MovieLens Project Submission

Kirtimay Pendse

6/15/2020

Introduction

This project is part of the HarvardX: PH125.9x Data Science: Capstone course[1], the final course for the Data Science Professional Certificate. This project is centered around creating a movie recommendation algorithm using a subset of the MovieLens dataset.

The algorithm was developed using a training set (referred to as the 'edx' set) and was tested on the 'validation' set; then, the random mean squared error (RMSE) was calculated to evaluate the proximity of the predictions generated by the algorithm to the true values in the validation set. This report is split into four sections: first, the objective and motivation behind the project is highlighted, then exploratory data analysis is conducted, following which the modeling approach to develop the predicted movie rating algorithm is presented. Finally, the modeling resutls are presented along with a discussion on the algorithm's performance and its limitations.

Objective

The MovieLens dataset[2] is compiled by the GroupLens research group at the University of Minnesota. The dataset contains over 20 million ratings for more than 27,000 movies with around 140,000 users. The overall aim of this project is to develop an algorithm that can predict user ratings (ranging from 0.5 to 5) in the validation set using an algorithm created via the 'edx' dataset.

The algorithm's performance is evaluated using the RMSE, a measure of the difference between the model's predicted values and the observed values which can be written as the following function[3], where u represents a user, i represents a movie and N is the total number of user/movie combinations:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

An 'accurate' model entails obtaining an RMSE of < 0.86490.

Methods and Analysis

Preparing the data

First, the dataset is downloaded from the MovieLens website and split into a training set (the 'edx' set) and a 'validation' set. The 'edx' dataset is further split into a train set and a test set, and is used to create the prediction algorithm. Once the final model reaches the desired RMSE, it is tested on the 'validation' dataset for a final validation.

[1]https://courses.edx.org/courses/course-v1:HarvardX+PH125.9x+1T2020/course/
[2]https://grouplens.org/datasets/movielens/latest/
[3]Taken from Prof. Irizarry's "Introduction to Data Science", section 33.7.3

```r
#Loading required packages
library(lubridate)
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(scales))
  install.packages("scales", repos = "http://cran.us.r-project.org")
library(dplyr)
library(knitr)
library(ggplot2)
library(dslabs)
library(lubridate)

#The following code was given by the edX staff

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

# 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))[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") #use set.seed(1) if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>% #validation will be our test set
  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)
edx <- rbind(edx, removed) #edx is training set

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

The 'edx' dataset was split into a train set (90% of the data in edx) and a test set (10% of the data in edx) using the following code-

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

#To ensure userId and movieId in the test set are also in the train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

#Adding rows removed from the test set back
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
rm(test_index, temp, removed)
```

Exploratory Analysis

For the initial data exploration, the head() function was used to get a broad understanding of the data.

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

From the data frame above, it's observed that the dataset contains 6 variables, with each user and movie having a unique user ID and movie ID respectively, along with a rating assigned to each user-movie pair, and a timestamp. It's also evident that the title of each movie has the year in brackets, and multiple genres can be assigned to a movie.

Then, the class for each variable was determined.

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "numeric"   "numeric"   "integer" "character" "character"
```

It's observed that 'userId' and 'timestamp' are integers, 'movieId' and 'rating' are numeric, and 'title' and 'genres' are characters.

A summary of the edx dataset was also determined:

```
##      userId         movieId          rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :7.90e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.00   1st Qu.:9.47e+08
##  Median :35738   Median : 1834   Median :4.00   Median :1.04e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1.23e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

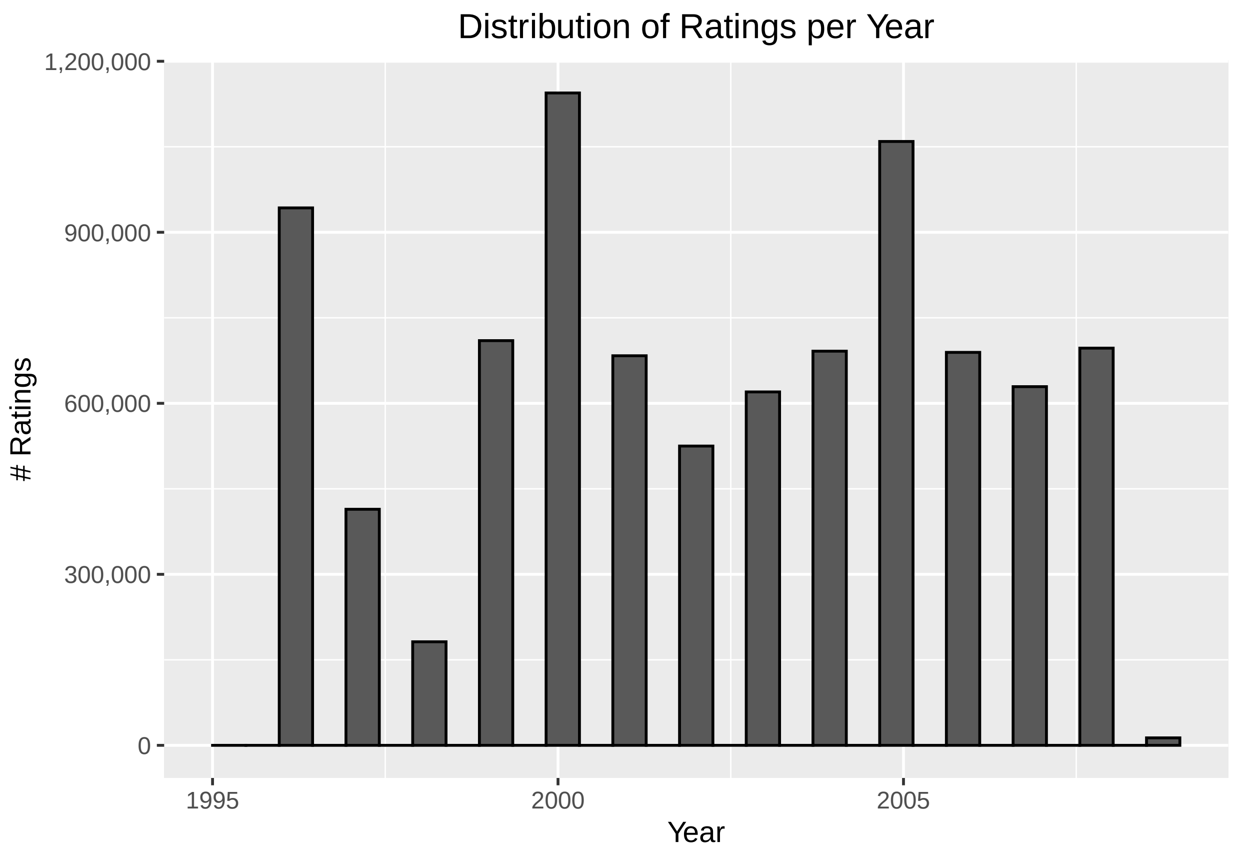
The initial observations made from the table above are that the mean rating is around 3.5, indicating a possibility that most users tend to rate movies slightly higher, and that there doesn't seem to be any missing values.

Plots

The following section inspects the variables and their relationships in more detail through some plots.

Date

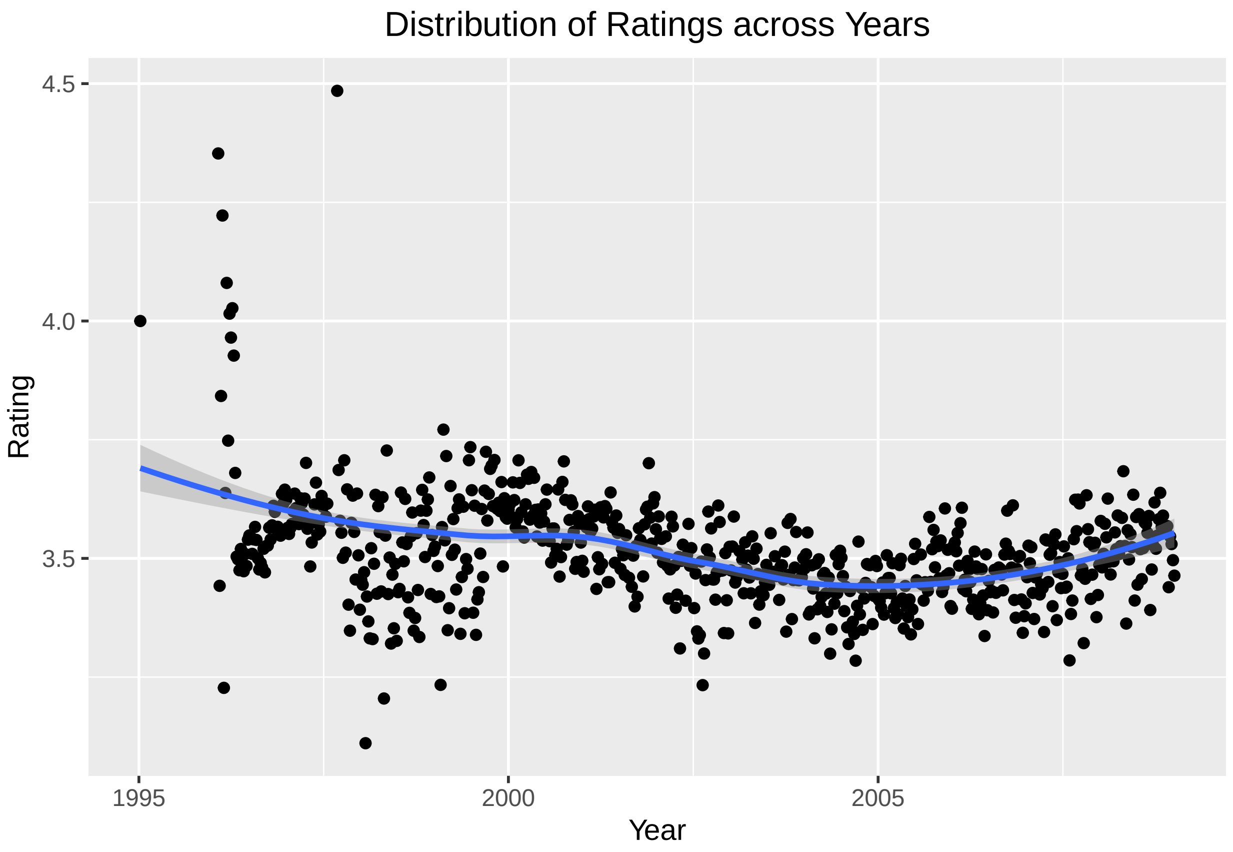
From the plot below, it's evident that the movies with the most ratings came out in 2000 and then 2005. As expected, the movies released most recently tend to have fewer ratings, but there are a few years (such as 1997 and 2002) where movies have fewer ratings.



Time

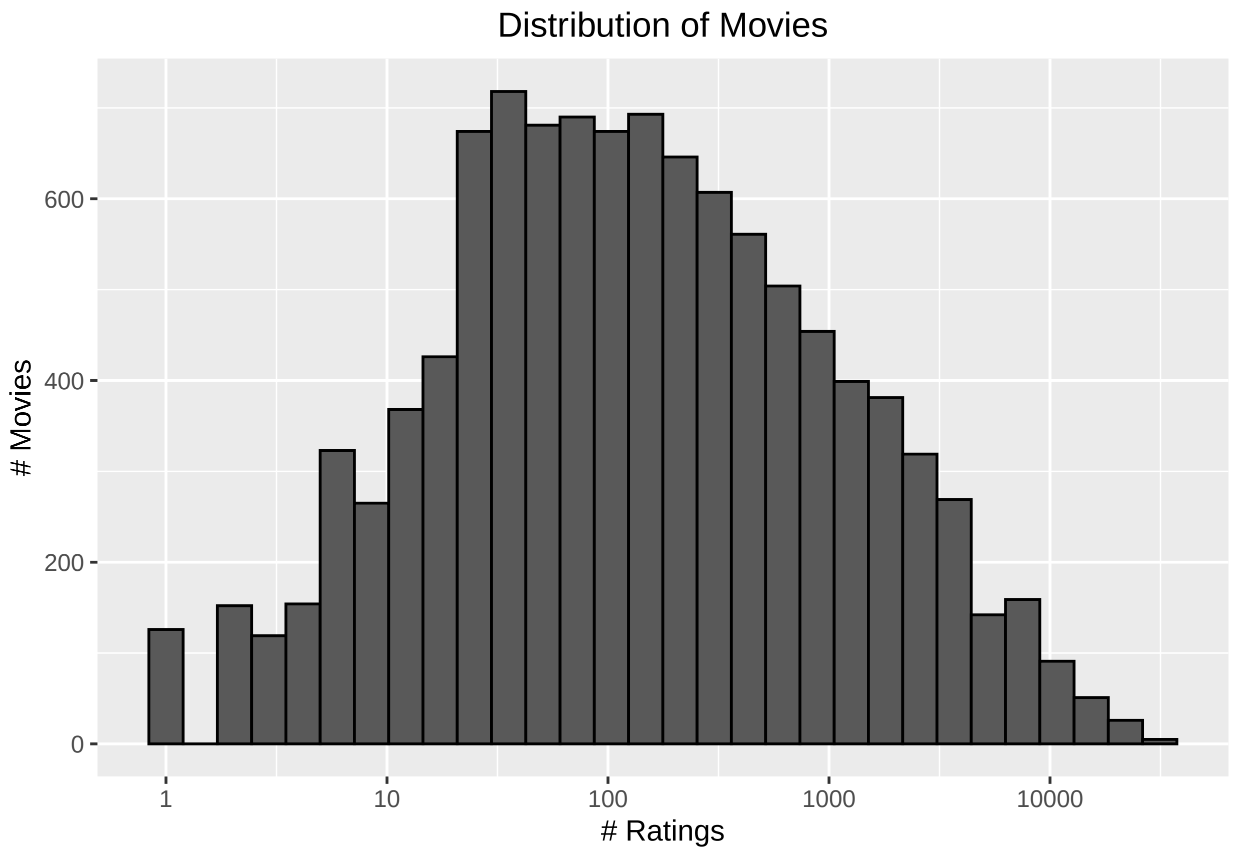
In the MovieLens dataset, the 'timestamp' variable represents the time and date at which ratings were provided, captured in seconds since January 1, 1970. The as_datetime functio in the 'lubridate' package allows users to meaningfully view the time stamp. The plot below shows that there's limited evidence of a time effect when looking at the average rating against date, and that the subsequent models built do not necessarily need to include a time effect.

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



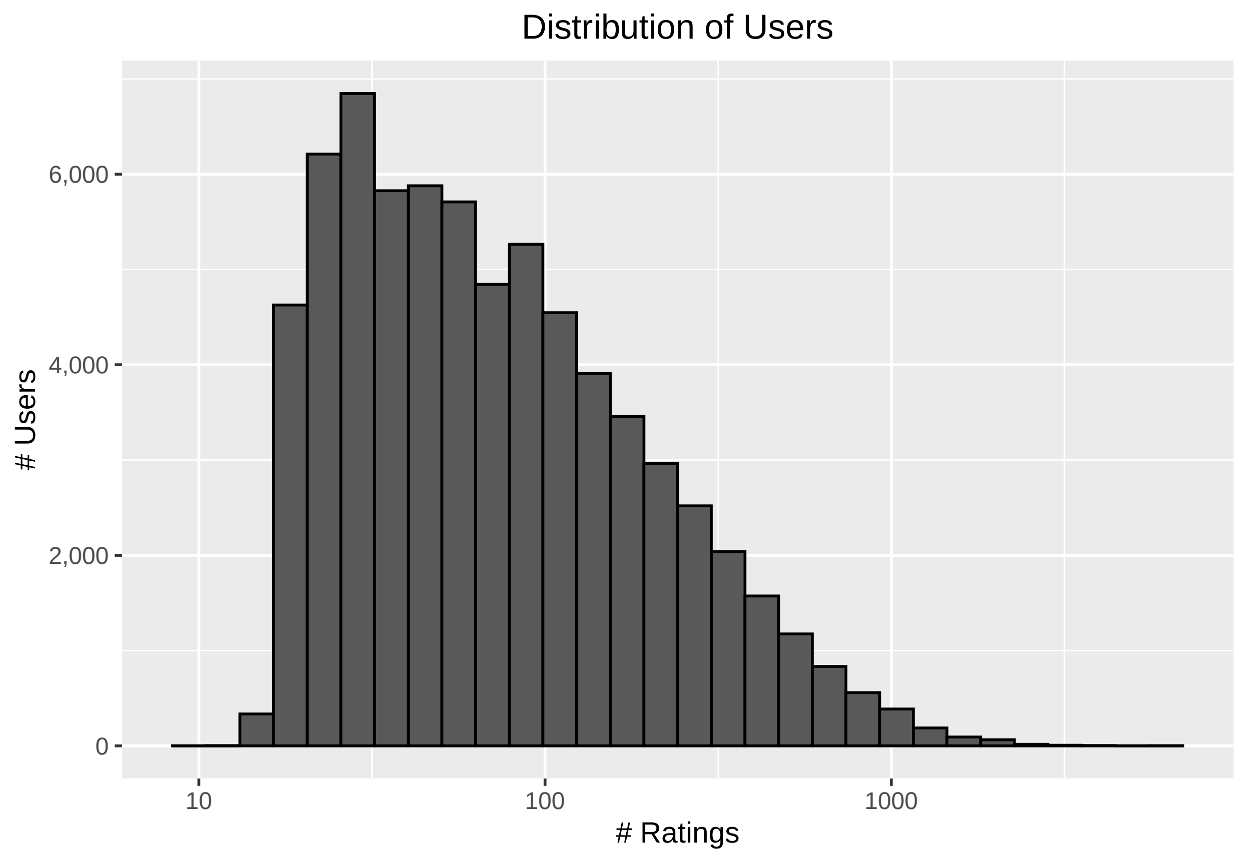
Movies

In the plot below, it can be seen that some movies are rated much more frequently than others, and that there's a few movies with as few as one rating. Thus, a movie effect capturing this movie to movie variability is included in the models built. It's important to note that low numbers of ratings could result in faulty estimates for the predictions made by the model, and is explained further whilst regularizing the final linear model.



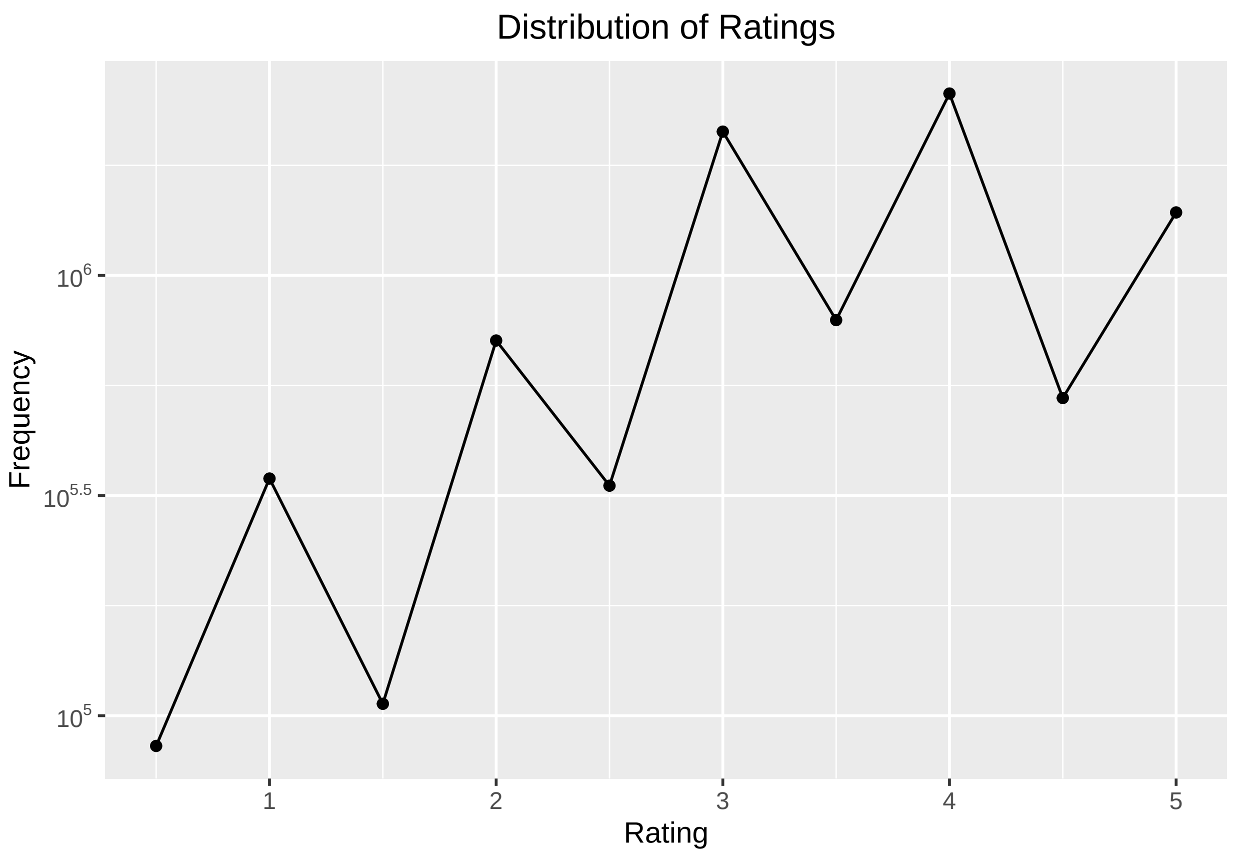
Users

In the plot below, it's evident that most of the users have rated between 40 to 110 movies, and some users are very active and have rated in numbers close to a thousand. This highlights the importance of adding a user effects term in the models built in the next section.



Ratings

As highlighted in the summary statistics and the graph below, users tend to be more positive than critical while rating movies. 4, 3, and 5 (in that order) are the most common ratings given, and half point ratings are not as common as whole numbers. 0.5 is the rating least commonly assigned to a movie.



Modeling

To build an efficient recomemndation system algorithm, the most important features that would help predict a user's rating need to be identified. The analysis begins with an initial model that randomly predicts a user's rating based on the observed probabilities in the training subset of the edx dataset, followed by a simple model built around the mean of the observed values, following which three linear models are built- one accounting for user effects, one accounting for movie effects, and one accounting for both movie and user effects. Then, a regularization parameter is added to the movie and user effects model, and a final validation (on the validation set), using the regularized movie and user effects linear model, is performed. As mentioned earlier, RMSE values are used to evaluate each model's performance.

The RMSE is calculcated using the following code:

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

Predicting Randomly

In this simple model, user ratings are randomly predicted using the observed probabilities in the training subset of the edx dataset. After calculcating the probability of each rating, the model predicts ratings for the test subset and the RMSE compares the predictions with the actual rating. This model should produce the worst RMSE of all models generated, and mostly serves as a stepping stone towards creating a more accurate model. This model uses a Monte Carlo simulation to approximate the rating distribution, since the real distribution of ratings for the entire population is unknown.

Building Linear Models

1. The Average Rating Model

The first linear model is centered around the idea that users will ascribe the same rating to all movies, and that variation within movies is accounted in a randomly distributed error term. This can be written as the following equation:

$$\hat{Y}_{u,i} = \mu + \epsilon_{i,u}$$

$\hat{Y}$ represents the predicted user rating, μ represents the mean of the observed data (since the average minimizes RMSE, this model uses the predicted rating as the mean of observed ratings), and $\epsilon_{i,u}$ is an error term.

2. Including User Effects

Users tend to have different rating distributions; for instance, some users may like most of the movies they rate and consistently give them high ratings, whereas some users could be more critical and give movies lower ratings. The user effect (b_u) can be captured through this linear model:

$$\hat{Y}_{u,i} = \mu + b_u + \epsilon_{i,u}$$

3. Including Movie Effects.

Similar to the user effect, variability within movies can be explained through different rating distributions for different movies; intuitively speaking this makes sense, as some movies are very popular and garner huge public support, whereas some artsy, independent movies have limited support, and might have lower ratings due to being 'unpopular'. The following equation highlights a linear model accounting for the movie effect (b_i:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{i,u}$$

4. Including User and Movie Effects

A more accurate model would account for both user and movie effects, and can be written as the following equation:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Using Regularization

The linear models created above provide reasonable user rating predictions, but fail to account for the fact that many movies have a small number of ratings, and that some users have rated a very small number of movies. Having these small sample sizes for movie-user combinations can lead to large estimated errors, and so a penalty term should be added. Regularization, a form of regression that 'regularizes' (constrains) estimates to zero, allows for penalizing these small sample sizes and also helps in avoiding overfitting (creating an algorithm that functions extremely well on a training set but doesn't generalize well on additional data).

To regularize movie and user effects, a penalty term λ is used; λ can be thought of as a tuning parameter, and so can be ascribed a range of values from which the best value (the one that minimizes RMSE) will be picked. The penalized movie and user effects can be written as the following respectively:

$$\hat{b}_i = \frac{1}{n_i + \lambda} \sum_{u=1}^{n_i} (y_{u,i} - \hat{\mu})$$

$$\hat{b}_u = \frac{1}{n_u + \lambda} \sum_{i=1}^{n_u} (y_{u,i} - \hat{b}_i - \hat{\mu})$$

Values of n_i and n_u that are smaller than or close to λ, $\hat{b}_i$ and $\hat{b}_u$ are 'regularized' and will be smaller than their original values, whereas for larger values of n_i and n_u the estimates do not change by much. Simulations are run using several values of λ, and the optimal value for λ is selected as the one that minimizes the RMSE.

Results

Predicting Randomly

After setting a seed, a Monte Carlo simulation is run 10,000 times on a 100 samples, and the observed probabilities are used to randomly predict user ratings.

```
set.seed(43, sample.kind = "Rounding")

#To create a probability distribution of each rating
p <- function(x, y) mean(y == x)
rating <- seq(0.5,5,0.5)

#Using a MC simulation to estimate the probability
B <- 10000
M <- replicate(B, {
  z <- sample(train_set$rating, 100, replace = T)
  sapply(rating, p, y=z)
})
prob_1 <- sapply(1:nrow(M), function(x) mean(M[x,]))

#To randomly predict ratings
y_hat <- sample(rating, size = nrow(test_set),
                       replace = TRUE, prob = prob_1)

#Store results in a table
results_table <- tibble('Method of Analysis' = "Desired RMSE", RMSE = 0.8649)
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis' = "Predicting Randomly",
                           RMSE = RMSE(test_set$rating, y_hat)))
results_table
```

```
## # A tibble: 2 x 2
##   `Method of Analysis`  RMSE
##   <chr>                <dbl>
## 1 Desired RMSE         0.865
## 2 Predicting Randomly  1.50
```

The RMSE of this model is extremely high, almost 1.5, and the following models would bring the value down closer to the desired RMSE.

Linear Models

1. The Average Rating Model

```
#Calculating the mean of observed values in the train_set
mu <- mean(train_set$rating)

#Adding the results to the RMSE table
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis'= "Using the Average Rating",
                           RMSE = RMSE(test_set$rating, mu)))
results_table
```

```
## # A tibble: 3 x 2
##   `Method of Analysis`      RMSE
##   <chr>                    <dbl>
## 1 Desired RMSE             0.865
## 2 Predicting Randomly      1.50
## 3 Using the Average Rating 1.06
```

Using μ as the prediction for user ratings significantly lowers the RMSE value from predicting randomly, but is still too large compared to the desired RMSE.

2. Including Movie Effects

```
#Calculating bi (the movie effect)
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

#Predicting user rating with mean and bi
y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i

#Add the results
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis' = "Including Movie Effect",
                           RMSE = RMSE(test_set$rating, y_hat_bi)))
results_table
```

```
## # A tibble: 4 x 2
##   `Method of Analysis`      RMSE
##   <chr>                    <dbl>
## 1 Desired RMSE             0.865
## 2 Predicting Randomly      1.50
## 3 Using the Average Rating 1.06
## 4 Including Movie Effect   0.943
```

Including the movie effect reduced the RMSE to 0.943.

3. Including User Effects

```
#Calculating bu (the user effect)
bu <- train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))

#Predicting user rating with mean and bu
y_hat_bu <- mu + test_set %>%
  left_join(bu, by = "userId") %>%
  .$b_u

#Add the results
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis' = "Including User Effect",
                           RMSE = RMSE(test_set$rating, y_hat_bu)))
```

```
results_table
```

```
## # A tibble: 5 x 2
##   `Method of Analysis`      RMSE
##   <chr>                    <dbl>
## 1 Desired RMSE             0.865
## 2 Predicting Randomly      1.50
## 3 Using the Average Rating 1.06
## 4 Including Movie Effect   0.943
## 5 Including User Effect    0.978
```

Including the user effect lowered the RMSE from the average rating prediction of 1.06 to 0.978.

4. Including User and Movie Effects

```
#Predicting user rating with mu, bi and bu
y_hat_bi_bu <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

#Add the results
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis' = "Including Movie and User Effects",
                           RMSE = RMSE(test_set$rating, y_hat_bi_bu)))
results_table
```

```
## # A tibble: 6 x 2
##   `Method of Analysis`              RMSE
##   <chr>                            <dbl>
## 1 Desired RMSE                     0.865
## 2 Predicting Randomly              1.50
## 3 Using the Average Rating         1.06
## 4 Including Movie Effect           0.943
## 5 Including User Effect            0.978
## 6 Including Movie and User Effects 0.884
```

Including both movie and user effects significantly lowered the RMSE to 0.884.

Using Regularization

To regularize the previous model, a range of lambdas is created, and using the equations highlighted in the previous section, new estimates of b_i and b_u are obtained. Then, the value for λ that minimizes RMSE is determined.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
```

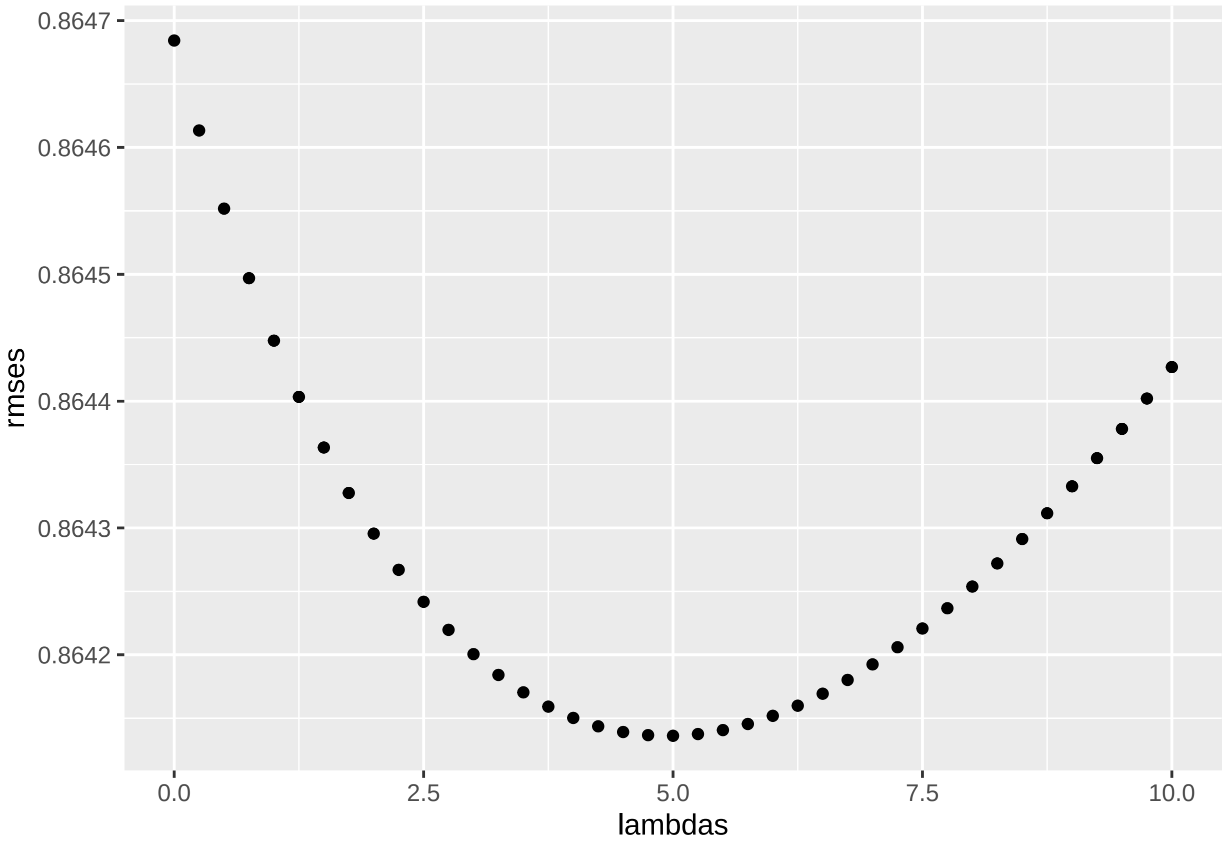
```r
predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

#Pick the lambda that minimizes rmse
lambda <- lambdas[which.min(rmses)]
lambda
```

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

```r
#This can be confirmed via the following plot as well:
qplot(lambdas, rmses)
```



The optimal value of λ is 5; this is the value that is used to shrink the estimates b_i and b_u in case of a small movie-user combination sample size.

λ is then added to the movie and user effect linear model.

```r
#Building the linear model

##Calculating mu again
mu <- mean(train_set$rating)

###Calculating regularized bi (the regularized movie effect)
```

```r
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

###Calculating regularized bu (the regularized user effect)
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

#Predict a regularized estimate of y_hat
reg_y_hat <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#Add the results
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis' = "Regularized Movie and User Effects",
                           RMSE = RMSE(test_set$rating, reg_y_hat)))
results_table
```

```
## # A tibble: 7 x 2
##   `Method of Analysis`                RMSE
##   <chr>                              <dbl>
## 1 Desired RMSE                       0.865
## 2 Predicting Randomly                1.50
## 3 Using the Average Rating           1.06
## 4 Including Movie Effect             0.943
## 5 Including User Effect              0.978
## 6 Including Movie and User Effects   0.884
## 7 Regularized Movie and User Effects 0.864
```

It can be seen that regularizing movie and user effects lowers the RMSE to 0.864, and is lower than the desired RMSE.

Final Validation

Regularizing the movie and user effects linear model achieved two objectives: firstly, it lowered the RMSE enough to be lower than the desired RMSE, and second, it improves upon the movie and user effects linear model by penalizing estimates in the case of small sample sizes. Thus, this will be the model used to validate the model on the validation set.

```r
#calculate the mean of the edx set
mu_edx <- mean(edx$rating)

#Calculating bi (the movie effect) on the edx dataset
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))

#Calculating bu (the user effect) on the edx dataset
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))

#Predict a regularized estimate of y_hat
y_hat_edx <- validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_u + b_i) %>%
  pull(pred)

#Add the results
results_table <- bind_rows(results_table,
                    tibble('Method of Analysis' = "Regularized Movie and User Effects- Final Validation
                           RMSE = RMSE(validation$rating, y_hat_edx)))
results_table
```

```
## # A tibble: 8 x 2
##   `Method of Analysis`                                 RMSE
##   <chr>                                               <dbl>
## 1 Desired RMSE                                        0.865
## 2 Predicting Randomly                                 1.50
## 3 Using the Average Rating                            1.06
## 4 Including Movie Effect                              0.943
## 5 Including User Effect                               0.978
## 6 Including Movie and User Effects                    0.884
## 7 Regularized Movie and User Effects                  0.864
## 8 Regularized Movie and User Effects- Final Validation 0.865
```

On the validation set, the regularized movie and user effects model achieves an RMSE of 0.8648, lower than the desired RMSE of 0.8649.

Conclusion

The objective of this project was to build an algorithm that predicts user ratings for movies from the MovieLens dataset with an RMSE lower than 0.8649. The models were built using a training subset of the 'edx' dataset, a subset of the movielens dataset, and the best model was then validated against the 'validation' dataset. Five models were built in this analysis- a model that predicts user ratings randomly using observed probabilities obtained from a Monte Carlo simulation, three linear models that respectively account for user effects (user to user variation in rating movies), movie effects (movie to movie variation in the dataset) and both user and movie effects. The latter (combined) model was then regularized using the regularization parameter λ to account for small sample sizes for movies that have very few ratings. The regularized model achieved an RMSE lower than the desired RMSE on the test set, and was then validated against the 'validation' set for a final validation. The RMSE achieved was 0.8648, thus fulfilling the project objective.

Limitations

There are several limitations of the final model in this project. Firstly, only RMSE is being used to evaluate model performance, and other measurements such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) can also be used to compare model performance. Furthermore, the movielens dataset contains data on other variables that weren't incorporated in this analysis. While it was determined that 'timestamp' did not affect movie ratings too much, it could be used along with 'genre' to account for time and genre effects in future models to further lower the RMSE.

Future Work

In addition to adding time and genre effects, Singular Value Decomposition (SVD) or k-means clustering can be used for feature reduction to determine the most important predictors. The 'recosystem' package can be used to create a recommendation system using matrix factorization, and the 'recommenderlab' package can allow for more complex analysis (for machines with more processing power) by making powerful and accurate recommendation systems. There are some missing data challenges to address as well, and future analysis can remove users or movies that do not meet a minimum number of ratings.

Appendix

Acknowledgments

I would like to thank Prof. Irizarry and the entire edX staff (especially the discussion forums) for giving me this opportunity to pursue the Data Science Professional Certificate. COVID-19 has been a challenging time for all, and this platform and project have allowed me to use my time productively.

References

1. GroupLens (last updated 2018), MovieLens Latest Datasets
2. Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R

Environment

```r
print("Operating System:")
```

```
## [1] "Operating System:"
```

```r
version
```

```
##                _                           
## platform       x86_64-apple-darwin15.6.0   
## arch           x86_64                      
## os             darwin15.6.0                
## system         x86_64, darwin15.6.0        
## status                                     
## major          3                           
## minor          6.1                         
## year           2019                        
## month          07                          
## day            05                          
## svn rev        76782                       
## language       R                           
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes          
```