EdX Data Science Capstone Project: Movielens

Christian D. Smithson

5/31/2020

Contents

Preface

This project is part of the capstone course for the Data Science Professional certificate series of courses made available through EdX by HarvardX[1].

The source code for the project along with the Rmarkdown report and pdf report are available on my GitHub.

[1] https://www.edx.org/professional-certificate/harvardx-data-science

Chapter 1

Introduction

"You may also like..." Unless you've been living under a rock, you should be very familiar with that phrase. One strategic driver of business success is the ability to give your customers what they want or recommend items they may not even realize that they want yet. Being able to determine what customers like and recommend similar items can drive customer retention, satisfaction, and, ultimately revenue growth. The systems needed to elevate customers' awareness of other products that they may want or need are referred to as "recommendation systems". Having a robust recommendation system creates a competitive advantage for any firm and the importance of recommendation systems has been realized in almost every industry. Whether you are shopping on Amazon or browsing for the next thing to watch on Netflix, we have all experienced recommendation systems at some point.

The purpose of this project is to explore some of the data science techniques used when building recommendation systems. We will be looking at data consisting of movie ratings and explore how we can use that data to predict what a movie goer's rating for a particular film would be.

1.1 Data Description

The data used in this project was made available by GroupLens[1]. GroupLens is a "research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems."

The full dataset[2] provided by GroupLens contains approximately 27 million ratings applied to 58,000 movies by 280,000 users gathered from the Movielens[3] website.

For the purposes of this project we will be using a smaller dataset[4] consisting of 10 million ratings applied to 10,000 movies by 72,000 users.

[1] https://grouplens.org/
[2] https://grouplens.org/datasets/movielens/latest/
[3] https://movielens.org/
[4] https://grouplens.org/datasets/movielens/10m/

1.2 Evaluating our models

For each model we run, we will be measuring accuracy by calculating the RMSE (root mean squared error):

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

Where, N is number of ratings, $y_{u,i}$ is rating by user u for movie i and $\hat{y}_{u,i}$ is the prediction of the rating by user u for movie i.

The RMSE penalizes large deviations from the mean and is useful in cases where small errors are not a concern. One advantage of the RMSE, as opposed to other metrics such as MSE, is that the error has the same unit of measure as the outcome.

The model with the lowest RMSE will be used as the final model.

1.3 Analysis approach

The general approach for this data science project was:

1. Extract, transform, and load the data:

The data used was downloaded from the GroupLens website and split into two sets, one for training (`edx`) and another for validation (`validation`). The `edx` set was subdivided further into one set used for training (`train_set`) and one set for testing (`test_set`).

2. Analyze the data to understand structure, characteristics, and unique features:

In this step we will take a look at the data structure and features to determine variables that will be useful for constructing our models.

3. Create and evaluate models using insights gained from exploratory analysis;

Multiple models will be fit to the data and the model with the lowest RMSE on the `test_set` will be selected as the final model. That model will then be retrained on the entire `edx` set and used to make predictions for the `validation` set.

Chapter 2

Analysis

2.1 Extract, transform, and load data

Below is the code used to download the data and create the different data partitions used for this project (`edx`, `validation`, `train_set`, `test_set`). This data comes to us in tidy format and has been cleaned up already. This rarely occurs in the real world and any data cleaning would occur during this phase of the analysis.

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

# Note: this process could take a couple of minutes

repo = "http://cran.us.r-project.org"

if(!require(tidyverse)) install.packages("tidyverse", repos = repo)
if(!require(caret)) install.packages("caret", repos = repo)
if(!require(data.table)) install.packages("data.table", repos = repo)
if(!require(recosystem)) install.packages("recosystem", repos = repo)
if(!require(ggplot2)) install.packages("ggplot2", repos = repo)
if(!require(lubridate)) install.packages("lubridate", repos = repo)

# 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)
```

```r
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],
         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")
test_index <- createDataPartition(
  y = movielens$rating, times = 1, p = 0.1, list = FALSE)

edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

# Remove unwanted objects
rm(dl, ratings, movies, test_index, temp, movielens, removed)

# Create test and train sets from edx set. Test set will be 10% of edx set.
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,]

# Make sure userId and movieId in test 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)

# Remove unwanted objects
rm(test_index, temp, removed)
```

2.2 Explore the data

Now that we have the data downloaded, we can begin to explore the structure and characteristics to help inform our model building process.

We will begin by taking a quick look at the structure of the data using the following code:

```r
str(edx, vec.len = 1)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 ...
##  $ movieId  : num  122 185 ...
##  $ rating   : num  5 5 ...
##  $ timestamp: int  838985046 838983525 ...
##  $ title    : Factor w/ 10680 levels "'burbs, The (1989)",..: 1310 6762 ...
##  $ genres   : chr  "Comedy|Romance" ...
```

From the output we can see that there are 9,000,055 observations with 6 variables:

`userID`, `movieID`, `rating`, `timestamp`, `title`, and `genres`

Lets take a peek at the first six records:

```r
head(edx) %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("scale_down", "striped")) %>%
  column_spec(1:7, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

	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

2.2.1 Ratings

Since our objective is to predict movie ratings we need to get an idea what the rating data looks like:

```
summary(edx$rating) %>%
  as.matrix() %>%
  t() %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:6, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.5	3	4	3.512465	4	5

From the output we can see that the ratings range from 0.5 to 5 with the mean rating being approximately 3.5. This is a common scale for consumer ratings and is widely used in many industries. Lower stars mean lower perceived quality of the good or service and higher stars mean a greater perceived quality of the good or service.

2.2.2 Genres

We'll investigate the genres variable next, although it is not used in the modeling process.

```
edx %>%
  summarize(n_genres = n_distinct(genres)) %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

n_genres
797

There are 797 unique genres. How can this be? Upon further investigation its apparent that movies can be classified in to multiple genres. These unique combinations of genres create 797 unique classifications. Additionally, there are "genres" that rate higher on average than others. Below is the top 10 highest rated, on average, genres:

```r
edx %>%
  group_by(genres) %>%
  summarize(mean_rating = mean(rating)) %>%
  top_n(mean_rating, n = 10) %>%
  arrange(desc(mean_rating)) %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

genres	mean_rating
Animation|IMAX|Sci-Fi	4.714286
Drama|Film-Noir|Romance	4.304115
Action|Crime|Drama|IMAX	4.297068
Animation|Children|Comedy|Crime	4.275429
Film-Noir|Mystery	4.239479
Crime|Film-Noir|Mystery	4.216803
Film-Noir|Romance|Thriller	4.216470
Crime|Film-Noir|Thriller	4.210157
Crime|Mystery|Thriller	4.198981
Action|Adventure|Comedy|Fantasy|Romance	4.195557

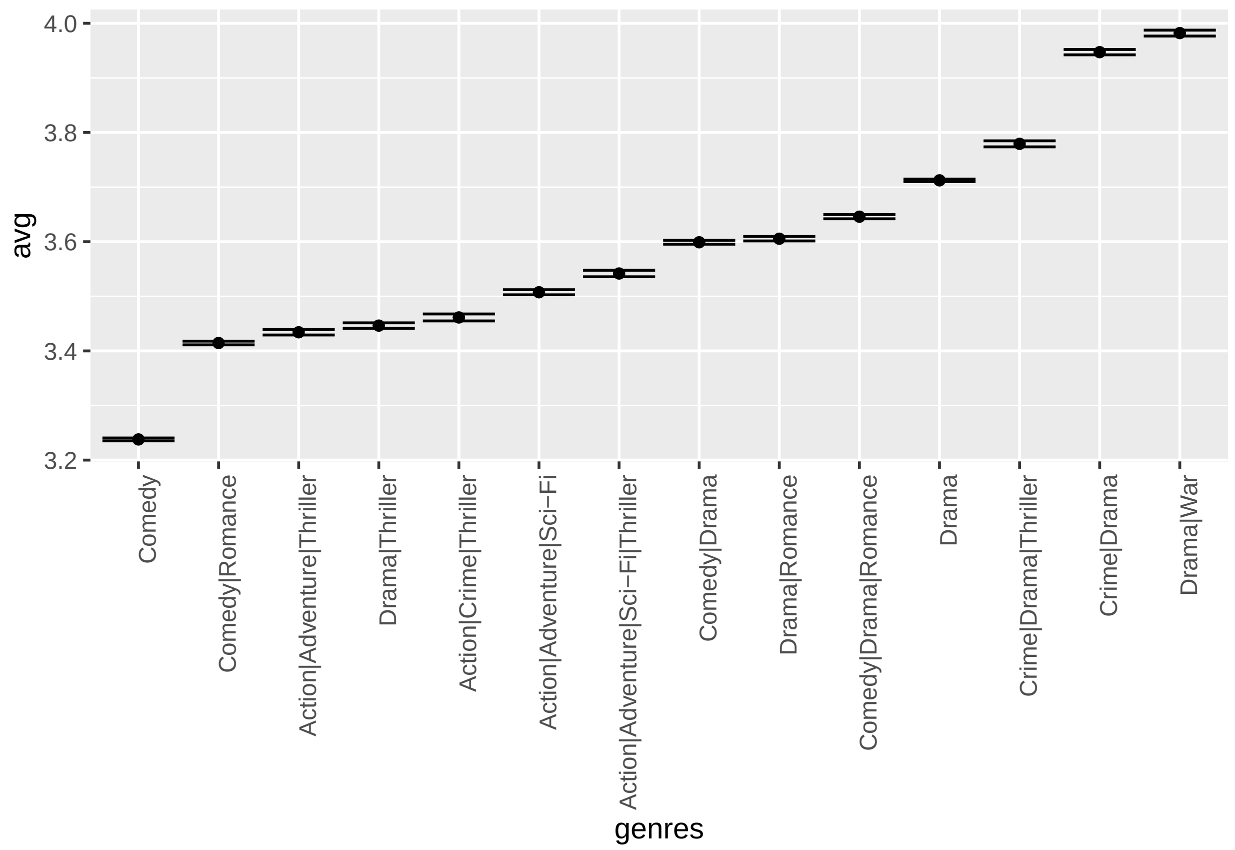
Conversely, the table below shows the 10 lowest rated, on average, genres:

```r
edx %>%
  group_by(genres) %>%
  summarize(mean_rating = mean(rating)) %>%
  top_n(mean_rating, n = -10) %>%
  arrange(mean_rating) %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

genres	mean_rating
Documentary|Horror	1.449112
Action|Animation|Comedy|Horror	1.500000
Action|Horror|Mystery|Thriller	1.607034
Comedy|Film-Noir|Thriller	1.642857
Action|Drama|Horror|Sci-Fi	1.750000
Adventure|Drama|Horror|Sci-Fi|Thriller	1.751152
Action|Adventure|Drama|Fantasy|Sci-Fi	1.903509
Action|Children|Comedy	1.910232
Action|Adventure|Children	1.915048
Adventure|Animation|Children|Fantasy|Sci-Fi	1.924747

We can visualize this variability by taking a subset of the genres that have greater than 100,000 ratings and plotting the average rating:

```r
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



These unique "genre effects" could potentially be used to further extend the predictive ability of our recommendation system, but will not be used in this project.

2.2.3 Users

Next, we can explore some of the structure of `userID`:

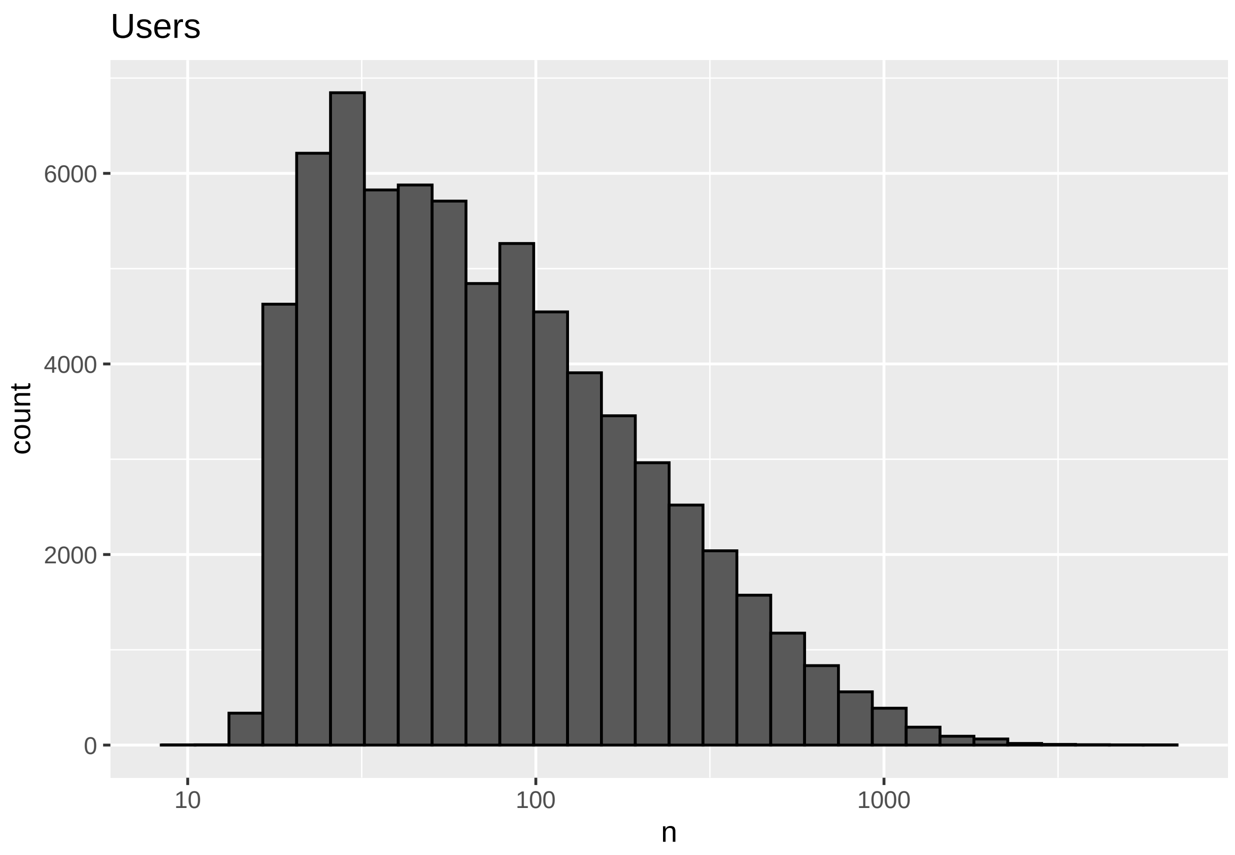
```
edx %>%
  summarize(n_users = n_distinct(userId)) %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

n_users
69878

Based on the output, there are 69,878 unique users.

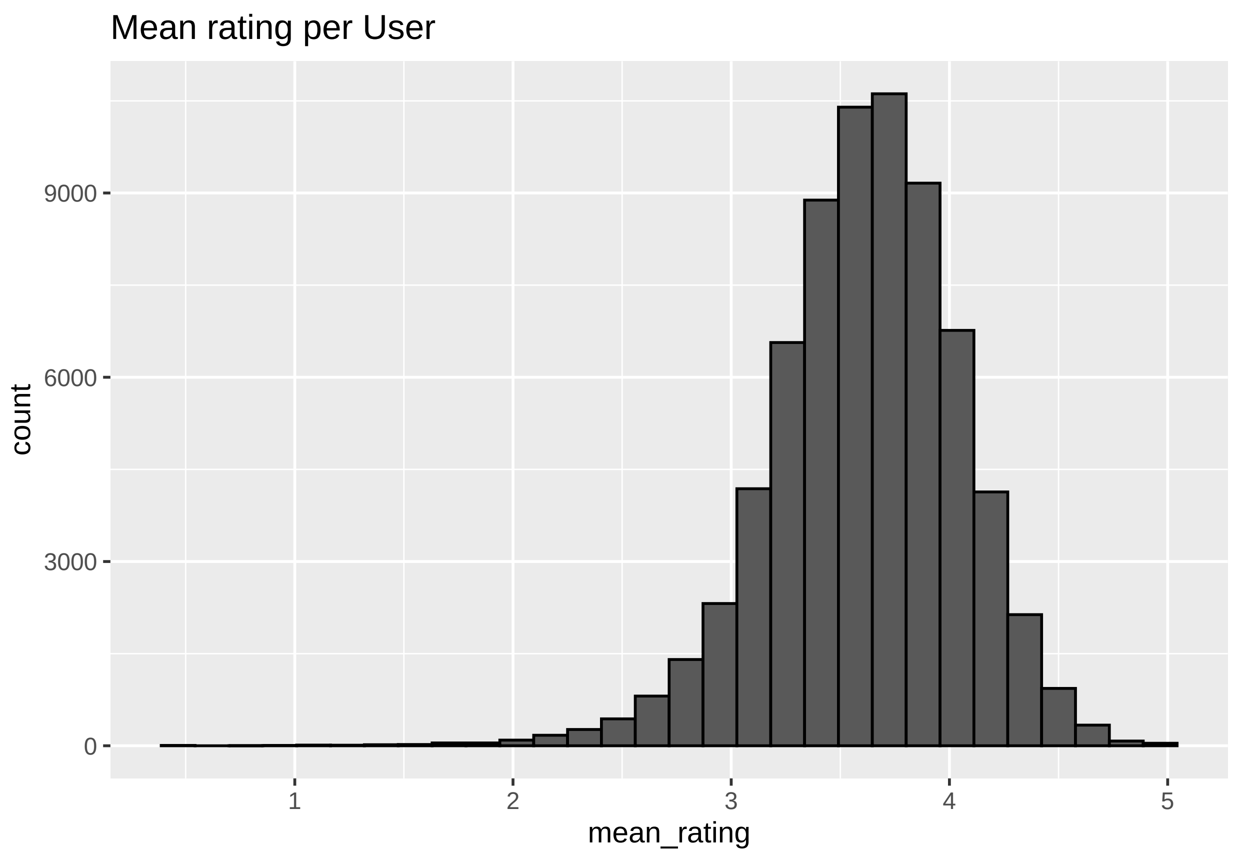
To understand our users a little better lets take a look at the distribution of ratings per user:

```r
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



Looking at the plot, most users have between 10 and 100 ratings, while some users have given over 1,000 ratings! Who has that much time?!?

```r
edx %>%
  group_by(userId) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins = 30, color = "black") +
  ggtitle("Mean rating per User")
```



The above plot shows users and their mean rating. From the distribution we can see that some users give higher mean ratings than others, while some are lower. Some users give everything a 5 star rating while others give everything a 2 star or less rating. There appears to be enough variability to use a user's individual rating behavior to predict how they may rate something new. We'll call this effect the "user effect" or user "bias".

One caveat to note is that if a user has rated very few movies, then the estimate of their "user effect" could potentially be over-weighted. We can can penalize the low rating frequency of users through a process known as regularization that will be explored later.

2.2.4 Movies

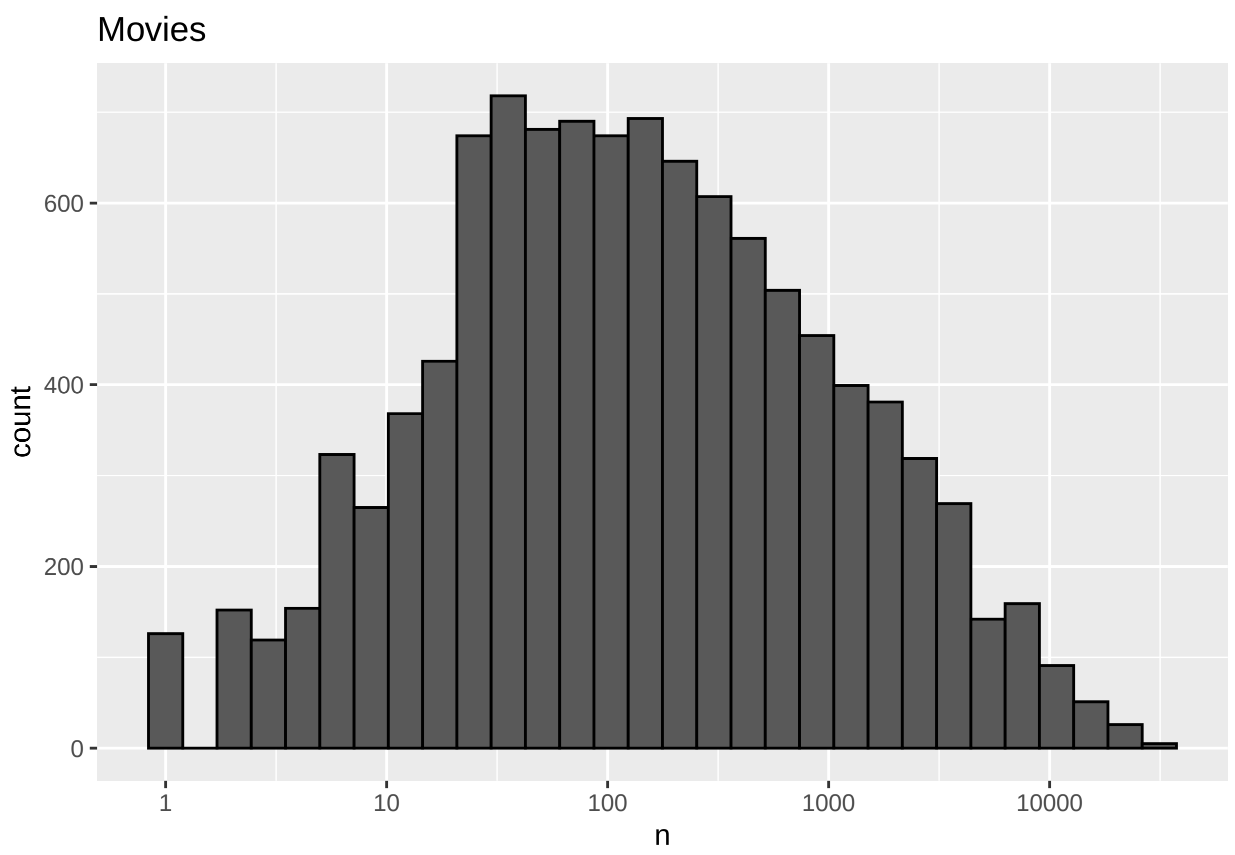
```r
edx %>%
  summarize(n_movies = n_distinct(movieId)) %>%
  knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

n_movies
10677

Based on the output, there are 10,677 unique movies.

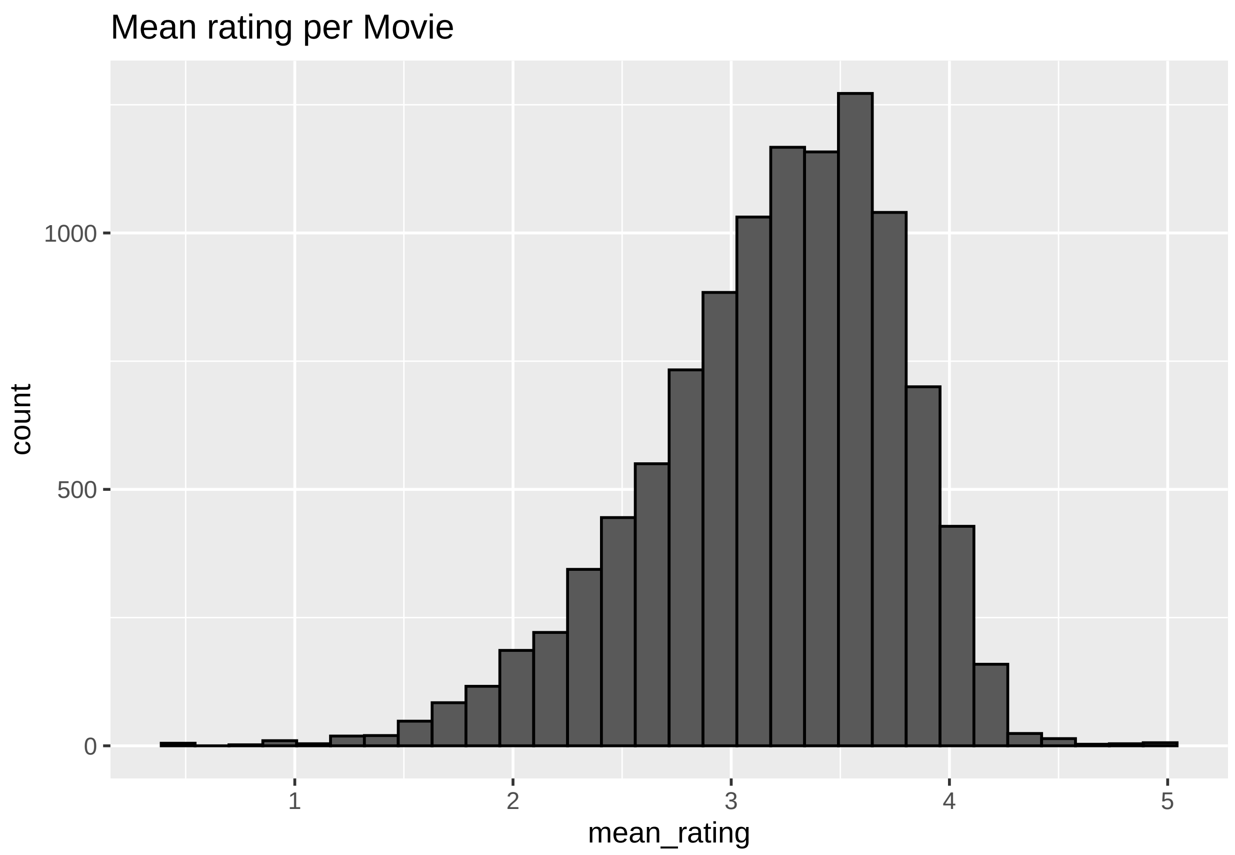
To understand the movies in our dataset a little better lets take a look at the distribution of ratings per movie:

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



As we can see, some movies have over 10,000 ratings while others have less than 10. This makes sense given some movies are more popular than others and thus will have more ratings.

```
edx %>%
  group_by(movieId) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins = 30, color = "black") +
  ggtitle("Mean rating per Movie")
```



Similar to mean ratings given by users, we also see that some movies tend to, on average, rate higher than others. Likewise, some rate lower than others, on average. This variability gives us the ability to use a "movie effect" or movie "bias" to predict the rating a user may give a film based on how others tend to rate that movie.

Similar to low frequency users, if a film has very few ratings, then the estimate of it's "movie effect" could potentially be over-weighted. We can can penalize the low rating frequency movies through regularization.

When we begin to think about how we can forecast movie ratings, we can imagine that we are dealing with a matrix consisting of users in the rows and movies in the columns. See example below:

```
keep <- edx %>%
  dplyr::count(movieId) %>%
  top_n(5) %>%
  pull(movieId)
```

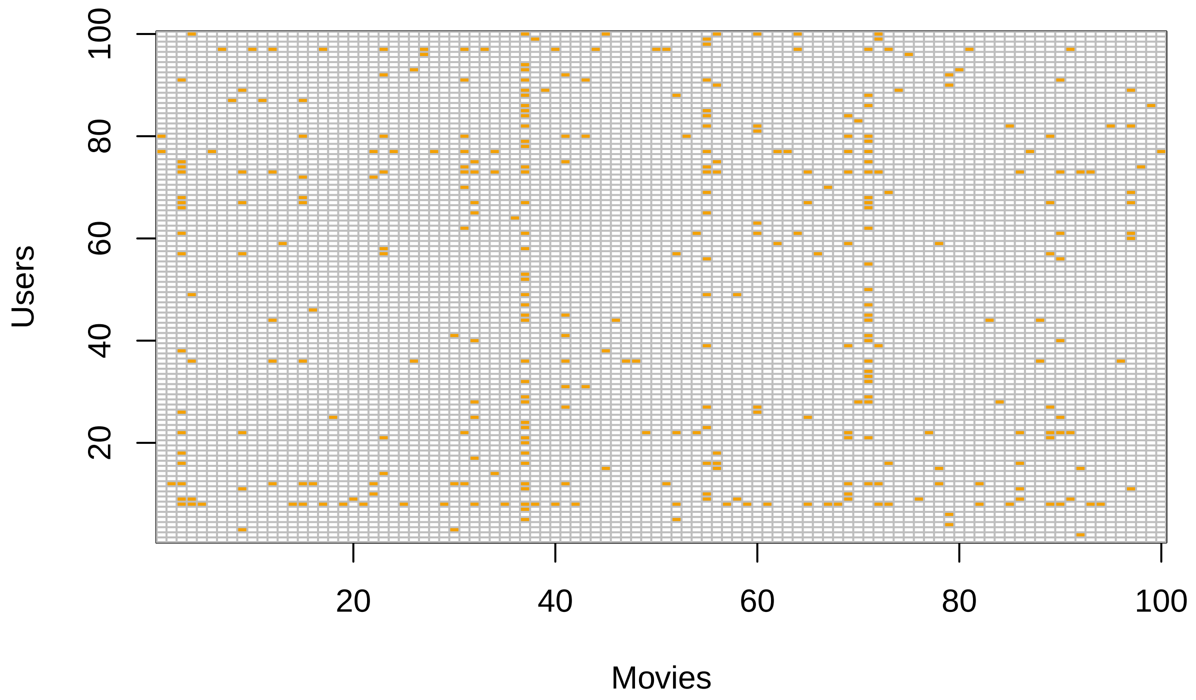
```r
tab <- edx %>%
  filter(userId %in% c(13:20)) %>%
  filter(movieId %in% keep) %>%
  select(userId, title, rating) %>%
  spread(title, rating)

tab %>% knitr::kable() %>%
  kable_styling(latex_options=c("scale_down", "striped")) %>%
  column_spec(1:6, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption, The (1994)	Silence of the Lambs, The (1991)
13	NA	NA	4	NA	NA
16	NA	3	NA	NA	NA
17	NA	NA	NA	NA	5
18	NA	3	5	4.5	5
19	4	1	NA	4.0	NA

In the output we can see that some have values (these are observed user/movie interactions). However, there is also quite a few NAs. Our goal is to fill in these NAs. For example, user *13* gave *Pulp Fiction* four stars (although I'd give it five stars because its an epic film). If user *13* gave *Pulp Fiction* 4 stars, then what would they rate *The Shawshank Redemption*? *Jurassic Park*? *Forrest Gump*?

If we sample a bit more and plot this matrix we can get a real sense of just how sparse this matrix is:

```r
users <- sample(unique(edx$userId), 100)
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```



The colored cells represent a user/movie interaction and the blank cells are the interactions that we must predict by using what we observe in the data (the colored cells).

2.3 Modeling Approach

The modeling approach will be to begin with the simplest linear models such as just using the average for all predictions all the way to more complex modeling techniques such as matrix factorization. Through each step we will add predictor variables or use a different technique and measure the RMSE.

2.3.1 Naive Model

The simplest, and likely the worst performing, technique is to predict that every movie will get the same rating, the average rating for all movies. The formula for this model would be:

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

Where, $\hat{Y}_{i,u}$ is the predicted rating for movie i by user u, and μ is the average rating for all movies. $\epsilon_{i,u}$ is a random error term. This model implies that there is no unique rating differences for different movies and all variability is explained by randomness (we subjectively know this to be false because some movies are simply better than others).

2.3.2 Movie Effect Model

The next model to test is one that incorporates the movie effect or "bias" b_i. We'll use the initial model and add in this movie effect. The formula follows:

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

Where, $\hat{Y}_{i,u}$ is the predicted rating for movie i by user u, μ is the average rating for all movies, and b_i is the movie effect. $\epsilon_{i,u}$ is a random error term.

2.3.3 Movie & User Effect Model

The next model will incorporate the user effect or "bias" b_u in to the movie effect model that has already been developed. The formula follows:

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

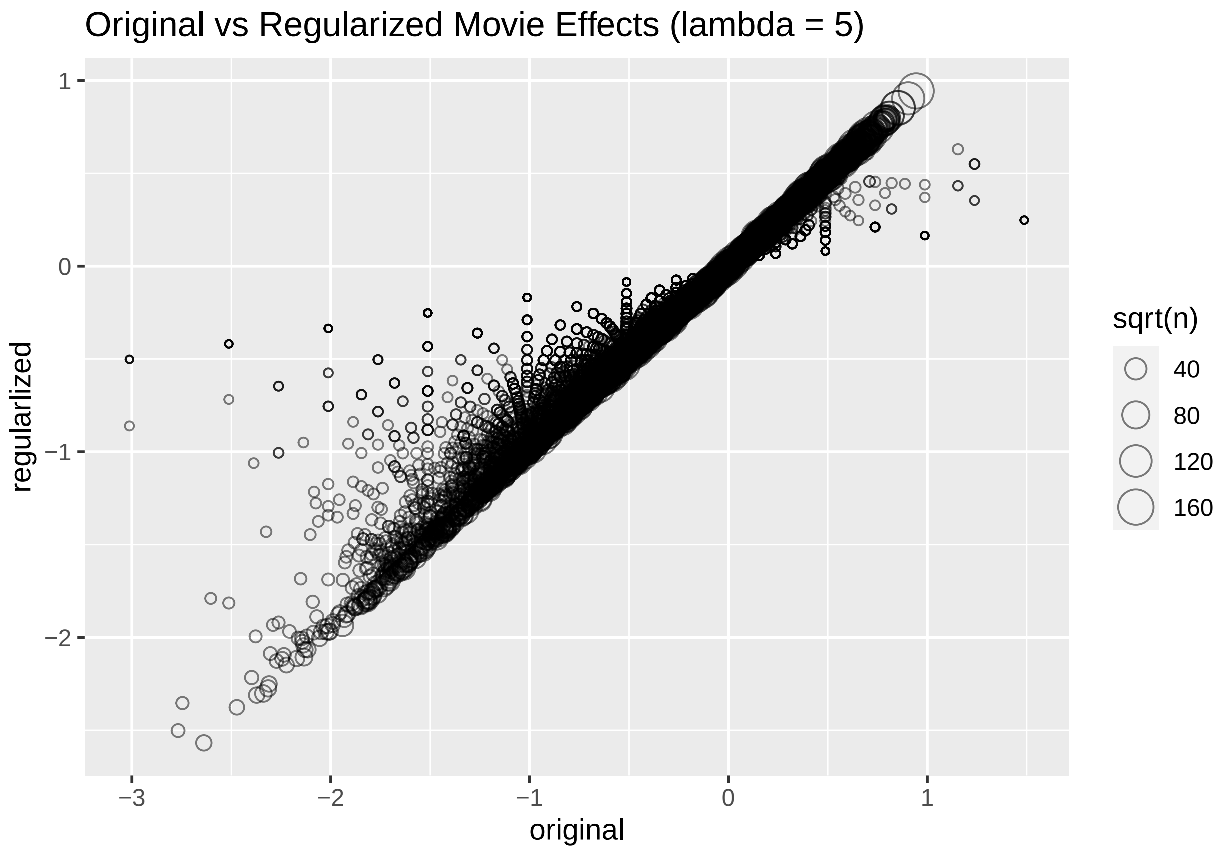
Where, $\hat{Y}_{i,u}$ is the predicted rating for movie i by user u, μ is the average rating for all movies, b_i is the movie effect, and b_u is the user effect. $\epsilon_{i,u}$ is a random error term.

2.3.4 Regularized Movie & User Effect Model

The basic concept of regularization is that for low frequency items there needs to be some penalization so they do not have an outsized effect on the predictions. The penalty factor λ (lambda) will be used to adjust the estimate for a given user's or movie's effect towards zero if the frequency is low. To illustrate this point, lets look at a visual representation of movie effects before and after regularization (lambda has been arbitrarily set to 5 for illustrative purposes):

```r
mu <- mean(train_set$rating)

movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

lambda <- 5
movie_reg_avgs <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

```
data_frame(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
     ggplot(aes(original, regularlized, size=sqrt(n))) +
     geom_point(shape=1, alpha=0.5) +
  ggtitle(
    paste("Original vs Regularized Movie Effects (lambda = ",
          lambda,
          ")",
          sep = ""))
```

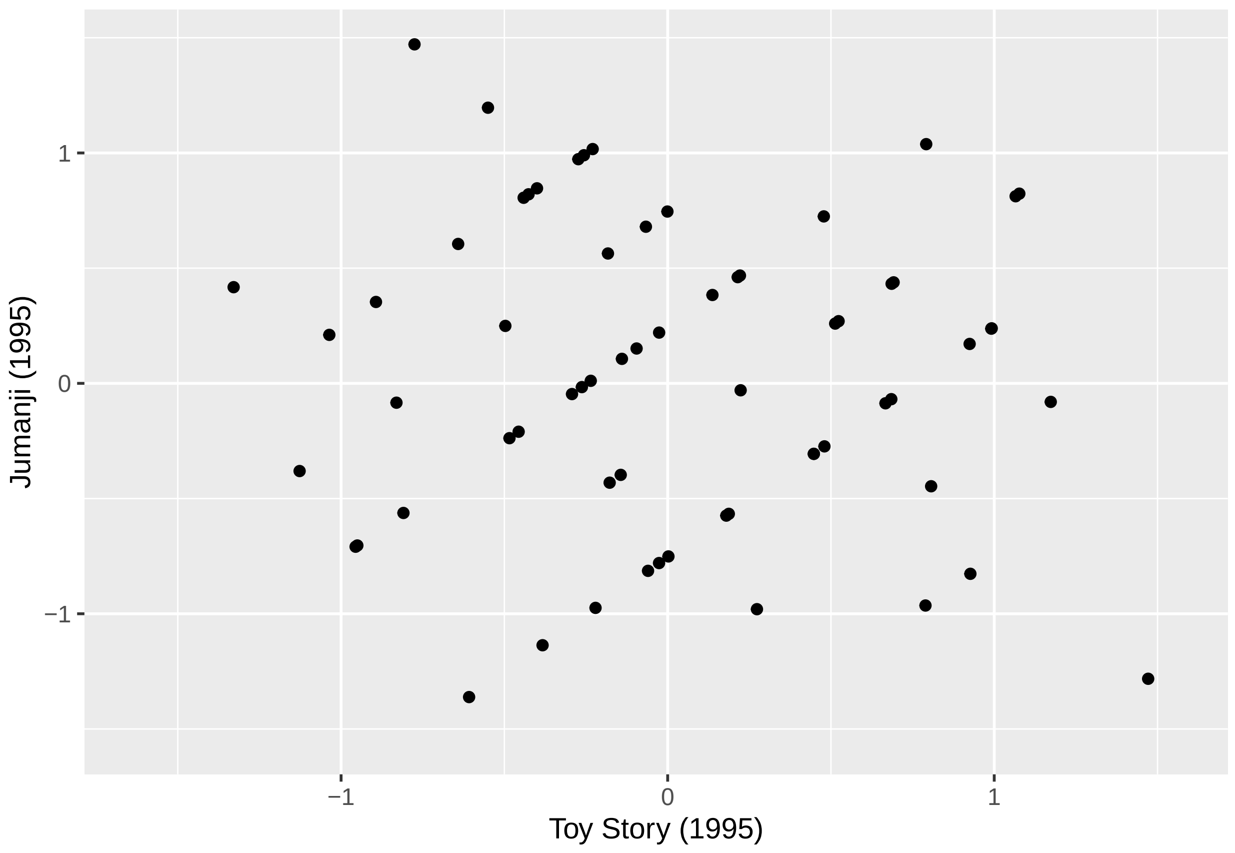


Notice how the movies with a very low number of ratings (smaller circles) have had their estimated effect shrunken towards zero. In one extreme case a movie's effect was shrunken from -3 to -0.5. In practice, λ would not be arbitrarily set, but rather cross validation would be used to select the appropriate λ. This process entails testing various values of λ on the training set and selecting the one that minimizes the RMSE.

2.3.5 Matrix Factorization

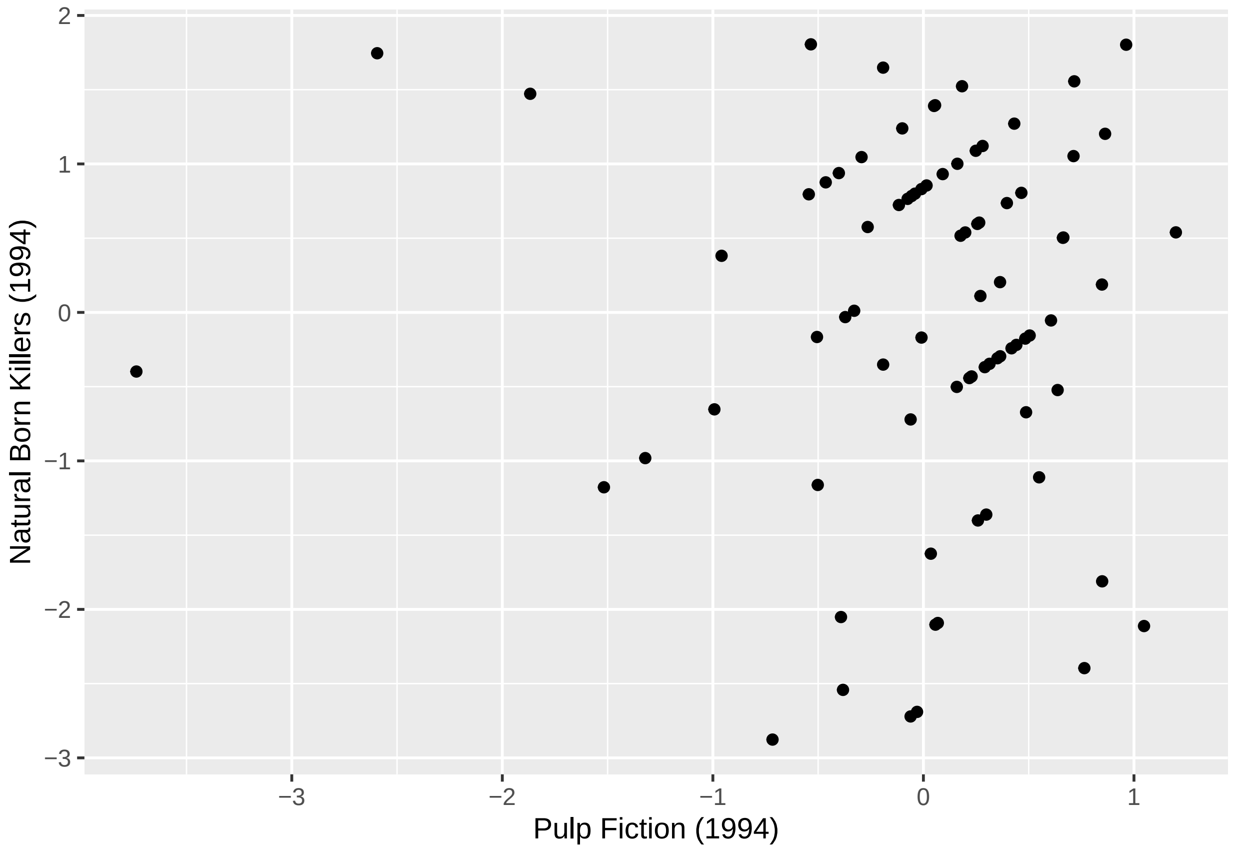
The basic premise of matrix factorization is that we use the relationships between similar users and similar movies to use for prediction. To illustrate this point, lets look at a few examples of how the user and movie effect model doesn't account for this structure in the data by examining the residuals. We will create a smaller subset of the data for illustrative purposes.

```
train_small <- edx %>%
  group_by(movieId) %>%
  filter(n() >= 1000) %>% ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 1000) %>% ungroup()

y <- train_small %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()

rm(train_small)

movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()

rownames(y)<- y[,1]
y <- y[,-1]
colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])

y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))
y <- sweep(y, 2, colMeans(y, na.rm=TRUE))

m_1 <- "Toy Story (1995)"
m_2 <- "Jumanji (1995)"
qplot(y[ ,m_1], y[,m_2], xlab = m_1, ylab = m_2)
```



From this plot we can see that there is a significant correlation between the residuals of *Toy Story* and *Jumanji*. Both of these movies would fall in to the category of “kid movies”. “Kid movies” would be considered a factor for the purposes of matrix factorization. We find this pattern between other groups of movies as well:

```r
m_3 <- "Pulp Fiction (1994)"
m_4 <- "Natural Born Killers (1994)"
qplot(y[ ,m_3], y[,m_4], xlab = m_3, ylab = m_4)
```



```r
rm(y)
```

As we can see there is a high degree of correlation of the residuals between the cult classics *Pulp Fiction* and *Natural Born Killers*. “Cult classics” would be considered a factor for the purposes of matrix factorization. Similarly, you would also find “factors” between groups of users. These correlations in residuals just cement the idea that there is structure in the data that our linear user and movie effect model isn't taking in to account.

Matrix factorization uses the data to approximate the two smaller factor matrices and uses those two matrices to construct the original rating matrix.

Chapter 3

Results

3.1 Naive Model

```r
#############
# Naive Model
#############

mu <- mean(train_set$rating)

naive_rmse <- RMSE(test_set$rating, mu)

rmse_results <- data_frame(method = "Just the average",
                           RMSE = naive_rmse)

rmse <- round(naive_rmse, 4)

rmse_results %>% knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

method	RMSE
Just the average	1.060054

After running the model and assessing the accuracy, on average the model was off in it's predictions by approximately `1.0601` stars. This is not very good. Nevertheless, this model will be our baseline for assessing subsequent models. Let's see how we can improve that accuracy!

3.2 Movie Effect Model

```
#####################
# Movie Effect Model
####################

# Calculate average rating
mu <- mean(train_set$rating)

# Calculate the movie effect by removing the mean
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Prediction based on mean plus movie effect
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Movie Effect Model",
                                     RMSE = model_1_rmse ))

rmse <- round(model_1_rmse, 4)

imp <- round(naive_rmse - model_1_rmse, 4)

imp_perc <- paste(
  round(imp / naive_rmse, 2)*100,
  "%",
  sep = "")

rmse_results %>% knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615

This model produced an RMSE of approximately `0.943`. By adding in the movie effect, the model accuracy improved by about `0.1171` stars or about `11%` improvement from the initial model. There is still room for improvement though. Next we will explore how incorporating user effects change the accuracy of the model.

3.3 Movie & User Effect Model

```r
############################
# Movie & User Effect Model
###########################

# Calculate user effects by removing the mean and movie effects
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Prediction based on mean plus movie effect and user effect
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

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

rmse <- round(model_2_rmse, 4)

imp <- round(naive_rmse - model_2_rmse, 4)

imp_perc <- paste(
  round(imp / naive_rmse, 2)*100,
  "%",
  sep = "")

rmse_results %>% knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

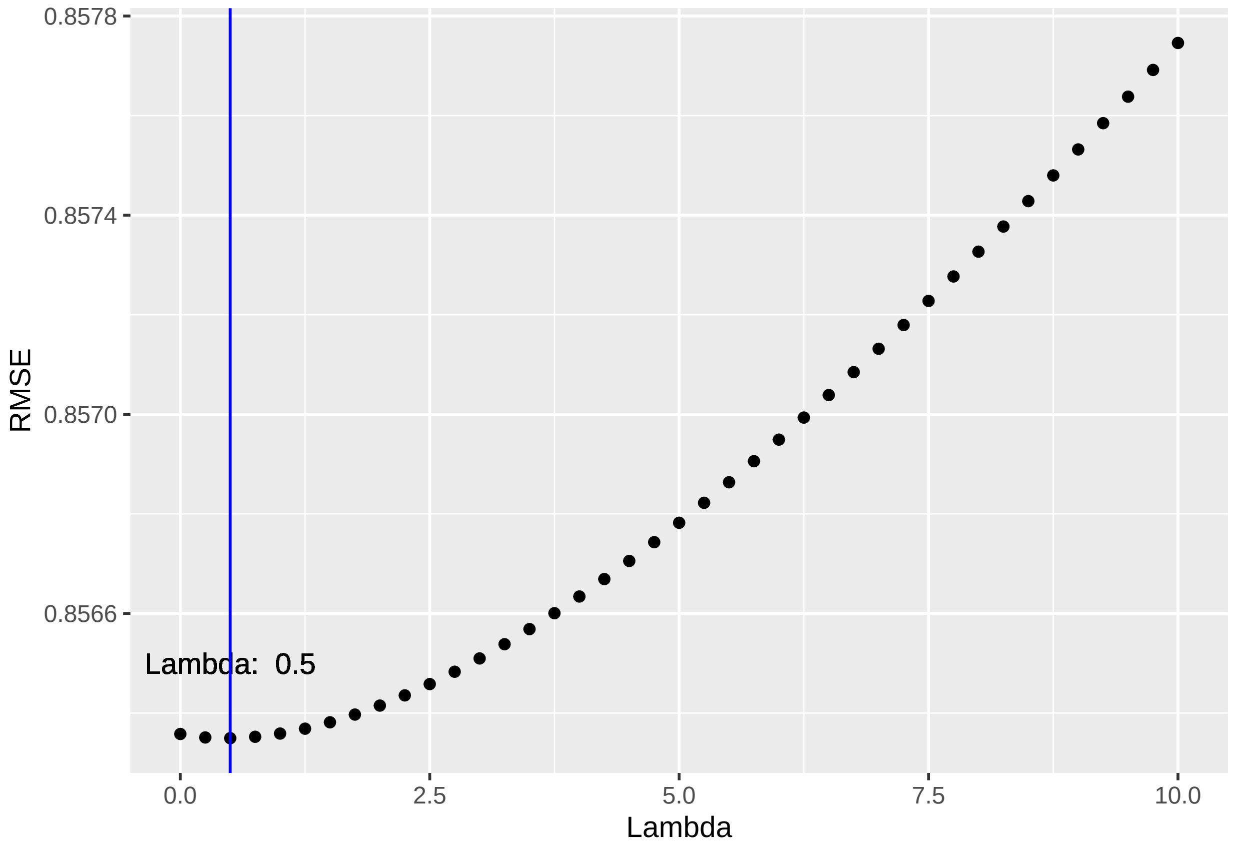
method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effect Model	0.8646843

Adding in the user effect has improved the model even further with an RMSE of `0.8647`. In fact, the model has improved by about `0.1954` stars or about `18%` from the original model. Further improvement can still be made by adjusting for the movies with a low number of ratings and users that have rated very few movies. The technique referred to as regularization will be used to account for the low frequency rated movies and users.

3.4 Regularized Movie & User Effect Model

In order to select the appropriate λ, we will use cross validation on the `train_set` and select the value for λ that minimizes the RMSE.

```
##########################################################
# Regularized Movie & User Effect Model Lambda Selection
#########################################################

# Create vector of lambdas to use for cross validation
lambdas <- seq(0, 10, 0.25)

# Since lambda is a tuning parameter, we can't use the test set to
# select it. Only the train_set can be used to select lambda.
rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  # Compute regularized movie effects
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings <-
    # This is where the example from the course had the test set called.
    # Train set was substituted to make sure we are not cheating in our
    # selection of lambda.
    train_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, train_set$rating))
})

# Plot RMSE vs Lambda
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(Lambda, RMSE)) +
  geom_point() +
  geom_text(aes(x = lambdas[which.min(rmses)],
                y = 0.8565,
                label = paste("Lambda: ",
                              lambdas[which.min(rmses)]))) +
  geom_vline(xintercept = lambdas[which.min(rmses)], color = "blue")
```



```
lambda <- lambdas[which.min(rmses)]
```

Based on the results, a λ of `0.5` minimizes the RMSE on the training data. This will be the value of λ used for regularization. This should help improve the accuracy of the movie and user effect model:

```
########################################
# Regularized Movie & User Effect Model
#######################################

lambda <- lambdas[which.min(rmses)]

mu <- mean(train_set$rating)

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

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

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

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = model_3_rmse))

rmse_results %>% knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effect Model	0.8646843
Regularized Movie + User Effect Model	0.8645518

There was a slight improvement over the original movie and user effect model by using regularization on the effect estimates. Although it may appear that the improvement was not worth the trouble in this example, when you are dealing with much larger datasets the improvements would likely be much more noticeable.

3.5 Matrix Factorization

```r
########################
# Matrix Factorization
########################

set.seed(1, sample.kind="Rounding")

# Create train and test, user by movie matrices
train_reco <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))

test_reco <- with(test_set, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))

# Create reco model object
r <- recosystem::Reco()

# Select optimal tuning parameters
options <- r$tune(train_reco, opts = list(dim = seq(10,30,10),
                                          lrate = c(0.05, 0.1, 0.2),
                                          nthread = 4,
                                          costp_l2 = c(0.01, 0.1),
                                          costq_l2 = c(0.01, 0.1),
                                          niter = 10,
                                          nfold = 10))

# Train reco model
r$train(train_reco, opts = c(options$min,
                              nthread = 4,
                              niter = 20,
                              verbose = FALSE))

# Predict test set using model
predict_reco <-  r$predict(test_reco, out_memory())

model_4_rmse <- RMSE(predict_reco, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix factorization",
                                     RMSE = model_4_rmse ))

rmse <- round(model_4_rmse, 4)

imp <- round(naive_rmse - rmse, 4)

imp_perc <- paste(
  round(imp / naive_rmse, 2)*100,
  "%",
  sep = "")

rmse_results %>% knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effect Model	0.8646843
Regularized Movie + User Effect Model	0.8645518
Matrix factorization	0.7888818

By utilizing matrix factorization, the model produced has the highest accuracy thus far with an RMSE of `0.7889`. This is an improvement of about `0.2712` stars or about `26%` from the original model. This shows that matrix factorization is able to account for more of the structure in the data than the other models and thus produces more accurate predictions.

3.6 Evaluation of final model

Since matrix factorization had the lowest RMSE it will be chosen as the final model to use on the validation set. The model will be retrained on the entire `edx` set and then used to predict the `validation` set.

```
##############################
# Final Matrix Factorization
##############################

set.seed(1, sample.kind="Rounding")

# Create edx and validation, user by movie matrices
edx_reco <- with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))

valid_reco <- with(validation, data_memory(user_index = userId,
                                            item_index = movieId,
                                            rating = rating))

# Create reco model object
r_final <- recosystem::Reco()

# Select optimal tuning parameters
options_final <- r_final$tune(edx_reco, opts = list(dim = seq(10,40,10),
                                            lrate = c(0.05, 0.1, 0.2),
                                            nthread = 4,
                                            costp_l2 = c(0.01, 0.1),
                                            costq_l2 = c(0.01, 0.1),
                                            niter = 10,
                                            nfold = 10))

# Train final reco model
r_final$train(edx_reco, opts = c(options_final$min,
                           nthread = 4,
                           niter = 20,
                           verbose = FALSE))

# Predict validation set using model
predict_final_reco <-  r_final$predict(valid_reco, out_memory())

final_rmse <- RMSE(predict_final_reco, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix factorization
                                     on Validation Set",
                                     RMSE = final_rmse ))

rmse <- round(rmse_results$RMSE[6], 4)
```

```r
rmse_results %>% knitr::kable() %>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1:2, color = "black") %>%
  row_spec(0, color = "white", background = "#5a5cd6")
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effect Model	0.8646843
Regularized Movie + User Effect Model	0.8645518
Matrix factorization	0.7888818
Matrix factorization on Validation Set	0.7819374

The final RMSE on the validation set is `0.7819`.

Chapter 4

Conclusion

In this project we explored how recommendation systems were built. We started with very simple models such as predicting every rating to be the mean to more complex machine learning techniques such as matrix factorization. Through this process, it was made clear that matrix factorization outperformed all other methods discussed with a final RMSE on the validation set of `0.7819`. Some limitations worth noting is the amount of compute power require to perform the matrix factorization. It took quite a few hours to tune and train the matrix factorization model on my laptop, and that is not even using the full MovieLens dataset that GroupLens has available.

For future work, it would be worthwhile to explore techniques that factor in selection bias, i.e. why do people rate what they rate. Since users are freely able to rate what they want, they may just not rate something altogether if they didn't like it much. This can present selection bias in the data. Also, it would be interesting to extend the analysis out further to understand what makes a "good" movie good and what makes a "bad" movie bad. This type of modeling is known as content based, i.e. who directed it, who starred in it, etc... Overall, this was a great learning experience and I feel much more comfortable in my understanding of recommendation systems.