MovieLens Project

Michael Twiston Davies

07/12/2020

Introduction

History

In 2006 Netflix created a $1m competition challenging data scientists to create a movie recommendation system better than their own "Cinematch". In order to qualify for the prize the challengers had to be 10% more accurate than Cinematch's predictions. In June 2009 it was beaten by multinational team called BellKor's Pragmatic Chaos. The winning model used a combination of Normalisation of Global Effects, Neighbourhood Models, Matrix Factorisation and Regression.

The dataset

The data set provided contains Userid, movieid, rating, timestamp, title and genres. Though we will note later on that in order to perform our complete analysis we will have to convert the timestamp into a more useful format and also will split the release year which currently is within the title.

We will visualise the data within the method section.

Aim

This analysis aims to create a machine learning algorithm using techniques learnt in the Harvardx Data Science course. We will be utilising the MovieLens dataset provided to us and will go through a few different techniques to try and improve upon this model. The final model will then be validated against the validation data set. RMSE will be used to evaluate how close our predictions are to the true values in the validation set (the final hold-out test set). We intend to gain a RMSE < 0.86490 with our final validation.

Method

Method - Overview

We will use the "edx" data set created from the "Movie 10M data set" to develop our algorithm and the "validation" set to predict movie ratings as if they were unknown. RMSE will be used to evaluate how close the predictions are to the true values in the validation set (the final hold-out test set).

Firstly we will split the "edx" data set into testing and training data sets.

We will apply the following methods, the first of which were shown on the course, though we will build on these as we go through:

- Model 1 - Just the average
- Model 2 - Movie Effect Model
- Model 3 - Movie + User Effects Model
- Model 4 - Movie + User + Genre Effects Model
- Model 5 - Movie + User + Genre + Time Effects Model
- Model 6 - Regularized Movie Effect Model
- Model 7 - Regularized Movie + User Effect Model
- Model 8 - Regularized Movie + User + Genre Effect Model
- Model 9 - Matrix Factorisation

Regularisation will apply less weight to ratings with smaller n. For example some movies may have only been rated a handful of times and achieved a 5 star rating, this would likely decrease upon subsequent ratings therefore regularisation accounts for this.

Our final model utilises matrix factorisation which decomposes the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.

Data Preparation

As mentioned above the below code will firstly pull the data, wrangle it into a useful format and split out the validation set.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

# packages to be used
library(tidyverse) # a multitude of useful functions
library(caret) # for prediction formulas
library(data.table)
library(lubridate) # easy date/time manipulation
library(recosystem) # Matrix factorisation

# Suppress summarise info
library(dplyr, warn.conflicts = FALSE) # use to supress grouping warning messages
options(dplyr.summarise.inform = FALSE) # use to supress grouping warning messages

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

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

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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

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

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

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

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

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

# Remove items no longer needed.
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The below code will then make the Timestamp more useful by creating Date and Weekday columns from it whilst also extracting the release year from the title.

```
############################################################
# Timestamp/title split and Partition Edx dataset into test and training datasets
############################################################

# Though firstly lets just add a few extra columns now which we will be using when adjusting for variat
edx <- edx %>%
  # The below creates Weekday/Date from Timestamp
  mutate( Date = date(as.POSIXct(timestamp, origin='1970-01-01')) , Weekday = weekdays(Date)) %>%
  #The below separates the title into title/release year.
  extract(title, c("title_1", "releaseyear"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F) %>%
  mutate(releaseyear = if_else(str_length(releaseyear) > 4, as.integer(str_split(releaseyear, "-", simpl
  mutate(title = if_else(is.na(title_1), title, title_1)) %>%
  select(-title_1)  %>%
  # For genres with a blank for genre we will replace this with "no genre".
  mutate(genres = if_else(genres == "(no genre)", `is.na<-`(genres), genres)) %>%
  mutate( ReleasetoRatingTime = year(Date) - releaseyear)
```

```
# Partition Edx dataset into test and training datasets
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)

train_set <- edx[-test_index,]
test_set <- edx[test_index,]
rm(test_index)

# To make sure we don't include users and movies in the test set that do not appear in the training set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

The below is a function created to calculate the residual means squared error for a vector of ratings and their corresponding predictors.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

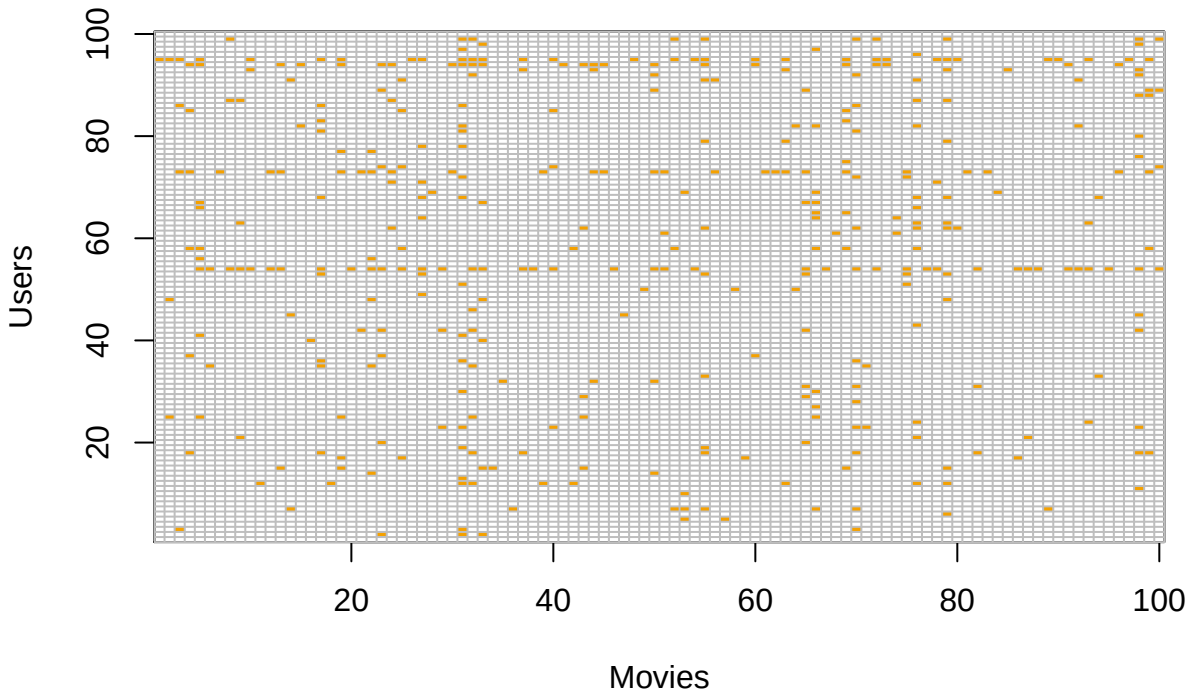
Method - Data exploration

To start us off lets just have a quick look at the first 6 lines in the data set.

```
##    userId movieId rating timestamp                 title releaseyear
## 1:      1     122      5 838985046              Boomerang        1992
## 2:      1     185      5 838983525               Net, The        1995
## 3:      1     292      5 838983421               Outbreak        1995
## 4:      1     316      5 838983392               Stargate        1994
## 5:      1     329      5 838983392 Star Trek: Generations        1994
## 6:      1     355      5 838984474       Flintstones, The        1994
##                            genres       Date Weekday ReleasetoRatingTime
## 1:                Comedy|Romance 1996-08-02  Friday                    4
## 2:          Action|Crime|Thriller 1996-08-02  Friday                    1
## 3:  Action|Drama|Sci-Fi|Thriller 1996-08-02  Friday                    1
## 4:       Action|Adventure|Sci-Fi 1996-08-02  Friday                    2
## 5: Action|Adventure|Drama|Sci-Fi 1996-08-02  Friday                    2
## 6:       Children|Comedy|Fantasy 1996-08-02  Friday                    2

## [1] "There are 69878 users and 10677 movies in the edx dataset"
```

If we take 100 userids and view their ratings we can see how sparsely populated the database is. Now we start to get an idea of the mountainous challenge we face in predicting the gaps.

```
OneHundredusers <- sample(unique(edx$userId), 100)

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



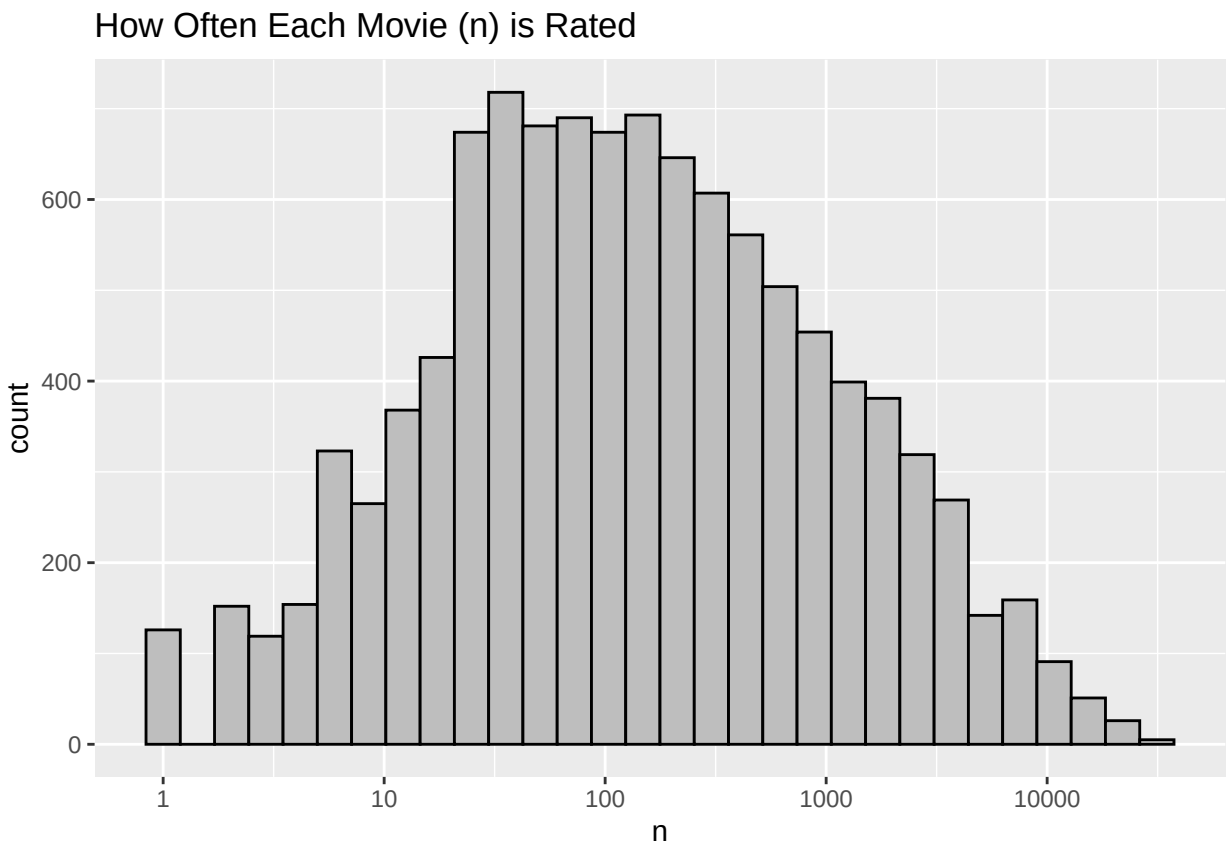

```
## integer(0)
```

We are attempting to predict the ratings for movies (i) based on the users (u) though they have rated different movies and have given different ratings.

Method - Data exploration - Movies

If we count the number of times each movie has been rated we can see that some are rated more than others.

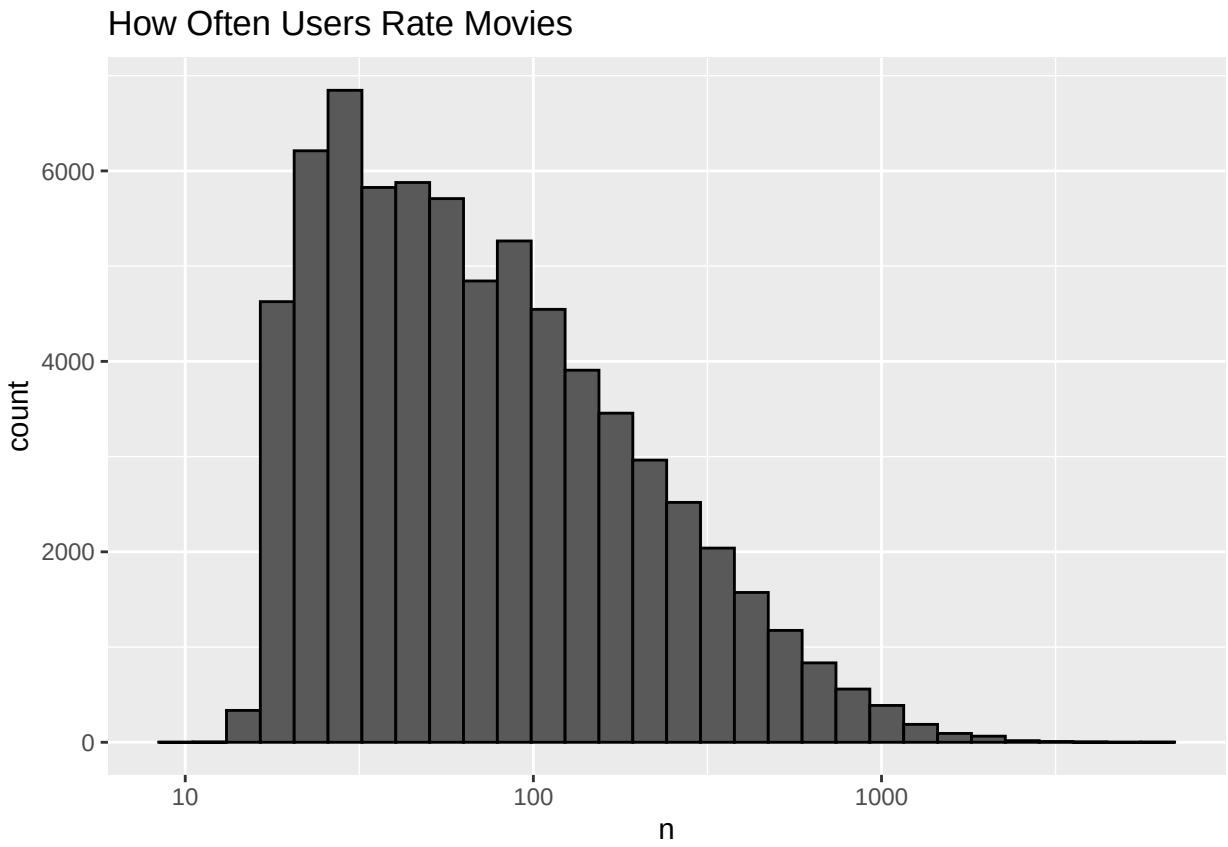
```
# Lets look at the distribution of the data. We can see that some movies are rated more than others.
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "gray") +
  scale_x_log10() +
  ggtitle("How Often Each Movie (n) is Rated")
```



Method - Data exploration - Users

Further to this we can see some users are more active than others when rating movies.

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



Method - Data exploration - Genres

The genres are collated as a string in each data row therefore currently the exploration of this data in this format is quite difficult. We will split these out for visual representation of the data though will not be doing so for the analysis.

We can see from the data that there are a large number of genres. This is actually due to most films having more than 1 film genre, therefore we are seeing the 797 combinations of films in the database.

```
length(unique(edx$genres))
```

```
## [1] 797
```

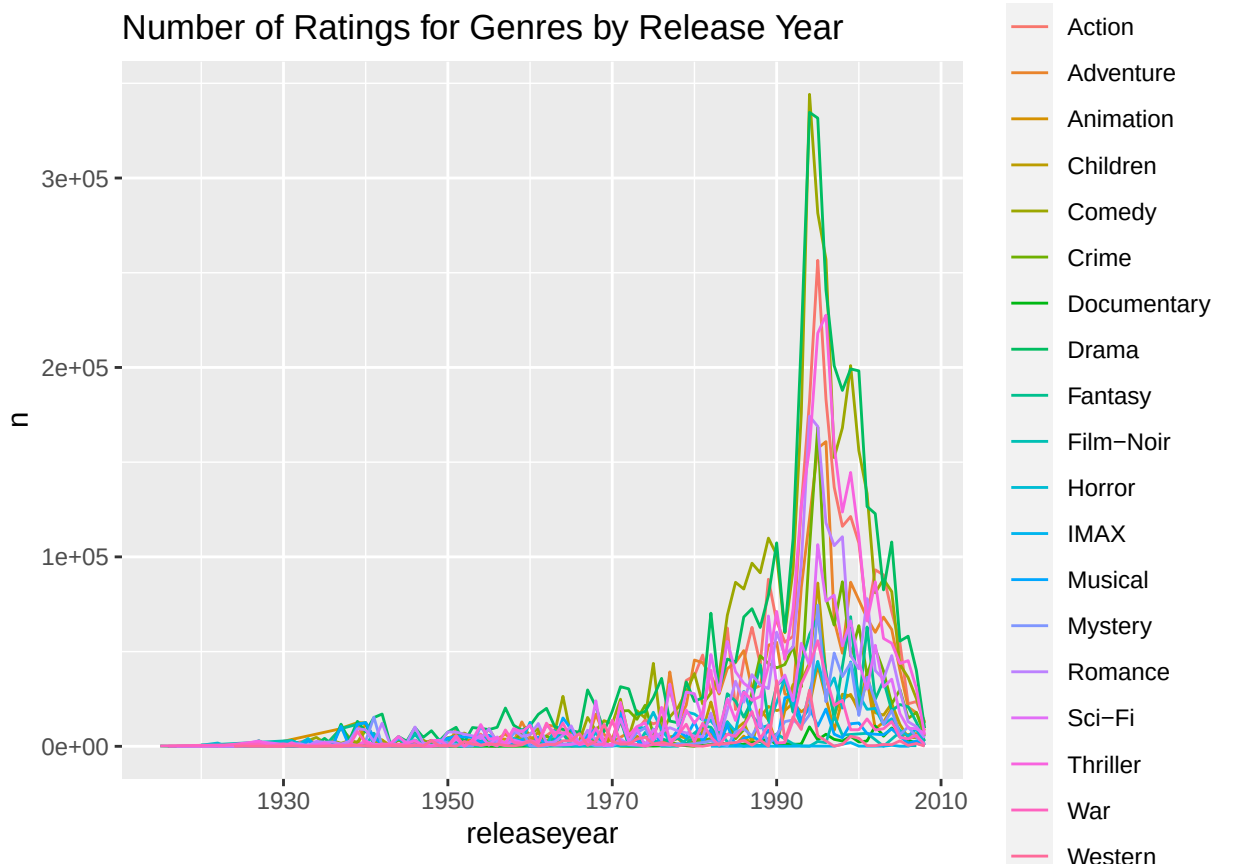
```
# Therefore we have to split the Genres (for which most films have more than 1) into separate lines in
genresep <- edx %>%
  separate_rows(genres,sep = "\\|")

# The below summarises the actual number of genres in the data.
genresexp <- genresep %>%
  group_by(genres) %>%
  summarise(n = n()) %>%
  arrange(desc(n))

knitr::kable(genresexp)
```

genres	n
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

```
# The below shows us that there are actually 19 genres (and 7 instances of no genres).
length(unique(genresexp$genres))
```

```
## [1] 20
```

```
# We can graph how popular each film genre is in each year with the below. This shows the number of rat
GenreRatingsbyYear <- genresep %>%
  na.omit() %>% # remove missing values
  select(movieId, releaseyear, genres) %>% # select columns
  mutate(genres = as.factor(genres)) %>% # genres in factors
  group_by(releaseyear, genres) %>% # group by release year and genre
  summarise(n = n())  # summarise by number of ratings

GenreRatingsbyYear %>%
  filter(genres != "(no genres listed)") %>%
  na.omit() %>%
  ggplot(aes(x = releaseyear, y = n)) +
  geom_line(aes(color=genres)) +
  ggtitle("Number of Ratings for Genres by Release Year")
```

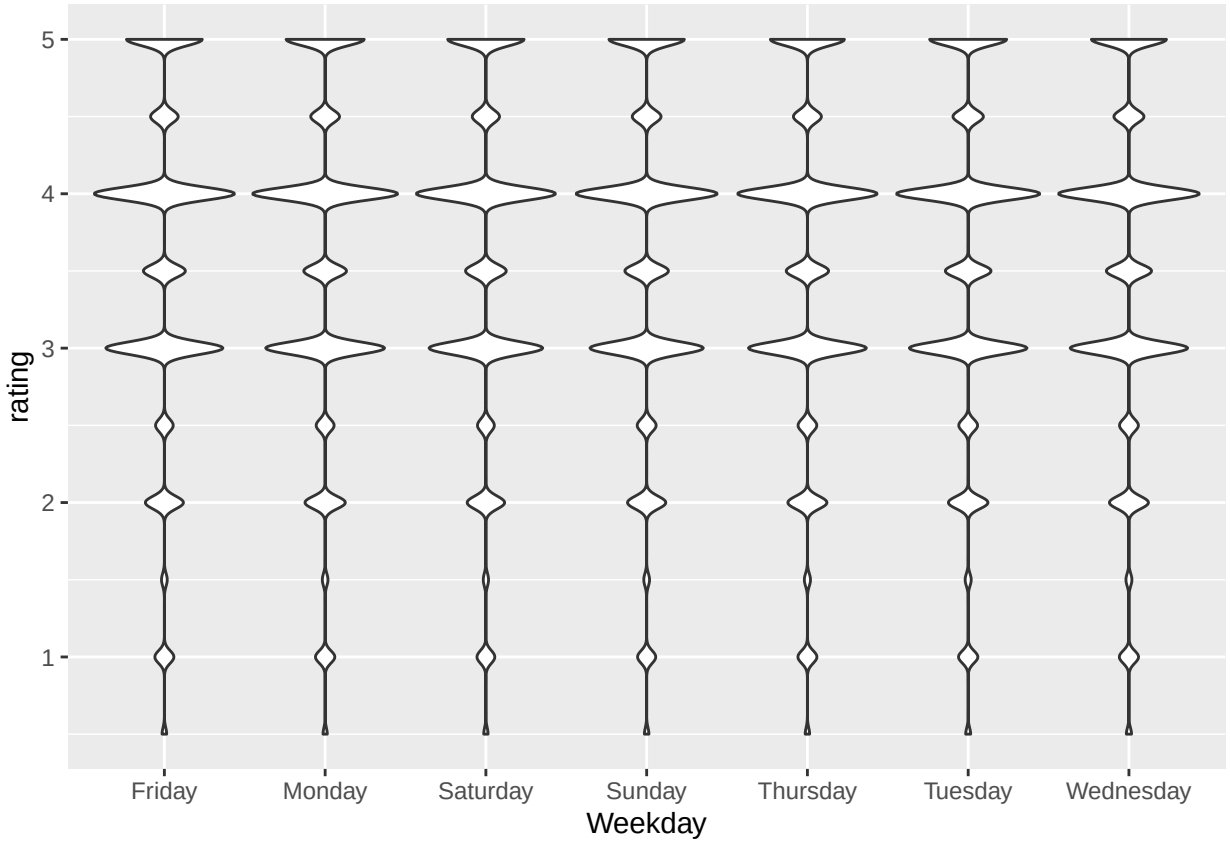


We would utlise the data split into genres with the 19 different outcomes rather than the 797 different combinations but this unfortunately leads to each rating being duplicated by the number of genres for each film. This will massively skew the data and therefore is not useful.

Method - Data exploration - Time

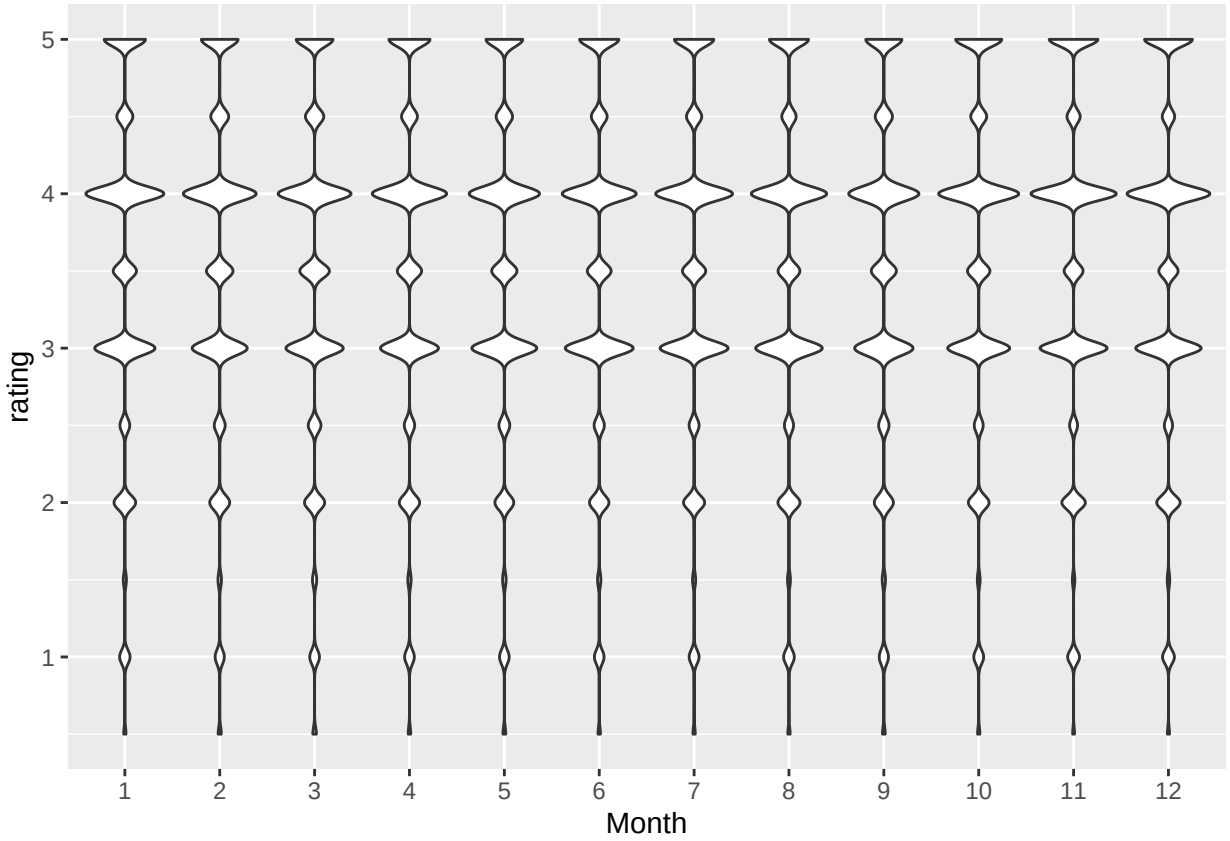
We will now look at the effect of time on ratings.

```
# Weekday
edx %>%
  ggplot(aes( x = Weekday , rating)) +
  geom_violin()
```



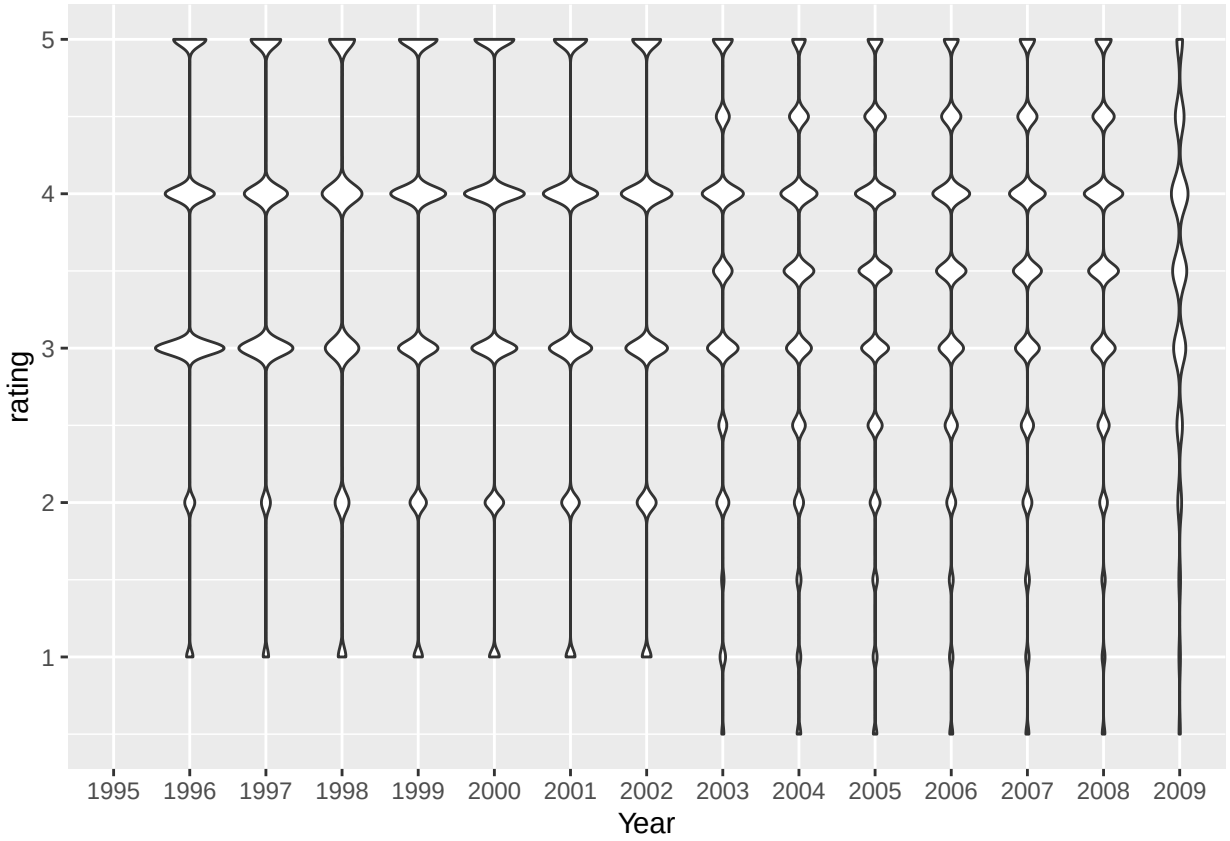
There does seem to be some variation here though it doesn't seem very over bearing.

```
# Month
edx %>%
  mutate(Year = year(Date) , Month = as.factor(month(Date))) %>%
  ggplot(aes( x = Month , rating)) +
  geom_violin()
```



There seems to be even less variation when we look at ratings by months. There may still be a seasonal effect but we will continue to investigate.

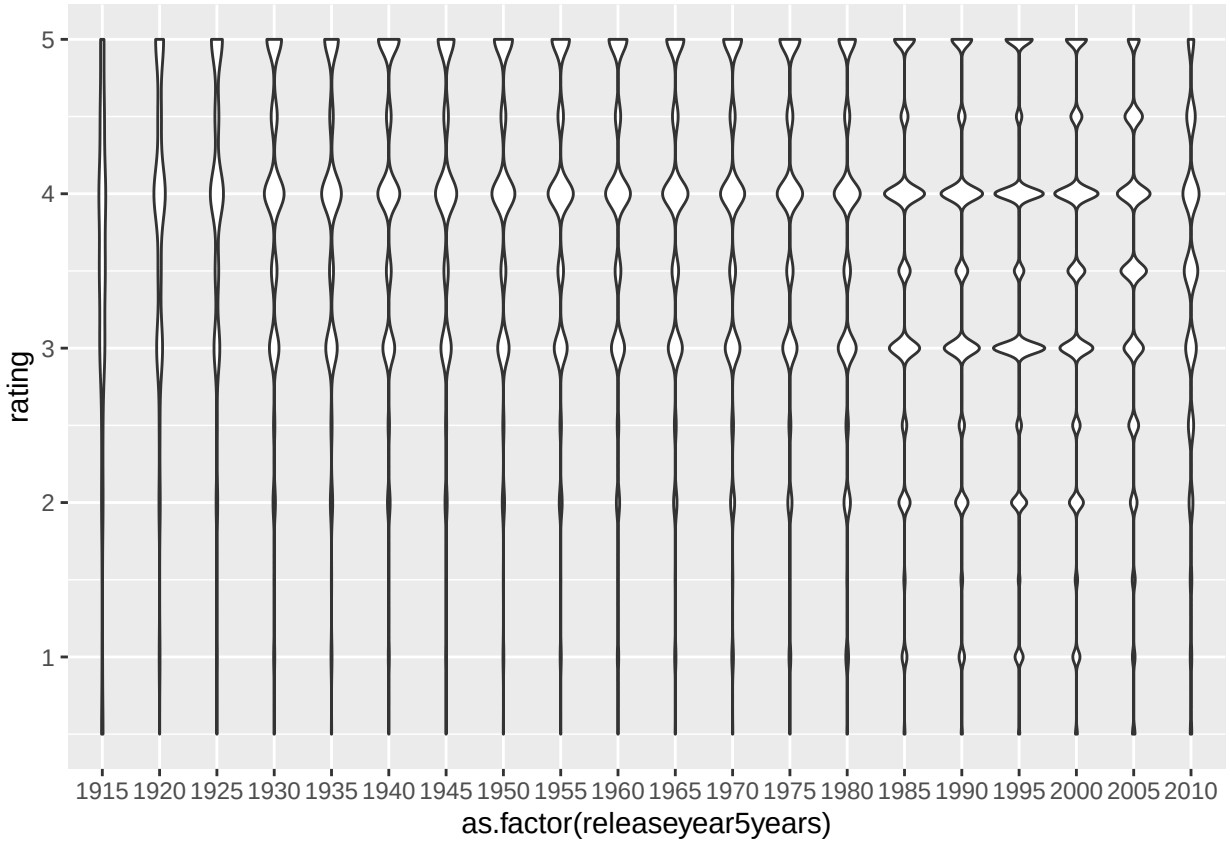
```
# Year
edx %>%
  mutate(Year = as.factor(year(Date))) %>%
  ggplot(aes( x = Year , rating)) +
  geom_violin()
```



It seems from this output that 0.5 ratings were only introduced from 2003. Therefore they could be some unintended bias here due to this but we will again continue to investigate the data.

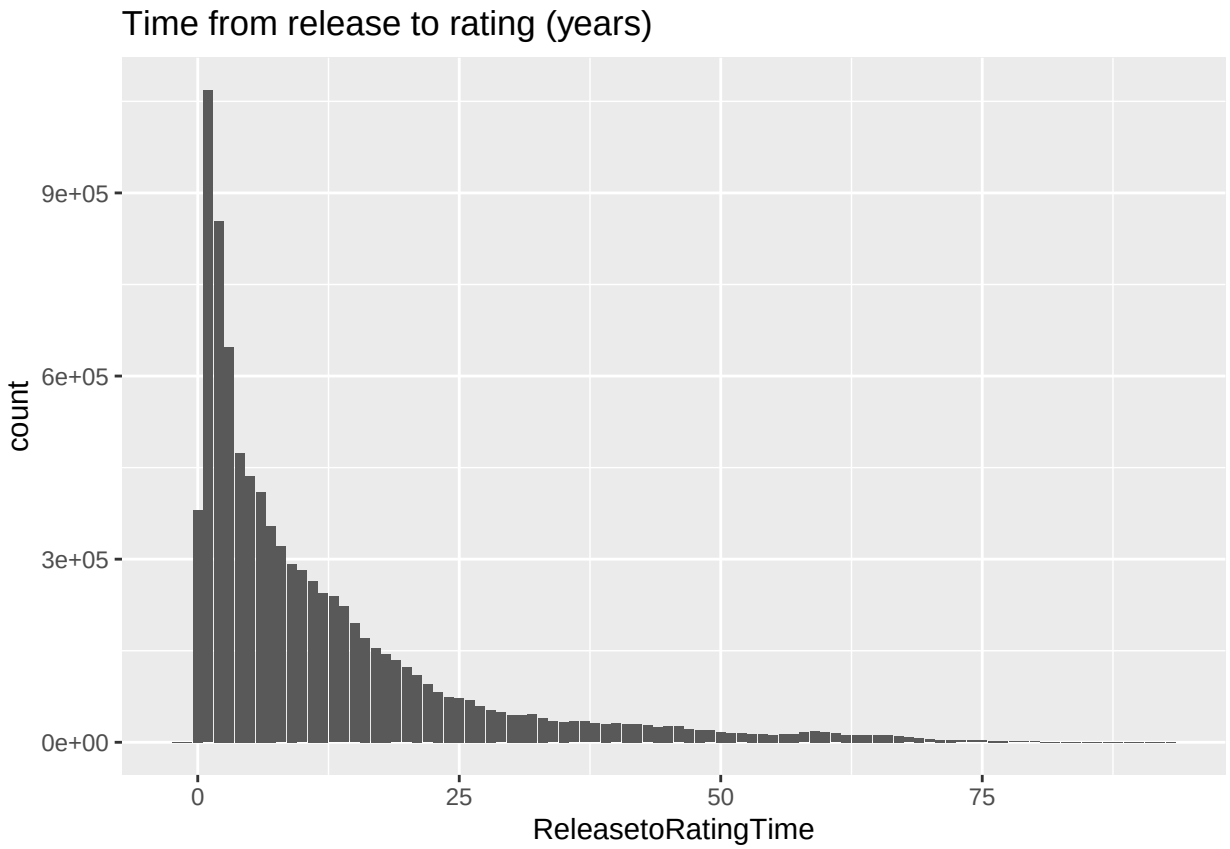
We will look at the time since release in the below code. In order to visualise this a little easier (due to overcrowding by number of years) we will group release years into every 5 years.

```
# Time since release

# Code to round to every 5th year.
mround <- function(x,base){
  base*round(x/base)
}

# Here we have rounded the release dates to every 5th year. This allows us to visualise the violin plot
edx %>%
  mutate(releaseyear5years = mround(releaseyear, 5)) %>%
  ggplot(aes( x = as.factor(releaseyear5years) , rating)) +
  geom_violin()
```

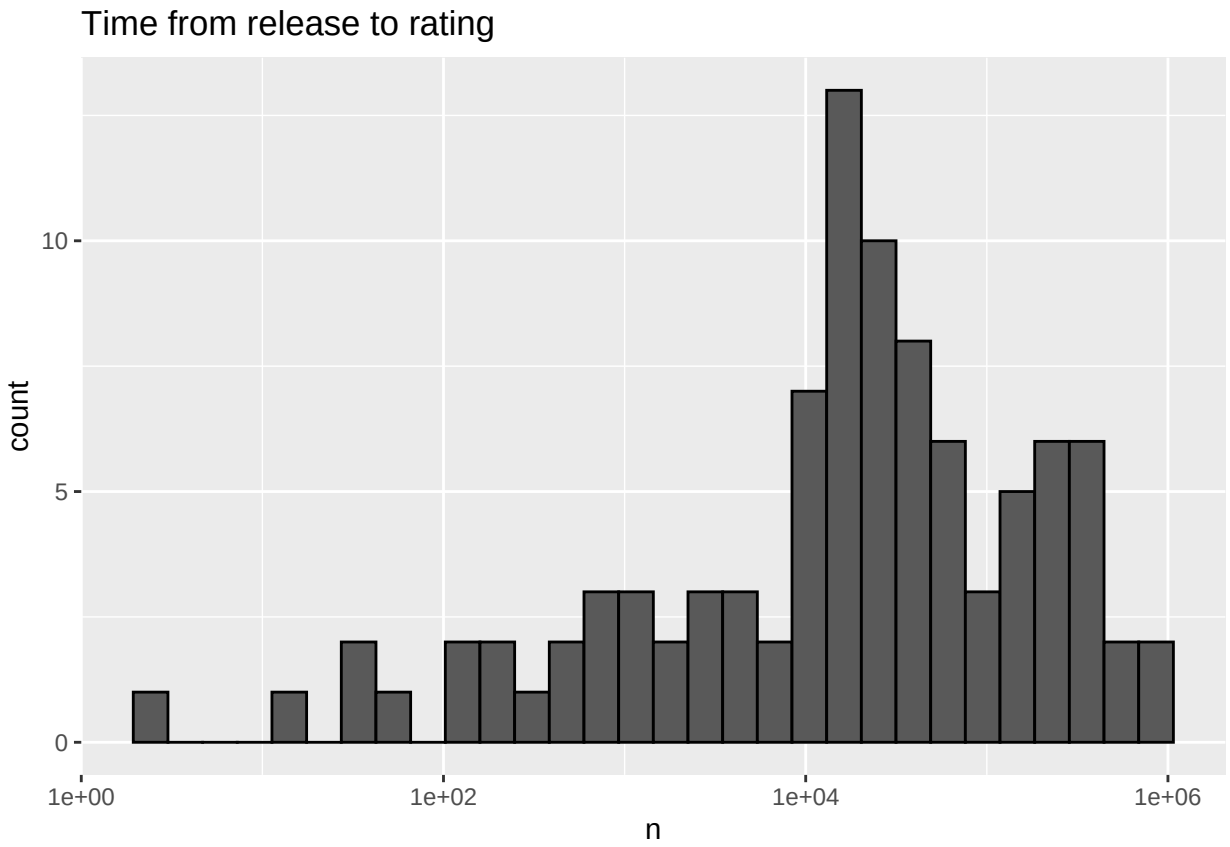


We can definitely see a change over the years, though what is probably more useful to utilise as a bias is the length of time between the films release and each rating.

```
# Time between release and rating.

edx %>%
  ggplot(aes(ReleasetoRatingTime)) +
  geom_bar() +
  ggtitle("Time from release to rating (years)")
```




```
edx %>%
  dplyr::count(ReleasetoRatingTime) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Time from release to rating")
```



Results - Building the Recommendation System

Model 1 - Just the average

The estimate that reduces the residual mean squared error is the least squares estimate of mu. Here it would be the average of all ratings which we can calculate as below.

```
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512574
```

We can calculate this on the training data to get the residual mean squared error ("RMSE") as below.

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060704
```

```
# Here we add the results to a table.
rmse_results <- tibble(method = "Model 1 - Just the average", RMSE = naive_rmse)
rmse_results
```

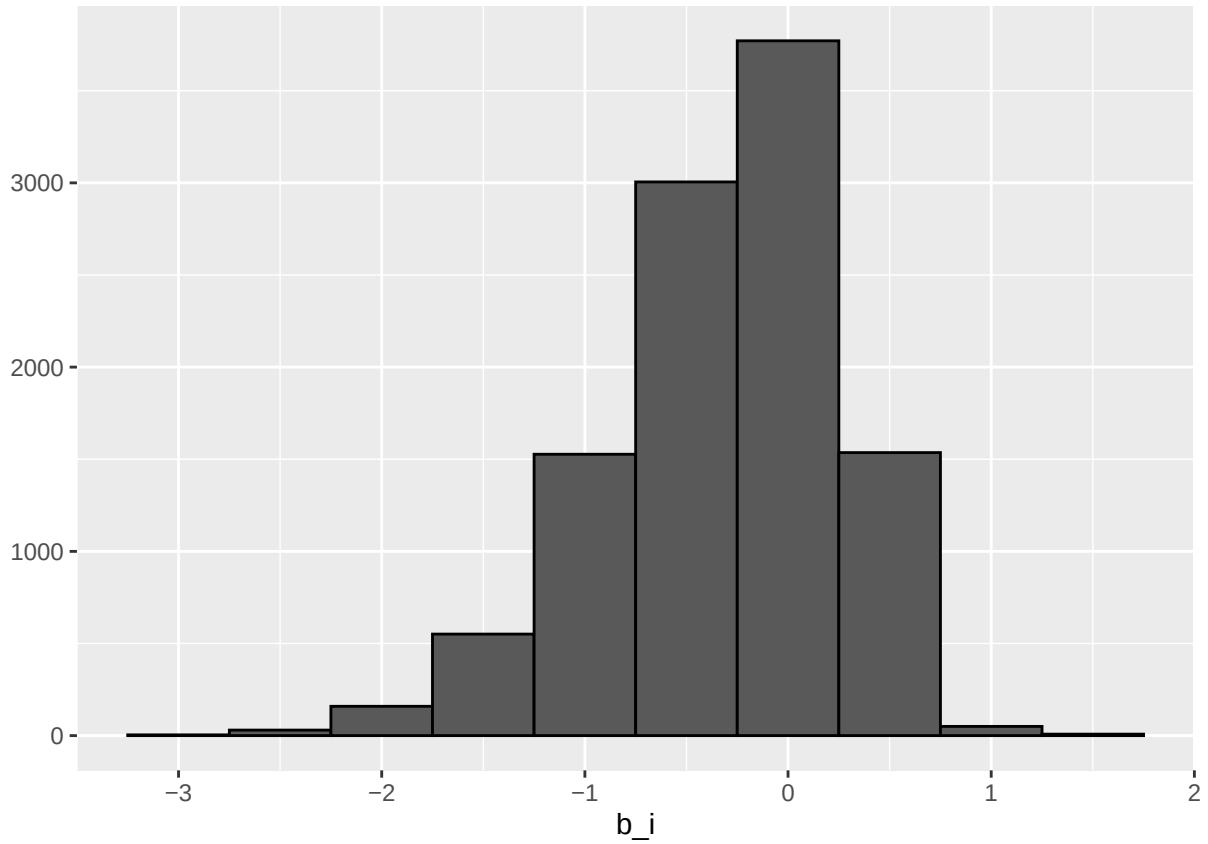
```
## # A tibble: 1 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Model 1 - Just the average  1.06
```

This is quite large, though we will be improving this as we go on.

Model 2 - Movie Effect Model

We will firstly improve upon this model by accounting for our first bias, movie bias, which we will call b_i. Some movies we saw in the data exploration section are rated higher than others, we want to adjust our model for this.

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

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

movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



We can see from the above q plot that these vary somewhat. The average rating is 3.5 so a 1.5 score shows a perfect 5.0 rating.

Now lets see if accounting for movie bias improves our model.

```
predicted_ratings2 <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i


model_2_rmse <- RMSE(predicted_ratings2, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 2 - Movie Effect Model",
                                     RMSE = model_2_rmse ))

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

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144

We can see that this has improved, though we will now try to account for User, Genre and Time biases.

Model 3 - Movie + User Effects Model

Now we will add in a bias for user effects which we will call b_u. Again, as seen in the data exploration stage different users give different ratings and rate different numbers of movies. So we should be able to improve the model based on this.

We will add this onto the model along with b_i.

```
#User bias
user_avgs <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings3 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings3, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 3 - Movie + User Effects Model",
                                     RMSE = model_3_rmse ))

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

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874

We again have made an improvement, we have added this to the results table above.

Model 4 - Movie + User + Genre Effects Model

We will now build genre bias into the model which we will call b_g.

The data will be left with the genres in their "non split out" format. Although we had split this to visualise the 19 different genres previously if we did this for the calculation it would create duplicate rating rows. We previously showed that different genres were rated differently in different release years.

```
#Genre bias
genre_avgs <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

predicted_ratings4 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

model_4_rmse <- RMSE(predicted_ratings4, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 4 - Movie + User + Genre Effects Model",
                                 RMSE = model_4_rmse ))

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

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425

The model has again improved but not as much. We will now add a final bias before moving on to other methods.

Model 5 - Movie + User + Genre + Time Effects Model

We showed in the inspection stage that the release year seems to have an effect on ratings. We will create another bias b_t to be the number of years from release year to year of rating.

```
time_avgs <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by="genres") %>%
  group_by(ReleasetoRatingTime) %>%
```

```
  summarize(b_t = mean(rating - mu - b_i - b_u - b_g))

predicted_ratings5 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(time_avgs, by='ReleasetoRatingTime') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  .$pred

model_5_rmse <- RMSE(predicted_ratings5, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 5 - Movie + User + Genre + Time Effects Model",
                                 RMSE = model_5_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425
Model 5 - Movie + User + Genre + Time Effects Model	0.8426097

Our final 4th bias has barely made an improvement on the model. We shall now move onto the method of regularisation.

Model 6 - Regularized Movie Effect Model

The last two models didn't improve much, so we will now introduce regularisation which was one of the techniques used by the Netflix challenge winners.

Below we summarise the 10 best and 10 worst films per the data. We can see that these are all pretty niche films. There is something awry here.

```
# Create a list of movie titles
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()

# Below are the top 10 best films according to our estimates.
movie_avgs %>% right_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail	1.487426
Shanghai Express	1.487426
Satan's Tango (SÃ¡tÃ¡ntangÃ³)	1.487426

title	b_i
Fighting Elegy (Kenka erejii)	1.487426
Sun Alley (Sonnenallee)	1.487426
Bullfighter and the Lady	1.487426
Blue Light, The (Das Blaue Licht)	1.487426
Human Condition II, The (Ningen no joken II)	1.320760
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)	1.237426
Life of Oharu, The (Saikaku ichidai onna)	1.237426

```
# Below are the top 10 worst films according to our estimates.
movie_avgs %>% right_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Besotted	-3.012573
Grief	-3.012573
Confessions of a Superhero	-3.012573
War of the Worlds 2: The Next Wave	-3.012573
Disaster Movie	-2.729240
SuperBabies: Baby Geniuses 2	-2.712573
From Justin to Kelly	-2.652325
Hip Hop Witch, Da	-2.612573
Criminals	-2.512573
Mountain Eagle, The	-2.512573

```
## Lets see how often these top 10 good movies were rated.
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  right_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

title	b_i	n
Hellhounds on My Trail	1.487426	1
Shanghai Express	1.487426	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³)	1.487426	2
Fighting Elegy (Kenka erejii)	1.487426	1
Sun Alley (Sonnenallee)	1.487426	1
Bullfighter and the Lady	1.487426	1
Blue Light, The (Das Blaue Licht)	1.487426	1
Human Condition II, The (Ningen no joken II)	1.320760	3
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)	1.237426	4

title	b_i	n
Life of Oharu, The (Saikaku ichidai onna)	1.237426	2

```
# We can see the same for the bad movies.
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  right_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

title	b_i	n
Besotted	-3.012573	2
Grief	-3.012573	1
Confessions of a Superhero	-3.012573	1
War of the Worlds 2: The Next Wave	-3.012573	1
Disaster Movie	-2.729240	30
SuperBabies: Baby Geniuses 2	-2.712573	40
From Justin to Kelly	-2.652325	161
Hip Hop Witch, Da	-2.612573	10
Criminals	-2.512573	2
Mountain Eagle, The	-2.512573	1

So it seems the top best and worst are skewed by niche films with a low number of ratings. We would define these as noisy estimates.

With regularisation we want to add a penalty (lambda) for large values of b to the sum of squares equations that we minimize.

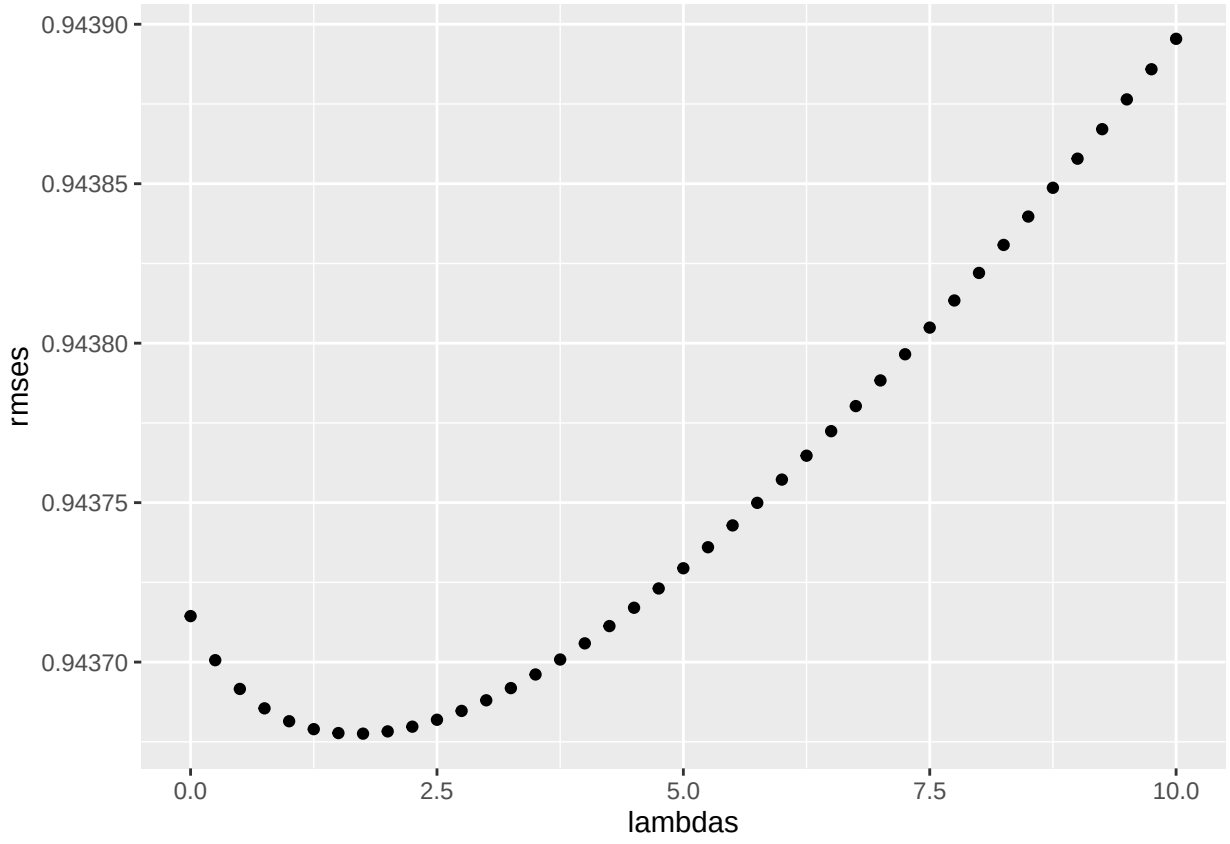
The larger the lambda the more we shrink these values. Lambda is a tuning parameter, we will use cross-validation to pick the ideal value.

```
##### Picking a Lambda and running regularisation for movie effect.
# Now note that lambda is a tuning parameter.
# We can use cross-validation to choose it.

lambdas <- seq(0, 10, 0.25) # Choose a range of lambas to test

mu <- mean(train_set$rating) # define mu

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

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
```

```
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```

The below graph gives us a visualisation of why we pick the lamda we do (the lowest).

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




```
# Below is the lambda selected
lambdas[which.min(rmses)]
```

```
## [1] 1.75
```

```
lambda <- lambdas[which.min(rmses)]  # make lambda the lowest value.
```

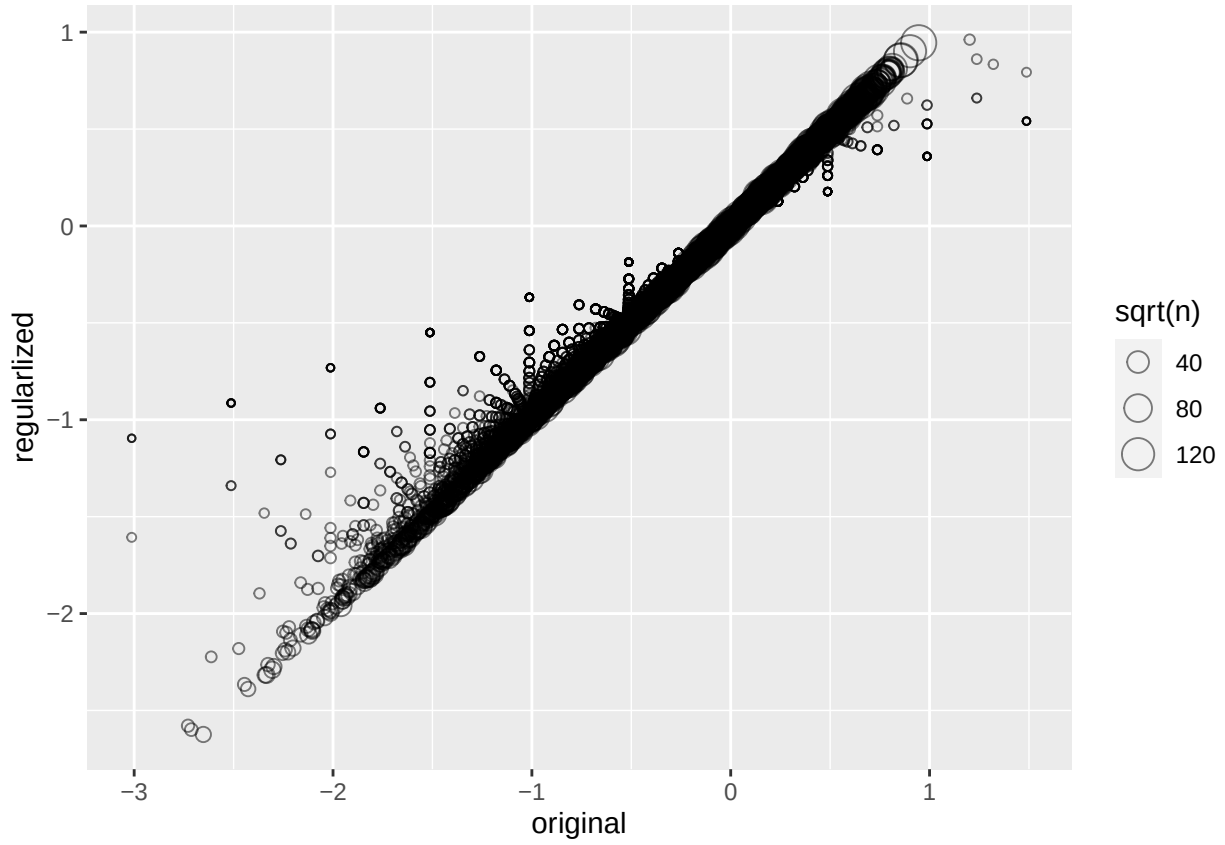
We will run the model again but this time with the optimal lambda.

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

We can visualise how the estimates shrink by plotting the regularised estimates vs the least squares estimates with the size of the circles being how large n_i was.

When n is small the values shink more towards zero.

```
tibble(original = movie_avgs$b_i,
           regularized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



So if we now look at the top 10 best/worst movies we can see what affect this has had. These now look a lot more reasonable!

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i	n
More	0.9613697	7
Shawshank Redemption, The	0.9453326	22406
Godfather, The	0.9016522	14166
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)	0.8608184	4
Schindler's List	0.8560531	18573

title	b_i	n
Usual Suspects, The	0.8533599	17357
Human Condition II, The (Ningen no joken II)	0.8341641	3
Casablanca	0.8121959	8982
Double Indemnity	0.8089041	1722
Sunset Blvd. (a.k.a. Sunset Boulevard)	0.8083493	2303

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i	n
From Justin to Kelly	-2.623806	161
SuperBabies: Baby Geniuses 2	-2.598873	40
Disaster Movie	-2.578810	30
PokÃ©mon Heroes	-2.389318	111
Carnosaur 3: Primal Species	-2.365652	52
Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie)	-2.316984	169
Gigli	-2.316966	238
Glitter	-2.290322	270
Barney's Great Adventure	-2.273311	165
Faces of Death 6	-2.263209	62

Though do we improve our RMSE? Yes we do and can see this below. Please note that this is an improvement from Model 2, not Model 6.

```
predicted_ratings6 <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

model_6_rmse <- RMSE(predicted_ratings6, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 6 - Regularized Movie Effect Model",
                                 RMSE = model_6_rmse ))
rmse_results %>% knitr::kable()
```

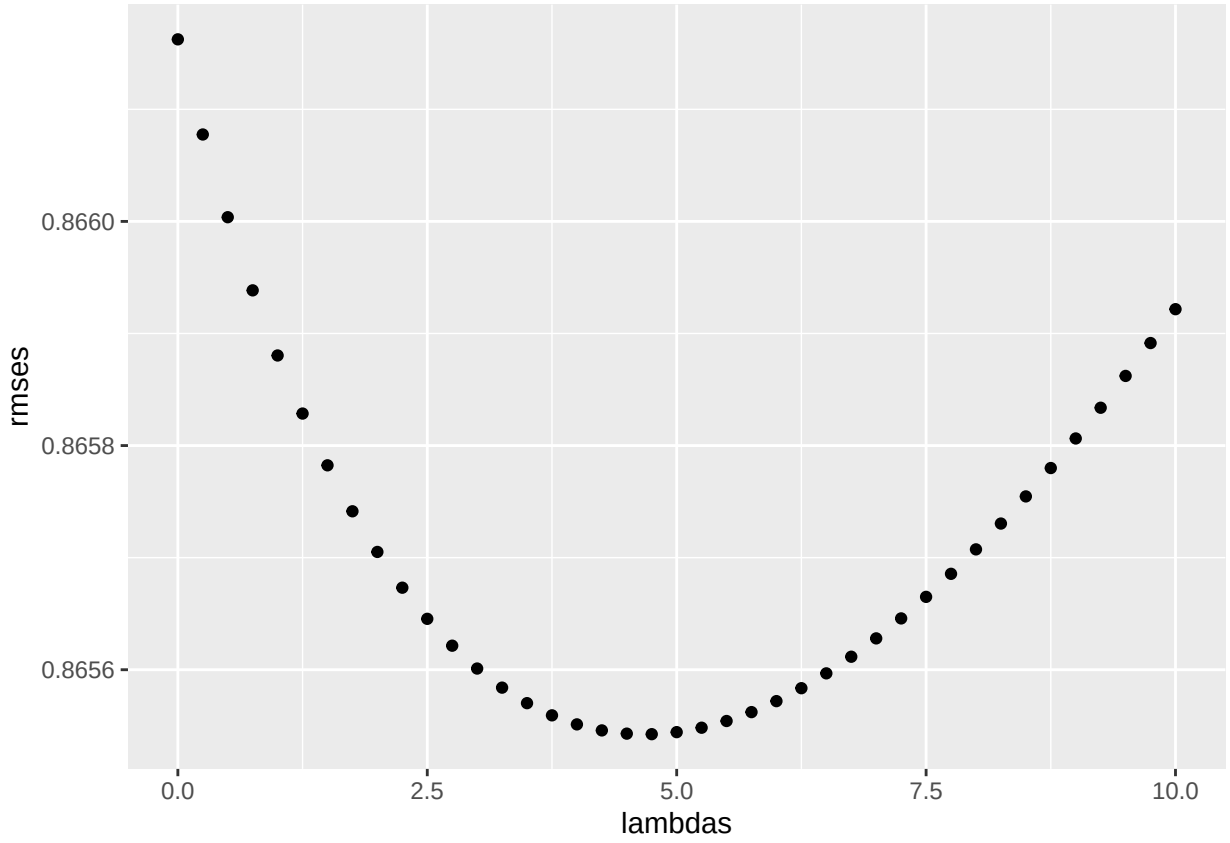
method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425
Model 5 - Movie + User + Genre + Time Effects Model	0.8426097

method	RMSE
Model 6 - Regularized Movie Effect Model	0.9436776

Now lets do the same but add in b_u and then again with b_g.

Model 7 - Regularized Movie + User Effect Model

```
##### Picking a Lambda and running regularisation for movie + user effect.

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```




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

```
## [1] 4.75
```

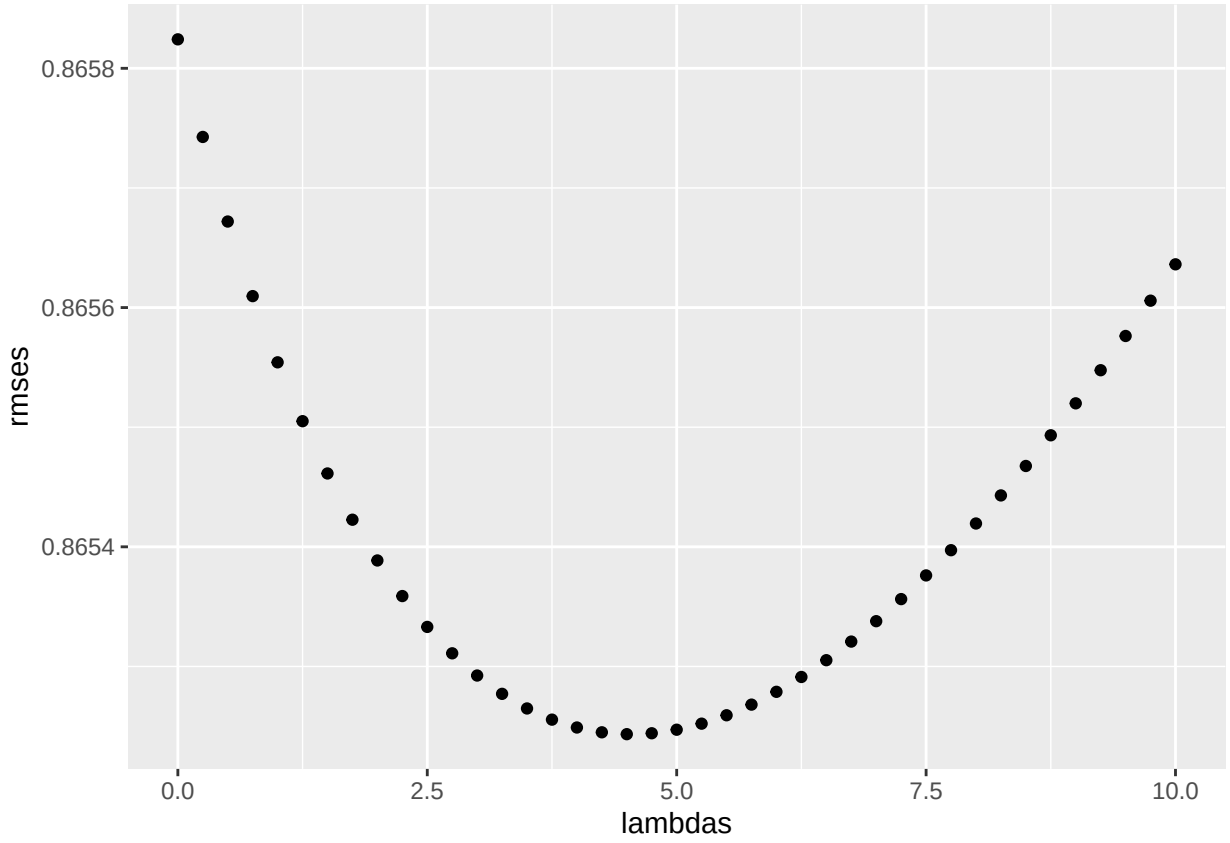
```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 7 - Regularized Movie + User Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425
Model 5 - Movie + User + Genre + Time Effects Model	0.8426097
Model 6 - Regularized Movie Effect Model	0.9436776
Model 7 - Regularized Movie + User Effect Model	0.8655425

It doesnt seem to have improved from Model 3. Perhaps b_u wasn't the best bias to add in? We can try with b_g but if it doesn't improve more we will move onto a different method.

Model 8 - Regularized Movie + User + Genre Effect Model

```
##### Picking a Lambda and running regularisation for movie + user + genre effect.

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  b_g <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```




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

```
## [1] 4.5
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 8 - Regularized Movie + User + Genre Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425
Model 5 - Movie + User + Genre + Time Effects Model	0.8426097
Model 6 - Regularized Movie Effect Model	0.9436776
Model 7 - Regularized Movie + User Effect Model	0.8655425
Model 8 - Regularized Movie + User + Genre Effect Model	0.8652434

Again we didnt seem to add an improvement. Now we could likely improve this again if we optimised the lambda for each bias rather than using one for all.

However, we shall instead try Matrix Factorisation instead.

Model 9 - Matrix Factorisation

We will be using the recosystem package for the matrix factorisation analysis.

Using this package we will do the below steps.

- Make our data into matrices to be used by the package
- Create a model (calling the function reco())
- Tune the model to use the best parameters using tune()
- Train the model
- Calculate predicted values

```
# We firstly need to change our data into matrices to be used by the recosytem package.
train_set_MF <- train_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()

test_set_MF <- test_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()


train_set_MF <-  with(train_set, data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating     = rating))

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

# Now we create a model object by calling Reco()
r <- Reco()

# Now we tune our model to use the best parameters.
opts <- r$tune(train_set_MF, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                         costp_l1 = 0, costq_l1 = 0,
                                         nthread = 1, niter = 10))

# Train the model
r$train(train_set_MF, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9933   1.0024e+07
##    1       0.8785   8.0811e+06
##    2       0.8469   7.5010e+06
##    3       0.8256   7.1593e+06
##    4       0.8083   6.9096e+06
##    5       0.7953   6.7293e+06
##    6       0.7845   6.5861e+06
```

```
##    7       0.7753   6.4742e+06
##    8       0.7671   6.3817e+06
##    9       0.7601   6.3050e+06
##   10       0.7538   6.2368e+06
##   11       0.7482   6.1791e+06
##   12       0.7434   6.1335e+06
##   13       0.7388   6.0881e+06
##   14       0.7347   6.0494e+06
##   15       0.7310   6.0150e+06
##   16       0.7275   5.9854e+06
##   17       0.7244   5.9587e+06
##   18       0.7215   5.9330e+06
##   19       0.7187   5.9091e+06
```

```
# Calculate the predicted values
y_hat_reco <-  r$predict(test_set_MF, out_memory())

rmses <- RMSE(test_set$rating, y_hat_reco)

# Save the results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 9 - Matrix Factorisation with Recosystem",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425
Model 5 - Movie + User + Genre + Time Effects Model	0.8426097
Model 6 - Regularized Movie Effect Model	0.9436776
Model 7 - Regularized Movie + User Effect Model	0.8655425
Model 8 - Regularized Movie + User + Genre Effect Model	0.8652434
Model 9 - Matrix Factorisation with Recosystem	0.7907496

This has shown a massive improvement! Now we shall test it on the validation set to be sure we have created a real result.

Present Modeling Results

Now it seems that Models 3, 4, 5 and 9 all have given us a RMSE < 0.86490. However we will utilise Model 9, the best outcome to test against our validation set.

```
validation_set_MF <- validation %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()


validation_set_MF <-  with(validation, data_memory(user_index = userId,
```

```
                                          item_index = movieId,
                                          rating     = rating))

# Calculate the predicted values
y_hat_reco <-  r$predict(validation_set_MF, out_memory())

rmses <- RMSE(validation$rating, y_hat_reco)

# Save the results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Model 10 - Final Validation by Matrix Factorisation with Recosy
                                 RMSE = min(rmses)))

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

method	RMSE
Model 1 - Just the average	1.0607045
Model 2 - Movie Effect Model	0.9437144
Model 3 - Movie + User Effects Model	0.8435874
Model 4 - Movie + User + Genre Effects Model	0.8430425
Model 5 - Movie + User + Genre + Time Effects Model	0.8426097
Model 6 - Regularized Movie Effect Model	0.9436776
Model 7 - Regularized Movie + User Effect Model	0.8655425
Model 8 - Regularized Movie + User + Genre Effect Model	0.8652434
Model 9 - Matrix Factorisation with Recosystem	0.7907496
Model 10 - Final Validation by Matrix Factorisation with Recosystem	0.7904513

Discuss Model Performance

Our final model had held true shown by our final value of RMSE < 0.86490.

As we went through the various models it seems that movie and user biases had a significant effect on reducing our RMSE, regularisation did somewhat to improve it but the real game changer was matrix factorisation.

Conclusion

Summary

We managed to gain a value of RMSE < 0.86490 at around 0.79 whilst showing significant improvement as we went on. Matrix factorisation was by far the best and (in terms of complexity of code) the simplest to run.

Limitations/Future work

When applying bias to genres it would have been great to have applied this to the 19 genres rather than the 797 combinations. If we would devise a way to analyse these without creating duplicate rows we may be able to improve the model further.

The final model provides our best results but the real question is "Can this be applied in real time in the real world?" This final code took a few minutes to run and I doubt netflix users would consider waiting for

this to run when looking for a film to watch. A way to improve would perhaps to be able to store results on a dataframe and update this each time new data is added.