edX MovieLens Project

James Cruikshanks

7/24/2021

Introduction

Using a large database of movie ratings, this project builds a machine learning model that predicts how a user will rate a given movie. The dataset includes 10 million ratings of more than 10,000 movies by over 72,000 users and can be downloaded from https://grouplens.org/datasets/movielens/10m/. Each rating contains 6 pieces of information: the user ID, the movie ID and title, the timestamp of the rating, and the genre(s) of the film. We can see the data structure by looking at the first few rows:

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

Using the features present in the dataset, a model is created which is able to approximate and adjust for the effect of each one. How well each movie is liked by all users, how picky each user appears to be by their movie ratings, how well rated each genre appears to be, and even the week a movie was rated all present predictable patterns. These patterns are explored and exploited to predict how any user will rate any movie. The model performance is evaluated by the root mean square error between the predicted ratings and the actual ratings of a partitioned validation dataset, as determined by the following function:

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

Analysis

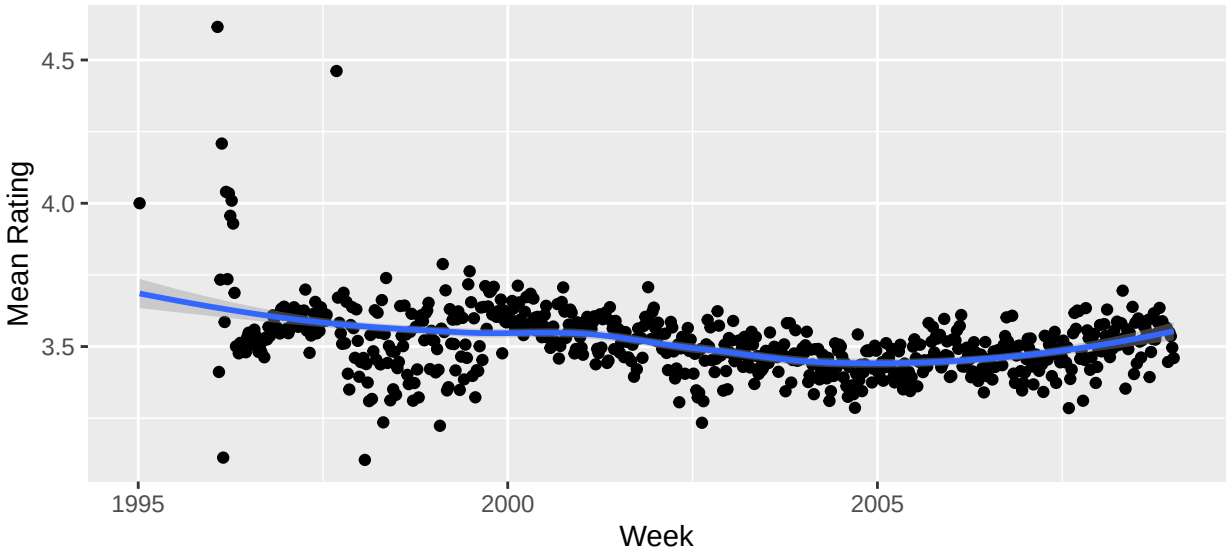
Firstly, the training dataset 'edx' is partitioned into another 'train' set and a 'test' set for various models to be cross-validated while avoiding overfitting of the data to the 'validation' set. 7200045 randomly selected ratings make up the training set and the remaining 1799975 ratings are used for model evaluation.

The prediction of a given user's rating of a given movie might be simply estimated by taking the average of all observed movie ratings. Expanding on this, we take the average of each movie's rating across all users, and each users rating across all movies. Called 'The User + Movie Effect Model', it is calculated as follows:

```r
#calculate the average rating across all movies and users
mean <- mean(train_set$rating)

#calculate and average user rating above or below the mean, per movie (the 'movie effect')
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(movie_eff = mean(rating - mean))

#calculate average movie rating above or below the mean, per user (the 'user effect')
user_avgs <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(user_eff = mean(rating - mean - movie_eff))

#predict ratings using user and movie effects model
prediction <- test_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(pred = mean + user_eff + movie_eff) %>%
  .$pred
```

This model takes into account the effect of the movie, as well as the user. Using this model a RMSE of 0.86553 is achieved

Next, the average of all ratings in a given week is calculated across all of the given time stamps and plotted.



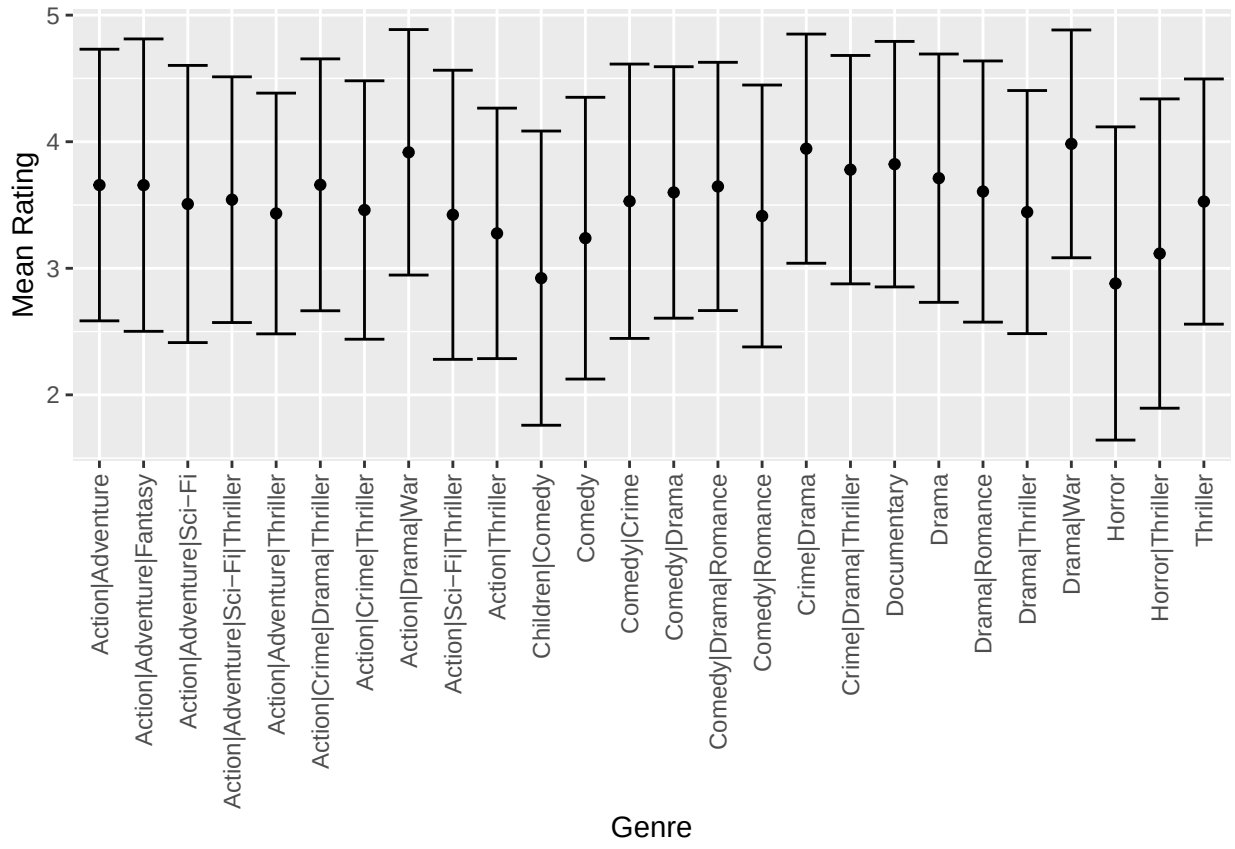
The plot shows a trend that can then be integrated into a new model called 'The User + Movie + Date Effect Model' as follows:

```r
#approximate linear model to calculate the effect of date on user, movie rating
date_avgs <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%

  #add column with week of rating
```

```
  mutate(date = as_datetime(timestamp), week = round_date(date, unit = "week")) %>%

  #calculate average movie rating above or below the mean for each user and movie, by week
  group_by(week) %>%
  summarize(date_eff = mean(rating-mean-user_eff-movie_eff))

##predict ratings using user, movie, and date effects model
#calculate week from test set timestamps
test_set_date <- test_set %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "week"))

#calculate predicted rating
prediction <- test_set_date %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(date_avgs, by = 'week') %>%
  mutate(pred = mean + user_eff + movie_eff + date_eff) %>%
  .$pred
```

Using this model a RMSE of 0.86545 is achieved, a slight improvement.

Next, the average rating by genre is calculated and plotted with the standard error for genres with over 50000 ratings.

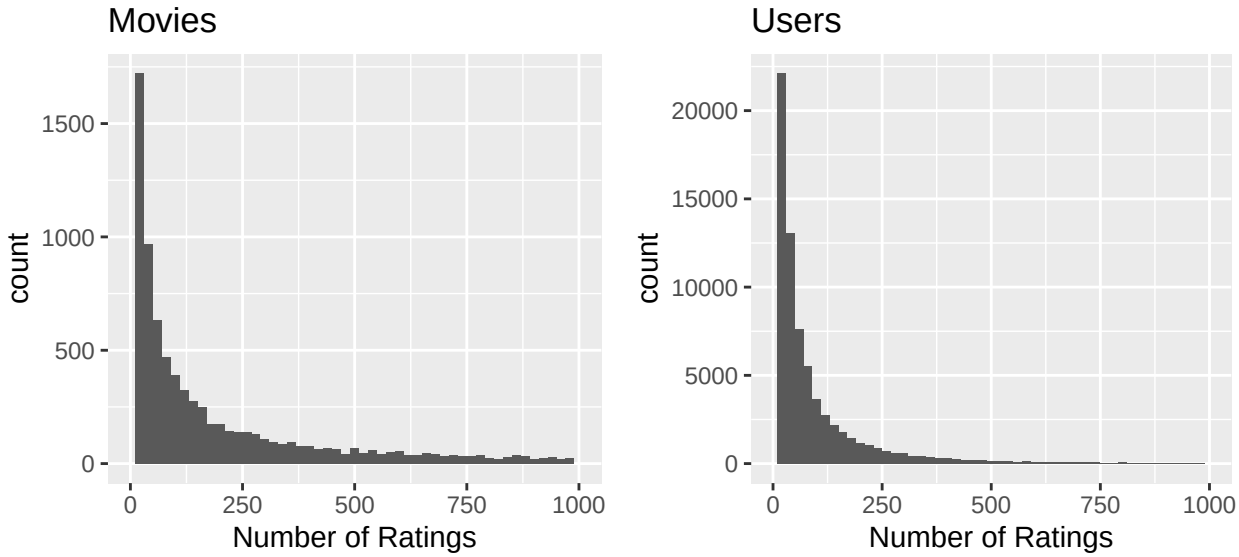


There is variation evident that may be modeled by calculating the average difference for each group of genres, after accounting for the other three effects modeled previously. The model is calculated as follows:

```
#calculate average impact of genre on user rating of movie
genre_avgs <- train_set %>%

  #add column with week of rating
  mutate(week = round_date(as_datetime(timestamp), unit = "week")) %>%

  #add user, movie, and date effects
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(date_avgs,by = 'week') %>%

  #calculate average movie rating above or below the mean for each user and movie, by genre
  group_by(genres) %>%
  summarize(genre_eff = mean(rating-mean-user_eff-movie_eff-date_eff))

#calculate predicted rating
prediction <- test_set_date %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(date_avgs, by = 'week') %>%
  left_join(genre_avgs, by = 'genres') %>%
  mutate(pred = mean + user_eff + movie_eff + date_eff + genre_eff) %>%
  .$pred
```

'The User + Movie + Date + Genre Model' achieves a RMSE of 0.86513, another improvement. Since the available features have been considered in a simple and consistent manner, we move to investigating the predicted results.

The first insight is that most movies and users have small sample sizes.

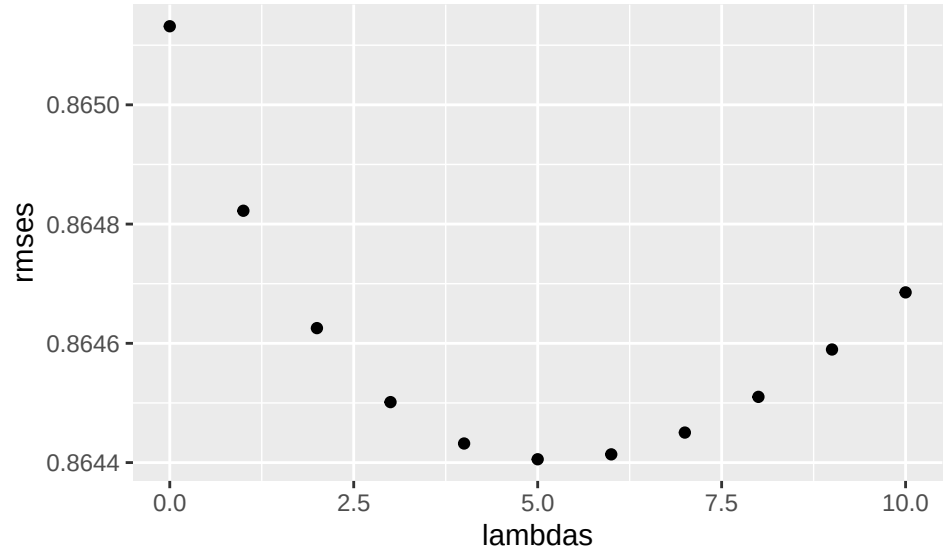


These can lead to overconfident high or low predictions, when a more conservative estimate closer to the mean rating of all movies would be preferable. To model this outcome we regularize each of the four effects, which is calculated as follows:

```
##regularize effects to be conservative when estimating based on small sample sizes
#calculate movie effect with each lambda
```

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(user_eff = sum(rating - mean)/(n()+l))

#calculate user effect with each lambda
user_avgs <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(movie_eff = sum(rating - mean - user_eff)/(n()+l))

#calculate date effect with each lambda
date_avgs <- train_set %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(movie_avgs, by = 'movieId') %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "year")) %>%
  group_by(week) %>%
  summarize(date_eff = sum(rating-mean-user_eff-movie_eff)/(n()+l))

#calculate genre effect with each lambda
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "year")) %>%
  left_join(date_avgs, by = 'week') %>%
  group_by(genres) %>%
  summarize(genre_eff = sum(rating-mean-user_eff-movie_eff-date_eff)/(n()+l))

#calculate predictions
prediction <- test_set %>%
  mutate(week = round_date(as_datetime(timestamp), unit = "year")) %>%
  left_join(movie_avgs, by = 'movieId') %>%
  left_join(user_avgs, by = 'userId') %>%
  left_join(date_avgs, by = 'week') %>%
  left_join(genre_avgs, by = 'genres') %>%
  mutate(pred = mean + user_eff + movie_eff + date_eff + genre_eff) %>%
  .$pred
```

Lambda can be tuned using cross validation. The results of this are plotted below, where the optimal lambda of 5 produces a model with the lowest RMSE, 0.86441:



The next insight is that the highest predicted rating is 5.96769 and the lowest is -0.62462, which is guaranteed to result in an error on a scale of 0-5. Adding a floor of 0 and a ceiling of 5 to the predictions is sure to improve the model.

Results

'The Regularized User + Movie + Date + Genre Model' achieves an RMSE of 0.86441 using the training and test sets partitioned for cross-validation. With a consistent and robust model utilizing all the available features of the dataset, this machine learning model is selected as the final model. Next, the model is trained on the full 'edx' training set and tested on the 'validation' set which so far has not been incorporated in any of the analysis.

The results show an RMSE of 0.8642 using 'The Final Model'. The additional data improves the model accuracy for the lowest RMSE score seen so far in the analysis. All model results so far are summarized in the table below:

Models	RMSE
The Movie + User Model	0.86553
The Movie + User + Date Model	0.86545
The Movie + User + Date + Genre Model	0.86513
The Regularized User + Movie + Date + Genre Model	0.86441
The Final Model	0.86420

Conclusions

This report demonstrates the capability of even simple methods using machine learning on large datasets. The MovieLens data contains 10 million observations which were used to predict how any users in the dataset would rate any of the movies. Using linear models of the various features provided in the data led to an algorithm which is capable of predicting, within 0.8642, a users rating from 0-5.

Future work is plentiful. The model could search online databases for much more information on each of the movies rated, finding any number of insights using the properties of each film. The user effect could be expanded by grouping users by similar preferences, potentially increasing the accuracy of ratings by

accounting for similar tastes between users. All of this work, and likely much more, has been completed and is available in the public domain for study and reference.