Movie Rating Predictions using the MovieLens 10M Dataset

Trevor King

2024-05-14

Contents

```r
Table1<-data.frame(
  Ratings=sum(edx$rating!=0),
  Movies = n_distinct(edx$movieId),
  Users = n_distinct(edx$userId),
  `Genre Combonations Count` = n_distinct(UniqGenGroups)
)
```

Introduction

MovieLens 10M is a stable benchmark dataset of 10 million of the online movie recommender service MovieLens.org Released 1/2009.(Description from Grouplens Website) The edx dataset is approximately 9 million records long and contains the following fields: **userId, movieId, rating, timestamp, title, genres**

```r
# Display the table generated from the code above

knitr::kable(Table1,col.names = c('Ratings','Movies','Users',
        'Count of Unique Genre Combonations'),
        caption = "Number of Unique Users, Movies, and Genres")
```

Table 1: Number of Unique Users, Movies, and Genres

Ratings	Movies	Users	Count of Unique Genre Combonations
9000055	10677	69878	797

The data consist of 3 inter-related text files:

movies.dat (MovieID, Title, and Genres) ratings.dat (UserID, MovieID, Rating, Timestamp) tags.dat (UserID, MovieID, Tag, Timestamp)

Each user is represented by an id. No information about the user is provided. The ratings and tags are in separate files with MovieID as the common primary key relation to the movie.dat The Genres field contain pipe separated combinations selected from the 25 film genres.

We will predict ratings for the test data and determine the lowest Root Mean Square Error (RSME).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$$

Methods/Analysis

Downloading and parsing the Data

- The data is contained in three text files where each line in these files represents a row of data with fields separated by a double colon ("::"). We used a string splitting function to separate the data into the the fields of the dataframe. Using the read_lines function we retrive a line of text, then using the str_split function with the parameter fixed("::") to separate the text into vectors which are inseted into the dataframe.

- **Movies.dat:** Each row in this file represents a movie title, and has the following format: MovieID::Title::Genres, Titles are identical to titles provided by the IMDB (including year of release) Genres are pipe-separated and are selected from the following genres:
  - Action, Adventure, Animation, Children's, Comedy, Crime,
  - Documentary, Drama, Fantasy, Film-Noir, Horror, Musical
  - Mystery, Romance, Sci-Fi,Thriller, War, Western (As described in the Movielense readme.txt)
- **ratings.dat**

  Each row in this file represents a movie that was rating by a user. The ratings representing stars have a range of 1 to 5 with 0.5 increments, with 5 being the highest rating. The time stamp is the number of seconds since January 1, 1970 (UTC)

  The data in the file has the following format: UserID::MovieID::Rating::Timestamp
- **tags.dat** Not used in this analisys

Exploratory Data Analysis

Dimenions of the dataset

The combined number of records for the training set 'edx' and test set 'final_holdout_test'. 10000054 records.

```
# Get the dimensions of the two datasets
dims<-data.frame(edx=dim(edx),final_holdout_test=dim(final_holdout_test),
row.names=c('Number of Rows', 'Number of Columns')
)
dimsT<-t(dims)
```

Table

	Number of Rows	Number of Columns
edx	9000055	6
final_holdout_test	999999	6

Most Rated Movies

```
# List the ten most rated movies
# Group the data by movieId and title
# Summarize the data for each title group
# Count the number of MovieId/titles
# Display in decending order
edx %>% group_by(movieId, title) %>%
        summarize(count = n()) %>% arrange(desc(count))%>%head(10)%>%knitr::kable(col.names = c('Moviel
```

MovieId	Title	Count
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212
457	Fugitive, The (1993)	25998
589	Terminator 2: Judgment Day (1991)	25984
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
150	Apollo 13 (1995)	24284

Highest rated movies with 50+ ratings

```
# Group the data by movie title
# Summarize the data for each title group
# Count the number of ratings for each title
# Calculate the average rating for each title
```

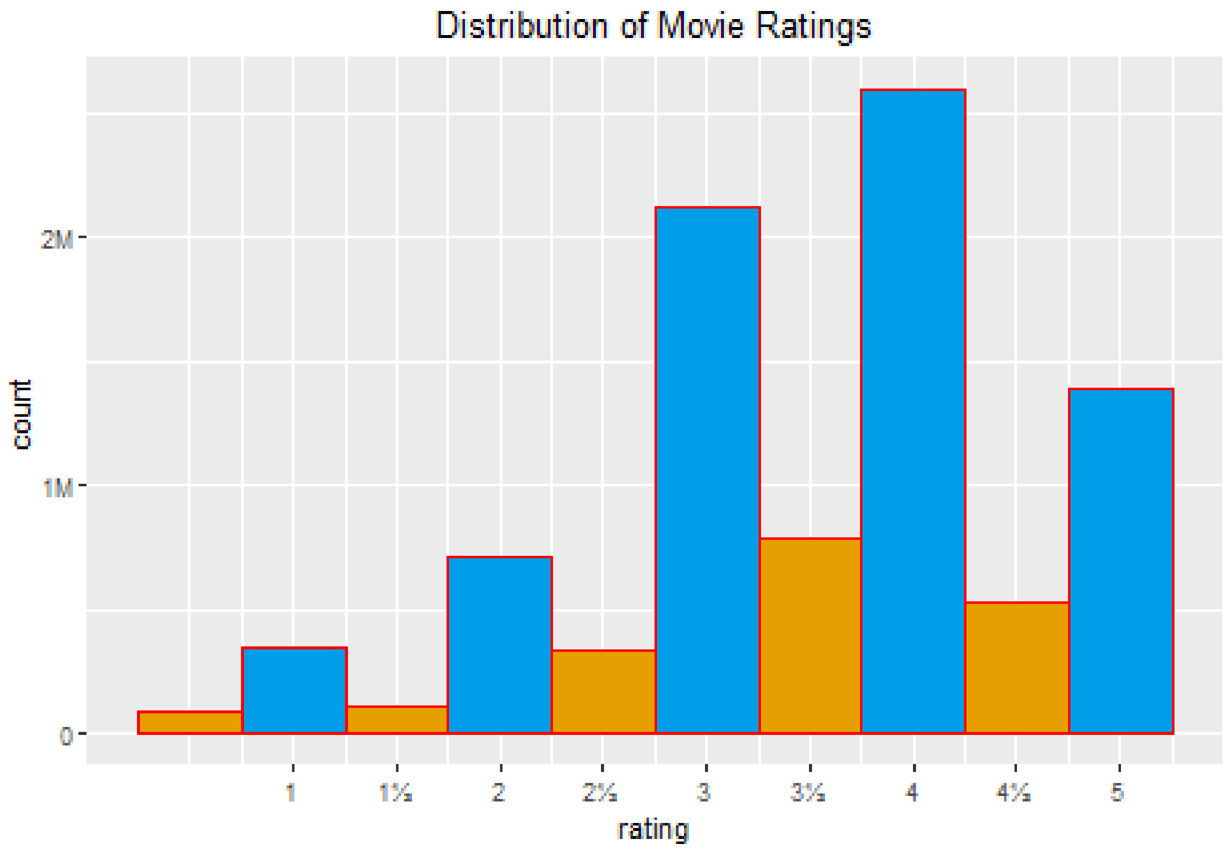
```r
# Filter movies with more than 50 ratings
# Sort the data by average rating in descending order (highest first)
# Select the top 10 movies with the highest average ratings
# Create a Kable table with custom column names
edx %>% group_by(title) %>%
      summarize(numberOfRatings = n(), averageRating = mean(rating)) %>%
      dplyr::filter(numberOfRatings > 50) %>%
      arrange(desc(averageRating)) %>%top_n(10)%>%
  knitr::kable(col.names = c('Title','Number of Ratings','Average Rating'))
```

```
## Selecting by averageRating
```

Title	Number of Ratings	Average Rating
Shawshank Redemption, The (1994)	28015	4.455131
Godfather, The (1972)	17747	4.415366
Usual Suspects, The (1995)	21648	4.365854
Schindler's List (1993)	23193	4.363493
Casablanca (1942)	11232	4.320424
Rear Window (1954)	7935	4.318651
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	2922	4.315880
Third Man, The (1949)	2967	4.311426
Double Indemnity (1944)	2154	4.310817
Paths of Glory (1957)	1571	4.308721

Count of Movies by Rating

```r
# Create a histogram of the count of movies by rating
# Alternating Colours by Half star/Full Star
# Save as Plot1
# unicode character for 1/2 \u00bd

lbls<-c('1','1\u00bd','2','2\u00bd','3','3\u00bd','4','4\u00bd','5')

Plot1 <-edx%>%ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5,fill = rep(c("#e69f00","#009ee9"),5),
  color = "red") + ggtitle("Distribution of Movie Ratings") +
  theme(plot.title = element_text(hjust = 0.5))


# Change Y access to show count in Millions (M)
Plot1  + scale_y_continuous(labels = scales::label_number_si())+
  scale_x_continuous(breaks=seq(1,5,0.5),labels=lbls)
```


Distribution of Movie Ratings
count
2M
1M
0
1
1½
2
2½
3
3½
4
4½
5
rating

Check for missing values in the data set

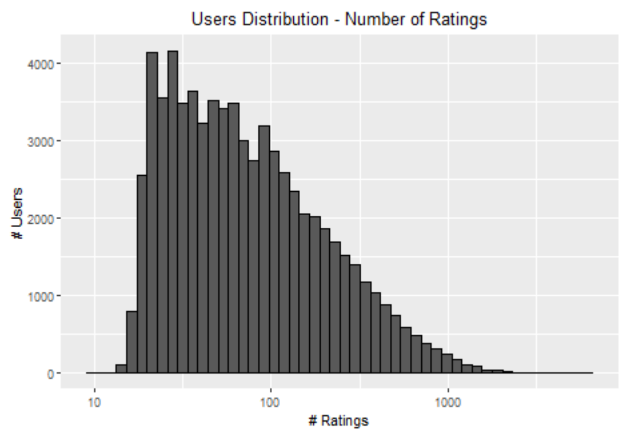
	# of Missing Values
userId	None
movieId	None
rating	None
timestamp	None
title	None
genres	None

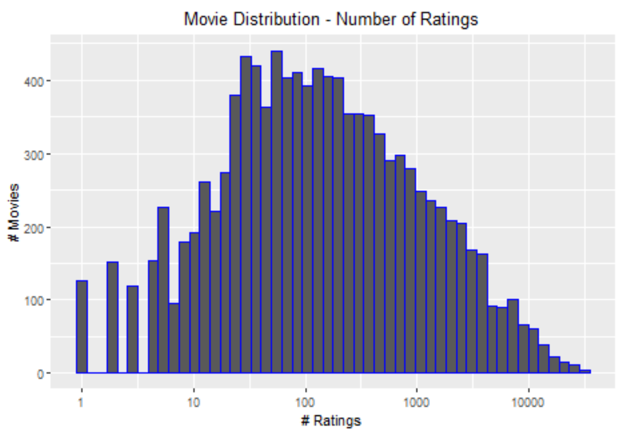
Rating distribution (number of ratings per user)

The overall ratings matrix is sparse since 65% of user rates less than 100 movie titles, and only 4% of the users rate more than 500 movies.

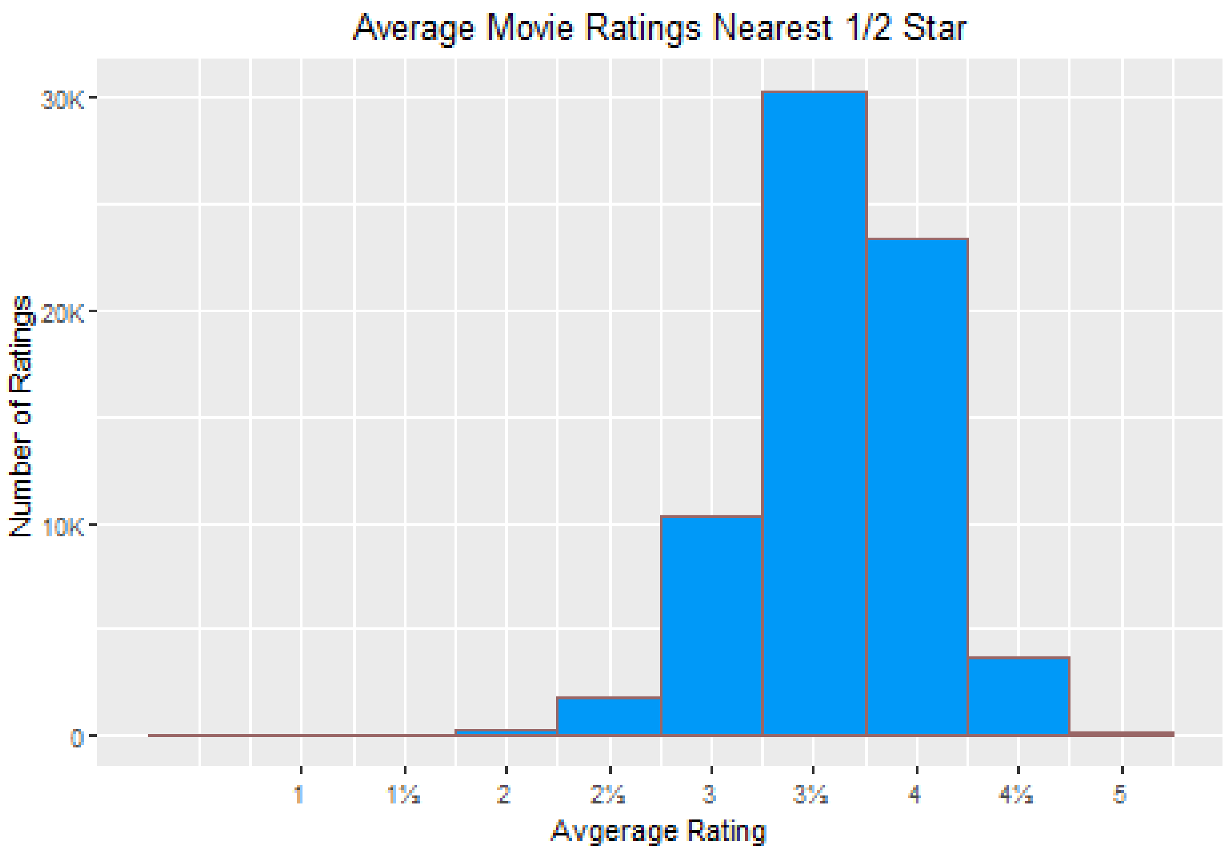
```r
# Create a histogram of the count of movies by rating
# Save as Plot2

Plot2<-edx %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(color = "black", bins = 50) +
scale_x_log10() +
xlab("# Ratings") +
ylab("# Users") +
ggtitle("Users Distribution - Number of Ratings") +
theme(plot.title = element_text(hjust = 0.5))
```

```r
# Create a histogram of the count of movies by rating
# Save as Plot3

Plot3<-edx %>%
count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(color = "blue", bins = 50) +
scale_x_log10() +
xlab("# Ratings") +
ylab("# Movies") +
ggtitle("Movie Distribution - Number of Ratings") +
theme(plot.title = element_text(hjust = 0.5))

Plot2

Plot3
```





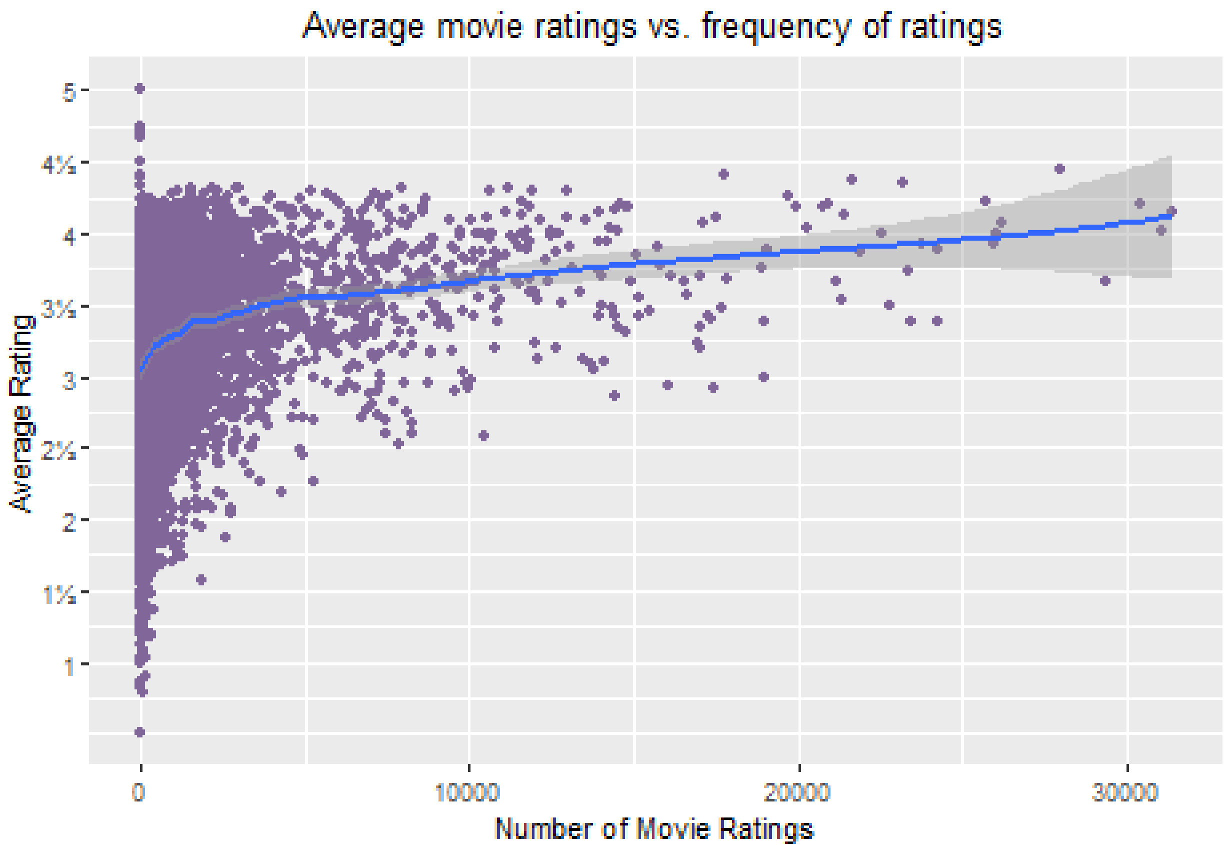
```
# Number of ratings by UserID with Average Rating
# rounded to the nearest 1/2 star
u_ratings <- edx %>% group_by(userId) %>%
  summarize(n= n(), avg.rating = round(mean(rating)*2,digits = 0)/2)

# Plot the userID against the number of ratings


u_ratings %>% ggplot(aes(avg.rating)) +
  geom_histogram(colour = "#996666", fill = "#0099F8",bins = 10) +
  labs(title = "Average Movie Ratings Nearest 1/2 Star",
       x='Avgerage Rating', y='Number of Ratings')+
  theme(plot.title = element_text(hjust = 0.5))+
  scale_x_continuous(breaks=seq(1,5,0.5),labels=lbls)+
  scale_y_continuous(labels = scales::label_number_si())
```



Rating distribution (number of ratings per movie)

Some movies are streaming and rated by tens of thousands of users, while others receive less than 100 ratings

```
# Another plot (number of ratings per movie)
mv_ratings <- edx %>% group_by(movieId) %>%
summarize(nMovie_Rtings= n(), avg.rating = mean(rating))%>%
arrange(desc(avg.rating))
mv_ratings %>% ggplot(aes(x = nMovie_Rtings, y = avg.rating)) +
geom_point(colour = "#806699") + geom_smooth(method = "loess",
span=0.1,formula = 'y ~ x') +
labs(title = "Average movie ratings vs. frequency of ratings",
  y='Average Rating',x='Number of Movie Ratings') +
  theme(plot.title = element_text(hjust = 0.5))+
  scale_y_continuous(breaks=seq(1,5,0.5),labels=lbls)
```



Top 10 Most Popular Genres

```
# Split genres text into separate genres for each movie
# Combine all genres from all movies into a single vector
# Count the occurrences of each genre
# Sort genres by count in descending order (most frequent first)
# Select the top 10 most frequent genres
# Convert the data frame to a matrix (easier for kable)
# Transpose the matrix (genres as rows, counts as columns for kable)
# Create a Kable table to display the results
strsplit(edx$genres,"\\|")%>%
  unlist(use.names = FALSE)%>%
  table()%>%sort(decreasing=TRUE)%>%head(10)%>%
  as.matrix()%>%t()%>%knitr::kable()
```

Drama	Comedy	Action	Thriller	Adventure	Romance	Sci-Fi	Crime	Fantasy	Children
3910127	3540930	2560545	2325899	1908892	1712100	1341183	1327715	925637	737994

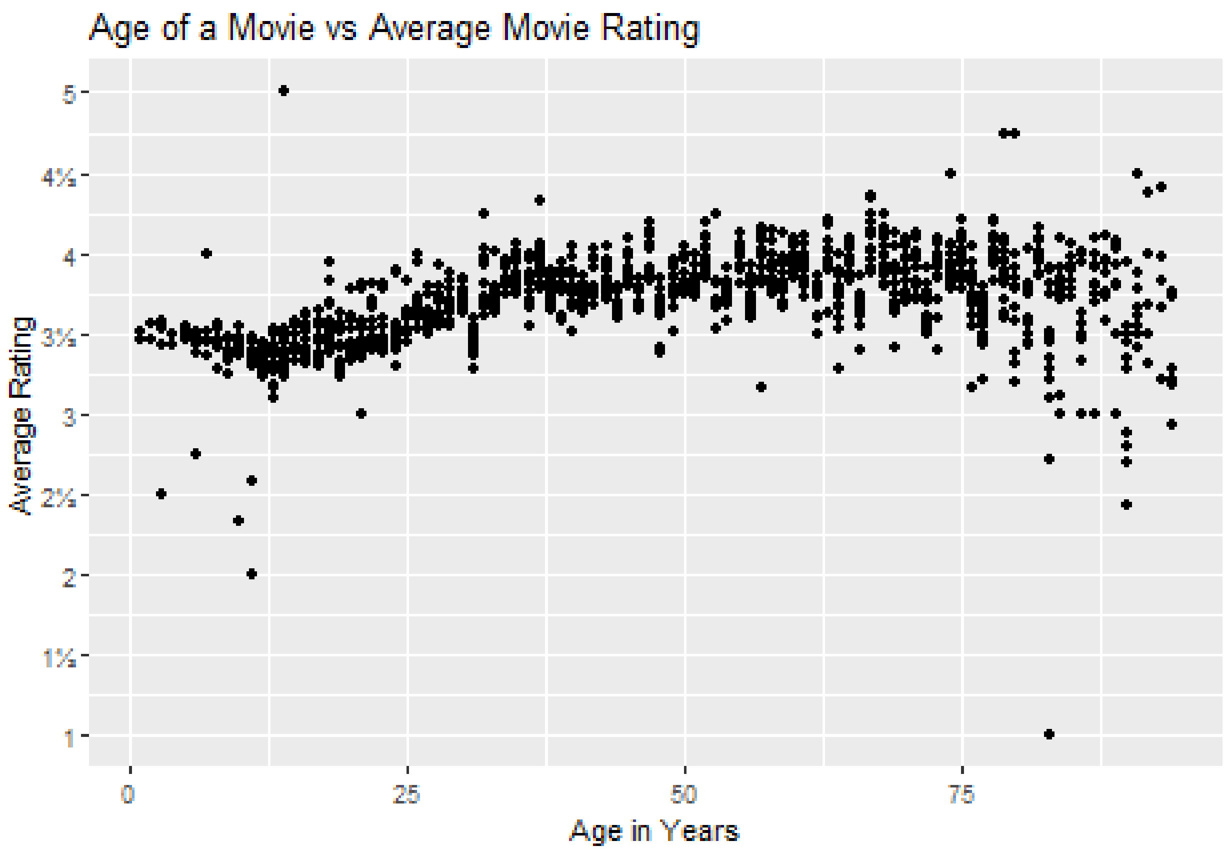
### Movie Age

```r
library(stringr)
library(lubridate)
# Code to calculate Age of the movie from when the database was created
DBYr <- edx$timestamp%>%max()%>%as_datetime()%>%year()

# Pattern to match release year (in title)
# Matches exactly four consecutive digits
# between brackets at the end of the string.
Yr_patt<-"(\\(\\d{4}\\))$"

# Extract the release date from the movie title field
# encapsulated in brackets at the end of the field,
# Calculate the age of the movie and extract the year
# from the timestamp (Year Rated)
edx_with_dates<-edx%>%
mutate(releaseYR=as.numeric(substr(str_extract(title, Yr_patt),2,5))
,title = sub(Yr_patt,'',title),
year_rated = year(as_datetime(timestamp)),
MovieAge=DBYr-releaseYR # Calculate the Age from DBYr
)

# Generate a table of the top ten
edx_with_dates %>%
dplyr::select(title,genres,releaseYR,year_rated,MovieAge)%>%
head() %>%
knitr::kable(caption = 'edx dataset with movie age',
col.names = c('Title','Genres','Release Year',
    'Year Rated','Movie Age'))
```

Table 7: edx dataset with movie age

Title	Genres	Release Year	Year Rated	Movie Age
Boomerang	Comedy|Romance	1992	1996	17
Net, The	Action|Crime|Thriller	1995	1996	14
Outbreak	Action|Drama|Sci-Fi|Thriller	1995	1996	14
Stargate	Action|Adventure|Sci-Fi	1994	1996	15
Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	1994	1996	15
Flintstones, The	Children|Comedy|Fantasy	1994	1996	15
# Average movie rating vs	. Frequency of rating			

```r
# Get the average by movie age
AvgByAge<-edx_with_dates %>%
  group_by(year_rated,MovieAge)%>%
  summarize(AvgRatingByAge=mean(rating))

# And Plot it against the ratings
AvgByAge%>%ggplot(aes(MovieAge,AvgRatingByAge))+geom_point()+
  ggtitle("Age of a Movie vs Average Movie Rating")+
  xlab("Age in Years")+ylab("Average Rating")+
  scale_y_continuous(breaks=seq(1,5,0.5),labels=lbls)
```


Age of a Movie vs Average Movie Rating
Average Rating
5
4½
4
3½
3
2½
2
1½
1
0
25
50
75
Age in Years

Simple Model

The most basic model assumes an equal rating for all movies and users, irrespective of the user, while attributing any variations solely to random factors.

Where μ represents the true rating and ε represents the independent errors

$$Y_{u,i} = \mu + \varepsilon_{\mu,m}$$

We have split the edx data into 20% training and 80% test.

Using the training data, we will determine μ and will use it to calculate RMSE for the test data using the simple model.

```r
# Determine the average rating in the training data
muHat <- mean(train_set$rating)

# Lets build a tibble and add the RMSE for each method
# Initializing the tibble and adding the first result

RMSE_results<- tibble(Method= "Average without Bias",
        RMSE = RMSE(test_set$rating, muHat))
```

To improve the model we will assume that widely promoted movies often receive higher ratings than less well know movies.

For each movie, we determine how much its average rating deviates from the total mean. If a movie tends to receive higher ratings than the overall average, its bias will be positive. Conversely, if it receives lower ratings, the bias will be negative.

Adding in the Movie Bias bi

$$Y_{u,i} = \mu + bi + \varepsilon_{\mu,m}$$

```r
# Calculating b_i
movie_avgs <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - muHat))

predicted_ratings <- muHat + test_set %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i
m_1_rmse <- RMSE(predicted_ratings, test_set$rating)
# adding RMSE for the Movie Effect to the tibble
RMSE_results <- RMSE_results%>%add_row(Method="Movie Effect Model",
                RMSE = m_1_rmse )
# Use Movie_avgs in the next chunk
```

Adding in the User Bias bu

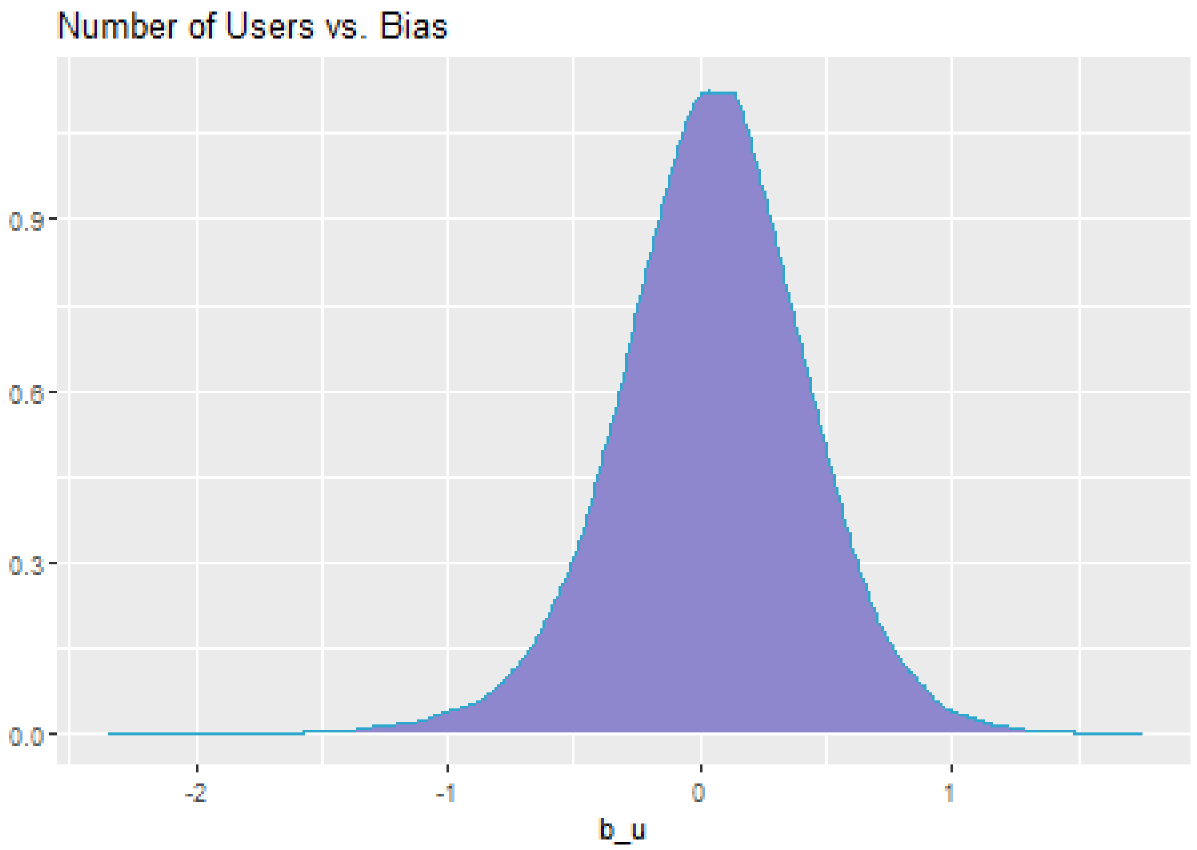
$$Y_{u,i} = \mu + bi + bu + \varepsilon_{\mu,m}$$

Many users that watch movies to be entertained will provide higher ratings than those who are critical of the Writing, acting, cinematography, directing.

For each user, we determine how their average rating deviates from the total mean. If a user tends to rate lower than the overall average, a negative bias will be applied. We will only calculate the average for users who have rated at least 50 movies.

```
# Graphing Averages for users who have rated at least 50 movies

LimitN <-50
user_avgs <- edx %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
      dplyr::filter(n()>LimitN)%>%
     summarize(b_u = mean(rating - muHat - b_i))

Plot7<- user_avgs%>% qplot(b_u, geom ="density", data = .,
    colour = I("#2EA7CE"), fill = I("#8E87CE"),
    main = "Number of Users vs. Bias")
Plot7
```

```
## Warning: Removed 842 rows containing non-finite values (stat_density).
```



```
# Calculating User Averages b_u using muHat and b_i from the previous chunk

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

# Predicting Ratings using the test data

predicted <- test_set %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     mutate(pred = muHat + b_i + b_u) %>%
     .$pred
m_2_rmse <- RMSE(predicted, test_set$rating)

# adding RMSE for the Movie and User Effect to the tibble
RMSE_results <- RMSE_results%>%add_row(Method="Movie and User Effects Model",
  RMSE = m_2_rmse )
```

Add in the Genres Effect

```
# Lets add the Genres effect
muHat <- mean(train_set$rating)

# Calculating Genres Averages b_g using muHat and b_i, b_u from the previous chunk

 genre_avgs <- train_set %>%
   left_join(movie_avgs, by = "movieId") %>%
   left_join(user_avgs, by = "userId") %>%
   group_by(genres) %>%
   summarize(b_g = mean(rating - muHat - b_i - b_u))

# Predicting Ratings using the test data
  predicted <- test_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(genre_avgs, by = "genres") %>%
    mutate(pred = muHat + b_i + b_u + b_g) %>%
    .$pred
  m_3_rmse <- RMSE(predicted, test_set$rating)
  RMSE_results <- RMSE_results%>%
    add_row(Method="Movie, User, Genres Effects Model", RMSE = m_3_rmse )
```

Add in the Release Year Effect

```
# Lets add the release year
# Calculating Release Year Averages b_y using muHat and
# b_i, b_u, b_g from the previous chunk
 relyear_avgs <- train_set %>%
     left_join(movie_avgs, by = "movieId") %>%
     left_join(user_avgs, by = "userId") %>%
     left_join(genre_avgs, by = "genres") %>%
     group_by(releaseYR) %>%
      summarize(b_y = mean(rating - muHat - b_i - b_u - b_g))
```

```r
# Predicting Ratings using the test data
   predicted <- test_set %>%left_join(movie_avgs, by = "movieId") %>%
     left_join(user_avgs, by = "userId") %>%
     left_join(genre_avgs, by = "genres") %>%
     left_join(relyear_avgs, by = "releaseYR") %>%
     mutate(pred = muHat + b_i + b_u + b_g + b_y) %>%
      .$pred
   m_4_rmse <- RMSE(predicted, test_set$rating)

 RMSE_results <- RMSE_results%>%
   add_row(Method="Movie, User, Genres, Release Year Effects Model",
   RMSE = m_4_rmse )
```

Create the predictor from the training set ratings:

The model used for developing the prediction algorithm follows that from the course: the mean rating μ is modified by one or more "bias" terms b with a residual error ε expected.

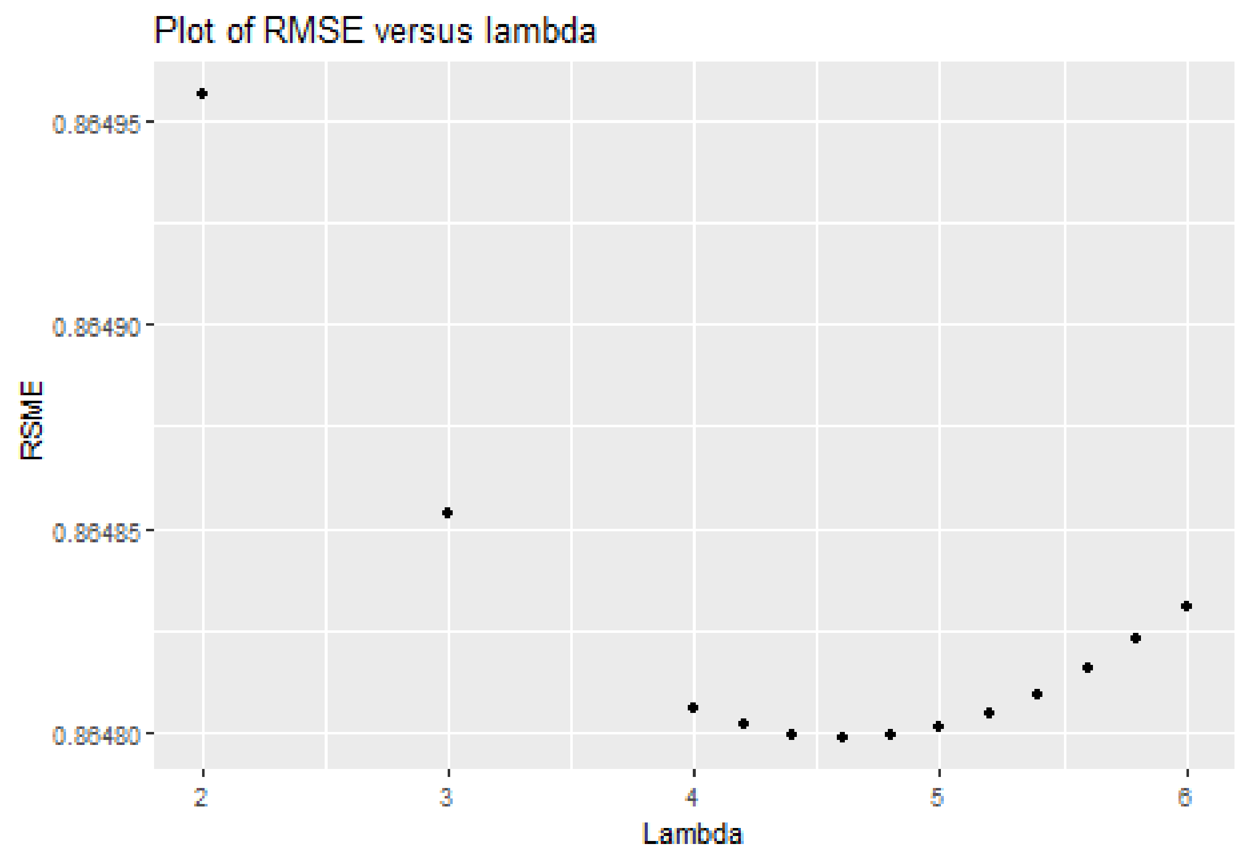
$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + \varepsilon_{i,u,g}$$

Regularize by Movie, User, Genres, Release Year

Determine the best Lambda

```r
# lambda is used as a tuning parameter

# Function used to calculate RMSE for a given lambda

get_rmses<-function(l) {
muHat <- mean(train_set$rating)
# message(paste("Trying Lambda", as.character(l)))
# setWinProgressBar(pb, l, sprintf("Lamda (%s)", l), l)
bi <- train_set %>%
 group_by(movieId) %>%
 summarize(b_i = sum(rating - muHat)/(n() + l))

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

bg <- train_set %>% left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - muHat)/(n() + l))

by <- train_set %>% left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(bg, by = "genres") %>%
  group_by(releaseYR) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_g - muHat)/(n() + l))

# Predicting Ratings using the test data
predicted <- test_set %>%left_join(bi, by = "movieId") %>%
     left_join(bu, by = "userId") %>%
     left_join(bg, by = "genres") %>%
     left_join(by, by = "releaseYR") %>%
      mutate(p = muHat + b_i + b_u + b_g + b_y) %>% .$p
return(RMSE(predicted, test_set$rating))

}
```

```r
# Get approximately the best lambda
# To save on processing steps we start with 5 values

lambdas <- seq(2, 6, 1)

m_avg <- mean(train_set$rating)

# Get predictions from the lambdas using the function get_rmses
p_rmses <- sapply(lambdas, get_rmses)

# Get the lowest value
good<-lambdas[which.min(p_rmses)]

# Save these for the graph
P_temp<-p_rmses
L_temp<-lambdas

# Create a new set of lambda values with smaller increments
# centered around the best value we've identified in the
# previous step

lambdas<-seq(good - 0.8, good +0.8, 0.2)

# Get predictions from the new lambdas using the function get_rmses
p_rmses <- sapply(lambdas, get_rmses)
# Get the lowest value
better<-lambdas[which.min(p_rmses)]

# for the graph
P_temp<-c(P_temp,p_rmses)
L_temp<-c(L_temp,lambdas)

# Plot with title and axis labels
qplot(L_temp,P_temp,xlab='Lambda',ylab='RSME',
      main = 'Plot of RMSE versus lambda')
```



```
# Create a new set of lambda values with even smaller increments
# centered around the best value we've identified in the
# previous step

lambdas<-seq(better - 0.1, better +0.1, 0.05)
p_rmses <- sapply(lambdas, get_rmses)

best<-lambdas[which.min(p_rmses)]

best # Print the best RMSE
```

```
## [1] 4.6
```

```
# adding RMSE for the Regularize by Movie, User, Genres, Release Year
# to the tibble
RMSE_results <- RMSE_results%>%
add_row(
  Method="Regularize by Movie, User, Genres, Release Year Effects Model",
  RMSE = min(p_rmses) )

# Printing all the RMSE results
RMSE_results %>% knitr::kable()
```

Method	RMSE
Average without Bias	1.0599043
Movie Effect Model	0.9437429
Movie and User Effects Model	0.8659334
Movie, User, Genres Effects Model	0.8655951
Movie, User, Genres, Release Year Effects Model	0.8654197
Regularize by Movie, User, Genres, Release Year Effects Model	0.8647985

Using the Movie/User regularized with the best lambda

Results

We calculate RMSE with the regularized user/movie model using the lambda that achieved the best RMSE.

```r
# testing the optimized lambda on the Final Hold out data
# Fist get the average rating from the final_holdout_test data
m_avg <- mean(final_holdout_test$rating)

# get dataframe with the movieId and Release year
Movie_Release<-edx_with_dates%>%group_by(movieId)%>%distinct(releaseYR)

# Need to add the release year to the holdout data with a left join
# since it wasn't created with the edx supplied code

final_holdout_test1<-final_holdout_test %>%
left_join(Movie_Release, by = "movieId")

# Function to get the RMSE on the final_holdout_test data
get_holdout<-function(l){

    message(paste("Using Lambda", as.character(l),"on the holdout data"))

# b_i
bi <- final_holdout_test1%>%
 group_by(movieId) %>%
 summarize(b_i = sum(rating - muHat)/(n() + l))
# b_u
bu <- final_holdout_test1 %>%
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - muHat)/(n() + l))
# b_g
bg <- final_holdout_test1%>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - muHat)/(n() + l))
# b_y
by <- final_holdout_test1%>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(bg, by = "genres") %>%
  group_by(releaseYR) %>%
  summarize(b_y = sum(rating - b_i - b_u - b_g - muHat)/(n() + l))

  # using Holdout test data for the predictions
    predicted <- final_holdout_test1%>%
     left_join(bi, by = "movieId") %>%
     left_join(bu, by = "userId") %>%
     left_join(bg, by = "genres") %>%
     left_join(by, by = "releaseYR") %>%
     mutate(p = muHat + b_i + b_u + b_g +b_y) %>% .$p
return(RMSE(predicted, final_holdout_test1$rating))


return(RMSE(predicted, final_holdout_test$rating))
}

RMSE_Holdout<-get_holdout(best)
```

```
## Using Lambda 4.6 on the holdout data
```

```r
RMSE_results <- RMSE_results%>%add_row(Method=
"Regularize by Movie, User, Genres, Release Year Effects Model using holdout data",
  RMSE = RMSE_Holdout )
```

Table of results

```r
RMSE_results %>% knitr::kable()
```

Method	RMSE
Average without Bias	1.0599043
Movie Effect Model	0.9437429
Movie and User Effects Model	0.8659334
Movie, User, Genres Effects Model	0.8655951
Movie, User, Genres, Release Year Effects Model	0.8654197
Regularize by Movie, User, Genres, Release Year Effects Model	0.8647985
Regularize by Movie, User, Genres, Release Year Effects Model using holdout data	0.8386442

RMSE for the final holdout data is 0.8386442

Conclusion

This project aimed to build a movie recommendation system using the MovieLens 10M dataset. Using the code from “Create edx and final_holdout_test sets” I created a combined edx dataframe by merging user ratings with the movie data.

To evaluate the performance of the recommendation system, I divided the MovieLens 10M dataset (provided by edX) into training and test sets.

After evaluating six recommendation methods, I found that Regularize by Movie, User, Genres, and movie release year Effects model achieved the lowest Root Mean Squared Error (RMSE). This suggests that leveraging both user preferences and overall movie popularity leads to more accurate predictions.

I achieved an RMSE of 0.8386442 on the final hold-out data by tuning the lambda parameter from the regularized user/movie model. The most optimal value for lambda was 4.6.

References

Irizarry, Rafael A., “Introduction to Data Science: Data Analysis and Prediction Algorithms in R” https://rafalab.github.io/dsbook/

https://grouplens.org/datasets/movielens/10m/

https://bookdown.org/yihui/bookdown/get-started.html