Movie Recommendation

madibaa

2024-03-25

Introduction

In this project, we are looking to create a movie recommendation system using the 10M version of the MovieLens dataset.The goal of this project is the create an algorithm that minimises the root mean squared error (RMSE) < 0.86490, which translates to greater accuracy in predicting a user's movie preferences.

Here is a broad overview of the project's implementation:

1. Import the MovieLens dataset and generate the validation and final holdout sets using the provided code.
2. Conduct exploratory analyses on the validation dataset
3. Generate training and test sets from the validation set
4. Implement a content-based filtering strategy
5. Implement matrix factorisation using the recosystem package
6. Evaluate the model using the final holdout set

Let's dive right in!

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

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

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 4.3.2
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tidyverse)
library(caret)

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

options(timeout = 120)

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

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

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

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

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

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

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

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

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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

Exploratory analyses

Let's load some additional packages to aid our analysis.

```
library(data.table)
```

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

## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

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

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

```
library(lubridate)
```

Now, we start by conducting some initial exploratory analyses of edx.

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
```

```
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```
dim(edx)
```

```
## [1] 9000055       6
```

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
```

```
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
##
##
##
```

We see that edx comprises 6 columns: userId, movieId, rating, timestamp, title, and genres, with 900,055 observations.

Further exploratory analyses

To see the number of users and movies:

```
edx %>%
  summarise(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

We see that there are 69,878 unique users and 10,677 unique movies.

What is the mean rating and mean rating per user??

```
mean(edx$rating)
```

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

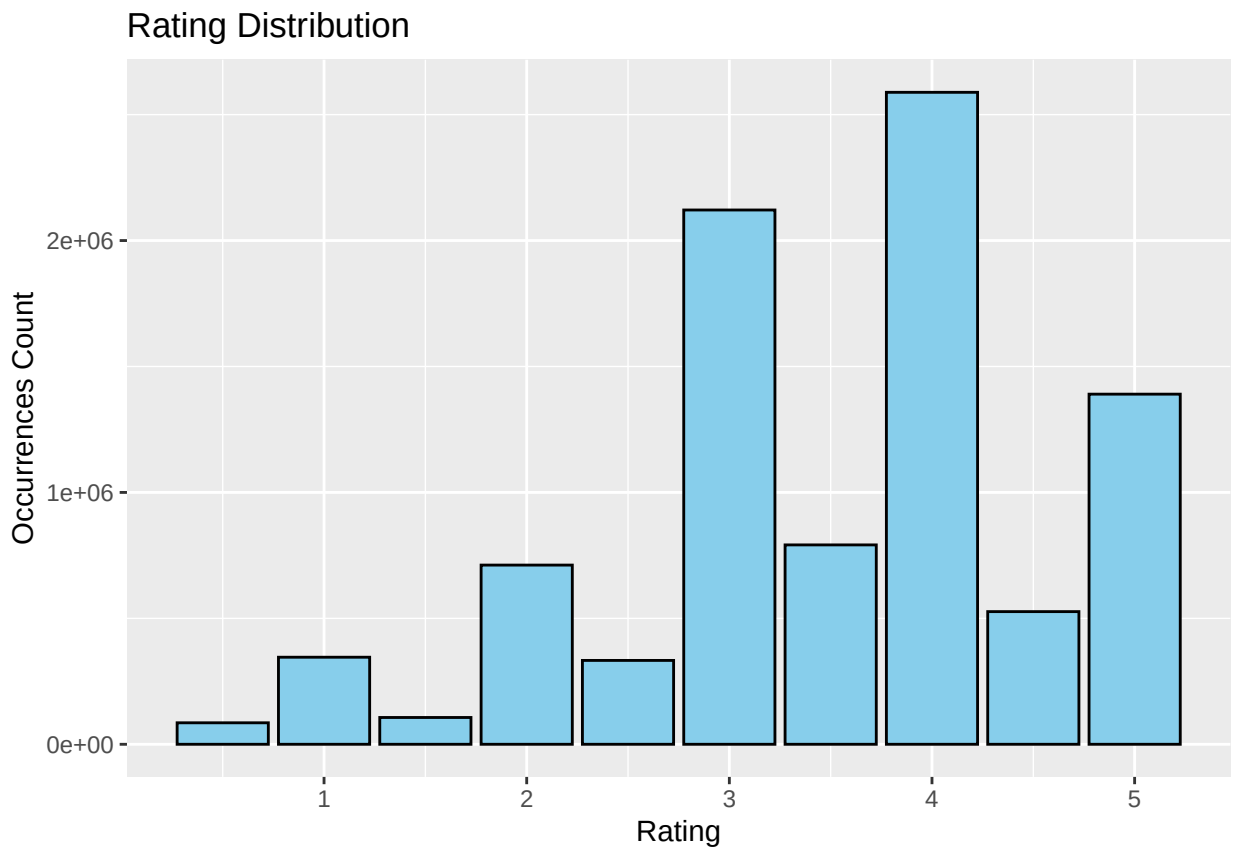
```
mean_per_user <- edx %>%
  group_by(userId) %>%
  summarise(mean_rating = mean(rating))
mean(mean_per_user$mean_rating)
```

```
## [1] 3.613602
```

The mean rating is 3.512, and mean rating per user is 3.613.

Let's visualise the distribution of ratings with a plot.

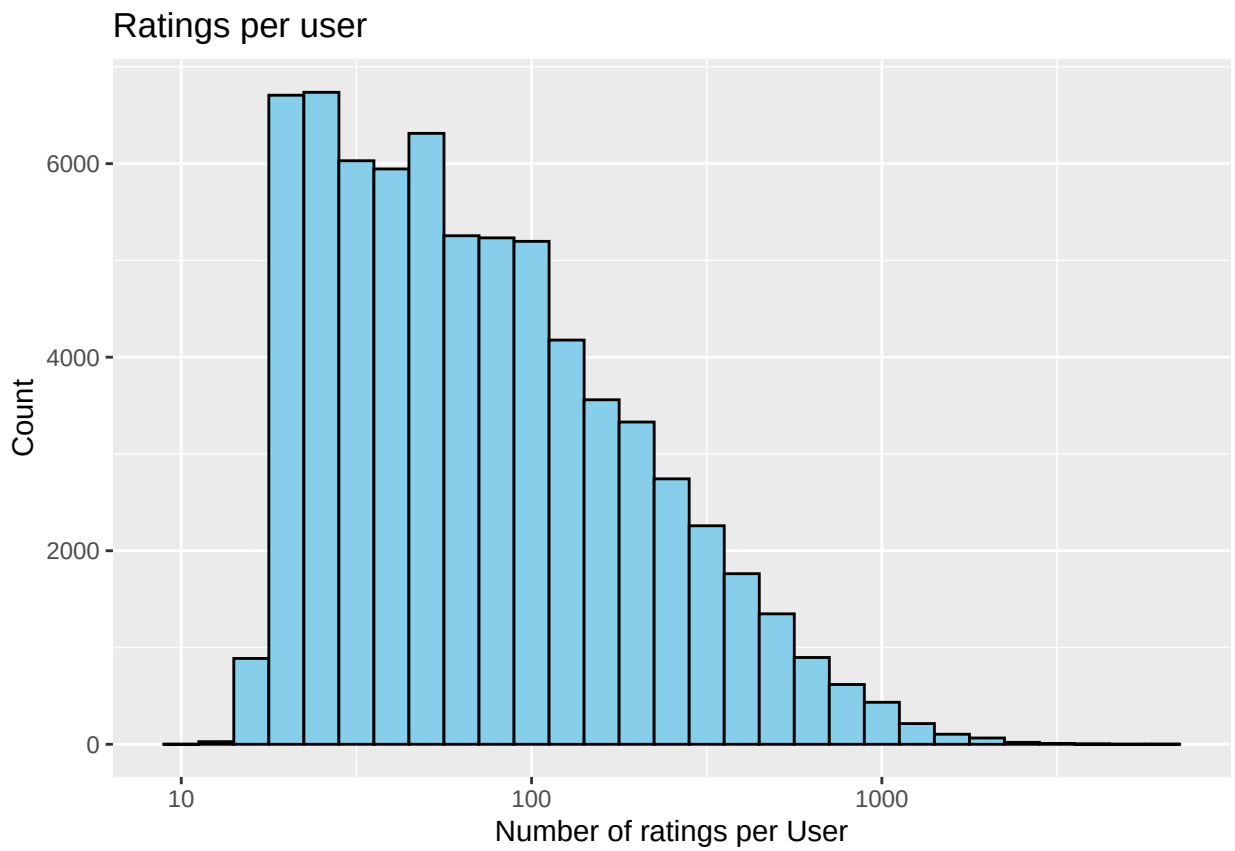
```
edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_bar(stat = "identity", fill = "skyblue", colour = "black") +
  ggtitle("Rating Distribution") +
  xlab("Rating") +
  ylab("Occurrences Count")
```



We see that users tend to favour higher ratings and whole number ratings.

Let's create another plot to visualise the number of movies rated per user.

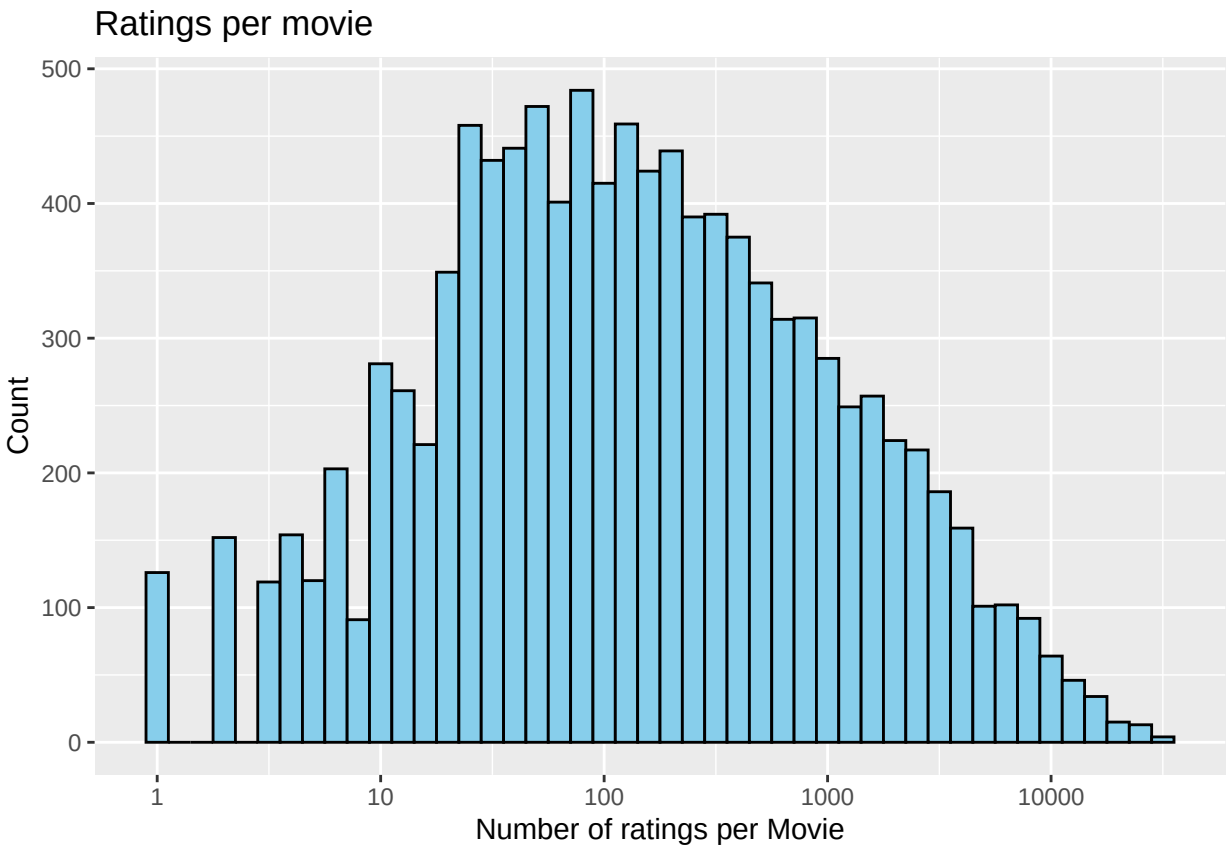
```
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = count)) +
  geom_histogram(binwidth = 0.1, fill = "skyblue", colour = "black") +
  ggtitle("Ratings per user") +
  xlab("Number of ratings per User") +
  ylab("Count") +
  scale_x_log10()
```



We see that most users rate less than 100 movies.

Let's create another plot to visualise the number of ratings per movie.

```
edx %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = count)) +
  geom_histogram(binwidth = 0.1, fill = "skyblue", colour = "black") +
  ggtitle("Ratings per movie") +
  xlab("Number of ratings per Movie") +
  ylab("Count") +
  scale_x_log10()
```



We see that most movies are rated between 10 and 1000 times.

Let's see the top 10 rated movies.

```
top10 <- edx %>%
  group_by(movieId) %>%
  summarise(title = title[1], rating = mean(rating)) %>%
  top_n(10, rating) %>%
  arrange(desc(rating)) %>%
  slice(1:10)
top10
```

```
## # A tibble: 10 x 3
##    movieId title                                                      rating
##      <int> <chr>                                                       <dbl>
##  1    3226 Hellhounds on My Trail (1999)                                5
##  2   33264 Satan's Tango (Sátántangó) (1994)                           5
##  3   42783 Shadows of Forgotten Ancestors (1964)                       5
##  4   51209 Fighting Elegy (Kenka erejii) (1966)                        5
##  5   53355 Sun Alley (Sonnenallee) (1999)                              5
##  6   64275 Blue Light, The (Das Blaue Licht) (1932)                    5
##  7    5194 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko t~   4.75
##  8   26048 Human Condition II, The (Ningen no joken II) (1959)         4.75
##  9   26073 Human Condition III, The (Ningen no joken III) (1961)       4.75
## 10   65001 Constantine's Sword (2007)                                  4.75
```

These are pretty obscure movies. Let's see the number of ratings each movie has.

```
top10 %>%
  inner_join(edx, by = "movieId") %>%
  group_by(movieId, title.x) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 10 x 3
## # Groups:   movieId [10]
##    movieId title.x                                                    count
##      <int> <chr>                                                      <int>
##  1    3226 Hellhounds on My Trail (1999)                                 1
##  2    5194 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to~     4
##  3   26048 Human Condition II, The (Ningen no joken II) (1959)           4
##  4   26073 Human Condition III, The (Ningen no joken III) (1961)         4
##  5   33264 Satan's Tango (Sátántangó) (1994)                            2
##  6   42783 Shadows of Forgotten Ancestors (1964)                        1
##  7   51209 Fighting Elegy (Kenka erejii) (1966)                         1
##  8   53355 Sun Alley (Sonnenallee) (1999)                               1
##  9   64275 Blue Light, The (Das Blaue Licht) (1932)                     1
## 10   65001 Constantine's Sword (2007)                                   2
```

Each of these movies has a pretty low number of ratings.

Let's see if the same holds true for the bottom 10 rated movies.

```
bottom10 <- edx %>%
  group_by(movieId) %>%
  summarise(title = title[1], rating = mean(rating)) %>%
  top_n(-10, rating) %>%
  arrange(rating) %>%
  slice(1:10)
bottom10
```

```
## # A tibble: 10 x 3
##    movieId title                                    rating
##      <int> <chr>                                     <dbl>
##  1    5805 Besotted (2001)                           0.5
##  2    8394 Hi-Line, The (1999)                       0.5
##  3   61768 Accused (Anklaget) (2005)                 0.5
##  4   63828 Confessions of a Superhero (2007)         0.5
##  5   64999 War of the Worlds 2: The Next Wave (2008) 0.5
##  6    8859 SuperBabies: Baby Geniuses 2 (2004)      0.795
##  7    7282 Hip Hop Witch, Da (2000)                 0.821
##  8   61348 Disaster Movie (2008)                    0.859
##  9    6483 From Justin to Kelly (2003)              0.902
## 10     604 Criminals (1996)                         1
```

```
bottom10 %>%
  inner_join(edx, by = "movieId") %>%
  group_by(movieId, title.x) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 10 x 3
## # Groups:   movieId [10]
##    movieId title.x                                  count
##      <int> <chr>                                    <int>
##  1     604 Criminals (1996)                             2
##  2    5805 Besotted (2001)                              2
##  3    6483 From Justin to Kelly (2003)                199
##  4    7282 Hip Hop Witch, Da (2000)                    14
##  5    8394 Hi-Line, The (1999)                          1
##  6    8859 SuperBabies: Baby Geniuses 2 (2004)         56
##  7   61348 Disaster Movie (2008)                       32
##  8   61768 Accused (Anklaget) (2005)                    1
##  9   63828 Confessions of a Superhero (2007)            1
## 10   64999 War of the Worlds 2: The Next Wave (2008)    2
```

The pattern of low ratings is still broadly true, with 6/10 of the worst rated movies having 1 or 2 ratings only.

From str(edx), we see that the timestamp is an integer. Let's extract the year and conduct a yearly rating count.

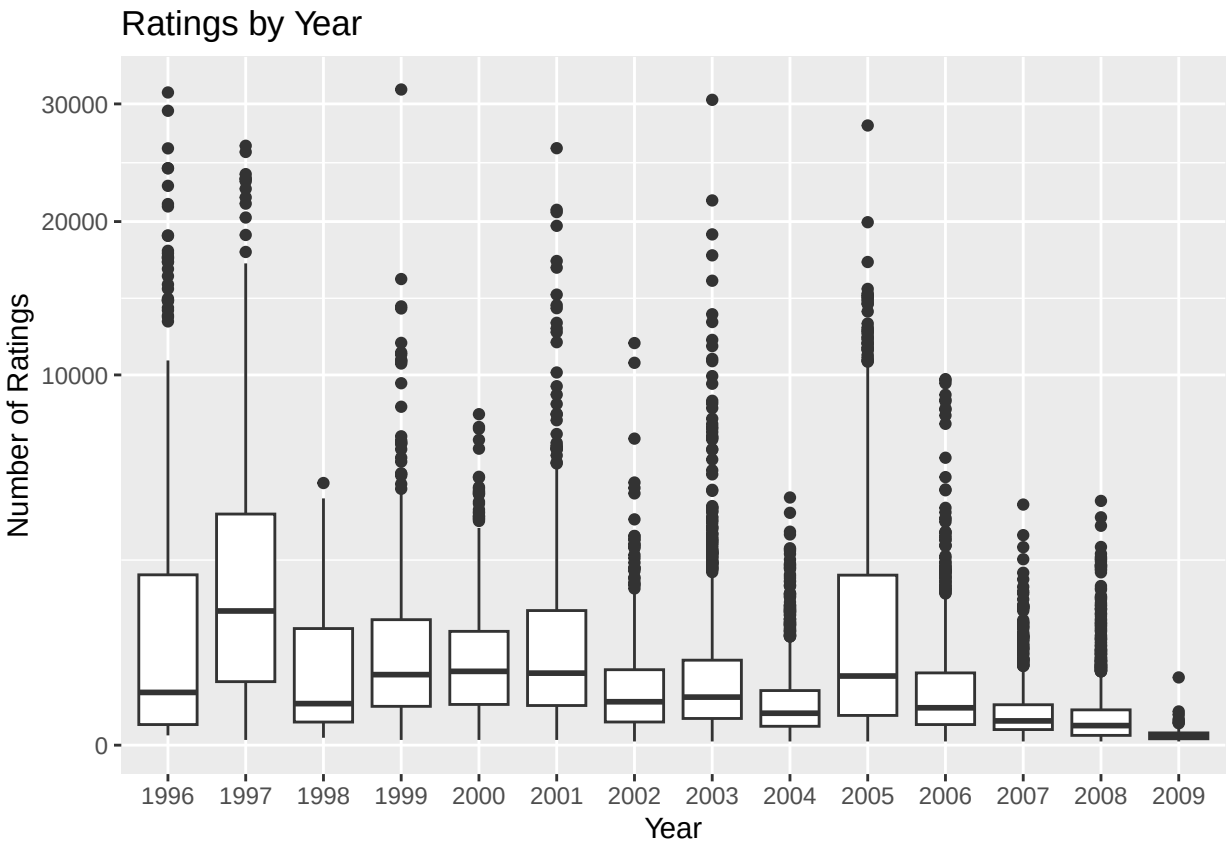
```
edx <- edx %>%
  mutate(year = year(as_datetime(timestamp, origin = "1970-01-01")))
edx %>%
  group_by(year) %>%
  summarize(count = n())
```

```
## # A tibble: 15 x 2
##     year   count
##    <int>   <int>
##  1  1995       2
##  2  1996  942772
##  3  1997  414101
##  4  1998  181634
##  5  1999  709893
##  6  2000 1144349
##  7  2001  683355
##  8  2002  524959
##  9  2003  619938
## 10  2004  691429
## 11  2005 1059277
## 12  2006  689315
## 13  2007  629168
## 14  2008  696740
## 15  2009   13123
```

Let's create a plot to visualise the distribution of ratings per year.

```
edx %>% group_by(movieId) %>%
  summarize(n = n(), year = as.character(first(year))) %>%
```

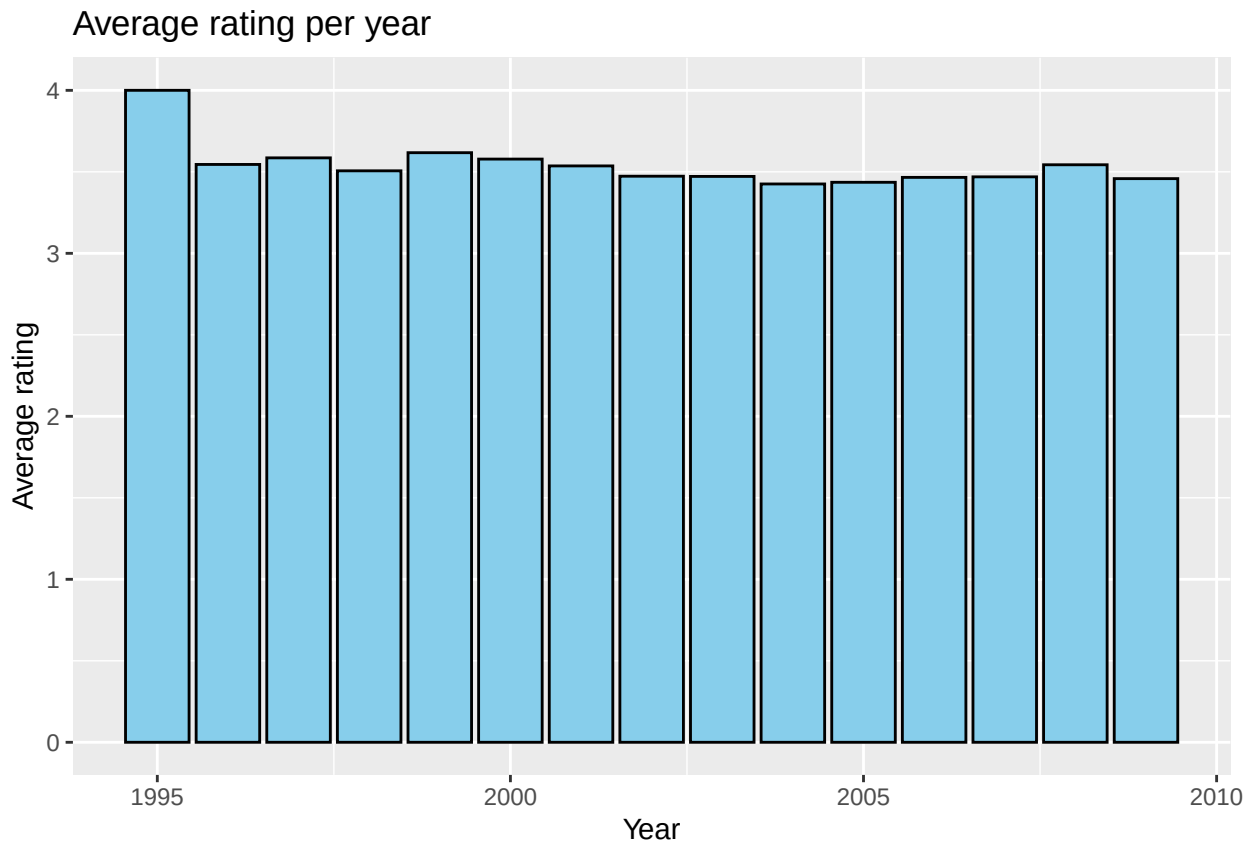
```
  ggplot(aes(x = year, y = n)) +
  geom_boxplot() +
  ggtitle("Ratings by Year") +
  scale_y_sqrt() +
  xlab("Year") +
  ylab("Number of Ratings")
```



We see that newer movies have fewer ratings, as they haven't had sufficient time to accumulate ratings.

Let's create a plot to visualise the average distribution of ratings for each year.

```
edx %>%
  group_by(year) %>%
  summarise(avg = mean(rating)) %>%
  ggplot(aes(x = year, y = avg)) +
  geom_bar(stat = "identity", fill = "skyblue", colour = "black") +
  ggtitle("Average rating per year") +
  xlab("Year") +
  ylab("Average rating")
```



Pretty uniform ratings average each year (the anomaly being 1995, which only has 2 ratings). We don't need to be concerned that newer movies with fewer ratings are rated more poorly.

Now let's take a look at genres.

```
edx %>% group_by(genres) %>%
  summarise(n())
```

```
## # A tibble: 797 x 2
##    genres                                          `n()`
##    <chr>                                           <int>
##  1 (no genres listed)                                  7
##  2 Action                                          24482
##  3 Action|Adventure                                68688
##  4 Action|Adventure|Animation|Children|Comedy        7467
##  5 Action|Adventure|Animation|Children|Comedy|Fantasy  187
##  6 Action|Adventure|Animation|Children|Comedy|IMAX     66
##  7 Action|Adventure|Animation|Children|Comedy|Sci-Fi   600
##  8 Action|Adventure|Animation|Children|Fantasy        737
##  9 Action|Adventure|Animation|Children|Sci-Fi          50
## 10 Action|Adventure|Animation|Comedy|Drama           1902
## # i 787 more rows
```

Many (or most) movies are listed as having multiple genres. Let's look at the spread of individual genres.

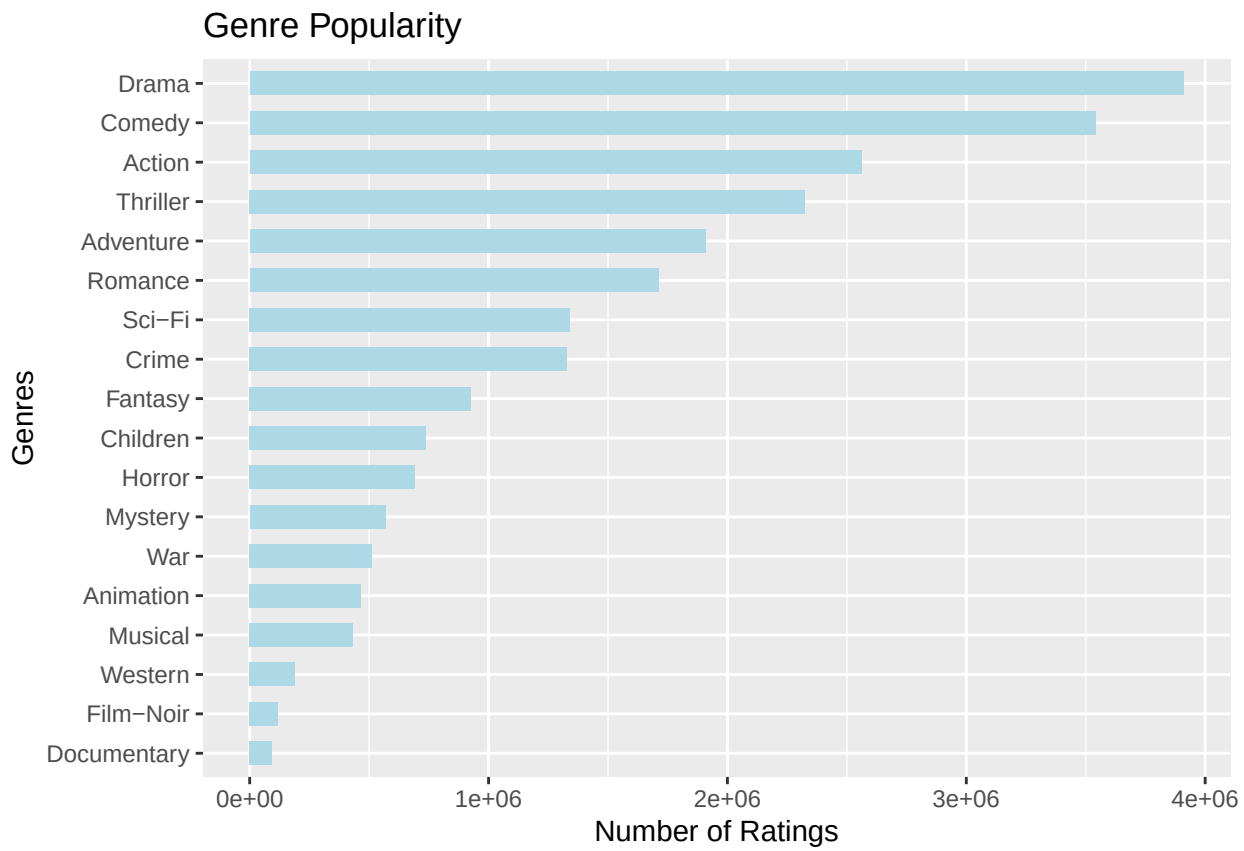
```
genres <- c("Action", "Adventure", "Animation",
            "Children", "Comedy", "Crime",
            "Documentary", "Drama", "Fantasy",
            "Film-Noir", "Horror", "Musical",
            "Mystery", "Romance", "Sci-Fi",
            "Thriller", "War", "Western")

genres_count <- data.frame(Genres = genres, Count = sapply(genres, function(x){
    sum(str_detect(edx$genres, x))
  })
)
genres_count %>% arrange(desc(Count))
```

```
##                  Genres   Count
## Drama             Drama 3910127
## Comedy           Comedy 3540930
## Action           Action 2560545
## Thriller       Thriller 2325899
## Adventure     Adventure 1908892
## Romance         Romance 1712100
## Sci-Fi           Sci-Fi 1341183
## Crime             Crime 1327715
## Fantasy         Fantasy  925637
## Children       Children  737994
## Horror           Horror  691485
## Mystery         Mystery  568332
## War                 War  511147
## Animation     Animation  467168
## Musical         Musical  433080
## Western         Western  189394
## Film-Noir     Film-Noir  118541
## Documentary Documentary   93066
```

We see that the most popular genre has almost 4 million ratings while the least popular has under 100,000 ratings.

Let's create a plot to visualise genre popularity.

```
genres_count %>%
  ggplot(aes(x = Count, y = reorder(Genres, Count))) +
  geom_bar(stat = "identity", width = 0.6, fill = "lightblue") +
  ggtitle("Genre Popularity") +
  xlab("Number of Ratings") +
  ylab("Genres")
```



We can see that drama and comedy are among the most popular genres, and film-noir and documentary are among the least popular.

Let's look at the average rating for each genre.

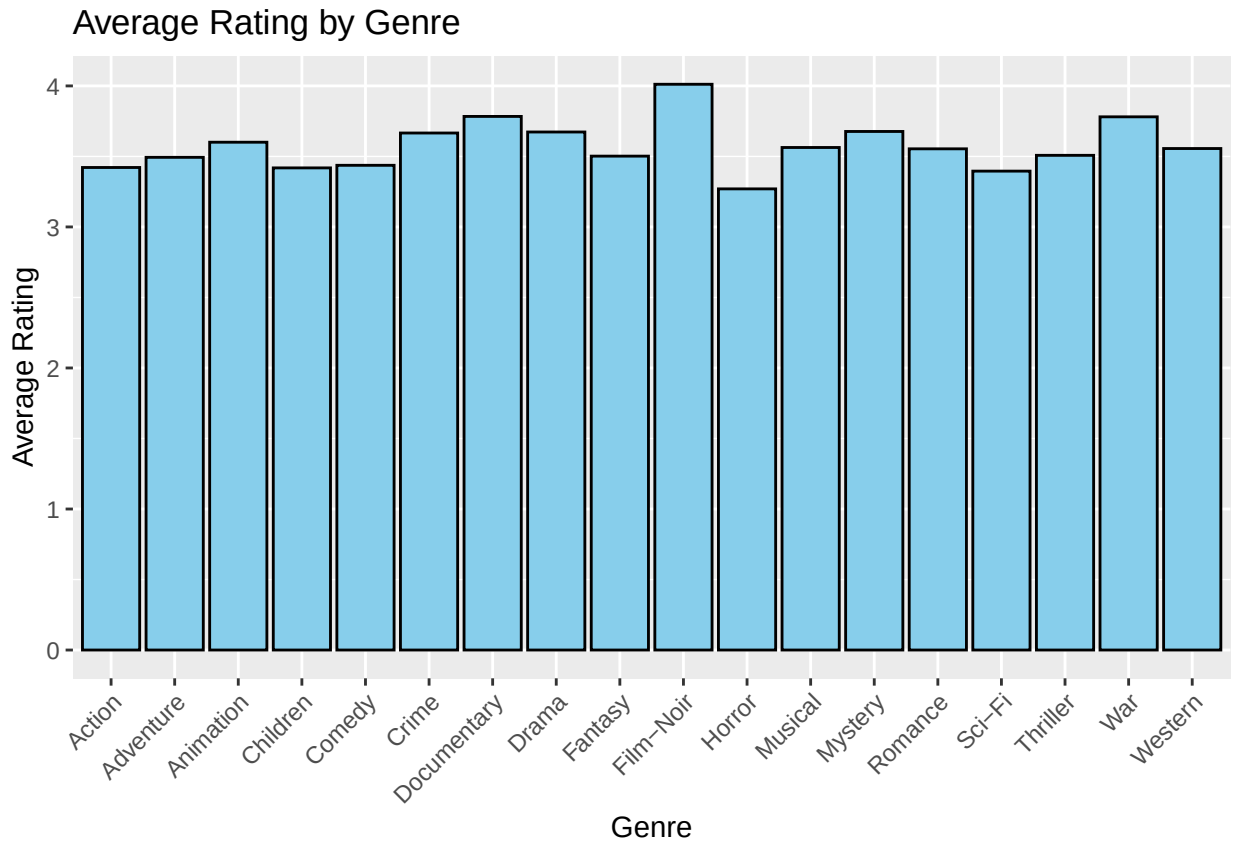
```
genres_rating <- data.frame(Genres = genres, Rating = sapply(genres, function(x){
  mean(edx %>%
    filter(str_detect(genres, x)) %>%
    pull(rating))
  })
)
genres_rating
```

```
##                  Genres   Rating
## Action           Action 3.421405
## Adventure     Adventure 3.493544
## Animation     Animation 3.600644
## Children       Children 3.418715
## Comedy           Comedy 3.436908
## Crime             Crime 3.665925
## Documentary Documentary 3.783487
## Drama             Drama 3.673131
## Fantasy         Fantasy 3.501946
## Film-Noir     Film-Noir 4.011625
## Horror           Horror 3.269815
## Musical         Musical 3.563305
```

```
## Mystery          Mystery 3.677001
## Romance          Romance 3.553813
## Sci-Fi            Sci-Fi 3.395743
## Thriller        Thriller 3.507676
## War                  War 3.780813
## Western          Western 3.555918
```

And to visualise that with a plot.

```
genres_rating %>% ggplot(aes(x = Genres, y = (Rating))) +
  geom_bar(stat = "identity", fill = "skyblue", color = "black") +
  ggtitle("Average Rating by Genre") +
  xlab("Genre") +
  ylab("Average Rating") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



We see that there are some differences in average ratings for different genres.

Model building

Now that we have thoroughly inspected the validation dataset, it is time to begin building our model. We will use a content-based filtering strategy accounting for user bias, movie bias, and genre bias. We will then apply regularisation to the model to see if it further minimises RMSE. Finally, we will evaluate a matrix factorisation model using the recosystem package.

Let's begin by creating our training and test sets. The test set will be 10% of edx data.

```r
set.seed(123) # Set seed for replicability
test_index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
test_set <- edx[test_index, ]
train_set <- edx[-test_index, ]

# To make sure userId and movieId in test set are also in training set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Next, let's load the rmse function from the Metrics package.

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

```
## Loading required package: Metrics
```

```
## Warning: package 'Metrics' was built under R version 4.3.3
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
library(Metrics)
```

With that, let's dive in and build our first model! Here's the first model: average of all ratings (mu).

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

```
## [1] 3.512461
```

```r
rmse_base <- rmse(test_set$rating, mu)
rmse_results <- tibble(method = "Mean baseline", rmse = rmse_base)
rmse_results
```

```
## # A tibble: 1 x 2
##   method         rmse
##   <chr>         <dbl>
## 1 Mean baseline  1.06
```

With the mean baseline model, we achieve an RMSE of 1.06061. Let's see if we can improve on that.

Next, we know that certain movies are generally more highly rated than other movies. Thus, let us factor in movie bias (b_i) into our model.

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

pred_i <- mu + test_set %>%
  left_join(b_i, by="movieId") %>%
  pull(b_i)

rmse_i <- rmse(pred_i, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 rmse = rmse_i))
rmse_results
```

```
## # A tibble: 2 x 2
##   method              rmse
##   <chr>              <dbl>
## 1 Mean baseline      1.06
## 2 Movie Effect Model 0.943
```

With the movie effect model, we effectively lowered the RMSE to 0.94321.

We also know that users differ in how they rate movies - some tend to rate conservatively, others tend to give very high ratings. Thus, let us factor in user bias (b_u) into our model.

```
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

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

rmse_i_u <- rmse(pred_i_u, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effect Model",
                                 rmse = rmse_i_u))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                     rmse
##   <chr>                     <dbl>
## 1 Mean baseline             1.06
## 2 Movie Effect Model        0.943
## 3 Movie + User Effect Model 0.865
```

With our movie + user effect model, we managed to lower the RMSE to 0.86504.

From our exploratory analyses, we saw that there were some differences in ratings among individual genres. Some genres were more well loved than others. Let us attempt to factor in a genre bias (b_g) into our model.

```r
# We begin by teasing out the average rating for each genre
train_genres <- data.frame(genres = genres, rating = sapply(genres, function(x){
  mean(train_set %>%
         filter(str_detect(genres, x)) %>%
         pull(rating))
})
)
train_genres
```

```
##                   genres   rating
## Action            Action 3.421344
## Adventure     Adventure 3.493614
## Animation     Animation 3.600428
## Children       Children 3.418530
## Comedy           Comedy 3.436992
## Crime             Crime 3.665573
## Documentary Documentary 3.782728
## Drama             Drama 3.673111
## Fantasy         Fantasy 3.501944
## Film-Noir     Film-Noir 4.012701
## Horror           Horror 3.269855
## Musical         Musical 3.563176
## Mystery         Mystery 3.677135
## Romance         Romance 3.554005
## Sci-Fi           Sci-Fi 3.395472
## Thriller       Thriller 3.507529
## War                 War 3.780411
## Western         Western 3.555762
```

```r
# We then subtract the average rating for each genre by the average of all ratings to obtain specific g
b_g <- data.frame(genres = genres, b_g = train_genres$rating - mu)

# Now, we build it into our model
test_set_genres <- test_set %>%
  mutate(genres_list = strsplit(genres, "\\|")) %>%
  unnest(genres_list) %>%
  left_join(b_g, by = c("genres_list" = "genres")) %>%
  group_by(userId, movieId) %>%
  summarise(b_g = sum(b_g, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'userId'. You can override using the
## `.groups` argument.
```

```r
pred_i_u_g <- test_set %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(test_set_genres, by=c("userId" = "userId", "movieId" = "movieId"),
            relationship = "many-to-many") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

rmse_i_u_g <- rmse(pred_i_u_g, test_set$rating)
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre Effect Model",
                                 rmse = rmse_i_u_g))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                              rmse
##   <chr>                              <dbl>
## 1 Mean baseline                      1.06
## 2 Movie Effect Model                 0.943
## 3 Movie + User Effect Model          0.865
## 4 Movie + User + Genre Effect Model 0.887
```

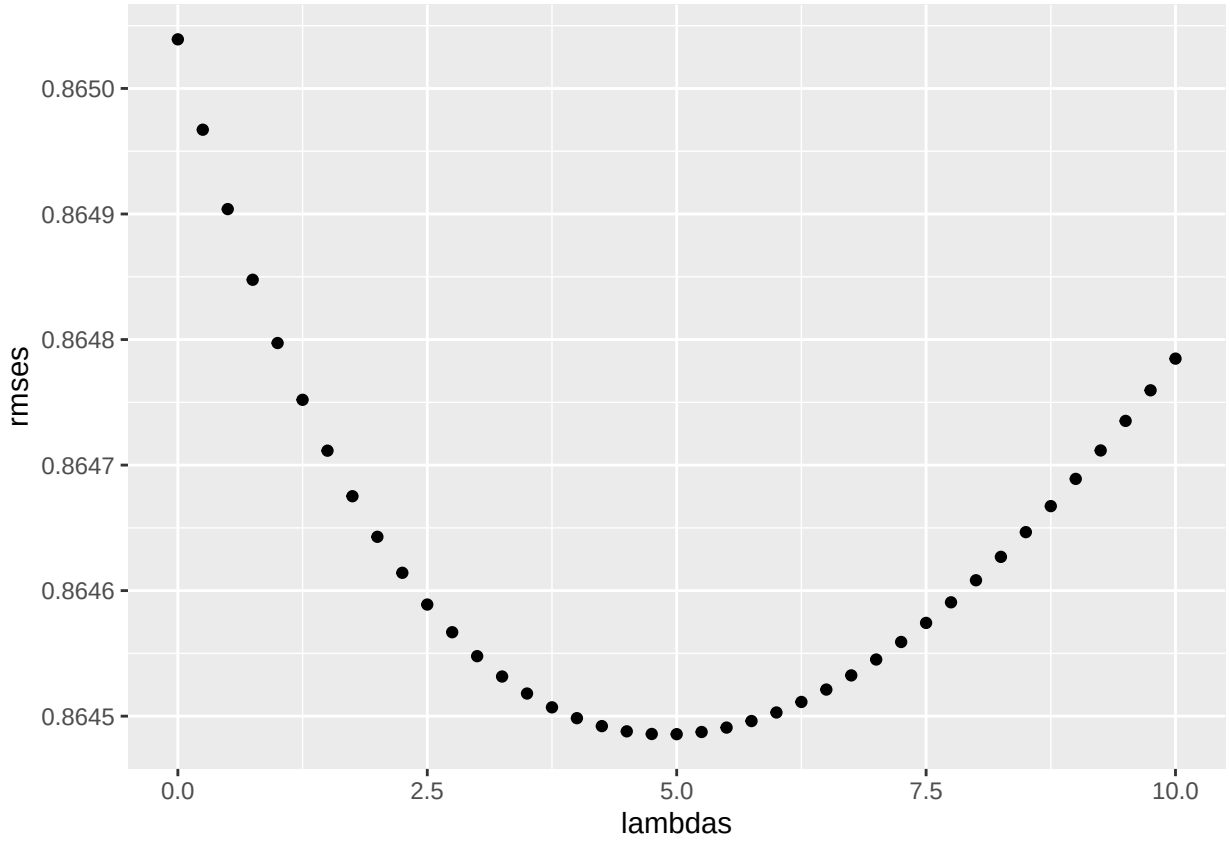
With the movie, + user + genre effect model, the RMSE actually increased slightly to 0.88702. Somewhat disheartening given our efforts to include a genre bias. As including a genre bias is not helpful, we will continue building our model without it.

Next, recall that our top 10 and bottom 10 movies in terms of ratings are mostly obscure movies that were rated by very few users. We should not trust these ratings in making useful predictions. To counter this, we can use regularisation to penalise large estimates resulting from small sample sizes.

Let's proceed to regularise our movie + user effect model.

```
# Regularisation involves adding a tuning parameter (lambda) that penalises large estimates
# To find the optimal lambda, let's use cross validation
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

  b_i_reg <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i_reg = sum(rating - mu)/(n()+l))

  b_u_reg <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarise(b_u_reg = sum(rating - b_i_reg - mu)/(n()+l))

  pred_reg <- test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(pred = mu + b_i_reg + b_u_reg) %>%
    pull(pred)

  return(rmse(pred_reg, test_set$rating))
})

#to observe and obtain the optimal lambda
qplot(lambdas, rmses)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




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

# Having established the optimal lambda of 5, let's build our model
b_i_reg <- train_set %>%
group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

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

pred_reg <- test_set %>%
  left_join(b_i_reg, by='movieId') %>%
  left_join(b_u_reg, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

rmse_reg <- rmse(pred_reg, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularised Movie + User Effect Model",
                                 rmse = rmse_reg))
rmse_results
```

```
## # A tibble: 5 x 2
```

```
##   method                                  rmse
##   <chr>                                  <dbl>
## 1 Mean baseline                          1.06
## 2 Movie Effect Model                     0.943
## 3 Movie + User Effect Model              0.865
## 4 Movie + User + Genre Effect Model      0.887
## 5 Regularised Movie + User Effect Model 0.864
```

With regularisation of the movie + user effect model, we obtain an RMSE of 0.86449, which represents a slight improvement over our unregularised move + user effect model. However, this may not allow us to achieve an RMSE < 0.86490 on our final holdout set.

Let us try our hand at building a matrix factorisation model using the recosystem package. Matrix factorisation decomposes a user-item matrix into two low-ranked matrices (in this case, user and movie), that can predict a user's movie preference.

```
# To begin, install and load the recosystem package
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: recosystem
```

```
## Warning: package 'recosystem' was built under R version 4.3.3
```

```
library(recosystem)

# Next, we convert the training and test sets into recosystem input format
set.seed(123)
train_recosystem <- with(train_set,
                         data_memory(user_index = userId,
                                     item_index = movieId,
                                     rating = rating))
test_recosystem <- with(test_set,
                        data_memory(user_index = userId,
                                    item_index = movieId,
                                    rating = rating))

# Next we create and tune the model using various parameters
reco_system <- Reco()
tuning <- reco_system$tune(train_recosystem,
                           opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       costp_l1 = 0,
                                       costq_l1 = 0,
                                       nthread  = 4,
                                       niter = 10))

# We then train the model with the best tuning parameters
reco_system$train(train_recosystem,
                  opts = c(tuning$min, nthread = 4,niter = 100))
```

```
## iter      tr_rmse          obj
```

```
##    0       0.9821   1.1035e+07
##    1       0.8751   8.9871e+06
##    2       0.8430   8.3427e+06
##    3       0.8216   7.9680e+06
##    4       0.8049   7.6940e+06
##    5       0.7923   7.5067e+06
##    6       0.7819   7.3577e+06
##    7       0.7733   7.2426e+06
##    8       0.7659   7.1449e+06
##    9       0.7594   7.0650e+06
##   10       0.7537   6.9983e+06
##   11       0.7486   6.9399e+06
##   12       0.7441   6.8895e+06
##   13       0.7399   6.8474e+06
##   14       0.7361   6.8062e+06
##   15       0.7327   6.7737e+06
##   16       0.7295   6.7419e+06
##   17       0.7266   6.7134e+06
##   18       0.7239   6.6874e+06
##   19       0.7214   6.6660e+06
##   20       0.7191   6.6441e+06
##   21       0.7170   6.6251e+06
##   22       0.7150   6.6076e+06
##   23       0.7130   6.5898e+06
##   24       0.7113   6.5745e+06
##   25       0.7098   6.5624e+06
##   26       0.7082   6.5503e+06
##   27       0.7068   6.5370e+06
##   28       0.7054   6.5273e+06
##   29       0.7042   6.5158e+06
##   30       0.7030   6.5064e+06
##   31       0.7018   6.4968e+06
##   32       0.7008   6.4875e+06
##   33       0.6997   6.4796e+06
##   34       0.6987   6.4730e+06
##   35       0.6978   6.4648e+06
##   36       0.6969   6.4586e+06
##   37       0.6961   6.4522e+06
##   38       0.6953   6.4456e+06
##   39       0.6945   6.4408e+06
##   40       0.6938   6.4356e+06
##   41       0.6931   6.4299e+06
##   42       0.6923   6.4241e+06
##   43       0.6918   6.4203e+06
##   44       0.6911   6.4149e+06
##   45       0.6905   6.4118e+06
##   46       0.6899   6.4073e+06
##   47       0.6894   6.4034e+06
##   48       0.6889   6.4001e+06
##   49       0.6884   6.3954e+06
##   50       0.6878   6.3923e+06
##   51       0.6874   6.3893e+06
##   52       0.6869   6.3864e+06
##   53       0.6865   6.3834e+06
```

```
##   54       0.6860   6.3794e+06
##   55       0.6856   6.3777e+06
##   56       0.6851   6.3744e+06
##   57       0.6848   6.3720e+06
##   58       0.6844   6.3687e+06
##   59       0.6840   6.3663e+06
##   60       0.6836   6.3639e+06
##   61       0.6832   6.3609e+06
##   62       0.6829   6.3587e+06
##   63       0.6826   6.3562e+06
##   64       0.6823   6.3544e+06
##   65       0.6820   6.3537e+06
##   66       0.6816   6.3503e+06
##   67       0.6813   6.3486e+06
##   68       0.6810   6.3471e+06
##   69       0.6807   6.3453e+06
##   70       0.6804   6.3437e+06
##   71       0.6801   6.3401e+06
##   72       0.6799   6.3399e+06
##   73       0.6796   6.3366e+06
##   74       0.6793   6.3357e+06
##   75       0.6791   6.3350e+06
##   76       0.6789   6.3336e+06
##   77       0.6786   6.3309e+06
##   78       0.6783   6.3303e+06
##   79       0.6781   6.3284e+06
##   80       0.6778   6.3262e+06
##   81       0.6776   6.3251e+06
##   82       0.6774   6.3238e+06
##   83       0.6772   6.3223e+06
##   84       0.6770   6.3216e+06
##   85       0.6768   6.3193e+06
##   86       0.6766   6.3193e+06
##   87       0.6763   6.3180e+06
##   88       0.6762   6.3179e+06
##   89       0.6759   6.3151e+06
##   90       0.6757   6.3140e+06
##   91       0.6755   6.3127e+06
##   92       0.6754   6.3125e+06
##   93       0.6752   6.3121e+06
##   94       0.6750   6.3105e+06
##   95       0.6748   6.3086e+06
##   96       0.6746   6.3066e+06
##   97       0.6744   6.3063e+06
##   98       0.6743   6.3058e+06
##   99       0.6740   6.3049e+06
```

```
pred_mf <-  reco_system$predict(test_recosystem, out_memory())
rmse_mf <- rmse(pred_mf, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Matrix factorisation",
                                 rmse = rmse_mf))
rmse_results
```

```
## # A tibble: 6 x 2
##   method                                rmse
##   <chr>                                <dbl>
## 1 Mean baseline                        1.06
## 2 Movie Effect Model                   0.943
## 3 Movie + User Effect Model            0.865
## 4 Movie + User + Genre Effect Model    0.887
## 5 Regularised Movie + User Effect Model 0.864
## 6 Matrix factorisation                 0.788
```

With a matrix factorisation model using recosystem, we obtain an RMSE of 0.78743, which blows our regularised movie + user effect model away.

Evaluation of our final model

The RMSE of our matrix factorisation model is pretty satisfactory. Let us now evaluate the model on our final holdout set.

```
set.seed(123)
final_recosystem <- with(final_holdout_test,
                         data_memory(user_index = userId,
                                     item_index = movieId))
pred_final <-  reco_system$predict(final_recosystem, out_memory())
rmse_final <- rmse(pred_final, final_holdout_test$rating)
rmse_final
```

```
## [1] 0.7877562
```

With the matrix factorisation model, we obtain an RMSE of 0.78771 on our final holdout set.

Conclusion

In this project, we attempt to build a movie recommendation system using the 10M version of the MovieLens dataset. The working goal was to create a model that produces an RMSE < 0.86490 so as to achieve a high score for our project.

We began by conducting some exploratory analyses to understand the dataset better. We saw that there were almost 7 times as many unique users as unique movies, and that users tend to rate high and using whole numbers. We also saw that most users rated less than 100 movies, and that most movies received between 10 and 1,000 ratings. Looking at the top and bottom rated movies, we found that most movies in the list were obscure movies. We concluded that the year of the movie likely did not affect it's ratings, but thought that the genre(s) might.

With that, we built our first model using the mean baseline rating, upon which we factored movie, user, and genre biases. We found that genre was not a useful predictor, and continued without it. We then applied regularisation to our movie + user effect model bringing our RMSE just below the end goal. Finally, we tested out a matrix factorisation model, built using the recosystem package. With that we obtained a very satisfactory RMSE score on our validation set, and ultimately, on our final holdout set.

I hope this exercise helped the reader glimpse into my thought process in building my first movie recommendation model. It was fun, frustrating, and everything in between!