Movielens Capstone Project

Ray Z

1/17/2021

Movielens Capstone Project

Introduction

For this assignment, we will create a movie recommendation system using the 10M version of the MovieLens dataset in the in dslabs package, compiled by the GroupLens research lab. This data consists of 10 million ratings applied to 10,000 movies by 72,000 users, according to the GroupLens website. Variables in the dataset include:

1. userId
2. movieId
3. rating
4. timestamp
5. title
6. genres

The goal of this project is to predict the movie ratings of an individual with preferences as suggested by the predictor variables in the dataset.

We will train a supervised learning algorithm on the training subset to predict movie ratings in the validation set, treating this problem as a regression problem. We will first load the required packages, clean and explore the data, and then fit various supervised learning algorithms to predict a user's rating given his/her preferences. To evaluate how close our predictions are to values in the test set, we will be using the Root Mean Square Error (RMSE).

Methods/Analysis

Load Packages

```
if (!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

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

```
## Loading required package: lubridate

## Warning: package 'lubridate' was built under R version 4.0.5

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

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

## Loading required package: stringi

## Warning: package 'stringi' was built under R version 4.0.5

library(ggplot2)
library(lubridate)
library(stringi)
```

Data Cleaning

We can run the provided code to load and clean the dataset.

```
############################################################ 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

## Warning: package 'tidyverse' was built under R version 4.0.5

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1
## v purrr   0.3.4
```

```
## Warning: package 'tibble' was built under R version 4.0.5

## Warning: package 'tidyr' was built under R version 4.0.5

## Warning: package 'readr' was built under R version 4.0.5

## Warning: package 'purrr' was built under R version 4.0.5

## Warning: package 'dplyr' was built under R version 4.0.5

## Warning: package 'stringr' was built under R version 4.0.5

## Warning: package 'forcats' was built under R version 4.0.5

## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

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

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 4.0.5

## 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

## Warning: package 'data.table' was built under R version 4.0.5

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

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

```
## The following object is masked from 'package:purrr':
##
##     transpose

## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
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/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 3.6 or earlier: movies <-
# as.data.frame(movies) %>% mutate(movieId =
# as.numeric(levels(movieId))[movieId], title =
# as.character(title), genres = as.character(genres)) if
# using R 4.0 or later:
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")

# 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)`
```

```
## 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)
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")
```

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

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

Examination/Exploratory Data Analysis

Before we fit a model, we must first examine the data.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

```
n_distinct(edx$movieId)
```

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

```
n_distinct(edx$genres)
```

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

```
n_distinct(edx$userId)
```

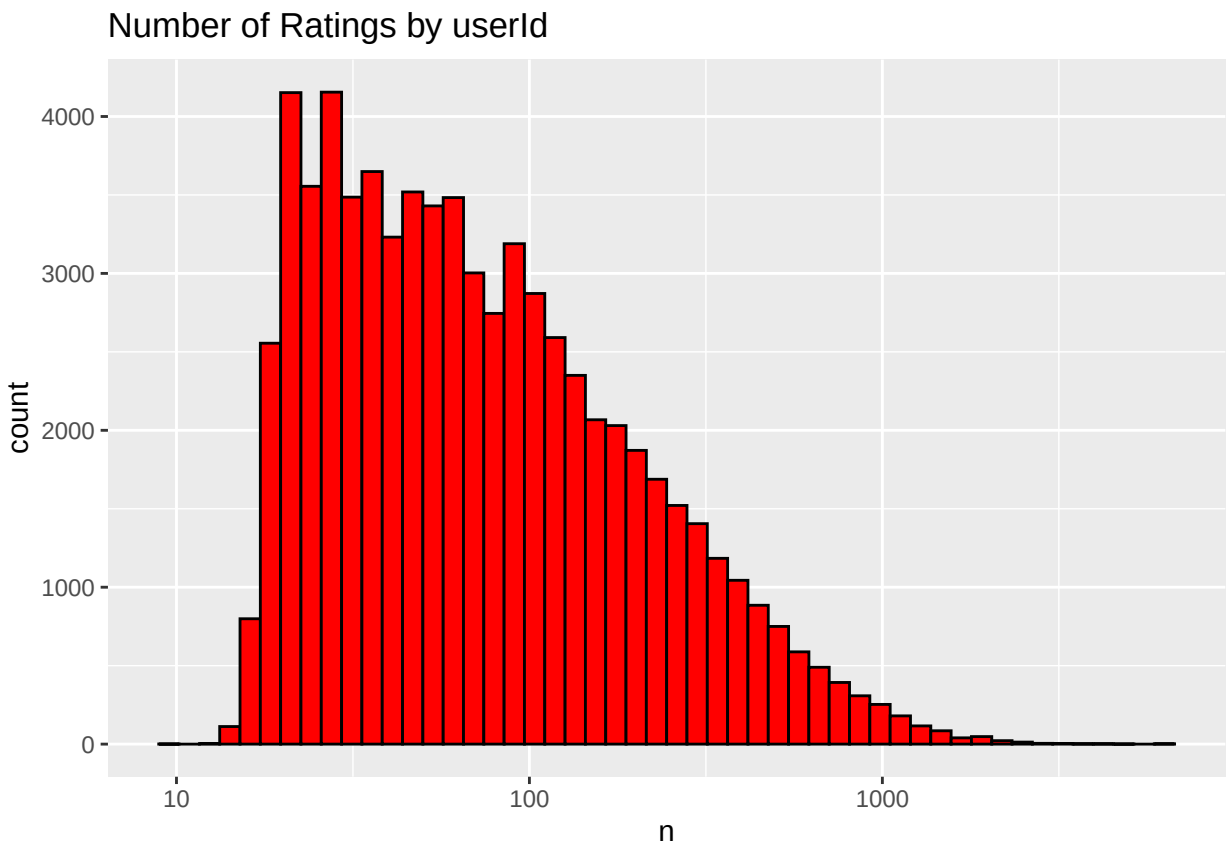
```
## [1] 69878
```

As we can see, we have a data table with these six factors: userId, movieId, rating, timestamp, title, and genres. After counting using n_distinct, we have 10,677 distinct movies, 797 distinct genres, 69,878 distinct users. Additionally, userId, movieId, timestamp, and rating are quantitative variables; title and genres are categorical.

Now we can explore relationships between the aforementioned variables using ggplot2. We can observe a few trends for each variable to help us hypothesize which variables are most predictive of movie preference.

userId

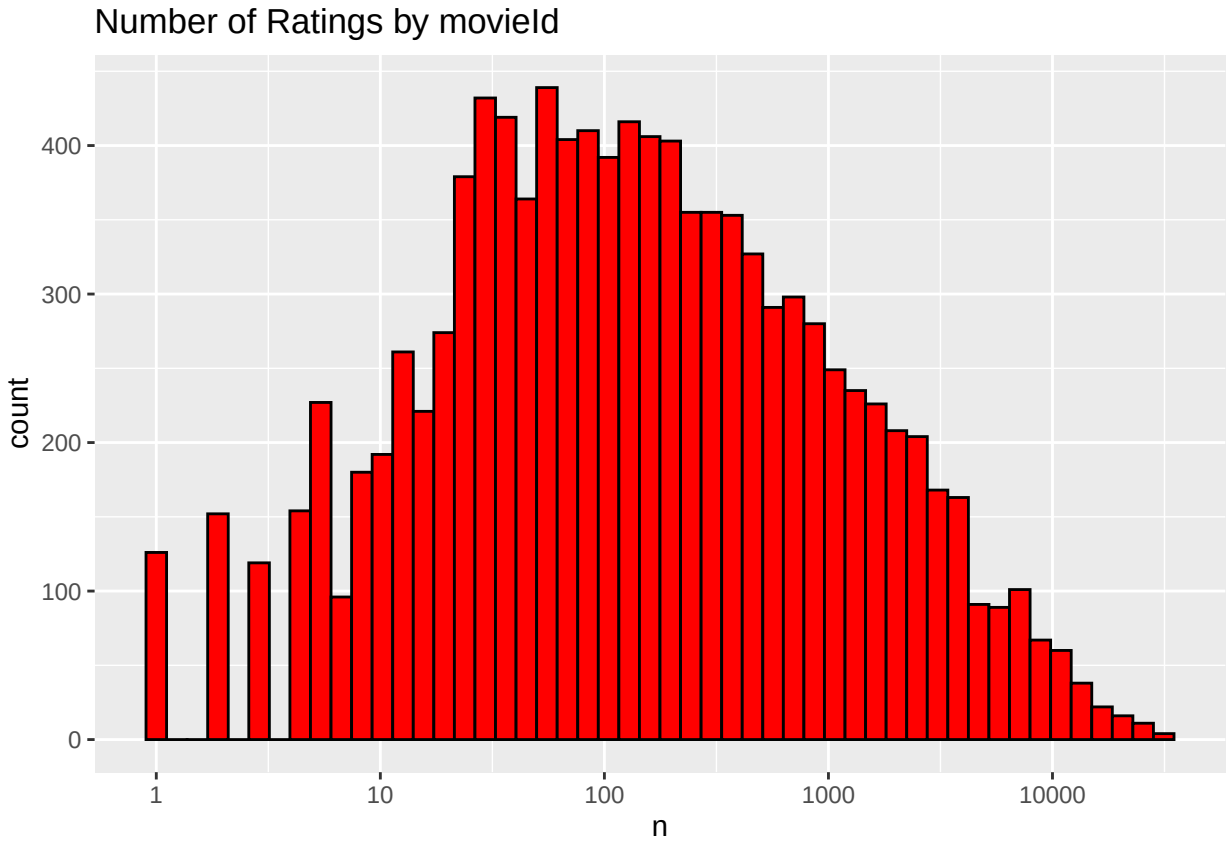
```
edx %>%
    count(userId) %>%
    ggplot(aes(n)) + geom_histogram(bins = 50, fill = "red",
    color = "black") + scale_x_log10() + ggtitle("Number of Ratings by userId")
```



The histogram above is skewed right. This could suggest a few relationships. It is possible that some users use the rating site to only post a few good or bad reviews. Some users might only post ratings when they really like or dislike a movie. Some users see more movies than others, and their ratings might be important for the prediction.

movieId

```
edx %>%
    count(movieId) %>%
    ggplot(aes(n)) + geom_histogram(bins = 50, fill = "red",
    color = "black") + scale_x_log10() + ggtitle("Number of Ratings by movieId")
```



Some movies in the histogram seem to be outliers as they have only been rated a few times, and could be less significant than those with more reviews. Note that this is log10 scaled.

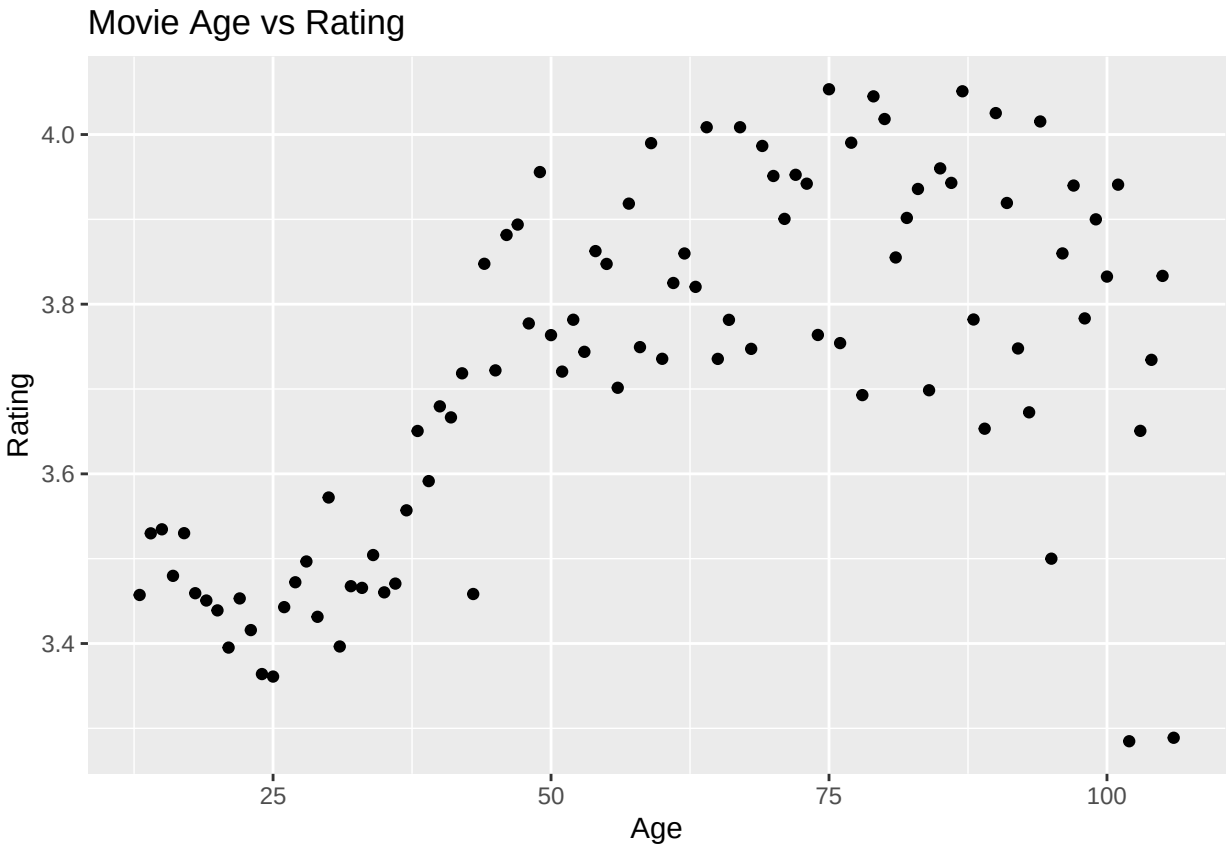
timestamp

We can use the stringi package and regex to locate the year sequence in the title column of each instance. Since some titles have parenthesis/years, we can use str_sub (substring) additionally to make it more accurate. Let's do this for both the test set and the train set.

```
edx <- mutate(edx, age = 2021 - as.numeric(stri_extract(str_sub(edx$title,
    -5, -1), regex = "(\\d{4})", comments = TRUE)))
```

```
validation <- mutate(validation, age = 2021 - as.numeric(stri_extract(str_sub(validation$title,
    -5, -1), regex = "(\\d{4})", comments = TRUE)))

edx %>%
    group_by(age) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(age, rating)) + labs(title = "Movie Age vs Rating",
    x = "Age", y = "Rating") + geom_point()
```

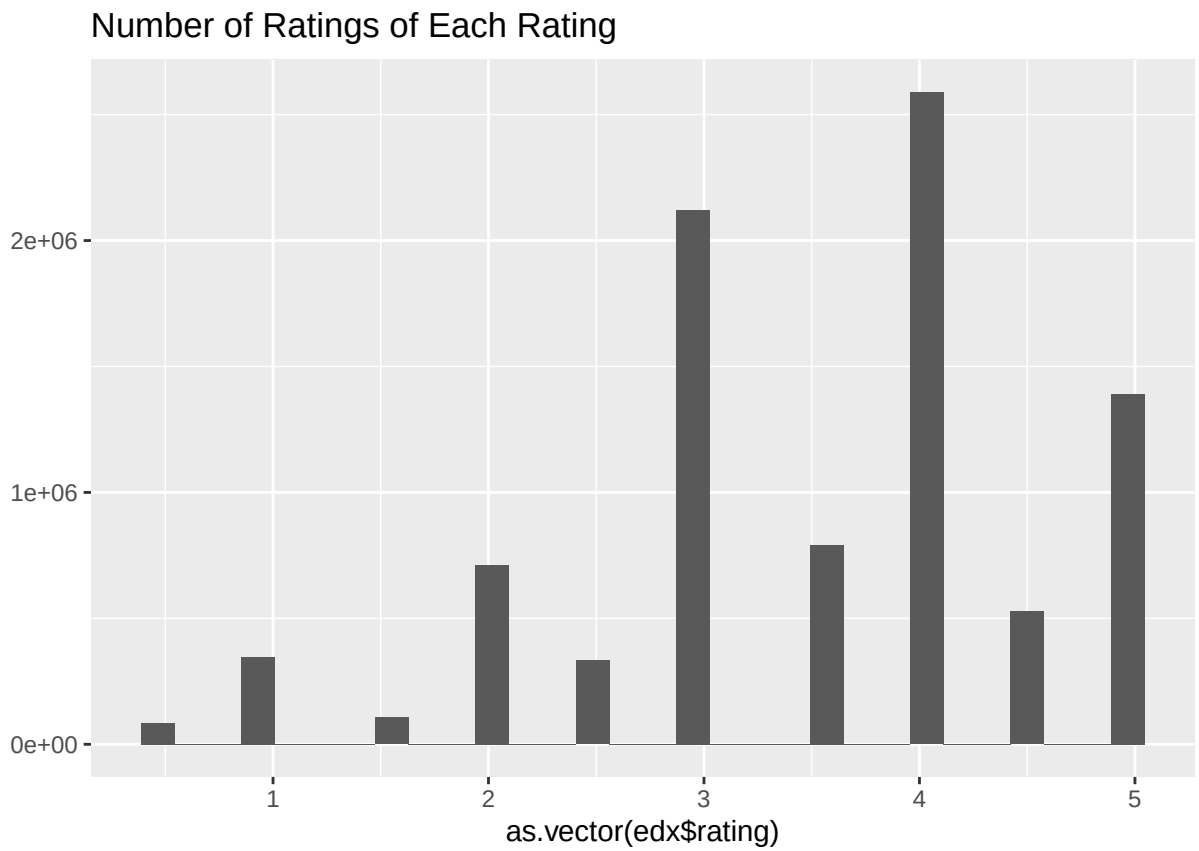


It appears from this plot that more recent movies are generally rated lower than older ones. Additionally, there are a few outliers in which older movies were rated lower. This seems to be a fairly significant trend.

rating

```
qplot(as.vector(edx$rating)) + ggtitle("Number of Ratings of Each Rating")
```

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

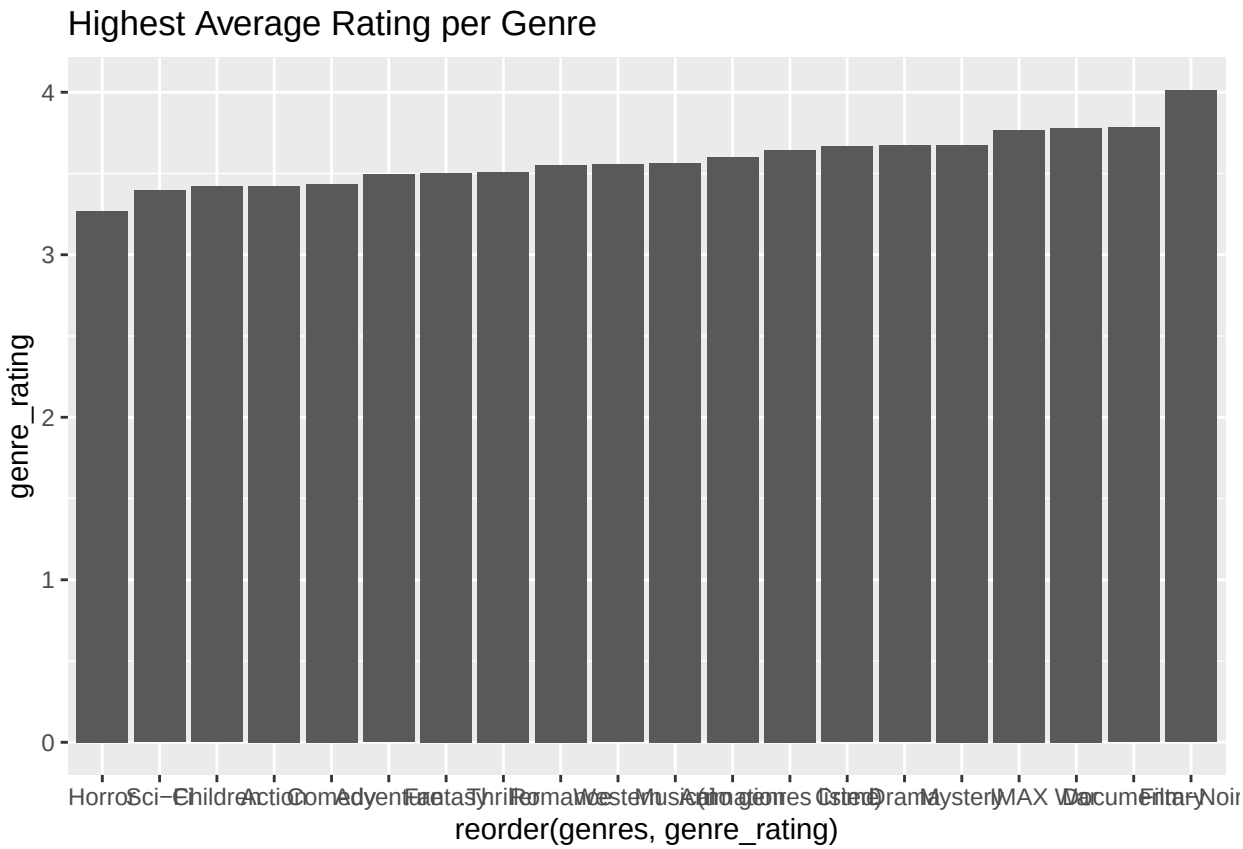


The majority of the movies fall under whole numbers ranging from 0.5 to 5 (inclusive), although there are a few increments of 0.5.

genres

Genre is a categorical variable with many different states. It can be hypothesized that users' interest in a genre would affect their rating. We will examine the distribution of rankings below.

```
edx %>%
    separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarize(genre_rating = mean(rating)) %>%
    arrange(desc(genre_rating)) %>%
    ggplot(aes(reorder(genres, genre_rating), genre_rating)) +
    geom_bar(stat = "identity") + ggtitle("Highest Average Rating per Genre")
```



Some movies appear to be more popular than others, which we can take into account in our analysis.

Measurement of Accuracy

For this recommendation system, we will be using RMSE to gauge the accuracy of our fitting algorithm. We can define a method that will do just this.

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

Modeling Approach - Regression Problem

We will use a regression approach toward this regression recommender system. RMSE will penalize larger deviations from the true value, so we will try to minimize this.

For our first model, we can first calculate d, which is simply the mean of all reviews, and then calculate the movie effect; movies that are liked a lot by other users are likely to be liked by a new user, which is an example of user-based collaborative filtering. We will incorporate item-based collaborative filtering in our second model which adds the effect of the specific user based on his preferences of other movies. As this employs the ratings of other movies of a specific user to predict that user's preference of a new movie, this is content-based. In our third model, we will use the timestamp variable to predict the effect of age on the user's preference of the movie. As this uses a feature based on the movie itself, this is content based filtering, as it uses features of the item itself.

Model 1 - Mean + Movie Effect

Let's begin by calculating the mean.

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

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

The mean is 3.5125.

```
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(movie_effect = mean(rating - d))
b_i %>%
    qplot(movie_effect, geom = "histogram", data = ., fill = I("red"),
        col = I("black"), main = "Distribution of Movie Effect",
        xlab = "Movie Effect", ylab = "Count")
```

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

This appears to be skewed left and unimodal. Some movies have a strong negative effect on ratings.

Now we can apply this model to the validation set and calculate the RMSE to determine the accuracy of the model.

```r
predicted_ratings_model_1 <- d + validation %>%
    left_join(b_i, by = "movieId") %>%
    pull(movie_effect)

rmse_model_1 <- RMSE(predicted_ratings_model_1, validation$rating)
rmse_model_1
```

```
## [1] 0.9439087
```

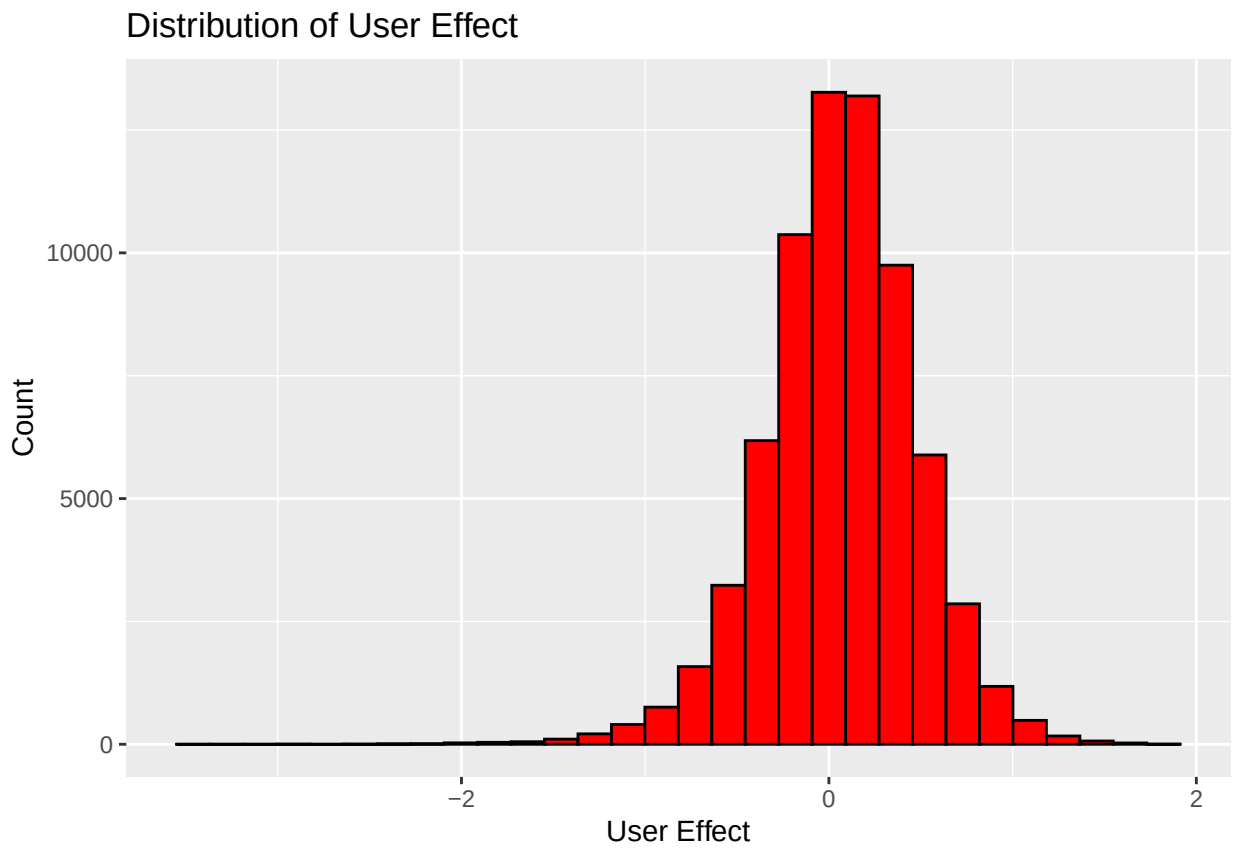
This baseline modeling approach using only the movie effect gives and error of 0.9439. We can decrease this by incorporating more effects.

Model 2 - Mean + Movie Effect + User Effect

Different users can also add bias. Some users rate movies worse or better, and we will incorporate this into the second model. We will first generate the user effect due to their past reviews.

```r
b_u <- edx %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(user_effect = mean(rating - d - movie_effect))
b_u %>%
    qplot(user_effect, geom = "histogram", data = ., fill = I("red"),
        col = I("black"), main = "Distribution of User Effect",
        xlab = "User Effect", ylab = "Count")
```

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



This appears to be unimodal, suggesting that there exists users who rate both above and below the mean roughly evenly. We can see whether the user effect has an impact on the RMSE in the code below.

```
predicted_ratings_model_2 <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(prediction = d + movie_effect + user_effect)

rmse_model_2 <- RMSE(predicted_ratings_model_2$prediction, validation$rating)
rmse_model_2
```

```
## [1] 0.8653488
```

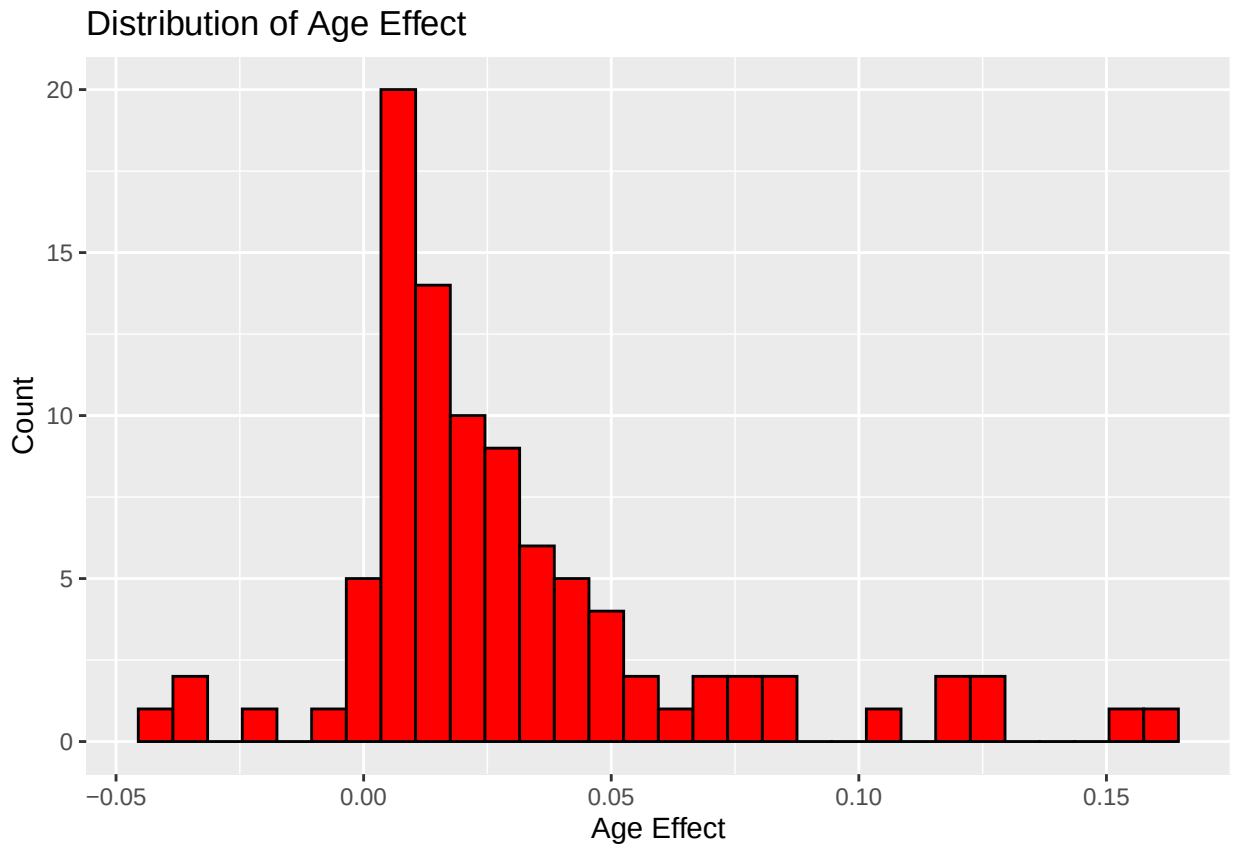
Model 3 - Mean + Movie + User + Age effects

The age of the movie can also influence the rating of the movie, as suggested in the scatterplot of age versus rating above.

```
b_a <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(age) %>%
    summarize(age_effect = mean(rating - d - movie_effect - user_effect))
b_a %>%
    qplot(age_effect, geom = "histogram", data = ., fill = I("red"),
```

```
        col = I("black"), main = "Distribution of Age Effect",
        xlab = "Age Effect", ylab = "Count")
```

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



This distribution is skewed to the right with a mean above 0. Movies with greater age tend to be rated more favorably.

```
predicted_ratings_model_3 <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by = "age") %>%
    mutate(prediction = d + movie_effect + user_effect + age_effect)

rmse_model_3 <- RMSE(predicted_ratings_model_3$prediction, validation$rating)
rmse_model_3
```

```
## [1] 0.8650043
```

We see some improvement by factoring in the age, however the RMSE (given a distribution from 0.5 to 5 stars) of 0.865 is still rather high. So far, we have been fitting multivariate regressions to the edx set. Let's try some more advanced algorithms.

Model 4 - Regression Tree

As the relationship between our predictor variables and response variable is likely nonlinear, we can apply the non linear method of classification and regression trees (CART). As we are trying to predict the ordinal variable of movie ratings, a regression tree is suitable for this task, and we will use the rpart package for recursive partitioning of a decision tree. We can first build and prune the tree given an initial complexity parameter 0.001 (keep creating splits if the r^2 value increases more than this parameter). The optimum parameter will then be found and used to prune the tree.

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

```
## Loading required package: rpart
```

```
library(rpart)

# build a rpart decision tree noting that the control
# continues to split if the increase in r^2 is greater than
# the complexity parameter cp
model_4 <- rpart(rating ~ userId + movieId + age, control = rpart.control(cp = 0.001),
    data = edx)
bestcp <- model_4$cptable[which.min(model_4$cptable[, "xerror"]),
    "CP"]  # identify the best complexity parameter
model_4 <- prune(model_4, cp = bestcp)  # produce pruned tree based on the best complexity parameter
```

We can then plot the splits of the tree using the rpart.plot package.

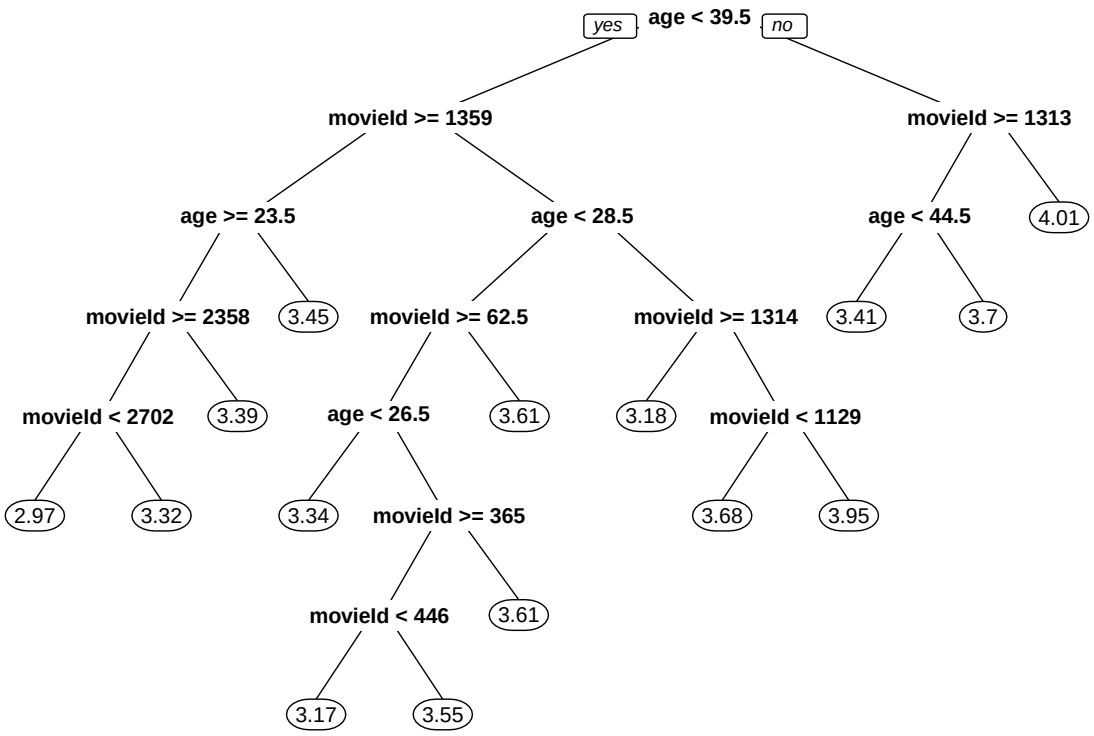
```
if (!require(rpart.plot)) install.packages("rpart.plot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart.plot
```

```
## Warning: package 'rpart.plot' was built under R version 4.0.5
```

```
library(rpart.plot)

# plot the rpart model using the function in rpart.plot
prp(model_4, roundint = F, digits = 3)
```



We will use this model to predict ratings for the validation data and analyze the RMSE.

```
dt_pred <- predict(model_4, newdata = validation)
rmse_model_4 <- RMSE(dt_pred, validation$rating)
rmse_model_4
```

```
## [1] 1.038318
```

This nonlinear model performed rather poorly compared to the previous regressions, despite the tree having many splits of different variables.

Model 5 - Matrix Factorization Using Recosystem

Another collaborative filtering mechanism is matrix factorization, which factorizes the data frame we are working with into the product of two matrices with lower rank. We can use the recosystem recommender system package for this. For the opts parameter in recosystem's train function, we can control the number of latent factors, the regularization parameter lambda, number of iterations, and more. Too few latent factors will not be able to predict the validation set, while too many would likely overfit, so we will go with 20. We can go with the default values of L1 and L2 regularization, which address overfitting. We can leave the number of iterations default to avoid taking too long on slower computers. We will set verbose = FALSE to avoid showing too much detailed information.

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

```
## Loading required package: recosystem
```

```
## Warning: package 'recosystem' was built under R version 4.0.5
```

```
library(recosystem)

r <- Reco()
train_data <- data_memory(edx$userId, edx$movieId, edx$rating)
test_data <- data_memory(validation$userId, validation$movieId,
    validation$rating)
r$train(train_data, opts = list(dim = 20, verbose = FALSE))
mf_pred <- r$predict(test_data, out_memory())
rmse_model_5 <- RMSE(mf_pred, validation$rating)
rmse_model_5
```

```
## [1] 0.8203789
```

This RMSE is significantly lower than the other methods. By using latent factors and factorizing the outcome matrix, we were able to get an accurate recommender system with a RMSE under our target value of 0.8649.

Results

Let's generate a table summarizing the RMSE of all the models.

```
tab <- matrix(c(rmse_model_1, rmse_model_2, rmse_model_3, rmse_model_4,
    rmse_model_5), ncol = 1, byrow = TRUE)
colnames(tab) <- "Root Mean Squared Error"
rownames(tab) <- c("Movie Effect", "User + Movie Effect", "User + Movie + Age Effect",
    "Decision Tree", "Matrix Factorization")
tab <- as.table(tab)
tab
```

```
##                            Root Mean Squared Error
## Movie Effect                             0.9439087
## User + Movie Effect                      0.8653488
## User + Movie + Age Effect                0.8650043
## Decision Tree                            1.0383181
## Matrix Factorization                     0.8203789
```

Initially we used the expected value of all ratings (3.5125) and mean rating of a movie to generate a predicted value of a rating, giving us a RMSE of 0.94. We then added the mean ratings of individual users giving us the RMSE of 0.86. After adding the age effect the RMSE decreased slightly but remained at 0.86 after rounding. Using all of these predictive variables combines user-based and item-based collaborative filtering, as well as content-based filtering; ratings are suggested as a function of how all users rated a movie (movie effect/item-based collaborative filtering), as a function of how the user likes movies in general (user effect/user-based collaborative filtering), and as a function of the movie itself (content-based). We threw in some more complex algorithms used for regression type problems, including decision tree and matrix factorization, resulting in our lowest RMSE being found using matrix factorization.

Conclusion

In this project, we used the 10M MovieLens dataset and fitted a recommendation system on the edx test set. We measured the accuracy of algorithms tested using the root mean square error. We trained a

recommendation system based on the movie effect, movie + user effects, movie + user + age effects, regression tree, and matrix factorization. We were able to generate a final RMSE of 0.82, far lower than our target value of 0.8649.

Movie recommendation systems have been a textbook example of recommender systems, publicized particularly by the Netflix Prize. While we worked with four main factors in this inquiry: "userId", "movieId", "rating", "timestamp" further work toward movie recommender systems could evaluate the "title" and "genres", perhaps performing text mining or encoding genres to be analyzed as categories. Further analysis could also look at the relationship between various subcategories of movies (i.e. some movies could be longer/shorter or there could be sequels that users enjoy).