The Development of a Recommendation Model for Movies: A Case Study

Colin Howlett

03/05/2020

Introduction

The purpose of this project was to develop an effective model that can predict the ratings individual users of a streaming service such as Netflix would give to movies. Predicted ratings of this kind could then inform movie recommendations to users.

The dataset used to train and validate the model was the MovieLens 10M dataset from grouplens.org.[1] This dataset contains nearly 10 million movie ratings of approximately 10,000 movies by 72,000 users according to GroupLens.

The key steps in this project were first of all to download the data and read it into a data frame within R. The data was then split into a training set and a validation set.[2] The validation data was set aside to be used only for final assessment of the overall performance of the final model, while the training data was used to build the model itself.

Next the data was cleaned, and the training data was further split into a training set and a test set to be used for model assessment during the model building process.[3]

The data was then explored using summary statistics and visualisation to understand it's critical characteristics, before embarking on the process of building a model.

This model was built incrementally, testing each time using the test set (not the validation data set aside earlier) to assess the performance of the model at each step.[4]

Finally the model was validated against the validation data that was set aside for the purpose, and the RMSE value was calculated and compared to the target value. This target, based on previous experience of building similar models, was to acheive an RMSE of 0.86490 or below.

[1] The MovieLens 10M Dataset is described by GroupLens as a stable benchmark dataset. It was released in January 2009 and is available here: `https://grouplens.org/datasets`

[2] The training data and validation data are stored in the `edx` dataframe and the `validation` dataframe respectively

[3] The training set and test set derived from the `edx` training data and used for model training and performnce assessment during model building are stored in the `training_set` dataframe and `test_set` dataframe respectively.

[4] The performance assessment took the form of calculating the Root Mean Squared Error (RMSE) of the predicted values with respect to the actual ratings within the model building test data: `test_set`.

Method

As set out above, the data was downloaded from grouplens.org. GroupLens supplied two data tables as text files - one with data about the movies themselves - their titles and genres - and one with data about the ratings given to movies by users. These were read into R as two separate dataframes and then joined.

```
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")
```

Once available in R as a single dataframe, the first step is to set aside a portion of the data for final assessment of the performance of the model:

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

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

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

In this case a random sample of 10% of the data was set aside. Also critically, any movies or users in the validation data `validation` but not in the training data `edx` are filtered out.[5]

[5] This is done to ensure that a model trained on the training data will work reliably when assessed using the validation data.

```
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

These removed rows are then added back into the `edx` training data.

```
removed <- anti_join(temp, validation)

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

Now that the data has been imported and a validation data set has been set aside, the data needs to be cleaned. An important first step is simply to examine the data to see what needs to be done:

```
knitr::kable(
  edx[1:5,4:6]
)
```

	timestamp	title	genres
1	838985046	Boomerang (1992)	Comedy|Romance
2	838983525	Net, The (1995)	Action|Crime|Thriller
4	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

Focusing just on the last three columns of data, we can immediately identify three problems. Firstly the `timestamp` column (which represents the moment that the user submitted their rating) is just a number. In fact this is the number of seconds since the 'epoch'.[6] Secondly, we can see that the year of release is included as part of the title of the movie. Thirdly, we can see that the `genres` column contains a variable number of different genres separated by a `|` pipe character.

[6] The 'epoch' is defined as starting at midnight on the 1st January 1970.

How we choose to clean up this data will depend on the strategy we choose for building a model.

The choice of strategy for this project was guided by two references. Firstly, Irizarry[7] proposes a model of the form

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

[7] Irizarry, Rafael. 2020. Introduction to Data Science., ch. 33.7.5, https://rafalab.github.io/dsbook/.

Where $Y_{u,i}$ is the predicted rating, μ is the average of all ratings, b_i is the bias due to each item (in this case, due to each movie), b_u is the bias due to each movie and $\varepsilon_{u,i}$ is a term representing the remaining unexplained variation in ratings.

Secondly, Chen's article[8] outlining the winning solution to the Netflix Prize, sets out how this model can be extended by considering additional sources of bias, such as the time since a user's first rating, the time since a movie's first rating and the number of people who have rated the movie.

[8] Chen, Edwin. 2011. "Winning the Netflix Prize: A Summary." http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/.

Further, I examined the data to see what information was available and tested the effectiveness of additional biases due to the year of the release of the movie and it's genre.

Having decided to pursue this modelling approach, it was clear that the key data cleaning actions necessary, were firstly to extract the year of release of each movie into it's own column, and to convert the timestamp representing the moment a rating was made by a user into a more usable date format. The first of these tasks was completed up front, and was applied to both the training and validation data.[9] The second was addressed when the timestamp was used as part of the model building process, and so was built into the model itself.

[9] Both the `edx` set and the `validation` set had the same cleaning actions performed on them to ensure the `validation` set would be compatible with the model trained on the `edx` set when the time came to assess the performance of the model.

```
library(stringr)

edx <- edx %>%
  mutate(year = as.numeric(
    str_match(title, "\\((\\d{4})\\)$")[,2]))

validation <- validation %>%
```

```
  mutate(year = as.numeric(
    str_match(title, "\\((\\d{4})\\)$")[,2]))
```

Consideration was given at this stage, as to whether to split the genres column into individual data items for each genre. The difficulty with this approach is that there are a variable number of genres for each sample. They could not be gathered into a single column as this would increase the number of samples, while giving each a column would leave some samples with empty columns. It was therefore decided to leave them as a single composite column, for ease of model building.

The resulting 'clean' data looks like this:

```
knitr::kable(
  edx[1:5,4:7]
)
```

timestamp	title	genres	year
838985046	Boomerang (1992)	Comedy|Romance	1992
838983525	Net, The (1995)	Action|Crime|Thriller	1995
838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995
838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1994
838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994

Having completed this initial data cleaning, the training data needed to be further split into a training set `train_set` and a test set `test_set` which would be used for checking the performance of the model as it was built, leaving the validation data untouched.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(
  y = edx$rating,
  times = 1,
  p = 0.2,
  list = FALSE)

train_set <- edx[-test_index,]
test_set <- edx[test_index,]

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

The function for calculating the Root Mean Squared Error to evaluate model performance also needed to be created:

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

Now we are ready to test the first very simple model:

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

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

result <- RMSE(test_set$rating, mu)
```

This very simple model gives an RMSE of 1.059904

The next step is for us to look at the effect of the bias due to the variation of the average rating of a movie on the ratings movies receive. That is, to consider a model of the form:

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

We can calculate these biases like this:

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

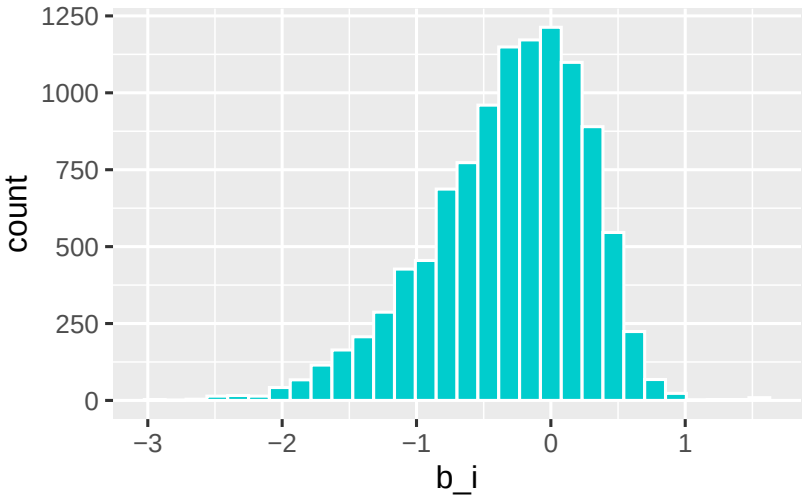
When we calculate biases in that way, we find we get data which can be tabulated like this:

```
knitr::kable(
  movie_avgs[1:5,]
)
```

title	b_i
...All the Marbles (a.k.a. The California Dolls) (1981)	-1.3249821
...And God Created Woman (Et Dieu... créa la femme) (1956)	-0.3992746
...And God Spoke (1993)	-0.3065998
...And Justice for All (1979)	0.1315088
'burbs, The (1989)	-0.5214954

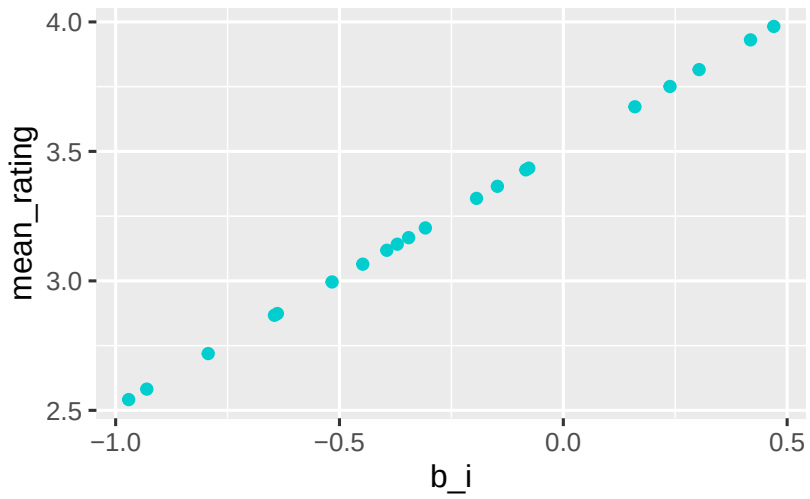
If we plot this data, we can see the variation in average rating:

```
movie_avgs %>% ggplot(aes(b_i)) +
  geom_histogram(bins=30, color="white", fill="cyan3")
```



We can also see the correlation between b_i and the mean $Y_{u,i}$ for the first 20 movies with a plot like this:

```
movie_avgs %>% filter(movieId <= 20) %>%
  left_join(train_set, by='movieId') %>%
    group_by(b_i) %>%
    summarize(mean_rating=mean(rating)) %>%
    ggplot(aes(b_i, mean_rating)) +
    geom_point(color="cyan3", fill="cyan3")
```



This is not suprising, as the b_i is simply $Y_{u,i} - \mu$.

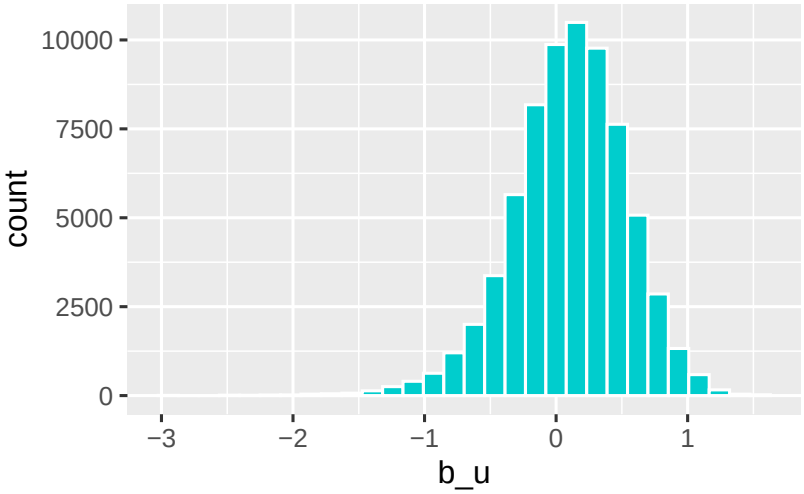
If we try to fit this model, we see that we get an improved RMSE of 0.9437429

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

result <- RMSE(test_set$rating, predicted_ratings)
```

Next, we move on to examine the effect on the movie rating due to the variability in users' propensity to give higher or lower ratings.

We can see this variation in this plot:

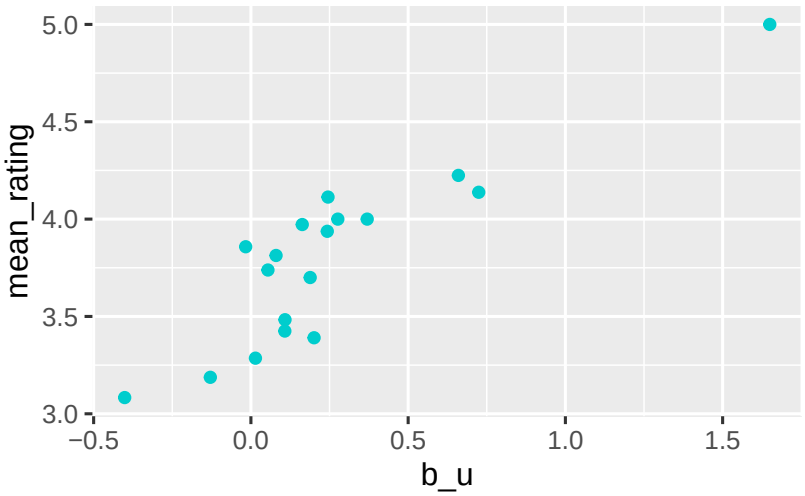
```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "white", fill="cyan3")
```



This strongly suggests that the variability in users' propensity to award high or low ratings will also be a useful predictor of the actual rating awarded to a movie.

We can see the correlation for the first 20 users here:

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

user_avgs %>% filter(userId <= 20) %>%
  left_join(train_set, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
    group_by(b_u) %>%
    summarize(mean_rating=mean(rating)) %>%
    ggplot(aes(b_u, mean_rating)) +
    geom_point(color="cyan3", fill="cyan3")
```



So now we can try fitting a model of the form:

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

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

result <- RMSE(test_set$rating, predicted_ratings)
```

This gives us an improved RMSE of 0.865932.

Now it is time to consider additional bias terms. The first one we will consider is the bias due to genre

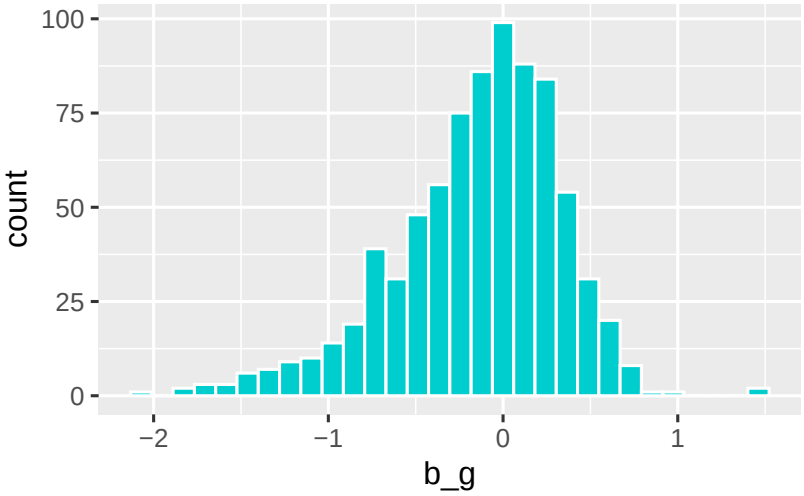
Firstly we want to see how many compound genres there are: 797

```
num_genres <- train_set %>%
  group_by(genres) %>%
  summarize(n()) %>%
  nrow()
```

This is small enough that we can consider the data to be categorical in the same way as userIds and movieIds are categorical.

Again to understand the value of this bias as a predictive value we should first explore the distribution of bias by compound genre:

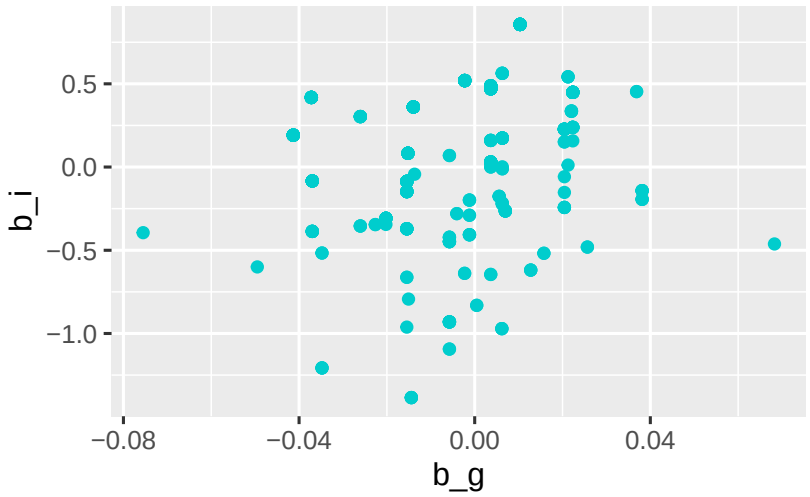
```
train_set %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu)) %>%
  ggplot(aes(b_g)) +
  geom_histogram(bins = 30, color = "white", fill="cyan3")
```



So we can see that the bias is normally distributed.

This time it may be useful to explore the correlations between the movie and genre biases to better understand how they influence the overall rating

```
genre_avgs <- train_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))

cor_check <- genre_avgs %>%
  left_join(train_set, by='genres') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  filter(movieId <= 100) %>%
  filter(userId <= 100)

cor_check %>%
    ggplot(aes(b_g, b_i)) +
    geom_point(color="cyan3", fill="cyan3")
```



We see that there is a positive correlation, but it is not particularly strong: only 0.1855112 [10]

[10] Remember though that b_i is very strongly correlated with $Y_{i,u}$.

Nevertheless, if we add this term to our model:

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

We can make a fresh set of predictions: and we find a somewhat improved RMSE of 0.8655941

```
predicted_ratings <- 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) %>%
  pull(pred)

# Once again, we get an improved RMSE: 0.8655941
result <- RMSE(test_set$rating, predicted_ratings)
```

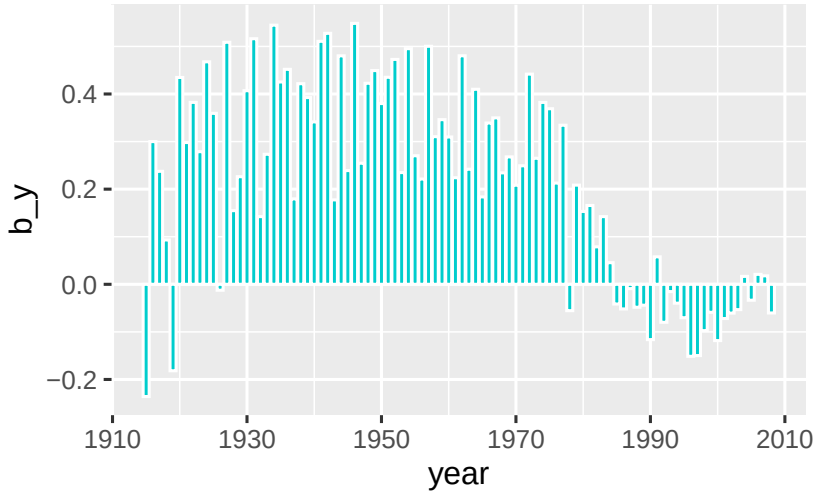
Next we turn our attention to the time effects referred to by Chen. Perhaps the year of release will have an effect on the rating.

First let's examine how many different release years there are: 94

```
num_years <- train_set %>%
  group_by(genres) %>%
  summarize(n()) %>%
  nrow()
```

Next we want to explore how ratings vary by year of release

```
train_set %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu)) %>%
  ggplot(aes(year, b_y)) +
  geom_bar(stat="identity", color = "white", fill="cyan3")
```



We can see that older movies tend to receive higher ratings. So a new model including this bias may give improved results.

So now our model takes the form:

$$Y_{u,i,g,y} = \mu + b_i + b_u + b_g + b_y + \varepsilon_{u,i,g,y}$$

If we run this model, we do in fact see an improved RMSE

```
year_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>%
```

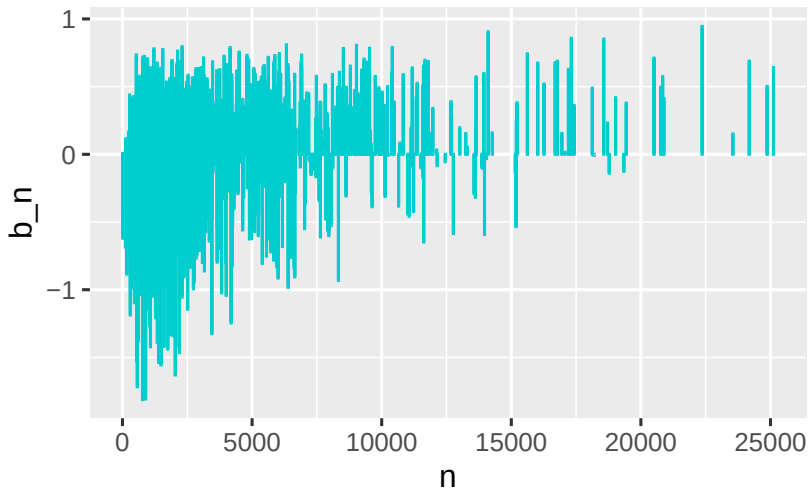
```
  pull(pred)

result <- RMSE(test_set$rating, predicted_ratings)
```

Now the RMSE is 0.8654189

The next intuition to consider, prompted by Chen, is that a movie's popularity (or rather, the number of ratings it received) may influence the rating. That is, people may be more likely to give a rating to a movie they enjoyed or that they hated. So we should examine this bias.

Firstly, how do movies differ in the number of ratings they receive?

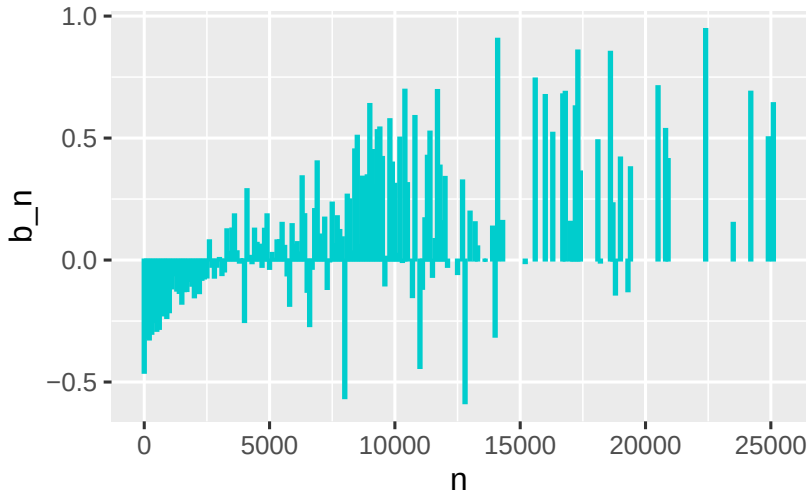
```
train_set %>%
  group_by(movieId) %>%
  summarize(n = n(), rating=mean(rating)) %>%
  group_by(n) %>%
  summarize(b_n = mean(rating - mu)) %>%
  ggplot(aes(n, b_n)) +
  geom_bar(stat="identity", color = "cyan3", fill="cyan3")
```



We can see that movies with a lower number of ratings, have a much greater variability in the ratings they receive. We can also see that movies with a higher number of ratings tend to receive more positive ratings.

Another effect we can see here is that the distinct values of n at the higher numbers are quite sparse. This suggests that for our model we would be well advised to adopt a binning approach. If we do that, and run the data again, we will see a smoother distribution and a more obvious effect.

```
train_set %>%
  group_by(movieId) %>%
  summarize(n = round(n(), -2), rating=mean(rating)) %>%
  group_by(n) %>%
  summarize(b_n = mean(rating - mu)) %>%
  ggplot(aes(n, b_n)) +
  geom_bar(stat="identity", color = "cyan3", fill="cyan3")
```



So again, we can include this term within our model.

$$Y_{u,i,g,y,n} = \mu + b_i + b_u + b_g + b_y + b_n + \varepsilon_{u,i,g,y,n}$$

If we do so and generate predictions, we can see that the RMSE improves again.

```
movie_pop <- train_set %>%
  group_by(movieId) %>%
  summarize(num_ratings=round(n(), -2))

pop_avgs <- train_set %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  group_by(num_ratings) %>%
  summarize(b_n = mean(rating - mu - b_i - b_u - b_g - b_y))

predicted_ratings <- test_set %>%
```

```
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(pop_avgs, by='num_ratings') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y + b_n) %>%
  pull(pred)

result <- RMSE(test_set$rating, predicted_ratings)
```

This time the RMSE has improved slightly to: 0.8652957

Now we want to look at the influence of the number of days since a movie was first rated, as suggested by Chen. He suggests that early ratings tend to come from fans, and so tend to be higher, while later raters may be more lukewarm.

To do this, we will have to convert the timestamp field into a date and then calculate the number of days since that date, then take the maximum value of this for each movie to find the number of days since that movie was first rated.

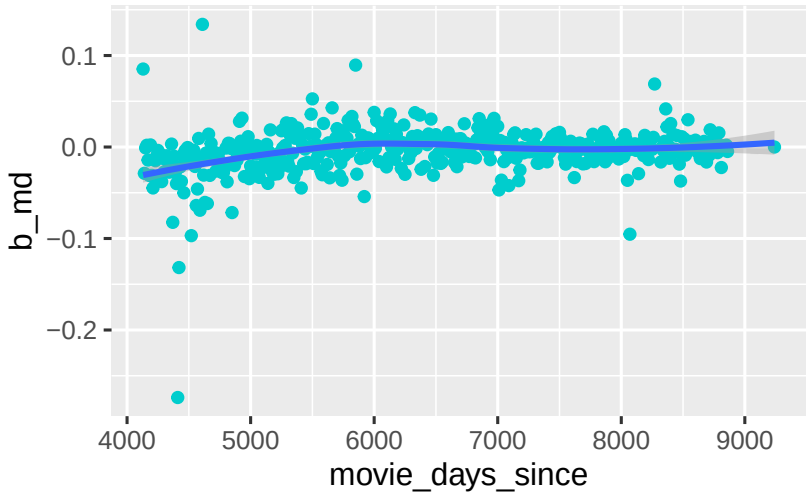
Again, we are binning the day since value - rounding it to the nearest 10 days.

```
movie_days <- train_set %>%
  mutate(days_since=as.numeric(
                      as.Date("2020-04-30") -
                      as.Date(as.POSIXct(
                        timestamp,
                        origin="1970-01-01")))) %>%
  group_by(movieId) %>%
  summarize(movie_days_since=round(max(days_since),-1))
```

Now we can assess the effect that the number of days since a movie was first rated has on ratings for each movie

```
movie_day_avgs <- train_set %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
```

```
  left_join(pop_avgs, by='num_ratings') %>%
  group_by(movie_days_since) %>%
  summarize(b_md = mean(
    rating - mu - b_i - b_u - b_g - b_y - b_n))

movie_day_avgs %>%
    ggplot(aes(movie_days_since, b_md)) +
    geom_point(color="cyan3", fill="cyan3") +
    geom_smooth()
```



We see a relatively small variation of the bias depending on the days since a movie was first rated.

Nevertheless, once we add this bias into the model

$$Y_{u,i,g,y,n,md} = \mu + b_i + b_u + b_g + b_y + b_n + b_{md} + \varepsilon_{u,i,g,y,n,md}$$

We again see an incremental improvement in the RMSE

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(pop_avgs, by='num_ratings') %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y + b_n + b_md) %>%
```

```
  pull(pred)

result <- RMSE(test_set$rating, predicted_ratings)
```

We now see an RMSE of 0.8652373

Now we can look at the influence of the number days since a user first rated any movie on movie ratings. Chen suggests that a user's ratings may decline over time. Once again, we need to calculate the days since a user first rated any movie in much the same way we calculated the days since a movie was first rated by any user.

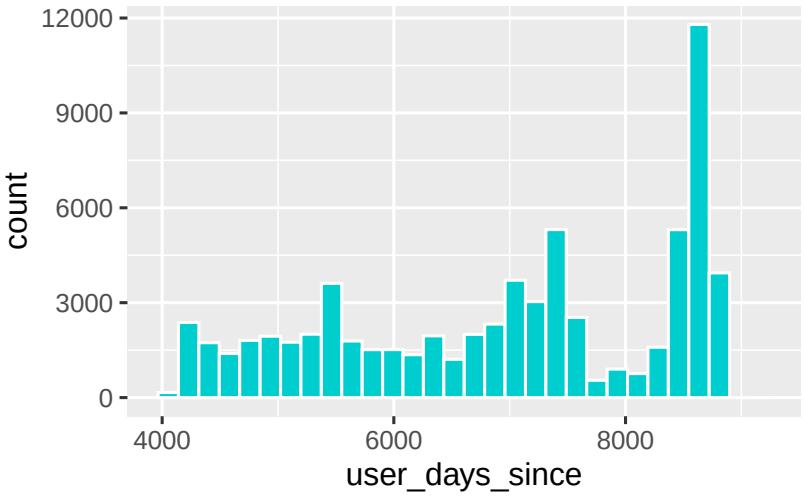
To do this, we will have to convert the timestamp field into a date and then calculate the number of days since that date, then take the maximum value of this for each movie to find the number of days since that movie was first rated.

Again, we are binning the days since value - rounding it to the nearest 10 days.

```
user_days <- train_set %>%
  mutate(days_since=as.numeric(
    as.Date("2020-04-30") -
      as.Date(as.POSIXct(
        timestamp,
        origin="1970-01-01")))) %>%
  group_by(userId) %>%
  summarize(user_days_since=round(max(days_since),-1))
```

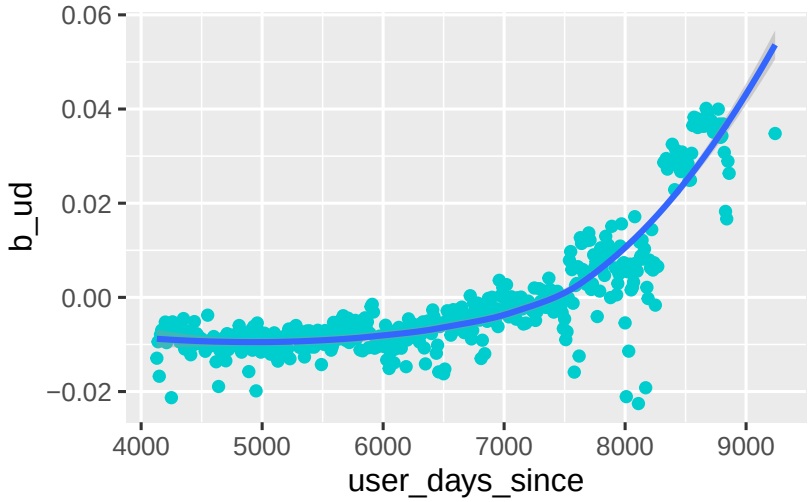
It may be helpful to understand the distribution of this (it will tell us how long users have been using the platform).

```
user_days %>% ggplot(aes(user_days_since)) +
  geom_histogram(bins=30, color="white", fill="cyan3")
```



We can see we have quite a lot of users who have been on the platform for some time. Now we can assess the effect that the number of days since a movie was first rated has on ratings for each movie

```
user_day_avgs <- train_set %>%
  left_join(user_days, by='userId') %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(pop_avgs, by='num_ratings')%>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  group_by(user_days_since) %>%
  summarize(b_ud = mean(
    rating - mu - b_i - b_u - b_g - b_y - b_n - b_md))

user_day_avgs %>%
    ggplot(aes(user_days_since, b_ud)) +
    geom_point(color="cyan3", fill="cyan3") +
    geom_smooth()
```



Now we can see clearly how much more strongly this bias varies depending on how many days it is since the user first rated any movie. This is clearly worth including as a term in our model.

$$Y_{u,i,g,y,n,md,ud} = \mu + b_i + b_u + b_g + b_y + b_n + b_{md} + b_{ud} + \varepsilon_{u,i,g,y,n,md,ud}$$

When we do so, we again see an incremental improvement

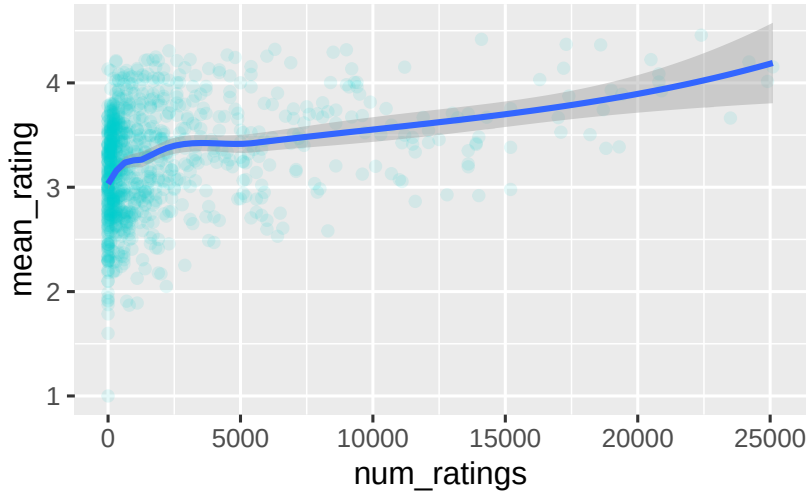
```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(pop_avgs, by='num_ratings') %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  left_join(user_days, by='userId') %>%
  left_join(user_day_avgs, by='user_days_since') %>%
  mutate(pred = mu + b_i + b_u + b_g +
           b_y + b_n + b_md + b_ud) %>%
  pull(pred)

result <- RMSE(test_set$rating, predicted_ratings)
```

The RMSE on our test set is now at 0.8651212

A critical part of the modelling process that we have ignored so far is regularization. Earlier we plotted the bias on the prediction due to the number of ratings a movie has, and we saw that movies with relatively few ratings

have great variability in their bias. We can see this more clearly by plotting the ratings of movies against the number of ratings a movie has.

```
train_set %>%
  group_by(movieId) %>%
  summarize(mean_rating=mean(rating)) %>%
  left_join(movie_pop, by='movieId') %>%
  filter(movieId <= 1000) %>%
  ggplot(aes(num_ratings, mean_rating)) +
  geom_point(color="cyan3", fill="cyan3", alpha=0.1) +
  geom_smooth()
```



We can see that regularizing, that is giving less weight to the biases of movies with few ratings, could improve our prediction accuracy further.

Regularization is performed by adding a term lambda to the number of items we divide by when calculating averages. More specifically, while the unregularized bias is calculated thus:

$$\left(\mu + \sum_{x=2}^{n} b_x\right) / n$$

The regularized bias is calculated as:

$$\left(\mu + \sum_{x=2}^{n} b_x\right) / \left(n + \lambda\right)$$

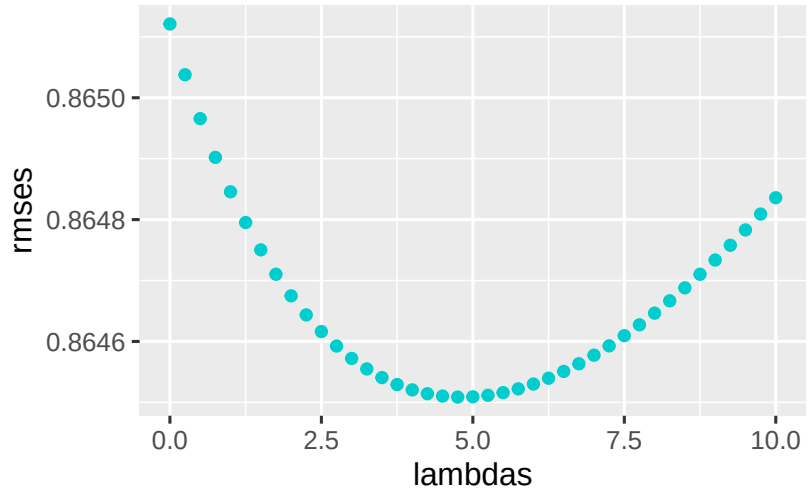
Thus λ is a parameter to be optimised. If we apply regularization we can attempt to optimize λ.

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

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

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+1))

user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+1))

genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+1))

year_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(year) %>%
  summarize(
    b_y = sum(rating - mu - b_i - b_u - b_g)/(n()+1))

movie_pop <- train_set %>%
  group_by(movieId) %>%
  summarize(num_ratings=round(n(), -2))

pop_avgs <- train_set %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  group_by(num_ratings) %>%
  summarize(
    b_n = sum(rating - mu - b_i - b_u - b_g - b_y)/(n()+1))

movie_days <- train_set %>%
  mutate(days_since=as.numeric(
    as.Date("2020-04-30") -
      as.Date(as.POSIXct(
        timestamp, origin="1970-01-01")))) %>%
  group_by(movieId) %>%
```

```
  summarize(movie_days_since=round(max(days_since),-1))

movie_day_avgs <- train_set %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(pop_avgs, by='num_ratings') %>%
  group_by(movie_days_since) %>%
  summarize(b_md = sum(rating - mu - b_i - b_u
                       - b_g - b_y - b_n)/(n()+1))

user_days <- train_set %>%
  mutate(days_since=as.numeric(
    as.Date("2020-04-30") -
      as.Date(as.POSIXct(
        timestamp, origin="1970-01-01")))) %>%
  group_by(userId) %>%
  summarize(user_days_since=round(max(days_since),-1))

user_day_avgs <- train_set %>%
  left_join(user_days, by='userId') %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(pop_avgs, by='num_ratings')%>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  group_by(user_days_since) %>%
  summarize(b_ud = sum(rating - mu - b_i - b_u - b_g
                       - b_y - b_n - b_md)/(n()+1))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(pop_avgs, by='num_ratings') %>%
```

```
  left_join(movie_days, by='movieId') %>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  left_join(user_days, by='userId') %>%
  left_join(user_day_avgs, by='user_days_since') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y
         + b_n + b_md + b_ud) %>%
  pull(pred)

return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses, color=I("cyan3"))
```



We can see from this plot that the optimum value of λ is almost 5.0 - more precisely it is 4.75

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

The RMSE acheived on the `test_set` data with the regularized model is 0.8645087

```
result <- min(rmses)
```

Finally we can use this optimum lambda and apply our model to the as yet untouched validation data

```
l <- 4.75

movie_avgs <- edx %>%
```

```
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+1))

user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+1))

genre_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(
    b_g = sum(rating - mu - b_i - b_u)/(n()+1))

year_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(year) %>%
  summarize(
    b_y = sum(rating - mu - b_i - b_u - b_g)/(n()+1))

movie_pop <- edx %>%
  group_by(movieId) %>%
  summarize(num_ratings=round(n(), -2))

pop_avgs <- edx %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  group_by(num_ratings) %>%
  summarize(b_n = sum(rating - mu - b_i - b_u
                      - b_g - b_y)/(n()+1))

movie_days <- edx %>%
  mutate(days_since=as.numeric(
    as.Date("2020-04-30") -
      as.Date(as.POSIXct(
        timestamp, origin="1970-01-01")))) %>%
  group_by(movieId) %>%
```

```
  summarize(movie_days_since=round(max(days_since),-1))

movie_day_avgs <- edx %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(pop_avgs, by='num_ratings') %>%
  group_by(movie_days_since) %>%
  summarize(b_md = sum(rating - mu - b_i - b_u
                       - b_g - b_y - b_n)/(n()+1))

user_days <- edx %>%
  mutate(days_since=as.numeric(
    as.Date("2020-04-30") -
      as.Date(as.POSIXct(timestamp,
                         origin="1970-01-01")))) %>%
  group_by(userId) %>%
  summarize(user_days_since=round(max(days_since),-1))

user_day_avgs <- edx %>%
  left_join(user_days, by='userId') %>%
  left_join(movie_days, by='movieId') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(pop_avgs, by='num_ratings')%>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  group_by(user_days_since) %>%
  summarize(b_ud = sum(rating - mu - b_i - b_u - b_g
                       - b_y - b_n - b_md)/(n()+1))

predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by='year') %>%
  left_join(movie_pop, by='movieId') %>%
  left_join(pop_avgs, by='num_ratings') %>%
```

```
  left_join(movie_days, by='movieId') %>%
  left_join(movie_day_avgs, by='movie_days_since') %>%
  left_join(user_days, by='userId') %>%
  left_join(user_day_avgs, by='user_days_since') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y
         + b_n + b_md + b_ud) %>%
  pull(pred)


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

Results

The modelling approach adopted in this study has been to progressively identify and assess the sources of bias in the training data set and build a linear model using those bias terms. Having done so, the terms were regularized to account for the uncertainty of ratings arrived at with a very small sample size. The results are tabulated here:

model	RMSE
$Y_{u,i} = \mu + \varepsilon_{u,i}$	1.0599040
$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$	0.9437429
$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$	0.8659320
$Y_{u,i,g} = \mu + b_i + b_u + b_g + \varepsilon_{u,i,g}$	0.8655941
$Y_{u,i,g,y} = \mu + b_i + b_u + b_g + b_y + \varepsilon_{u,i,g,y}$	0.8654189
$Y_{u,i,g,y,n} = \mu + b_i + b_u + b_g + b_y + b_n + \varepsilon_{u,i,g,y,n}$	0.8652957
$Y_{u,i,g,y,n,md} = \mu + b_i + b_u + b_g + b_y + b_n + b_{md} + \varepsilon_{u,i,g,y,n,md}$	0.8652373
$Y_{u,i,g,y,n,md,ud} = \mu + b_i + b_u + b_g + b_y + b_n + b_{md} + b_{ud} + \varepsilon_{u,i,g,y,n,md,ud}$	0.8651212
$Y_{u,i,g,y,n,md,ud} = \mu + b_i + b_u + b_g + b_y + b_n + b_{md} + b_{ud} + \varepsilon_{u,i,g,y,n,md,ud}$ (Regularized)	0.8645087

The final result acheived by applying the model to the validation data was an RMSE of **0.8640077**.

We can see that systematically and incrementally identifying sources of bias and adding each one as a term of a linear model - the approach advocated by Chen, is an effective method to drive incrementally improving results.

We can also see that regularization is a critical step that adds substantially to the accuracy of the model.

Conclusion

In this paper we discussed a model building approach for building a movie recommendation system that focused on systematically and incrementally identifying sources of bias in the data and adding them as terms to a linear model. Together with regularization of the terms in the model to elimate uncertainty due to movies or users with low numbers of reviews, this approach has proven to be very effective.

It is certainly possible to improve on this work. Due to the effectiveness of the initial approach selected, more computationally costly approaches, such as calculating SVD or PCAs for a sparse matrix transformation of the data were not explored. Similarly, no attempt to apply more sophisticated non-linear models to the data was undertaken.

It may well be that the application of an ensemble of non-linear models to the data, using the biases identified in this work as predictors would produce even better results. It is the author's belief that this would be a promising avenue for future investigations.