MovieLens recommender system project for course PH125.9x

Mike Woodward

2021-01-06

Contents

1 Executive summary **1**

2 Introduction **2**
2.1 Recommender systems 2
2.2 The project 2
2.3 Limitations 2

3 Methods and analysis **2**
3.1 Test, train, and holdout data 2
3.2 Exploratory data analysis 3
3.3 Model building and training - theory 8

4 Modeling results **10**

5 Regularization **10**

6 Final results **11**

7 Discussion and conclusions **11**

8 References **12**

1 Executive summary

I have submitted this report as part of the requirements for the HarvardX PH125.9X Capstone course. It explains how I built a recommender system to predict the ratings users gave to movies in the MovieLens data set.

The goal of the project was an RMSE (root mean square error) of less than 0.8649. The final RMSE I achieved was 0.8642942, with clipping reducing the RMSE still further to 0.8641097.

This is a large data set, so the computational burden is high. To find a reasonable result with available computing resources, I used a linear modeling method and considered the following features:

- Movie effects
- User effects
- Genre effects
- Movie release year effects
- Duration between movie release year and movie review year

I also considered review date but rejected it as a feature because it offered too little predictive power. To avoid overfitting, I used regularization.

2 Introduction

2.1 Recommender systems

The goal of recommender systems is clear: predict the ratings or choice a user would make and recommend that choice to them. For example, make a recommendation of a jacket size and style based on a user's prior clothing purchases [Winecoff].

Recommender systems are widely used by internet companies, covering everything from movie recommendations [Gomez-Uribe], to music [Jacobson], to clothing fashion and fit [Winecoff]. They've become so commercially important, there's at least one conference series dedicated to them (ACM Conference Series on Recommender Systems) and there are multiple books explaining how to build them (e.g. [Schrange], [Aggarwal], [Ricci]).

2.2 The project

The goal of this project was to build a recommender system based on a subset of the MovieLens movie review data. The recommender systems was to predict users' ratings of movies based on features in the data, for example, movie IDs, user IDs, genres, review times etc.

The MovieLens data was collected as part of the social computing research work at the University of Minnesota and is available from the movielens site. The data contains numeric reviews for movies by reviewer and has other fields such as the movie title, movie genre, and review timestamp.

2.3 Limitations

The main project limitation was computational power. Commercial recommender systems typically use cloud computing clusters [Sun], that are both time consuming and expensive to setup and run - which rules them out for this project. I have access to a 2018 high-spec Mac Book Pro, which I used for this project. Given this limitation, I choose to use less computationally demanding methods, specifically, linear modeling. The final project takes 12 minutes 38.3 seconds to run, the majority of which is consumed in regularization.

As required by the project instructions, this project uses the 10 million review subset. The full dataset has 27 million ratings. It's quite possible that other algorithms may give better results for the full dataset than those used here.

3 Methods and analysis

3.1 Test, train, and holdout data

The MovieLens data I used for this project consists of 10 million reviews. I found no missing (NA) fields. After initial cleaning and formatting, I partitioned this data into two groups using randomization (with a seed of 1):

- edx - 90% of the data. This data was used for training and regularization. All of the results in the 'Exploratory data analysis', 'Model building and training', and 'Regularization' sections of this report use this data *only*.
- validation - 10% of the data. This is a holdout data set and was *only* used for the final model evaluation. The only section in this report that uses this data is the 'Final results' section.

For model development and testing, I partitioned the edx group into two sub-groups (again using randomization and a seed of 1):

- train - 90% of the data - used for training algorithms.
- test - 10% of the data - used for testing the trained algorithms.

This process will result in some users and movies appearing in the validation set but not in the edx set (similarly for the train and test sets). To prevent this happening, I add 'missing' users and movies back to the edx and train data sets. This gives a split that isn't quite 90%:10% but it's necessary for the modeling process to work.

3.2 Exploratory data analysis

The edx data is a random subset of the full data set, with some movie and user data added back. This means exploratory analysis on the edx data set will be representative of the data set as a whole.

A view of the head of the edx data frame shows the following fields:

userId	movieId	timestamp	title	genres	rating
1	122	838985046	Boomerang (1992)	Comedy|Romance	5
1	185	838983525	Net, The (1995)	Action|Crime|Thriller	5
1	292	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	5
1	316	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	5
1	329	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	5
1	355	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	5

Note the movie release year is coded into the title and is always the last part of the title. The timestamp field is the UNIX-formatted timestamp when the user submitted their movie review. The genres field contains multiple values separated by a '|' character. Although there are six fields in the data set, the fact the movie title includes the release year suggests I should create a seventh field by extracting the release year from the movie title.

The movie ratings are the outcomes I want to predict, so I'll explore ratings first. Here are the sorted unique ratings:

0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5

Note the range of values (0.5 to 5) and the steps (0.5); a rating of 2.75 for example, does not occur in this data set.

I summarized the edx data as shown below:

no_users	no_movies	no_ratings	mean_rate_user	mean_rate_movie
69878	10677	9000055	128.7967	842.9386

The summary shows there are a large number of users, movies, and ratings. This suggests the modeling computational burden will be high. The high average number of ratings per user and ratings per movie indicates I should look closely at the underlying distributions, there may be some structure relevant to modeling.

The most obvious distribution is the number of reviews by rating, shown in Figure 1 (below). The quantization and the upper and lower bounds are obvious, but it's also obvious that the distribution is *not* uniform, which suggests there are underlying factors that affect ratings. Note the most common rating (mode) is 4 and that integer ratings are more common than half scores. I'll return to this topic in the 'Discussion and conclusions' section.

I show distribution of mean ratings by movie in Figure 2. This suggests a very strong movie effect on ratings and so it's a feature I need to include in my model. The distribution appears to be left-skewed compared to a

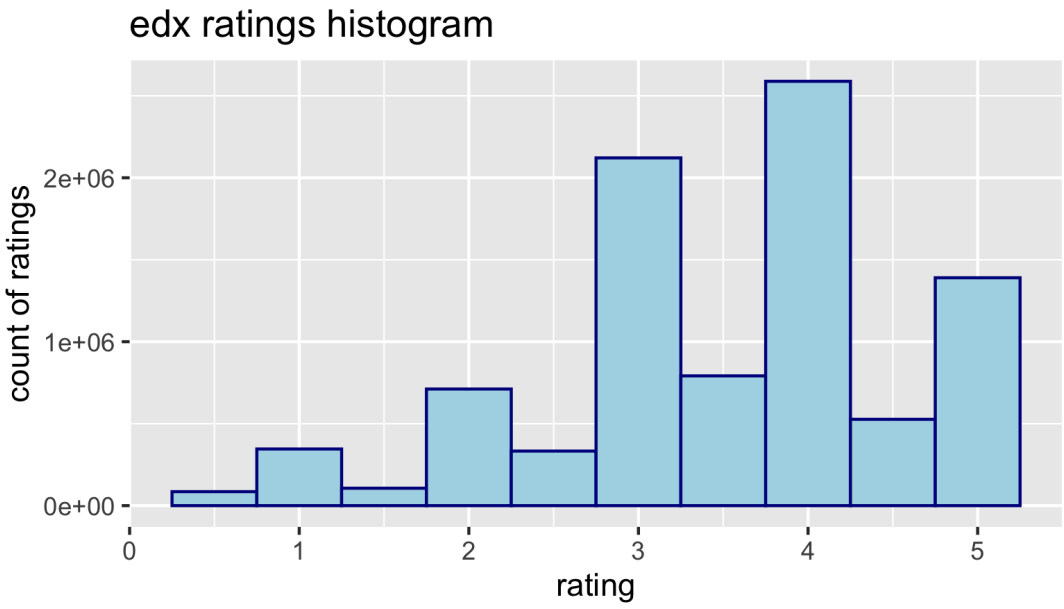

Figure 1: Ratings histogram

normal distribution.

Figure 3 plots the user count vs mean rating. Again, there's a very strong user effect on rating and as a result, I need to include this feature in my model. Note that it's an approximately normal distribution and its shape is different from the movie effect, suggesting movie and user effects have different dynamics.

Genres are a little harder to analyze because a movie can have multiple genres associated with it. If we look just at individual genres, we find these genres:

(no genres listed), Action, Adventure, Animation, Children, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, IMAX, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western

However, they can be combined in several ways. In the edx data, there are 797 unique values for the genres field. As a first approximation, it seems reasonable that combined genres (e.g 'Action | Adventure') may behave differently from the individual genres (e.g. 'Action', 'Adventure'). For now, I'm going to confine my analysis to combined genres, specifically, the contents of the genres column.

In Figure 4, I've plotted the mean rating and 95% confidence interval for all genres that have more than 50,000 reviews. I've ordered the data by decreasing mean rating. Obviously, there's a strong genre effect with a range of mean ratings from 4 down to about 2.8

This is just the top reviewed movies, but what about movies with less than 50,000 reviews? There are 797 genres, which means I can't display the genre names on the axis. As is obvious from Figure 5 that shows the full data set, there's a strong genres effect, but some genres have a wide confidence interval, which is something I'll come back to in the 'Discussion and conclusions' section.

The movie release year is another possible feature. I extracted release year from the title and plotted the mean rating by year (with 95% confidence interval) below (Figure 6). Ratings are dependent on release year, which is very clear after about 1970. Prior to 1970, the pattern is less obvious and I'll come back to this point in the 'Discussion and conclusion'. For now, I'll add movie release year as another model feature.

The final field in the data set is the review timestamp. To make timestamp data a little easier to analyze, I grouped it by week and plotted the mean rating for all movies reviewed in that week (Figure 7). Although there is an effect, it's smaller than the other effects I've seen so far. I decided to exclude it from my modeling work because of its low variation.

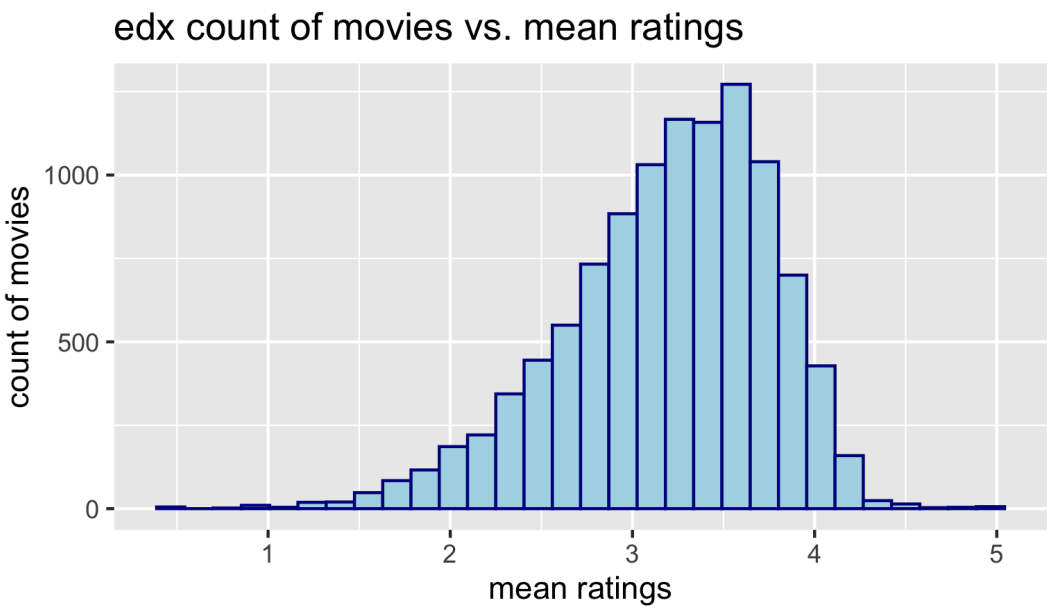

Figure 2: Movie ratings histogram

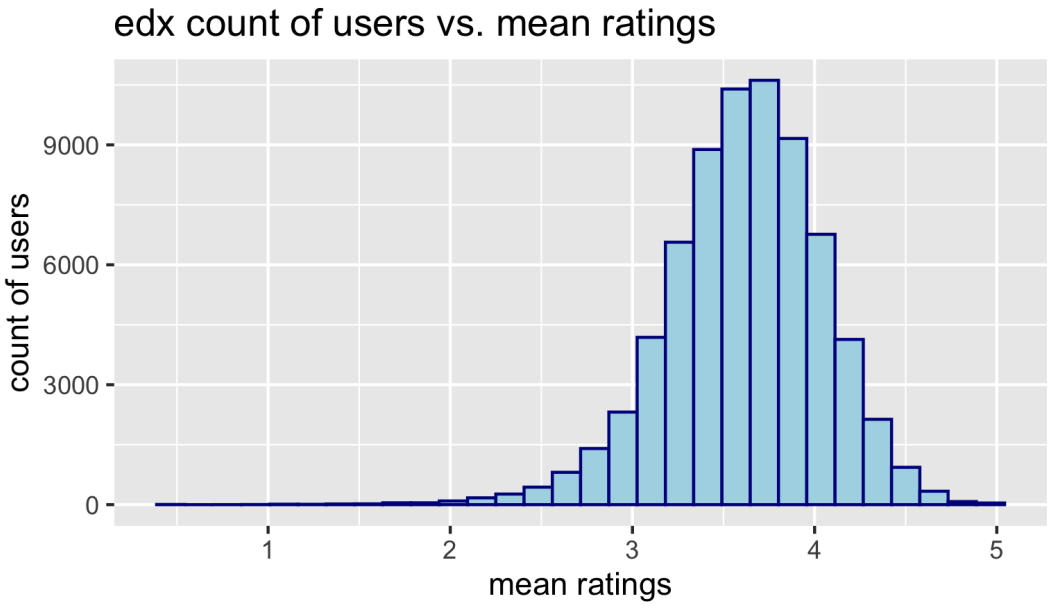

Figure 3: User ratings histogram

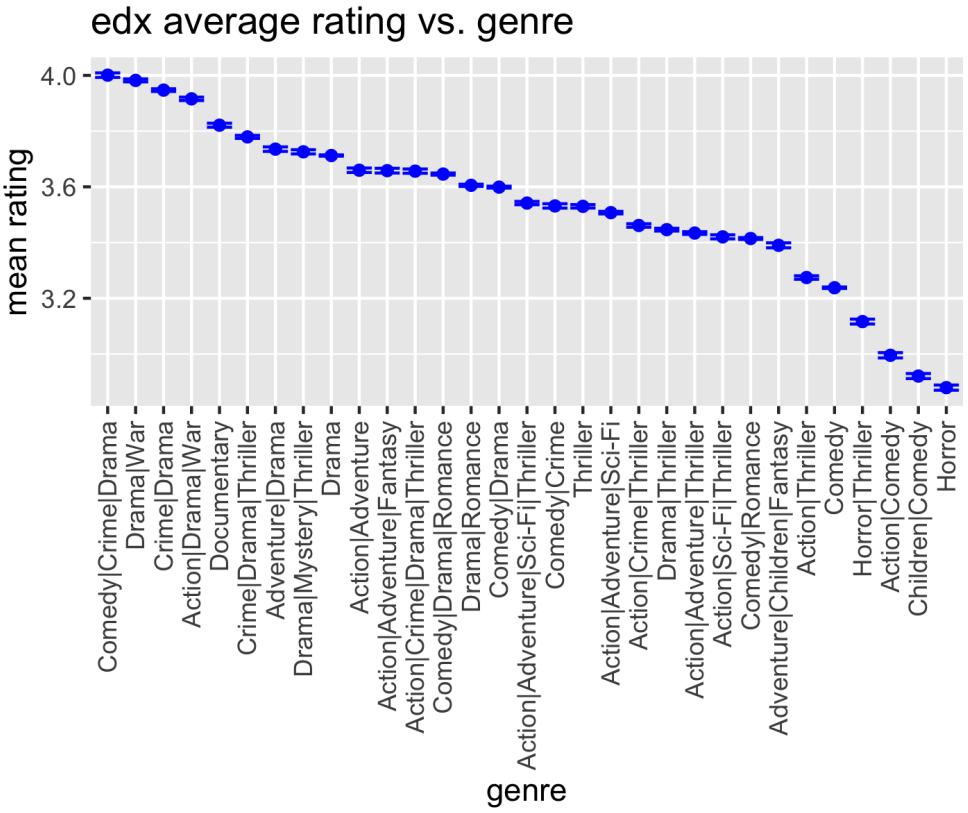

Figure 4: Genres ratings histogram - 50,000 ratings and more

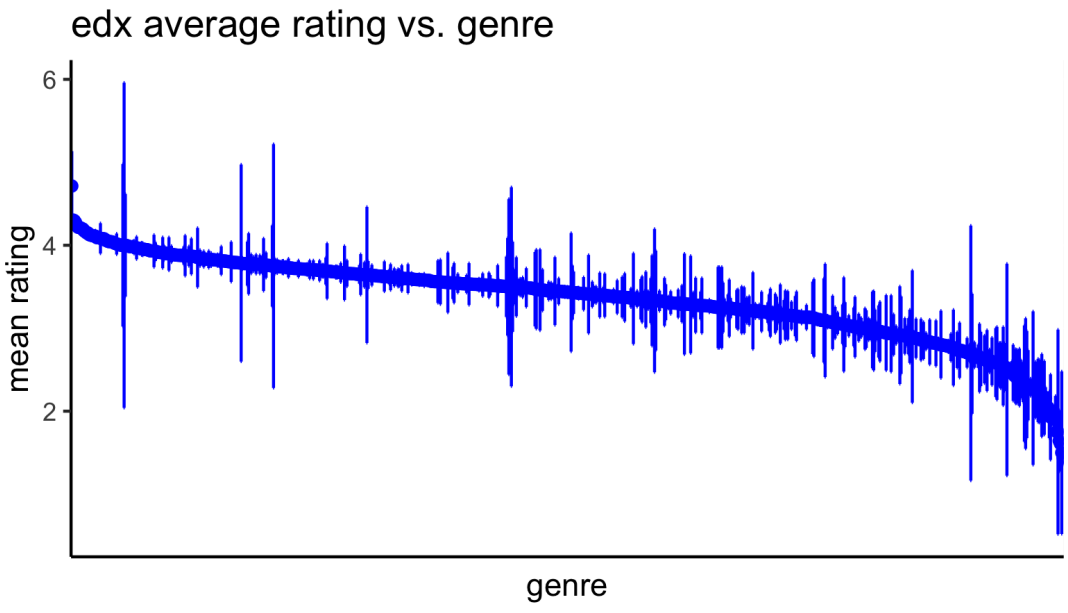

Figure 5: Ratings histogram - full genres set

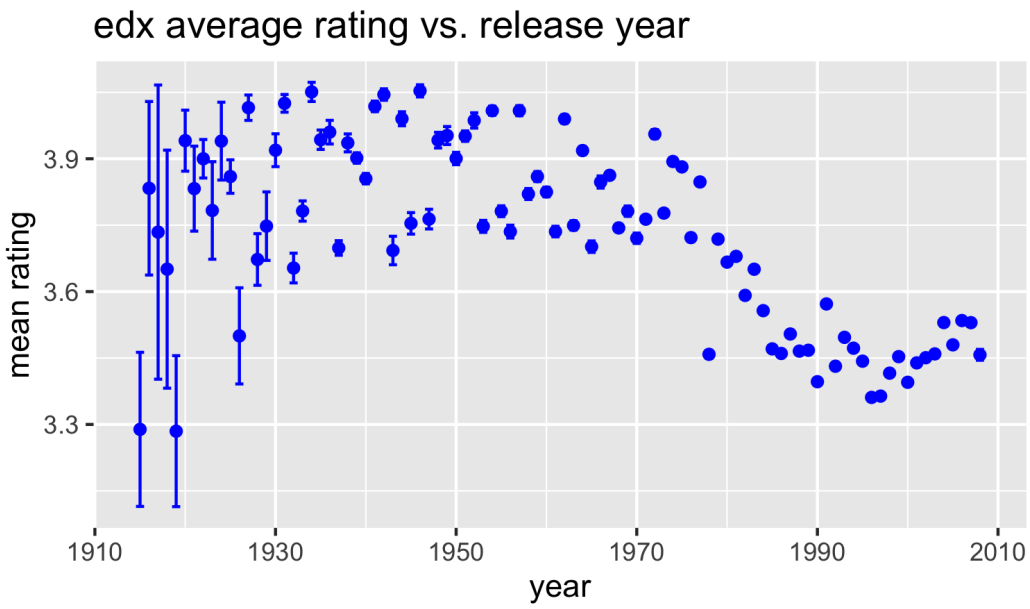

Figure 6: Ratings mean and confidence interval by movie release year

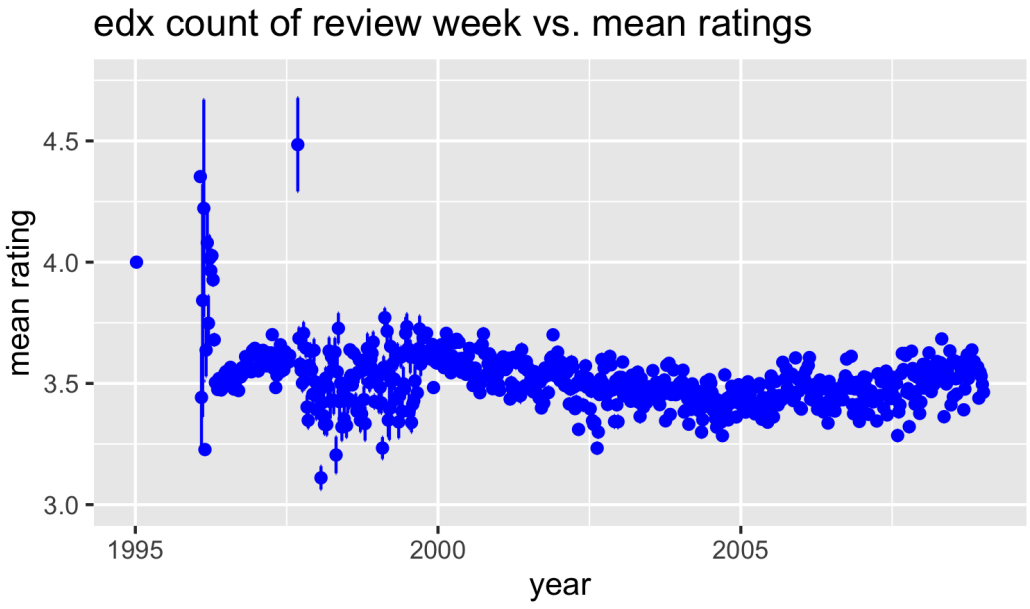

Figure 7: Ratings mean and confidence interval by review week

Although the timestamp of the review showed little variation, it's possible that the combination of release year and review time may be important. It's possible that older movies get better reviews because people may view them based on their pre-existng reputation. I calculated a feature called review_release which is $(year\ of\ review) - (year\ of\ movie\ release)$ and plotted mean rating against review_release (Figure 8).

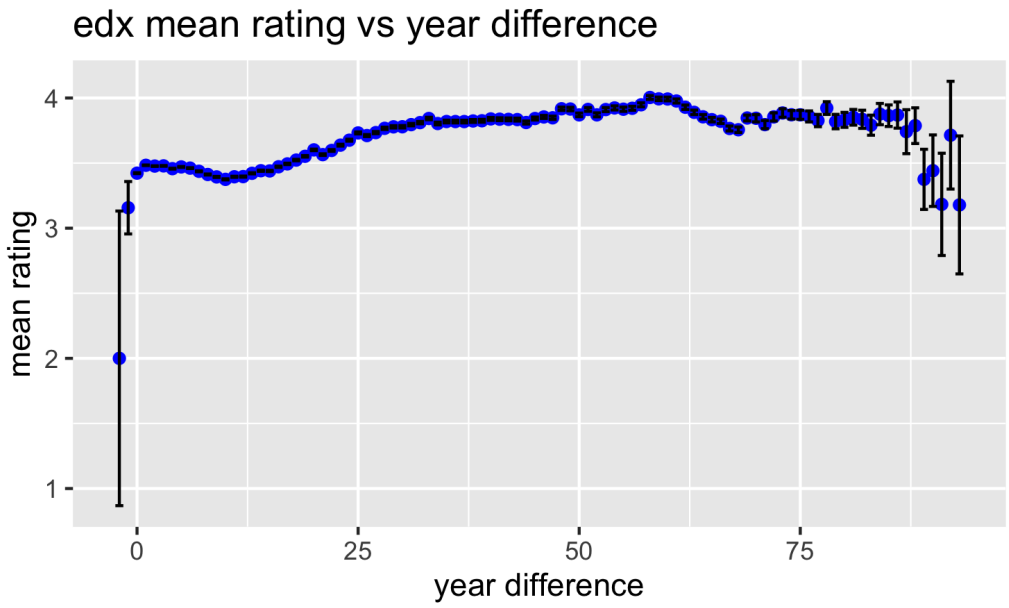

Figure 8: Ratings mean and confidence interval by release review

Note that there are a few cases where release_review (the year difference) is negative, suggesting the movie was reviewed before it was released. This does happen from time to time (a movie might be made available for early review in December but released to the public in January, and sometimes rough cuts are screened well before release to gauge audience reactions). For now, I'll leave these reviews in the data set.

The chart shows that, as expected, older movies have better reviews. I will include this feature in my model.

My analysis has suggested the following model features:

- movie
- user
- genres
- release_review (release year - review year)
- release year

My analysis also suggests that review timestamp, on its own, will not explain much ratings variation.

3.3 Model building and training - theory

Investigating appropriate modeling approaches takes time. With a large data set and limited computational resources (no cloud cluster for example), I need to select a relatively straightforward model and work to reduce the computation time. I therefore chose a linear model and took advantage of some computational short-cuts.

Given my four features, my linear prediction model is:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k \ + f(rr_{u,i}) + f(ry_{u,i}) + \epsilon_{u,i}$$

where:

- $Y_{u,i}$ is the rating prediction for movie i from user u.
- μ is the mean rating for all movies and all users.
- $\epsilon_{u,i}$ is the error (assuming a center of zero). I'll come back to this in the discussion.
- b_i is the mean rating for movie i.
- b_u is the mean rating for user u.
- $\sum_{k=1}^{K} x_{u,i}^k \beta_k$ where $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k. This represents the mean genre rating for genre k. For brevity, I will label this term b_g.
- $f(rr_{u,i})$ - where f is a smooth function of $rr_{u,i}$. This represents the mean rating for my feature release review. Again for brevity, I will label this term b_{rr}.
- $f(ry_{u,i})$ - where f is a smooth function of $ry_{u,i}$. This represents the mean rating for the release year feature. I will label this term b_{ry}.

Using linear modeling for a data set this large is computationally expensive, there is however a a shortcut I can take. I can write the estimate of b_i as:

$$\hat{b}_i = \frac{1}{N} \sum_{i=1}^{N} Y_{u,i} - \hat{\mu}$$

this is cheaper and easier to compute than using linear modeling in the caret package. But it's not just b_i I can calculate in this way, I can similarly calculate the other terms, for example:

$$\hat{b}_u = \frac{1}{N} \sum_{i=1}^{N} Y_{u,i} - \hat{\mu} - \hat{b}_i$$

My code is a straightforward implementation of this algorithm. For example, here's a code snippet to calculate the genres effect.

```
# Genres
b_g <- train %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u),
            .groups='keep')
predicted_ratings <- test %>%
  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) %>%
  pull(pred)
rmse_genres = RMSE(test$rating, predicted_ratings)
model_scores <- rbind(scores, c("Genres", rmse_genres))
```

I'll use the loss function RMSE as my metric of success. RMSE is defined by the function

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

where N is the number of recommendations, $\hat{y}_{u,i}$ is the predicted rating and $y_{u,i}$ is the actual rating. This is the course mandated measure.

If this approach is valid, we would expect to see the RMSE decrease as we build the model up from its components parts.

4 Modeling results

I modeled the data in six stages using the train data subset:

- Mean only model
- Mean and movie effects
- Mean, movie effects, and user effects
- Mean, movie effects, user effects, and genre
- Mean, movie effects, user effects, genre, and release review effects.
- Mean, movie effects, user effects, genre, release review, and release year effects.

At each stage, I calculated the RMSE using the train and test subset of the edx data set.

Here are the RMSE result for each stage of the model building process.

method	rmse
Naive means	1.06005370148846
Movie effects	0.941723874535076
User effects	0.855819325070824
Genres	0.855463119464715
Release review	0.855065119111752
Release year	0.854888523494502

As expected, adding more features decreases RMSE, taking me below the course target of 0.8649 However, I may have overfitted the model. To avoid overfitting, I need to regularize my model prior to the final analysis.

5 Regularization

Regularization means choosing a penalty term to constrain effect size variability. My penalty term is λ and I select it by minimizing this function using cross validation:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_g - b_{rr} - b_{ry})^2 + \lambda\left(\sum_i b_i^2 + \sum_u b_u^2 + \sum_g b_g^2 + \sum_{rr} b_{rr}^2 + \sum_{ry} b_{ry}^2\right)$$

Regularization is computationally expensive; to find a minimum λ means varying lambda to minimize RMSE. There are various computational methods to do this, for example, the well-known Newton-Raphson process, but for now, I'll vary the lambda step size manually, with smaller steps around the minima and larger steps elsewhere.

Here's a code snipped to show how I implemented the logic.

```
lambdas <- sort(c(seq(0,     0.1, 0.1),
                  seq(0.2,   0.6, 0.025),
                  seq(0.62,  0.8, 0.2),
                  seq(0.825, 4,   0.25)))

# Function to apply lambdas to regularization
rmses_regularization <- sapply(lambdas, function(lambda) {

  mu <- mean(train$rating)

  ... Code removed

  # Release year
  b_ry <- train %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    left_join(b_rr, by='review_release') %>%
    group_by(release_year) %>%
    summarize(b_ry = sum(rating - mu - b_i - b_u - b_g - b_rr)/(n() + lambda),
              .groups='keep')

  # Predicted rating
  predicted_ratings <- test %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    left_join(b_rr, by='review_release') %>%
    left_join(b_ry, by='release_year') %>%
    mutate(pred = mu + b_i + b_u + b_g + b_rr + b_ry) %>%
    pull(pred)

  RMSE(test$rating, predicted_ratings)
})
```

Figure 9 is a plot of RMSE vs lambda. The minimal lambda from the chart is 0.325.

6 Final results

The final stage is using the entire edx data to train the model, use the lambda value previously obtained, and use the validation (holdout) data to calculate the final RMSE.

The final RMSE value I obtain is 0.8642942, which is less than the course target of 0.8649.

A closer look at the final rating predictions shows there are ratings <0.5 and >5, which are below and above the range of ratings I found earlier. I can 'clip' ratings to a minimum of 0.5 and a maximum of 5. Unsurprisingly, clipping reduces the RMSE still further to 0.8641097.

7 Discussion and conclusions

Although the final results are better than the course target, more work could reduce the RMSE further.

As I pointed out earlier, ratings are quantized and the number of half star ratings is less than the number of whole star ratings. It may be worth investigating whether different factors drive full and half star ratings,

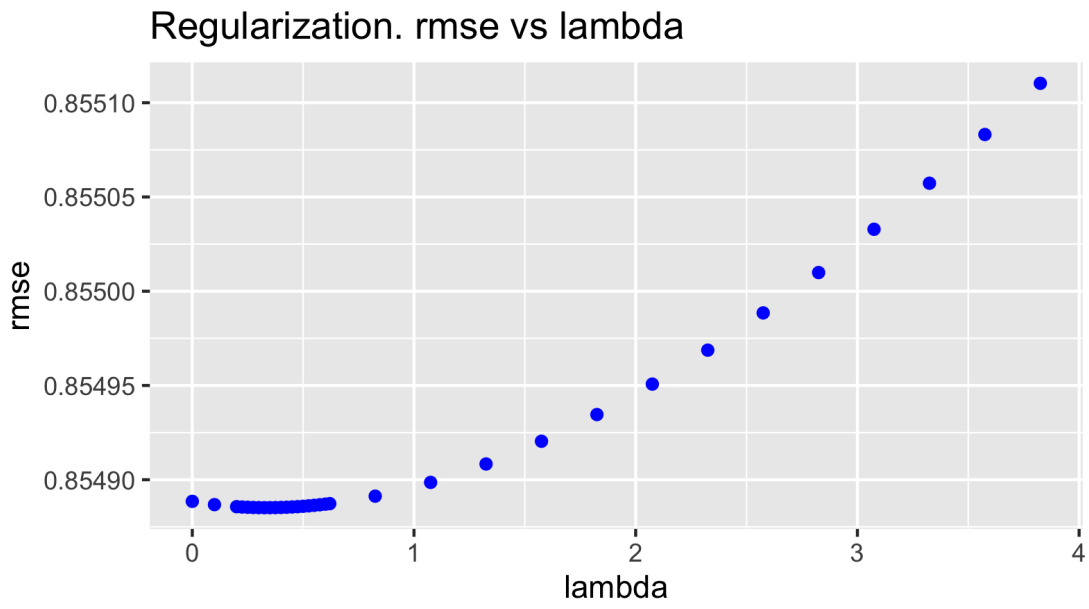

Figure 9: Regularization - minimizing RMSE

for example, some users may rate on full stars alone. This might be a case where k-means clustering could identify different clusters of reviewers.

When modeling genres, I considered *all* genres for a movie. But there may be specific genre effects, for example "Comedy" movies might be more highly reviewed than 'Western' movies. There may well also be an interaction between genre and year, for example, there are fewer 'Western' movies produced now than there were in the 1940's. As chart 5 showed, there are some genres that have a high confidence interval, we might expect our rating to be less accurate in these cases and maybe some conditional logic might be appropriate. A random forest approach may be useful here because of the branching logic.

The release year chart (Figure 6) showed little variation before about 1970. Because of the conditional logic, a random forest approach might work well.

My linear predictive model had a 'error noise' term ϵ with a mean of zero. This might not be the case here; there may be a bias in the noise and of course there are other (non-normal) error models (see for example [Honda]). It might be worth considering error models that are biased in some way.

Although an ensemble model is attractive, in practice the computational burden may be too high for very large datasets. More generally, for a large-scale system, the computational cost must be borne in mind, which suggests we should favor computational cheaper algorithms as the way forward.

8 References

[Aggarwal] Charu C. Aggarwal, Recommender Systems: The Textbook, Springer, 2016

[Gomez-Uribe] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. ACM Trans. Manage. Inf. Syst. 6, 4, Article 13 (January 2016)

[Honda] Testing the Error Components Model with Non-Normal Disturbances Yuzo Honda, The Review of Economic Studies, Volume 52, Issue 4, October 1985, Pages 681–690

[Jacobson] Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. 2016. Music Personalization at Spotify. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). Association for Computing Machinery, New York, NY, USA, 373

[Ricci] Francesco Ricci, Lior Rokach, Bracha Shapira, Recommender Systems Handbook, Springer, 2015

[Schrange] Michael Schrange, Recommendation Engines, MIT Press, 2020

[Winecoff] Amy A. Winecoff, Florin Brasoveanu, Bryce Casavant, Pearce Washabaugh, and Matthew Graham. 2019. Users in the loop: a psychologically-informed approach to similar item retrieval. In Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19). Association for Computing Machinery, New York, NY, USA, 52–59