Movielens Recommendation System Project

HarvardX PH125.9x Data Science: Capstone

Sara Darland

September 9, 2024

Contents

Introduction and Objective

According to Irizaary (2019), a recommendation system is a type of machine learning model that uses data such as user input and behavior to predict other items the user may be interested in consuming; these predicted items are then recommended to the user. Netflix uses a recommendation system based on user ratings to predict other movies a user might enjoy (Irizarry, 2019). Inspired by the Netflix Prize in October of 2006 where Netflix offered one million dollars to anyone that could improve their recommendation system by 10%, this project will create a movie recommendation system model that predicts the rating a specific user would give to a specific movie. The Root Mean Squared Error (RMSE) will be used to evaluate the accuracy of the model, with the goal being an RMSE of less than 0.8649 when the model is run on previously unseen data. This project is required coursework for the HarvardX PH125.9x Data Science: Capstone course.

Dataset

Netflix data is not publicly available so this project uses the Movielens 10M dataset. This dataset is generated by MovieLens, an online movie recommender service run by GroupLens, a research lab in the Department of Computer Science and Engineering at The University of Minnesota.

Code generating the movielens dataset, the train set "edx", and validation set "final_hold_out" was provided by the course. The provided code has been modified to ensure the necessary packages are installed and options for significant digits and scientific notation set.

```
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# Note: This code has been modified
# Note: this process could take a couple of minutes

# install and load needed libraries
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(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("GridExtra", repos = "http://cran.us.r-project.org")
if(!require(raster)) install.packages("raster", repos = "http://cran.us.r-project.org")
if(!require(float)) install.packages("float", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(knitr)
library(gridExtra)
library(raster)
library(float)

#set options
options(timeout = 120, digits = 4, scipen = 999)

# download data files
```

```r
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

# turn ratings_file into a dataframe
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# turn movies_file into a dataframe
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# join ratings and movies dataframes into one called "movielens"
movielens <- left_join(ratings, movies, by = "movieId")

# Create training set (edx) and validation set (final_holdout_test)
# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# 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,]

# Ensure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)
```

```r
# clean up environment by removing temp files
rm(dl, ratings, movies, test_index, temp, removed)
```

This code creates two datasets. The edx set will be used for data exploration and model development. The final_holdout_test set will only be used to calculate the RMSE of the final model; this will determine how accurately the model developed works on previously unseen data.

Data Exploration and Analysis

The first 5 rows of the edx set are observed; it is in tidy format:

Table 1: First 5 rows of edx dataset

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

Each row corresponds to a rating given by a user to a movie. The edx set is a data frame and consists of 9,000,055 rows with the following 6 columns, or features:

- userId – unique user identifier. It is an integer.
- movieId – unique movie identifier. It is an integer.
- rating – the rating given by the user to a movie. It is a number.
- timestamp – the time and date that a rating was given in Unix time. It is an integer.
- title – the title of the movie. It is a character.
- genres – the genres associated with a movie. Individual genres are separated by "|". It is a character.

Table 2: Number of ratings, users, and movies

n_ratings	n_users	n_movies
9000055	69878	10677

There are 69878 unique users and 10677 unique movies. If every user rated every movie there would be more than 746 million rows in the dataset. Instead there are only approximately 9 million rows; this means every user did not rate every movie.

Rating

Users rated movies on a scale between 0.5 and 5 stars in increments of 0.5 stars. The higher the rating the more the user liked the movie. The mean rating for the edx set is 3.5125 and represented by a dashed red line in plots.

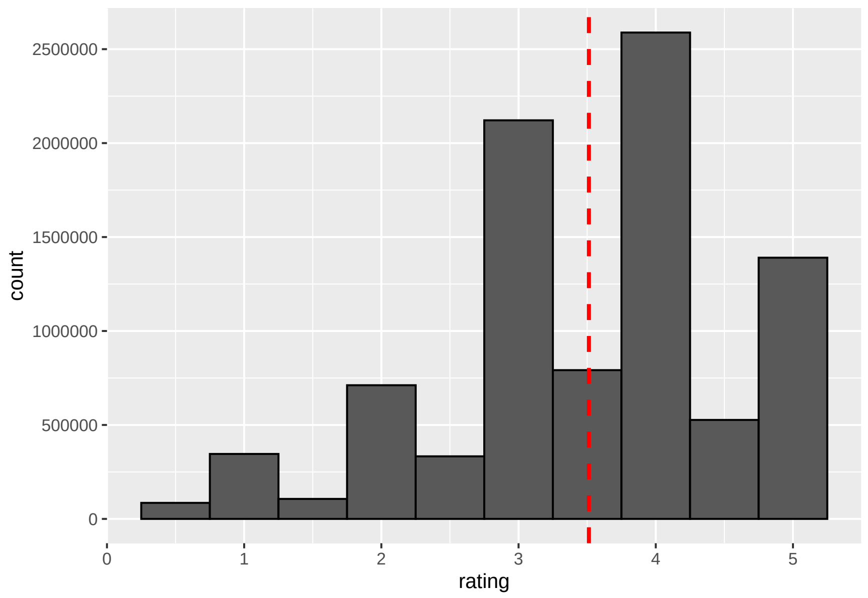

Figure 1: Rating distribution

The most commonly occurring rating is 4.0, followed by 3.0 and then 5.0. Whole number ratings are more common than half ratings.

Movie

The distribution of average rating by movie is created by grouping unique movies together using the movieId feature. Intuitively, we know some movies are rated higher than others because some movies are good and other are bad.

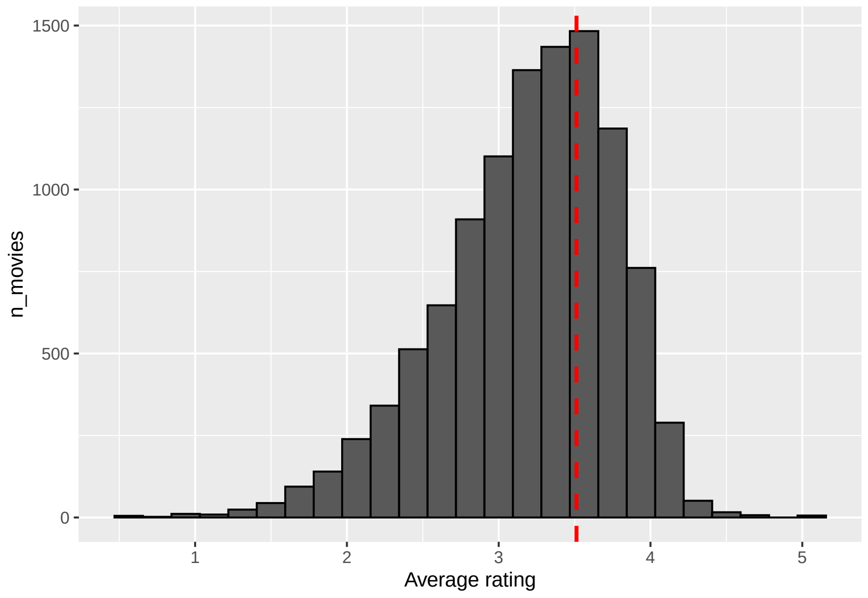

Figure 2: Movie Distribution by average rating

The difference between a unique movie's average rating and the overall average rating, otherwise known as the movie effect or movie bias, will be taken into account when developing the model. The average number of ratings per movie is 842.9 and the distribution of number of ratings per movie is below.

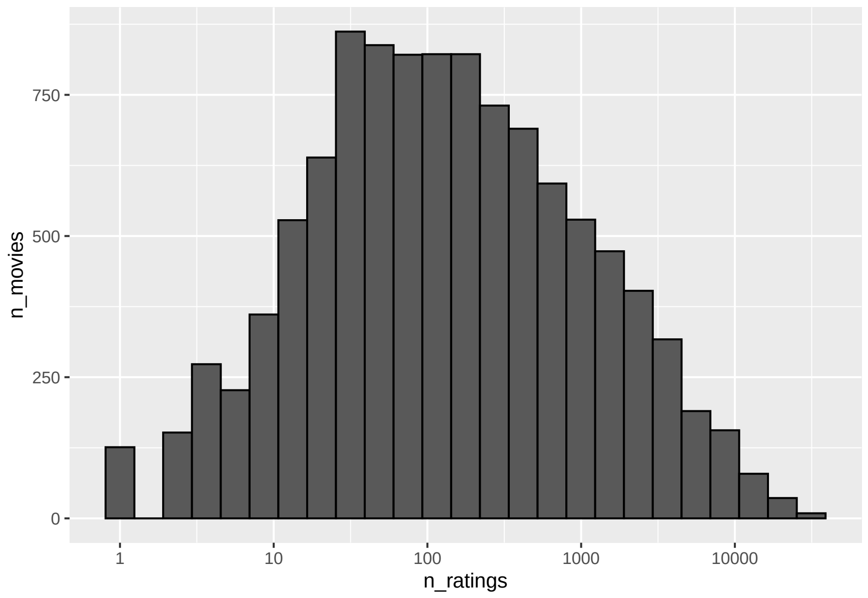

Figure 3: Number of ratings by movie

Some movies are rated more frequently than others, with the number of ratings per movie varying between 1 and 31362. As seen in Table 3, the movies with the highest number of ratings are all familiar blockbusters with ratings higher than the overall average rating of 3.5125.

Table 3: Average rating for movies with highest number of ratings

title	n_ratings	avg_rating
Pulp Fiction (1994)	31362	4.155
Forrest Gump (1994)	31079	4.013
Silence of the Lambs, The (1991)	30382	4.204
Jurassic Park (1993)	29360	3.663
Shawshank Redemption, The (1994)	28015	4.455
Braveheart (1995)	26212	4.082

Conversely some movies were only rated once. Below is a sample of one rating movies with their respective ratings.

Table 4: Average rating for movies with one rating

title	n_ratings	rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	1	2.0
100 Feet (2008)	1	2.0
4 (2005)	1	2.5
Accused (Anklaget) (2005)	1	0.5
Ace of Hearts (2008)	1	2.0

title	n_ratings	rating
Ace of Hearts, The (1921)	1	3.5
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di…) (1971)	1	1.5
Africa addio (1966)	1	3.0
Aleksandra (2007)	1	3.0
Bad Blood (Mauvais sang) (1986)	1	4.5

In total there were 126 movies that were only rated once. As seen in Table 4, the ratings for these one rating movies are varied. We can infer that the rating for a one rating movie is highly dependent on the preferences of the user who rated it, so further exploration of the user effect is needed to develop a robust model.

User

The distribution of average rating by user is created by grouping unique users together using the userId feature. Intuitively, we know users have different preferences; some users will give a movie a high rating while other users will give the same movie a low rating.

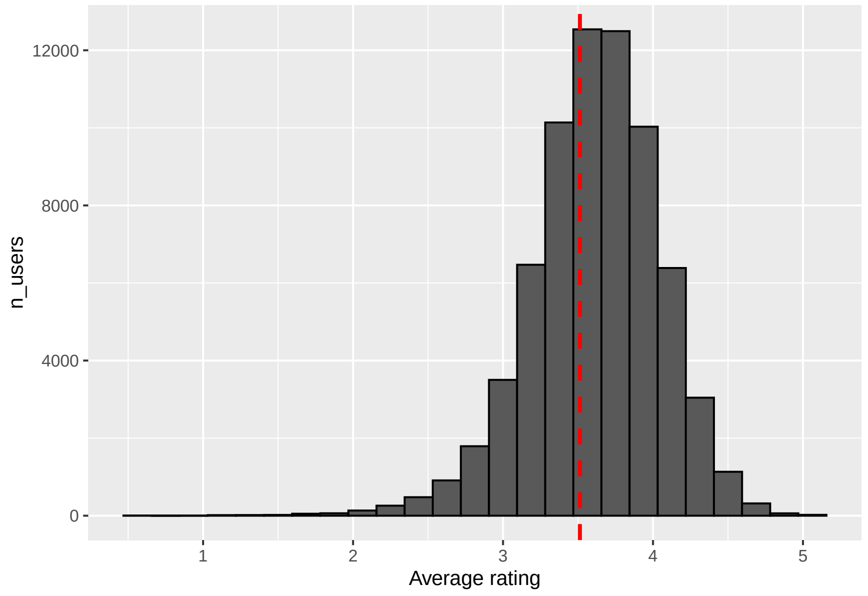

Figure 4: User distribution by average rating

The difference between a unique user's average rating and the overall average rating, otherwise known as the user effect or user bias, will be taken into account when developing the model. The average number of ratings given per user is 128.8 and the distribution of number of ratings per user is below.

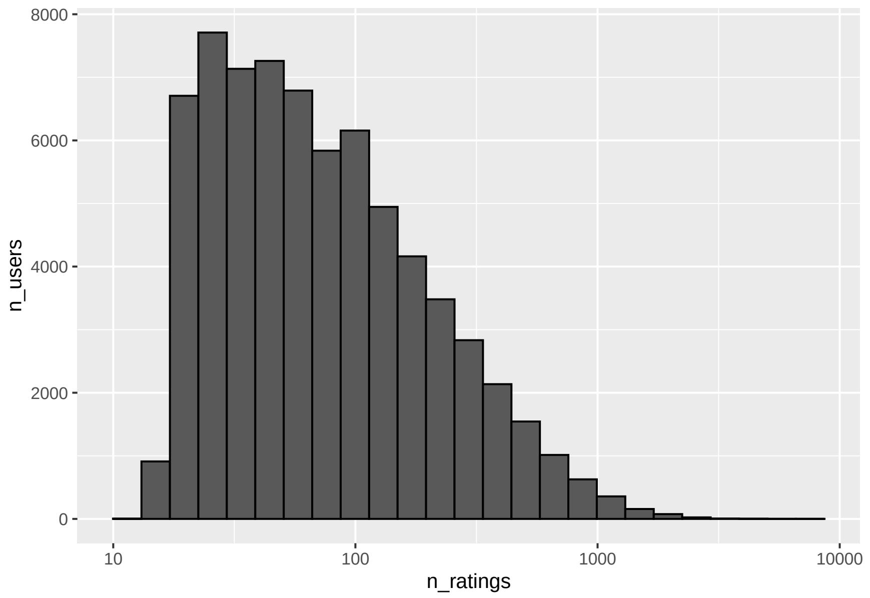

Figure 5: Number of ratings by user

Some users rate more movies than others, with the number of ratings per user varying between 10 and 6616. Since every user rated at least 10 movies, user bias should contribute to developing accurate predictions for movies with few ratings.

Genre

The genre feature includes all genres associated with a movie separated by the "|" symbol, as seen in Table 5:

Table 5: Sample of movies and genres

	title	genres
1	Boomerang (1992)	Comedy|Romance
2	Net, The (1995)	Action|Crime|Thriller
4	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	Stargate (1994)	Action|Adventure|Sci-Fi
6	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

The genres associated with a movie are referred to a genre combinations.There are a total of 797 unique genre combinations comprised of 20 individual genres. The number of individual genres per movie can be calculated using the following code:

```r
#individual genres per movie
genres_per_movie <- edx %>% group_by(movieId) %>%
  data.frame(count = str_count(edx$genres, "\\|") +
               1 - str_count(edx$genres, "no genres listed")) %>%
  group_by(movieId, title) %>% summarize(count = mean(count))
```

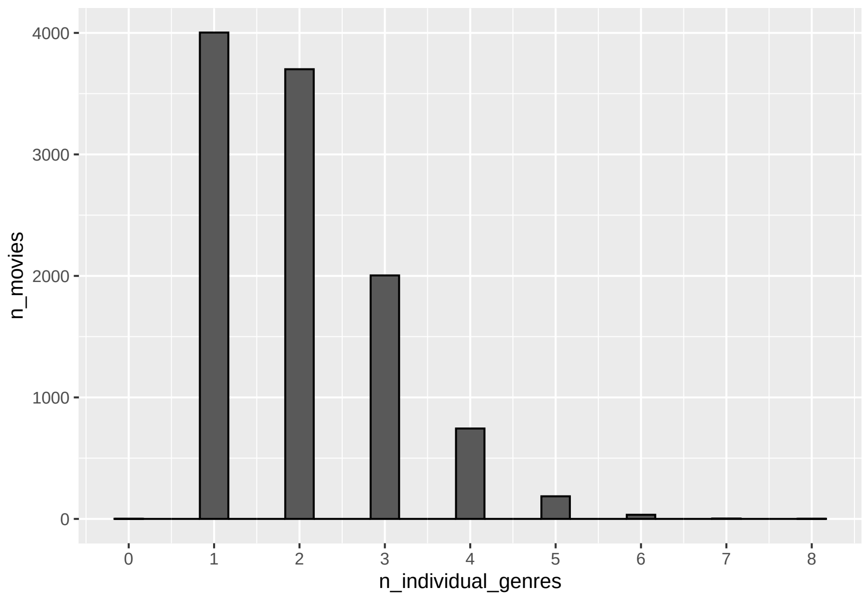

Figure 6: Number of individual genres per movie

The prevalence of a movie having more than one genre is 0.6251. The number of ratings per number of individual genres is also considered.

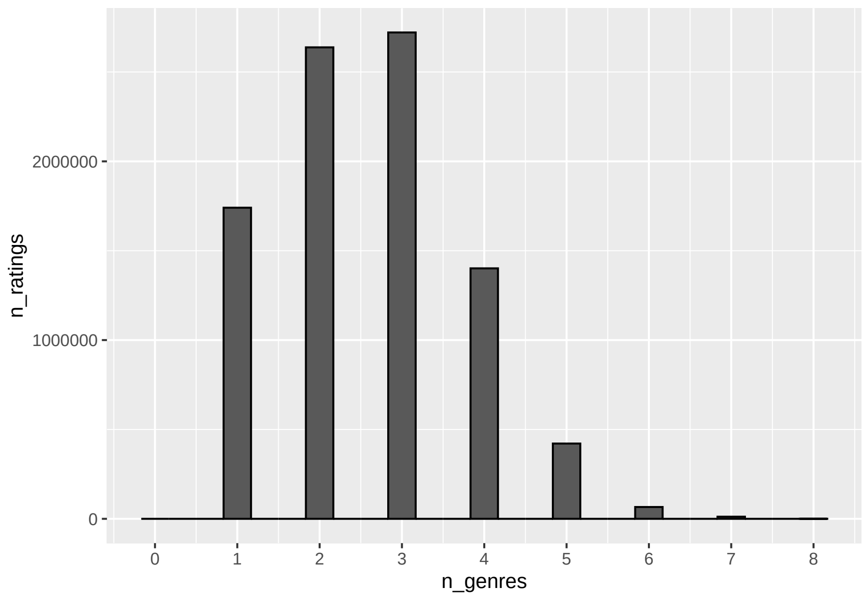

Figure 7: Number of individual genres per rating

The prevalence of a rating having more than one genre is 0.8066. Given that movies and ratings tend to have multiple genre's associated with them, the data exploration and model development will use genre combinations.

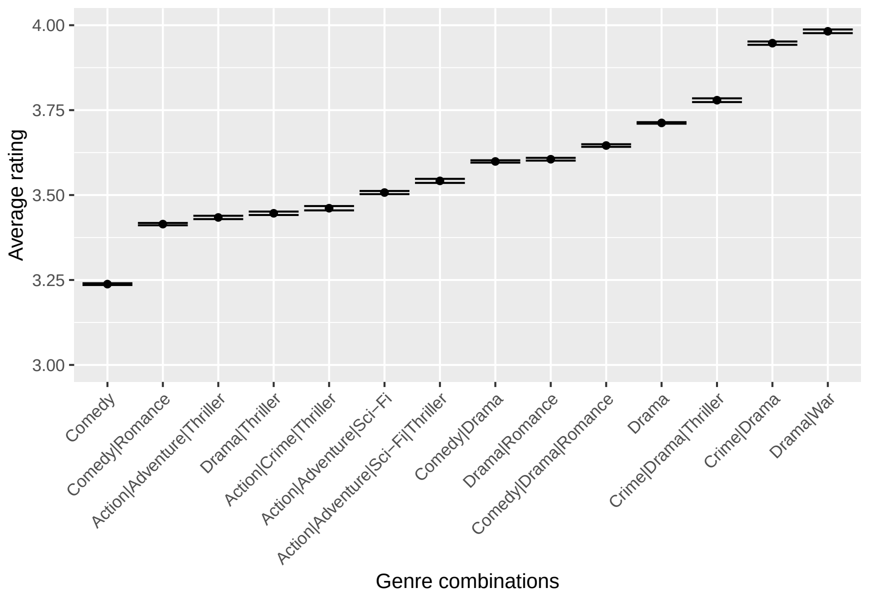

Figure 8: Average rating per genre combination over 100,000 ratings

There is evidence of a genre effect - Comedy has the lowest average rating and Drama|War has the highest average rating. The difference between a genre combination's average rating and the overall average rating, otherwise known as the genre effect or genre bias, will be taken into account when developing the model.

Release Year

To explore the impact of release year on the rating, the release year must first be extracted from the title feature using the code below:

```r
#extract release year from title
rel_yr <- "(?<=\\()\\d{4}(?=\\))"
edx <- edx %>%
  mutate(release_year = str_extract(title, rel_yr) %>%
           as.integer())
```

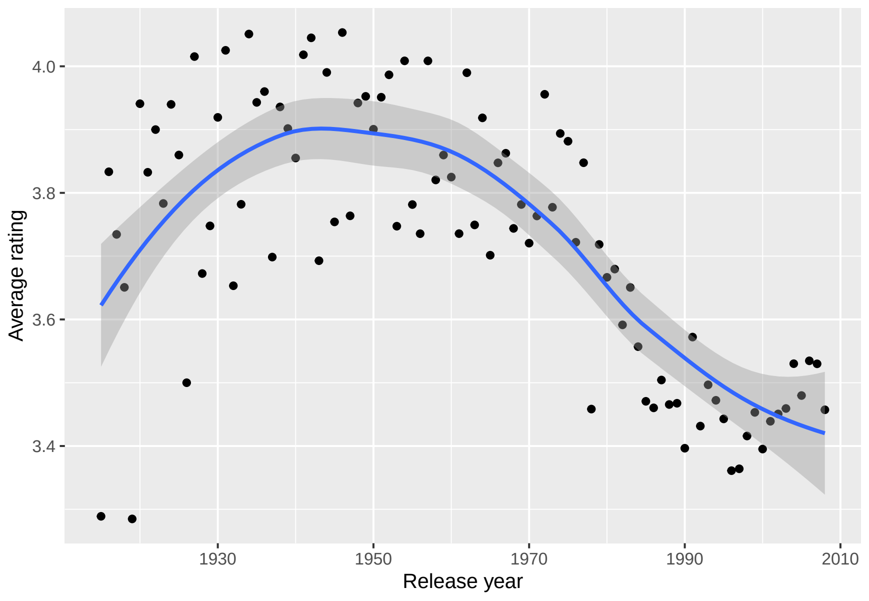

Figure 9: Average rating by release year

Release years range from 1915 to 2008 and older movies tend to have average higher ratings that newer movies, with the highest average ratings occurring between 1940-1950. It's possible the older a movie is, the more nostalgic users feel about it and therefore give it a higher rating. The difference between a release year's average rating and the overall average rating, otherwise known as the release year effect or release year bias, will be taken into account when developing the model.

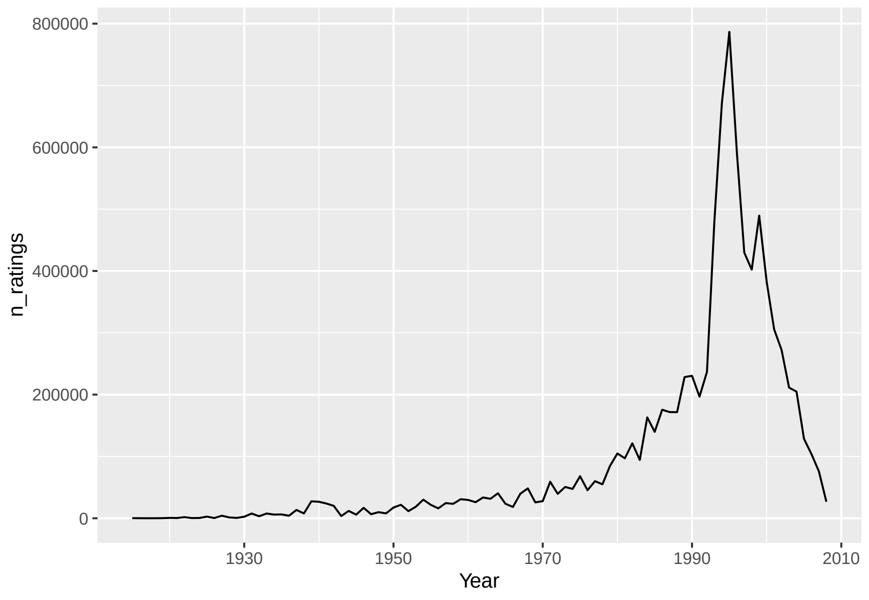

Figure 10: Number of ratings per release year

There are very few ratings in the edx set for movies released prior to 1970, with the highest number of ratings for movies released between 1990-2000. By overlaying Figures 9 and 10, one can see that the movies released between 1940-1950 have the highest average rating but few total ratings, while the movies released between 1990-2000 have the lowest average rating but the most total ratings. This implies the more ratings a movie has, the lower the rating is. This doesn't align with the earlier findings seen in Table 3 where the most rated movies generally had ratings that were higher than the average. Further exploration on the effect of time of rating or frequency of rating may be important to consider when developing the model.

Time of Rating

The timestamp feature provides the time and date a rating was given in Unix time, which is the number of seconds that have elapsed since 00:00:00 UTC on January 1, 1970. To explore the impact of the time of rating, the timestamp feature is converted to an easily understood date time format using the following code:

```r
#convert timestamp to date time
edx <- edx %>% mutate(timestamp = as_datetime(timestamp))
```

The first rating in the edx set was given on 1995-01-09 11:46:49 and the last was given on 2009-01-05 05:02:16. Figure 11 shows the average rating computed for each period of time - day, week, month, and year - plotted against the period of time:

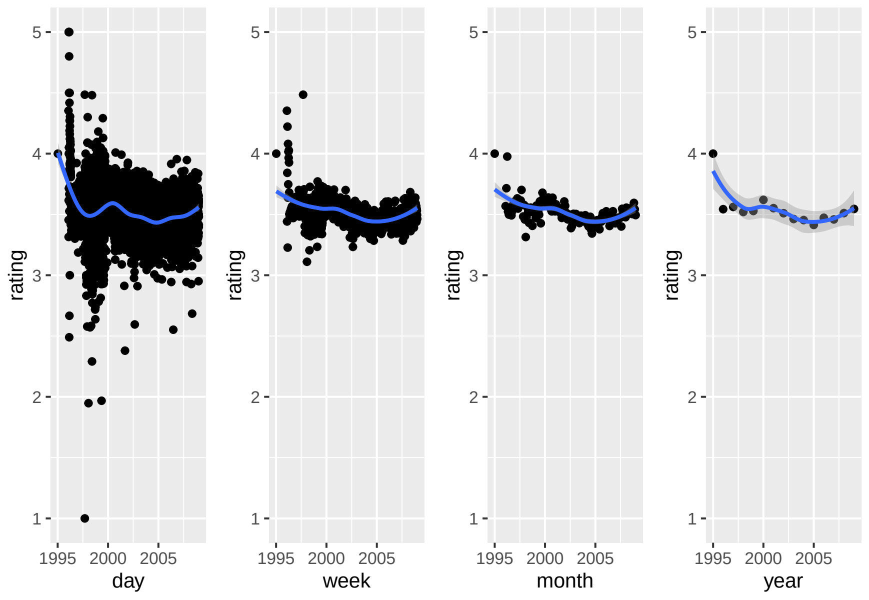

Figure 11: Average rating over time

All plots show evidence of a slight time of rating effect, however the plots using week, month, and year have a similar shape while day shows more exaggerated average ratings between 1995-2000. The impact of the time of rating on average rating, or time bias, will be taken into account when developing the model using average rating per week since it is the most granular time out of the time periods that have a consistent shape. The following code is used to create a new feature for the week the rating occurred:

```r
#create new column for week of rating
edx <- edx %>%
  mutate(date = round_date(timestamp, unit = "week"))
```

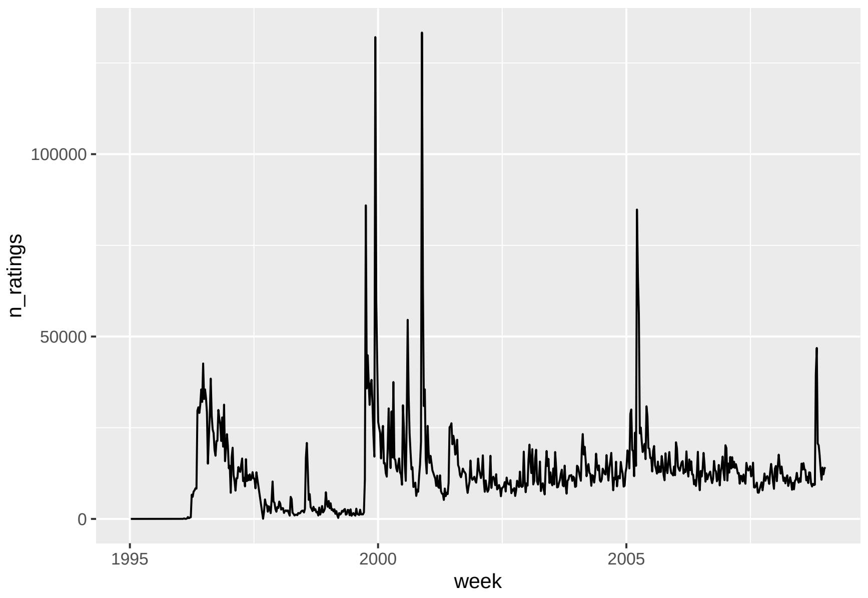

Figure 12: Number of ratings per week

There are a few noticeable peaks corresponding with a high number of ratings. By comparing Figure 11 for "week" and Figure 12, we can infer that a high number of ratings doesn't necessarily affect the average rating, as the average rating per week does not peak or even change drastically around the weeks that have a high number of ratings. Again, further exploration on the frequency of rating, while out of scope for this project, may be an interesting feature to consider for future development of this model.

Methods

Create train and test sets

The originally created train subset "edx" will be used to develop a machine learning model that will predict ratings. In order to avoid over training the model, edx will be further split into a train subset "train_set" and a test subset "test_set". The train subset will be used to develop the model and it will be tested on the test set throughout the process to test model accuracy.

```r
# create training and test sets - test set consists of 10% of edx data
set.seed(2024, 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,]

# Ensure userId and movieId in test_set set are also in train_set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
```

The train and test set contain the same users and movies. Any data removed from the test set is added back to the train set to maximize the data used for model development.

Loss Function

RMSE is a measure of the difference between values predicted by a model and actual values observed in the dataset. The equation for RMSE is defined as:

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

with $y_{u,i}$ representing the rating for movie i by user u, $\hat{y}_{\mathrm{u,i}}$ representing the prediction, and N representing the number of user/movie combinations, with the sum occurring over all the movie and user combinations. RMSE can be interpreted similarly to standard deviation - an RMSE of 1 indicates that the error in the model is equivalent to +/- 1 star. The goal of this project is to achieve an RMSE less than 0.8649. All models developed will be evaluated against this target and the difference to target will be tracked.

Least Squares Estimates

As seen in the Data Exploration and Analysis section, the distributions of movie ratings and user ratings are normal. According to Irizarry (2019), if data is bivariate normal then the conditional expectations follow the regression line, so these features can be predicted with linear regression. Release year and time of rating are linear and while genre is categorical, a line can be fitted to average rating per genre combination as seen by the trend in Figure 8. This project will use least squares estimates for all biases. The least squares estimate, or regression model, creates a line derived from the relationship between the outcome and feature. Each feature and its relationship to rating will be analyzed and a unique bias, represented by b, will be calculated.

Simple Average Model

The simplest model for a recommendation system is predicting the same rating for each movie. While this approach may seem overly simplistic, it establishes a baseline model to compare with future models. The mathematical equation for this model is:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

where $Y_{u,i}$ is the rating for movie i given by user u, μ is the "true" rating for all movies, and $\epsilon_{u,i}$ is the random error that is independently sampled by the same distribution centered at 0. The estimate of μ that minimizes RMSE is the least squares estimate, or the average of all ratings. Using the overall average rating as the predicted rating for all movies, the RMSE is:

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948

This model results in a predicted rating that is up to +/-1.0597 stars different than the actual rating.

Movie Effect

The earlier data exploration shows that different movies have different average ratings, and the previous model can be improved by adding a movie bias:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

where b_i represents the average rating for movie i. The least squares estimate of b_i is the mean of $Y_{u,i}$ - μ for each movie i. The following code is used to determine the bias associated with each movie and plot them as a histogram:

```r
#calculate movie bias
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating-mu))
#plot movie bias
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 30, data = ., color = I("black"))
```

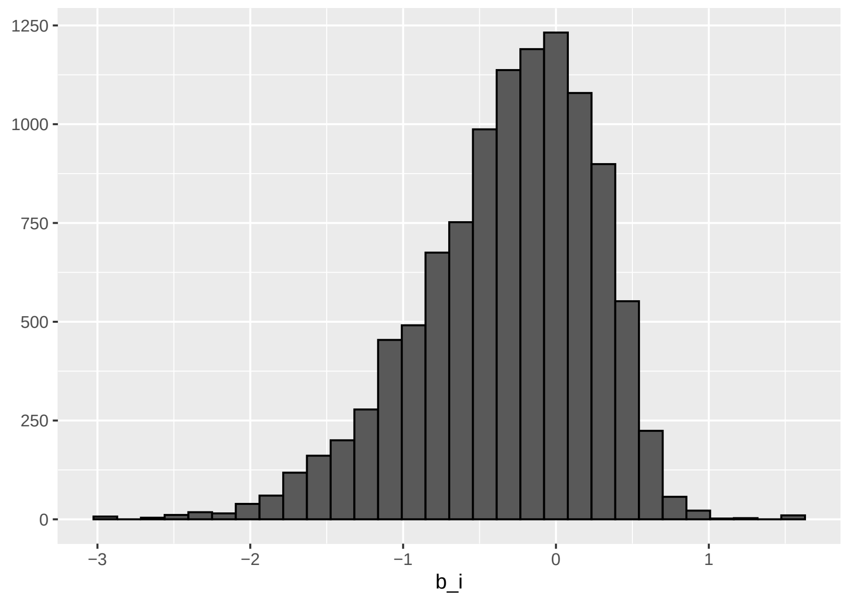

Figure 13: Movie biases

The bias is skewed to the left, indicating there are many movies with lower average ratings than the overall average rating of 3.5124. To demonstrate how to interpret b_i we will manually calculate the predicted rating for movieId 1. The bias for movieId 1, or b_1, is 0.4146. This means the movie's rating is 0.4146 higher than the overall average rating of 3.5124. Therefore the overall average rating for movieId 1 is 0.4146 + 3.5124 = 3.927. This is confirmed by pulling the rating for movieId 1 with the following code:

```r
train_set %>% filter(movieId == 1) %>% summarize(mean(rating))
```

```
##   mean(rating)
## 1        3.927
```

Adding movie bias to the predicted rating produces the following RMSE:

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948
Movie Effect	0.9424	0.0775

User Effect

The data exploration shows significant variation in average user rating, so the previous model can be improved by adding a user bias:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u represents the average rating for user u. The least squares estimate of b_u is the average of $Y_{u,i}$ - μ - b_i for each user u. The following code is used to compute the bias associated with each user and plot them as a histogram:

```
#calculate bias
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
#plot bias
user_avgs %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"))
```

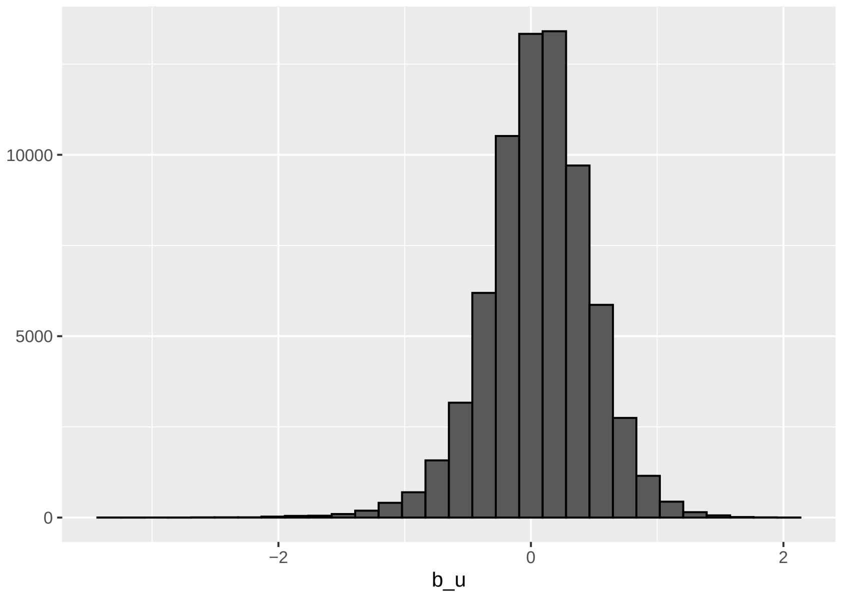

Figure 14: User biases

The user bias is centered around 0. Note that as a second features is added to the model, the bias range decreases. Adding user bias to the predicted ratings results in predictions that are outside of the possible range of ratings:

Table 8: lowest and highest predictions

x
-0.5868
6.1149

This can happen when multiple biases are combined in a model. The **clamp** function from the raster package is used to limit the predicted ratings to a low rating of 0.5 and a high rating of 5.0 to ensure all predicted ratings fall into the range of possible rating. Clamped ratings will be used for all RMSE calculations that include multiple biases. The resulting RMSE that includes user bias is:

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948
Movie Effect	0.9424	0.0775
Movie + User Effects	0.8636	-0.0013

While the goal RMSE has already been achieved, the other features will be explored in case they further reduce the RMSE.

Genre Effect

The data exploration shows that different genre combinations have different average ratings, so the previous model can be improved by adding a genre bias:

$$Y_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

where b_g represents the average rating for genre combination g. The least squares estimate of b_g is the average of $Y_{u,i}$ - μ - b_i - b_u for each genre combination g. The following code is used to determine the bias associated with each genre combination and vizualize the bias:

```r
#calculate bias
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
#plot bias
genre_avgs %>% qplot(b_g, geom ="histogram", bins = 30, data = ., color = I("black"))
```

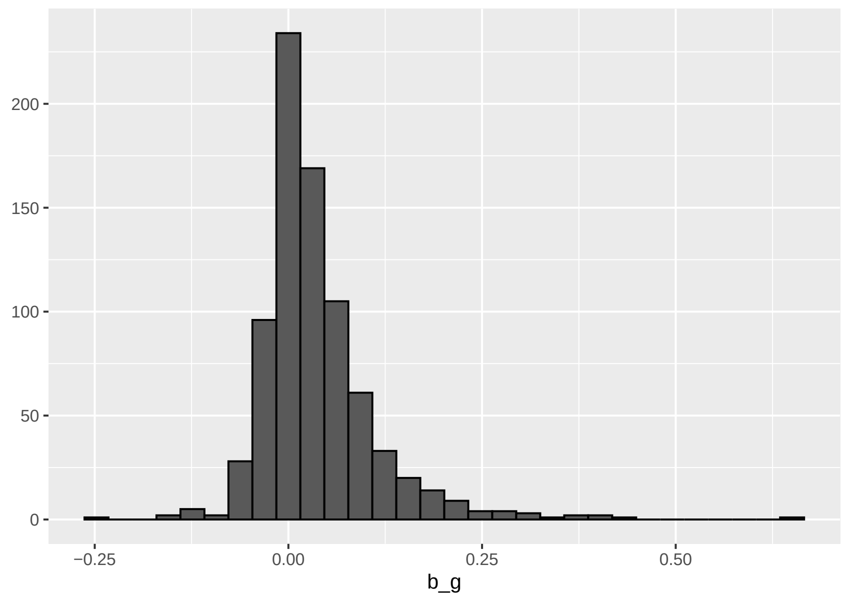

Figure 15: Genre biases

The genre bias, which also includes the impact of the movie and user biases, is slightly skewed to the right and is smaller than previous biases as most are within +/- 0.25. Adding genre bias to the predicted rating produces the following results:

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948
Movie Effect	0.9424	0.0775
Movie + User Effects	0.8636	-0.0013
Movie + User + Genre Effects	0.8633	-0.0016

The genre bias had a very small effect on the RMSE.

Release Year Effect

The data exploration shows that different release years have different average ratings, so the previous model can be improved by adding a release year bias:

$$Y_{u,i} = \mu + b_i + b_u + b_g + f(r_{u,i}) + \epsilon_{u,i}$$

where $f(r_{u,i})$ is a smooth function that represents the average rating for release year r. The least squares estimate of $f(r_{u,i})$ is the average of $Y_{u,i}$ - μ - b_i - b_u - b_g for each release year r. The following code is used to determine the bias associated with each release year, represented by b_r in the code, and visualize the bias:

```
#calculate release year bias
year_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(release_year) %>%
  summarize(b_r = mean(rating - mu - b_i - b_u - b_g))

#plot bias
year_avgs %>% qplot(b_r, geom ="histogram", bins = 30, data = ., color = I("black"))
```

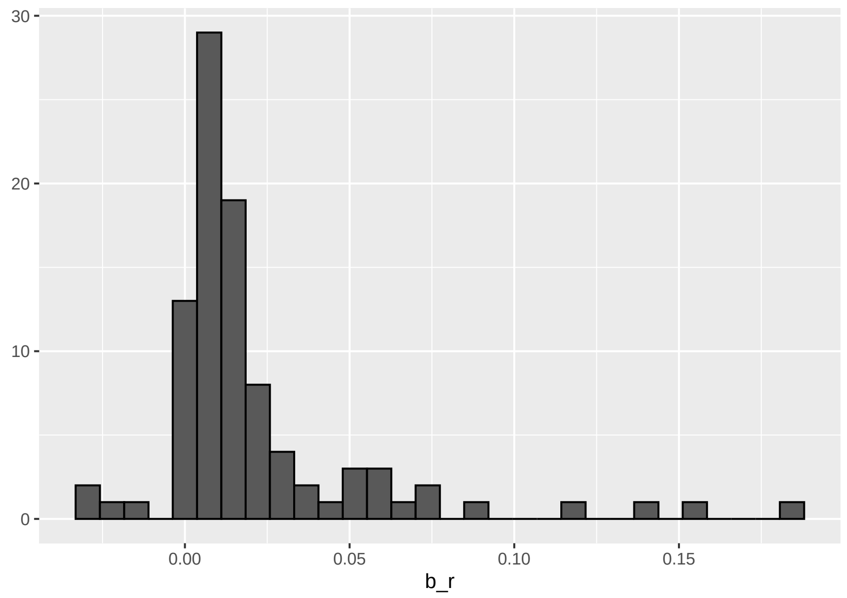

Figure 16: Release year bias

The distribution of release year bias is very small, slightly skewed to the right, and has a few outliers around + 0.15. This bias is not likely to change the model's performance drastically. Adding the release year bias to the predicted rating produces the following RMSE:

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948
Movie Effect	0.9424	0.0775
Movie + User Effects	0.8636	-0.0013
Movie + User + Genre Effects	0.8633	-0.0016
Movie + User + Genre + Release Year Effects	0.8631	-0.0018

The release year bias has a small positive effect on the RMSE.

Time of Rating Effect

The data exploration shows that different times of rating have different average ratings. For example, movie ratings can be influenced by current events so it's reasonable to include the time of rating in the model. The previous model can be improved by adding a time of rating effect, or time bias:

$$Y_{u,i} = \mu + b_i + b_u + b_g + f(r_{u,i}) + f(t_{u,i}) + \epsilon_{u,i}$$

where $f(t_{u,i})$ is a smooth function that represents the average rating for the time period t. The time period used in this evaluation is the week the rating was given by the user. The least squares estimate of $f(t_{u,i})$ is the average of $Y_{u,i}$ - μ - b_i - b_u - b_g - $f(r_{u,i})$ for each time period t. The following code is used to compute the bias associated with each week,, represented by b_t in the code, and plots these as a histogram:

```
#calculate time of rating bias
time_avgs <- train_set %>%
```

```r
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(year_avgs, by = "release_year") %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u - b_g - b_r))

#plot bias
time_avgs %>% qplot(b_t, geom ="histogram", bins = 30, data = ., color = I("black"))
```

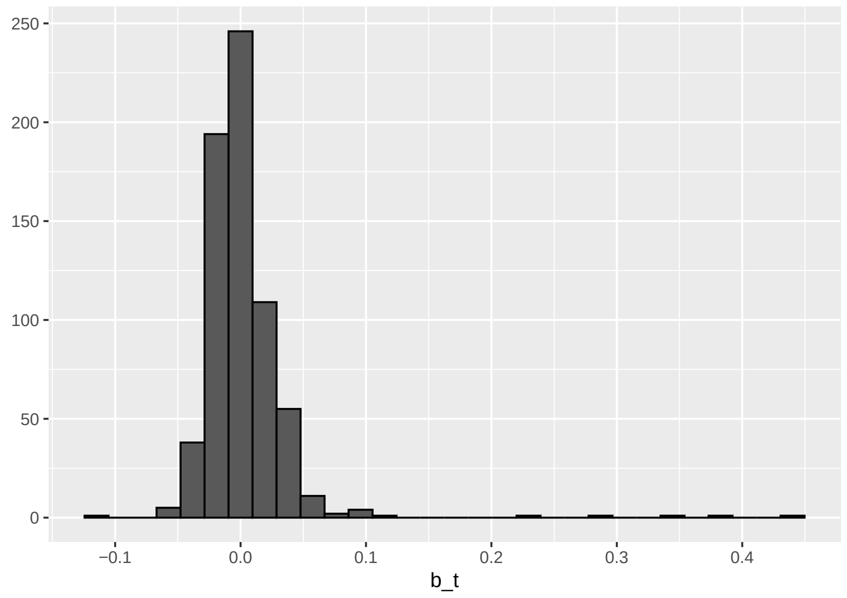

Figure 17: Release year biases

The time bias is centered around 0 and is very small at +/- < 0.05 with a few outliers to the right. While this appears to be a better predictor than release year given the symmetry of the bias, the very small range means it will likely not have a large effect on the model's performance. Adding a time bias to the predicted rating produces the following results:

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948
Movie Effect	0.9424	0.0775
Movie + User Effects	0.8636	-0.0013
Movie + User + Genre Effects	0.8633	-0.0016
Movie + User + Genre + Release Year Effects	0.8631	-0.0018
Movie + User + Genre + Release Year + Time Effects	0.8629	-0.0020

As predicted, the time of rating had very little effect on the model's performance, but the effect was overall positive so it will remain in the model.

Regularization

Motivation

With the model developed using movie, user, genre, release year, and time effects, we look at the data for insights to ensure that the model makes intuitive sense. First we look at the top 10 largest prediction mistakes using residuals. Residuals are calculated by subtracting the average predicted rating from the average actual rating and therefore show how far off the prediction is from the actual rating.

Table 13: Top 10 prediction mistakes

title	n_ratings	avg_pred	residual
Abbott and Costello Meet the Killer, Boris Karloff (1949)	1	3.8675	-3.368
Amati Girls, The (2000)	1	1.6469	3.353
Birds of America (2008)	1	1.9051	3.095
Son of Lassie (1945)	2	0.3353	2.915
Paralyzing Fear: The Story of Polio in America, A (1998)	1	3.8544	-2.854
Beau Pere (a.k.a. Stepfather) (Beau-père) (1981)	1	2.2508	2.749
Piñero (2001)	1	2.3933	2.607
Dead End (2003)	1	3.0943	-2.594
G (2002)	1	1.9268	2.573
Seven Girlfriends (1999)	1	3.0563	-2.556

These are all obscure movies with a low number of ratings. It is plausible that the model doesn't predict movies that only received one rating well because there are not many data points to reference in the model's development. Next we review the 10 highest rated movies ratings based on the model developed:

Table 14: 10 highest rated movies before regularization

title	n_ratings	avg_pred	residual
MC5*: A True Testimonial (2002)	1	5.264	-1.7638
Human Condition III, The (Ningen no joken III) (1961)	1	5.224	-0.2237
More (1998)	2	5.196	-0.4459
Proprietor, The (1996)	1	4.952	0.0477
Intended, The (2002)	1	4.898	-1.3984
Aerial, The (La Antena) (2007)	1	4.880	-1.8797
Parallel Sons (1995)	1	4.624	0.3763
Wrestler, The (2008)	1	4.499	-0.4986
Shawshank Redemption, The (1994)	2836	4.488	-0.0330
Godfather, The (1972)	1769	4.439	-0.0259

With the exception of "The Shawshank Redemption" and "The Godfather", the rest of these movies are obscure and have very few ratings. This doesn't make sense - the highest rated movies should in theory be major blockbusters that many people have seen and rated. Also, the residuals of the movies with only 1 or 2 ratings have large residuals, meaning that the predicted ratings are up to over 1 star off.

Method

Small sample size can lead to uncertainty (Irizarry, 2019). There is significant variation in the number of ratings per feature as previously explored, and the biases calculated for these features are less trustworthy when based on a small sample size of only a few ratings. To fix this, we can preform regularization on the biases. Regularization is a method of penalizing estimates that are formed using small sample sizes by adding a penalty, λ, to the sample size when calculating each bias (Irizarry 2019). The equation for the regularized movie bias is:

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

where n_i represents the number of ratings for movie i. When sample size n_i is very large, λ doesn't have an impact because $\lambda + n_i \approx n_i$. When sample size n_i is very small, λ has a large impact and it shrinks the estimated bias towards 0. Previously calculated biases for user, genre combination, release year, and time of rating are regularized similarly. Cross validation is used to select a value for λ that will minimize RMSE:

```r
#use cross validation to plot lambda vs rmse
lambdas <- seq(3, 7, 0.1)
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))
  b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_u - b_i - mu)/(n()+l))
  b_r <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(release_year) %>%
    summarize(b_r = sum(rating - b_g - b_u - b_i - mu)/(n()+l))
  b_t <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_r, by="release_year") %>%
    group_by(date) %>%
    summarize(b_t = sum(rating - b_r - b_g - b_u - b_i - mu)/(n()+l))
  preds <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_r, by = "release_year") %>%
    left_join(b_t, by = "date") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_r + b_t) %>%
    pull(pred)
  return(RMSE(preds, test_set$rating))
})

qplot(lambdas, rmses)
```

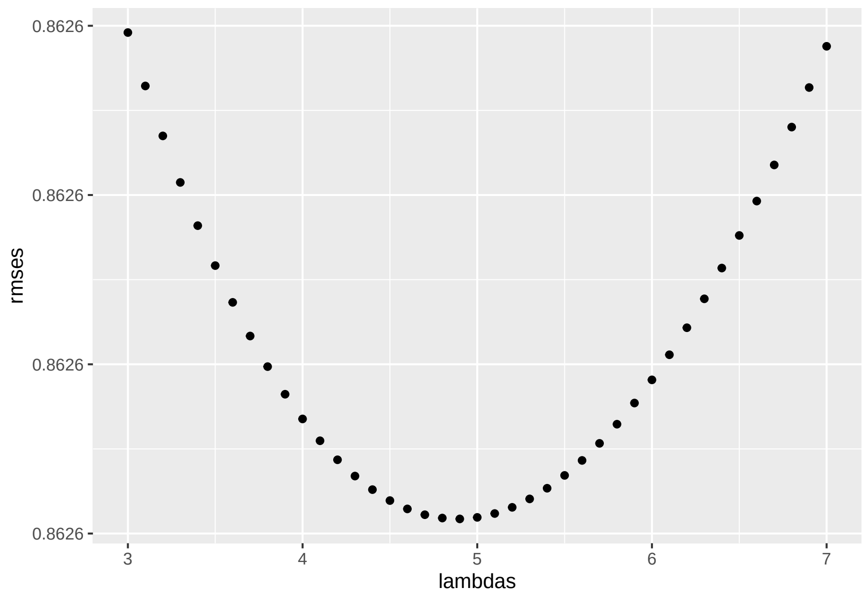

Figure 18: Lambdas vs RMSEs

The value of λ that minimizes the RMSE is 4.9.

Impact

Regularization has a positive impact on the model further reducing the RMSE. The model, developed using all features and only with the edx subset to avoid over training, surpasses the goal RMSE.

Model	RMSE	Difference
Goal RMSE	0.8649	0.0000
Average Rating	1.0597	0.1948
Movie Effect	0.9424	0.0775
Movie + User Effects	0.8636	-0.0013
Movie + User + Genre Effects	0.8633	-0.0016
Movie + User + Genre + Release Year Effects	0.8631	-0.0018
Movie + User + Genre + Release Year + Time Effects	0.8629	-0.0020
Regularized Movie + User + Genre + Release Year + Time Effects	0.8626	-0.0023

After performing regularization there are still some top 10 movies with only 1 or 2 ratings, however their corresponding residuals are small, indicating that the predictions are accurate. There are also more top 10 movies with many ratings, which is expected as regularization penalized small sample sizes.

Table 16: 10 highest rated movies after regularization

title	n_ratings	avg_pred	residual
Proprietor, The (1996)	1	4.875	0.1250
More (1998)	2	4.814	-0.0639
Parallel Sons (1995)	1	4.548	0.4523
Human Condition III, The (Ningen no joken III) (1961)	1	4.509	0.4913

title	n_ratings	avg_pred	residual
Shawshank Redemption, The (1994)	2836	4.488	-0.0335
Godfather, The (1972)	1769	4.438	-0.0244
Wrestler, The (2008)	1	4.386	-0.3858
Schindler's List (1993)	2243	4.382	-0.0439
Usual Suspects, The (1995)	2101	4.356	0.0298
Third Man, The (1949)	283	4.341	-0.0070

Results

The validation subset is updated with release year and rating time to match the edx subset:

```r
#update final holdout test set to match edx features
#release year
final_holdout_test <- final_holdout_test %>%
  mutate(release_year = str_extract(title, rel_yr) %>%
           as.integer())
#date time and week rating was given
final_holdout_test <- final_holdout_test %>%
  mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(date = round_date(timestamp, unit = "week"))
```

The final step of the project is to run the previously developed model, which uses all the features of the dataset and has been regularized to account for small sample sizes, on the validation subset "final_holdout_test", which has not been used to develop the model, and calculate RMSE:

```r
#train final algorithm using full edx dataset
mu <- mean(edx$rating)
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+1))
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+1))
b_r <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(release_year) %>%
  summarize(b_r = sum(rating - b_u - b_i - mu)/(n()+1))
b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_r, by="release_year") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_r - b_u - b_i - mu)/(n()+1))
b_t <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_r, by="release_year") %>%
  left_join(b_g, by="genres") %>%
  group_by(date) %>%
  summarize(b_t = sum(rating - b_g - b_r - b_u - b_i - mu)/(n()+l))
predicted_ratings <- final_holdout_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_r, by = "release_year") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "date") %>%
  mutate(pred = mu + b_i + b_u + b_r + b_g + b_t) %>%
  pull(pred)
range(predicted_ratings)
preds_clamped <- clamp(predicted_ratings, 0.5, 5)
final_rmse <- RMSE(preds_clamped, final_holdout_test$rating)
rmse_results <- tibble(Model="Goal RMSE", RMSE = goal_rmse, Difference = 0)
rmse_results <- bind_rows(rmse_results,
                          data.frame(Model="Final RMSE",
                                     RMSE = final_rmse,
                                     Difference = final_rmse - goal_rmse))
```

Model	RMSE	Difference
Goal RMSE	0.8649	0.000
Final RMSE	0.8639	-0.001

The final RMSE when calculated on the validation subset is -0.001 less than the goal RMSE. The RMSE can be interpreted as the amount of stars that a prediction may be from the true rating. The final model accurately predicts 71.0085% of models within 0.8639 stars.

Conclusions

For this project I created a movie recommendation system that predicted ratings users give to movies. I used least squares regression, commonly referred to as least squares estimates throughout this paper, to determine the effect each feature in the dataset has on the rating. Due to the large size of the dataset, this was the only feasible method that my computer's processing power allowed. The loss function used to evaluate accuracy of the model was RMSE, with a final RMSE of 0.8639 when the developed model was run on previously unseen data. The RMSE is still quite high, and this model will result in some movies being recommended to some users that they will not like.

If the dataset was smaller or my computer more powerful, I would like to use other established machine learning algorithms that can factor in multiple features and understand how these features interact with each other, rather than only focusing individually on each feature's relationship to rating. Options for future work includes matrix factorization, which identifies and uses patterns in the data to further explain variability. I'd like to predict ratings using other linear models, like lm and glm, and compare their results with the results of this project. I'd also like to explore the k-nearest neighbors model as it is a non-linear regression that takes a specific number of nearest neighbors into account when determining the best fit of the model. Alternatively, It would be interesting to explore categorical models; this would require the predictions to be translated into a binary outcome - the user either "likes" the movie that is recommended to them or not. Once the outcome is binary, sensitivity, specificity, and F_1 can be calculated for the model and optimized through a categorical model like a classification tree or random forest. Ideally multiple models would be tested, the best model selected, all parameters optimized through cross validation, and then a final k-cross validation performed to maximize the model's performance. Finally, recommender algorithms have been developed to solve problems like this one; I would like to explore the recommenderlab package which

was built to specifically interpret and predict ratings (Hahsler, 2016).
There are likely many other variables and interactions between variables that could be studied to further improve the model. I believe the rate of rating, or the frequency of which a movie was rated, is one such variable that would have a positive effect on the model, as previously exploration suggested the more a movie was rated the better the rating. To explore the rate of rating, the original movielens data set would need to be modified prior to splitting into train and test sets using the following code:

```r
#calculate rate of rating
movielens <- movielens %>%
  mutate(release_year = str_extract(title, "(?<=\\()\\d{4}(?=\\))") %>%
  as.integer()) %>%
  add_count(title) %>% #count title
  mutate(years = 2009 - release_year, rate = n/years) # divide title count by years movie has existed

#plot rating vs rate
movielens %>% group_by(title) %>%
  summarize(n = n(), years = 2009 - first(release_year),
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  ggplot(aes(rate, rating)) +
  geom_point() +
  geom_smooth()
```

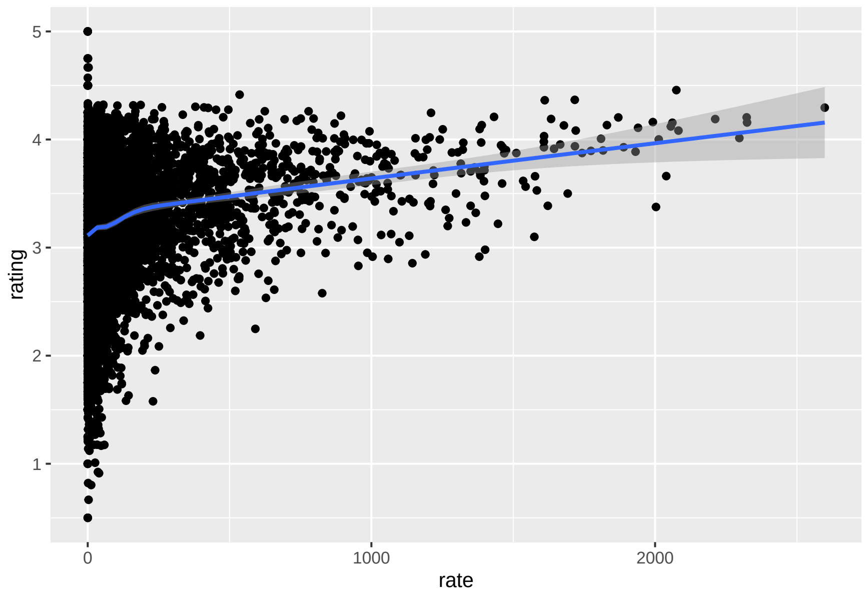

Figure 19: Rate vs rating

This plot clearly shows a relationship between the frequency of rating and the average rating, with higher average ratings for movies that are rated more frequently. This relationship is linear and very stable; it may be a more effective feature than time of rating or release year.

References

Hahsler, M. (2016). *recommenderlab: An R Framework for Developing and Testing Recommendation Algorithms.* https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf.

Irizarry, R.A. (2019). *Introduction to Data Science Data Analysis and Prediction Algorithms with R.* https://rafalab.dfci.harvard.edu/dsbook/.