MovieLens Recommendation System

Loren Grooms

05/18/2021

Introduction

MovieLens.org is a website created by GroupLens Research containing over 25 million ratings for thousands of movies by a multitude of online users. While this wealth of opinion data is certainly useful, a large database of raw information may not be of immediate use to the average user. How can we present this data to users in a simple, actionable manner? Using a subset of the MovieLens dataset, this project aims to form a movie recommendation algorithm that will help users find similar movies to ones they enjoy. To create this algorithm, we will explore the data, find factors that influence a movie's rating, and adjust our method accordingly. Once we have developed a suitable model, we will test its accuracy and analyze strategies for further fine-tuning.

Method

1 - Data Inspection & Cleaning

We will begin our analysis by first downloading and inspecting a 10 million-entry subset of MovieLens's online database from the GroupLens website. This zipped folder contains a **README.html** file to get us started, which lets us know the format for the data tables included in the folder.

ratings.dat - UserID::MovieID::Rating::Timestamp
movies.dat - MovieID::Title::Genres
tags.dat - UserID::MovieID::Tag::Timestamp

For this analysis we will be using **ratings.dat** and **movies.dat**, so we'll download the zipped folder using R then extract, clean, and label the data.

```r
# Download and store zipped file.
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

# Remove delimiters using a utility for processing strings, label
# columns appropriately.
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
tags <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/tags.dat")), "\\::", 4)
colnames(tags) <- c("userId", "movieId", "tag", "timestamp")

# Convert to data frames after converting strings to appropriate data types.
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
tags <- as.data.frame(tags) %>% mutate(userId = as.numeric(userId),
                                       movieId = as.numeric(movieId),
                                       tag = as.character(tag),
                                        timestamp = as.numeric(timestamp)) %>%
                                        select(tag,timestamp)

# Join tables into primary dataset.
movielens <- left_join(ratings, tags, by = "timestamp")
movielens <- left_join(movielens, movies, by = "movieId")
```

```
##    userId movieId rating  timestamp                       tag
## 1:  29722      50    4.5 1199787732                'H Christ!"
## 2:  21562    6679    4.0 1196503294 'Heads Or Tails...Call It'
## 3:  21672    4963    4.0 1196503294 'Heads Or Tails...Call It'
## 4:  23746    1931    4.0 1170638558                       007
## 5:  48585    1961    4.5 1152540548                       007
## 6:  53296    2916    2.0 1207766987                       007
##                          title                        genres
## 1:  Usual Suspects, The (1995)          Crime|Mystery|Thriller
## 2:        Revolution OS (2001)                   Documentary
## 3:       Ocean's Eleven (2001)           Comedy|Crime|Thriller
## 4: Mutiny on the Bounty (1935)                 Adventure|Drama
## 5:             Rain Man (1988)                          Drama
## 6:         Total Recall (1990) Action|Adventure|Sci-Fi|Thriller
```

```
## Rows:  10000371
```

```
## Columns:  7
```

Our data is now readable and easily usable within R.

2 - Data Exploration, Visualization, and Interpretation

We will now explore our dataset further by visualizing the relationships with graphs. These visuals will indicate which variables are affecting the overall rating so we can account for them to achieve a more accurate final rating.

Fig1

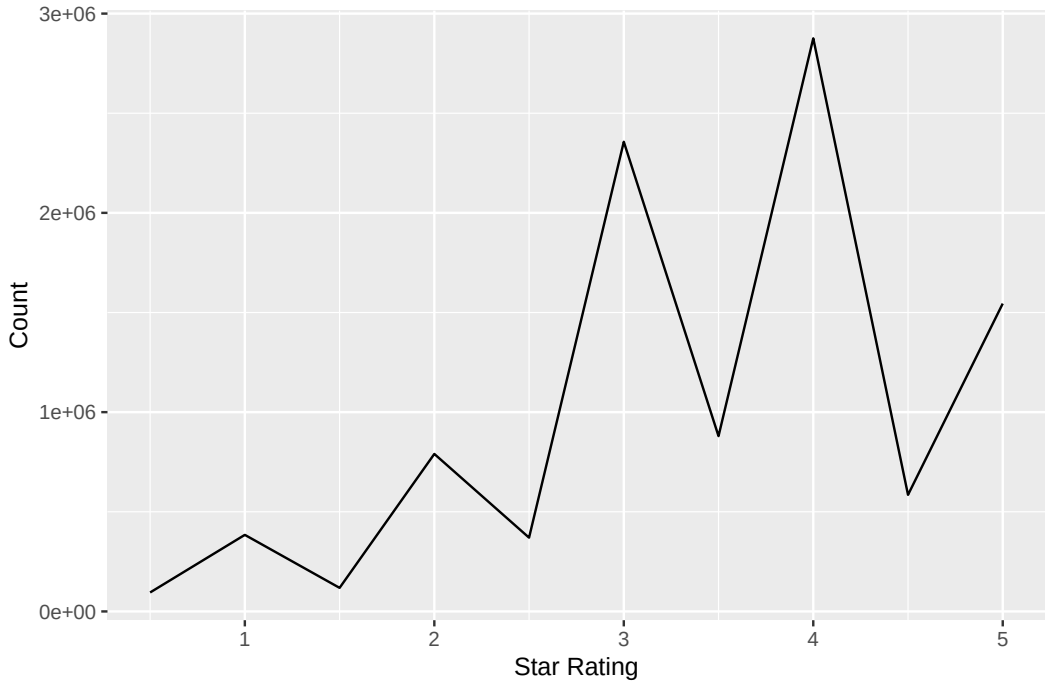


Fig2

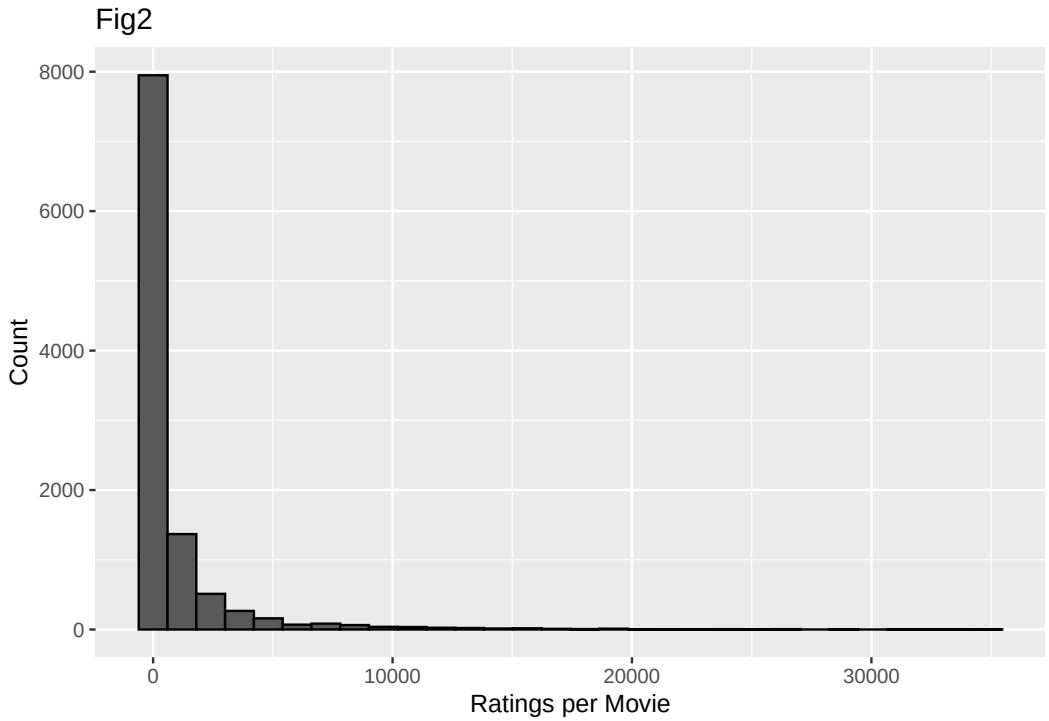


Fig3

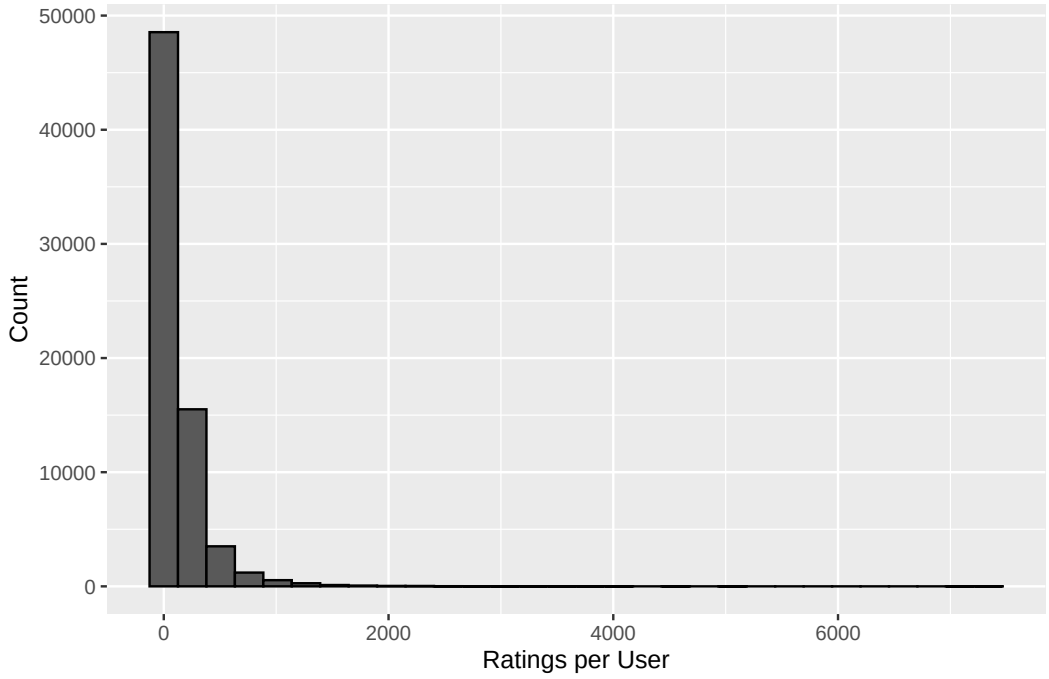
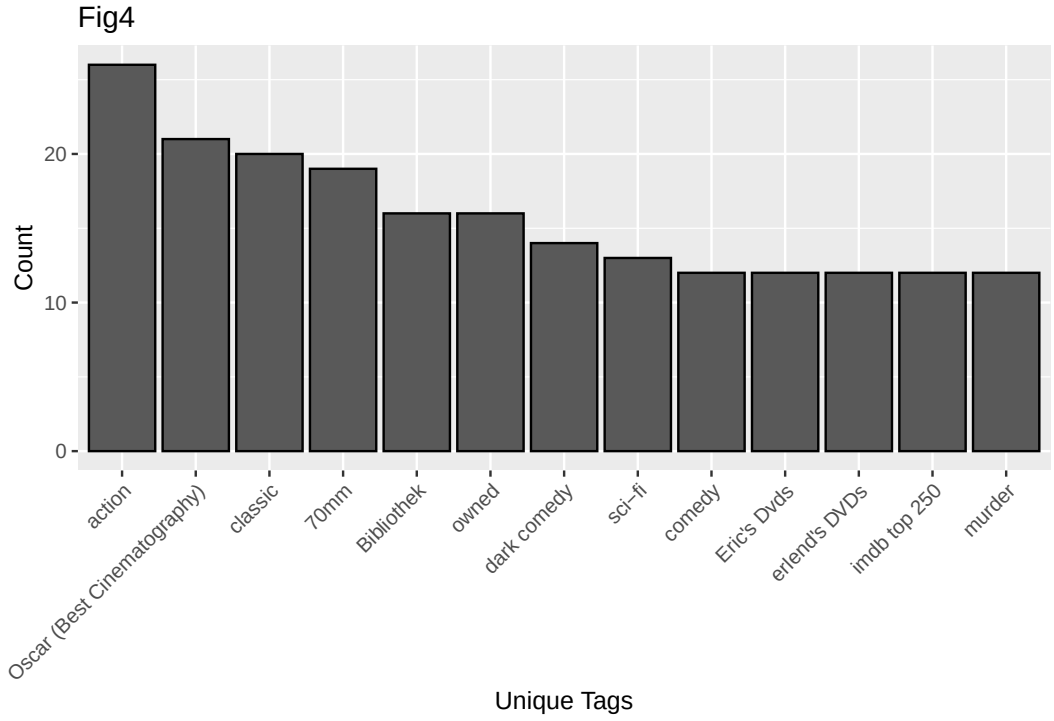

50000
40000
30000
20000
10000
0
Count
0
2000
4000
6000
Ratings per User

Fig4


20
10
0
Count
action
Oscar (Best Cinematography)
classic
70mm
Bibliothek
owned
dark comedy
sci-fi
comedy
Eric's Dvds
erlend's DVDs
imdb top 250
murder
Unique Tags

From these graphs, we can observe the following:

Fig1 - 3- and 4-star ratings are far more common than any others; half-star ratings are far less common than full-star.
Fig2 - Most movies have no rating, and many only a few. The most-rated movie was rated ~1500 times, with the next most-rated having less than half that.
Fig3 - Most users have not submitted a rating, and many only a few. There appears to be one prolific user with over 15,000 reviews, with the next user having submitted less than one third of that.
Fig4 - Tags do not seem to form a consistent pattern. Of the 10 most frequently used tags shown in the graph, even the top tag is only used a little over 20 times total. Additionally, tags do not necessarily relate directly to rating, so we will not be taking tags into account at this time.

We can see that overall ratings, per-movie ratings, and per-user ratings are the major influencers on our data, so we will take them into account when crafting our initial predictive model.

3 - Data Preparation

We will now separate our data into several sets:

Validation - Used at the very end to test the effectiveness our overall algorithm.
Test - Used within the algorithm to test the effectiveness of its components.
Training - Our core data, used as the input to the components of our algorithm.

We want our Validation and Test sets to be 10% of their greater datasets so we can use them to confirm our results reliably without removing too much of the core training data.

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

# Keep only rows which have corresponding rows in edx set.
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows that didn't have a corresponding edx row back into edx set.
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

# Test set will be 10% of remaining edx set, split by rating.
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# Keep only rows which have corresponding rows in training set.
test_set <- test_set %>%
     semi_join(train_set, by = "movieId") %>%
     semi_join(train_set, by = "userId")

# Remove unnecessary variables.
rm(dl, ratings, movies, tags, test_index, temp, movielens, removed)
```

4 - Predictive Model Formation

Our first, most basic predictive model consists of finding the average of each factor and simply adding them together using the following equation:

$$Y_{u,i} = \mu + b_i + b_u$$

$y_{u,i}$ = Predicted rating for movie i from user u.
μ = Overall average rating for all movies.
b_i = Average rating for movie i.
b_u = Average rating from user u.

This is a good start and addresses our main variables, but is not very nuanced and will likely not yeild a very accurate result on its own.

To improve our prediction, we will look at an additional variable our visuals revealed: **sample size**. As we saw previously, the majority of movies and users have low to no reviews, so we want to give more value to large sample sizes and lesser value to small ones. We will accomplish this by **regularizing** our data, which uses the following equation:

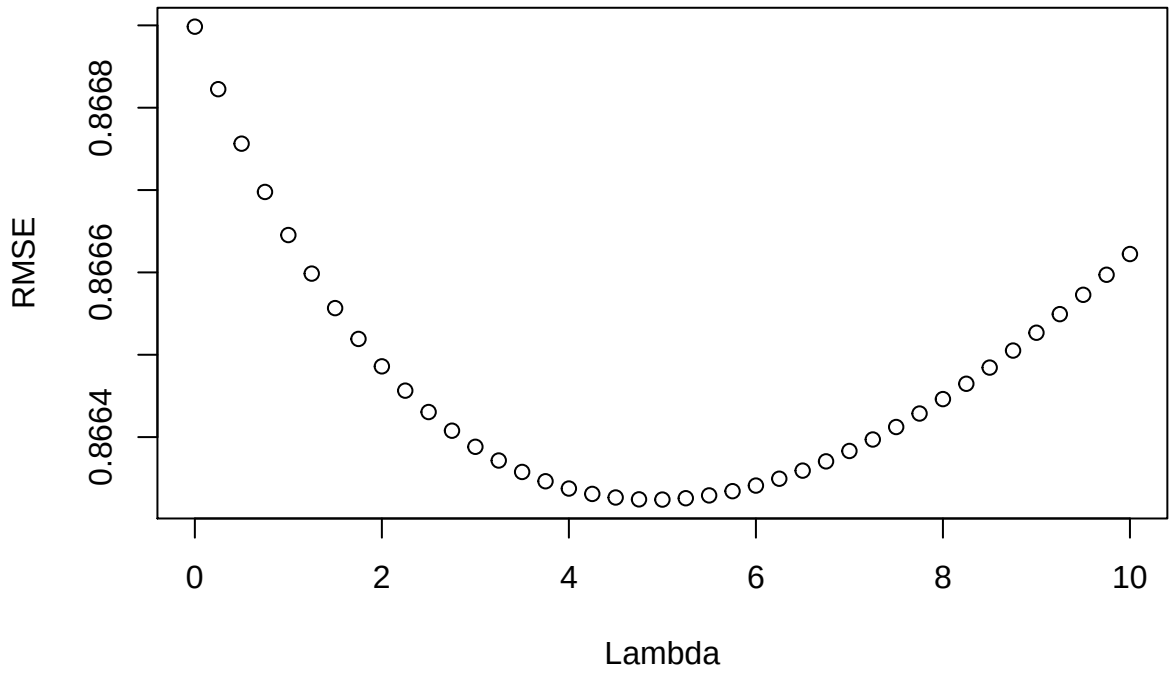
$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

Here, λ is a tuning parameter we can use to adjust our model parametrically, so we will use cross-validation to select the λ which results in the most accurate result. We will do this by testing a sequence of possible λs in our regularization equation, plugging that regularized data into the prediction equation, then testing the accuracy of each prediction compared to the test set. To test the accuracy, we will calculate the Root Mean Square Error using the following equation:

$$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

```r
# Define function to calculate Root Mean Square Error.
RMSE <- function(true_ratings, predicted_ratings){
     sqrt(mean((true_ratings - predicted_ratings)^2))}

# Begin process of regularization by using cross-validation to find the tuning
# parameter (Lambda) that provides the lowest RMSE.
# We will accomplish this by testing a series of possible Lambdas.
tuning_params <- seq(0, 10, 0.25)
test_errors <- sapply(tuning_params, function(l){

   # First we find the overall average rating for all movies (mu).
   mu <- mean(train_set$rating)

   # Next we use our regularization equation to find average movie
   # ratings (b_i).
   regularized_movie_avgs <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+l))

   # Again we use our regularization equation to find the user-specific
   # effect (b_u).
   regularized_user_avgs <- train_set %>%
     left_join(regularized_movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = sum(rating - b_i - mu)/(n()+l))

   # Finally we will use these calculated values to form our predicted ratings.
   predicted_ratings <- test_set %>%
     left_join(regularized_movie_avgs, by='movieId') %>%
     left_join(regularized_user_avgs, by='userId') %>%
     mutate(pred = mu + b_i + b_u) %>%
     .$pred

   # We now test each prediction against the test set and return all calculated
   # RMSE values.
   return(RMSE(predicted_ratings, test_set$rating))
})

# Checking the calculated RMSEs, we select the tuning parameter which resulted
# in the lowest RMSE value and assign it to Lambda.
lambda <- tuning_params[which.min(test_errors)]
```

The results of the cross-validation can be seen in the following graph:



This indicates that our ideal λ resides between 4 and 6, and we can retrieve its exact value with the following code:

```
tuning_params[which.min(test_errors)]
```

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

Now that we have identified the ideal λ, we can apply it to the equation, to the overall set using and calculate the RMSE using the validation set.

```
# Now that we have selected the appropriate tuning parameter to regularize
# our prediction, we can repeat the process on our edx set and compare it
# against the hold-out validation set.
mu <- mean(edx$rating)

regularized_movie_avgs <- edx %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+lambda))

regularized_user_avgs <- edx %>%
     left_join(regularized_movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

predicted_ratings <- validation %>%
     left_join(regularized_movie_avgs, by='movieId') %>%
     left_join(regularized_user_avgs, by='userId') %>%
     mutate(pred = mu + b_i + b_u) %>%
     .$pred

# We now calculate the final RMSE of our model using our predicted value
# from the edx set and the final validation set.
model_rmse <- RMSE(predicted_ratings, validation$rating)
```

Results

After running the final regularization and prediction equations with the ideal λ, we can calculate our RMSE using the validation set:

```
## 0.8650455
```

This tells us that when we apply the predictive model we formed here to the MovieLens data, its predictions are off by an average of **0.86505** stars.

Conclusion

Our results may not seem that impressive when aiming for an idealized 0 star error result, but considering that we took only 4 general variables into account, this is quite a notable outcome. We have observed patterns in our data and accounted for them to the degree that we have a greater chance at prediction than blind guessing, which is a major step in the right direction. To improve our model in the future, we need to analyze more data to uncover other affecting factors, such as user activity level, movie popularity, or release year. We can also utilize the additional data to discover new meaning in our existing data, such as patterns or relations in our Tags dataset. Once further variables and congruences are identified, we can look to various machine learning algorithms and formulas to create appropriate applications and further develop our model.