Movielens Project

Sarthak Pant

1/8/2021

Introduction

The goal of a movie recommendation system is to recommend movies based on previous ratings and through user taste/preference through a prediction model. A real world example of this application are streaming services such as Netflix, Hulu, and Amazon Prime which recommend movies they believe the user would prefer based on the parameters previously mentioned. This report explains the process of constructing a movie recommendation in R using the material that was taught in the Harvard Edx Data Science course.

The primary dataset used for this project was the Movielens dataset which is a public dataset that can be found on grouplens. The Movielens data has 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. For the purposes of this project a subset of the Movielens dataset was created which consisted of 9,000,055 ratings of 10677 movies which were given by 69878 users. This subset was stored under the object edx.

Project Goal

The goal of this project was to train an algorithm, using machine learning, to predict user ratings based on the data from the edx subset. In order to judge the accuracy of the prediction model, a loss function was used to measure the difference between the predicted and actual values. In order to get the specific measure of accuracy the root mean squared error (RMSE) was calculated. The RMSE is one of the most widely used metrics in order to judge the performance of a machine learning algorithm and the lower the RMSE value, the more accurate the model is. The aim for this project is to achieve an RMSE value less than 0.86490.

Prior to developing the model the data was prepared, cleaned, and analysed in order to find some insights and patterns that would be useful during the development of the model. We downloaded the Movielens data set and split it into a training set called edx, and a testing set called validation. The validation set is only to be used to determine the accuracy of the final model which is why the edx dataset is split into two more subsets with the training set called training, and a testing set called testing. Following this step we examined the data by creating summaries in order to understand how certain parameters could have an effect on the prediction model.

Loading Data

```
knitr::opts_chunk$set(echo = FALSE)
##########################################################
# Create edx set, validation set (final hold-out test set)
##########################################################

# Note: this process could take a couple of minutes

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

## Loading required package: tidyverse

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.
3.0 ──

## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.4     ✓ dplyr   1.0.2
## ✓ tidyr   1.1.2     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.0

## ── Conflicts ────────────────────────────────────────── tidyverse_conflict
s() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

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

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

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

## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
library(tidyverse)
library(caret)
library(data.table)

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/rati
ngs.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")


# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),titl
e = as.character(title),genres = as.character(genres))


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

# Validation set will be 10% of MovieLens data
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, l
ist = 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 validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genr
es")
```

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

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


#Split the edx set into Training and Test sets#
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list =
FALSE)
training <- edx[-test_index,]
temp <- edx[test_index,]

#Add movieId, userId, and genres are in the training set#
testing <- temp %>%
  semi_join(training, by = "movieId") %>%
  semi_join(training, by = "userId")


# Add rows removed from test set back into train set
removed <- anti_join(temp, testing)

## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genr
es")

training <- rbind(training, removed)

rm(test_index, temp, removed)
```

Data OverView

In order to familiarize ourselves with the data we look at the row headings of the dataset. This introduces us to the six variables of the data set which are movieId, userId, ratings, timestamp, title, and genres. While we explored all six of these variables the three variables used in our prediction model were movieId, userId, and genres.

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 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
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi

##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 10,677 x 2
##    movieId     n
##      <dbl> <int>
##  1       1 23826
##  2       2 10717
##  3       3  7053
##  4       4  1579
##  5       5  6415
##  6       6 12385
##  7       7  7273
##  8       8   811
##  9       9  2280
## 10      10 15250
## # … with 10,667 more rows

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 69,878 x 2
##    userId     n
##     <int> <int>
##  1      1    21
##  2      2    18
##  3      3    30
##  4      4    35
##  5      5    80
##  6      6    39
##  7      7   101
##  8      8   713
##  9      9    20
## 10     10   110
## # … with 69,868 more rows
## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 797 x 2
##    genres                                             n
##    <chr>                                          <int>
##  1 (no genres listed)                                 6
##  2 Action                                         24575
##  3 Action|Adventure                               68611
##  4 Action|Adventure|Animation|Children|Comedy      7438
##  5 Action|Adventure|Animation|Children|Comedy|Fantasy   191
##  6 Action|Adventure|Animation|Children|Comedy|IMAX      62
##  7 Action|Adventure|Animation|Children|Comedy|Sci-Fi   600
##  8 Action|Adventure|Animation|Children|Fantasy      743
##  9 Action|Adventure|Animation|Children|Sci-Fi        51
## 10 Action|Adventure|Animation|Comedy|Drama         1896
## # … with 787 more rows
```

Results

Initial Prediction

We constructed our initial model by simply taking the mean of the edx dataset

```
## [1] 3.512377

## [1] 1.06058
```

Movie Effect

The RMSE from our initial model is far from our goal of 0.86490. In order to improve the accuracy of our model we introduce the movie effect. It is a given that some movies are rated higher than others based on factors such as popularity. In order to mitigate this issue we introduced the movie effect stored under the object movieEffect. Running the model with this effect we achieve an RMSE value of 0.9431724.

```
## `summarise()` ungrouping output (override with `.groups` argument)

## [1] 0.9431724
```

Movie + User Effect

There is always going to be variability in the ratings given by the users, based on tastes and preferences, since there is no rational way to rate a movie. In order to account for this variability and increase accuracy we introduce the user effect into the previous model, which is stored under the object userEffect. Running the model with this effect we achieve an RMSE value of 0.8655154. This RMSE value is much closer to our goal for this project

```
## `summarise()` ungrouping output (override with `.groups` argument)
## [1] 0.8655154
```

Movie + User + Genre Effect

The last effect that we are adding to our model is the genre effect. The genres are not evenly distributed in the dataset (for example there are more dramas than comedy). In order to mitigate this issue and increase the accuracy of our model we introduce the genre effect which is stored under the object genreEffect. Running the model with this effect we achieve an RMSE value of 0.8651674. This RMSE value is slightly lower than the Movie + User effect model but is still higher than our goal of achieving an RMSE of 0.86490.

```
## `summarise()` ungrouping output (override with `.groups` argument)

## [1] 0.8651674
```

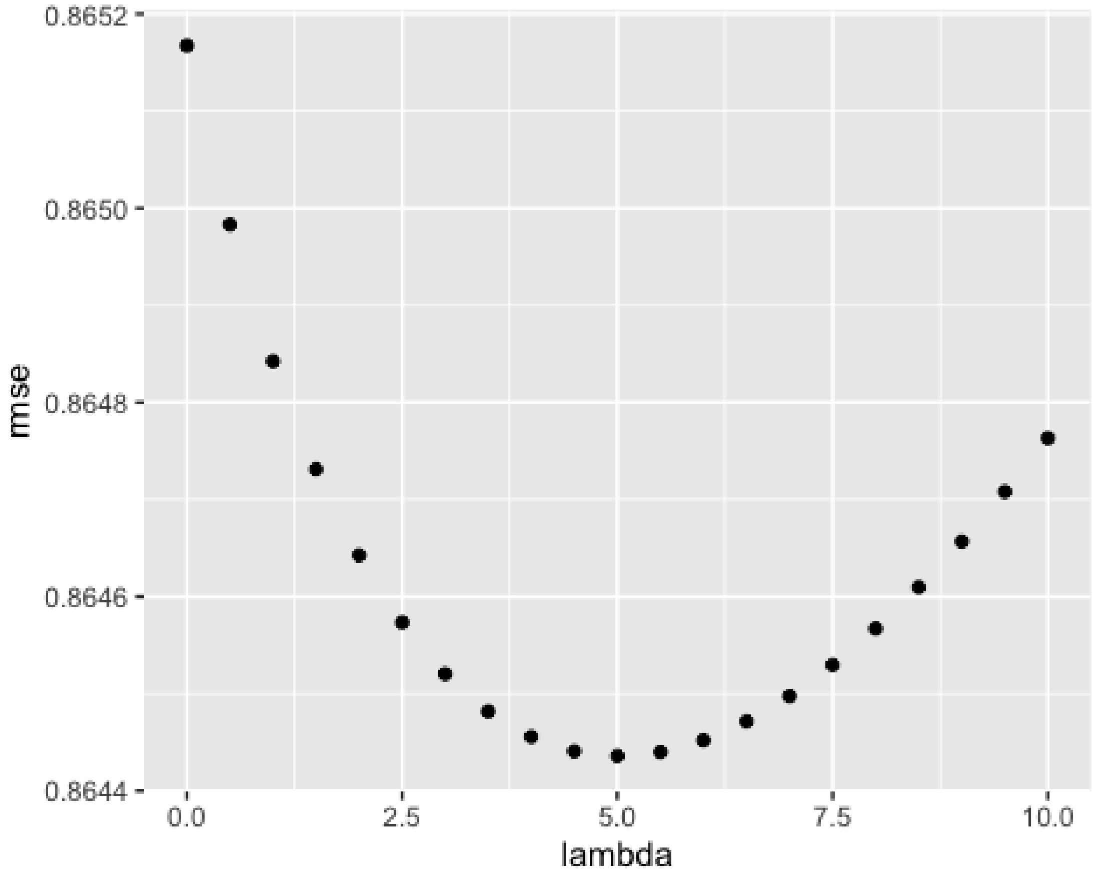
Regularization

In order to further reduce the RMSE value we need to regularize the data. There are some movies in the dataset that have less ratings than others or there are certain users that have rated less movies than others. These discrepancies lead to a higher RMSE value and a more inaccurate model. In order to remove these outliers we utilize regularization by running the Movie effect + User effect + Genre effect model and finding the lambda value that returns the lowest RMSE. After regularizing the model we were able to achieve an RMSE value of 0.8644359 which is below the target of 0.86490.

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## [1] 0.8644359
```

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



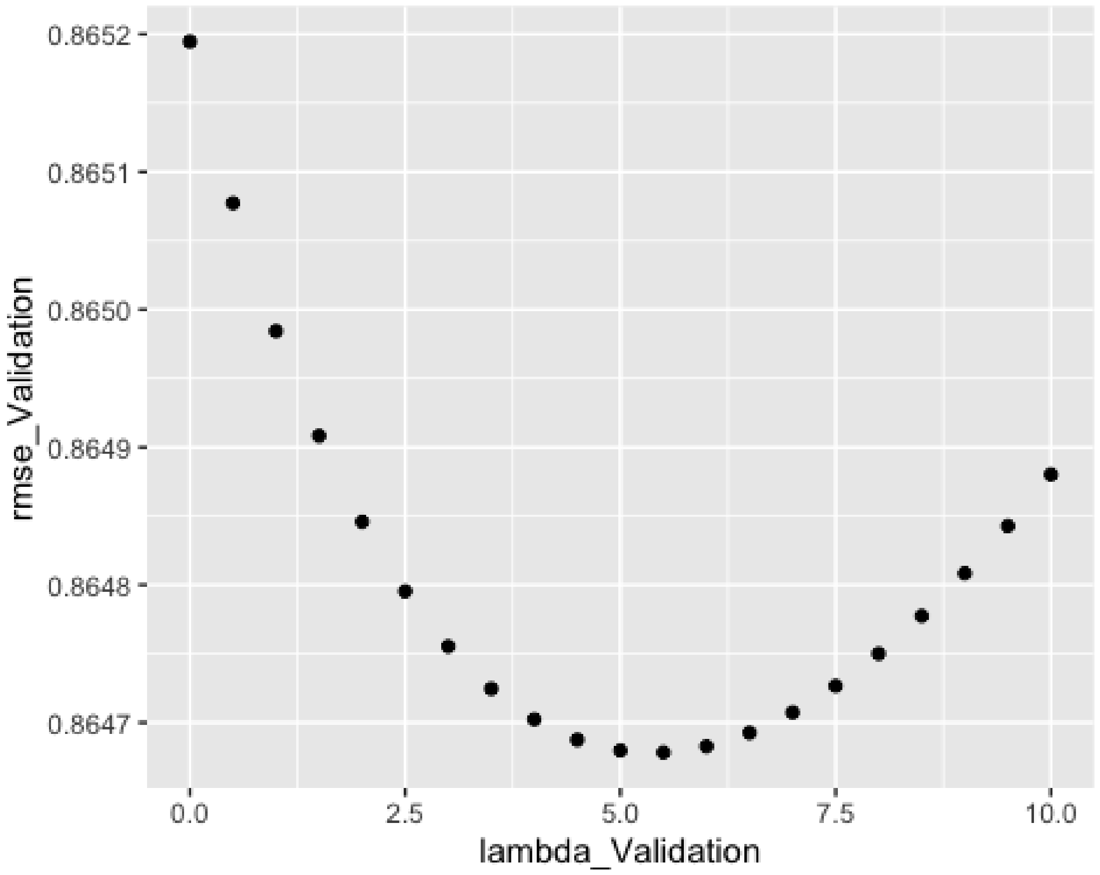
Regularization Validation Run

Since we were able to achieve an RMSE less than our goal with regularization we do a final run of our model using the edx and validation sets with the aim of achieving an RMSE less than 0.86490. After running the model we see that we get an RMSE value of 0.8646784 Which is still below our target.

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)

## [1] 0.8646784
## [1] 5.5
```

#Visual plot of lambda vs rmse



Conclusion

We can confidently conclude that we have built a machine learning algorithm that can give us predictions of movie ratings from the edx dataset. While we were able to achieve an RMSE below our target we could further reduce that value by incorporating more variables into our model.