MovieLens Report

Ying Liu

20/11/2019

1. Introduction

This report is to discuss the creation of a movie recommendation system using the MovieLens dataset included in the dslabs package.

The goal of the project is to produce a machine learning algorithm which predicts the ratings of a separate unseen set of movies. An accuracy metric, residual mean squared error (RMSE), is used to evaulate the performance of the algorithm. The target is to produce a machine learning model which predicts the ratings within the validation set to an RMSE of below 0.865.

The MovieLens dataset used contains over 10 million movie reviews. Each review in the dataset contains:

- The ID of the user who reviews the movie
- The ID of the movie reviewed
- A rating for the movie by the user
- A timestamp for the movie
- The title of the movie
- The genre(s) of the movie

The key steps to this project were:

1. Prepare the problem via loading libraries & dataset and splitting out the validation set
2. Summarise data and key statistics via tables or graphs
3. Clean and wrangle data whilst considering feature selection
4. Evaulate a series of machine learning algorithmns on the training set and improve results (RMSE Metric); step 3 will be revisited during this step to support the process
5. Report on results; create standalone model for predictions on the validation set and conclusion

2. Method / Analysis

2.1 Prepare Problem

2.1.1 Load Data & Split out Validation Set

The following code was used to download the data and create a training set called edx and a validation set:

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

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ----------------------------------------------------------------------------- t
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------------------ tidyvers
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

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

# Download File
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Set seed for R versions above 3.5
set.seed(1, sample.kind="Rounding")

# Validation set will be 10% of MovieLens data
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

# Remove unused datasets
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

2.1.2 Load Libraries

In addition to the libraries loaded during the setup of the data; the following libraries are required to run the code in the rest of the report:

```
# All libraries required in the rest of the code is located here

if(!require(pbapply)) install.packages("pbapply", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repo = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("tidyverse", repo = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("tidyverse", repo = "http://cran.us.r-project.org")
```

- *pbapply* is a library with contains functions that allow us to track the progress of the apply family of functions
- *tidyverse* is a collection of packages with functions designed for data science
- *lubridate* is a package which contains functions to manipulate date time
- *recosystem* is a package which helps approximate an incomplete matrix using the product of two matices in a latent space allowing us to perform matrix factorisation via gradient descent

2.2 Summarise Data

2.2.1 Decriptive Statistics

Here is a first look at the loaded data:

```
# Take a preview of the dataset
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

```
glimpse(validation)
```

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 434, 8...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3.0, 3....
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 868245645, 86...
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Home Alo...
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Children|C...
```

There are over 9 million rows in the training set and just under 1 million values in our validation set which we will test our models on. Each dataframe has 6 variables.

UserID and MovieID look like unique identifiers for users and movies. Ratings is the value we are trying to predict. Timestamp is a field we need to transform to be human readable. The title field gives us both the name of the movie and the year of release. Genres is the field which holds the genre of the movie, however any given movie has a range of genres all in one field which needs to be split out.

```r
# Count of distinct users who revied movies
n_distinct(edx$userId)
```

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

```r
# Count of distinct movies
edx %>%
  filter(!is.na(movieId)) %>%
  summarise(uni = n_distinct(movieId))
```

```
##     uni
## 1 10677
```

```r
# Average movie rating
mean(edx$rating)
```

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

We have 68978 distinct users in our edx dataset who have rated 10677 different movies, the average rating across the dataset is 3.512. Movies can be split into their genres as per below

```r
# Extract genres of movies into a data frame
genresCountTable <- edx %>% tidyr::separate_rows(genres, sep = "\\|") %>%
  dplyr::group_by(genres) %>%
  dplyr::summarize(count = n(), averageRating = mean(rating)) %>%
  plyr::arrange(desc(count))

# Look at genres table
genresCountTable %>% knitr::kable()
```

genres	count	averageRating
Drama	3910127	3.673131
Comedy	3540930	3.436908
Action	2560545	3.421405
Thriller	2325899	3.507676
Adventure	1908892	3.493544
Romance	1712100	3.553813
Sci-Fi	1341183	3.395743
Crime	1327715	3.665925
Fantasy	925637	3.501946
Children	737994	3.418715
Horror	691485	3.269815

genres	count	averageRating
Mystery	568332	3.677001
War	511147	3.780813
Animation	467168	3.600644
Musical	433080	3.563305
Western	189394	3.555918
Film-Noir	118541	4.011625
Documentary	93066	3.783487
IMAX	8181	3.767693
(no genres listed)	7	3.642857

Drama, comedy, action and thrillers are the most numerous genres in the edx dataset, there are also 7 reviews without a genre. The following shows the 10 movies with the most reviews in the edx dataset:

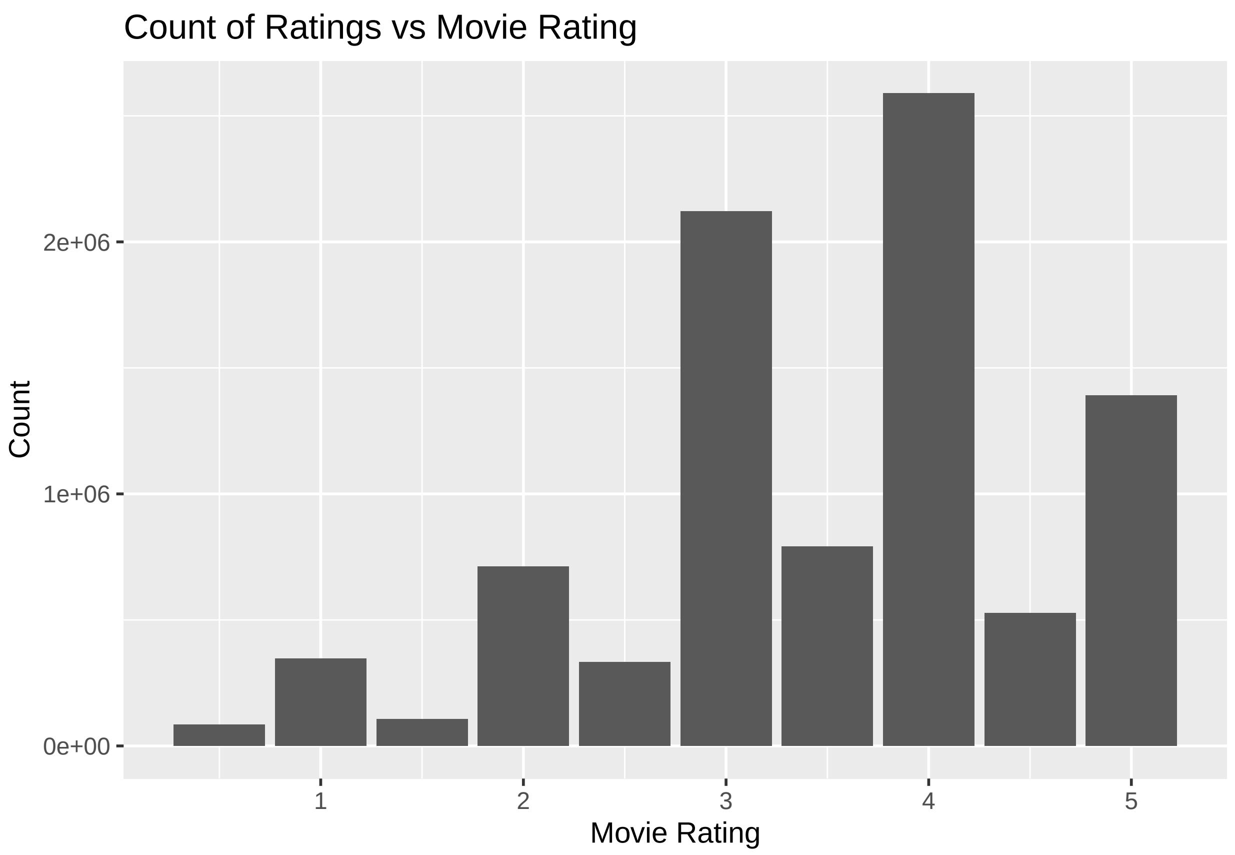
```r
# Get count of top 10 most reviewed movies, arranged in descending order
edx %>% group_by(title) %>%
  summarise(count = n(), averageRating = mean(rating)) %>%
  arrange(desc(count)) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	count	averageRating
Pulp Fiction (1994)	31362	4.154789
Forrest Gump (1994)	31079	4.012822
Silence of the Lambs, The (1991)	30382	4.204101
Jurassic Park (1993)	29360	3.663522
Shawshank Redemption, The (1994)	28015	4.455131
Braveheart (1995)	26212	4.081852
Fugitive, The (1993)	25998	4.009155
Terminator 2: Judgment Day (1991)	25984	3.927859
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672	4.221311
Apollo 13 (1995)	24284	3.885789

Movies with more reviews also have a high average rating compared with the average of ratings for the whole dataset.

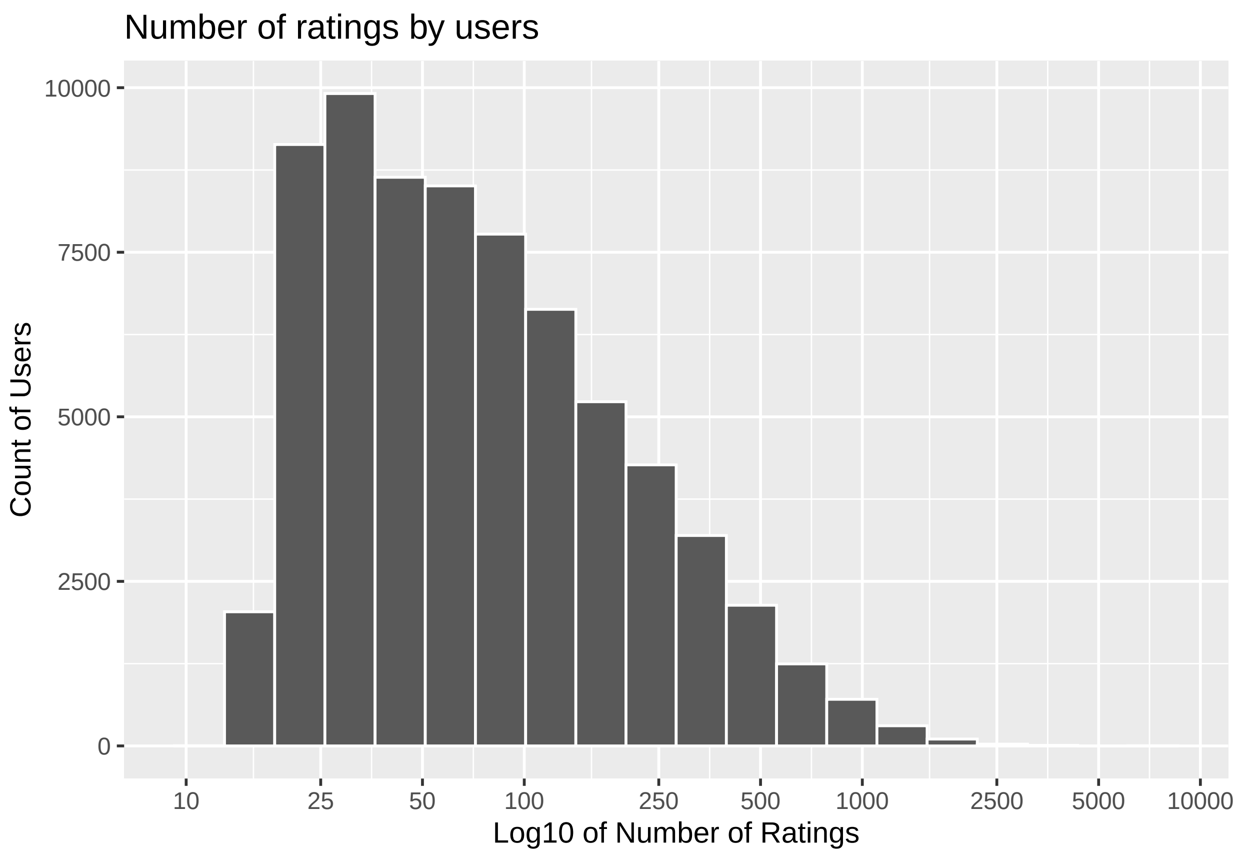
2.2.2 Data Visualisation

```r
# Plot number of times a rating was given in the edx dataset
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_col() +
  labs(title = "Count of Ratings vs Movie Rating",
       x = "Movie Rating",
       y = "Count")
```



The chart above shows that whole number ratings are significantly more popular than half step ratings.

```r
# Plot number of ratings a user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 20, colour = "white") +
  scale_x_log10(breaks = c(10,25,50,100,250,500,1000,2500,5000,10000)) +
  labs(title = "Number of ratings by users",
      x = "Log10 of Number of Ratings",
      y = "Count of Users")
```



Most user only rate up to 100 movies, the count of users who rate up to 1000 movies decreases rapidly and none get past 2500 reviews. A log scale on the x axis is used otherwise all the data would be in one bin on the histogram.

2.3 Prepare Data

2.3.1 Data Cleaning - Missing Values

```r
# Look for missing values
summary(edx)
```

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

```r
# Do the same for the validation set
summary(validation)
```

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08  
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres         
##  Length:999999      Length:999999     
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
## 
```

There does not appear to be any missing values in either dataset. However there may still be missing values in our character class variables.

2.3.2 Data Transformation

2.3.2.1 Transform Timestamp and Title

```
# Transform the timestamp value into the useable POSIXct format
edx <- edx %>%
  mutate(timestamp = as_datetime(timestamp))
summary(edx$timestamp)
```

```
##                   Min.                1st Qu.                 Median 
## "1995-01-09 11:46:49" "2000-01-01 23:11:23" "2002-10-24 21:11:58" 
##                   Mean                3rd Qu.                   Max. 
## "2002-09-21 13:45:07" "2005-09-15 02:21:21" "2009-01-05 05:02:16"
```

```
# Also transform the validation set
validation <- validation %>%
  mutate(timestamp = as_datetime(timestamp))
```

From the table above we can see that the timestamp value has now been transformed to a human readable format for time of the movie review. A potential variable to explore is to see if there is any correlation between the year of release of the movie (in the title field) and the year of the rating. The following code will extract the year of movie release:

```
# Extract year of movie from title from edx and validation sets
edx <- edx %>%
  mutate(year = as.numeric(substr(title,
                                  nchar(title) - 4,
                                  nchar(title) - 1)))

validation <- validation %>%
  mutate(year = as.numeric(substr(title,
                                  nchar(title) - 4,
                                  nchar(title) - 1)))

head(edx$year)
```

```
## [1] 1992 1995 1995 1994 1994 1994
```

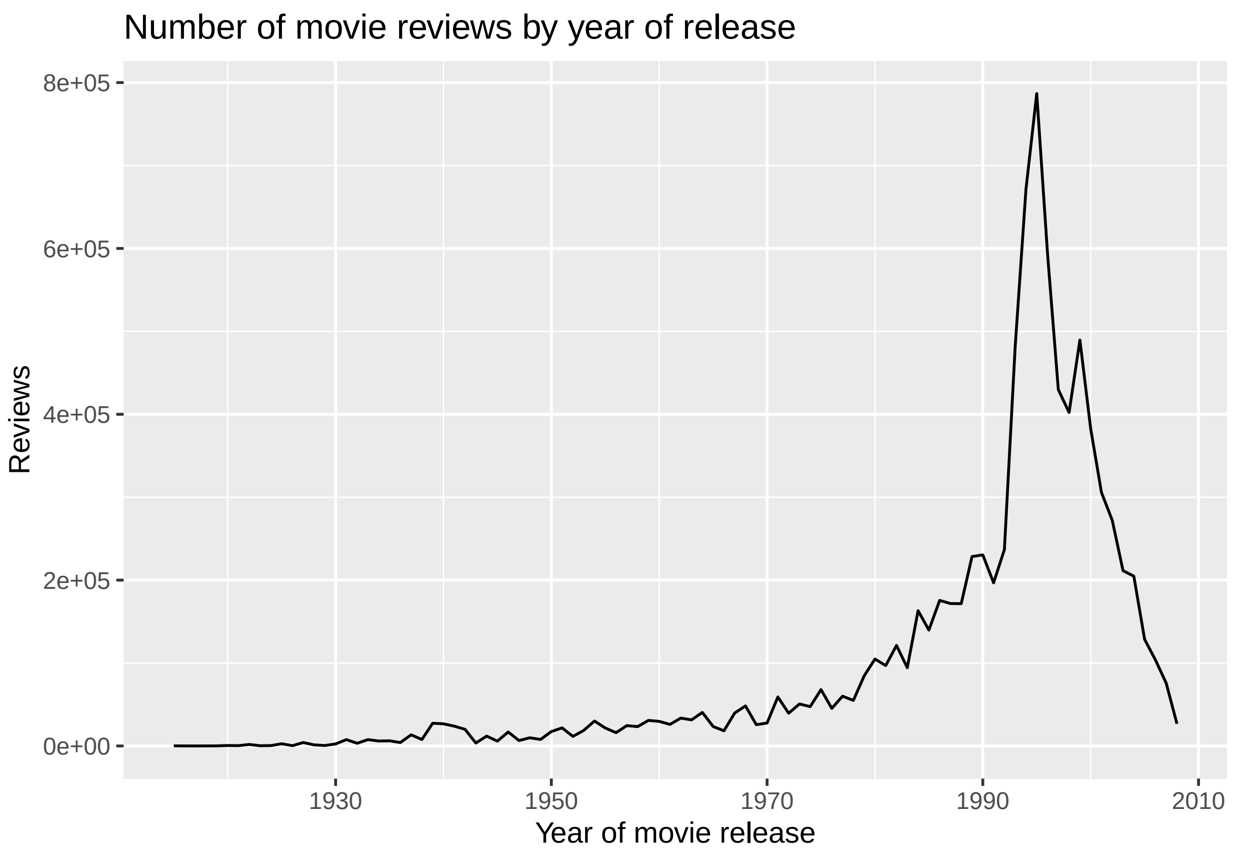
```r
# Show the first and latest 10 years of movie releases by the number of reviews
edx %>%
  select(movieId, year) %>%
  group_by(year) %>%
  summarise(countOfReviews = n()) %>%
  arrange(desc(year)) %>%
  slice(1:10) %>% knitr::kable()
```

year	countOfReviews
2008	26741
2007	75788
2006	103870
2005	128613
2004	204811
2003	211397
2002	272180
2001	305705
2000	382763
1999	489537

```r
edx %>%
  select(movieId, year) %>%
  group_by(year) %>%
  summarise(countOfReviews = n()) %>%
  arrange(year) %>%
  slice(1:10) %>% knitr::kable()
```

year	countOfReviews
1915	180
1916	84
1917	32
1918	73
1919	158
1920	575
1921	406
1922	1825
1923	316
1924	457

```r
# Graph showing movie reviews by year of movie release
edx %>%
  select(movieId, year) %>%
  group_by(year) %>%
  summarise(count = n()) %>%
  arrange(year) %>%
  ggplot(aes(x = year, y = count)) +
    geom_line() +
    ggtitle("Number of movie reviews by year of release") +
    xlab("Year of movie release") +
    ylab("Reviews")
```



The tables and the graph show we have more movies reviewed that were released in the last twenty years. This indicates that users are more likely to watch recent movies, or there are many more movies to choose from in the last 20 years for users to review. Next we explore if there are any trends between the time between rating of the movie and release.

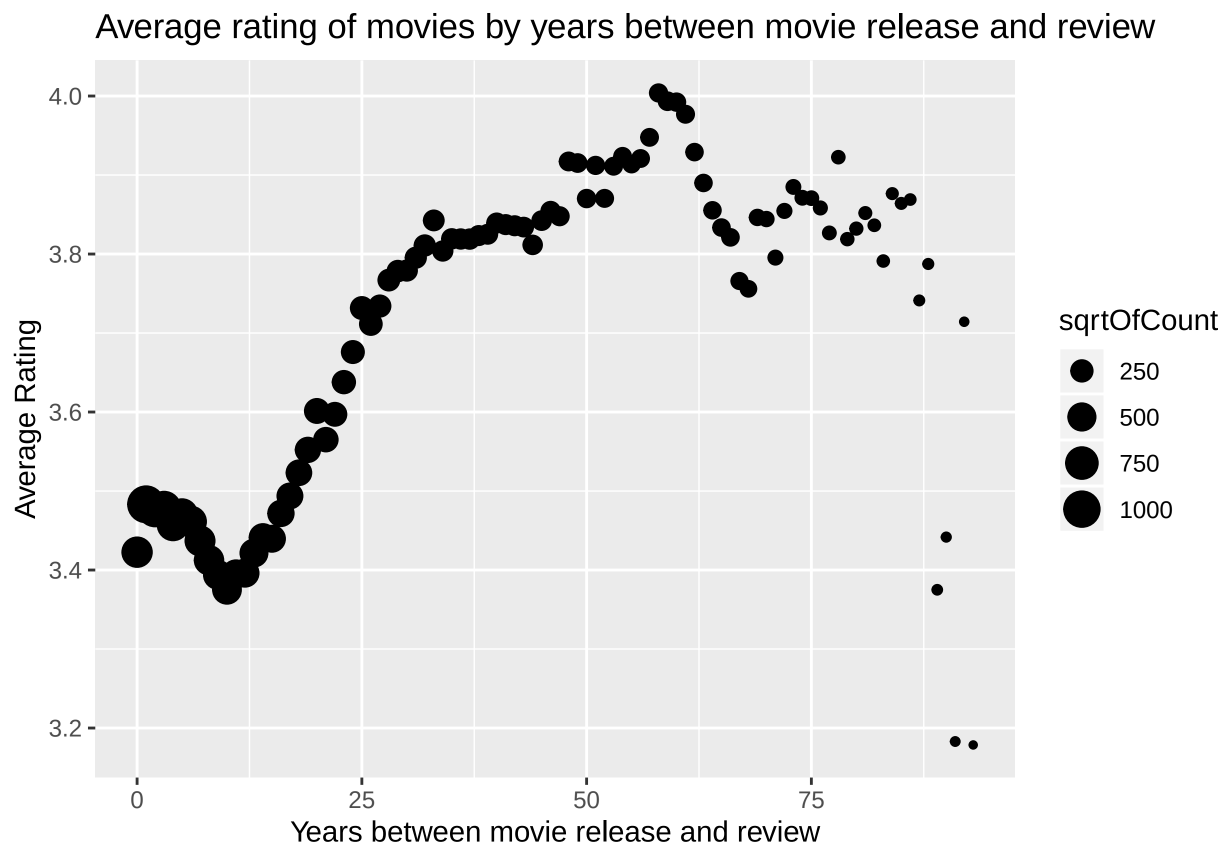
```
# Add variable for number of years between release of movie and year of review
edx <- edx %>%
  mutate(reviewLag = year(timestamp) - year)

# Look at new variable
summary(edx$reviewLag)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -2.00    2.00    7.00   11.98   16.00   93.00
```

```
# Look at negative numbers
edx %>%
  select(title, timestamp, year, reviewLag) %>%
  filter(reviewLag < 0) %>%
  arrange(reviewLag) %>%
  slice(1:10)
```

```
##                         title           timestamp year reviewLag
## 1    Talk of Angels (1998) 1996-10-10 08:57:49 1998        -2
## 2    Talk of Angels (1998) 1996-09-17 11:56:51 1998        -2
## 3    Talk of Angels (1998) 1996-09-23 21:49:48 1998        -2
## 4  Dangerous Ground (1997) 1996-10-04 21:27:42 1997        -1
## 5         Relic, The (1997) 1996-08-29 08:49:28 1997        -1
## 6  Dangerous Ground (1997) 1996-09-16 08:09:47 1997        -1
## 7      Gone Fishin' (1997) 1996-08-07 18:17:56 1997        -1
## 8        Relic, The (1997) 1996-08-23 15:04:47 1997        -1
## 9  Dangerous Ground (1997) 1996-09-05 18:13:46 1997        -1
## 10   One Man's Hero (1999) 1998-12-10 17:32:42 1999        -1
```

```
# Replace negative numbers with NAs
edx <- edx %>%
  mutate(reviewLag = ifelse(reviewLag < 0, "NA", reviewLag))

# Plot Trends
edx %>%
  group_by(reviewLag) %>%
  filter(reviewLag != "NA") %>%
  summarise(mean = mean(rating), sqrtOfCount = sqrt(n())) %>%
  ggplot(aes(x = as.numeric(reviewLag), y = mean)) +
  geom_point(aes(size = sqrtOfCount)) +
    ggtitle("Average rating of movies by years between movie release and review") +
    xlab("Years between movie release and review") +
    ylab("Average Rating")
```



```
# Convert edx set to numeric and process validation set for modelling, ignore errors for NA's introduce
edx <- edx %>%
  mutate(reviewLag = as.numeric(reviewLag))

validation <- validation %>%
  mutate(reviewLag = as.numeric(ifelse(year(timestamp) - year < 0,
                                       "NA",
                                       year(timestamp) - year)))
```

The time lag effect from release to review of a movie has a few interesting trends. Any negative values were considered erroneous and converted to NA's. The graph shows that the mean of movie reviews the falls in the first ten years post release and then steadily climbs. There are also significantly more movies reviewed close to their release dates as can be seem from the size of the square root of the count of movies legend. In conclusion the data seems to indicate that users seem to only watch older movie which are highly rated.

2.3.2.2 Transform Genres

From section 2.2.1 we know we have 20 different genres in the edx set, including a 7 reviews where the movie had no genre. Below we extract each genre a movie has creating a new binary variable for if the movie belongs to that particular genre.

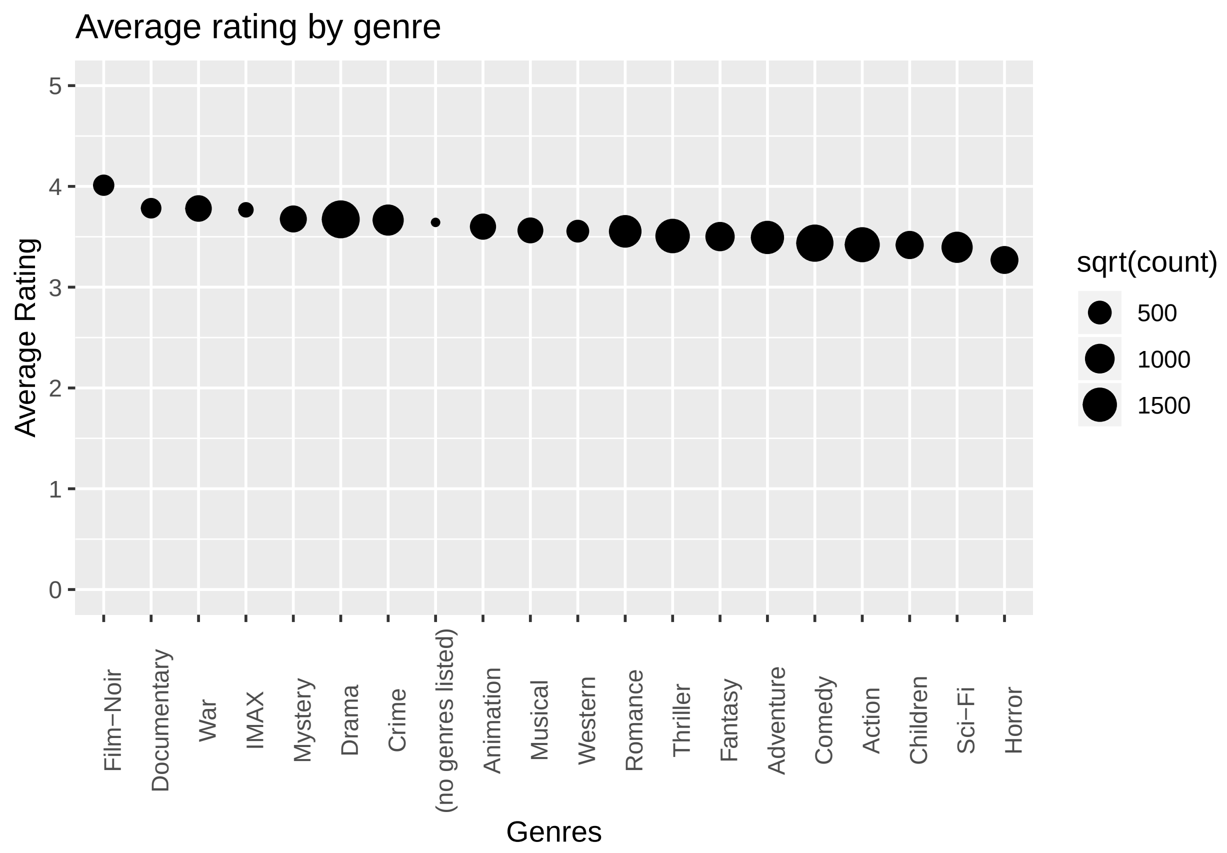
```
# Create a variable in dataframe for every genre, then match for movie
edxGenres <- edx %>%
  mutate(noGenre = ifelse(str_detect(genres, "no genres listed"), 1, 0),
         Action = ifelse(str_detect(genres, "Action"), 1, 0),
         Adventure = ifelse(str_detect(genres, "Adventure"), 1, 0),
         Animation = ifelse(str_detect(genres, "Animation"), 1, 0),
         Children = ifelse(str_detect(genres, "Children"), 1, 0),
         Comedy = ifelse(str_detect(genres, "Comedy"), 1, 0),
         Crime = ifelse(str_detect(genres, "Crime"), 1, 0),
         Documentary = ifelse(str_detect(genres, "Documentary"), 1, 0),
         Drama = ifelse(str_detect(genres, "Drama"), 1, 0),
         Fantasy = ifelse(str_detect(genres, "Fantasy"), 1, 0),
         FilmNoir = ifelse(str_detect(genres, "Film-Noir"), 1, 0),
         Horror = ifelse(str_detect(genres, "Horror"), 1, 0),
         IMAX = ifelse(str_detect(genres, "IMAX"), 1, 0),
         Musical = ifelse(str_detect(genres, "Musical"), 1, 0),
         Mystery = ifelse(str_detect(genres, "Mystery"), 1, 0),
         Romance = ifelse(str_detect(genres, "Romance"), 1, 0),
         SciFi = ifelse(str_detect(genres, "Sci-Fi"), 1, 0),
         Thriller = ifelse(str_detect(genres, "Thriller"), 1, 0),
         War = ifelse(str_detect(genres, "War"), 1, 0),
         Western = ifelse(str_detect(genres, "Western"), 1, 0))

# Get sum of all genres
colSums(edxGenres[,9:28])
```

```
##     noGenre      Action   Adventure   Animation    Children      Comedy
##           7     2560545     1908892      467168      737994     3540930
##       Crime Documentary       Drama     Fantasy    FilmNoir      Horror
##     1327715       93066     3910127      925637      118541      691485
##        IMAX     Musical     Mystery     Romance       SciFi    Thriller
##        8181      433080      568332     1712100     1341183     2325899
##         War     Western
##      511147      189394
```

```
# Check sum of all genres in new df against edx set
genresCountTable
```

```
## # A tibble: 20 x 3
##    genres                count averageRating
##    <chr>                 <int>         <dbl>
##  1 Drama               3910127          3.67
##  2 Comedy              3540930          3.44
##  3 Action              2560545          3.42
##   4 Thriller           2325899          3.51
##   5 Adventure          1908892          3.49
##   6 Romance            1712100          3.55
##   7 Sci-Fi             1341183          3.40
##   8 Crime              1327715          3.67
##   9 Fantasy             925637          3.50
##  10 Children            737994          3.42
##  11 Horror              691485          3.27
##  12 Mystery             568332          3.68
##  13 War                 511147          3.78
##  14 Animation           467168          3.60
##  15 Musical             433080          3.56
##  16 Western             189394          3.56
##  17 Film-Noir           118541          4.01
##  18 Documentary          93066          3.78
##  19 IMAX                  8181          3.77
##  20 (no genres listed)       7          3.64
```

```r
# Add binary genre variables to edx set
edx <- edxGenres

# Complete same steps for the Validation set, creating a validation set with the same predictors
validation <- validation %>% mutate(noGenre = ifelse(str_detect(genres, "no genres listed"), 1, 0),
         Action = ifelse(str_detect(genres, "Action"), 1, 0),
         Adventure = ifelse(str_detect(genres, "Adventure"), 1, 0),
         Animation = ifelse(str_detect(genres, "Animation"), 1, 0),
         Children = ifelse(str_detect(genres, "Children"), 1, 0),
         Comedy = ifelse(str_detect(genres, "Comedy"), 1, 0),
         Crime = ifelse(str_detect(genres, "Crime"), 1, 0),
         Documentary = ifelse(str_detect(genres, "Documentary"), 1, 0),
         Drama = ifelse(str_detect(genres, "Drama"), 1, 0),
         Fantasy = ifelse(str_detect(genres, "Fantasy"), 1, 0),
         FilmNoir = ifelse(str_detect(genres, "Film-Noir"), 1, 0),
         Horror = ifelse(str_detect(genres, "Horror"), 1, 0),
         IMAX = ifelse(str_detect(genres, "IMAX"), 1, 0),
         Musical = ifelse(str_detect(genres, "Musical"), 1, 0),
         Mystery = ifelse(str_detect(genres, "Mystery"), 1, 0),
         Romance = ifelse(str_detect(genres, "Romance"), 1, 0),
         SciFi = ifelse(str_detect(genres, "Sci-Fi"), 1, 0),
         Thriller = ifelse(str_detect(genres, "Thriller"), 1, 0),
         War = ifelse(str_detect(genres, "War"), 1, 0),
         Western = ifelse(str_detect(genres, "Western"), 1, 0))

# Clean up environment
rm(edxGenres)

# Average rating by genre in order
genresCountTable %>%
  ggplot(aes(x = reorder(genres, -averageRating), y = averageRating)) +
  geom_point(stat = "identity", aes(size = sqrt(count))) +
  labs(title = "Average rating by genre",
       x = "Genres",
       y = "Average Rating") +
  scale_y_continuous(limits = c(0, 5)) +
  theme(axis.text.x = element_text(angle = 90))
```



The count of the reviews are square rooted to fit into the chart; genres with less reviews have a higher average rating. There is some variablity to the average rating of different genres.

2.4 Modelling Method - Test & Evaluate Algorithms

2.4.1 RMSE Metric

To assess our model, we will check true ratings in the validation set against the predictions we will generate with our model. RMSE takes the sum of the square of the residual of our predicted ratings and the actual value.

```r
# RMSE function
RMSE <- function(rating, prediction){
  sqrt(mean((rating - prediction)^2))
}
```

Our target in this project is to have an RMSE of below 0.865.

2.4.2 Split edx set into training and testing set

We will create a test & training set from our edx set to assess the accuracy of different models we implement.

```r
# Set seed for R versions above 3.5
set.seed(17, sample.kind="Rounding")

# Create testing & training set
index <- createDataPartition(y = edx$rating,
                             times = 1,
                             p = 0.1,
                             list = F)
training <- edx[-index,]
testing <- edx[index,]

# Remove users & movies in test set that do not appear in training set
testing <- testing %>%
  semi_join(training, by = "movieId") %>%
  semi_join(training, by = "userId")
```

Movies removed during this process will be considered when a final model is selected for optimising parameters on the whole edx set so we can predict the ratings on the validation set.

2.4.3 Naive Model

Due to the size of the dataset, training models using *train* in the *caret* package is not feasible. We will use the approach outlined in Rafael Irizarry's Book Introduction to Data Science chapter 33.7 where we calculate residuals of different parameters and add them to our equation improving predictions.

The first model we will produce is a baseline for us to assess other models against. Movies are predicted to have the same rating as the average of the movies and any variation explained by noise.

```r
# MODEL 1 - Naive model; Baseline model using total average ratings to predict movie ratings
# Calculate average
average <- mean(training$rating)
average
```

```
## [1] 3.51251
```

```r
# Create table to compare RMSEs of different models and add naive mode RMSE
rmseResults <- data_frame(method = "Naive Model", RMSE = RMSE(testing$rating, average))
rmseResults %>% knitr::kable()
```

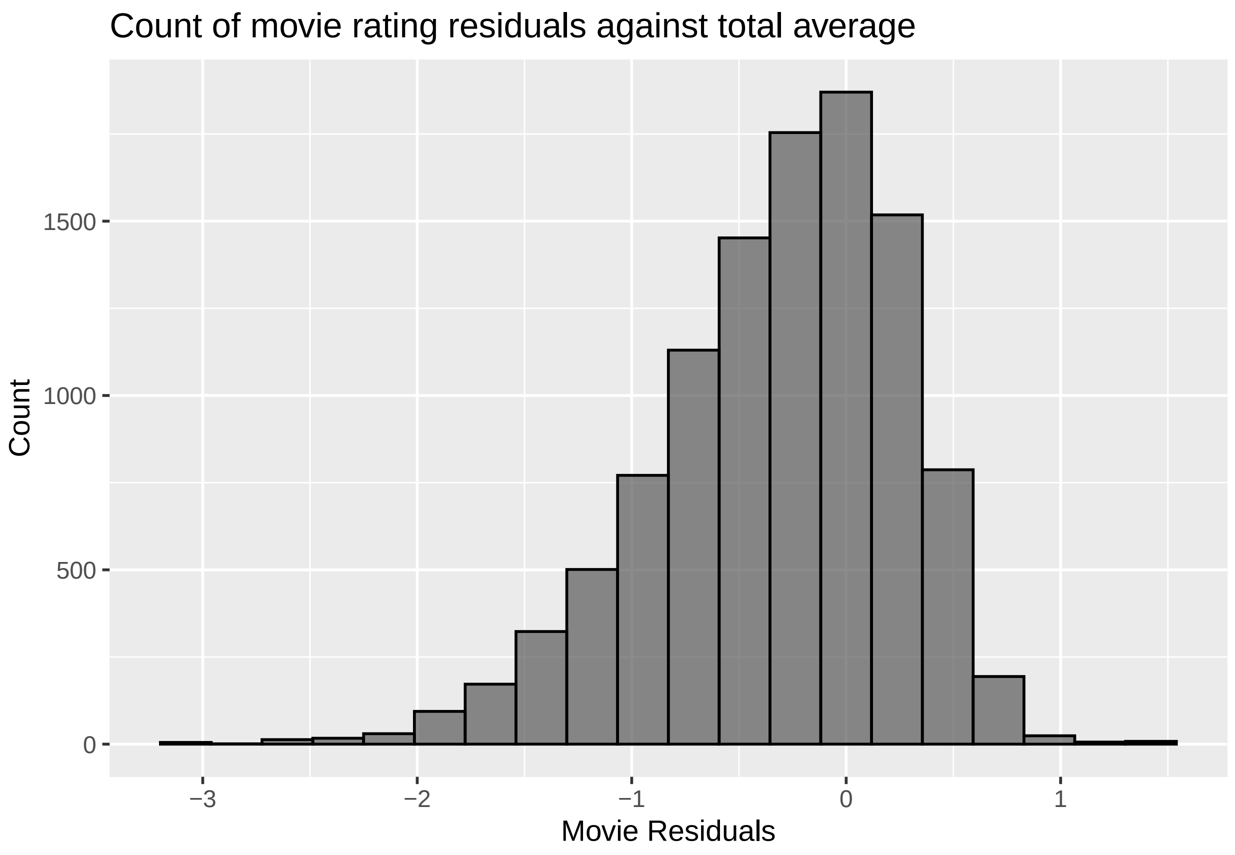
method	RMSE
Naive Model	1.060696

The average of ratings in our training set is 3.513; the RMSE of predicting that all the movies in the testing set is 3.513 is 1.061.

2.4.4 Movie Effects Model

The next step is to consider how each movie's average rating can improve our model. The residual of movie's average rating with the dataset's average rating is calculated. The prediction will be the average of all movie plus residual of that particular movie.

```r
# MODEL 2 - Movie Average; Model that adjusts prediction by quantifying average rating of the movie
# Work out residual of movie ratings vs total average
movieAvgs <- training %>%
  group_by(movieId) %>%
  summarise(movieRsd = mean(rating - average))

# Plot of movie residuals vs count of movies
movieAvgs %>% ggplot(aes(movieRsd)) +
  geom_histogram(bins = 20,
               alpha = .7,
                 col = "black") +
  labs(title = "Count of movie rating residuals against total average",
       x = "Movie Residuals",
       y = "Count")
```



```r
# Calculate predictions of model
movieAvgsPred <- average + testing %>%
  left_join(movieAvgs, by = 'movieId') %>%
  pull(movieRsd)

# Calculate RMSE of new model, add value to table & compare
rmseResults <- bind_rows(rmseResults,
                          data_frame(method = "Movie Residuals Model",
                                     RMSE = RMSE(testing$rating, movieAvgsPred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571

Our RMSE has improved to 0.944. Our graph shows that most movies's residuals are centered around 0 and there is a large variance in average movie reviews. The graph's residual range also confirms that we have not made any mathematical errors; The minimum a movie can average is 0.5 and therefore the minimum residual has to be -3, while the maximum average for a movie is 5 and there a residual of 1.5. Not many movies average a rating of 0.5 or 5.

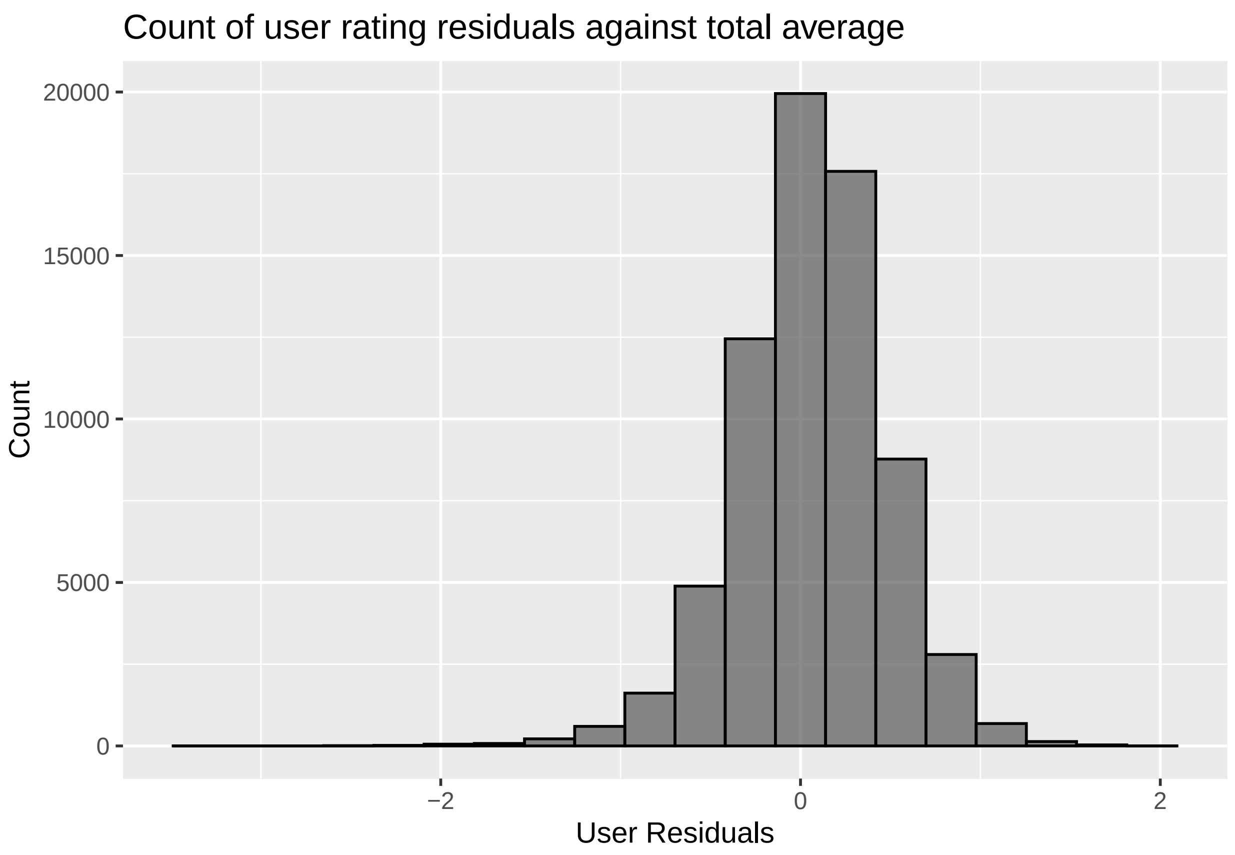
2.4.5 User & Movie Effects Model

We will do the same thing for users. The chart below shows on average some users rate movies better or worse than other users. For our new set of movie predictions we will consider what the user's average rating is compared to the movie's average. Once calculated, each movie's rating will be adjusted by both movie's average residual and the user's average residual.

```r
# MODEL 3 - Users Average; Model takes into account users average ratings
# Work out user review averages against total average and the movie residual
userAvgs <- training %>%
  left_join(movieAvgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(userRsd = mean(rating - average - movieRsd))

# Build predictions of model
userMoviePred <- testing %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  mutate(pred = average + movieRsd + userRsd) %>%
  pull(pred)

# Calculate RMSE of new model, add value to table & compare
rmseResults <- bind_rows(rmseResults, data_frame(method = "User & Movie Residuals Model",
                                   RMSE = RMSE(testing$rating, userMoviePred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206

```r
# Plot of user residuals vs count of movies
userAvgs %>% ggplot(aes(userRsd)) +
  geom_histogram(bins = 20, alpha = .7, col = "black") +
  labs(title = "Count of user rating residuals against total average",
       x = "User Residuals", y = "Count")
```



Our RMSE has now fallen to .8662, user ratings have less variablity than movie average ratings. From the graph, the user residuals have less variance than movie's residuals. This shows that people's ratings are less variable than a movie's average score.

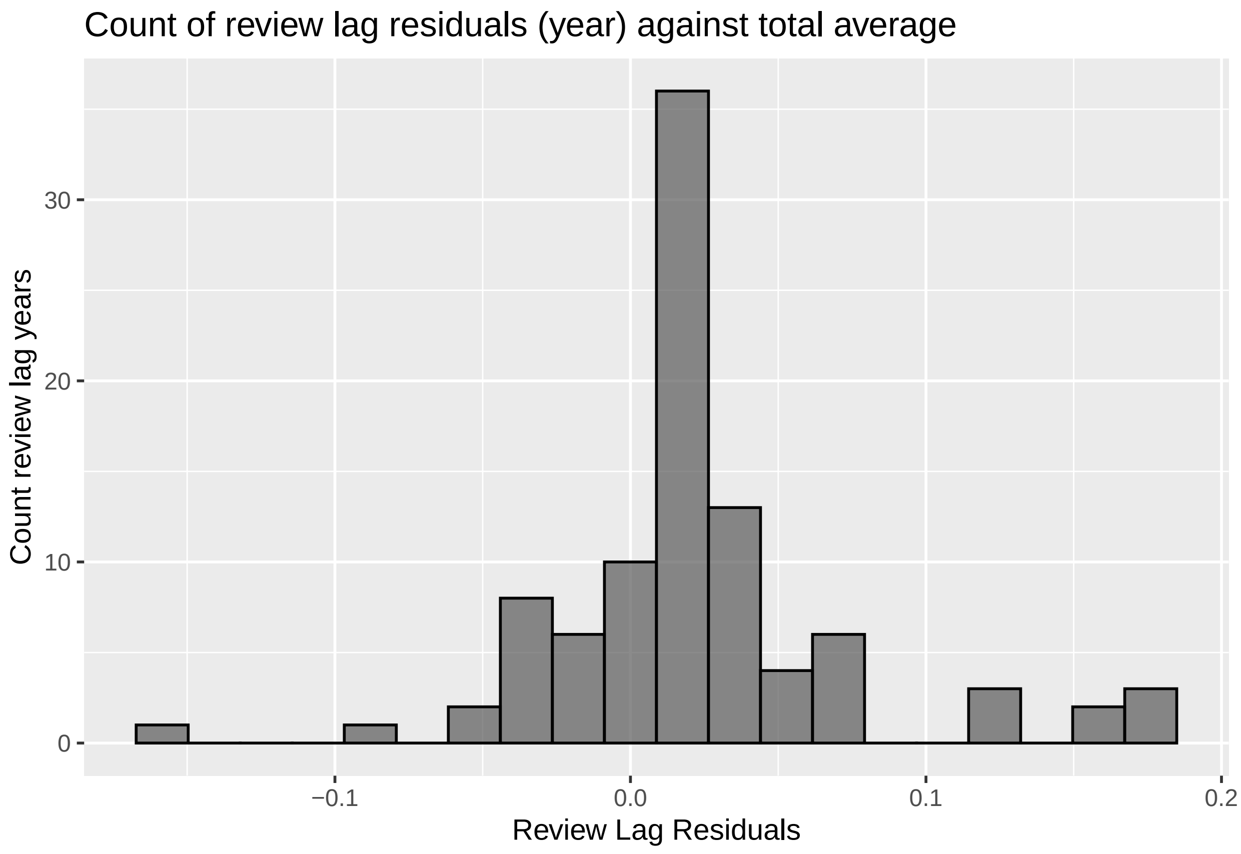
2.4.6 User & Movie & Review Lag Effects Model

Using the review lag variable calculated in section 2.3.2.1 Transform Timestamp and Title we can augment our previous model further. We add a new term, a residual for how many years between the release of the movie and the rating of the movie.

```r
## MODEL 4 - Movie & User & ReviewLag effects
# Work out review time lag averages against total average and the movie & user residual
reviewLagAvgs <- training %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  group_by(reviewLag) %>%
  summarise(reviewLagRsd = mean(rating - average - movieRsd - userRsd))

# Build predictions of model
userMoviePred <- testing %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  left_join(reviewLagAvgs, by = 'reviewLag') %>%
  mutate(pred = average + movieRsd + userRsd + reviewLagRsd) %>%
  pull(pred)

# Calculate RMSE of new model, add value to table & compare
rmseResults <- bind_rows(rmseResults, data_frame(method = "User & Movie & Review Lag Residuals Model",
                                    RMSE = RMSE(testing$rating, userMoviePred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206
User & Movie & Review Lag Residuals Model	0.8657548

```
# Plot of review lag residuals vs count of movies
reviewLagAvgs %>% ggplot(aes(reviewLagRsd)) +
  geom_histogram(bins = 20, alpha = .7, col = "black") +
  labs(title = "Count of review lag residuals (year) against total average",
       x = "Review Lag Residuals",
       y = "Count review lag years")
```



By adjusting our predictions by factoring the time between release and review we have further reduced the RMSE. The residuals and vairability for review lag are very small compared to user residuals and movie residuals, indicating even lower variance.

2.4.7 User & Movie & Review Lag & Genre Effects Model

This model takes the previous model created and adds the genres variables calculated earlier in the report. We calculate residuals for each genre and then adjust the final prediction based on if movie is part of the genre.

```r
## MODEL 5- Movie & User & Review Lag & Genre effects
# Create dataframe to calculate and hold residual for first genre
genreResids <- training %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  left_join(reviewLagAvgs, by = 'reviewLag') %>%
  filter(noGenre == 1) %>%
  summarise(Residual = mean(rating - average - movieRsd - userRsd - reviewLagRsd))

# Loop through, calculate and store residuals for the rest of the genres
for (col in c(10:28)) {

   genreResid <- training %>%
   left_join(movieAvgs, by = 'movieId') %>%
   left_join(userAvgs, by = 'userId') %>%
   left_join(reviewLagAvgs, by = 'reviewLag') %>%
   filter(.[[col]] == 1) %>%
   summarise(Residual = mean(rating - average - movieRsd - userRsd - reviewLagRsd))

genreResids <- rbind(genreResids, genreResid)
}

# Take genre residuals and add them to predictions if a movie falls within a genre
GenrePredictions <- testing %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  left_join(reviewLagAvgs, by = 'reviewLag') %>%
  mutate(pred = average + movieRsd + userRsd +
           if_else(noGenre == 1, genreResids$Residual[1], 0) +
           if_else(Action == 1, genreResids$Residual[2], 0) +
           if_else(Adventure == 1, genreResids$Residual[3], 0) +
           if_else(Animation == 1, genreResids$Residual[4], 0) +
           if_else(Children == 1, genreResids$Residual[5], 0) +
           if_else(Comedy == 1, genreResids$Residual[6], 0) +
           if_else(Crime == 1, genreResids$Residual[7], 0) +
           if_else(Documentary == 1, genreResids$Residual[8], 0) +
           if_else(Drama == 1, genreResids$Residual[9], 0) +
           if_else(Fantasy == 1, genreResids$Residual[10], 0) +
           if_else(FilmNoir == 1, genreResids$Residual[11], 0) +
           if_else(Horror == 1, genreResids$Residual[12], 0) +
           if_else(IMAX == 1, genreResids$Residual[13], 0) +
           if_else(Musical == 1, genreResids$Residual[14], 0) +
           if_else(Mystery == 1, genreResids$Residual[15], 0) +
           if_else(Romance == 1, genreResids$Residual[16], 0) +
           if_else(SciFi == 1, genreResids$Residual[17], 0) +
           if_else(Thriller == 1, genreResids$Residual[18], 0) +
           if_else(War == 1, genreResids$Residual[19], 0) +
           if_else(Western == 1, genreResids$Residual[20], 0)) %>%
  pull(pred)

# Calculate RMSE of new model, add value to table & compare
rmseResults <- bind_rows(rmseResults, data_frame(method = "User & Movie & Review Lag & Genres Residuals
                                    RMSE = RMSE(testing$rating, GenrePredictions)))
```

```
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206
User & Movie & Review Lag Residuals Model	0.8657548
User & Movie & Review Lag & Genres Residuals Model	0.8661525

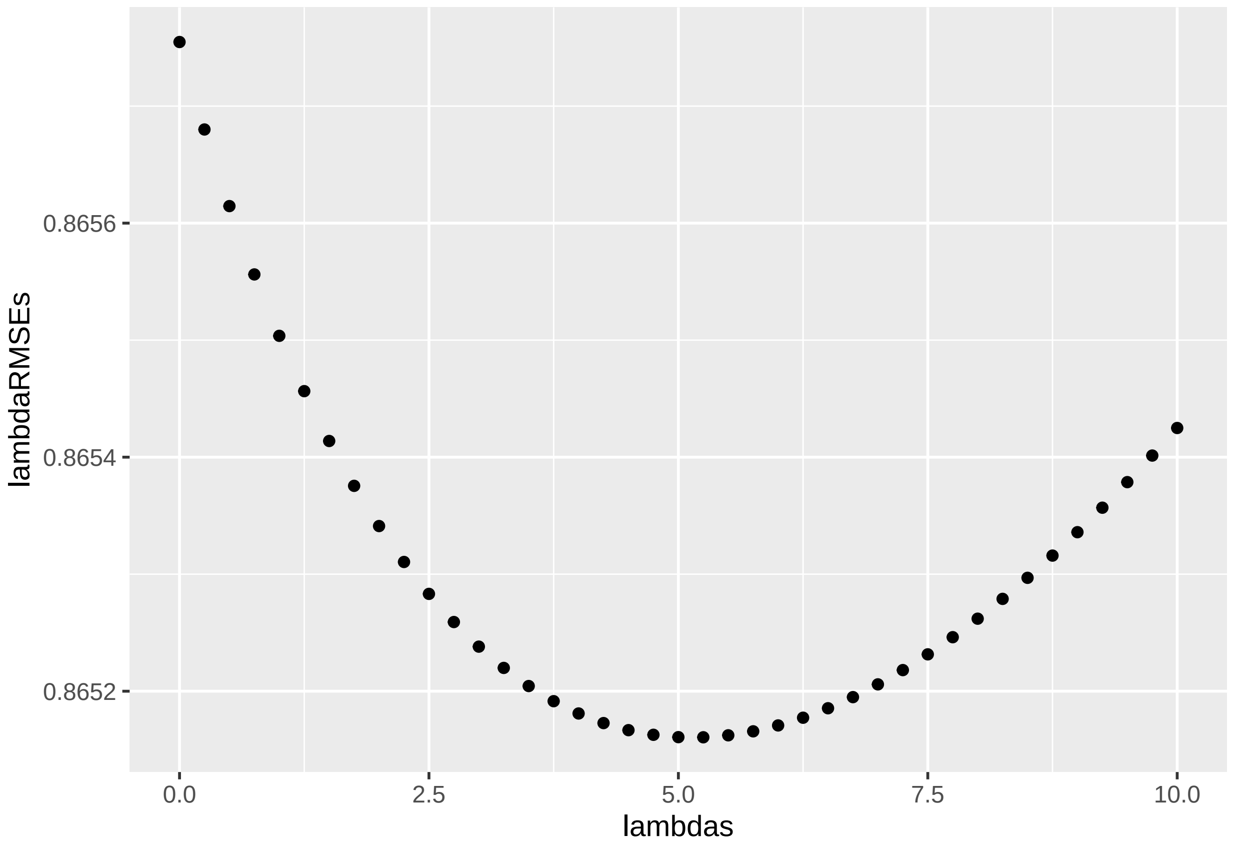
Incorporating genres made our RMSE's slighty worse. My hypothesis here is that genres are related to users and their preference for genres and that is how they should be incorporated into any model. The genre's residuals themselves do not seem to improve our model and will not be incorporated in going forwards.

2.4.8 Regularisation User & Movie & Review Lag Effects Model

Regularisation is a method which allows us to penalise large estimates that come from a small sample. Therefore rather than treating say a movie's redisuals which has 10 reviews the same as one which has 1000 reviews; small sample sizes are penalised. In the code below we apply regularisation to all variables in model 4 built in section 2.4.6, it is anticipated that regularsation will have the largest effect on a movie's residuals because the variance is greatest in that variable.

To implement this penalty we have to choose a penalty value, lambda. The approach is to add lambda to the count of residuals; so for each movie's residual we will get the sum of all a movie's ratings minus the average rating for all movie, the movie's total residuals. Then divide by the amount of ratings recieved by the movie plus the penalty value lambda. When we have a lot of reviews, the effect of the penalty will be small because we are almost dividing by the same number. But when the total number of instances is small, adding that same penalty figure will have a relatively larger effect and therefore reduced the residual for that movie towards the mean. The same is done for user and review lag residuals. Because lambda is a tuning pararmeter we will test for the value which reduces our RMSE the most.

```
# MODEL 6 - Regularisation; Model penalises large estimates from small sample sizes
# Sequence of lambdas to test
lambdas <- seq(0, 10, .25)

# Run sequence through model and calculate all their RMSEs, with progress bar
lambdaRMSEs <- pbsapply(lambdas, function(l){

  averageRating <- mean(training$rating)

  movieAvgs <- training %>%
    group_by(movieId) %>%
    summarise(movieRsds = sum(rating - average) / (n() + l))

  userAvgs <- training %>%
    left_join(movieAvgs, by = 'movieId') %>%
    group_by(userId) %>%
    summarise(userRsds = sum(rating - movieRsds - average) / (n() + l))

  reviewLagAvgs <- training %>%
    left_join(movieAvgs, by = 'movieId') %>%
    left_join(userAvgs, by = 'userId') %>%
    group_by(reviewLag) %>%
    summarise(reviewLagRsds = sum(rating - average - movieRsds - userRsds) / (n() + l))
  predictions <- testing %>%
    left_join(movieAvgs, by = 'movieId') %>%
    left_join(userAvgs, by = 'userId') %>%
    left_join(reviewLagAvgs, by = 'reviewLag') %>%
    mutate(pred = average + movieRsds + userRsds + reviewLagRsds) %>%
    pull(pred)

  return(RMSE(testing$rating, predictions))
})

# Plot of lambdas & select lambda which gives lowest RMSE
qplot(lambdas, lambdaRMSEs)
```



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

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

```
# Work out new residuals with generated lambda
movieAvgs <- training %>%
  group_by(movieId) %>%
  summarise(movieRsds = sum(rating - average) / (n() + lambda))

userAvgs <- training %>%
  left_join(movieAvgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(userRsds = sum(rating - movieRsds - average) / (n() + lambda))
```

```
reviewLagAvgs <- training %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  group_by(reviewLag) %>%
  summarise(reviewLagRsd = sum(rating - average - movieRsds - userRsds) / (n() + lambda))

regPred <- testing %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  left_join(reviewLagAvgs, by = 'reviewLag') %>%
  mutate(pred = average + movieRsds + userRsds + reviewLagRsd) %>%
  pull(pred)

# Calculate RMSE of new model, add value to table & compare
rmseResults <- bind_rows(rmseResults,
                         data_frame(method = "User & Movie & Review Lag Regularised Model",
                                    RMSE = RMSE(testing$rating, regPred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206
User & Movie & Review Lag Residuals Model	0.8657548
User & Movie & Review Lag & Genres Residuals Model	0.8661525
User & Movie & Review Lag Regularised Model	0.8651606

As can be seem from our plot, the best penalty value to use is 5.25, from which we reduce our RMSE to 0.865 which is close to our target. Because we have split our edx dataset into a training and testing set, therefore reducing our total number of observations, if we trained this model on the whole edx dataset and test on the validation set we will lower our RMSE further. Remember when we split for the training / testing sets, we did not eliminate many movies which did not match in the two datasets, therefore most movies match in the two sets and combining the two sets will give use better residuals to work from to test on the validation dataset.

2.4.9 Recommenderlab - Matrix Factorisation

Matrix factorisation is a popular technique to use for recommendation systems. Model 5 showed when we tried to include residuals for genres we made our RMSEs worse. The hypothesis for this is that all genres themselves do not have an effect on movie ratings, but it is user's who prefer particular genres and eschew others. This can be considered using matrix factorisation.

Matrix factorisation will approximate ratings by decomposing the user-movie interaction matrix into the product of a couple of lower dimensional rectangular matrices P and Q. Matrix factorisation will predict ratings for movies that users have not rated based on their existing reviews. Matrix P is the latent factors of users, while matrix Q represents each movies. The recommender system predicts unknown entries in the decomposed rating matrix based on observed values. More details including the optimisation algorithm are available in the documentation of the *recosystem* package located here: https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

```
# MODEL 7 - Matrix Factorisation;
# Clear unused memory
invisible(gc())
```

```r
# Create a matrix copy to disk of training and testing set where we only take userID, movieID & ratings
train2 <- training %>%
  select(userId, movieId, rating) %>%
  as.matrix() %>%
  write.table(file = "train2.txt", sep = " ", row.names = F, col.names = F)

test2 <- testing %>%
  select(userId, movieId, rating) %>%
  as.matrix() %>%
  write.table(file = "test2.txt", sep = " ", row.names = F, col.names = F)

# Build reco object used to construct recommender model and conduct prediction.
mfModel <- Reco()

# Load data in format reco requires
set.seed(17, sample.kind = "Rounding")
trainSet <- data_file(paste0(getwd(), "/train2.txt"), index1 = FALSE)
testSet <- data_file(paste0(getwd(), "/test2.txt"), index1 = FALSE)

# Tune RECO parameters, parármeters optimised to lower computation time
opts <- mfModel$tune(trainSet,
               opts = list(dim = c(1:15), # Number of latent factors
                  lrate = c(0.1, 0.2), # Learning rate, step size of gradient decent
                  costp_l1 = 0,
                  costq_l1 = 0,
                  nthread = 4,
                  niter = 10,
                  verbose = FALSE))

# Train RECO model
mfModel$train(trainSet, opts = c(opts$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9587   9.7970e+06
##    1       0.8664   8.3342e+06
##    2       0.8276   7.7782e+06
##    3       0.8061   7.4855e+06
##    4       0.7926   7.3127e+06
##    5       0.7835   7.1953e+06
##    6       0.7770   7.1159e+06
##    7       0.7721   7.0582e+06
##    8       0.7683   7.0111e+06
##    9       0.7655   6.9780e+06
##   10       0.7630   6.9474e+06
##   11       0.7610   6.9264e+06
##   12       0.7592   6.9055e+06
##   13       0.7575   6.8885e+06
##   14       0.7560   6.8715e+06
##   15       0.7547   6.8597e+06
##   16       0.7534   6.8469e+06
##   17       0.7521   6.8355e+06
##   18       0.7510   6.8248e+06
##   19       0.7498   6.8152e+06
##   20       0.7487   6.8070e+06
##   21       0.7475   6.7945e+06
##   22       0.7464   6.7873e+06
##   23       0.7453   6.7766e+06
##   24       0.7443   6.7685e+06
##   25       0.7431   6.7601e+06
##   26       0.7422   6.7528e+06
##   27       0.7411   6.7445e+06
##   28       0.7402   6.7382e+06
##   29       0.7392   6.7302e+06
```

```
# Create Predictions
predFile = tempfile()
mfModel$predict(testSet, out_file(predFile))
```

```
## prediction output generated at /var/folders/vr/n4ryb0s97dd1yc59z1k9jrn00000gn/T//RtmpjtnBLn/file579e
```

```
# Create prediction data and real review results data set
scoresReal <- read.table("test2.txt", header = FALSE, sep = " ")$V3
scoresPred <- scan(predFile)

# Find RMSE
rmseResults <- bind_rows(rmseResults, data_frame(method = "Matrix Factorisation with Gradient Descent",
                                      RMSE = RMSE(scoresReal, scoresPred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206
User & Movie & Review Lag Residuals Model	0.8657548
User & Movie & Review Lag & Genres Residuals Model	0.8661525
User & Movie & Review Lag Regularised Model	0.8651606
Matrix Factorisation with Gradient Descent	0.7957248

Using matrix factorisation we achieve an RMSE of lower than 0.8 on our testing data, this is by far the best result on the training data so far.

3 Results

The models produced above show gradual improvement in our RMSE scores when we used a regression modelling approach and add new elements. In our regression models we explored the effect of movies, users, time lag between review and release of the movie and also genres. Each effect improved the overall model apart from genres, movie genres have the distinction of being based upon user preference, users prefer some types of movies rather than others, therefore just adding the calculated residuals for each genre made our model worse, because user preference was not taken into account. Our best regression model was produced by combining movie, user and time lag effects. As observed variance decreases with each variable explored, the improvement in RMSE also reduced.

Regularation was then built upon the regression model, where we penalise large estimates that come from small sample sizes, this also helps us avoid overfitting by penalising the magnitude of the parameters. We then used cross validation to help us pick the best penalty value. From the RMSE calculated in 2.4.8 Regularisation, we observe that it improved our RMSE to just above our target of 0.865. The hypothesis is, when training on the whole edx set, this method will improve our results again as there will be more training data available.

Finally, matrix factorisation with gradient descent using the *recosystem* engine produced the best results. An RMSE of 0.795 was achieved. Tuning this model was difficult due to the number of penalty parameters involved. The recommender engine in the package produces a whole ratings matrix of movies and users by the product of 2 lower dimensional matrices. This overcomes the problem of user genre preference we saw in the regression models. However further analysis of this is difficult due to the black box nature of the engine.

In gerneral, modelling such a large dataset is a challenge in the RStudio environment. The size of the dataset is the reason why we did not use *train* in the caret package for our regression modelling choosing to use residuals instead. Other models such as random forest were explored, but the size of the dataset really affects performance. One important learning garnered from exploring a dataset of this size is to be sure of coding before execution. Time wasted from typos in code is one of the biggest holdups to the project.

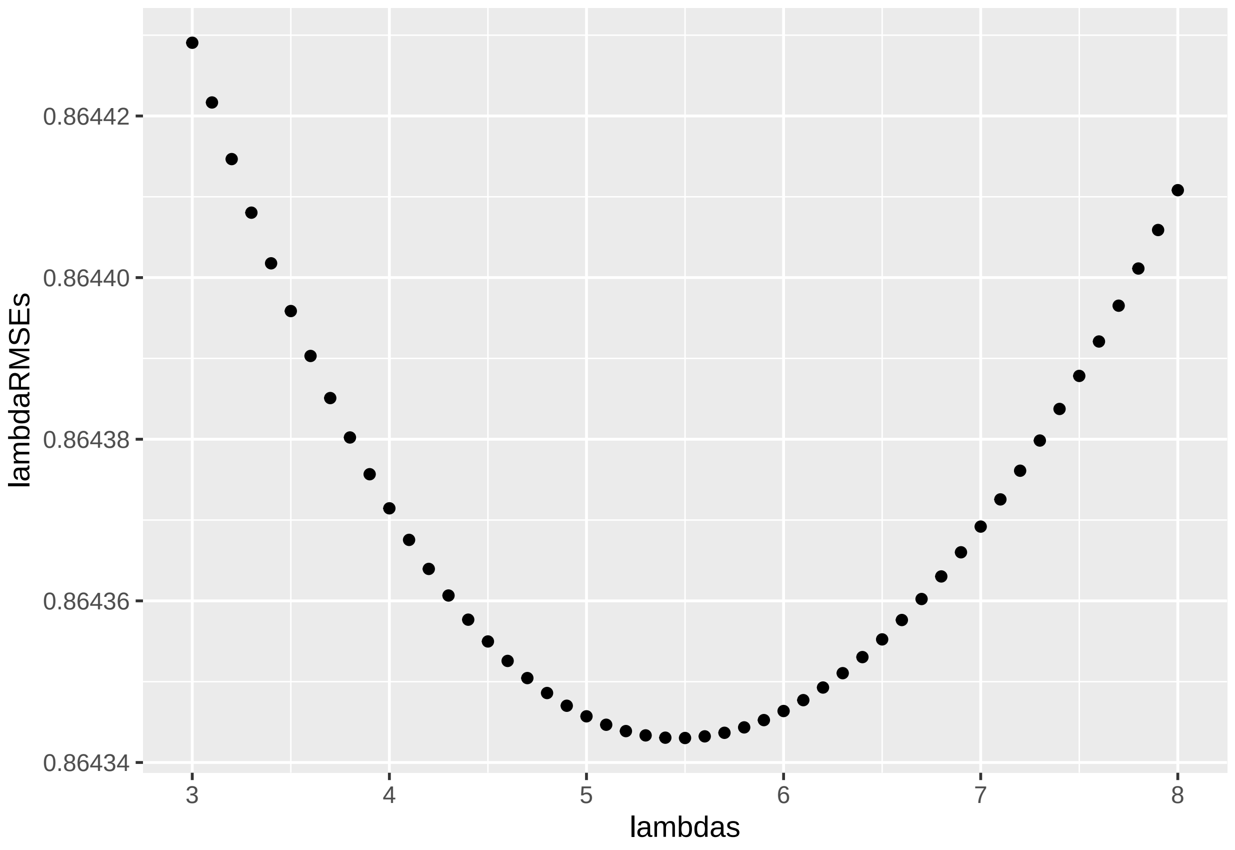
A final RMSE is be calculated below by using the whole edx to retrain Model 6: Regularisation on User, Movie and ReviewLag and Model 7: Matrix Factorisation using Gradient Descent as they both show the potential to garner an RMSE of lower than our target of .865.

3.1 Regularisation Final Model

In this model run, we will decrease the step size of the lambdas we test increasing the amount of lambdas we test; but decrease the range thereby improving processing time. We also calculate the RMSE on the validation set.

```r
# FINAL MODEL 1 - Regularisation;
# Sequence of lambdas to test
lambdas <- seq(3, 8, .1)

# Run sequence through model and calculate all their RMSEs, with progress bar
lambdaRMSEs <- pbsapply(lambdas, function(l){

  averageRating <- mean(edx$rating)

  movieAvgs <- edx %>%
    group_by(movieId) %>%
    summarise(movieRsds = sum(rating - average) / (n() + l))

  userAvgs <- edx %>%
    left_join(movieAvgs, by = 'movieId') %>%
    group_by(userId) %>%
    summarise(userRsds = sum(rating - movieRsds - average) / (n() + l))

  reviewLagAvgs <- edx %>%
    left_join(movieAvgs, by = 'movieId') %>%
    left_join(userAvgs, by = 'userId') %>%
    group_by(reviewLag) %>%
    summarise(reviewLagRsds = sum(rating - average - movieRsds - userRsds) / (n() + l))

  predictions <- validation %>%
    left_join(movieAvgs, by = 'movieId') %>%
    left_join(userAvgs, by = 'userId') %>%
    left_join(reviewLagAvgs, by = 'reviewLag') %>%
    mutate(pred = average + movieRsds + userRsds + reviewLagRsds) %>%
    pull(pred)

  return(RMSE(validation$rating, predictions))
})
```

```
# Plot of lambdas & select lambda which gives lowest RMSE
qplot(lambdas, lambdaRMSEs)
```



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

```
## [1] 5.5
```

```
# Work out new residuals with generated lambda
movieAvgs <- edx %>%
  group_by(movieId) %>%
  summarise(movieRsds = sum(rating - average) / (n() + lambda))

userAvgs <- edx %>%
  left_join(movieAvgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarise(userRsds = sum(rating - movieRsds - average) / (n() + lambda))

reviewLagAvgs <- edx %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  group_by(reviewLag) %>%
  summarise(reviewLagRsd = sum(rating - average - movieRsds - userRsds) / (n() + lambda))

finalRegPred <- validation %>%
  left_join(movieAvgs, by = 'movieId') %>%
  left_join(userAvgs, by = 'userId') %>%
  left_join(reviewLagAvgs, by = 'reviewLag') %>%
  mutate(pred = average + movieRsds + userRsds + reviewLagRsd) %>%
  pull(pred)

# Calculate RMSE of new model, add value to table & compare
rmseResults <- bind_rows(rmseResults,
                         data_frame(method = "Edx Set User & Movie & Review Lag Regularised Model on Va
                                    RMSE = RMSE(validation$rating, finalRegPred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206
User & Movie & Review Lag Residuals Model	0.8657548
User & Movie & Review Lag & Genres Residuals Model	0.8661525
User & Movie & Review Lag Regularised Model	0.8651606
Matrix Factorisation with Gradient Descent	0.7957248
Edx Set User & Movie & Review Lag Regularised Model on Validation Data	0.8643430

As predicted, with more data in our edx set, our predictions on the validation set are more accurate than when we split out our data. The lambda value we used for regularisation also increased to 5.5 due to the fact we have more data. Validation data was not used in the training of this model.

3.2 Matrix Factorisation Final Model

10 fold cross validation is used to train our final Matrix Factorisation model.

```
# FINAL MODEL 2 - Matrix Factorisation;
# Clear unused memory
invisible(gc())

# Create a matrix copy to disk of training and testing set where we only take userID, movieID & ratings
edxTrain <- edx %>%
  select(userId, movieId, rating) %>%
  as.matrix() %>%
  write.table(file = "edx.txt", sep = " ", row.names = F, col.names = F)

valid <- validation %>%
  select(userId, movieId, rating) %>%
  as.matrix() %>%
  write.table(file = "valid.txt", sep = " ", row.names = F, col.names = F)

# Build reco object used to construct recommender model and conduct prediction.
mfModel <- Reco()

# Load data in format reco requires
set.seed(17, sample.kind = "Rounding")
```

```
## Warning in set.seed(17, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
edxSet <- data_file(paste0(getwd(), "/edx.txt"), index1 = FALSE)
validSet <- data_file(paste0(getwd(), "/valid.txt"), index1 = FALSE)
```

```
# Tune RECO parameters, parameters optimised to lower computation time
opts <- mfModel$tune(edxSet,
               opts = list(dim = c(1:15), # Number of latent factors
                  lrate = c(0.1, 0.2), # Learning rate, step size of gradient decent
                  costp_l1 = 0,
                  costq_l1 = 0,
                  nthread = 4,
                  niter = 10,
                  nfold = 10, # Use 10 fold cross validation
                  verbose = FALSE))

# Train RECO model
mfModel$train(edxSet, opts = c(opts$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9507   1.0706e+07
##    1       0.8627   9.1791e+06
##    2       0.8254   8.5895e+06
##    3       0.8049   8.2847e+06
##    4       0.7917   8.1017e+06
##    5       0.7827   7.9765e+06
##    6       0.7761   7.8913e+06
##    7       0.7716   7.8284e+06
##    8       0.7682   7.7850e+06
##    9       0.7655   7.7494e+06
##   10       0.7632   7.7204e+06
##   11       0.7612   7.6969e+06
##   12       0.7593   7.6763e+06
##   13       0.7577   7.6573e+06
##   14       0.7562   7.6404e+06
##   15       0.7548   7.6250e+06
##   16       0.7534   7.6140e+06
##   17       0.7521   7.6019e+06
##   18       0.7508   7.5885e+06
##   19       0.7495   7.5752e+06
##   20       0.7483   7.5648e+06
##   21       0.7471   7.5545e+06
##   22       0.7460   7.5444e+06
##   23       0.7449   7.5352e+06
##   24       0.7437   7.5241e+06
##   25       0.7428   7.5175e+06
##   26       0.7417   7.5088e+06
##   27       0.7407   7.5000e+06
##   28       0.7398   7.4911e+06
##   29       0.7389   7.4844e+06
```

```
# Predictions
predFile = tempfile()
mfModel$predict(validSet, out_file(predFile))
```

```
## prediction output generated at /var/folders/vr/n4ryb0s97dd1yc59z1k9jrn00000gn/T//RtmpjtnBLn/file579e
```

```
# Create prediction data and real review results data set
scoresReal <- read.table("valid.txt", header = FALSE, sep = " ")$V3
scoresPred <- scan(predFile)
```

```r
# Calculate RMSE
rmseResults <- bind_rows(rmseResults,
                         data_frame(method = "Edx Set Matrix Factorisation with Gradient Descent on Val
                                    RMSE = RMSE(scoresReal, scoresPred)))
rmseResults %>% knitr::kable()
```

method	RMSE
Naive Model	1.0606957
Movie Residuals Model	0.9440571
User & Movie Residuals Model	0.8662206
User & Movie & Review Lag Residuals Model	0.8657548
User & Movie & Review Lag & Genres Residuals Model	0.8661525
User & Movie & Review Lag Regularised Model	0.8651606
Matrix Factorisation with Gradient Descent	0.7957248
Edx Set User & Movie & Review Lag Regularised Model on Validation Data	0.8643430
Edx Set Matrix Factorisation with Gradient Descent on Validation Data	0.7902988

Using matrix factorisation trained with the whole edx we improves RMSEs further. This is due to the extra actual values we have, allowing the optimisation engine to more accurately predict missing values. From the results table the best model to use is matrix factorisation with gradient descent. Again, no validation data is used to train the model.

4 Conclusion

This report explored the movielens dataset and built machine learning models to predict ratings on an unseen validation set. The goal of the project is to predict movie reviews made by users on an unseen validation set to an accuracy (RMSE, Root mean square error) of below 0.865. In the report above, we have constructed 2 models which predict the ratings of validation set to below our goal. Out of the two model, Matrix Factorisation and Regularisation on regression of movie & user & time lag effects; Matrix Factorisation is the best performing model. However Matrix Factorisation is also a black box to explore and much more complex to tune. I believe that further work such as exploring other tuning criteria in the matrix factorisation engine such as the number of latent factors, learning rate and cost on the Matrix Factorisation model would lead to better results again. Other models such as ensemble methods may also be possible, however computational requirements are always a consideration when using a interpreter language such as R on such a large dataset.