HarvardX Capstone - MovieLens Project

Stephen Clarke

18/11/2020

Executive Summary

The purpose of this report was to create a model for recommending movies using the large data set provided. The data was split into testing and training sets to perform the analyses. Through data exploration and visualisation techniques, it was found that the genre of films and year of their release do seem to have an impact on their ratings, but these are fairly nominal relative to effect of the users (film raters), and the movies themselves. The bias from the users and movies was accounted for and used to improve the RMSE of the models to 0.8653. This was then further improved using regularisation, to achieve an RMSE of 0.8648.

Method and Analysis

I have broken down our 'Method and Analysis' section into four sub-sections; Data Cleaning, Data Exploration, Insights Gained, and Modelling Approach.

Data Cleaning

To start, we load the required libraries and import the data set, before breaking it into test and train sets, called 'edx' and 'validation'.

```
#Load all potential libraries that may be required
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(dslabs)
data(movielens)
library(data.table)
library(dplyr)
library(ggplot2)
library(ggthemes)
library(ggrepel)
ds_theme_set()
library(Lahman)
library(lattice)
library(e1071)
library(caret)
library(knitr)
library(tidyr)
library(stringr)
memory.limit(56000)
```

```
## [1] 56000
```

```
#Create edx set and validation set
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

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


movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

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

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

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

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

We then take a look at the data to see what we are working with. Using the ‘head’ and ‘summary’ functions, we can get a good overview of the data set.

```
kable(head(edx))
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller

userId	movieId	rating	timestamp	title	genres
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

```
kable(summary(edx))
```

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000055	Length:9000055
1st Qu.:18124	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
Median :35738	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35870	Mean : 4122	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

We find that the data set has 6 different headings; userId, movieId, rating, timestamp, title and genre. Additionally, we find that movies in edx data set have a mean rating of 3.512 out of 5, and the most common rating that users give a film is 4.

We count how many individual users there are in the data set, and how many movies.

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

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

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

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

There are 69,878 unique individuals who have rated films. 10,677 different movies have been rated.

We then use the ‘any’ and ‘is.na’ functions to check for any missing data from the rows and columns in the edx data set.

```
any(is.na(edx))
```

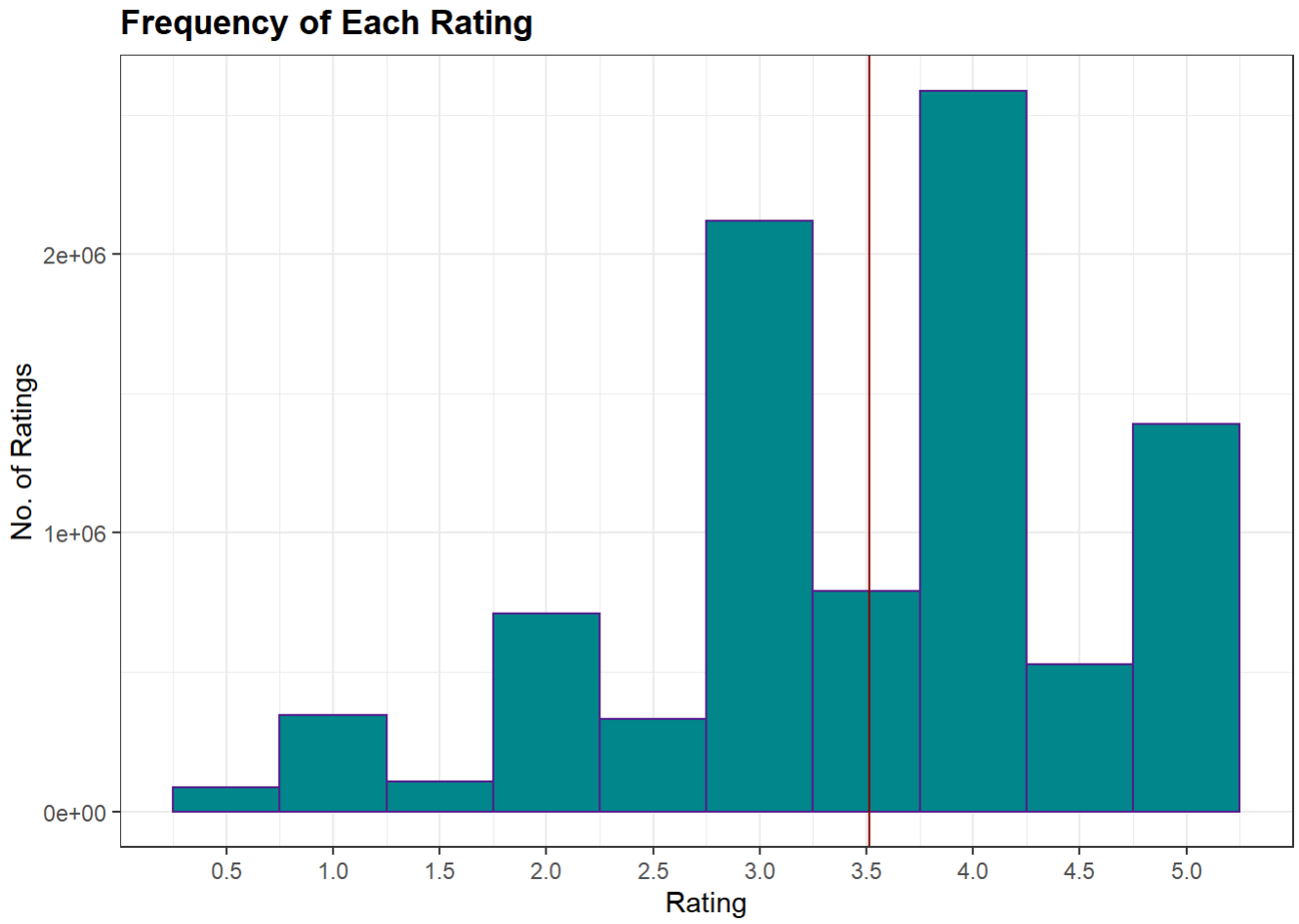
```
## [1] FALSE
```

The result comes back as ‘FALSE’, meaning that there are no blank spaces and the edx data set should be fully workable.

Data Exploration

The below graph gives a visual representation of what the summary tells us about the edx data set, in a more aesthetically pleasing format. It also shows a more comprehensive look at the spread of ratings given.

```
Avg_Rating <- mean(edx$rating)
edx %>% ggplot(aes(rating)) +
geom_histogram(col='purple4',bins=10,fill='turquoise4') +
  scale_x_continuous(breaks = seq(0.5,5,0.5)) +
  geom_vline(xintercept=Avg_Rating, col='red4',linetype='solid') +
xlab("Rating") +
  ylab("No. of Ratings") +
  ggtitle("Frequency of Each Rating")
```



What this graph also tells us, is that the standard of films that tend to get watched and rated, tend to be the better films. Furthermore, users tend to provide more positive ratings than negative ones.

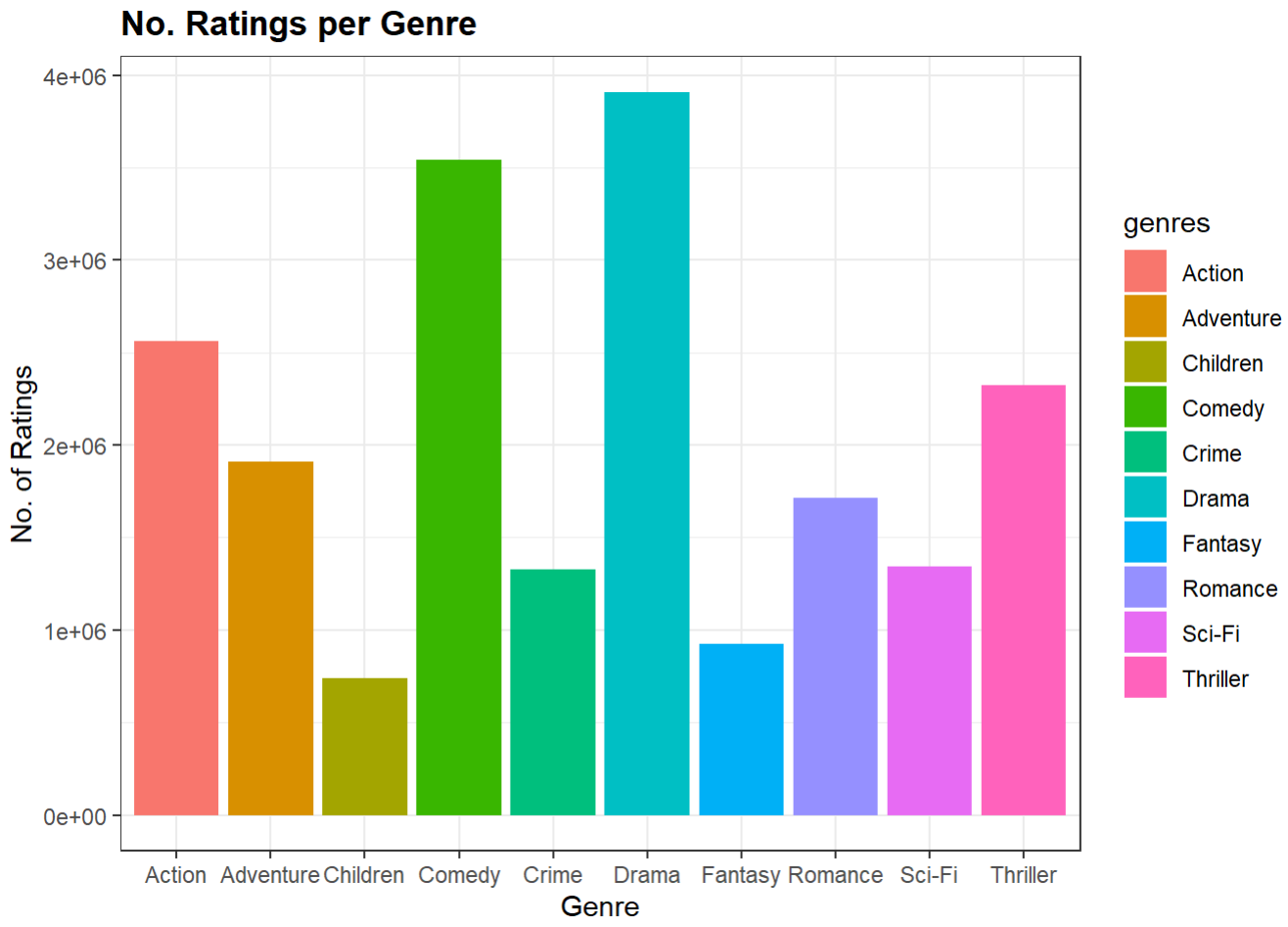
To get more information on the edx data set, we must look at the other factors that could contribute to the ratings of movies, starting with the genres.

```
Genre_by_num_ratings <- edx %>% separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarise(ratings_per_genre = n()) %>%
    arrange(desc(ratings_per_genre))

head(Genre_by_num_ratings, n=10)
```

genres <chr>	ratings_per_genre <int>
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994

1-10 of 10 rows

```
edx %>% separate_rows(genres, sep = "\\|") %>%
     group_by(genres) %>%
     summarise(ratings_per_genre = n()) %>%
     arrange(desc(ratings_per_genre)) %>%
     head(n=10) %>%
     ggplot(aes(genres, ratings_per_genre)) + geom_col(aes(fill = genres)) +
  xlab("Genre") +
  ylab("No. of Ratings")+
  ggtitle("No. Ratings per Genre")
```



It is clear that different genres of films have different popularity; e.g. Dramas get watched and rated significantly more than Fantasy Films. However, we also want to know how well they have scored.

```
Genre_by_avg_rating <-edx %>% separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarise(rating_of_genre = mean(rating)) %>%
    arrange(desc(rating_of_genre))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

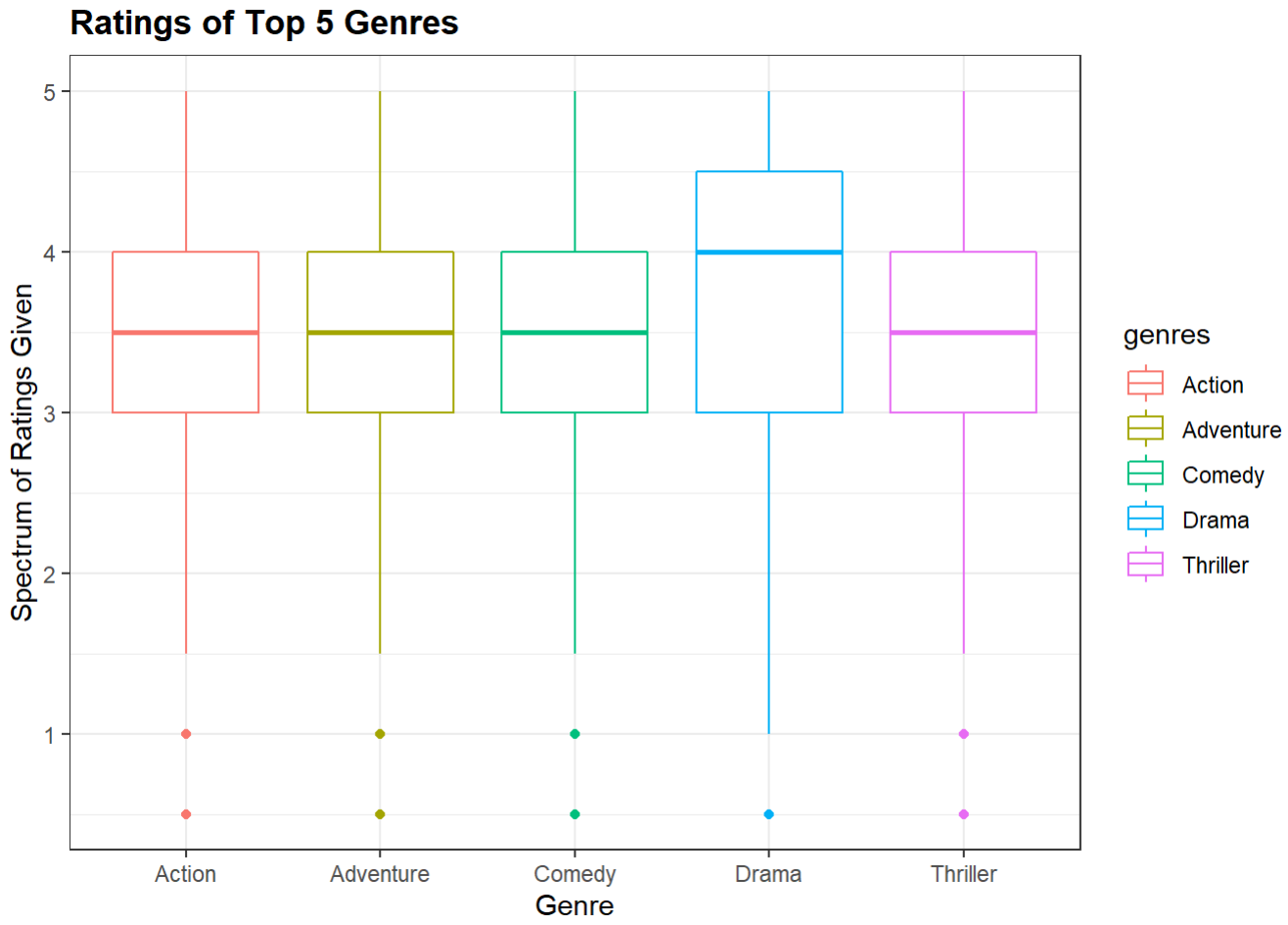
```
head(Genre_by_avg_rating, n=10)
```

genres <chr>	rating_of_genre <dbl>
Film-Noir	4.011625
Documentary	3.783487
War	3.780813
IMAX	3.767693
Mystery	3.677001
Drama	3.673131

genres <chr>	rating_of_genre <dbl>
Crime	3.665925
(no genres listed)	3.642857
Animation	3.600644
Musical	3.563305

1-10 of 10 rows

```
edx %>%
     separate_rows(genres, sep = "\\|") %>%
     select(genres, rating) %>%
     group_by(genres) %>%
     filter(genres %in% c("Drama","Comedy","Action","Thriller","Adventure")) %>%
     ggplot(aes(x = genres, y = rating, col = genres)) +
     geom_boxplot()+
  xlab("Genre") +
  ylab("Spectrum of Ratings Given")+
  ggtitle("Ratings of Top 5 Genres")
```



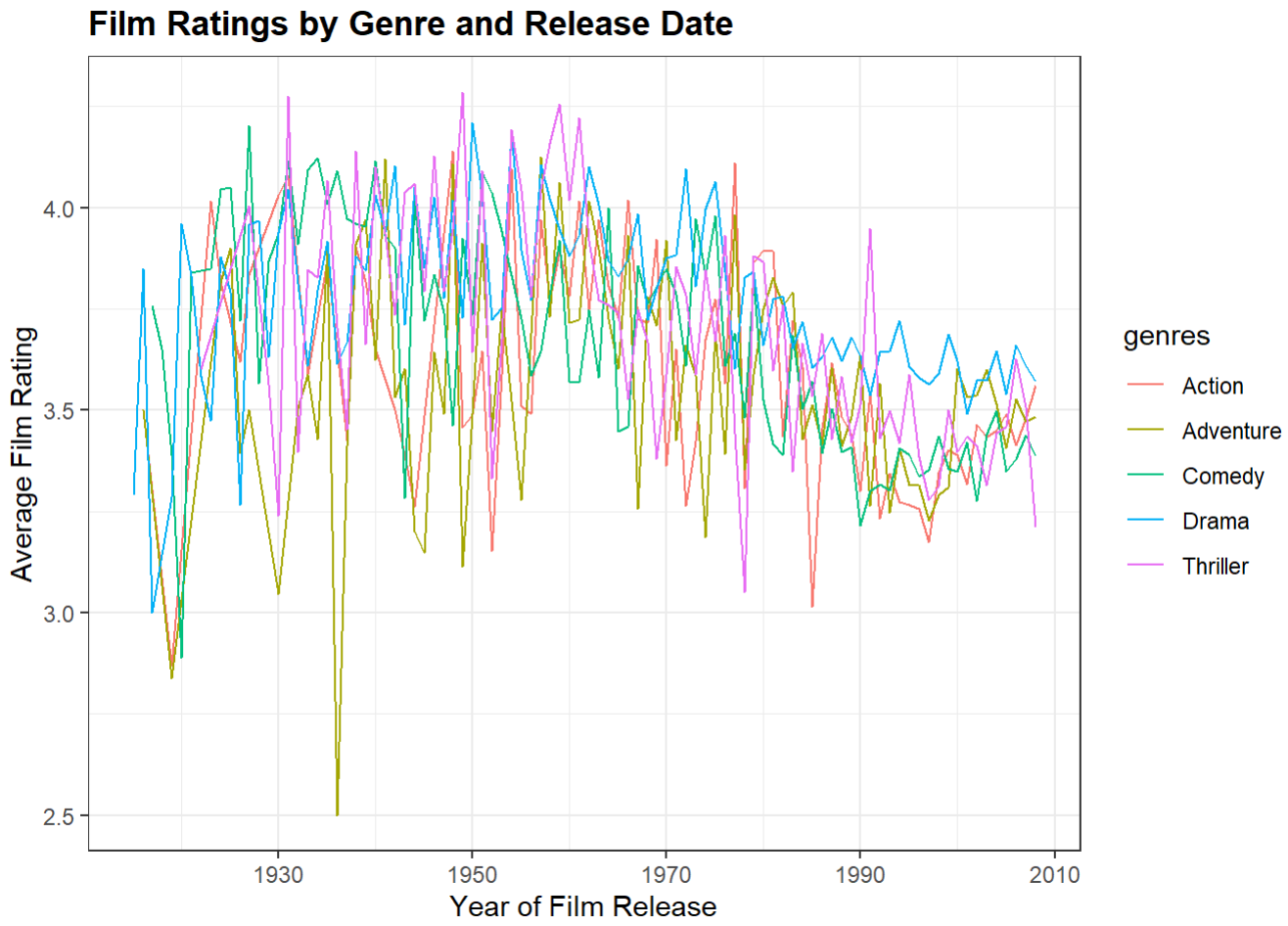
From looking at the top 5 genres (based on how many times they have been rated), it is clear that Dramas tend to score the highest on average, yet also have the lowest scores given. The other top film genres have very similar medians and interquartile ranges.

We convert our timestamp into an actual date to make our data easier to read. We also can separate our movie titles from their release dates to make our dataset easier to work with.

```
edx <- edx %>% mutate(Date_of_Rating = as.Date(as.POSIXct(timestamp, origin="1970-01-01")))

edx <- edx %>% mutate(title = str_trim(title)) %>%
     extract(title, c('title', 'Year_Released'), regex = '(.*)\\s\\((\\d+)\\)', convert = TRUE)
```

Having added a Release Year and a Date of Rating to our dataset, we can now get more insights into viewing and rating habits.

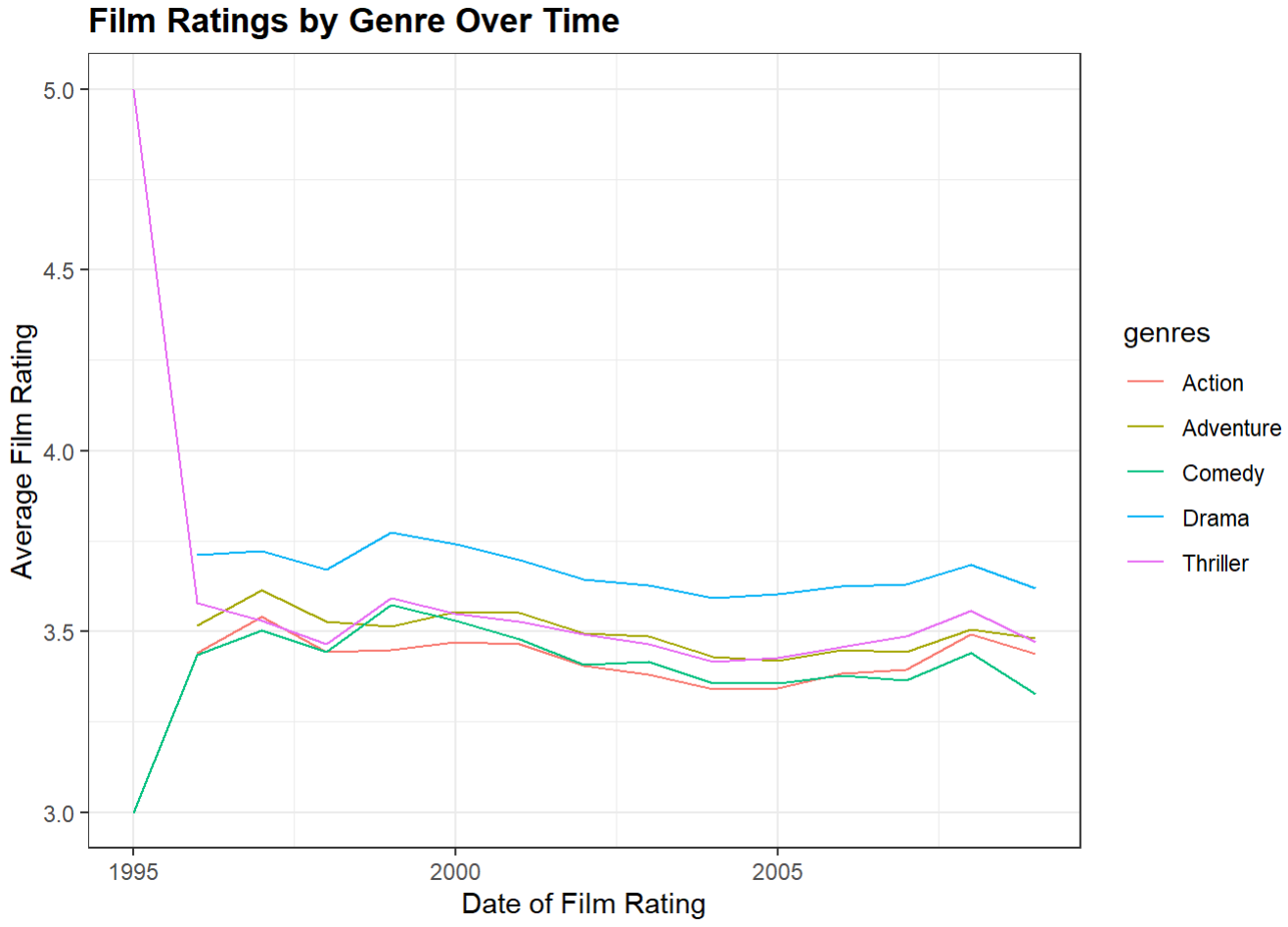
```
edx %>%
     separate_rows(genres, sep = "\\|") %>%
     select(genres, Year_Released, rating) %>%
  group_by(genres, Year_Released) %>%
     filter(genres %in% c("Drama","Comedy","Action","Thriller","Adventure")) %>%
  summarise(rating_of_genre_by_Year = mean(rating)) %>%
  ggplot(aes(x = Year_Released, y = rating_of_genre_by_Year, col = genres)) +
     geom_line() +
  xlab("Year of Film Release") +
  ylab("Average Film Rating") +
  ggtitle("Film Ratings by Genre and Release Date")
```



The above graph shows how some years have had more highly-rated films than others. Separating this into our top 5 genres, gives a good idea of when these genres were at their best. There is a lot of fluctuation year by year for each genre, but it seems that in the last 20-30 years, ratings have been more harsh, or films have been a lower standard on average.

Below, shows how the ratings provided by viewers have varied since they started to be recorded in 1995.

```
edx <- edx %>% mutate(Year_of_Rating = as.numeric(format(Date_of_Rating,'%Y')))

edx %>%
     separate_rows(genres, sep = "\\|") %>%
     select(genres, Year_of_Rating, rating) %>%
     group_by(genres, Year_of_Rating) %>%
     filter(genres %in% c("Drama","Comedy","Action","Thriller","Adventure")) %>%
  summarise(rating_of_genre_by_Year_Rated = mean(rating)) %>%
    ggplot(aes(x = Year_of_Rating, y = rating_of_genre_by_Year_Rated, col = genres)) +
     geom_line() +
  xlab("Date of Film Rating") +
  ylab("Average Film Rating") +
  ggtitle("Film Ratings by Genre Over Time")
```



From the above graph it does seem that the users have become slightly more harsh with their film rating over time, but from the graph, this does not look particularly significant.

Out of interest, I followed this up by looking at which of the most viewed films has scored the most highly among the raters.

```
top10_movies <- edx %>% select(title, rating) %>% group_by(title) %>% summarise(ratings_per_film = n(), rating_of_film = mean(rating)) %>% filter(ratings_per_film > 20000) %>% arrange(desc(rating_of_film)) %>% head(10)

kable(top10_movies)
```

title	ratings_per_film	rating_of_film
Shawshank Redemption, The	28015	4.455131
Usual Suspects, The	21648	4.365854
Schindler's List	23193	4.363493
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	25672	4.221311
Silence of the Lambs, The	30382	4.204101
Matrix, The	20908	4.202578
Star Wars: Episode V - The Empire Strikes Back	20729	4.192918
Pulp Fiction	31362	4.154789
Fargo	21395	4.134821
Braveheart	26212	4.081852

Of all films that have been rated more than 20,000 times, 'Shawshank Redemption' is the best rated film in the data set.

Insights Gained

Through the Data Exploration section, we have looked at numerous factors that seem to change over time, across genres and across films. Genres and the date that films are released, do appear to have an impact from the above graphs. The movie itself, and the user who rates the movies, will be the first things we look at when trying to get a predictive RMSE. For example, the Shawshank Redemption will generally get a higher score than the average movie. This means that the films will have their own 'bias'. As well as this, users will also have a bias based on their rating history and preferences. It is clear that across genres, some films get watched a lot more than other, and this is also true of films. Regularisation is required to take into account the fact that Shawshank Redemption has been rated over 28,000 times, while some films may only have been rated once.

Modelling Approach

Below is the standard formula for calculating root mean squared error (RMSE), which assesses the effectiveness of a predictive model.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

We ideally want our RMSE to be below 0.87 to indicate that our model is truly predictive.

Naive Baseline RMSE

We start our approach to working out the RMSE using the Naive Baseline Model.

The formula for this is:

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

Where $\hat{\mu}$ is the mean and $\varepsilon_{i,u}$ is the independent errors centered at 0.

The rating for all films in the data set is as below:

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

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

```
naive_rmse <- RMSE(validation$rating, mu)

naive_rmse
```

```
## [1] 1.061202
```

```
model_results <- data.frame(model = "Naive Baseline Model", RMSE = naive_rmse)

model_results
```

model <fctr>	RMSE <dbl>
Naive Baseline Model	1.061202

1 row

The RMSE of the Naive Baseline Model is 1.06, which is far too high to be an acceptable RMSE value.

Using Movie Rating Bias to Improve RMSE

Earlier the data was partioned into test and train sets to allow for more in depth modelling. These are called validation and edx respectively.

In the data exploration section, it was clear that some movies are generally better than others and therefore, rated more highly.

We add the term b_i to compensate for this, where b_i is the average rating for a film in this data set - or the bias. The new formula will be:

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

To get b_i we use the least squares estimate.

```
movie_avg <- edx %>%
   group_by(movieId) %>%
   summarise(b_i = mean(rating - mu))
```

```
movie_avg
```

movieId <dbl>	b_i <dbl>
1	0.4151724613
2	-0.3070658139
3	-0.3654817070
4	-0.6481658991
5	-0.4437933266
6	0.3028191010
7	-0.1538758642
8	-0.3778732406
9	-0.5146601094
10	-0.0866404831

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

We then use our RMSE formula to see how effective this method is for prediction.

```
predicted_ratings <- mu + validation %>%
  left_join(movie_avg, by = 'movieId') %>%
  .$b_i

#.$ means pull() function

movie_rating_bias <- RMSE(validation$rating, predicted_ratings)
```

```
model_results <- bind_rows(model_results, data_frame(model = "Movie Rating Bias Model", RMSE = m
ovie_rating_bias))

model_results
```

model <chr>	RMSE <dbl>
Naive Baseline Model	1.0612018
Movie Rating Bias Model	0.9439087

2 rows

By including a factor for the bias of movie ratings, we have reduced the RMSE down to 0.94. This is much better but must be improved upon further for a more robust model.

Using User Rating Bias to Improve RMSE

Another factor to consider to improve the RMSE, is the preference of users for particular movies. Therefore, we must add in another term, b_u, to compensate for this.

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

Where b_u is the user bias.

```
user_avg <- edx %>%
  left_join(movie_avg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
user_avg
```

userId <int>	b_u <dbl>
1	1.6792347055
2	-0.2364085616
3	0.2643303142
4	0.6520780624
5	0.0852677236
6	0.3462454255
7	0.0238214065
8	0.2030957376
9	0.2320727527
10	0.0833310735

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

We then use our RMSE formula to see how effective this method is for prediction.

```
predicted_ratings <- validation %>%
  left_join(movie_avg, by = 'movieId') %>%
  left_join(user_avg, by = 'userId') %>%
  mutate(b_i_b_u = mu + b_i + b_u) %>%
  .$b_i_b_u

user_bias <- RMSE(validation$rating, predicted_ratings)
```

```
model_results <- bind_rows(model_results, data_frame(model = "Movie and User Bias Model", RMSE =
user_bias))

model_results
```

model <chr>	RMSE <dbl>
Naive Baseline Model	1.0612018
Movie Rating Bias Model	0.9439087
Movie and User Bias Model	0.8653488

3 rows

By including a factor for the bias of users, we have reduced the RMSE down to 0.865. This is below our 0.87, and hence, is an effective method of prediction.

Regularisation

Our Movie Rating Model has significantly reduced RMSE but we must look at what this means for the Movie Ratings themselves.

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


validation %>% count(movieId) %>%
left_join(movie_avg) %>%
left_join(edx_movie_titles, by="movieId") %>%
arrange(desc(b_i)) %>%
select(title, b_i, n) %>%
head(n=10)
```

```
## Joining, by = "movieId"
```

title <chr>	b_i <dbl>	n <int>
Hellhounds on My Trail	1.4875348	1
More	1.2018205	1
Valerie and Her Week of Wonders (Valerie a tÃ½den divu)	0.9875348	1
Kansas City Confidential	0.9875348	1
Shawshank Redemption, The	0.9426660	3111
Red Desert, The (Deserto rosso, Il)	0.9042015	1
Godfather, The	0.9029008	2067
Man Who Planted Trees, The (Homme qui plantait des arbres, L')	0.8875348	2
Usual Suspects, The	0.8533885	2389
Schindler's List	0.8510281	2584

1-10 of 10 rows

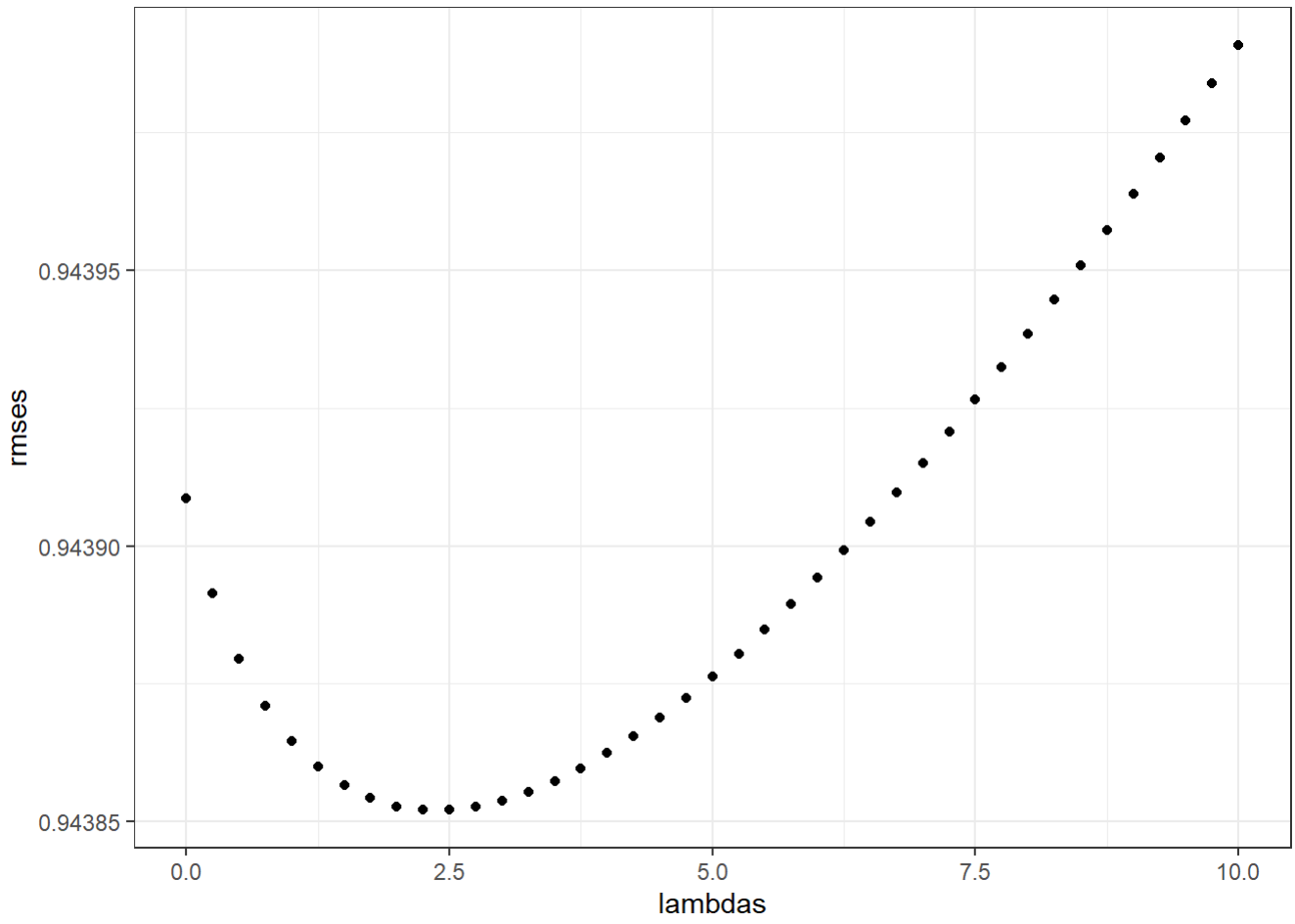
When we put the Movies in order from our Movie Rating Model, we notice that some of the top movies have only been rated once. Therefore, we must include another technique to account for this. We will use Regularisation.

Regularisation allows us to penalise films that have got high scores but low sample sizes.

We use λ as a 'tuning parameter' and use cross-validation to get its value.

```
lambdas <- seq(0, 10, 0.25)

mu <- mean(edx$rating)
just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())


rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
```

```
qplot(lambdas, rmses)
```



```
lambdas[which.min(rmses)]
```

```
## [1] 2.5
```

```
min(rmses)
```

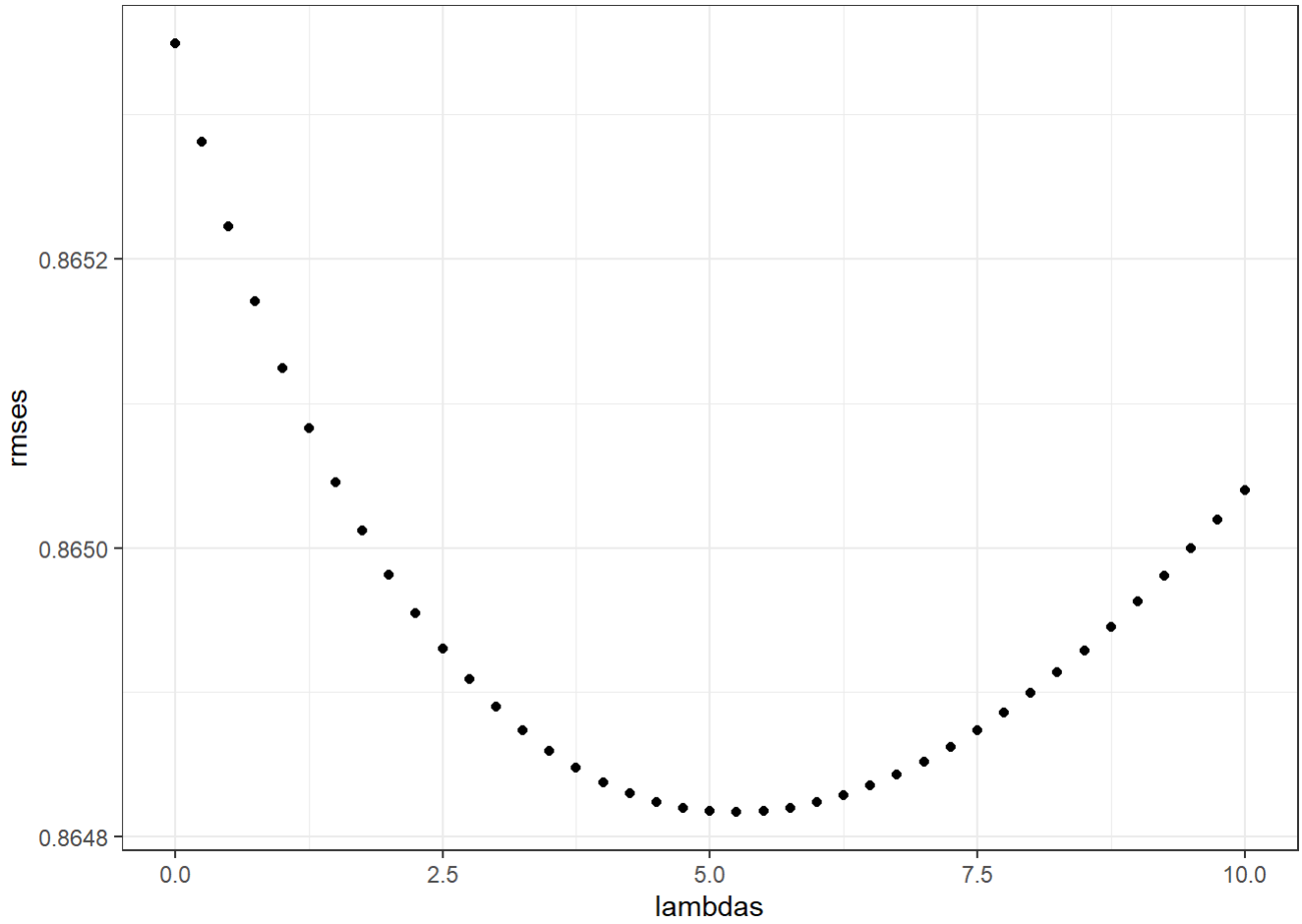
```
## [1] 0.9438521
```

The above graph shows the lambda value at which the RMSE is at its lowest, having regularised for movie bias. Then the RMSE is an improvement on when we use our movie bias model alone.

However, we want to use regularisation on the model that accounts for both movie bias and user bias.

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

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

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

  predicted_ratings <-
    validation %>%
    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, validation$rating))
})
```

```
qplot(lambdas, rmses)
```




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

```
## [1] 5.25
```

The above graph shows the lambda value at which the RMSE is at its lowest, having regularised for movie and user bias.

Results

Our table is completed below, having provided regularisation to our movie and user bias model.

```
model_results <- bind_rows(model_results, data_frame(model = "Regularised Movie and User Bias Mo
del", RMSE = min(rmses)))

model_results
```

model <chr>	RMSE <dbl>
Naive Baseline Model	1.0612018
Movie Rating Bias Model	0.9439087

model <chr>	RMSE <dbl>
Movie and User Bias Model	0.8653488
Regularised Movie and User Bias Model	0.8648170

4 rows

We see that we get an RMSE of 0.8648170 which means that our model is very predictive.

Conclusion

I was able to provide an RMSE of 0.8648, which would mean that this system would be provide a solid prediction of what each user would enjoy. To improve the RMSE number further, the genre and ‘year of release’ could be taken into account, although this would most likely only provide a slight improvement.