Movie Recommendation System HarvardX's Professional Certificate in Data Science Machine Learning Capstone Project

Usha Saravanakumar

October 2022

Overview

Online streaming platforms and e-commerce most widely use Recommendation models to recommend products, movies, or music to users based on their preferences, increasing user satisfaction and retention. This project aims to develop a movie recommendation model that recommends movies to users using the MovieLens dataset provided by GroupLens, a research lab at the University of Minnesota. The main steps in the project involve the following: - Exploring the dataset's structure. - Performing Exploratory Data Analysis (EDA). - Developing a machine-learning model for the movie recommendation system.

Methods/Analysis

Process and Techniques used

1. We begin by downloading the dataset.
2. Split the dataset into training (edx) and testing (validation) datasets. We are not supposed to use the Validation set (the final hold-out test set) to experiment with various parameters or cross-validation. So we will create an additional partition of training and testing datasets from the edx dataset to experiment with multiple parameters or cross-validation.
3. Next, we will explore the data by creating visualizations and gaining insights into the dataset.
4. Based on the insights gained from the EDA process, we will create machine learning models starting with the most straightforward approach and progressively improving the model performance. We will be using Residual Mean Square Error (RMSE) to measure the performance of various models, and our goal is to reduce the RMSE below the target value of 0.8649. When the model reaches the RMSE target in the testing set, we will train the edx dataset with the model and use the validation dataset to test the model performance.

Data Extraction

Downloading MovieLens Dataset

Full Version of MovieLens Dataset has 27 million ratings and 1.1 million tag applications applied to 58,000 movies by 280,000 users. However, this project will use the 10M version of the MovieLens dataset released in Jan 2009, with 10 million ratings on 10,000 movies by 72,000 users.

```r
# Import required libraries
if(!require(formatR)) install.packages("formatR", repos = "http://cran.us.r-project.org")

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse, warn.conflicts=F, quietly=T)
library(caret, warn.conflicts=F, quietly=T)
library(data.table, warn.conflicts=F, quietly=T)

# Download 10M version of MovieLens dataset
dl <- tempfile()
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)

colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

Data Transformation

Splitting MovieLens Dataset into Training and Testing Datasets

We split the MovieLens dataset into a training dataset called edx and an evaluation dataset called validation. The edx and validation set will have 90% and 10% of MovieLens data, respectively. While we create the validation set, we ensure that all users and movies in the validation set are also available in the edx set.

```r
# Validation set will be 10% of MovieLens data
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1,
    p = 0.1, list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

# Make sure userId and movieId in validation set are also
# in edx set
validation <- temp %>%
    semi_join(edx, by = "movieId") %>%
    semi_join(edx, by = "userId")

# Add rows removed from the validation set back into the
# edx set
removed <- anti_join(temp, validation)
```

```
edx <- rbind(edx, removed)

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

Split the edx dataset into Training and Testing Dataset

We are not supposed to use the Validation set (the final hold-out test set) to experiment with multiple parameters or cross-validation. So we will create an additional partition of training and test sets from the edx dataset to experiment with various parameters or use cross-validation. While splitting the testing set, we ensure that all users and movies in the training set are also in the training set.

```
test_index <- createDataPartition(y = edx$rating, times = 1,
    p = 0.1, list = FALSE)
train_set <- edx[-test_index, ]
temp <- edx[test_index, ]

# Matching userId and movieId in both train and test sets
test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

# Adding back rows into the train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Data Exploration and Visualization

Structure of the dataset

Let us begin by exploring the dataset structure. Training Dataset (edx) has 9,000,055 rows while Validation dataset has 999,999 rows. Each row contains a rating given by a user for a movie. Both datasets contain six columns - userId, movieId, rating, timestamp, title, and genres.

```
# str() function in R Language is used for compactly
# displaying the internal structure of an R object We use
# str() to analyze the structure of the training and
# testing datasets
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  - attr(*, ".internal.selfref")=<externalptr>
```

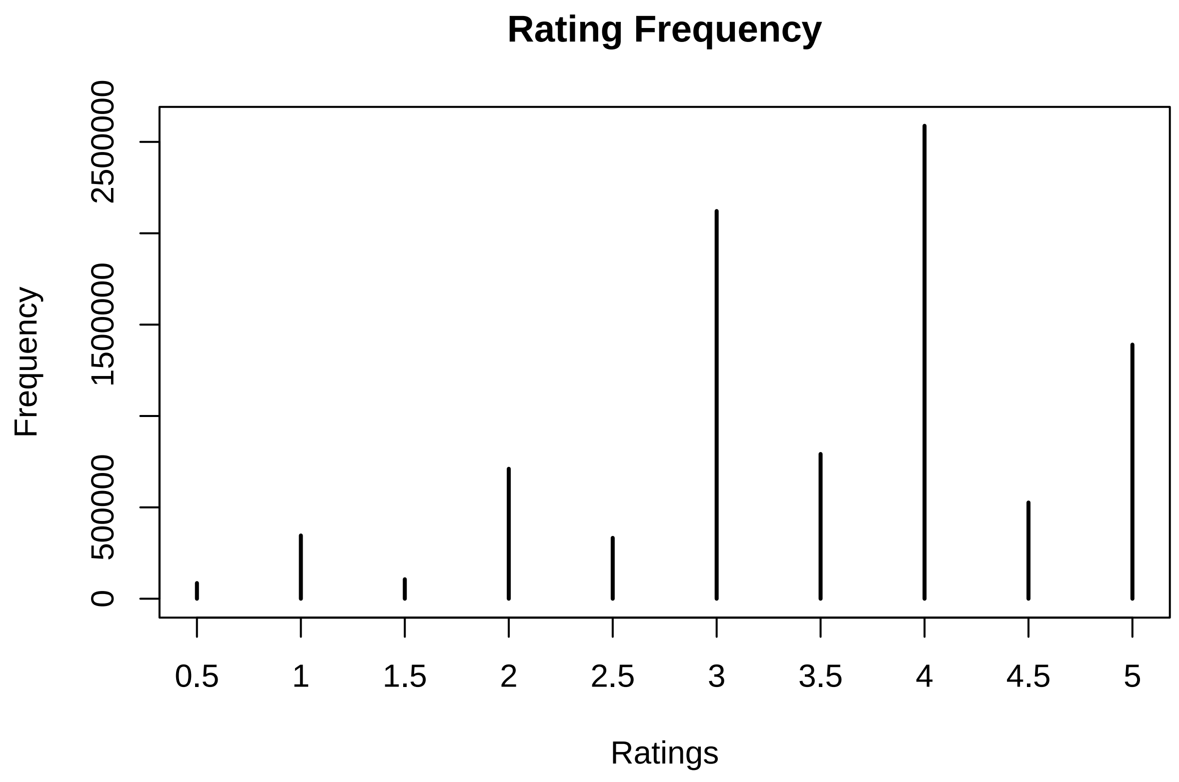
```
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999993 obs. of  6 variables:
##  $ userId   : int  1 2 2 3 3 3 4 4 4 5 ...
##  $ movieId  : num  588 1210 1544 151 1288 ...
##  $ rating   : num  5 4 3 4.5 3 3 3 3 5 3 ...
##  $ timestamp: int  838983339 868245644 868245920 1133571026 1133571035 1164885617 844416656 84441707
##  $ title    : chr  "Aladdin (1992)" "Star Wars: Episode VI - Return of the Jedi (1983)" "Lost World:
##  $ genres   : chr  "Adventure|Animation|Children|Comedy|Musical" "Action|Adventure|Sci-Fi" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

Visualization #1: Rating Frequency

There are 10 rating categories ranging from 0.5 to 5. The following table and plot show that users often use whole numbers to rate a movie, and rating 4 has the highest frequency.

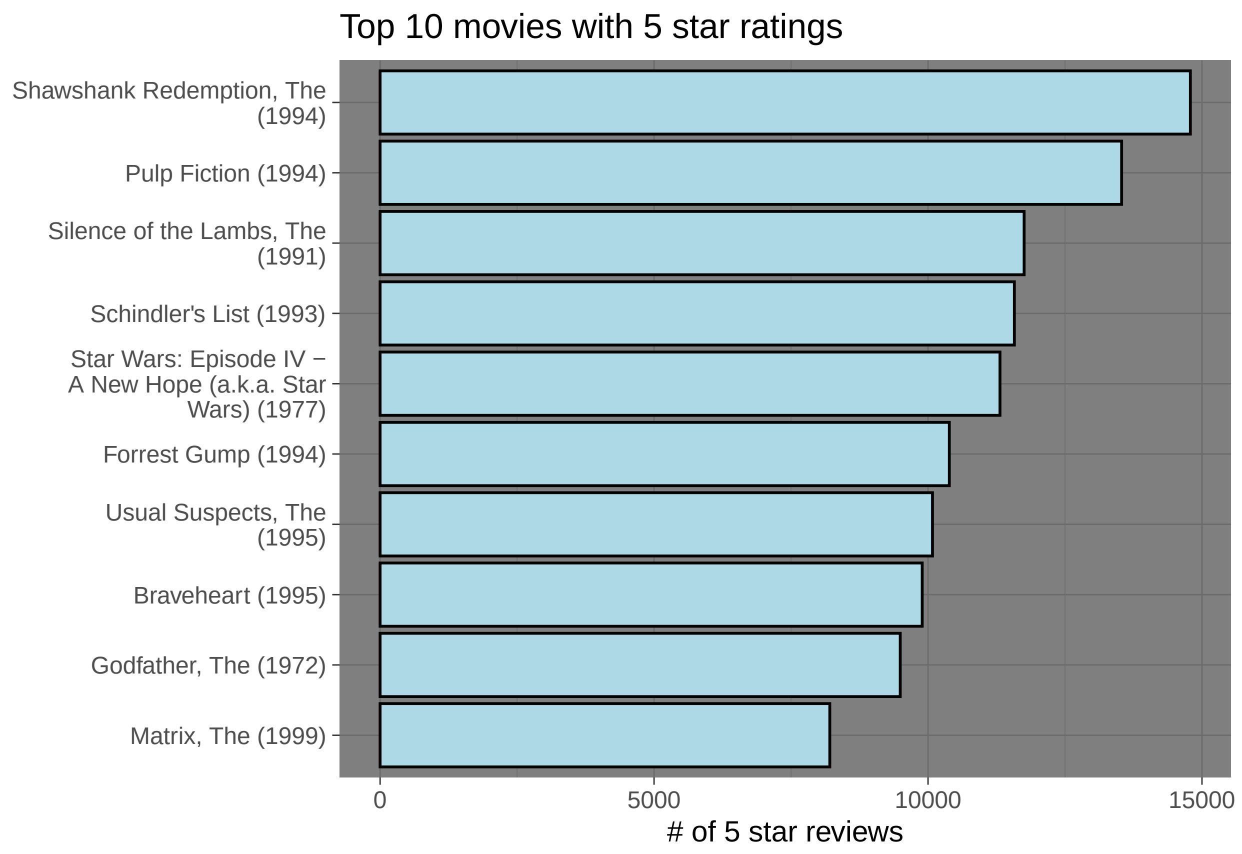
```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85420  345935  106379  710998  332783 2121638  792037 2588021  526309 1390541
```



Visualization #2: Movies with most five-star ratings

The following plot shows the Top 10 movies with the most five-star ratings in this dataset. It is evident from the barplot that some movies get higher ratings than others. We could notice that "Shawshank Redemption, The (1994)" received the highest (closer to 15,000) 5-star ratings.

```
# Plot top 10 movies with the most five-star ratings
edx %>%
    filter(rating == 5) %>%
    group_by(title) %>%
    summarize(count = n()) %>%
    arrange(desc(count)) %>%
    top_n(10, count) %>%
    ggplot(aes(count, reorder(title, count))) + ggtitle("Top 10 movies with 5 star ratings") +
    geom_bar(color = "black", fill = "lightblue", stat = "identity") +
    scale_y_discrete(labels = function(x) str_wrap(x, width = 25)) +
    xlab("# of 5 star reviews") + ylab(NULL) + theme_dark()
```



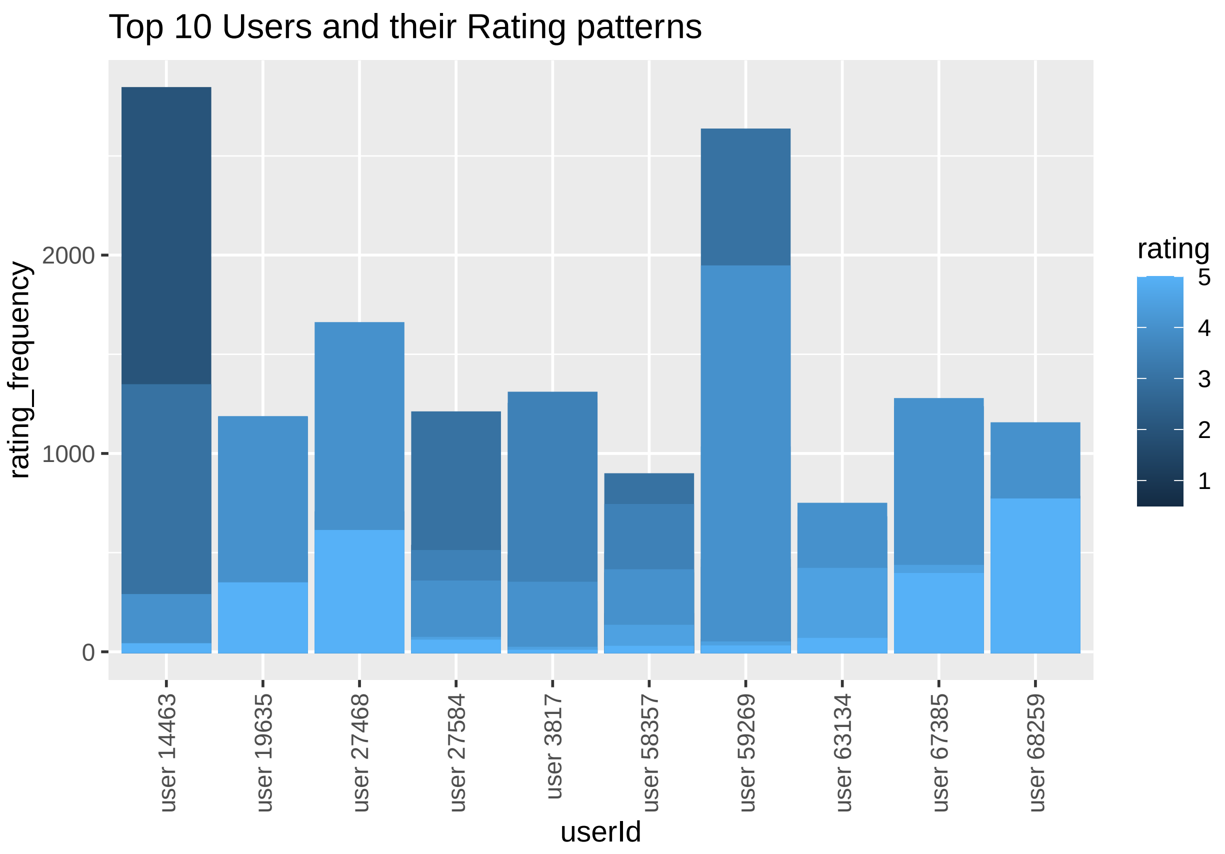
Visualization #3: Top 10 Users and their Rating pattern

The following plot shows the Top 10 users with the highest rating counts. We could notice from the stacked barplot that some users (e.g., user 66259) are more generous in providing higher ratings, while some (e.g., user 14463) provide relatively lower ratings.

```
# Find the top 10 users with the highest rating counts
top_10_users <- edx %>%
    group_by(userId) %>%
    summarize(user_rating_count = n()) %>%
    arrange(desc(user_rating_count)) %>%
    top_n(10, user_rating_count)
```

```
# Find the rating frequency of the top 10 users
data <- inner_join(edx, top_10_users) %>%
    group_by(userId, rating) %>%
    summarize(rating_frequency = n())
data[["userId"]] = paste("user", data$userId)

# Plot the Top 10 users and their rating patterns
p <- ggplot(data, aes(fill = rating, y = rating_frequency, x = userId,
    color = rating)) + geom_bar(position = "dodge", stat = "identity") +
    ggtitle("Top 10 Users and their Rating patterns")
p + theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
    hjust = 1))
```



Modeling Approach

We will develop several machine learning models and compare their performance to determine the best model. However, we need to use a standard method to measure the performance across all models. We will use Residual-Mean-Square Error (RMSE), a frequently used measure of the differences between values predicted by a model and the observed values. We can find the RMSE of a model by:

1. Finding the residuals - the difference between values predicted by the model and the observed values.
2. Squaring the residuals.
3. Finding the average of the squared residuals.
4. Finding the square root of the average.

We will use the training and testing dataset partitioned from the edx dataset to train and evaluate our models. Once the model reaches the goal with an RMSE below the target value of 0.8649, we will train the model with the edx dataset and test the model against the final hold-out validation test set.

Model 1 - Average of all movie ratings

We will begin with a simple model by making predictions only using the average of all movie ratings given by all users and ignoring all other predictors and biases. The average movie rating of our training dataset is 3.512459, and if we use this average value as the prediction, the RMSE of the model is 1.126149. This RMSE is pretty much higher than our goal of 0.8649. So, let us continue to improve this model further.

```
# Find the average of all movie ratings given by all users
# in the training set
y_pred_model1 <- mean(train_set$rating)
paste("Average movie rating of training dataset           :",
    y_pred_model1)
```

```
## [1] "Average movie rating of training dataset           : 3.51245112696557"
```

```
# Measure model performance by finding RMSE
model1_rmse <- sqrt(mean((y_pred_model1 - test_set$rating)^2))

# Print the results
results <- tibble(Method = "Model 1: Average", RMSE = model1_rmse)
results %>%
    kbl(caption = "MODEL PERFORMANCE RESULTS", digits = 4) %>%
    kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 1: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598

Model 2 - Address the bias in movie rating patterns

We noticed from Visualization #2 in the EDA section that some movies get higher ratings than the rest. We also know that while we created the testing sets, we ensured that all movie Ids in the testing set were also in our training set. So, let us check if we could improve the model performance further by addressing the bias in movie ratings by using each movie's average rating instead of the average of all movie ratings given by all users.

```
# Find the average rating of each movie
avg_movie_rating_df <- train_set %>%
    group_by(movieId) %>%
    summarize(avg_movie_rating = mean(rating - y_pred_model1))

# Predict ratings in the testing set
y_pred_model2 <- y_pred_model1 + inner_join(test_set, avg_movie_rating_df) %>%
    pull(avg_movie_rating)

# Measure model performance by finding RMSE
model2_rmse <- sqrt(mean((y_pred_model2 - test_set$rating)^2))
```

```r
# Print the results
results <- bind_rows(results, tibble(Method = "Model 2: Average + Movie rating pattern bias", RMSE = mod
results %>%
  kbl(caption="MODEL PERFORMANCE RESULTS", digits=4) %>%
  kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 2: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598
Model 2: Average + Movie rating pattern bias	0.9424

Model 3 - Addressing the bias in Users' rating patterns

Let us see what we could do to improve our model further. We gained insight from Visualization #3 in the EDA section that some users could be more generous than others. We also know that while we created the testing sets, we ensured that all user Ids in the testing set were also in our training set. So, to address the bias in users' rating patterns, we could take the average rating given by each user.

```r
# Find the average rating given by each user in the
# training set
avg_user_rating_df <- inner_join(train_set, avg_movie_rating_df) %>%
    group_by(userId) %>%
    summarize(avg_user_rating = mean(rating - y_pred_model1 -
        avg_movie_rating))

# Predict ratings in the testing set
y_pred_model3 <- y_pred_model1 + inner_join(test_set, avg_movie_rating_df) %>%
    pull(avg_movie_rating) + inner_join(test_set, avg_user_rating_df) %>%
    pull(avg_user_rating)

# Measure model performance by finding RMSE
model3_rmse <- sqrt(mean((y_pred_model3 - test_set$rating)^2))
```

```r
# Print the results
results <- bind_rows(results, tibble(Method = "Model 3: Average + Movie & User rating pattern bias", RMS
results %>%
  kbl(caption="MODEL PERFORMANCE RESULTS", digits=4) %>%
  kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 3: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598
Model 2: Average + Movie rating pattern bias	0.9424
Model 3: Average + Movie & User rating pattern bias	0.8645

Model 4 - Addressing the bias in genres rating pattern

There could be bias in user rating based on the movie genres. Let us see if the model improves further by addressing the genre effect.

```r
# Find the average rating given by each user in the
# training set
avg_genres_rating_df <- inner_join(train_set, avg_movie_rating_df) %>%
    inner_join(avg_user_rating_df) %>%
    group_by(genres) %>%
    summarize(avg_genres_rating = mean(rating - y_pred_model1 -
        avg_movie_rating - avg_user_rating))

# Predict ratings in the testing set
y_pred_model4 <- y_pred_model1 + inner_join(test_set, avg_movie_rating_df) %>%
    pull(avg_movie_rating) + inner_join(test_set, avg_user_rating_df) %>%
    pull(avg_user_rating) + left_join(test_set, avg_genres_rating_df) %>%
    pull(avg_genres_rating)

# Measure model performance by finding RMSE
model4_rmse <- sqrt(mean((y_pred_model4 - test_set$rating)^2))
```

```r
# Print the results
results <- bind_rows(results, tibble(Method = "Model 4: Average + Movie, User & Genres rating pattern bi
results %>%
  kbl(caption="MODEL PERFORMANCE RESULTS", digits=4) %>%
  kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 4: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598
Model 2: Average + Movie rating pattern bias	0.9424
Model 3: Average + Movie & User rating pattern bias	0.8645
Model 4: Average + Movie, User & Genres rating pattern bias	0.8642

Model 5 - Regularization

There seem to be obscure movies with big predictions in our model. More significant estimates are more likely to occur when fewer users rate the film. Regularization helps penalize large estimates from smaller samples by adding a penalty for large values, thus reducing the RMSE. So, let us try regularization and see how it could improve our model results even more.

```r
# Define a set of lambdas to tune
lambdas <- seq(0, 10, 0.25)

# Save RMSE for each lambda into a variable rmses
rmses <- sapply(lambdas, function(x) {
    avg_movie_rating_df <- train_set %>%
        group_by(movieId) %>%
        summarize(avg_movie_rating = sum(rating - y_pred_model1)/(n() +
            x))
    avg_user_rating_df <- train_set %>%
        left_join(avg_movie_rating_df, by = "movieId") %>%
        group_by(userId) %>%
        summarize(avg_user_rating = sum(rating - y_pred_model1 -
            avg_movie_rating)/(n() + x))
    avg_genres_rating_df <- train_set %>%
        left_join(avg_movie_rating_df, by = "movieId") %>%
        left_join(avg_user_rating_df, by = "userId") %>%
        group_by(genres) %>%
        summarize(avg_genres_rating = sum(rating - y_pred_model1 -
            avg_movie_rating - avg_user_rating)/(n() + x))
    y_pred_model5 <- test_set %>%
        left_join(avg_movie_rating_df, by = "movieId") %>%
        left_join(avg_user_rating_df, by = "userId") %>%
        left_join(avg_genres_rating_df, by = "genres") %>%
        mutate(pred = y_pred_model1 + avg_movie_rating + avg_user_rating +
            avg_genres_rating) %>%
        pull(pred)
    return(RMSE(y_pred_model5, test_set$rating))
})

# Find the lambda that returns the lowest RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5
```

```
# Print the results
model5_rmse = min(rmses)
results <- bind_rows(results, tibble(Method = "Model 5: Regularized Movie, User & Genres rating pattern
results %>%
  kbl(caption="MODEL PERFORMANCE RESULTS", digits=4) %>%
  kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 5: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598
Model 2: Average + Movie rating pattern bias	0.9424
Model 3: Average + Movie & User rating pattern bias	0.8645
Model 4: Average + Movie, User & Genres rating pattern bias	0.8642
Model 5: Regularized Movie, User & Genres rating pattern bias	0.8636

Model 6 - Matrix Factorization using Recosystem

'Recosystem' is an R package for fast matrix factorization. Unlike most other R packages for statistical modeling that store the whole dataset and model object in memory, recosystem significantly reduces memory use, for instance, the constructed model that contains information for the prediction can be stored in the hard disk, and the output result can also be directly written into a file rather than be kept in memory. You can find more details on the recosystem here. Let us start by converting data into the recosystem format, finding the best tuning parameters, training, and finally testing it.

```
if (!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")

set.seed(123, sample.kind = "Rounding")

# Convert the train and test sets to recosystem input
# format
train_data <- with(train_set, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
test_data <- with(test_set, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
# Create the model object
r <- recosystem::Reco()

# Select the best tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30), lrate = c(0.1,
    0.2), costp_l2 = c(0.01, 0.1), costq_l2 = c(0.01, 0.1), nthread = 4,
    niter = 10))

# Train the model
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9825   1.1050e+07
##    1       0.8754   8.9875e+06
##    2       0.8424   8.3430e+06
##    3       0.8202   7.9515e+06
##    4       0.8039   7.6858e+06
##    5       0.7916   7.4963e+06
##    6       0.7813   7.3493e+06
##    7       0.7727   7.2360e+06
##    8       0.7652   7.1397e+06
##    9       0.7588   7.0609e+06
##   10       0.7532   6.9943e+06
##   11       0.7482   6.9374e+06
##   12       0.7438   6.8872e+06
##   13       0.7397   6.8452e+06
##   14       0.7360   6.8066e+06
##   15       0.7326   6.7715e+06
##   16       0.7295   6.7413e+06
##   17       0.7266   6.7124e+06
##   18       0.7240   6.6870e+06
##   19       0.7216   6.6666e+06
```

```
# Predicted ratings in the testing dataset
y_pred_model6 <- r$predict(test_data, out_memory())

# Measure model performance by finding RMSE
model6_rmse <- sqrt(mean((y_pred_model6 - test_set$rating)^2))
```

```
# Print the results
results <- bind_rows(results, tibble(Method = "Model 5: Matrix Factorization using recosystem", RMSE = 
results %>%
  kbl(caption="MODEL PERFORMANCE RESULTS", digits=4) %>%
  kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 6: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598
Model 2: Average + Movie rating pattern bias	0.9424
Model 3: Average + Movie & User rating pattern bias	0.8645
Model 4: Average + Movie, User & Genres rating pattern bias	0.8642
Model 5: Regularized Movie, User & Genres rating pattern bias	0.8636
Model 5: Matrix Factorization using recosystem	0.7860

Results

RMSE of Model 6 reached below our target of 0.8649. Let us train model 6 using the edx dataset and test its accuracy on the final hold-out validation dataset.

```
set.seed(123, sample.kind = "Rounding")

# Convert the edx and validation sets to recosystem input
# format
edx_train_data <- with(edx, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
validation_test_data <- with(validation, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
# Create the model object
r <- recosystem::Reco()

# Select the best tuning parameters
opts <- r$tune(edx_train_data, opts = list(dim = c(10, 20, 30),
    lrate = c(0.1, 0.2), costp_l2 = c(0.01, 0.1), costq_l2 = c(0.01,
        0.1), nthread = 4, niter = 10))

# Train the model
r$train(edx_train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9727   1.2030e+07
##    1       0.8718   9.8627e+06
##    2       0.8390   9.1666e+06
##    3       0.8181   8.7640e+06
##    4       0.8023   8.4800e+06
##    5       0.7902   8.2819e+06
##    6       0.7804   8.1286e+06
##    7       0.7723   8.0108e+06
##    8       0.7653   7.9120e+06
##    9       0.7594   7.8302e+06
##   10       0.7541   7.7631e+06
##   11       0.7495   7.7057e+06
##   12       0.7453   7.6544e+06
##   13       0.7414   7.6089e+06
##   14       0.7379   7.5692e+06
##   15       0.7347   7.5351e+06
##   16       0.7317   7.5017e+06
##   17       0.7290   7.4756e+06
##   18       0.7264   7.4464e+06
##   19       0.7242   7.4260e+06
```

```
# Predicted ratings in the validation dataset
final_pred_model7 <- r$predict(validation_test_data, out_memory())

# Measure model performance by finding RMSE
final_model7_rmse <- sqrt(mean((final_pred_model7 - validation$rating)^2))
```

```
# Print the results
results <- bind_rows(results, tibble(Method = "Final Matrix Factorization - Recosystem Model with Hold-(
results %>%
  kbl(caption="MODEL PERFORMANCE RESULTS", digits=4) %>%
  kable_classic_2(full_width = T, latex_options = "hold_position")
```

Table 7: MODEL PERFORMANCE RESULTS

Method	RMSE
Model 1: Average	1.0598
Model 2: Average + Movie rating pattern bias	0.9424
Model 3: Average + Movie & User rating pattern bias	0.8645
Model 4: Average + Movie, User & Genres rating pattern bias	0.8642
Model 5: Regularized Movie, User & Genres rating pattern bias	0.8636
Model 5: Matrix Factorization using recosystem	0.7860
Final Matrix Factorization - Recosystem Model with Hold-Out Validation Test Set	0.7832

Conclusion

We started with a simple model and kept improving the model further until our model reached the target accuracy. Our best model uses the Recosystem package for Matrix Factorization with RMSE 0.7832073, significantly less than our target of 0.8649.

Limitations

Most modern recommendation models use many predictors. We used only three predictors, the movie, user, and genre information ignoring other features. This model would work only for existing users and films and not provide recommendations for new users or movies.

Future Work

There are other recommendation system packages like Lab for Developing and Testing Recommender Algorithms(recommenderlab), Smart Adaptive Recommendations(SAR), Sparse Linear Method to Predict Ratings and Top-N Recommendations(slimrec), Collective Matrix Factorization for Recommender Systems (cmfrec), A Group-Specific Recommendation System (gsrs) available in CRAN that we could try to further generalize and improve the model in the future.

Acknowledgments

I want to thank the instructor, Rafael A. Irizarry of HarvardX's Professional Certificate in Data Science, for the detailed and clear explanation in his lectures throughout the series. I would also like to thank and appreciate the staff and peers for providing their support, help, and guidance in the discussion forums.

References

1. Rafael A. Irizarry, Introduction to Data Science: Data Analysis and Prediction Algorithms with R
2. Yixuan Qiu (2017), recosystem: recommendation System Using Parallel Matrix Factorization