Data Science Capstone - HarvardX PH125.9x - MovieLens

Seth Smithson

25SEP2022

Introduction

The aim of this project is to create a movie recommendation system using the 10M version of the MovieLens dataset, which was collected by GroupLens Research. Specifically, the goal is to train a machine learning algorithm which uses input, feature variables to predict the target variable, users' movie ratings. The performance of the algorithm is assessed by applying the model to a held-out validation dataset and measuring the root mean squared error (RMSE).

The MovieLens dataset was collected at a website (http://www.movielens.org/) that allowed users to rate films from one to five stars at half-star increments. These datasets are commonly used by video content providers (e.g. Netflix), online stores (e.g. Amazon), and other companies to utilize data collected from users to develop recommender systems for users. These recommender systems are statistical machine learning systems that allow for the creation of lists of content items to recommend to users based on user data that has been collected, everything from customers' shopping carts to product ratings and their similarities.

The MovieLens dataset consists of a total of 10,000,054 observations (rows) with 6 variables (columns): userId, movieId, rating, timestamp, title, and genres. There are a total of 95,580 tags, 10,681 movies, and 71,567 users. For the first part of this project, the features of the training dataset ("edx") is explored to determine the possible trends in movie ratings. The validation subset is 10% of the total MovieLens data, and contains 999,999 observations. The code needed to create these data subsets were provided in the online course materials. The characteristics and trends which were observed in the data exploration phase will guide the data analysis section. This will require data transformation in order to organize and clean the data.

After exploration, machine learning models are built and tested for their accuracy via RSME. The final RSME from the model building was XXYY.

Data Preparation

In this section, we will load the required R libraries, the source data, perform data cleaning, and data preparation. This will install an/or load all the required packages here.

Note: This process could take a couple of minutes.

```
# Clean the R environment
rm(list = ls())

# Trigger garbage collection and free up some memory
gc(TRUE, TRUE, TRUE)

# Time to check if the prerequisite packages are installed.  If they are not, R will download
them.
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(h2o)) install.packages("h2o", repos = "http://cran.us.r-project.org")
```

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

# Time to load the libraries.
library(tidyverse) # the swiss army knife for everything
library(caret) # A library that makes modeling easier
library(h2o) # A library that makes modeling much faster and use less memory
library(lubridate) # A helpful library for formatting data
library(data.table) # loading for fread function in the data loading
library(Rcpp) # contains functions useful for splitting up and assigning data
library(DataExplorer) # contains a function useful for EDA plotting
```

Here are the system and packages loaded:

```
# Let's display the system info
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22000)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] DataExplorer_0.8.2 Rcpp_1.0.9         data.table_1.14.2  lubridate_1.8.0
##  [5] caret_6.0-93       lattice_0.20-45    h2o_3.38.0.1       forcats_0.5.2
##  [9] stringr_1.4.1      dplyr_1.0.10       purrr_0.3.5        readr_2.1.3
## [13] tidyr_1.2.1        tibble_3.1.8       ggplot2_3.3.6      tidyverse_1.3.2
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-160         bitops_1.0-7         fs_1.5.2
##  [4] httr_1.4.4           tools_4.2.1          backports_1.4.1
##  [7] bslib_0.4.0          utf8_1.2.2           R6_2.5.1
## [10] rpart_4.1.16         DBI_1.1.3            colorspace_2.0-3
## [13] nnet_7.3-18          withr_2.5.0          gridExtra_2.3
## [16] tidyselect_1.2.0     compiler_4.2.1       cli_3.4.1
## [19] rvest_1.0.3          xml2_1.3.3           sass_0.4.2
```

```
## [22] scales_1.2.1          digest_0.6.29         rmarkdown_2.17
## [25] pkgconfig_2.0.3       htmltools_0.5.3       parallelly_1.32.1
## [28] dbplyr_2.2.1          fastmap_1.1.0         htmlwidgets_1.5.4
## [31] rlang_1.0.6           readxl_1.4.1          rstudioapi_0.14
## [34] jquerylib_0.1.4       generics_0.1.3        jsonlite_1.8.2
## [37] ModelMetrics_1.2.2.2 googlesheets4_1.0.1  RCurl_1.98-1.9
## [40] magrittr_2.0.3        Matrix_1.5-1          munsell_0.5.0
## [43] fansi_1.0.3           lifecycle_1.0.3       pROC_1.18.0
## [46] stringi_1.7.8         yaml_2.3.5            MASS_7.3-58.1
## [49] plyr_1.8.7            recipes_1.0.1         grid_4.2.1
## [52] parallel_4.2.1        listenv_0.8.0         crayon_1.5.2
## [55] haven_2.5.1           splines_4.2.1         hms_1.1.2
## [58] knitr_1.40            pillar_1.8.1          igraph_1.3.5
## [61] stats4_4.2.1          reshape2_1.4.4        future.apply_1.9.1
## [64] codetools_0.2-18      reprex_2.0.2          glue_1.6.2
## [67] evaluate_0.17         modelr_0.1.9          vctrs_0.4.2
## [70] tzdb_0.3.0            foreach_1.5.2         networkD3_0.4
## [73] cellranger_1.1.0      gtable_0.3.1          future_1.28.0
## [76] assertthat_0.2.1      cachem_1.0.6          xfun_0.33
## [79] gower_1.0.0           prodlim_2019.11.13    broom_1.0.1
## [82] class_7.3-20          survival_3.4-0        googledrive_2.0.0
## [85] gargle_1.2.1          timeDate_4021.106     iterators_1.0.14
## [88] hardhat_1.2.0         lava_1.6.10           globals_0.16.1
## [91] ellipsis_0.3.2        ipred_0.9-13
```

Time to Load the Data. The MovieLens 10M dataset can be found here: * https://grouplens.org/datasets/movielens/10m/ * http://files.grouplens.org/datasets/movielens/ml-10m.zip

This following script section will download the data, load it, and create the initial datasets.

```
dl <- tempfile()
options(timeout = 300, download.file.method="libcurl", url.method="libcurl") # Added to try an
d work around the download timeout
download.file("http://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")

# if using R 4.0 or later versions:
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")

# if using R 3.6 or earlier:
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId], t
itle = as.character(title), genres = as.character(genres))
```

Analysis

Time to perform some Exploratory Data Analysis.

The Head of each Table (first 10 rows)

```
# Display the very top 10 rows of the data table
head(movielens, 10)
```

```
##     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)
##  7:      1     355      5 838984474       Flintstones, The (1994)
##  8:      1     356      5 838983653           Forrest Gump (1994)
##  9:      1     362      5 838984885       Jungle Book, The (1994)
## 10:      1     364      5 838983707         Lion King, The (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
##  7:                     Children|Comedy|Fantasy
##  8:                    Comedy|Drama|Romance|War
##  9:                  Adventure|Children|Romance
## 10: Adventure|Animation|Children|Drama|Musical
```

```
head(movies, 10)
```

```
##    movieId                                title
## 1        1                    Toy Story (1995)
## 2        2                      Jumanji (1995)
## 3        3             Grumpier Old Men (1995)
## 4        4            Waiting to Exhale (1995)
## 5        5 Father of the Bride Part II (1995)
## 6        6                         Heat (1995)
## 7        7                      Sabrina (1995)
## 8        8                 Tom and Huck (1995)
## 9        9                 Sudden Death (1995)
## 10      10                    GoldenEye (1995)
##                                        genres
## 1  Adventure|Animation|Children|Comedy|Fantasy
## 2                   Adventure|Children|Fantasy
## 3                              Comedy|Romance
## 4                        Comedy|Drama|Romance
## 5                                      Comedy
## 6                        Action|Crime|Thriller
## 7                              Comedy|Romance
```

```
## 8                           Adventure|Children
## 9                                      Action
## 10                   Action|Adventure|Thriller
```

```
head(ratings, 10)
```

```
##     userId movieId rating timestamp
##  1:      1     122      5 838985046
##  2:      1     185      5 838983525
##  3:      1     231      5 838983392
##  4:      1     292      5 838983421
##  5:      1     316      5 838983392
##  6:      1     329      5 838983392
##  7:      1     355      5 838984474
##  8:      1     356      5 838983653
##  9:      1     362      5 838984885
## 10:      1     364      5 838983707
```

The column headers, data type, and initial values

```
# Display the Column Names, data type, and the initial values.
str(movielens)
```

```
## Classes 'data.table' and 'data.frame':   10000054 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 83
8983653 838984885 838983707 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1
995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|T
hriller" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(movies)
```

```
## 'data.frame':    10681 obs. of  3 variables:
##  $ movieId: num  1 2 3 4 5 6 7 8 9 10 ...
##  $ title  : chr  "Toy Story (1995)" "Jumanji (1995)" "Grumpier Old Men (1995)" "Waiting to
Exhale (1995)" ...
##  $ genres : chr  "Adventure|Animation|Children|Comedy|Fantasy" "Adventure|Children|Fantasy"
 "Comedy|Romance" "Comedy|Drama|Romance" ...
```

```
str(ratings)
```

```
## Classes 'data.table' and 'data.frame':   10000054 obs. of  4 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  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 83
8983653 838984885 838983707 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Checking for Missing data:

```
# Before we move to much on, let's check the data tables for any missing data.
sum(is.na(movielens))
```

```
## [1] 0
```

```
sum(is.na(movies))
```

```
## [1] 0
```

```
sum(is.na(ratings))
```

```
## [1] 0
```

We see that there are no missing values in any of the datasets. This makes analyzing the dataset convenient.

Checking for Dupliacte data:

```
# Check the number of rows before
sum(duplicated(movielens))
```

```
## [1] 0
```

```
sum(duplicated(movies))
```

```
## [1] 0
```

```
sum(duplicated(ratings))
```

```
## [1] 0
```

There were no rows of duplicated data. So, the dataset is ready for analysis. Thus, let's check some summary statistics.

Data Exploration

```
# Summary - Displays the Mean, Median, 25th Quartile, 75th Quartile, Min, Max
summary(movielens)
```

```
##      userId         movieId          rating        timestamp
```

```
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   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:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# Let's find the number of unique movies
movielens$movieId %>% unique() %>% length()
```

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

```
# Let's find the number of unique users
movielens$userId %>% unique() %>% length()
```

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

```
# Let's find the number of unique movie genres
movielens$genres %>% unique() %>% length()
```

```
## [1] 797
```

This seems like a lot of movie genres. However, this is because there are typically multiple genres listed for each film. This makes for many combinations. It would be much simpler if we separated these out individually.

```
# Let's find the number of movies within each genre
genres <- c("Action", "Adventure", "Animation", "Children", "Comedy", "Crime","Documentary", "
Drama", "Fantasy", "FilmNoir", "Horror", "Musical", "Mystery","Romance", "SciFi", "Thriller",
"War", "Western")
sapply(genres, function(g) {sum(str_detect(movielens$genres, g))})
```

```
##      Action   Adventure   Animation    Children      Comedy       Crime
##     2845349     2121074      519112      820149     3934068     1474957
## Documentary       Drama     Fantasy    FilmNoir      Horror     Musical
##      103454     4344198     1028482           0      768225      481174
##     Mystery     Romance       SciFi    Thriller         War     Western
##      630944     1901883           0     2584435      568063      210459
```

```
# Let's find the top ten movies with the most ratings
movielens %>% group_by(movieId) %>%
  summarise(n_ratings=n(), title=first(title)) %>%
```

```
  top_n(10, n_ratings)
```

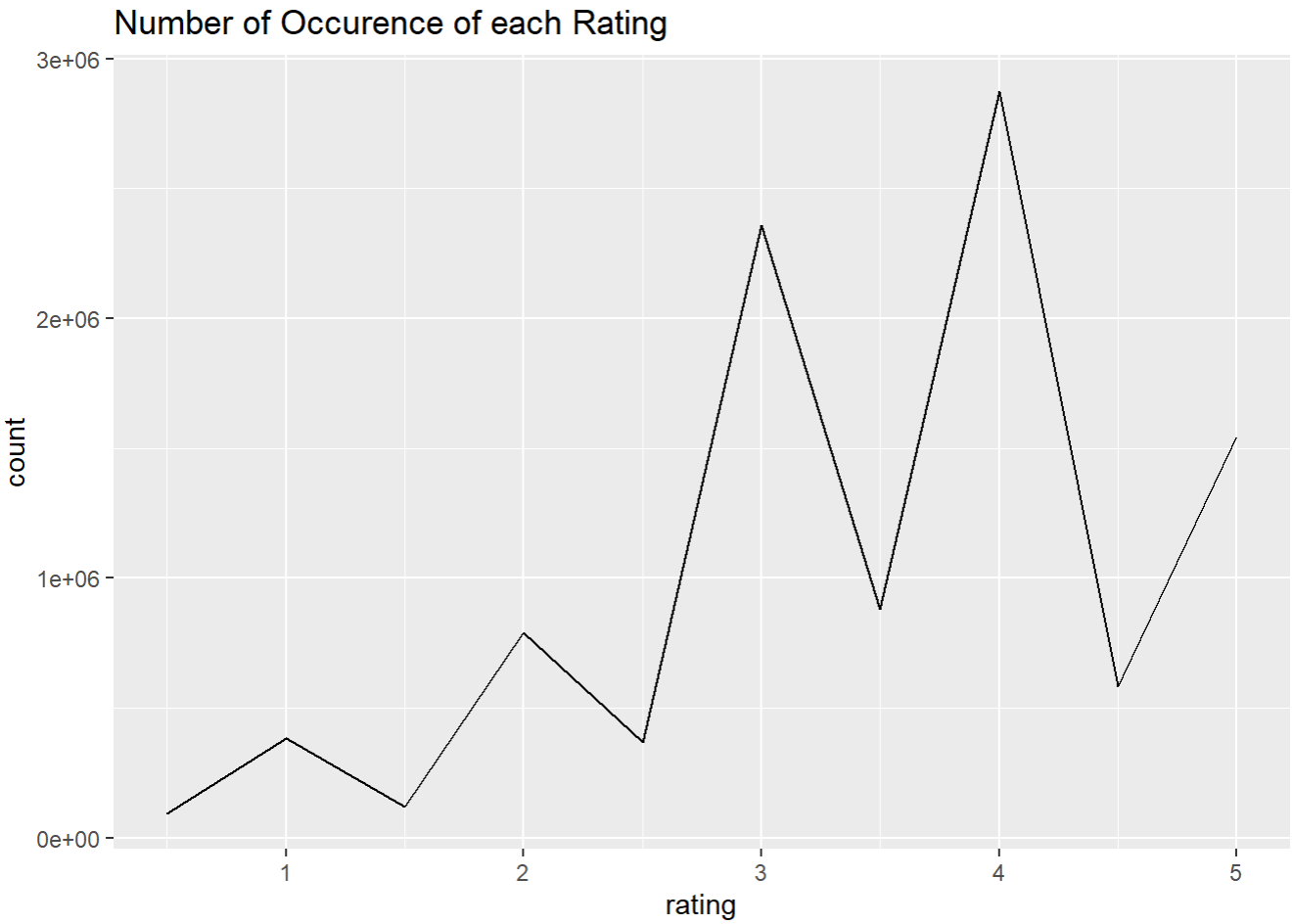
```
## # A tibble: 10 × 3
##    movieId n_ratings title
##      <dbl>     <int> <chr>
##  1     110     29154 Braveheart (1995)
##  2     150     27035 Apollo 13 (1995)
##  3     260     28566 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (197…
##  4     296     34864 Pulp Fiction (1994)
##  5     318     31126 Shawshank Redemption, The (1994)
##  6     356     34457 Forrest Gump (1994)
##  7     457     28951 Fugitive, The (1993)
##  8     480     32631 Jurassic Park (1993)
##  9     589     28948 Terminator 2: Judgment Day (1991)
## 10     593     33668 Silence of the Lambs, The (1991)
```

```
# Let's find the frequency of ratings overall
movielens %>% group_by(rating) %>%
  summarise(n_ratings=n()) %>%
  top_n(10, n_ratings) %>%
  arrange(desc(n_ratings))
```

```
## # A tibble: 10 × 2
##    rating n_ratings
##     <dbl>     <int>
##  1    4     2875850
##  2    3     2356676
##  3    5     1544812
##  4    3.5    879764
##  5    2      790306
##  6    4.5    585022
##  7    1      384180
##  8    2.5    370178
##  9    1.5    118278
## 10    0.5     94988
```

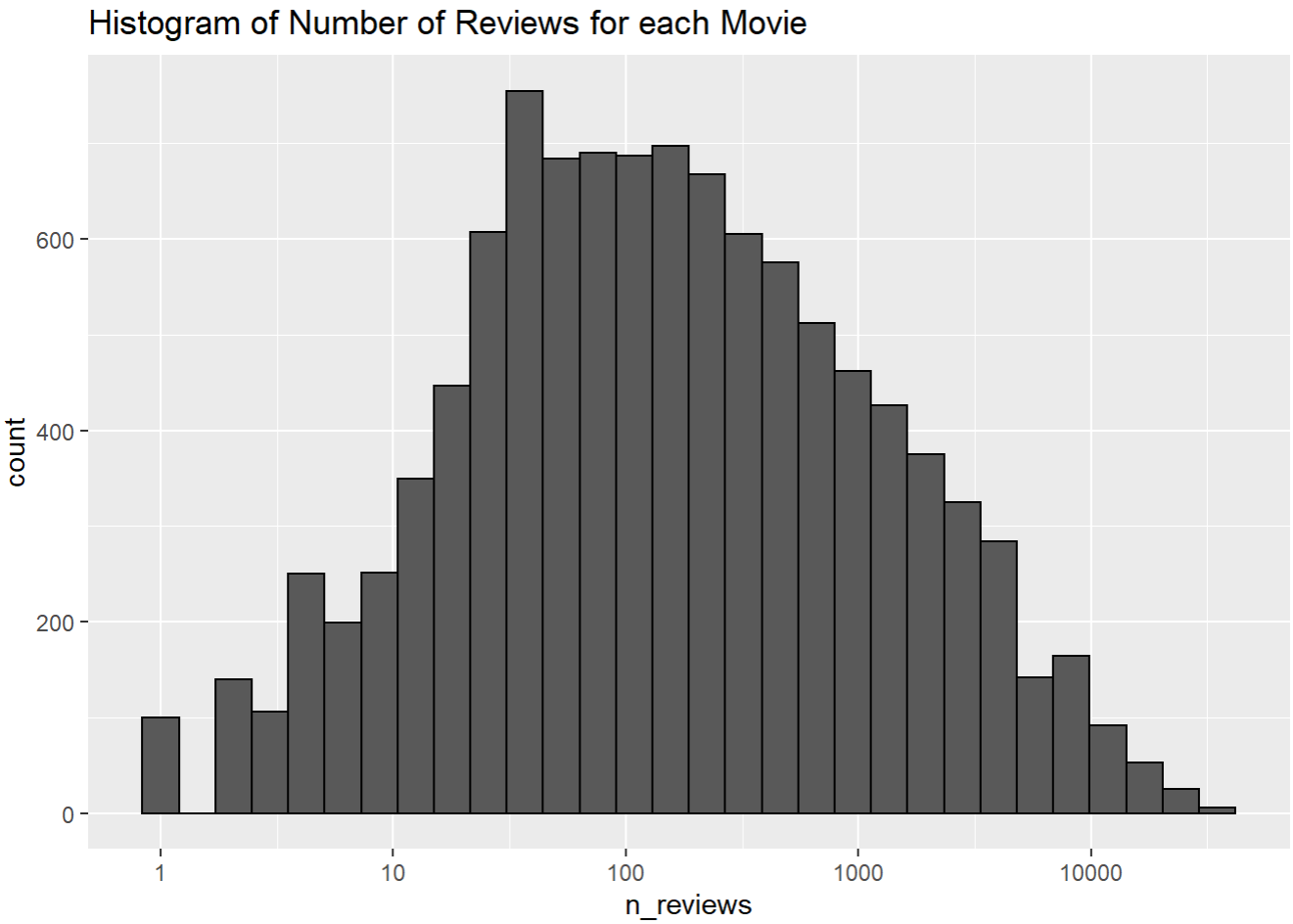
Dataset Visualization

```
# Let's graph the occurrence of each rating
movielens %>%
    group_by(rating) %>%
    summarize(count = n()) %>%
    ggplot(aes(x = rating, y = count)) +
    geom_line() +
    ggtitle("Number of Occurence of each Rating")
```



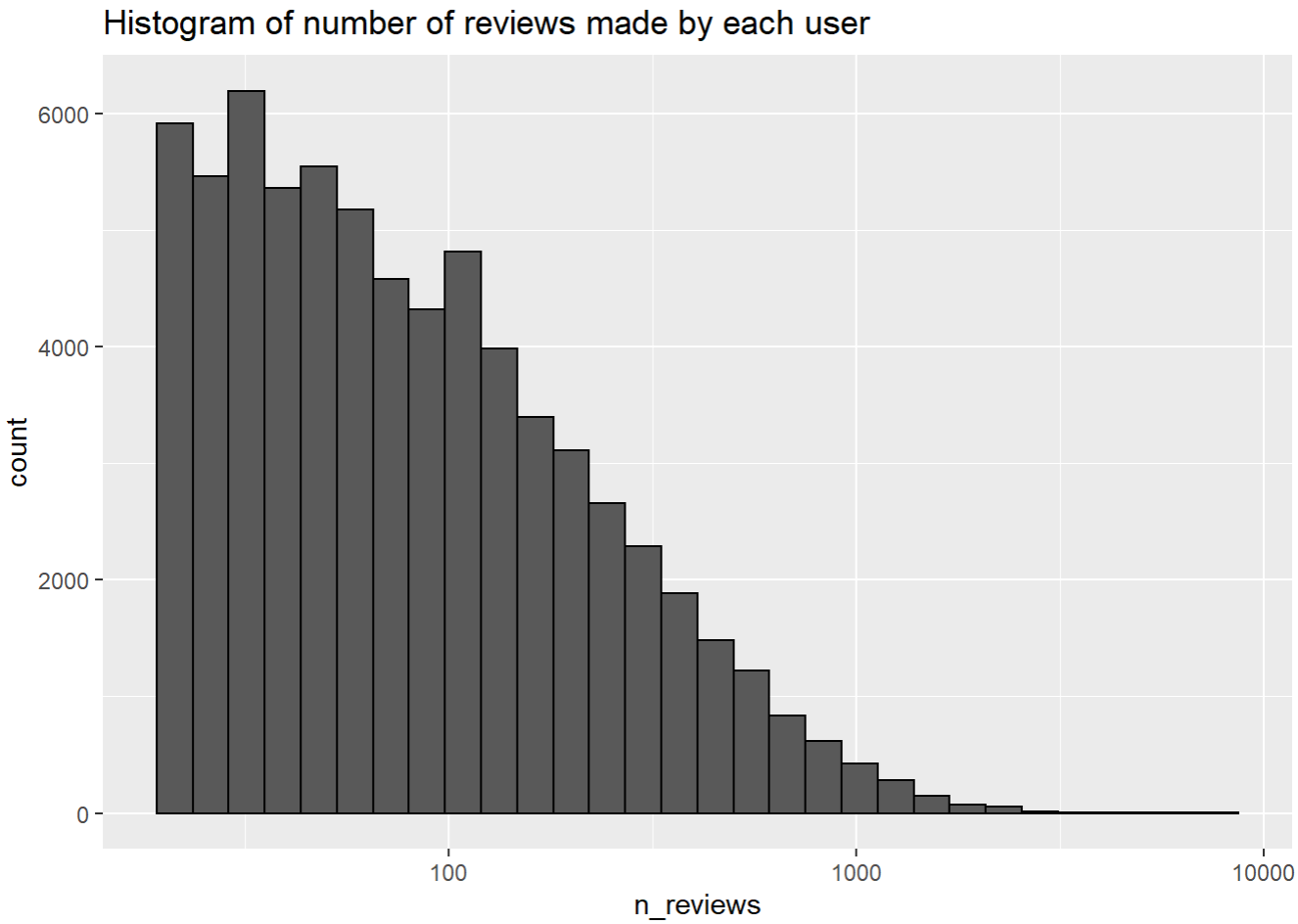

```
# Let's see the number of ratings for each movie
movielens %>%
  group_by(movieId) %>%
  summarise(n_reviews=n()) %>%
  ggplot(aes(n_reviews)) +
  geom_histogram(color="black") +
  scale_x_log10() +
  ggtitle("Histogram of Number of Reviews for each Movie")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



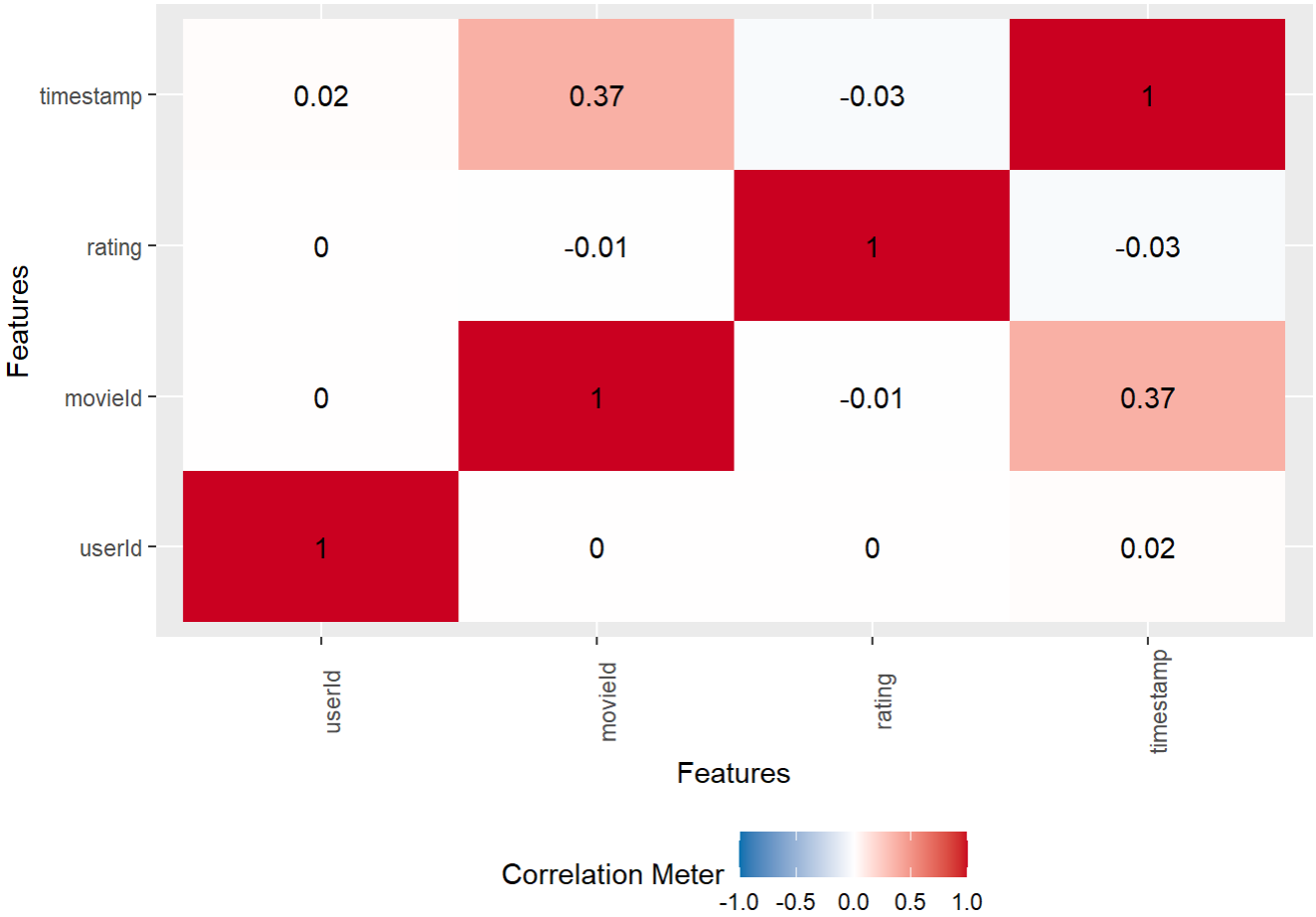

```
# Let's see the number of ratings made by each user
movielens %>%
  group_by(userId) %>%
  summarise(n_reviews=n()) %>%
  ggplot(aes(n_reviews)) +
  geom_histogram(color="black") +
  scale_x_log10() +
  ggtitle("Histogram of number of reviews made by each user")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Let's see the correlations between variables
plot_correlation(movielens)
```

```
## Warning in dummify(data, maxcat = maxcat): Ignored all discrete features since
## `maxcat` set to 20 categories!
```



Insights Gained

From the data exploration, it can be seen that most users have made 100 reviews or fewer and that the mode review ratings are 3 and 4. Furthermore, reviewers tend to give ratings as whole-numbers more often than half-star ratings. The movies in the Action, Adventure, and Animation genres receive the most reviews.

Data Cleaning

We can see that the timestamp column in the movielens and ratings data tables are displaying incorrectly. Let's update this.

```
# Formatting the timestamps
ratings <- ratings %>% mutate(timestamp=as_datetime(timestamp)) #converting
ratings %>% rename(rating_datetime = timestamp) # rename the timestamp column to be more repre
sentative
```

```
##           userId movieId rating     rating_datetime
##        1:      1     122      5 1996-08-02 11:24:06
##        2:      1     185      5 1996-08-02 10:58:45
##        3:      1     231      5 1996-08-02 10:56:32
##        4:      1     292      5 1996-08-02 10:57:01
##        5:      1     316      5 1996-08-02 10:56:32
##       ---
```

```
## 10000050:  71567    2107      1 1998-12-02 06:35:53
## 10000051:  71567    2126      2 1998-12-03 01:39:03
## 10000052:  71567    2294      5 1998-12-02 05:52:48
## 10000053:  71567    2338      2 1998-12-02 05:53:36
## 10000054:  71567    2384      2 1998-12-02 05:56:13
```

```
movielens <- movielens %>% mutate(timestamp=as_datetime(timestamp))
movielens %>% rename(rating_datetime = timestamp) # rename the timestamp column to be more rep
resentative
```

```
##           userId movieId rating     rating_datetime
##        1:      1     122      5 1996-08-02 11:24:06
##        2:      1     185      5 1996-08-02 10:58:45
##        3:      1     231      5 1996-08-02 10:56:32
##        4:      1     292      5 1996-08-02 10:57:01
##        5:      1     316      5 1996-08-02 10:56:32
##       ---
## 10000050:  71567    2107      1 1998-12-02 06:35:53
## 10000051:  71567    2126      2 1998-12-03 01:39:03
## 10000052:  71567    2294      5 1998-12-02 05:52:48
## 10000053:  71567    2338      2 1998-12-02 05:53:36
## 10000054:  71567    2384      2 1998-12-02 05:56:13
##                                                 title
##        1:                              Boomerang (1992)
##        2:                                Net, The (1995)
##        3:                           Dumb & Dumber (1994)
##        4:                                Outbreak (1995)
##        5:                                Stargate (1994)
##       ---
## 10000050:         Halloween H20: 20 Years Later (1998)
## 10000051:                              Snake Eyes (1998)
## 10000052:                                   Antz (1998)
## 10000053: I Still Know What You Did Last Summer (1998)
## 10000054:                  Babe: Pig in the City (1998)
##                                               genres
##        1:                                Comedy|Romance
##        2:                          Action|Crime|Thriller
##        3:                                       Comedy
##        4:                  Action|Drama|Sci-Fi|Thriller
##        5:                       Action|Adventure|Sci-Fi
##       ---
## 10000050:                                Horror|Thriller
## 10000051:                Action|Crime|Mystery|Thriller
## 10000052: Adventure|Animation|Children|Comedy|Fantasy
## 10000053:                        Horror|Mystery|Thriller
## 10000054:                                Children|Comedy
```

```
# Check the results
head(ratings)
```

```
##    userId movieId rating           timestamp
```

```
## 1:      1     122      5 1996-08-02 11:24:06
## 2:      1     185      5 1996-08-02 10:58:45
## 3:      1     231      5 1996-08-02 10:56:32
## 4:      1     292      5 1996-08-02 10:57:01
## 5:      1     316      5 1996-08-02 10:56:32
## 6:      1     329      5 1996-08-02 10:56:32
```

```
head(movielens)
```

```
##    userId movieId rating           timestamp                         title
## 1:      1     122      5 1996-08-02 11:24:06              Boomerang (1992)
## 2:      1     185      5 1996-08-02 10:58:45               Net, The (1995)
## 3:      1     231      5 1996-08-02 10:56:32          Dumb & Dumber (1994)
## 4:      1     292      5 1996-08-02 10:57:01               Outbreak (1995)
## 5:      1     316      5 1996-08-02 10:56:32               Stargate (1994)
## 6:      1     329      5 1996-08-02 10:56:32 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
```

There is still more data cleaning to do. If you notice, the timestamps were of movie ratings, not necessarily the release date of the movies themselves. The movie release year is still in the same feature as the Movie name. These need to be separated.

```
#extracting the premier date from the movies data frame
movies$premier_date <- stringi::stri_extract(movies$title, regex = "(\\d{4})", comments = TRUE
 ) %>% as.numeric()

#extracting the premier date from the movielens data frame
movielens$premier_date <- stringi::stri_extract(movielens$title, regex = "(\\d{4})", comments
= TRUE ) %>% as.numeric()

# Check the results
head(movies)
```

```
##   movieId                               title
## 1       1                    Toy Story (1995)
## 2       2                      Jumanji (1995)
## 3       3             Grumpier Old Men (1995)
## 4       4            Waiting to Exhale (1995)
## 5       5 Father of the Bride Part II (1995)
## 6       6                         Heat (1995)
##                                        genres premier_date
## 1 Adventure|Animation|Children|Comedy|Fantasy         1995
## 2                  Adventure|Children|Fantasy         1995
## 3                              Comedy|Romance         1995
## 4                        Comedy|Drama|Romance         1995
```

```
## 5                                     Comedy         1995
## 6                       Action|Crime|Thriller         1995
```

```
head(movielens)
```

```
##    userId movieId rating           timestamp                          title
## 1:      1     122      5 1996-08-02 11:24:06               Boomerang (1992)
## 2:      1     185      5 1996-08-02 10:58:45                Net, The (1995)
## 3:      1     231      5 1996-08-02 10:56:32           Dumb & Dumber (1994)
## 4:      1     292      5 1996-08-02 10:57:01                Outbreak (1995)
## 5:      1     316      5 1996-08-02 10:56:32                Stargate (1994)
## 6:      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
##                            genres premier_date
## 1:                Comedy|Romance         1992
## 2:         Action|Crime|Thriller         1995
## 3:                         Comedy         1994
## 4:  Action|Drama|Sci-Fi|Thriller         1995
## 5:       Action|Adventure|Sci-Fi         1994
## 6: Action|Adventure|Drama|Sci-Fi         1994
```

There is one more collosal task. The genre section can contain multiple genres for each movie. This is not very helpful. To increase the utility of the genre information, it is going to be converted to new columns of genres with 0 and 1 encoding for each row of data.

```
# Let's one-hot encode the movies data frame
movies_one_hot <- movies %>% separate_rows(genres,sep = "\\|") %>% mutate(value=1)
movies_one_hot <- movies_one_hot %>% spread(genres, value, fill=0) %>% select(-"(no genres lis
ted)")

# Let's one-hot encode the movielens data frame
movielens_one_hot <- movielens %>% separate_rows(genres,sep = "\\|") %>% mutate(value=1)
movielens_one_hot <- movielens_one_hot %>% spread(genres, value, fill=0) %>% select(-"(no genr
es listed)")
```

Now, we are ready to split the data into a training and validation set, utilizing the script provided within the capstone assignment on edx. It will create a Validation set consisting of 10% of MovieLens data called “validation” and a training set called “edx” that contains 90% of the data. For training the models, only the caret library will be used for the fitting process, which will automatically split the training data into a train and test set. So, only a hold-out validation set is needed to evaluate the models created.

```
# There are some errors that may occur that can only be fixed by installing and loading the Rc
pp package in the current instance of R.
# install.packages("Rcpp")
# library(Rcpp) # Reloading the Rcpp package to address any potential error message
# Rcpp::Rcpp_precious_remove()

# Time to proceed with the edx R script to split the data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE) # Th
e only caret function utilized (for splitting data)
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", "genres",
## "premier_date")
```

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

# Time to proceed with the edx R script to split the data
test_index_one_hot <- createDataPartition(y = movielens_one_hot$rating, times = 1, p = 0.1, li
st = FALSE) # The only caret function utilized (for splitting data)
edx_one_hot <- movielens_one_hot[-test_index,]
temp_one_hot <- movielens_one_hot[test_index,]

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

# Add rows removed from validation set back into edx set
removed_one_hot <- anti_join(temp_one_hot, validation_one_hot)
```

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

```
edx_one_hot <- rbind(edx_one_hot, removed_one_hot)

# Remove data no longer needed
rm(dl, ratings, movies, test_index, temp, movielens, removed, genres)
```

Modeling

Now that we have an understanding of the dataset, and the data has been prepared, it is time to build models.

For the modeling, H2O is going to be utilized for its speed and memory compression. Nevertheless, the modeling process will consume up to 64 GB of RAM while it creates various regression, ensemble, or neural network models. These steps were attemped before using Caret, NerualNet, and RandomForest libraries and required even more RAM (up to 450GB of RAM). This is beyond the capacity of most users and may require access to a High-Performance-Computer at an

institution or renting cloud computing clusters. H2O is much more pragmatic for a high-end desktop personal computer. The H2O library utilizes Java to enable greater parallel processing and utilize memory more efficiently. It will use a greater amount of the processor clock cycles than other packages (e.g., Caret) while using up to 20x less RAM.

Note: Ensure that **Java 64-bit** and **R 64-bit** are installed prior to running these scripts.

Initialize H2O

```
## Initialize H2O
h2o.init(
    nthreads = -1, # Specifying to use all processor cores available
    max_mem_size = "64G" # Specifying to use 64GB of RAM.
)
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\sirhu\AppData\Local\Temp\RtmpEdUoh5\filef3f034ecb/h2o_sirhu_started_from_r.out
##     C:\Users\sirhu\AppData\Local\Temp\RtmpEdUoh5\filef3f026821a18/h2o_sirhu_started_from_r.
err
##
##
## Starting H2O JVM and connecting:  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 seconds 93 milliseconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    1 month and 12 days
##     H2O cluster name:           H2O_started_from_R_sirhu_rcn141
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   56.86 GB
##     H2O cluster total cores:    24
##     H2O cluster allowed cores:  24
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.1 (2022-06-23 ucrt)
```

```
## Convert current dataframes to H2O dataframes
## This might also use a large amount of memory, so one data frame will be converted and remov
ed at a time.

validation_one_hot_h2o <- as.h2o(validation_one_hot) # Convert current dataframe to h2o datafr
ame
```

```
##
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
validation_one_hot_h2o[c("rating", "userId", "movieId", "premier_date")] <- as.factor(validati
on_one_hot_h2o[c("rating", "userId", "movieId", "premier_date")]) #  Changing target column ty
pe to factor for classification work in h2o
# rm(validation_one_hot) # Remove old data file to preserve memory

edx_one_hot_h2o <- as.h2o(edx_one_hot) # Convert current dataframe to h2o dataframe
```

```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
edx_one_hot_h2o[c("rating", "userId", "movieId", "premier_date")] <- as.factor(edx_one_hot_h2o
[c("rating", "userId", "movieId", "premier_date")]) # Changing target column type to factor fo
r classification work in h2o
# rm(edx_one_hot) # Remove old data file to preserve memory
```

```
# Assign the training and target variables.
y <- "rating" # target feature
x <- setdiff(names(edx_one_hot_h2o), y) # all the training features minus the target feature
```

```
# Set hyperparameter grid for GLM:
glm_hyper_grid.h2o <- list(
  alpha = seq(0, 1, by = 0.05)
)

# Create a Random Discrete Grid Search
glm_search_criteria <- list(
    strategy = "RandomDiscrete", # Selecting Random Search of the hyperparameter grid
    stopping_metric = "RMSE", # The metric being optimized for in model training
    seed = 42, # Setting the seed to the answer to life, the universe, and everything.
    max_runtime_secs = 120*60 # Two hour run time
)

# Model 3:  Generalized Linear Model (GLM)
glm_random_grid <- h2o.grid(
    algorithm = "glm", # Selecting the Algorithm to build the model
    grid_id = "glm_grid", # Naming the Grid
    x = x, # Specifying the training data labels
    y = y, # Specifying the target attribute label
    nfolds = 5, # K-Fold Cross Validation
    training_frame = edx_one_hot_h2o, # Training Data selection
    hyper_params = glm_hyper_grid.h2o, # Specifying the hyperparameter grid to try
    search_criteria = glm_search_criteria # Specifying the search criteria to utilize
)
```

```
## Warning in h2o.getGrid(grid_id = grid_id): Adding alpha array to hyperparameter
## runs slower with gridsearch. This is due to the fact that the algo has to
## run initialization for every alpha value. Setting the alpha array as a model
## parameter will skip the initialization and run faster overall.
```

```
# Collect the results and sort by our models:
glm_grid_perf <- h2o.getGrid(
    grid_id = "glm_grid", # Grid of models
    sort_by = "RMSE", # Sorting the leaderboard by the desired metric
    decreasing = FALSE # Sort by Ascending/Increasing Values
)
```

```
## Warning in h2o.getGrid(grid_id = "glm_grid", sort_by = "RMSE", decreasing =
## FALSE): Adding alpha array to hyperparameter runs slower with gridsearch. This
## is due to the fact that the algo has to run initialization for every alpha
## value. Setting the alpha array as a model parameter will skip the initialization
## and run faster overall.
```

```
print(glm_grid_perf)
```

```
## H2O Grid Details
## ================
##
## Grid ID: glm_grid
## Used hyper parameters:
##   -  alpha
## Number of models: 5
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##   alpha        model_ids    rmse
## 1  0.85 glm_grid_model_4 0.79182
## 2   0.6 glm_grid_model_2 0.79216
## 3   0.4 glm_grid_model_1 0.79276
## 4   0.3 glm_grid_model_3 0.79322
## 5   0.2 glm_grid_model_5 0.79325
```

```
# Select the best model
best_glm_model <- h2o.getModel(glm_grid_perf@model_ids[[1]])
print(best_glm_model@model[["model_summary"]])
```

```
## GLM Model: summary
##        family        link                                regularization
## 1 multinomial multinomial Elastic Net (alpha = 0.85, lambda = 9.53E-5 )
##   number_of_predictors_total number_of_active_predictors number_of_iterations
## 1                     806830                         489                    2
##    training_frame
## 1 RTMP_sid_bef0_5
```

```
# Retrieve the variable importance
glm_varimp <- h2o.varimp(best_glm_model)
print(glm_varimp)
```

```
## Variable Importances:
##            variable relative_importance scaled_importance percentage
## 1         timestamp            7.652309          1.000000   0.065738
## 2 premier_date.1995            3.122599          0.408060   0.026825
## 3 premier_date.1994            2.923728          0.382071   0.025116
## 4       movieId.593            2.912545          0.380610   0.025020
## 5 premier_date.1996            2.520487          0.329376   0.021652
##
## ---
##                variable relative_importance scaled_importance percentage
## 80677 premier_date.1978            0.000000          0.000000   0.000000
## 80678 premier_date.2010            0.000000          0.000000   0.000000
## 80679 premier_date.2046            0.000000          0.000000   0.000000
## 80680 premier_date.3000            0.000000          0.000000   0.000000
## 80681 premier_date.5000            0.000000          0.000000   0.000000
## 80682 premier_date.9000            0.000000          0.000000   0.000000
```

```
# Check model performance
glm_perf <- h2o.performance(best_glm_model, validation_one_hot_h2o)
glm_rmse <- h2o.rmse(glm_perf)
print(glm_rmse)
```

```
## [1] 0.7936825
```

```
# Set hyperparameter grid:
gbm_hyper_grid.h2o <- list(
    ntrees = c(10, 50, 100, 150, 200),
    max_depth = c(3, 5, 7, 9, 11, 13, 15, 17, 19, 21),
    learn_rate = c(0.001, 0.01, 0.05, 0.1, 0.15, 0.2)
)

# Create a Random Discrete Grid Search
gbm_search_criteria <- list(
    strategy = "RandomDiscrete", # Selecting Random Search of the hyperparameter grid
    stopping_metric = "RMSE", # The metric being optimized for in model training
    seed = 42, # Setting the seed to the answer to life, the universe, and everything.
    max_runtime_secs = 120*60 # Specifying a max run-time for this command in seconds (two hou
rs)
)

# Model 3:  Gradient Boosting Machine modeling
gbm_random_grid <- h2o.grid(
    algorithm = "gbm", # Selecting the Algorithm to build the model
    grid_id = "gbm_grid", # Naming the grid
    x = x, # Specifying the training data labels
    y = y, # Specifying the target attribute label
```

```
    nfolds = 5, # K-Fold Cross Validation
    training_frame = edx_one_hot_h2o, # Training Data selection
    hyper_params = gbm_hyper_grid.h2o, # Specifying the hyperparameter grid to try
    search_criteria = gbm_search_criteria # Specifying the search criteria to utilize
)
```

```
# Collect the results and sort by our models:
gbm_grid_perf <- h2o.getGrid(
    grid_id = "gbm_grid", # Grid of models
    sort_by = "RMSE", # Sorting the leaderboard by the desired metric
    decreasing = FALSE # Sort by Ascending/Increasing Values
)
print(gbm_grid_perf)
```

```
## H2O Grid Details
## ================
##
## Grid ID: gbm_grid
## Used hyper parameters:
##   -  learn_rate
##   -  max_depth
##   -  ntrees
## Number of models: 3
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##   learn_rate max_depth   ntrees        model_ids    rmse
## 1    0.05000   9.00000 67.00000 gbm_grid_model_1 0.77255
## 2    0.05000   3.00000 10.00000 gbm_grid_model_2 0.85265
## 3    0.20000  19.00000  1.00000 gbm_grid_model_3 0.86531
```

```
# Best model:
best_gbm_model <- h2o.getModel(gbm_grid_perf@model_ids[[1]])
print(best_gbm_model@model[["model_summary"]])
```

```
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1              12                      120             1448533         9
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         9    9.00000        282        511   479.15000
```

```
# Retrieve the variable importance
gbm_varimp <- h2o.varimp(best_gbm_model)
print(gbm_varimp)
```

```
## Variable Importances:
##       variable relative_importance scaled_importance percentage
## 1    timestamp      1941721.000000          1.000000   0.576100
## 2      movieId       505533.812500          0.260353   0.149990
## 3 premier_date       449338.437500          0.231412   0.133317
```

```
## 4        Drama       151986.890625          0.078274   0.045094
## 5     Children        34541.589844          0.017789   0.010248
##
## ---
##     variable relative_importance scaled_importance percentage
## 18    Sci-Fi         9667.466797          0.004979   0.002868
## 19 Film-Noir         8082.934570          0.004163   0.002398
## 20    userId         7280.960938          0.003750   0.002160
## 21   Musical         3696.338379          0.001904   0.001097
## 22   Western         1254.696777          0.000646   0.000372
## 23      IMAX          961.571411          0.000495   0.000285
```

```
# Check model performance
gbm_perf <- h2o.performance(best_gbm_model, validation_one_hot_h2o)
gbm_rmse <- h2o.rmse(gbm_perf)
print(gbm_rmse)
```

```
## [1] 0.8473865
```

```
# Model 3:  Naive Bayes

# Set hyperparameter grid:
hyper_params <- list(
    laplace = seq(0, 1, by = 0.1)
)

# Create a Random Discrete Grid Search
nb_search_criteria <- list(
    strategy = "RandomDiscrete", # Selecting Random Search of the hyperparameter grid
    stopping_metric = "RMSE", # The metric being optimized for in model training
    seed = 42, # Setting the seed to the answer to life, the universe, and everything.
    max_runtime_secs = 120*60 # Specifying a max run-time for this command in seconds (two hou
rs)
)

# Parameters for Naieve Bayes model
nb_random_grid <- h2o.grid(
    algorithm = "naivebayes", # Selecting the Algorithm to build the model
    grid_id = "nb_grid", # Naming the grid
    x = x, # Specifying the training data labels
    y = y, # Specifying the target attribute label
    nfolds = 5, # K-Fold Cross Validation
    training_frame = edx_one_hot_h2o, # Training Data selection
    hyper_params = hyper_params, # Specifying the hyperparameter grid to try
    search_criteria = nb_search_criteria # Specifying the search criteria to utilize
)
```

```
# Collect the results and sort by our models:
nb_grid_perf <- h2o.getGrid(
    grid_id = "nb_grid", # Grid of models
    sort_by = "RMSE", # Sorting the leaderboard by the desired metric
```

```
    decreasing = FALSE # Sort by Ascending/Increasing Values
)
print(nb_grid_perf)
```

```
## H2O Grid Details
## ================
##
## Grid ID: nb_grid
## Used hyper parameters:
##   -  laplace
## Number of models: 11
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##    laplace        model_ids    rmse
## 1  0.10000  nb_grid_model_6 0.78023
## 2  0.20000  nb_grid_model_7 0.78049
## 3  0.30000  nb_grid_model_5 0.78070
## 4  0.40000  nb_grid_model_2 0.78087
## 5  0.50000  nb_grid_model_4 0.78100
## 6  0.60000  nb_grid_model_9 0.78112
## 7  0.70000  nb_grid_model_1 0.78125
## 8  0.80000  nb_grid_model_3 0.78135
## 9  0.90000 nb_grid_model_11 0.78143
## 10 1.00000  nb_grid_model_8 0.78152
## 11 0.00000 nb_grid_model_10 0.83434
```

```
# Best model:
best_nb_model <- h2o.getModel(nb_grid_perf@model_ids[[1]])
print(best_nb_model@model[["model_summary"]])
```

```
## Model Summary:
##   number_of_response_levels min_apriori_probability max_apriori_probability
## 1                        10                 0.00949                 0.28760
```

```
# Check model performance
nb_perf <- h2o.performance(
  best_nb_model,
  newdata = validation_one_hot_h2o)
nb_rmse <- h2o.rmse(nb_perf)
print(nb_rmse)
```

```
## [1] 0.7411595
```

```
# Set hyperparameter grid for RF:
rf_hyper_grid <- list(
    ntrees = seq(1, 100, by = 3),
    mtries = seq(1, 100, by = 3),
    max_depth = seq(1, 25, by = 3),
    min_rows = seq(1, 10, by = 1),
```

```
    nbins = seq(1, 1000, by = 25),
    sample_rate = seq(0.05, 0.95, by = 0.1)
)

# Create a Random Discrete Grid Search
rf_search_criteria <- list(
    strategy = "RandomDiscrete", # Selecting Random Search of the hyperparameter grid
    stopping_metric = "RMSE", # The metric being optimized for in model training
    max_runtime_secs = 120*60, # Specifying a max run-time for this command in seconds (two ho
urs)
    stopping_tolerance = 0.005, # stop if improvement is < 0.5%
    stopping_rounds = 10 # Over the last 10 models
)

# Parameters for Random Forest Model
random_grid <- h2o.grid(
    algorithm = "randomForest", # Selecting the Algorithm to build the model
    grid_id = "rf_grid", # Naming the grid
    x = x, # Specifying the training data labels
    y = y, # Specifying the target attribute label
    seed = 42, # Setting the seed to the answer to life, the universe, and everything.
    nfolds = 5, # K-Fold Cross Validation
    training_frame = edx_one_hot_h2o, # Training Data selection
    hyper_params = rf_hyper_grid, # Specifying the hyperparameter grid to try
    search_criteria = rf_search_criteria # Specifying the search criteria to utilize
)
```

```
## Warning in h2o.getGrid(grid_id = grid_id): Some models were not built due to a
## failure, for more details run `summary(grid_object, show_stack_traces = TRUE)`
```

```
# Collect the results and sort by our models:
rf_grid_perf <- h2o.getGrid(
    grid_id = "rf_grid", # Grid of models
    sort_by = "RMSE", # Sorting the leaderboard by the desired metric
    decreasing = FALSE # Sort by Ascending/Increasing Values
)
```

```
## Warning in h2o.getGrid(grid_id = "rf_grid", sort_by = "RMSE", decreasing =
## FALSE): Some models were not built due to a failure, for more details run
## `summary(grid_object, show_stack_traces = TRUE)`
```

```
print(rf_grid_perf)
```

```
## H2O Grid Details
## ================
##
## Grid ID: rf_grid
## Used hyper parameters:
##   -  max_depth
##   -  min_rows
##   -  mtries
```

```
##   -  nbins
##   -  ntrees
##   -  sample_rate
## Number of models: 3
## Number of failed models: 11
##
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##   max_depth min_rows   mtries     nbins   ntrees sample_rate        model_ids
## 1  22.00000 10.00000 22.00000 751.00000 10.00000     0.25000  rf_grid_model_5
## 2   4.00000  4.00000 19.00000 476.00000 49.00000     0.35000  rf_grid_model_1
## 3   4.00000  6.00000  4.00000  76.00000 12.00000     0.55000 rf_grid_model_14
##      rmse
## 1 0.76656
## 2 0.80632
## 3 0.81417
## Failed models
## -------------
##  max_depth min_rows mtries nbins ntrees sample_rate status_failed
##         19      9.0      1     1      4        0.75          FAIL
##          4      9.0     52   726     37        0.65          FAIL
##         16      5.0     61   176     40        0.45          FAIL
##         10      8.0     79   901     37        0.95          FAIL
##         13      8.0     49   951     70        0.75          FAIL
##         10      1.0     37   251     73        0.15          FAIL
##          7     10.0     55   651      7        0.15          FAIL
##         19      4.0     43   376     25        0.85          FAIL
##         25      2.0    100   401     43        0.15          FAIL
##         13      7.0     58   426     79        0.55          FAIL
##          4      2.0     55   851     28        0.25          FAIL
##
```

```
                                                                msgs_failed
##
```

```
                                      "Illegal argument(s) for DRF model: rf_grid_model_2.
 Details: ERRR on field: _nbins: nbins must be > 1.\nERRR on field: _nbins: nbins must be > 1.
\nERRR on field: _nbins: nbins must be > 1.\nERRR on field: _nbins: nbins must be > 1.\nERRR o
n field: _nbins: nbins must be > 1.\nERRR on field: _nbins: nbins must be > 1.\n"
##         "Illegal argument(s) for DRF model: rf_grid_model_3.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 52\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 52\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 52\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 52\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 52\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 52\n"
##         "Illegal argument(s) for DRF model: rf_grid_model_4.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 61\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 61\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 61\nERRR on field: _m
```

```
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 61\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 61\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 61\n"
##         "Illegal argument(s) for DRF model: rf_grid_model_6.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 79\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 79\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 79\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 79\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 79\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 79\n"
##         "Illegal argument(s) for DRF model: rf_grid_model_7.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 49\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 49\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 49\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 49\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 49\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 49\n"
##         "Illegal argument(s) for DRF model: rf_grid_model_8.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 37\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 37\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 37\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 37\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 37\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 37\n"
##         "Illegal argument(s) for DRF model: rf_grid_model_9.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 55\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\n"
##        "Illegal argument(s) for DRF model: rf_grid_model_10.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 43\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 43\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 43\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 43\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 43\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 43\n"
##  "Illegal argument(s) for DRF model: rf_grid_model_11.  Details: ERRR on field: _mtries: Co
mputed mtries should be -1 or -2 or in interval [1,24[ but it is 100\nERRR on field: _mtries:
Computed mtries should be -1 or -2 or in interval [1,25[ but it is 100\nERRR on field: _mtries
: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 100\nERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 100\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 100\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 100\n"
##        "Illegal argument(s) for DRF model: rf_grid_model_12.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 58\nERRR on field: _mtr
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 58\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 58\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 58\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 58\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 58\n"
##        "Illegal argument(s) for DRF model: rf_grid_model_13.  Details: ERRR on field: _mtri
es: Computed mtries should be -1 or -2 or in interval [1,24[ but it is 55\nERRR on field: _mtr
```

```
ies: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field: _mt
ries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field: _m
tries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field: _
mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\nERRR on field:
_mtries: Computed mtries should be -1 or -2 or in interval [1,25[ but it is 55\n"
```

```
# Best model:
best_rf_model <- h2o.getModel(rf_grid_perf@model_ids[[1]])
print(best_rf_model)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: drf
## Model ID:  rf_grid_model_5
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1               2                      20            14652821        22
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        22   22.00000      13324      55013 34629.70000
##
##
## H2OMultinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("RTMP_sid_bef0_5")`
## MSE: (Extract with `h2o.mse`) 0.5981312
## RMSE: (Extract with `h2o.rmse`) 0.7733894
## Logloss: (Extract with `h2o.logloss`) 4.075192
## Mean Per-Class Error: 0.8222351
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## R^2: (Extract with `h2o.r2`) 0.8670183
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0.5    1.0   1.5    2.0    2.5     3.0    3.5     4.0    4.5     5.0
## 0.5     5428   3294  2131   5792   5818   20280  10156   21915   2653    2622
## 1.0     3287  31451  2239  32526   6899  129742  12003   85893   2616   17496
## 1.5     2597   2606  2258   7099   7859   29223  14035   28514   3003    2587
## 2.0     3795  27032  3698  51423  16189  267017  33012  219118   7777   37896
## 2.5     3317   3960  3792  14616  20049   85307  48919  110830  11995    9405
## 3.0     4057  29018  4852  71532  30139  793937  97319  771424  29627  156543
## 3.5     2728   3431  3442  15135  25791  136552 113843  355587  46799   38784
## 4.0     2266  14756  2550  42511  21374  584528 123724 1218021  77555  339478
## 4.5      759    863   892   3968   7647   45492  55447  267548  52026   58955
## 5.0      615   5141   602  12512   4387  221028  33118  609335  40206  376594
## Totals 28849 121552 26456 257114 146152 2313106 541576 3688185 274257 1040360
```

```
##         Error                    Rate
## 0.5    0.9322 =       74,661 / 80,089
## 1.0    0.9030 =     292,701 / 324,152
## 1.5    0.9774 =       97,523 / 99,781
## 2.0    0.9229 =     615,534 / 666,957
## 2.5    0.9358 =     292,141 / 312,190
## 3.0    0.6007 = 1,194,511 / 1,988,448
## 3.5    0.8466 =     628,249 / 742,092
## 4.0    0.4981 = 1,208,742 / 2,426,763
## 4.5    0.8946 =     441,571 / 493,597
## 5.0    0.7111 =   926,944 / 1,303,538
## Totals 0.6841 = 5,772,577 / 8,437,607
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.315851
## 2   2  0.558203
## 3   3  0.721811
## 4   4  0.811732
## 5   5  0.867789
## 6   6  0.897927
## 7   7  0.916060
## 8   8  0.926014
## 9   9  0.931205
## 10 10  1.000000
##
##
##
##
##
## H2OMultinomialMetrics: drf
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdout predicti
ons) **
##
## Cross-Validation Set Metrics:
## =====================
##
## Extract cross-validation frame with `h2o.getFrame("RTMP_sid_bef0_5")`
## MSE: (Extract with `h2o.mse`) 0.5876207
## RMSE: (Extract with `h2o.rmse`) 0.7665642
## Logloss: (Extract with `h2o.logloss`) 1.695709
## Mean Per-Class Error: 0.8268873
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## R^2: (Extract with `h2o.r2`) 0.8693366
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,xval = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.337626
## 2   2  0.590691
```

```
## 3   3  0.767048
## 4   4  0.861222
## 5   5  0.921858
## 6   6  0.949950
## 7   7  0.970541
## 8   8  0.983172
## 9   9  0.992089
## 10 10  1.000000
##
##
##
##
## Cross-Validation Metrics Summary:
##                                 mean         sd     cv_1_valid     cv_2_valid
## accuracy                    0.337626   0.000281       0.338064       0.337571
## auc                              NA   0.000000             NA             NA
## err                         0.662374   0.000281       0.661936       0.662429
## err_count               1192280.800000 965.288940 1190669.000000 1192902.000000
## logloss                     1.695709   0.001133       1.695560       1.695162
## max_per_class_error         0.992295   0.000251       0.991947       0.992260
## mean_per_class_accuracy     0.173114   0.000256       0.173384       0.173374
## mean_per_class_error        0.826886   0.000256       0.826616       0.826626
## mse                         0.587621   0.000132       0.587720       0.587506
## pr_auc                           NA   0.000000             NA             NA
## r2                          0.869336   0.000120       0.869493       0.869381
## rmse                        0.766564   0.000086       0.766629       0.766489
##                           cv_3_valid     cv_4_valid     cv_5_valid
## accuracy                    0.337282       0.337582       0.337630
## auc                              NA             NA             NA
## err                         0.662718       0.662418       0.662370
## err_count               1192192.000000 1192544.000000 1193097.000000
## logloss                     1.697228       1.696340       1.694257
## max_per_class_error         0.992280       0.992335       0.992652
## mean_per_class_accuracy     0.172983       0.172806       0.173021
## mean_per_class_error        0.827017       0.827194       0.826979
## mse                         0.587803       0.587546       0.587529
## pr_auc                           NA             NA             NA
## r2                          0.869162       0.869317       0.869329
## rmse                        0.766683       0.766516       0.766505
```

```
# Retrieve the variable importance
rf_varimp <- h2o.varimp(best_rf_model)
print(rf_varimp)
```

```
## Variable Importances:
##       variable relative_importance scaled_importance percentage
## 1       userId       265755.343750          1.000000   0.349936
## 2      movieId       184869.937500          0.695640   0.243429
## 3    timestamp       170862.687500          0.642932   0.224985
## 4 premier_date        83323.968750          0.313536   0.109718
## 5        Drama        19553.054688          0.073575   0.025747
##
## ---
```

```
##       variable relative_importance scaled_importance percentage
## 18 Documentary         1512.686401          0.005692   0.001992
## 19     Mystery         1279.761841          0.004816   0.001685
## 20   Film-Noir          901.571411          0.003392   0.001187
## 21     Musical          809.086731          0.003044   0.001065
## 22     Western          423.584412          0.001594   0.000558
## 23        IMAX          102.616051          0.000386   0.000135
```

```
# Check Model performance
rf_perf <- h2o.performance(best_rf_model, validation_one_hot_h2o)
rf_rmse <- h2o.rmse(rf_perf)
print(rf_rmse)
```

```
## [1] 0.7997697
```

```
# Set hyperparameter grid:
nn_hyper_grid <- list(
    hidden = list(
      c(23), # One hidden layer with 23 nodes since there are 23 features/variables
      c(23,23), # two hidden layers
      c(23, 23, 23), # three hidden layers
      c(23, 23, 23, 23), # four hidden layers
      c(23, 23, 23, 23, 23) # five hidden layers
    ),
    input_dropout_ratio = seq(0, 0.1, by = 0.01),
    rate = seq(0, 0.05, by = 0.01),
    rate_annealing = c(1e-10,1e-9,1e-8,1e-7,1e-6,1e-5)
)

# Create a Random Discrete Grid Search
nn_search_criteria <- list(
    strategy = "RandomDiscrete", # Selecting Random Search of the hyperparamter grid
    stopping_metric = "RMSE", # The metric being optimized for in model training
    seed = 42, # Setting the seed to the answer to life, the universe, and everything.
    max_runtime_secs = 120*60 # Specifying a max run-time for this command in seconds (two hou
rs)
)

# Parameters for Gradient Boost Machine
nn_random_grid <- h2o.grid(
    algorithm = "deeplearning", # Selecting the Algorithm to build the model
    grid_id = "nn_grid", # naming the grid
    x = x, # Feature  variable selection
    y = y, # Target variable selection
    nfolds = 5, # K-Fold Cross Validation
    training_frame = edx_one_hot_h2o, # Training Data selection
    hyper_params = nn_hyper_grid, # Specifying the hyperparameter grid to try
    search_criteria = nn_search_criteria # Specifying the search criteria to utilize
)
```

```
# Collect the results and sort by our models:
```

```
nn_grid_perf <- h2o.getGrid(
    grid_id = "nn_grid", # Grid of models
    sort_by = "RMSE", # Sorting the leaderboard by the desired metric
    decreasing = FALSE # Sort by Ascending/Increasing Values
)
print(nn_grid_perf)
```

```
## H2O Grid Details
## ================
##
## Grid ID: nn_grid
## Used hyper parameters:
##   -  hidden
##   -  input_dropout_ratio
##   -  rate
##   -  rate_annealing
## Number of models: 1
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##   hidden input_dropout_ratio    rate rate_annealing       model_ids    rmse
## 1     23             0.04000 0.00000        0.00000 nn_grid_model_1 0.77394
```

```
# Best model:
best_nn_model <- h2o.getModel(nn_grid_perf@model_ids[[1]])
print(best_nn_model@model[["model_summary"]])
```

```
## Status of Neuron Layers: predicting rating, 10-class classification, multinomial distributi
on, CrossEntropy loss, 1,856,018 weights/biases, 22.2 MB, 4,178 training samples, mini-batch s
ize 1
##   layer units      type dropout       l1       l2 mean_rate rate_rms momentum
## 1     1 80685     Input  4.00 %       NA       NA        NA       NA       NA
## 2     2    23 Rectifier  0.00 % 0.000000 0.000000  0.986152 0.129583 0.000000
## 3     3    10   Softmax      NA 0.000000 0.000000  0.021612 0.043037 0.000000
##   mean_weight weight_rms mean_bias bias_rms
## 1          NA         NA        NA       NA
## 2    0.000005   0.005036 -0.131607 0.437676
## 3   -0.001416   0.981400 -0.123700 0.138411
```

```
# Check model performance
nn_perf <- h2o.performance(
  best_nn_model,
  newdata = validation_one_hot_h2o)
nn_rmse <- h2o.rmse(nn_perf)
print(nn_rmse)
```

```
## [1] 0.809207
```

```
# Setup the Auto Machine Learning Modeling
aml2 <- h2o.automl(
```

```
    x = x, # Specifying the training data labels
    y = y, # Specifying the target attribute label
    training_frame = edx_one_hot_h2o, # Specifying the data to be used for training the model
    validation_frame = validation_one_hot_h2o, # Specifying the data to be used for evaluating
 the model
    stopping_metric = c("RMSE"), # The metric being optimized for in model training
    stopping_rounds = 5, # Specifying that each algorithm-iteration should be judged against t
he moving average of the last five models
    sort_metric = c("RMSE"), # Sorting the leaderboard by the desired metric
    nfolds = 0, # Normally n-folds would be != 0 in order to use cross-validation.  However, f
or AutoML, when n-folds !=0, the validation_frame is ignored
    seed = 42, # Setting the seed for reproducibility and the answer to the universe, life, an
d everything
    max_runtime_secs = 120*60, # Specifying a max run-time for this command in seconds (two ho
urs)
)
```

```
# Best model:
best_aml2_model <- h2o.get_best_model(aml2)
print(best_aml2_model)

aml <- h2o.get_best_model(aml2)
preds <- h2o.predict(aml, validation_one_hot_h2o)
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Action': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Adventure': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Animation': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Children': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Comedy': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Crime': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Documentary': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Drama': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Fantasy': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Film-Noir': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Horror': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'IMAX': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Musical': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Mystery': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Romance': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Sci-Fi': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Thriller': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'War': substituting in a column of NaN
```

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): Test/Validation
## dataset is missing column 'Western': substituting in a column of NaN
```

```
aml@model$validation_metrics@metrics$max_criteria_and_metric_scores
print(aml@model$validation_metrics@metrics$max_criteria_and_metric_scores)

perf <- h2o.performance(model = aml, newdata = validation_one_hot_h2o)
autoML_rmse <- h2o.rmse(perf)
print(autoML_rmse)
```

Results

Modeling results and performance

```
## Display a table summarizing the RSME generated by models
rmse_results <- c(glm_rmse, gbm_rmse, nb_rmse, rf_rmse, nn_rmse, autoML_rmse) # Column of metr
ic results
rmse_names <-  c("GLM", "GBM", "NB", "RF", "NN", "AutoML") # Label column for the correspondin
g metrics
rmse_df <- base::data.frame(rmse_names, rmse_results)
 # Create a dataframe of results and sort in ascending order
print(rmse_df %>% dplyr::arrange(rmse_df$rmse_results)) # Display the sorted dataframe
```

```
##   rmse_names rmse_results
## 1         NB    0.7411595
## 2     AutoML    0.7808920
## 3        GLM    0.7936825
## 4         RF    0.7997697
## 5         NN    0.8092070
## 6        GBM    0.8473865
```

```
# Close the h2o cluster
# h2o.shutdown(prompt=F)
```

Conclusion

From the dataframe of results, it is observed that the ultimate, best-performing model was Naive Bayes. The penultimate model was one generated by the AutoML function, which produced a Deep Learning Neural Network. The antepenultimate model was the Generalized Linear Model leveraging a multinomial elastic net. Despite this, the Generalized Linear Model, Random Forest, and Neural Net all performed similarity with RMSE results around 0.8. The Gradient Boost Machine finished last with the largest RMSE. Nevertheless, all models created had RMSE results below the target of 0.86490 stated in the MovieLens Capstone assignment rubric. In this regard, the limited scope of this project was achieved.

However, there can always be improvements.

Further model building and optimization may be achieved by:

- Augmenting the dataset via web scraping to add additional features such as studio, director, budget, producers, actors, filming locations, etc.
- Reducing the number of features by performed such methods as filter methods (univariate and/or multivariate scores), search methods (e.g., forward-selection, backward-selection, or recursive feature elimination), or embedded methods such as regularization (e.g., L1 and L2). This would reduce the size of the dataset and potentially reduce overfitting, thereby improving model performance. Reducing the dataset may also allow
- Try exhaustive cross-validation grid search. By using Random Search, it is possible that a localized optimization was found instead of a global optimum. Furthermore, the training process for computationally expensive algorithms was probably interrupted before many hyperparameters were evaluated. By using grid-search and extending the maximum run time, more optimal models could be generated.
- In additional to n-fold cross-validation, Leave-p-out Cross-Validation could be performed to further produce a more optimal model if more computational resources are available.
- Normalize the ratings data. Most of the models utilized are not impacted by scaling (e.g. Random Forest), but some algorithms such as Naive Bayes may yield improved performance if the data distribution was closer to a normal distribution.

- Try additional models and packages:
    - recommenderlab. This package was not used for this project because it will require a smaller dataset than the one-hot-encoded dataset utilized in this project. Feature Selection or Bagging could accomplish this.
    - Support Vector Machines
    - XGBoost (requires Linux in H2O or use of a different package(s))
    - Deep Learning Models
- Try additional packages that may accelerate computation (e.g., H2O4GPU) and/or utilize larger computational systems (e.g. cloud virtual machines)
- Evaluate the models using other metrics for classification, including a confusion matrix, and other derived metrics such as Accuracy, F1 Score, and/or Critical Success Index.

Overall, this capstone project was challenging and I learned a lot about R and the packages I used to complete the assignment.