EDX MovieLens Capstone Project

Mark145

Summer 2019

#introduction/overview/executive summary

##Introduction

Do you want insight into how internet based video services 'tailor' movie reccomendations specifically to you? Or do you want to know which aspects of your user data is being used to generate these recommendations?

Here you will be able to view the type of data used by video services and observe machine leaning techniques used to generate movie recommendations.

##Overview

This report, and associated script, devlops and preforms a machine learning algorithm on the MovieLens 10M dataset.

The MovieLens 10M dataset if found on grouplens.org. Grouplens' stated summary of the dataset is as follows:

"This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided." - http://files.grouplens.org/

##Executive Summary

The report flows from preforming data wrangling, data exploration to algorithm development. Algorithm development consists of preforming 3 averages upon the rating data found within the dataset. The first average is agnostic to any other data and is used as a starting point. After establishment of the base average, movie bias was added or subtracted based on the the movie's average rating when compared to the base average. Lastly, the user average rating was calculated and also added or subtracted from the base rating average and incorporated into the algorithm. Algorithm preformance is based on Risdual Mean Square Error (RMSE) and is tested within this report upon a training set 'edx'. The associated script file 'movieLens-algorithm-Mark145.R' preforms the algorithm upon the validation set 'validation'.

Resulting RMSEs range between .80-.85 upon the MovieLens data set.

#methods/analysis a methods/analysis section that explains the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and your modeling approach a results section that presents the modeling results and discusses the model performance

##aquire the data

The following script will generate 2 tables for us to analyze and model a machine learning algorithm against. The two tables, edx set and validation set, are considered our train and test sets respectivly. Data is pulled directly from grouplens.org and formated into two tables used as our datasets.

```
################################
# Create edx set, validation set
################################

# 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/ratings.dat")))
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieI
                                           title = 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)#, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

```r
# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
```

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

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

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

##explore the data

Now we can explore our training data. By examining the structure of our training and test sets.

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thri
```

```r
str(validation)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 113607
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)"
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Act
```

By looking at each set's structure, we see a slight difference in the dimensions. 'edx' has 9000055 objects where as 'validation' only has 999999 objects (sumed together = our expected MovieLens 10M data of 10000054). Otherwise the data is setup in the same format, with varing values between the sets. Below are some visual representations of the data.

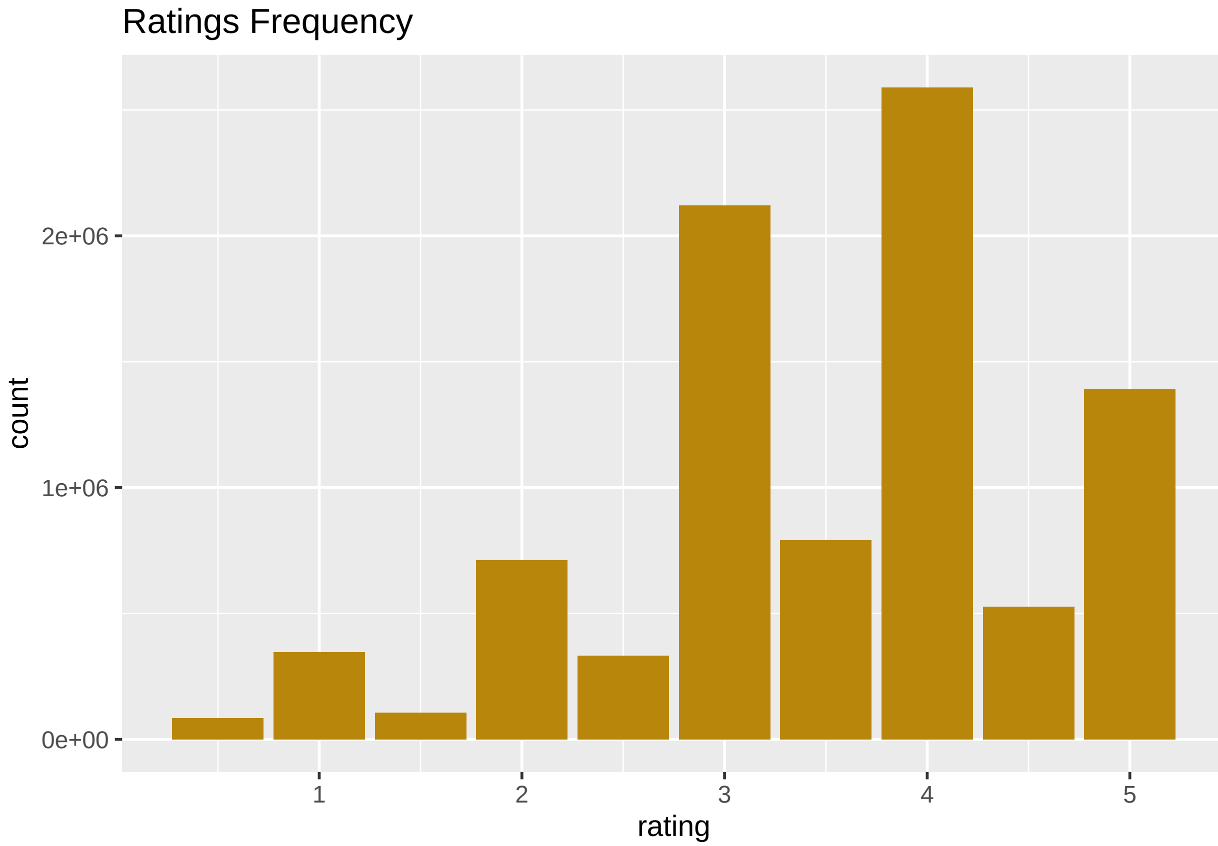
```r
dim(edx)[1]+dim(validation)[1]
```

```
## [1] 10000054
```

Our ratings are stepped in .5 increments from .5 to 5 with 5 representing a movie for which the viewer had the most praise for and a .5 value rating representing a movie for which the viewer had the least praise.

```r
ggplot(edx, aes(rating)) +
  geom_bar(fill = "darkgoldenrod") +
```

```
  ggtitle("Ratings Frequency")
```

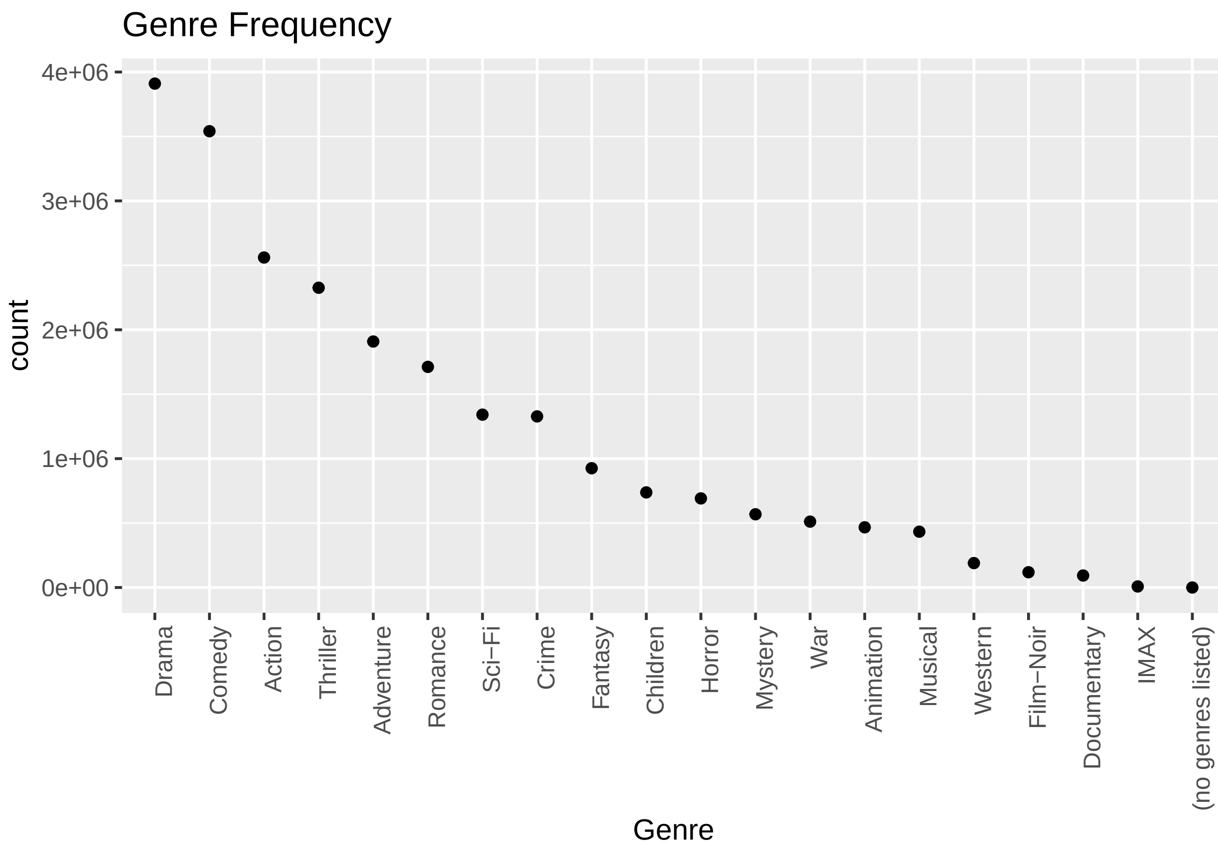


Each movie had one or more associated genres. In the plot below, the associated genres have been tallied and ordered with respect to their total associated count. This count provides insight into the genre's the users in this data set first, care to rate, and second, each genre's prevalence.

The first insight, 'care to rate', was arrived at by the assumption that while each user was required to rate 20 movies, a user would not rate a movie that the user did not watch, Therefore, the documented genre's prevalence for the data set can be considered valid.

```
genre_sep <- edx %>% separate_rows(genres, sep = "\\|") %>%
  select(genres) %>%
  group_by(genres) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
genre_sep %>%
  ggplot(aes(x = reorder(genres, -count), count)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("Genre") +
  ggtitle("Genre Frequency")
```

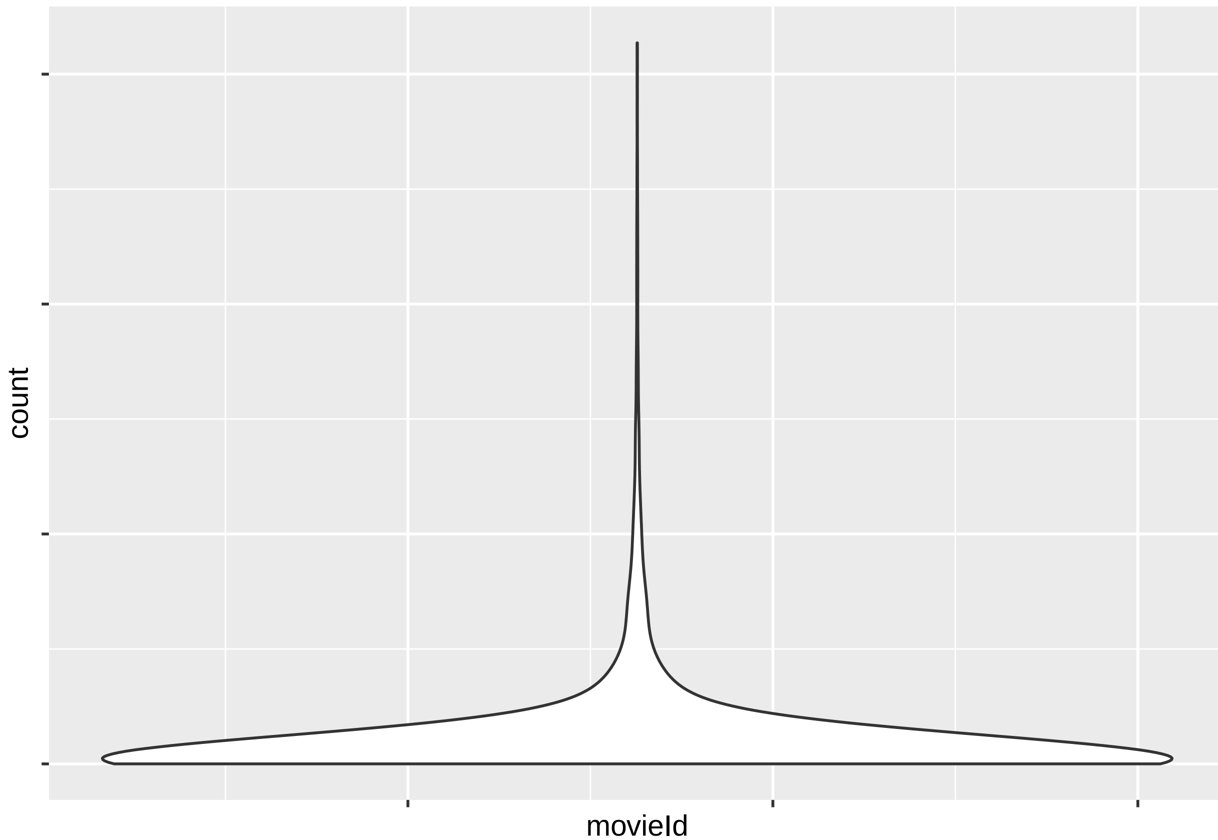


The last exploration of the data set that will be visualized is each movie's prevalence.

The chart below contains our movieIds along the x axis and the amount of times that movie was rated on the y axis. It is save to say that more than 75% of the movies have less than 5,000 ratings. Which could be valuable for start up video services. Why host the full gamut of videos when the majority of the population only watch ~500 movies? You can save on storage space, and work to optimize delivery of the top ~500 movies.

```
title_count <- edx %>%
  select(movieId, title) %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
title_count %>%
  ggplot(aes(x = movieId, count)) +
  geom_violin(trim = TRUE, bw = 1000) +
  theme(axis.text = element_blank())
```



For this use case, we will focus on optimizing an algorithm for all movies.

Our check algorithm, how well we did will be computed by 'risdual mean square error' (RMSE). We will find the difference for each user from our predicted rating for a movie, to what the user actually predicted. Then square this value and average it for all users and movies. Lastly, we will find the square root of that average and report the value. A value of 1 means 1 rating value off form reality.

For example: userID 4 rated movieID 3 as a 4 rating. We predicted (hopefully a 4) a 5 rating for userID4 on movieID3. Our check function would work like this: 5-4 = 1 1^2 = 1 mean(1) = 1 sqrt(1) = 1 RMSE = 1, not good.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

First we will compute the average rating and call this mu.

```
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512465
```

With the average rating at 3.512 our base RMSE can now be computed for us guessing this average for every user in our test set. The result of 1.06 below indicates we are off by a star if we always guess the average.

```r
RMSE(edx$rating, mu)
```

```
## [1] 1.060331
```

```r
naive_rmse <- RMSE(edx$rating, mu)
rmse_results <- data_frame(method = "Rating Average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

Now we will assume a movie can have bias (b_i). We will compute the average rating on a movie by movie basis and compaire the difference of the moivie average to the overall average (1.06). Then average this value of differences and add this into our recommendation algorithm.

```r
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs
```

```
## # A tibble: 10,677 x 2
##    movieId     b_i
##      <dbl>   <dbl>
##  1       1  0.415
##  2       2 -0.307
##  3       3 -0.365
##  4       4 -0.648
##  5       5 -0.444
##  6       6  0.303
##  7       7 -0.154
##  8       8 -0.378
##  9       9 -0.515
## 10      10 -0.0866
## # ... with 10,667 more rows
```

Now we can edit our recomendation algorithm by keeping our rating average and adding each movies average (as our bias).

```r
predicted_ratings <- mu + edx %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
```

The predicted_ratings table now contains 90000055 objects and a value. the order is tied to the order of our training data (edx) which means record one in edx has a current predicted ratining of record 1 in predicted_ratinings and accounts for overall average rating, plus or minus the bias that individual movie recieved.

```r
str(predicted_ratings)
```

```
##  num [1:9000055] 2.86 3.13 3.42 3.35 3.34 ...
```

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thri
```

```r
model_1_rmse <- RMSE(edx$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Bias added to Rating Average",
                                     RMSE = model_1_rmse))
```

Just like we did for movies, calculate each movies average rating, we can calculate each users' average rating. This will be user bias. An extreme example of user bias would be userId=1. We can see that this user really loves those 5 stars. To help offset the 'unmotivated' we will collect the average user rating.

```r
edx %>% filter(userId == 1) %>% select(userId, rating, title)
```

```
##    userId rating                                    title
## 1       1      5                         Boomerang (1992)
## 2       1      5                          Net, The (1995)
## 3       1      5                          Outbreak (1995)
## 4       1      5                          Stargate (1994)
## 5       1      5             Star Trek: Generations (1994)
## 6       1      5                   Flintstones, The (1994)
## 7       1      5                       Forrest Gump (1994)
## 8       1      5                   Jungle Book, The (1994)
## 9       1      5                     Lion King, The (1994)
## 10      1      5 Naked Gun 33 1/3: The Final Insult (1994)
## 11      1      5                             Speed (1994)
## 12      1      5               Beverly Hills Cop III (1994)
## 13      1      5                Hot Shots! Part Deux (1993)
## 14      1      5             Robin Hood: Men in Tights (1993)
## 15      1      5                Sleepless in Seattle (1993)
## 16      1      5                           Aladdin (1992)
## 17      1      5           Terminator 2: Judgment Day (1991)
## 18      1      5       Snow White and the Seven Dwarfs (1937)
## 19      1      5                    Aristocats, The (1970)
```

Finding the user's 'average' rating will be done by taking the users rating, subtracting the average rating for the whole data set (mu) and subtracting the movie bias (b_i) for each movie the user rated. We will then average this subtraction formula and call it user bias (user_avgs)

```
user_avgs <- edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

Now we will add user bias to our final algorithm. This will be done by adding the movie_avgs and user_avgs vectors to our training data (edx) and then summing the average rating (mu), movie bias (b_i) and user bias (b_u)

```
predicted_ratings <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

```
model_2_rmse <- RMSE(predicted_ratings, edx$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Movie + User Bias Model",
                                     RMSE = model_2_rmse))
```

```
rmse_results
```

```
## # A tibble: 3 x 2
##   method                              RMSE
##   <chr>                              <dbl>
## 1 Rating Average                     1.06
## 2 Movie Bias added to Rating Average 0.942
## 3 Movie + User Bias Model            0.857
```

Now our RMSE is at .85 which is a ~20% improvement over just guessing the average rating.

#conclusion

The algorithm developed is derived from a comparison of the average unique movie rating and average unique user rating compared to the overall average rating. This algorithm produces an RMSE of .8567039.

```
rmse_results
```

```
## # A tibble: 3 x 2
##   method                              RMSE
##   <chr>                              <dbl>
## 1 Rating Average                     1.06
## 2 Movie Bias added to Rating Average 0.942
```

```
## 3 Movie + User Bias Model            0.857
```

The limitation of this algorithm is the lack of regularization which would account for the extremes of the data set (that 5 star users true impact) or movies that only have 1 or 2 ratings. Future work would be to incorporate regularization and enable the algorithm's use in a movie recommendation program.