edX HarvardX - Data Science: MovieLens Report

Jason Katsaros

2023-10-07

Contents

1 Introduction

Movies are an important part of my life. As a tradition to wrap up every work week, every Friday evening, my wife and I order dinner in and watch a movie, a relaxing introduction to the weekend that we get to share together and that I look forward to every week. It is fortuitous then that a capstone to conclude the HarvardX: PH125.9x - Data Science program revolves around movies and movie recommendations.

1.1 Motivation

Recently, the company I currently work for put me in an interesting (albeit awkward) position by announcing that I am out of a job by the end of this year. Not one to mope and stand idly by, I took it upon myself to teach myself something new and prepare myself for the next adventure just over the horizon. I find life at its

most exciting when I am in the midst of a challenge where I can push through and come out better at the other end. Thus, I decided to participate in the HarvardX: PH125.9x - Data Science program to hone new skills, including but not limited to the R programming language, data science and analysis, and machine learning.

1.2 Overview

As a part of completing this program, I am required to construct two capstone projects that encompass the knowledge I have gained and practiced. This project is the first of the two. The goal of this capstone project is to leverage the MovieLens("GroupLens" 2021) data set and formulate a machine learning algorithm to produce movie recommendations based on user ratings. More specifically, the algorithm that I produce must target a Root Mean Squared Error result of `0.86490` or less. Root Mean Squared Error (or RMSE for short) is a loss function used to evaluate how well an algorithm predicts outcomes. Creating a model that minimizes the RMSE indicates that the model is making acceptably accurate predictions. Mathematically, RMSE is defined as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

where $\hat{y}$ is a predicted outcome, y is an observed outcome, N is the number of observations made, and i is the current prediction/observation in the summation. We can calculate the RMSE with the following R code:

```
RMSE <- function(observed, prediction) {
  sqrt(mean((observed - prediction)^2))
}
```

The CAReT(Kuhn and Max 2008), or Classification and Regression Training, R package also contains a function to calculate the RMSE:

```
?caret::RMSE()
```

I decided to use the `caret::RMSE()` function because it is tried and true and will ultimately simplify my R code.

Over the course of this capstone, I will split the MovieLens("GroupLens" 2021) data set into train and test sets, iteratively design a model using the train set which I will then use to make predictions on movie recommendations in the test set. I will evaluate the RMSEs of these predictions against the actual ratings in the test set, demonstrating the evolution (and hopeful improvement) of the model at making movie recommendations based on user ratings. I will also conduct some exploratory analysis, integrating seasonal movies into the model as well as investigating the `RecommenderLab`(Hahsler 2022) R package, which was designed around using machine learning to make recommendations. After all of that, I will evaluate the final model against the `final_holdout_test` set to see how well I did at the project.

2 Data and Analysis

2.1 Set Up

Throughout this capstone project I will use a number of R packages, including the previously mentioned `caret`(Kuhn and Max 2008) package. I will use `tidyverse`(Wickham et al. 2019) for data manipulation, `lubridate`(Grolemund and Wickham 2011) for working with dates and times, and `ggplot2`(Wickham 2016) for visualizing data. I will also use the `recommenderlab`(Hahsler 2022) package for some exploratory analysis later on in the capstone.

As a last note, I use Git LFS (Large File Storage)("Git Large File Storage" 2013) to store the various pieces of data in the Github repository for my capstone. As such, before anyone pulls down my code, they need to install the LFS command line extension for Git using the following command.

```
git install lfs
```

2.2 Initial Data

To start off this capstone project, I was given the following R script. This R script downloads the MovieLens("GroupLens" 2021) data set as a compressed file, extracts the movies data and the ratings data from the compressed file, tidys up the movies and ratings data, then finally merges the two data sets together into two large data set called `edx` and `final_holdout_test` respectively. The `edx` data set is the main data set I will use in my analysis during this capstone, reserving the `final_holdout_test` data set to evaluate my complete algorithm at the very end. Here, I extract the movies and ratings data into their own directory as well as save off the finalized `edx` data set as RData to easily grab it later on if need be.

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

# Note: this process could take a couple of minutes

if (!require(renv))
  install.packages("renv", repos = "http://cran.us.r-project.org")
if (!require(tidyverse))
  install.packages(
    "tidyverse",
    repos = "http://cran.us.r-project.org",
    dependencies = TRUE
  )
if (!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if (!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
if (!require(ggplot2))
  install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if (!require(recommenderlab))
  install.packages("recommenderlab", repos = "http://cran.us.r-project.org")

library(renv)
library(tidyverse)
library(caret)
library(lubridate)
library(ggplot2)
library(recommenderlab)

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

options(timeout = 120)

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

# Extract the ratings data
```

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

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

# Read the contents of the "ratings.dat" file into a data frame
ratings <- as.data.frame(
  str_split(
    read_lines(ratings_file),
    fixed("::"),
    simplify = TRUE
  ),
  stringsAsFactors = FALSE
)
# Set the column names for the ratings data frame
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
# Set the data types of the information in the ratings data frame
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# Read the contents of the "movies.dat" file into a data frame
movies <- as.data.frame(
  str_split(
    read_lines(movies_file),
    fixed("::"),
    simplify = TRUE
  ),
  stringsAsFactors = FALSE
)
# Set the column names for the movies data frame
colnames(movies) <- c("movieId", "title", "genres")
# Set the data types of the information in the movies data frame
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# Join the movies data frame to the ratings data frame by "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, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
```

```r
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)

# Save the edx data to an RData file
base::save(edx, file = "rda/edx.rda")

# Save the final_holdout_test data to an RData file
base::save(final_holdout_test, file = "rda/final_holdout_test.rda")
```

Peeking at the `edx` data set, we can see that there are 69,878 users who reviewed 10,677 movies with six different features I can use to make predictions on.

1. The unique identifier of a user who made a movie review with MovieLens("GroupLens" 2021) (`userId`)
2. The unique identifier of a movie within MovieLens (`movieId`)
3. The rating the user gave the movie (out of 5 stars, `rating`)
4. A time stamp of when the user made the movie review (`timestamp`)
5. The title of the movie the user reviewed (`title`)
6. The genres the movie belongs to within MovieLens("GroupLens" 2021) (separated by `|`, `genres`).

```r
# Visualize the general contents of the edx data set
edx %>% summarize(n_distinct(userId), n_distinct(movieId))
```

```
##   n_distinct(userId) n_distinct(movieId)
## 1              69878               10677
```

```r
head(edx, 5)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

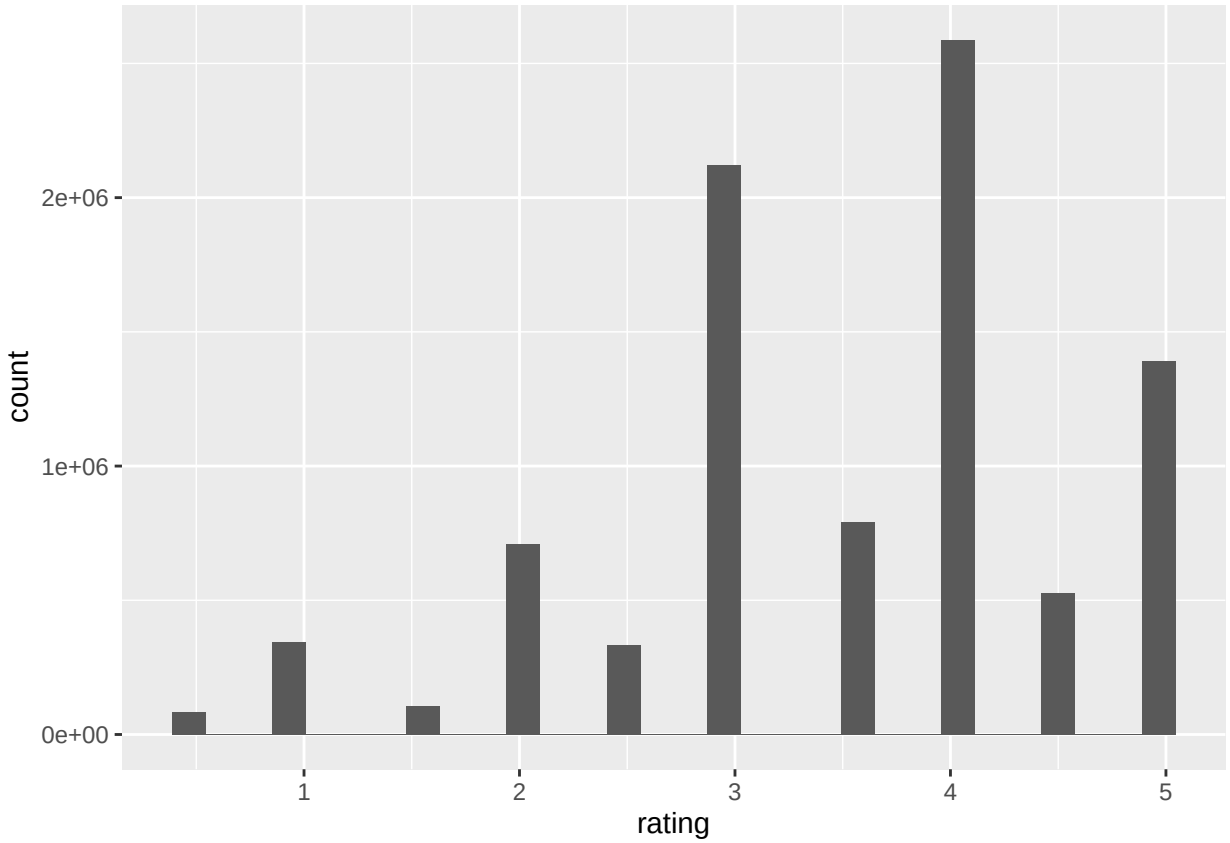
```r
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
```

```
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

I want to note before moving on that the median of all the ratings across all reviews in the MovieLens("GroupLens" 2021) data is greater than the mean of all the ratings. This means that the ratings (out of 5 stars) have a left skew, which can also be represented visually with this plot.

```
# Visualize the distribution of the ratings
edx %>%
  ggplot(aes(rating)) +
  geom_histogram()
```



This information will be useful for analyzing the MovieLens("GroupLens" 2021) data as well as creating and testing my algorithms. First, however, I want to do a little clean up of the data HarvardX has presented me.

2.3 Pre-Processing

Before I continue to analyzing the data, I am first going to perform some pre-processing on the edx data set to put the data in a tidier format, which will make data processing and analysis much easier in the long run.

1. The `timestamp` column is not human-readable. Currently, it is in the Unix time stamp format, which represents some number of seconds from January 1, 1970. Instead, I will convert the time stamp to a date and time.

2. Currently, the year a movie was released is included in the `title` column inside parentheses. Instead, I will separate the actual title and the year a movie was released into two separate columns.
3. The `genres` column is also not in a very human-readable format. Currently, the genres of a particular movie are included in one column separated by the `|` character. Instead, I will move all unique genres out into their own logical columns. If a movie belongs to a specific genre, that value will be `TRUE`, otherwise the value will be `FALSE`.

```
# Get a list of unique genres across all movies
# To use in mutating the edx data set
edx %>% filter(!grepl("[|]", genres)) %>% distinct(genres) %>% arrange(genres)
```

```
##                     genres
## 1025055 (no genres listed)
## 370                 Action
## 1720             Adventure
## 4025             Animation
## 1571              Children
## 16                  Comedy
## 6302                 Crime
## 324            Documentary
## 45                   Drama
## 108652             Fantasy
## 5339             Film-Noir
## 287                 Horror
## 393272                IMAX
## 4639               Musical
## 37304              Mystery
## 669                Romance
## 792                 Sci-Fi
## 204               Thriller
## 8636                   War
## 283                Western
```

```
# Clean up the edx data set before splitting it out
# into separate train and test sets
edx <- edx %>% mutate(
  date_reviewed = as_datetime(timestamp), # Convert the Unix time stamp to a date/time
  year_released = str_sub(title, -5, -2), # Extract the release year from the title of the movie
  title = str_sub(title, 0, -8), # Exclude the release year from the title of the movie
  action = grepl("Action", genres, ignore.case = TRUE), # Is this an "Action" movie?
  adventure = grepl("Adventure", genres, ignore.case = TRUE), # Is this an "Adventure" movie?
  children = grepl("Children", genres, ignore.case = TRUE), # Is this a "Children" movie?
  comedy = grepl("Comedy", genres, ignore.case = TRUE), # Is this a "Comedy" movie?
  crime = grepl("Crime", genres, ignore.case = TRUE), # Is this a "Crime" movie?
  documentary = grepl("Documentary", genres, ignore.case = TRUE), # Is this a "Documentary" movie?
  drama = grepl("Drama", genres, ignore.case = TRUE), # Is this a "Drama" movie?
  fantasy = grepl("Fantasy", genres, ignore.case = TRUE), # Is this a "Fantasy" movie?
  film_noir = grepl("Film-Noir", genres, ignore.case = TRUE), # Is this a "Film-Noir" movie?
  horror = grepl("Horror", genres, ignore.case = TRUE), # Is this a "Horror" movie?
  imax = grepl("IMAX", genres, ignore.case = TRUE), # Is this an "IMAX" movie?
  musical = grepl("Musical", genres, ignore.case = TRUE), # Is this a "Musical" movie?
  mystery = grepl("Mystery", genres, ignore.case = TRUE), # Is this a "Mystery" movie?
  romance = grepl("Romance", genres, ignore.case = TRUE), # Is this a "Romance" movie?
  sci_fi = grepl("Sci-Fi", genres, ignore.case = TRUE), # Is this a "Sci-Fi" movie?
  thriller = grepl("Thriller", genres, ignore.case = TRUE), # Is this a "Thriller" movie?
```

```
  war = grepl("War", genres, ignore.case = TRUE), # Is this a "War" movie?
  western = grepl("Western", genres, ignore.case = TRUE) # Is this a "Western" movie?
) %>%
  select(-c(timestamp, genres)) # Exclude "dirty" columns in favor of new columns
```

Currently, I have the data split out into the `edx` data set and a `final_holdout_test` set. However, to reserve the integrity of the `final_holdout_test` data set, I will split the `edx` data set out even further into another training and test set. While splitting the `edx` data set, I make sure to only include movies and users in the `test_set` that are also included in the `train_set` to avoid any potential difficulties in the future. I use the `caret`(Kuhn and Max 2008) package here to split the `edx` data set into train and test sets, placing 30% of my rows in the `train_set` and the remaining 70% in the `test_set`.

```
# Create training and testing partitions of the edx data
# These sets are separate from the final_holdout_test 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(edx$rating, times = 1, p = 0.7, list = FALSE)
train_set <- edx %>% slice(-test_index)
test_set <- edx %>% slice(test_index)

# Join the test set with the training set by both "movieId" and "userId"
# to ensure that the test set does not include movies and/or users that
# do not exist in the train set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

3 Model Testing and Results

After pre-processing the `edx` data set and splitting into new train and test sets, I am ready to begin creating models that predict movie ratings. Please note that over the iterations of my analysis, instead of using the `caret`(Kuhn and Max 2008) package, I will simply approximate a linear regression model as there is too much data for my personal computer to handle. If I was able to use the `caret`(Kuhn and Max 2008) package to train models and make predictions against those models, it would use the following function.

```
?train(method = "glm")
getModelInfo("glm")
```

The `glm` in this instance stands for "Generalized Linear Model". The "Generalized Linear Model" doesn't have any tuning parameters I could use to tweak the underlying operations the `caret`(Kuhn and Max 2008) package is conducting for me. This means that using the `caret`(Kuhn and Max 2008) package for this capstone particularly taxing on my system. I will go into detail about this limitation later on in this report. For now, though, let us continue with the analysis and building of models.

In this capstone, I will construct the following models and evaluate how well each model makes predictions on movie recommendations within the MovieLens("GroupLens" 2021) data:

1. Naive Model
2. Movie Effects Model
3. Movie and User Effects Model
4. Regularized Movie Effects Model
5. Regularized User Effects Model
6. Combined Regulared Effects Model
7. Regularized Model With Seasonality
8. RecommenderLab(Hahsler 2022) Models

To document how each model compares against the others, I will display a running table of each model's RMSE throughout this report.

3.1 Naive Model

The first (and easiest) model to create is a naive one. A naive model is essentially a model that guesses. This is helpful to create a baseline from which I can improve upon. To create this naive model, instead of guessing, I am going to assume that all users rate movies the same, regardless of factors like genre, how popular a movie is, and so on. To do this, I will simply take the average of all ratings across the `train_set` and evaluate that average against the ratings in the `test_set`.

```
# Naive Model
# Assume that all users rate all movies the same
# Take the average rating across all movies in the train set
naive_model <- mean(train_set$rating)
naive_model
```

```
## [1] 3.512765
```

```
# Compare the movie ratings in the test set against
# the average rating across all movies in the train set
naive_rmse <- caret::RMSE(test_set$rating, naive_model)
naive_rmse
```

```
## [1] 1.060337
```

```
# Print and save the RMSE to a results variable
rmse_results <- data.frame(method = "Naive Model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.060337

```
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

As we can see, the average RMSE is not very good, about `1.06`. I can definitely create a better model, one which takes at least some of the features into account.

3.2 Movie Effects Model

The first feature I would like to predict on is the movie itself. Users are more likely to rate some movies, like Oscar nominated films, higher than others. As such, we need to take the actual rating on the movie into account. To create this movie model, I will take the rating of the movie in the `train_set` and subtract the average rating across all movies in the `train_set`. Then, using this movie model, I can predict the ratings in the `test_set`. Comparing these predictions against the actual ratings in the `test_set` we can see how this model improves upon the naive model.

```
# Movie Effect Model
# Take the average rating across all movies in the train set
average_rating <- mean(train_set$rating)
average_rating
```

```
## [1] 3.512765
```

```r
# Account for individual ratings against the average rating
# across all movies in the train set to create the movie model
movie_model <- train_set %>%
  group_by(movieId) %>%
  summarize(movie_average = mean(rating - average_rating))

# Join the movie model to the test set to predict
# the rating of movies in the test set
movie_predictions <- average_rating + test_set %>%
  left_join(movie_model, by = "movieId") %>%
  pull(movie_average)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
movie_rmse <- caret::RMSE(test_set$rating, movie_predictions)
movie_rmse
```

```
## [1] 0.9449651
```

```r
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Movie Effects Model",
    RMSE = movie_rmse
  )
)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651

```r
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

An RMSE of `0.94` is better than the naive RMSE, as it is closer to `0`. I am on the right track, but there is still work to do.

3.3 Movie and User Effects Model

The next feature I would like to predict on are the users rating movies in MovieLens("GroupLens" 2021). Each user has preferences in movies and rates each movie differently from the next. As such, we need to take how users actually rate movies into account as well. To create this movie and user model, I will take the rating of the movie in the `train_set` and subtract the average rating across all movies in the `train_set` and also subtract the average rating across just that movie in particular from the previous movie model. Then, using this movie and user model, I can predict the ratings in the `test_set`. Comparing these predictions against the actual ratings in the `test_set` we can see how this model still improves upon the previous movie model.

```r
# Movie and User Effect Model
# Join the previously created movie model to the train set by "movieId"
# then find the average rating per user to create the user model
user_model <- train_set %>%
```

```
  left_join(movie_model, by = "movieId") %>%
  group_by(userId) %>%
  summarize(user_average = mean(rating - average_rating - movie_average))

# Join the movie model and the user model to the test set to predict
# the rating of movies in the test set
movie_and_user_predictions <- test_set %>%
  left_join(movie_model, by = "movieId") %>%
  left_join(user_model, by = "userId") %>%
  mutate(movie_and_user_prediction = average_rating + movie_average + user_average) %>%
  pull(movie_and_user_prediction)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
movie_and_user_rmse <- RMSE(test_set$rating, movie_and_user_predictions)
movie_and_user_rmse
```

```
## [1] 0.8760351
```

```
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Movie and User Effects Model",
    RMSE = movie_and_user_rmse
  )
)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351

```
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

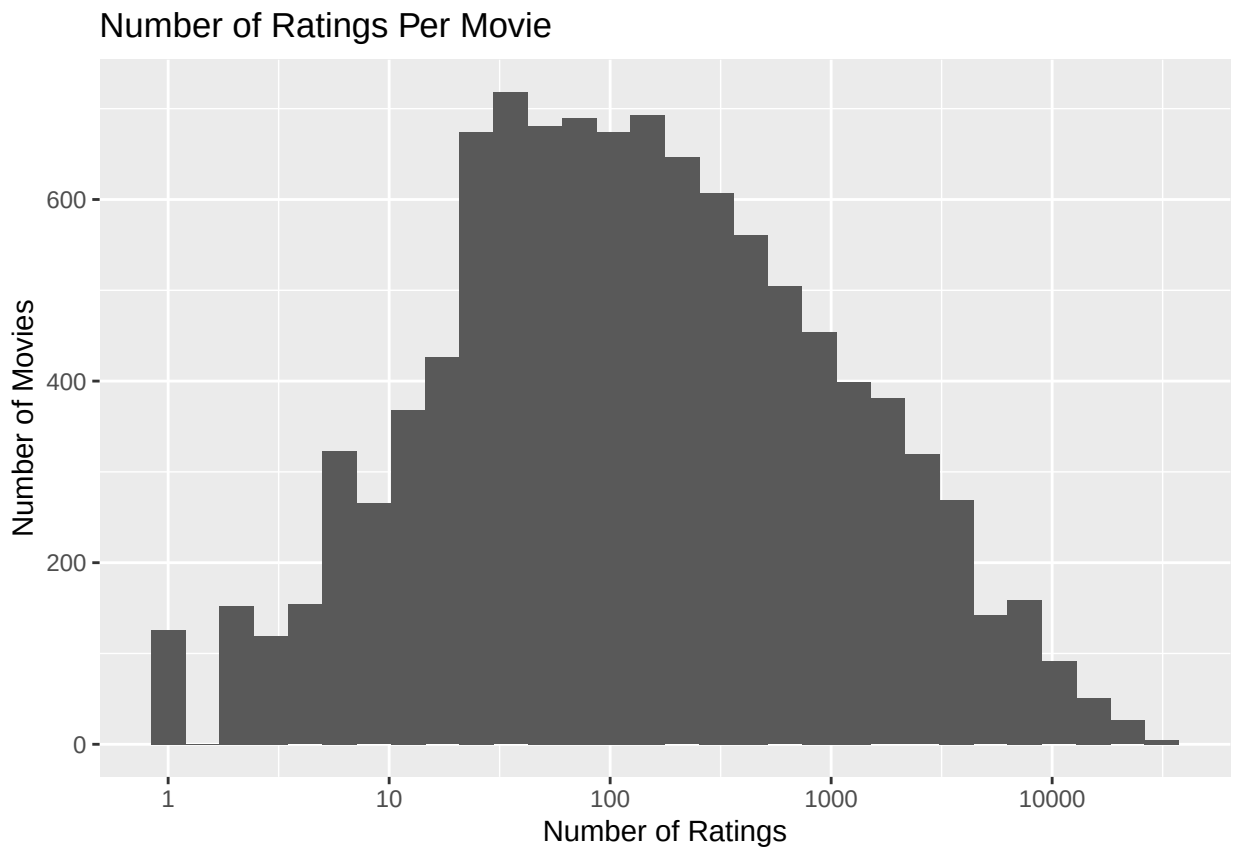
An RMSE of `0.88` is even better than the movie RMSE. With an RMSE target of `0.86490` or less, I am so close to achieving the goal set out by this capstone.

3.4 Regularizing the Models

Since the `edx` data set, and thus the `train_set` and `test_set`, cover a vast number of data points, it is possible that some of the data is skewing the results. We can improve our existing `Movie and User Effect Model` by filtering out any data that might be biasing our data one way or another.

3.4.1 Regularized Movie Effects Model

For example, using the chart below, it is possible to see that some movies in the `edx` data set only have one review.



It is possible to view these movies in a formatted table using the R code seen below.

```
edx %>%
  group_by(movieId) %>%
  summarize(rating_count = n()) %>%
  filter(rating_count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, rating_count = rating_count) %>%
  knitr::kable()
```

This indicates to me that these movies are obscure and will not likely receive very many reviews (if any) from new or existing users. Therefore, it should improve the `Movie and User Effects Model` by removing these obscure movies from the data set.

```
# Regularized Movie Effects Model
# Create the movie model again,
# but this time remove any movie that only has one review
regularized_movie_model <- train_set %>%
  group_by(movieId) %>%
  filter(n() > 1) %>%
  summarize(regularized_movie_average = mean(rating - average_rating))

# Filter the test set of the movies removed previously
filtered_test_set <- test_set %>%
  semi_join(regularized_movie_model, by = "movieId")
```

```r
# Join the regularized movie model to the test set to predict
# the rating of movies in the test set
regularized_movie_predictions <- average_rating + filtered_test_set %>%
  left_join(regularized_movie_model, by = "movieId") %>%
  pull(regularized_movie_average)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
regularized_movie_rmse <- caret::RMSE(filtered_test_set$rating, regularized_movie_predictions)
regularized_movie_rmse
```

```
## [1] 0.9448313
```

```r
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Regularized Movie Effects Model",
    RMSE = regularized_movie_rmse
  )
)
rmse_results %>% knitr::kable()
```

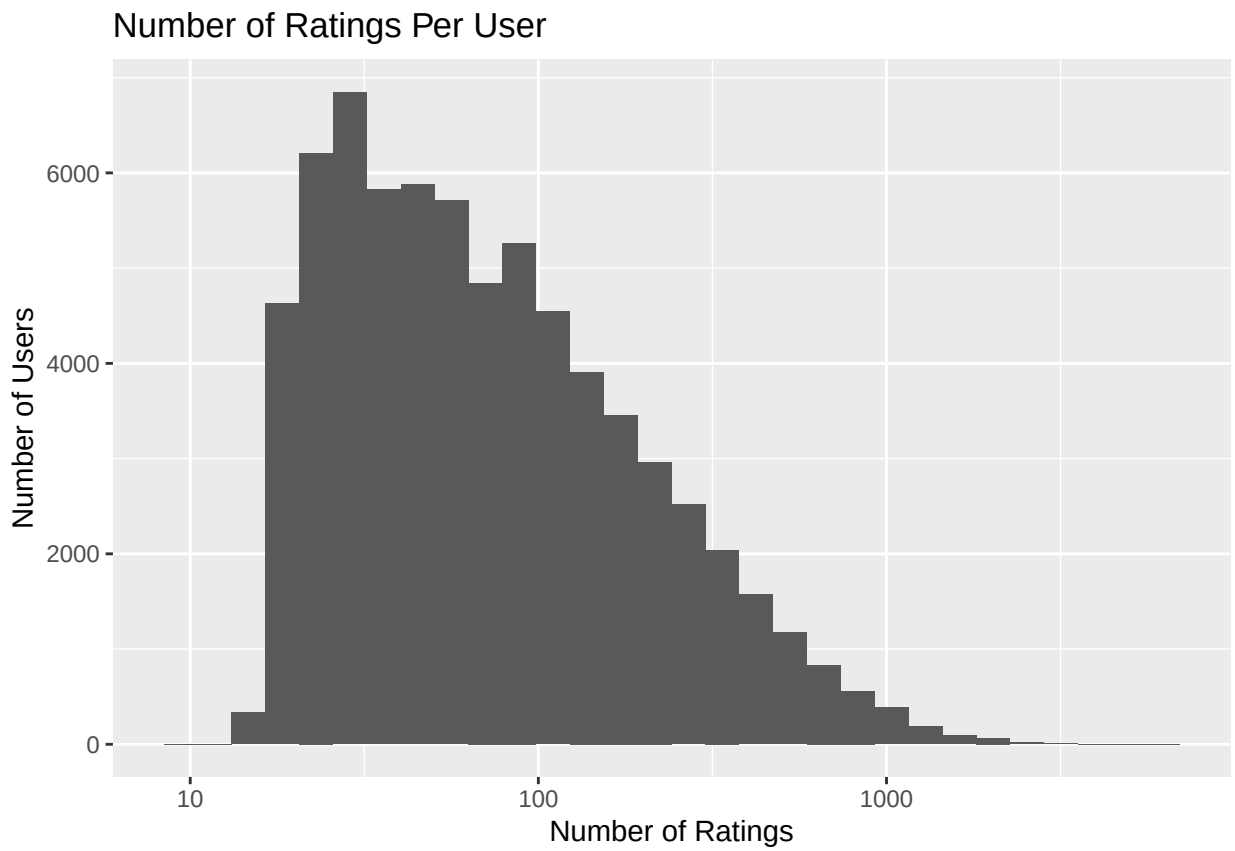
method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351
Regularized Movie Effects Model	0.9448313

```r
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

Comparing this RMSE to the previous `Movie Model` RMSE, we can see that it is an improvement over the previous model, although very slight.

3.4.2 Regularized User Effects Model

Using the chart below, it is possible to see the distribution of reviews each user has created for movies across the `edx` data set.



At the low end, some users have only left less than 100 reviews, whereas at the high end, some users have left over 1000 reviews. Since I take user reviews into account in the `Movie and User Effects Model` it should be helpful to remove users that have less than 100 reviews from the data, as reviews given from users who review movies more often should have a greater weight than reviews from users who review movies less often.

```
# Movie and Regularized User Effects Model
# Create the user model again,
# but this time remove any user with less than 100 movie reviews
regularized_user_model <- train_set %>%
  left_join(movie_model, by = "movieId") %>%
  group_by(userId) %>%
  filter(n() > 100) %>%
  summarize(regularized_user_average = mean(rating - average_rating - movie_average))

# Filter the test set of the users removed previously
filtered_test_set <- test_set %>%
  semi_join(regularized_user_model, by = "userId")

# Join the movie model and the regularized user model to the test set to predict
# the rating of movies in the test set
movie_and_regularized_user_predictions <- filtered_test_set %>%
  left_join(movie_model, by = "movieId") %>%
  left_join(regularized_user_model, by = "userId") %>%
  mutate(
    movie_and_regularized_user_prediction =
      average_rating +
      movie_average +
```

```
      regularized_user_average
  ) %>%
  pull(movie_and_regularized_user_prediction)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
# First we need to filter the test set of the users we removed previously
movie_and_regularized_user_rmse <- caret::RMSE(
  filtered_test_set$rating,
  movie_and_regularized_user_predictions
)
movie_and_regularized_user_rmse
```

```
## [1] 0.8388028
```

```
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Movie and Regularized User Effects Model",
    RMSE = movie_and_regularized_user_rmse
  )
)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351
Regularized Movie Effects Model	0.9448313
Movie and Regularized User Effects Model	0.8388028

```
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

Regularizing the users in our data has a much greater impact on our RMSE, which we will also be able to see once I combine both regularized models into one complete model.

3.4.3 Combined Regularized Model

Taking the two previously created regularized models, we can create one fully regularized model to see the overall improvements to the algorithm.

```
# Regularized Model
# Combine the two regularized models above
# to create one regularized model
# Filter the test set of the movies and users removed previously
regularized_test_set <- test_set %>%
  semi_join(regularized_movie_model, by = "movieId") %>%
  semi_join(regularized_user_model, by = "userId")

# Join the regularized movie model and the regularized user model to the test set to predict
# the rating of movies in the test set
regularized_predictions <- regularized_test_set %>%
```

```
  left_join(regularized_movie_model, by = "movieId") %>%
  left_join(regularized_user_model, by = "userId") %>%
  mutate(
    regularized_prediction =
      average_rating +
      regularized_movie_average +
      regularized_user_average
  ) %>%
  pull(regularized_prediction)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
regularized_rmse <- caret::RMSE(regularized_test_set$rating, regularized_predictions)
regularized_rmse
```

```
## [1] 0.8384961
```

```
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Regularized Model",
    RMSE = regularized_rmse
  )
)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351
Regularized Movie Effects Model	0.9448313
Movie and Regularized User Effects Model	0.8388028
Regularized Model	0.8384961

```
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

Combining both regularized models achieves the best RMSE yet. In fact, with an RMSE of `0.84`, this puts me below the target RMSE for this capstone. I would, however, still like to do some exploratory analysis, both to see if I can improve upon this model by taking more features into account but also to learn about the MovieLens("GroupLens" 2021) data in general.

3.4.4 Exploratory Model Testing and Analysis

3.4.4.1 Regularized With Seasonality Model One avenue I would like to investigate when it comes to movie recommendations is seasonality. For example, I am conducting this capstone project in October, the month where people tend to watch more movies from the Horror genre. Another example is the viewership increase of holiday-related movies in the months of November and December. While the MovieLens("GroupLens" 2021) data set does not have an explicit "Holiday" genre, I do believe this is an idea worth investigating. To prove this, I will conduct a Chi-Squared test of movies reviewed (and not reviewed) in October against Horror movies (and not Horror movies).

```
# Some movies recommendations should be weighted slightly seasonally
# as some users will prefer to watch seasonal movies at appropriate times of the year,
# while other users won't be affected by seasonally appropriate movies for a variety of reasons
# To test this relationship, here is a Chi-Squared test for Horror movies and October review dates
horror_october <- train_set %>%
  filter(month(date_reviewed) == "10" & horror == TRUE)
not_horror_october <- train_set %>%
  filter(month(date_reviewed) == "10" & horror == FALSE)
not_horror_not_october <- train_set %>%
  filter(month(date_reviewed) != "10" & horror == FALSE)
horror_not_october <- train_set %>%
  filter(month(date_reviewed) != "10" & horror == TRUE)

horror_movie_matrix <- matrix(
  c(
    nrow(horror_october),
    nrow(not_horror_october),
    nrow(horror_not_october),
    nrow(not_horror_not_october)
  ),
  ncol = 2
)
colnames(horror_movie_matrix) <- c("October", "Not October")
rownames(horror_movie_matrix) <- c("Horror", "Not Horror")
horror_movie_matrix
```

```
##            October Not October
## Horror       17871      189368
## Not Horror  231223     2261554
```

```
chisq.test(horror_movie_matrix)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  horror_movie_matrix
## X-squared = 97.148, df = 1, p-value < 2.2e-16
```

As evidenced by the Chi-Squared test above, with an incredibly small `p-value`, movies reviewed in October and the Horror movie genre are not independent factors. In order to explore this, I created a very slight weight for movies related to the Horror genre if the current date is within October. I only included a very slight weight here because seasonality is not a preference that reaches all movie viewers, due to culture or personal preference. Again, I would like to conduct this experiment against other movies and genres within the MovieLens("GroupLens" 2021) data set, but the current state of the MovieLens("GroupLens" 2021) data set does not support this. In the future, it would be a great improvement to include some sort of "Holiday" genre or to create a mapping between months of the year and movie genre preference to create some sort of weight to apply to the model.

```
# Regularized With Seasonality Model
# Currently, the only seasonal genre that exists in the MovieLens data set is Horror
# A future enhancement could be to include a Holiday genre
regularized_seasonal_movie_model <- train_set %>%
  group_by(movieId) %>%
  filter(n() > 1) %>%
  summarize(
```

```
    regularized_movie_average =
      mean(rating -
             average_rating -
             ifelse(month(now()) == 10 &
                      horror, 0.125, 0
              )
      )
  )

# Filter the test set of the movies and users removed previously
regularized_seasonal_test_set <- test_set %>%
  semi_join(regularized_seasonal_movie_model, by = "movieId") %>%
  semi_join(regularized_user_model, by = "userId")

# Join the regularized movie model and the regularized user model to the test set to predict
# the rating of movies in the test set
regularized_seasonal_predictions <- regularized_seasonal_test_set %>%
  left_join(regularized_seasonal_movie_model, by = "movieId") %>%
  left_join(regularized_user_model, by = "userId") %>%
  mutate(
    regularized_seasonal_prediction =
      average_rating +
      regularized_movie_average +
      regularized_user_average
  ) %>%
  pull(regularized_seasonal_prediction)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
regularized_seasonal_rmse <- caret::RMSE(
  regularized_seasonal_test_set$rating,
  regularized_seasonal_predictions
)
regularized_seasonal_rmse
```

```
## [1] 0.8396376
```

```
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Regularized With Seasonality Model",
    RMSE = regularized_seasonal_rmse
  )
)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351
Regularized Movie Effects Model	0.9448313
Movie and Regularized User Effects Model	0.8388028
Regularized Model	0.8384961

method	RMSE
Regularized With Seasonality Model	0.8396376

```
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

While this does increase the overall RMSE by a slight amount, I do believe that including more features in the model is useful in the long run. With some fine tuning, I believe the genre and month features will be a great addition to the final model.

3.4.4.2 RecommenderLab Mentioned, but not elaborated upon, in the HarvardX course was the `recommenderlab`(Hahsler 2022) package, whose purpose is to provide a library of R functions useful for creating recommendation systems, which is perfect to explore for this capstone project. While I was not able to use the caret(Kuhn and Max 2008) package over the course of this capstone due to hardware limitations (which I will elaborate on later on in this report), I am able to use the `recommenderlab`(Hahsler 2022) package to create a model, make predictions based on said model, and ultimately evaluate the model using the RMSE loss function.

```
# RecommenderLab
# Convert a subset of the edx data set into a ratings matrix
# (for demonstration purposes)
edx_r <- as(edx[1:100000,], "realRatingMatrix")

# The proportion of items from the edx set to use for a train set
train_proportion <- 0.75

# This determines the number of items that should be given
# for evaluation when building the model
# The "given" parameter for evaluationScheme should not be larger than this value
min(rowCounts(edx_r))
```

```
## [1] 15
```

```
# Define the "given" parameter for evaluationScheme
items_per_test_user_keep <- 10

# Define the threshold for a "good" movie rating
# (4 out of 5 stars is considered a "good" movie rating here)
good_threshold <- 4

set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
# Create the scheme used to train and evaluate the model
model_train_scheme <- edx_r %>%
  evaluationScheme(
    method = "split", # Splits the edx_r matrix into one train set and one test set
    train = train_proportion, # The proportion of the edx_r matrix to put into the train set
    given = items_per_test_user_keep, # The number of items to evaluate when building the model
    goodRating = good_threshold, # What is considered a "good" movie rating?
    k = 1
  )

# Create the parameters used to generate the model
model_parameters <- list(
```

```
  method = "cosine", # Use cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity)
  nn = 10, # Find each user's 1- most similar users in terms of preferences
  sample = FALSE, # The train set and test set are already created, so no need here
  normalize = "center"
)

# Describe User-based Collaborative Filtering
tail(recommenderRegistry$get_entries(dataType = "realRatingMatrix"), 1)
```

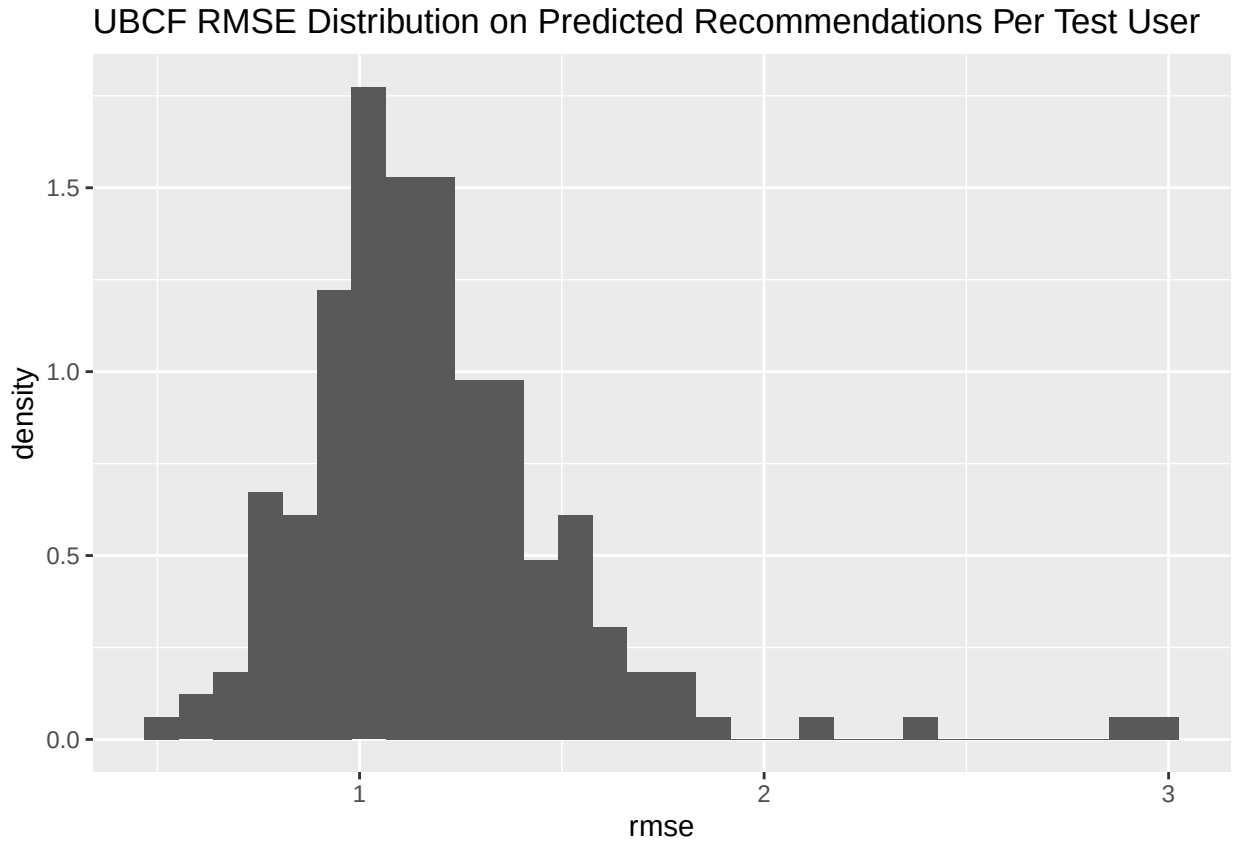
```
## $UBCF_realRatingMatrix
## Recommender method: UBCF for realRatingMatrix Description: Recommender
##   based on user-based collaborative filtering. Reference: NA
## Parameters:
##     method nn sample weighted normalize min_matching_items min_predictive_items
## 1 "cosine" 25  FALSE     TRUE  "center"                  0                     0
```

```
# Create the model
recommenderlab_model <- getData(
  model_train_scheme,
  "train"
) %>%
  Recommender(
    method = "UBCF", # User-based Collaborative Filtering
    parameter = model_parameters
  )

# Make predictions based on the model
recommenderlab_predictions <- recommenderlab::predict(
  recommenderlab_model,
  getData(
    model_train_scheme,
    "known" # Use what is known to predict the unknown portion (test set)
  ),
  type = "ratings"
)

# Evaluate the accuracy of the model
recommenderlab_error <- calcPredictionAccuracy(
  recommenderlab_predictions,
  getData(
    model_train_scheme,
    "unknown" # Use what is unknown to evaluate the accuracy of the model
  ),
  byUser = TRUE
)
```

The above R code I use the `User-based Collaborative Filtering` algorithm provided through the `recommenderlab`(Hahsler 2022) package. `User-based Collaborative Filtering` is a technique that attempts to predict an item (in this case, a movie) that a test user might like based on a ratings system (in this case, the five star system) where other users have already given said items ratings. In `User-based Collaborative Filtering` models, the algorithm tries to find users that like similar items, then create a prediction based on those groupings. In the R code above, I designated four out of five stars to be a good rating for a movie, which `recommenderlab`(Hahsler 2022) will take into account when creating and evaluating its different models. `recommenderlab`(Hahsler 2022) will automatically construct multiple models with varying groups of similar users to make predictions on, which is why there are several lines of output

below. The first column in the output is the number of similar grouped users. The RMSEs achieved by each of these models are shown below.

```
##         RMSE       MSE       MAE
## 2        NaN       NaN       NaN
## 4  1.3069710 1.7081731 1.1484198
## 6  0.8213936 0.6746874 0.6551014
## 7  1.0605297 1.1247232 0.8409920
## 14 1.1470466 1.3157160 0.9492431
## 22 1.0188993 1.0381558 0.7658330
```



As demonstrated by the plot above, it is possible to achieve a great RMSE with a model for movie recommendations using the `recommenderlab`(Hahsler 2022) package, and it is definitely worth investigating for anyone pursuing a recommendation problem.

4 Conclusion

After all the iterations on the algorithm, here is where the RMSEs stand.

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351
Regularized Movie Effects Model	0.9448313
Movie and Regularized User Effects Model	0.8388028
Regularized Model	0.8384961

method	RMSE
Regularized With Seasonality Model	0.8396376

It is now time to use the model I constructed over the course of this capstone against the `final_holdout_test` data set.

```
# Load in the edx data
base::load("rda/final_holdout_test.rda")

# Filter the final holdout test set
# just like the regular test set
filtered_final_holdout_test <- final_holdout_test %>%
  semi_join(regularized_movie_model, by = "movieId") %>%
  semi_join(regularized_user_model, by = "userId")

# Join the regularized movie model
# and the regularized user model
# to the final holdout test set to predict
# the rating of movies in the final holdout test set
final_predictions <- filtered_final_holdout_test %>%
  left_join(regularized_movie_model, by = "movieId") %>%
  left_join(regularized_user_model, by = "userId") %>%
  mutate(
    final_prediction =
      average_rating +
      regularized_movie_average +
      regularized_user_average
  ) %>%
  pull(final_prediction)

# Compare the predicted ratings in the test set
# against the actual ratings in the test set
final_rmse <- caret::RMSE(filtered_final_holdout_test$rating, final_predictions)
final_rmse
```

```
## [1] 0.8399653
```

```
# Print and save the RMSE to a results variable
rmse_results <- bind_rows(
  rmse_results,
  data.frame(
    method = "Final Model",
    RMSE = final_rmse
  )
)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0603369
Movie Effects Model	0.9449651
Movie and User Effects Model	0.8760351
Regularized Movie Effects Model	0.9448313
Movie and Regularized User Effects Model	0.8388028

method	RMSE
Regularized Model	0.8384961
Regularized With Seasonality Model	0.8396376
Final Model	0.8399653

```
# Save the results to a RData file
base::save(rmse_results, file = "rda/rmse_results.rda")
```

It looks like this algorithm is good to go. From a naive model to a fully fledged regression, the final model achieves the goal of this capstone project.

4.1 Limitations

As discussed previously, I was unable to utilize the `caret`(Kuhn and Max 2008) package at all as well as unable to use the `recommenderlab`(Hahsler 2022) package to its fullest due to my limited hardware. The `recommenderlab`(Hahsler 2022) issue is easy enough to explain away, though I was able to leverage the `recommenderlab`(Hahsler 2022) package against a subset of the `edx` data set at the end of the day. As far as the `caret`(Kuhn and Max 2008) package is concerned however, the reason behind why I was unable to make use of the `caret`(Kuhn and Max 2008) package is because of what the `caret`(Kuhn and Max 2008) package is doing behind the scenes. The `train()` function in the `caret`(Kuhn and Max 2008) package makes use of resampling automatically, which drastically increases the amount of system memory required for most training operations. Additionally, the `train()` function includes a number of default parameters that increase the system memory requirement substantially, namely `model = TRUE` and `trim = FALSE`. Reversing these default settings does improve memory quite a bit, but not enough for the `train()` function to properly run on my machine.("StackOverflow" 2008)

4.2 Future Considerations

In the future, I would love to use the `caret`(Kuhn and Max 2008) package to see how well my model would hold up against different algorithms, not just linear regression. I would also love to see a "Holiday" genre introduced to the MovieLens("GroupLens" 2021) data set so that I could investigate seasonality more fully. Lastly, using either of the date features, whether its the year released or the date of the review, in the final model would be an interesting experiment to see how that changes the overall RMSEs.

4.3 Final Remarks

Thanks to all who read through this report and look through my repository. I look forward to applying my newly gained machine learning knowledge on future projects, and hope to continue expanding that knowledge in this ever-growing field.

References

"Git Large File Storage." 2013. https://git-lfs.com/.

Grolemund, Garrett, and Hadley Wickham. 2011. "Dates and Times Made Easy with lubridate." *Journal of Statistical Software* 40 (3): 1–25. https://www.jstatsoft.org/v40/i03/.

"GroupLens." 2021. http://grouplens.org/datasets/movielens/10m/.

Hahsler, Michael. 2022. "Recommenderlab: An r Framework for Developing and Testing Recommendation Algorithms." arXiv:2205.12371 [cs.IR]. https://doi.org/10.48550/ARXIV.2205.12371.

Kuhn, and Max. 2008. "Building Predictive Models in r Using the Caret Package." *Journal of Statistical Software* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.

"StackOverflow." 2008. https://stackoverflow.com/questions/6543999/why-is-caret-train-taking-up-so-much-memory.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemun, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.