Data Science: Capstone Project Movielens

Hannes Windisch

1.6.2020

Contents

1 Overview

Recommendation systems use ratings that users have given to items to make specific recommendations. The goal of this project is to build a predictive model for more than 10 million movie ratings based on a dataset provided by the University of Minnesota.

First the given data set is be prepared and made tidy. After that some exploratory data analysis is carried out to get a better feeling for the underlying data.

Before machine learning algorithms can be applied, the data set has to be divided into a training and a test data set. Then several machine learning algorithms are evaluated and depending on their efficiency the best model is chosen.

As the final step this final model is used to make predictions against the test data set. The resulting Root Mean Squared Error (RMSE) will show the quality of the algorithm performance.

1.1 Model Evaluation

The model evaluation criterion used is the Root Mean Squared Error (RMSE), which captures the deviation of the predicted values from the actual values. A lower RMSE is better than a higher one. If this number is larger than 1, it means our typical error is larger than one movie rating star.

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

This R function is used to calculate RMSE:

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

1.2 Data set

The MovieLens data set is downloaded here: http://files.grouplens.org/datasets/movielens/ml-10m.zip

In this step the loaded data is prepared and tided and split into 2 subsets: The training set *edx* and the test set *validation*. The different algorithms are trained with the training set and finally tested against the validation set.

2 Methods and Analysis

2.1 Exploratory data analysis

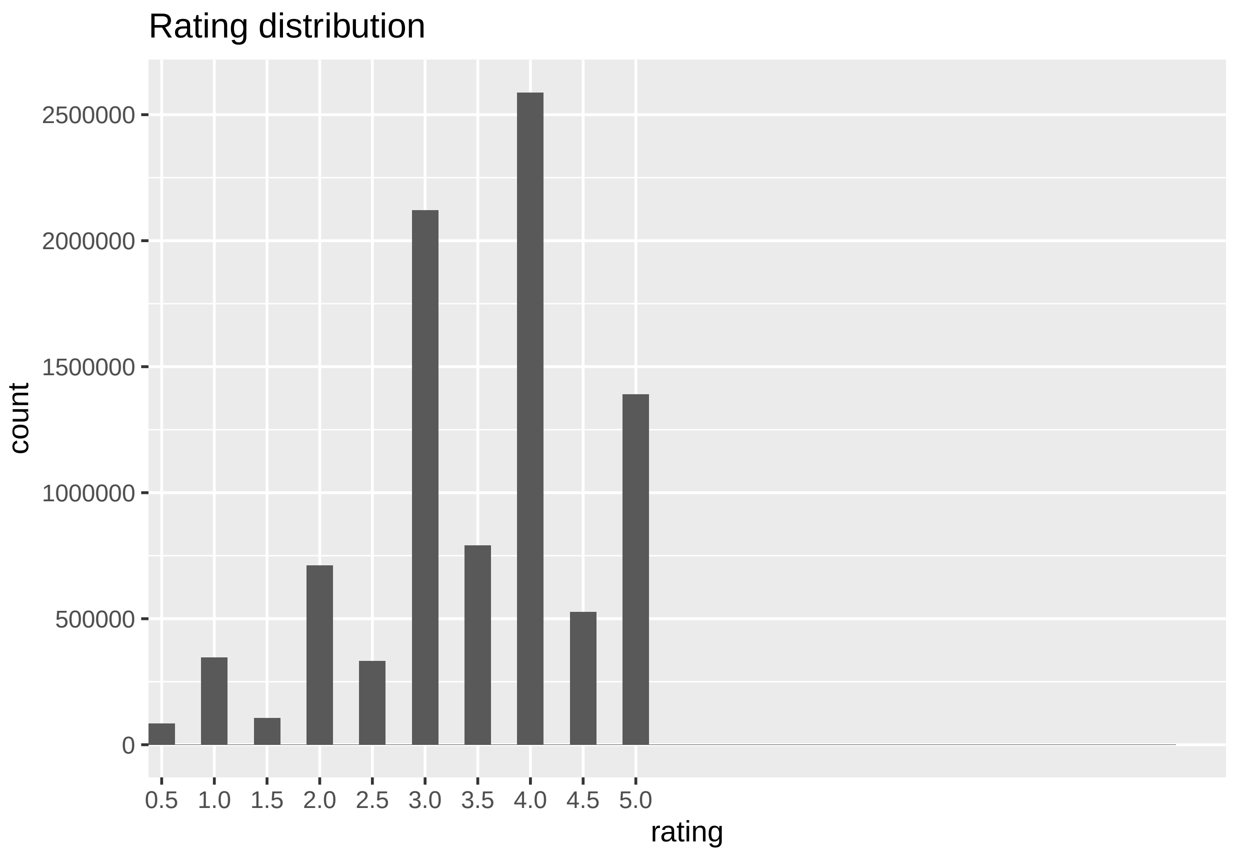
To get familiar with the data set, we make some exploratory data analysis. The subset contains 6 variables: "userID", "movieID", "rating", "timestamp", "title", and "genres". Each row represents a single user/movie rating.

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

The total of unique movies and users in the edx subset is about 70.000 unique users and about 10.700 different movies:

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

Users have a preference to rate movies rather higher than lower as shown by the distribution of ratings below. 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. In general, half rating are less common than whole star ratings.



2.2 Model 0 - Raw mean

Our first approach is a model that assumes the same rating for all movies and users:

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

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

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

```r
rmse_mu <- RMSE(validation$rating, mu)
rmse_mu
```

```
## [1] 1.061202
```

This naive model serves as a baseline for our future models and results in an RMSE of 1.06.

2.3 Model 1 - Movie effects

Some movies are just generally rated higher than others. Higher ratings are mostly linked to popular movies among users and the opposite is true for unpopular movies. We compute the estimated deviation of each movies' mean as variable b_i, that represents average ranking for movie i:

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

We could use the lm() function to calculate the model but because there are thousands of b_i, the lm() function would be very slow.

In this particular situation, we know that the least squares estimate b_i is the average of $Y_{u,i} - \mu$ for each movie i.

```r
m1 <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

pred_1 <- mu + validation %>%
  left_join(m1, by = 'movieId') %>%
  .$b_i

rmse_1 <- RMSE(pred_1, validation$rating)
rmse_1
```

```
## [1] 0.9439087
```

We can see a clear improvement (RMSE = 0.944) but we can still do better because we don´t consider the individual user rating effect yet.

2.4 Model 2 - Movie and User effects

We already considered the movie effect but there is a user effect as well. Some users are very cranky and others love every movie, so we have to take this behavior into our model:

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

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

```r
m2 <- edx %>%
  left_join(m1, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

pred_2 <- validation %>%
  left_join(m1, by = 'movieId') %>%
  left_join(m2, by = 'userId') %>%
  mutate(pred_2 = mu + b_i + b_u) %>%
  .$pred_2

rmse_2 <- RMSE(pred_2, validation$rating)
rmse_2
```

```
## [1] 0.8653488
```

This result is much better again, now we will try to improve this once more using regularization.

2.5 Model 3 - Regularized movie and user effects

The estimates of b_i and b_u are negatively influenced by movies with very few ratings and by users that only rated a very small number of movies. Regularization permits to penalize these cases. In a first step we have to find the right tuning parameter lambda on the training data set that minimize the RMSE:

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l) {
                mu <- mean(edx$rating)
                b_i <- edx %>%
                        group_by(movieId) %>%
                        summarize(b_i = sum(rating - mu)/(n()+l))
                b_u <- edx %>%
                        left_join(b_i, by = "movieId") %>%
                        group_by(userId) %>%
                        summarize(b_u = sum(rating - b_i - mu)/(n()+l))
                pred <- edx %>%
                        left_join(b_i, by = "movieId") %>%
                        left_join(b_u, by = "userId") %>%
                        mutate(pred = mu + b_i + b_u) %>%
                        pull(pred)
                return(RMSE(pred, edx$rating))
                })
```

The optimal lambda is:

```
lambdas[which.min(rmses)]
```

```
## [1] 0.5
```

With this lambda we can initiate the final step, calculating the regularized parameters b_i and b_u with the given lamda and calculate the RMSE on the validation data set:

```
# Calculate regularized parameters bi und bu for optimal RMSE using training set
l <- 0.5
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
b_u <- edx %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))

# Calculate prediction on validation set
pred <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Calculate RMSE on validation set
rmse_3 <- RMSE(pred, validation$rating)
rmse_3
```

```
## [1] 0.8652226
```

Using this optimal lambda the resulting RMSE is better than without regularization.

3 Results

We can see that we accomplished to improve our results with every step. The Regularized movie and user effect model has the lowest RMSE and is therefore the favored method.

method	RMSE
raw_mean	1.0612018
movie effects	0.9439087
movie & user effects	0.8653488
regularized movie and user effects	0.8652226

4 Conclusion

In this project, I have developed and evaluated several methods to build a predictive model for recommending movies. I started with the naive approach and added more detail to the method like the movie effects and the user effects. Both bring big improvements but the best result could be achieved by also regularizing the data.

Future work: There are still columns in the data set like genre that are not part of the algorithm yet. The result could be even better if these columns are integrated as well.