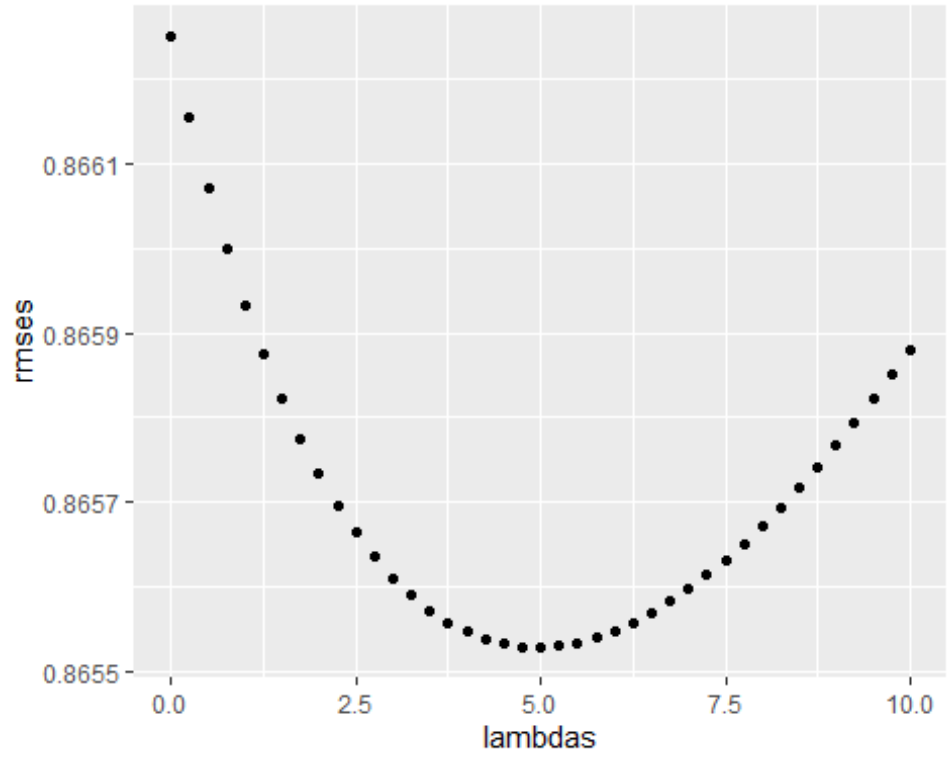


```r
#Identify minimum RMSE
lambdas[which.min(rmses)]
```

```
## [1] 2.5
```

```r
#Select optimal Lambda value for collaborative filtering model with plot
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```



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

## [1] 5

#Re-Identify minimum RMSE
lambdas[which.min(rmses)]

## [1] 5

#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect
Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179
Regularized Movie Effect Model	0.9438669
Regularized Movie + User Effect Model	0.8655286

The RMSE of this regularized movie + user effect model is not significantly different from that of the unregularized movie + user effect model, again suggesting that regularization may not be providing substantial benefits in predictive performance . However, we will now try regularization one more time by combining the movie effect and the user effect and the genre effect models.

Regularized Movie + User + Genre Effect Model

```
##################################################
#Regularized Movie + User +Genre Effect Model
##################################################

#Create regularization parameters for collaborative filtering
lambdas <- seq(0, 20, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
```

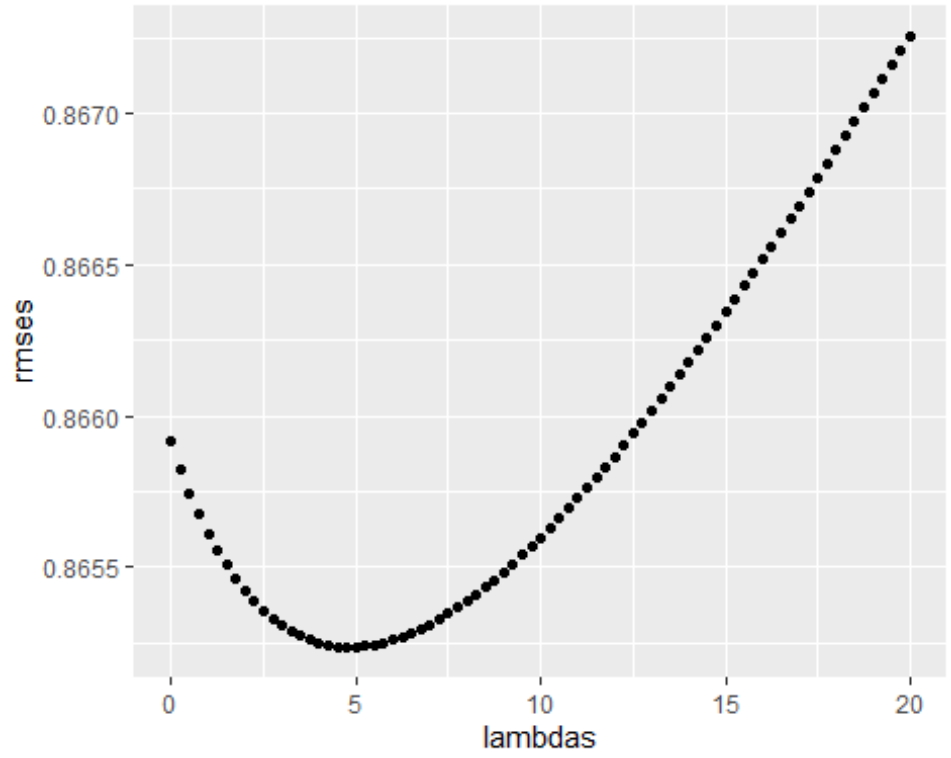
```
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  b_g <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - mu - b_u)/(n() + l))

  reg_genre_model_predict <- test_set %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  return(RMSE(reg_genre_model_predict, test_set$rating))
})

# Plot Lambdas & RMSES
qplot(lambdas, rmses)
```




```
# Find minimum lambda
min_genre_lambda <- lambdas[which.min(rmses)]
min_genre_lambda

## [1] 4.75
```

```
# Calculate the minimum RMSE
rmse_reg_genre_model <- min(rmses)
rmse_reg_genre_model

## [1] 0.865237

#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User + Genre
Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179
Regularized Movie Effect Model	0.9438669
Regularized Movie + User Effect Model	0.8655286
Regularized Movie + User + Genre Effect Model	0.8652370

The RMSE of this regularized movie + user + genre effect model is not significantly different from that of the unregularized movie + user + genre effect model, again suggesting that regularization may not be providing substantial benefits in predictive performance.

This concludes our regularization efforts, and we now move on to the Matrix Factorization technique.

Matrix Factorization Method

Matrix Factorization is a machine learning method to find patterns or relationships in large sets of data that are organized in a matrix format, like a table with rows or columns. The goal of matrix factorization is to break down the complex matrix into simpler, more interpretative parts.

Multiple attempts at Matrix Factorization were made that ultimately failed. We have included those attempts in the appendix. The successful Matrix Factorization method is included in this section.

Thanks and credit are due to Papacosmas Nicholas, Yixuan Qui, and Simon Funk for the inspiration and work to attempt the Matrix Factorization method.

We will apply Matrix Factorization with the help of the "recosystem" package with creates a recommender system using Matrix Factorization. For each iteration, the training will update the model parameters and evaluate the performance of the model and the objective

function value. The goal is to minimize the RMSE and the objective function value over the course of the iterations to improve the accuracy of the model.

```
##########################################################
#Matrix Factorization - Recosystem
##########################################################

#//code adapted from github.com/papacosmas/MovieLens and github/yixuan/recosy
stem
#//Final code is my own (author)

#Define RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#Arrange training data is sparse matrix form
edx_factorization <- edx %>% select(movieId,userId,rating)
validation_factorization <- final_holdout_test %>% select(movieId,userId,rati
ng)

#Save factorizations as tables on HD
write.table(edx_factorization , file = "trainingset.txt" , sep = " " , row.na
mes = FALSE, col.names = FALSE)
write.table(validation_factorization, file = "validationset.txt" , sep = " "
, row.names = FALSE, col.names = FALSE)

set.seed(1)
training_dataset <- data_file( "trainingset.txt")
validation_dataset <- data_file( "validationset.txt")

#create a model object called r
r = Reco()

#set tuning parameters and candidate values
opts = r$tune(training_dataset, opts = list(dim = c(10, 20, 30), lrate = c(0.
1, 0.2),
                                            costp_l1 = 0, costq_l1 = 0,
                                            nthread = 1, niter = 10))

#train the model using tuning parameters
r$train(training_dataset, opts = c(opts$min, nthread = 1, niter = 20))

## iter      tr_rmse          obj
##    0       0.9725   1.2028e+07
##    1       0.8718   9.8796e+06
##    2       0.8390   9.1796e+06
##    3       0.8174   8.7601e+06
##    4       0.8016   8.4792e+06
##    5       0.7898   8.2775e+06
```

```
##    6       0.7803   8.1272e+06
##    7       0.7724   8.0078e+06
##    8       0.7657   7.9131e+06
##    9       0.7598   7.8339e+06
##   10       0.7546   7.7669e+06
##   11       0.7500   7.7090e+06
##   12       0.7459   7.6589e+06
##   13       0.7422   7.6139e+06
##   14       0.7388   7.5766e+06
##   15       0.7357   7.5416e+06
##   16       0.7328   7.5112e+06
##   17       0.7302   7.4824e+06
##   18       0.7277   7.4575e+06
##   19       0.7254   7.4354e+06
```

```
## Store the iterative RMSE values in a vector
rmse_values <- c(0.9725, 0.8718, 0.8390, 0.8174, 0.8016, 0.7898, 0.7803, 0.77
24, 0.7657, 0.7598, 0.7546, 0.7500, 0.7459, 0.7422, 0.7388, 0.7357, 0.7328, 0
.7302, 0.7277, 0.7254)

# Extract the RMSE values
start_mf_rmse <- max(rmse_values)
end_mf_rmse <- min(rmse_values)
avg_mf_rmse <- mean(rmse_values)

# Create a data frame with all RMSE values
mf_rmse_df <- data.frame(
  method = c("Matrix Factorization - Start",
             "Matrix Factorization - End",
             "Matrix Factorization - Mean"),
  RMSE = c(start_mf_rmse, end_mf_rmse, avg_mf_rmse)
)

# Update and display rmse_results table
rmse_results <- bind_rows(rmse_results, mf_rmse_df)

rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179
Regularized Movie Effect Model	0.9438669
Regularized Movie + User Effect Model	0.8655286
Regularized Movie + User + Genre Effect Model	0.8652370

Method	RMSE
Matrix Factorization - Start	0.9725000
Matrix Factorization - End	0.7254000
Matrix Factorization - Mean	0.7776800

As we iterated using the Matrix Factorization tuning technique, the training RMSE decreases gradually with each iteration, indicating that the model is learning from the training data and improving its predictive performance. Similarly, the objective function value also decreases, which suggests that the model is optimizing its parameters to better fit the training data.

Since this method has achieved the lowest RMSE, we will now validate the Matrix Factorization model against the final_holdout_data set.

Matrix Factorization – Validation Testing

```
##########################################################
#Matrix Factorization – Validation Testing
##########################################################

# Write predictions to a tempfile on HD
stored_prediction = tempfile()

#make predictions on validation set
r$predict(validation_dataset, out_file(stored_prediction))

## prediction output generated at C:\Users\TBLACK~1\AppData\Local\Temp\Rtmp4M
ZF2n\file31c032e2cd6

#display first 20 predictions
print(scan(stored_prediction, n = 20))

##  [1] 4.21486 4.98825 4.68499 3.39018 4.44704 2.76951 4.07904 4.34693 4.413
95
## [10] 3.43969 3.73300 3.64155 3.41388 3.97336 3.73156 4.58589 3.78249 2.896
13
## [19] 3.91550 3.70823

#create dataframe from tables
real_ratings <- read.table("validationset.txt", header = FALSE, sep = " ")$V3
pred_ratings <- scan(stored_prediction)

#calculate RMSE
final_holdout_test_rmse <- RMSE(real_ratings,pred_ratings)
final_holdout_test_rmse
```

```
## [1] 0.7830654

#Update and display rmse_results table.
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Final Holdout Test",
                                     RMSE = final_holdout_test_rmse))

rmse_results %>% knitr::kable()
```

Method	RMSE
Just the average	1.0604545
Movie Effect Model	0.9439360
User Effect Model	0.8662513
Genre Effect Model	0.8659179
Regularized Movie Effect Model	0.9438669
Regularized Movie + User Effect Model	0.8655286
Regularized Movie + User + Genre Effect Model	0.8652370
Matrix Factorization – Start	0.9725000
Matrix Factorization – End	0.7254000
Matrix Factorization – Mean	0.7776800
Final Holdout Test	0.7830654

The final holdout test RMSE is 0.78306, which is in-line with our Matrix Factorization Mean of 0.77768, and below the course-required RMSE of <0.86490.

Conclusions

Summary

The goal of this project was to create a movie recommendation system from the MovieLens dataset using the tools and techniques that we learned throughout the Harvard Data Science courses, focused on machine learning techniques. We created a working dataset and took approaches to better understand the data we were working with. We split the data into a training set, a test set, and a final holdout set, taking care to not use the final holding for training, developing, or selecting our algorithm, and only using it to evaluate the RMSE of our final algorithm.

We start by creating a model called "Just the Average" to create a baseline from which to improve from. We then employed a variety of machine learning methods in an attempt to achieve a RMSE <0.86490, and then test that final method against the validation set.

Our best working algorithm was the Matrix Factorization method, which delivered a mean training RSMSE of 0.77768 and a predicted RMSE of 0.78306. This meets the course requirements of a predicted RMSE <0.86490.

Limitations

Due to the size of the dataset, R Studio constantly crashed or errored out, especially when attempting to tune the data or run the more complicated algorithms. This limited the number of iterations and methods that could be run. Knitting RMD to .pdf also errored out constantly due to size limitations.

External computing resources, such as cloud computing platforms or high-performance computing clusters that offer larger memory capacities and computational power could be used to process this larger dataset.

Future Work

Other more advanced deep learning approaches commonly used in recommender systems such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Long Short-Term Memory Networks (LSTM) could be applied to further refine this analysis.

Netflix replaced its traditional star rating system with a thumbs up and thumbs down system in early 2017 and added a two thumbs up option in early 2022. An analysis comparing the effectiveness of this rating system compared to the traditional star rating system would be interesting.

Appendix A – Alternate Matrix Factorization Attempts

Multiple attempts at Matrix Factorization were made where code errored out to the the size of the data set.
We have included those attempts in this appendix. Code resulted in an error but are shown to display multiple other attempts at machine learning methods.

```
# R code that resulted in an error
# This code will be displayed but not executed in R

###############################################
# Appendix A - Matrix Factorization Attempts
###############################################

#Multiple attempts at Matrix Factorization were made where code errored out t
o the the size of the data set.
#We have included those attempts in this appendix.
#Code will result in an error, but are shown to display multiple other attemp
ts at machine learning methods.

###################################
# Matrix Factorization - SVD
###################################
# Matrix factorization using Singular Value Decomposition (SVD) is a techniqu
e that breaks down a matrix into three smaller matrices to capture underlying
patterns and relationships within the data
# Error details: Error in mult(A, VJ) : requires numeric/complex matrix/vecto
r arguments

if(!require(bigmemory)) install.packages("bigmemory", repos = "http://cran.us
.r-project.org")
if(!require(irlba)) install.packages("irlba", repos = "http://cran.us.r-proje
ct.org")
library(bigmemory)
library(irlba)

# Convert train_set to a big.matrix object
train_bigmatrix <- as.big.matrix(as.matrix(train_set))
nrow(train_bigmatrix)
ncol(train_bigmatrix)

# Adjust the value of nv to be less than the minimum dimension of the input m
atrix
nv <- min(nrow(train_bigmatrix), ncol(train_bigmatrix)) - 4

# Perform Singular Value Decomposition (SVD) using irlba
svd_result <- irlba(train_bigmatrix, nv = nv)

# Extract singular vectors and values
```

```r
u <- svd_result$u
d <- svd_result$d
v <- svd_result$v

# Reconstruction using SVD
reconstructed <- u %*% diag(d) %*% t(v)

# Extract predicted ratings for test set
predicted_ratings <- reconstructed[test_set$userId, test_set$movieId]

# Calculate RMSE
svd_result <- RMSE(predicted_ratings, test_set$rating)
svd_result

##########################################
# Non-Negative Matrix Factorization (NMF)
##########################################
#Similar to SVD but constrains the factors to be non-negative, which can lead
to more interpretable results.
# Error details: error in evaluating the argument 'object' in selecting a met
hod for function 'basis': object 'nmf_result' not found

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("Biobase")
n
install.packages("NMF")
library(NMF)

# Convert train_set to a matrix if it's not already in matrix format
train_matrix <- as.matrix(train_set)

# Remove rows with NA values
train_matrix_numeric <- na.omit(train_matrix)

class(train_matrix_numeric)

# Replace negative values in train_matrix with zeros
train_matrix_numeric[train_matrix < 0] <- 0

# Filter out non-numeric columns from train_matrix
train_matrix_numeric <- as.matrix(as.numeric(train_matrix_numeric))

# Specify the number of latent factors (rank) for NMF
rank <- 10  # Adjust as needed

# Perform Non-Negative Matrix Factorization (NMF)
nmf_result <- nmf(train_matrix_numeric, rank = rank)
```

```r
# Extract factor matrices (basis and coefficient matrices)
basis <- basis(nmf_result)  # Basis matrix (W)
coef <- coef(nmf_result)    # Coefficient matrix (H)

# Reconstruct the ratings matrix using factor matrices
reconstructed_matrix <- basis %*% coef

##################################################
#Matrix Factorization - Recommenderlab
##################################################
# The recommenderlab package provides matrix factorization methods such as SV
D and NMF
# Since the previous SVD and NMF methods errored out, we are trying recommend
erlab
# Error details: Error: attempt to apply non-function

# Install and load required packages
install.packages("recommenderlab")
library(recommenderlab)

# Convert train_set to a realRatingMatrix object (if not already in that form
at)
train_matrix <- as(train_set, "realRatingMatrix")

# Define the SVD recommender function
SVD_recommender <- function(train_matrix, ...) {
  svd_model <- svd(as(train_matrix, "matrix"))
  list(U = svd_model$u, D = diag(svd_model$d), V = svd_model$v)
}

# Register the new SVD recommender method
recommenderRegistry$register(method = "SVD", recommender = SVD_recommender, d
ataType = "realRatingMatrix", parameter = list())

# Define a new SVD recommender method
set.seed(123)  # Set seed for reproducibility
SVD_recommender <- function(data, ...) {
  svd_model <- svd(as(data, "matrix"))
  list(U = svd_model$u, D = diag(svd_model$d), V = svd_model$v)
}

# Register the new SVD recommender method
recommenderRegistry$register(method = "SVD", recommender = SVD_recommender, d
ataType = "realRatingMatrix", parameter = list())

# Convert the training set to a realRatingMatrix
train_matrix <- as(train_set, "realRatingMatrix")

# Train the SVD model on the training set
```

```r
svd_model <- Recommender(train_matrix, method = "SVD")

# Generate recommendations for users in the test set
recommendations <- predict(svd_model, newdata = test_set, n = 10)

# Show the top recommended items for each user in the test set
top_recommendations <- as(recommendations, "list")
top_recommendations

################################################
#Matrix Factorization - Minibatch
################################################
# Because the previous matrix factorization methods have failed due to size c
ontraints, we make one last attempt
# We attempt to test our matrix in batches of 1,000 each for ~70,000 rows
# Error details: Error: cannot allocate vector of size 716.0 Gb

# Create user-item rating matrix from the training set
rating_matrix <- sparseMatrix(i = train_set[,1], j = train_set[,2], x = train
_set[,3])

# Minibatch control parameters (adjust as needed)
batch_size <- 1000  # Adjust batch size based on memory constraints
init <- "svdpp"    # Choose appropriate initialization method

# Define the matrix factorization model using caret
model_control <- trainControl(method = "list",
                              summaryFunction = oneSE,
                              returnData = TRUE)

# Define the matrix factorization function with minibatch support
mf_function <- function(train_data, ...) {
  # Extract user and item IDs for minibatch processing
  user_ids <- train_data[, 1]
  item_ids <- train_data[, 2]
  ratings <- train_data[, 3]
```

Appendix B - Random Forest Method

We also tested the edx dataset using Random Forest Methods. The RMSEs for these methods were above 1, much higher than other methods we used. Possible reasons for this are: A) The datasets patterns and relationships are challenging for the Random Forest model to capture accurately. B) The features in the random forest model may not represent the underlying patterns in the dataset. C) The random forest model may be overfitting the training data. Ultimately, the random forest method was not the best choice for this dataset. There are better machine learning algorithms better suited for this dataset.

```
# Your R code that resulted in an error
# This code will be displayed but not executed in R

##############################################
# Appendix B - Random Forest Methods
##############################################
# We also tested the edx dataset using Random Forest Methods.
# The RMSEs for these methods were above 1, much higher than other methods we
used.
# Possible reasons for this are:
# a. The datasets patterns and relationships are challenging for the Random F
orest model to capture accurately.
# b. The features in the random forest model may not represent the underlying
patterns in the dataset.
# c. The random forest model may be overfitting the training data.
# Ultimately, the random forest method was not the best choice for this datas
et.
# There are better machine learning algorithms better suited for this dataset
.

##################################################
#Distributed Random Forest Method - Primary Genre
##################################################

# Extract the first genre before the "|" character
train_set$primary_genre <- sub("\\|.*", "", train_set$genres)

# Print the first few rows to verify
head(train_set)
unique_primary_genres <- sort(unique(train_set$primary_genre))
unique_primary_genres

str(train_set)

# Create dummy variables for "primary_genre" using model.matrix()
genre_dummies <- model.matrix(~ primary_genre - 1, data = train_set)

# Print the first few rows to see the dummy variables
head(genre_dummies)
```

```r
# Train Random Forest model using dummy variables
rf_model <- randomForest(x = genre_dummies,
                         y = train_set$rating,
                         ntree = 50,  # Adjust as needed
                         importance = TRUE)  # Compute variable importance

# Print model summary
print(rf_model)
```