CYO_AHJ

Angus Jenner

09/03/2022

Overview:

This project was to use Collaborative Filtering (CF) methods to build a recommendation system for anime television series. **In the movielens project my laptop had insufficient memory to explore this approach - as a result, all of the analysis performed here is new work.** I was interested in using the recommenderlab package (Michael Hahsler (2016)) to explore these CF approaches. Root mean square error (RMSE) was used to evaluate the different approaches.

The dataset I used included ratings for anime films and television series and included explicit and implicit ratings - implicit ratings were listed as "-1" and were where a user had watched the programme, but had not reviewed it. This initial dataset had 7,813,737 ratings from 73,515 unique users. This volume of data would be too large to handle recommenderlab computations on my hardware and as a result, the dataset was reduced to a manageable size and data cleaning performed.

The recommenderlab package allowed a variety of approaches to be evaluated - these will be discussed in more detail in the results section.

Methods:

Data cleaning:

The initial data cleaning processes performed were to limit the dataset to television series, and to remove the implicit ratings. The justification for removing implicit ratings was that these were not determined by users, and as a result did not reflect their views. There could be many reasons why a user may watch a programme but subsequently not rate it.

```
anime_rating_all <- rating %>%
    left_join(anime, by="anime_id") %>%
    filter(rating.x != -1, type == "TV")
```

Following this, the dataset was reduced to a manageable size. This was done by examining the number of ratings users made, and the number of ratings that different anime series had. Within the dataset there were users who rated very few series, and series that had few ratings. These situations were likely to lead to inaccurate predictions since average ratings would be determined by relatively few ratings. As a result, the chosen way to reduce the dataset was to filter out users who had rated fewer than 200 anime series, and filter out anime series that had fewer than 2000 ratings. It might be said that this level of data reduction - particularly the strict user filter - reduced the dataset to consider only "expert" anime viewers. This process was achieved using the below commands:

```
user_list <- anime_rating_all %>% group_by(user_id) %>%
    summarise(n = n()) %>% filter(n >= 200)
anime_list <- anime_rating_all %>% group_by(anime_id) %>%
    summarise(n = n()) %>% filter(n >= 2000)

anime_rating <- anime_rating_all %>%
    filter(user_id %in% user_list$user_id & anime_id %in% anime_list$anime_id)

## Final dataset ~750,000 ratings
```

A key part of the data preparation process was to change the data from a data frame format into a real ratings matrix format. This step was required as it is this format that is required to run the recommenderlab tools.

Firstly the data frame needed to be reshaped into a matrix with users as the rows, anime series as the columns and ratings as the data filling the matrix. A small subset example (10 users x 10 anime series) is included below:

```
##      6 15 20 22 24 30 45 57 63 67
## 5    8  6  6  5  1  1  7  7  1  6
## 7   NA NA NA  7 NA 10 NA NA NA NA
## 17   7 NA NA NA NA  9 NA NA NA NA
## 38  NA NA  6 NA NA 10 NA NA NA NA
## 43  NA  7  8 NA NA NA NA NA NA  5
## 46  NA NA  7 NA NA NA  8 NA NA NA
## 123 NA NA NA NA NA NA  8 NA NA NA
## 160  7 NA NA NA  9 10 NA NA NA  9
## 210 NA 10 10 NA NA NA NA NA NA NA
## 226  8 NA  7 NA NA NA  8 NA  7  7
```

Secondly, ratings were normalised by removing column averages (i.e. minusing the anime average), and removing the column average (i.e. minusing the average user rating). The updated small subset example (10 users x 10 animes) is included again:

```
##         6    15    20    22    24    30    45   57    63    67
## 5    3.42  1.41  2.05  0.63 -3.46 -3.49  2.29 2.17 -2.23  2.00
## 7      NA    NA    NA -0.92    NA  1.96    NA   NA    NA    NA
## 17  -0.46    NA    NA    NA    NA  1.63    NA   NA    NA    NA
## 38     NA    NA -0.45    NA    NA  3.00    NA   NA    NA    NA
## 43     NA -1.30  0.35    NA    NA    NA    NA   NA    NA -2.71
## 46     NA    NA -1.22    NA    NA    NA -0.99   NA    NA    NA
## 123    NA    NA    NA    NA    NA    NA -0.04   NA    NA    NA
## 160 -1.61    NA    NA    NA  0.51  1.48    NA   NA    NA  0.97
## 210    NA  0.86  1.51    NA    NA    NA    NA   NA    NA    NA
## 226 -0.32    NA -0.68    NA    NA    NA -0.45   NA  0.04 -0.74
```

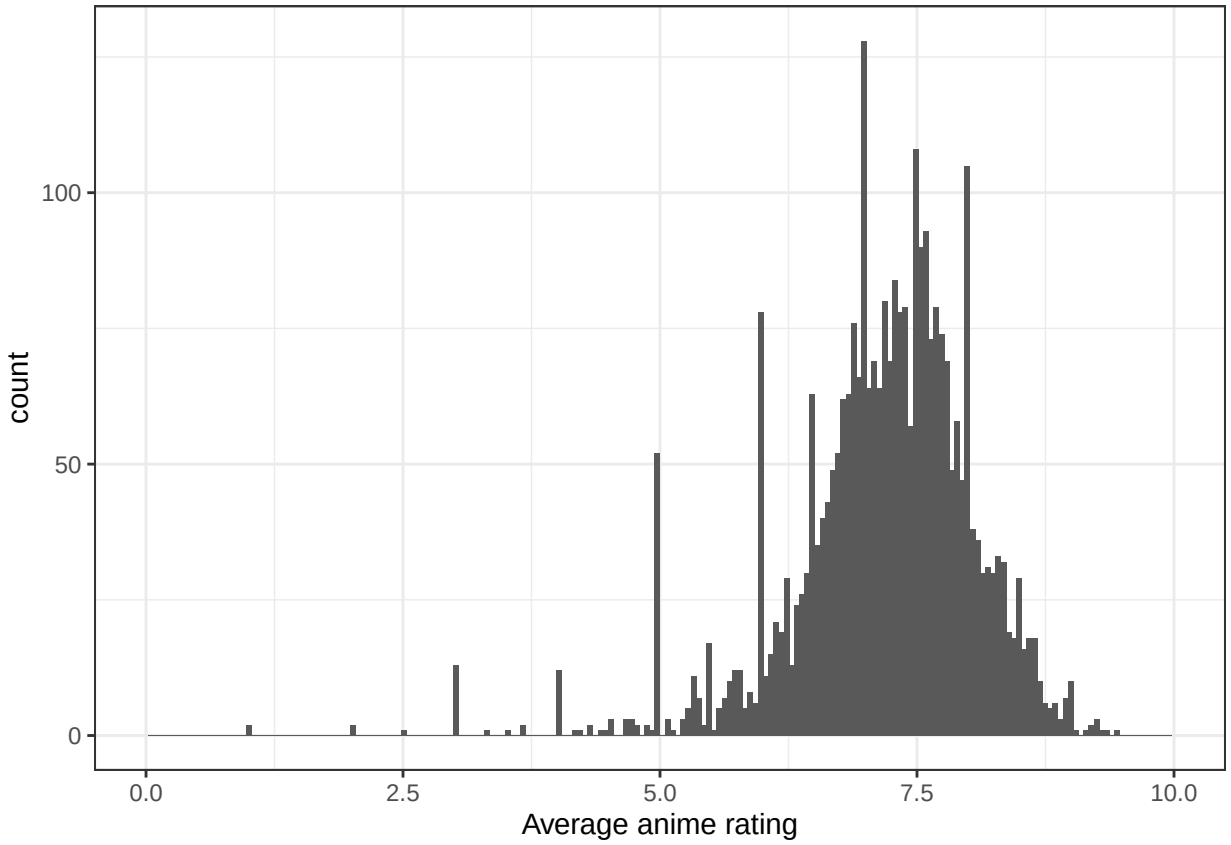
Data exploration, visualisation and relevant insights:

Data exploration and visualisation was completed before and after the data reduction process. For each, a histogram of average anime ratings, and a histogram of average user ratings was plotted.

Histograms - average anime rating: In the average anime rating histogram, it is clear that there are peaks at integer and half ratings (e.g. the highest peak is at 7). This suggests that there are anime series with few ratings which causes the clustering around these integer and half integer values. This justifies the filtering approach outlined above - by filtering by only animes with more than 2000 ratings, this peaking is not shown.

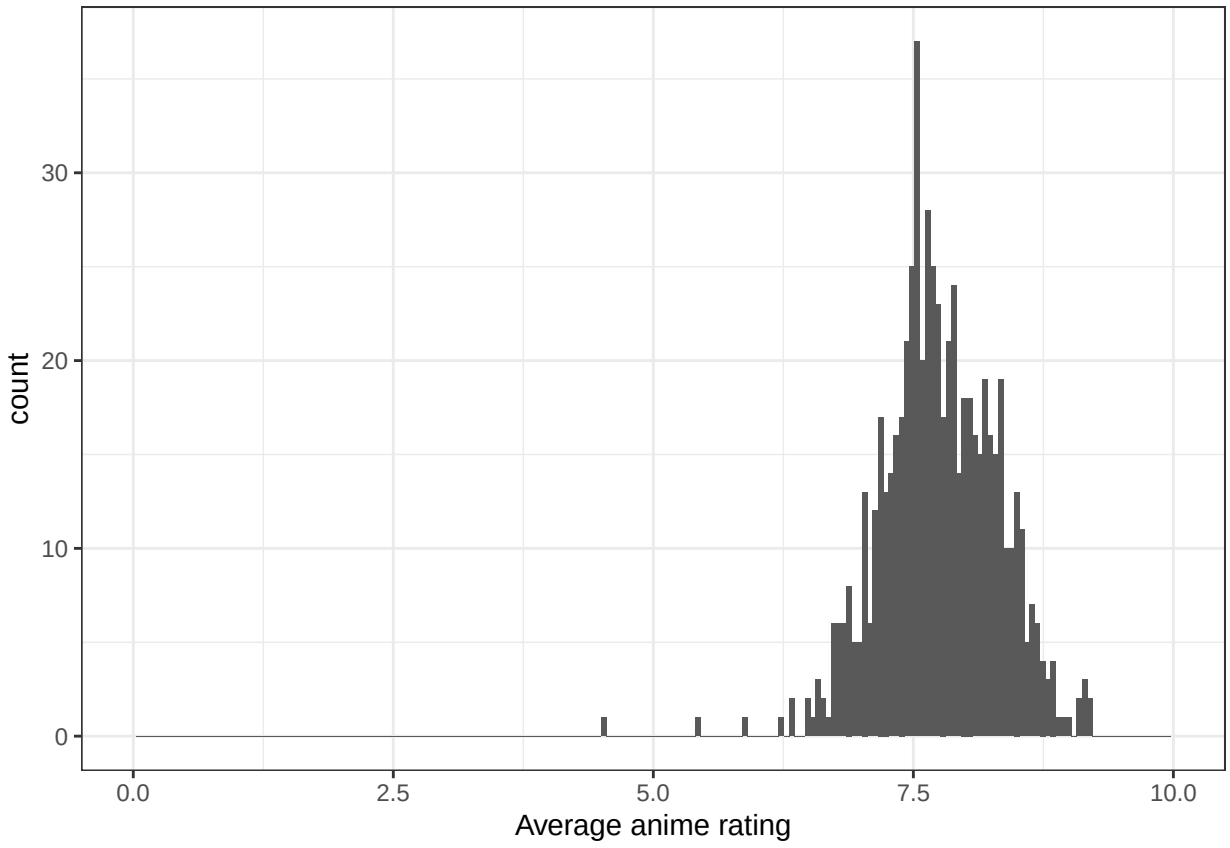
Initial filter only (TV and removal of implict ratings):

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



After data reduction:

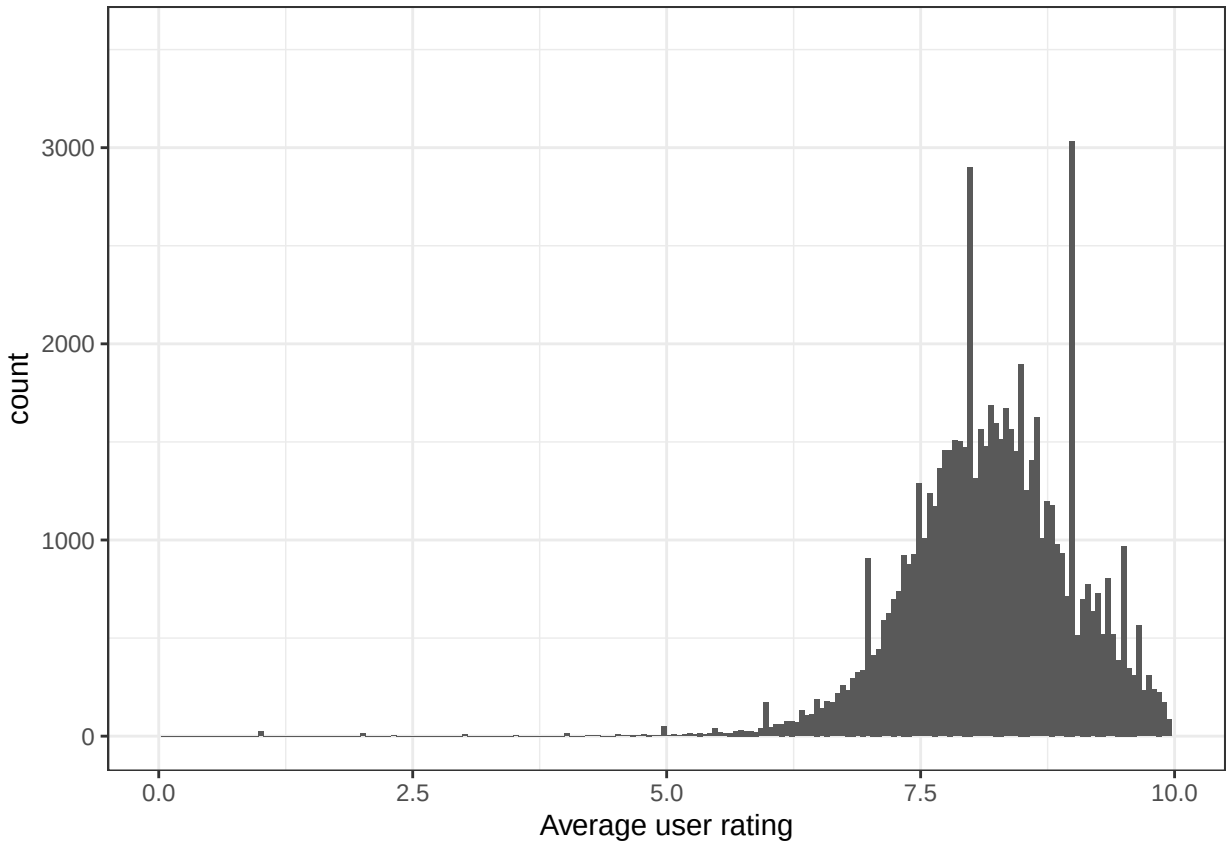
```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



Histograms - average user rating: In the average user rating histogram, I also saw peaks at integer values (e.g. the highest peak is at 9). This is likely to have occurred as a result of there being a variety of users who have made very few ratings - hence the clustering around these integer values. This justifies the filtering approach outlined above - by filtering by only users with more than 200 ratings, this peaking is not shown.

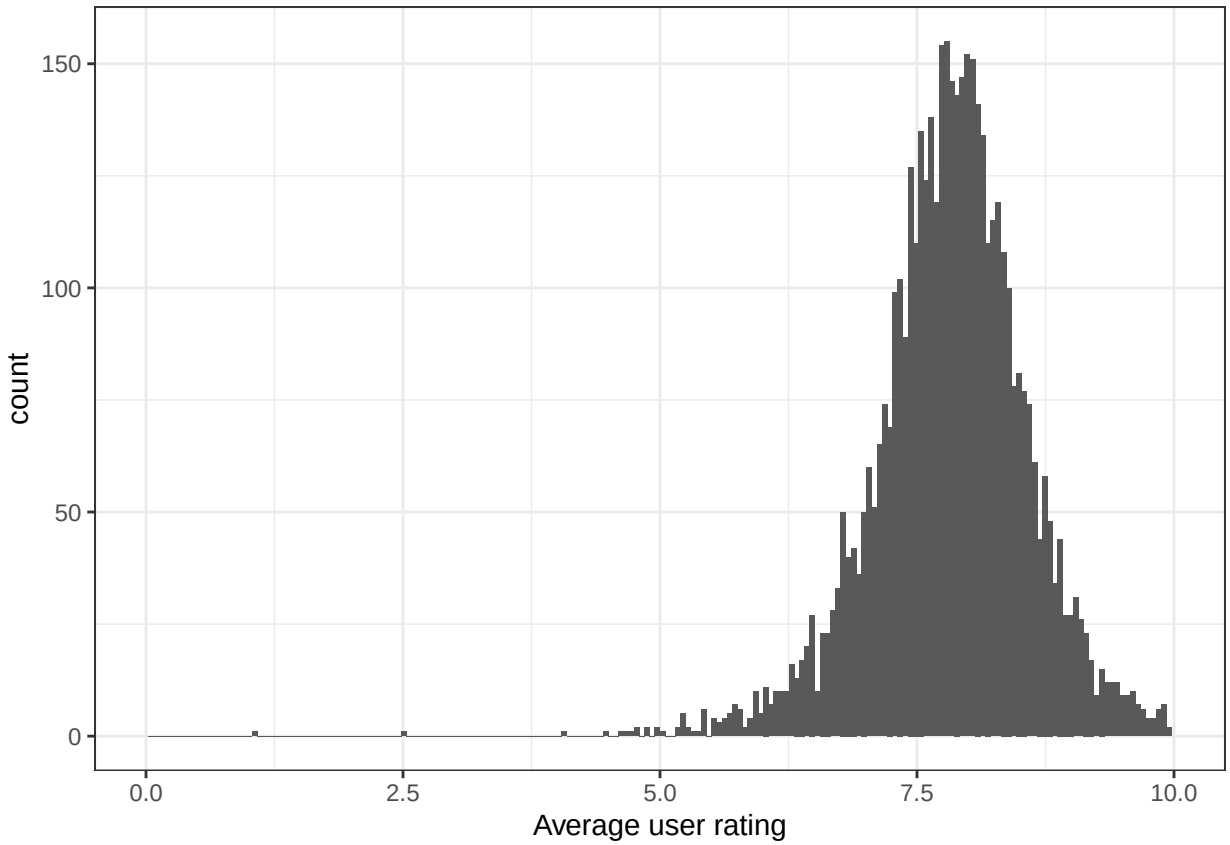
Initial filter only (TV and removal of implict ratings):

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



After data reduction:

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



The histograms demonstrate ratings variation between users (different users give different ratings) and anime series (different anime series receive different ratings). The CF methods utilised later in the report will make use of this variation - for example the item based collaborative filtering (IBCF) approach will consider similarities between anime series and how they can help prediction, and the user based collaborative filtering (UBCF) will consider similarities between users and how they can help prediction.

Modeling approach:

The recommenderlab packages allows one to consider five main approaches: IBCF, UBCF, Popularity, Singular Value Decomposition (SVD) and the Funk Singular Value Decomposition (SVDF). In addition, a "Random" approach is also available which suggests ratings randomly and can therefore be used as a baseline against which to compare other approaches.

The evaluationScheme function performs data splitting - allowing training, test and validation sets to be extracted from a single data set. For this reason, a train and test split was not needed to be performed individually prior to the use of the tools. It was chosen to use 10-fold cross validation as this was likely to lead to less biased estimates than from a simple train-test split. This function also requires one sets a measure of how many ratings are used in the evaluation. When this number is low, evaluations are made with limited information - as is often the case in recommender systems. In this situation I filtered the data so that all users have made more than 200 ratings and all anime series have more than 2000 ratings - since this data is more complete it was chosen to use a value of "-15" which means that all-but-15 ratings were considered.

```
evaluationScheme(data = anime_rrm,
                                 method = "cross-validation",
                                 k = 10,            ## number of cross-validation folds
```

```
                                    given = -15,       ## the number of ratings to withhold when validatin
                                    goodRating = 7.5)  ## what denotes a good rating
```

Predictions were able to be computed using the Recommend function where the method of recommendation was varied so that I could explore the output of different approaches.

Results:

Approach 1: RANDOM

As stated above, this is the baseline algorithm which suggests random recommendations. As a result I could use this as an RMSE floor against which to compare other approaches.

```
##   Algorithms     RMSE      MSE      MAE
## 1     RANDOM 1.589662 2.527025 1.215965
```

Approach 2: Item Based Collaborative Filtering (IBCF)

IBCF suggests anime series (items) based on their similarity to the anime series that the users have seen. They will recommend series that are closest in similarity to those that the user has rated highly. As expected the RMSE shows a large improvement on the random approach.

```
##   Algorithms     RMSE      MSE       MAE
## 1     RANDOM 1.589662 2.527025 1.2159645
## 2       IBCF 1.027738 1.056246 0.7735532
```

Approach 3: User Based Collaborative Filtering (UBCF)

UBCF is similar to IBCF but instead looks at similarities in users rather than in anime series. Anime series will be recommended where users similar to themselves rated them highly. The RMSE shows an improvement on the Random approach, but is higher than that of IBCF.

```
##   Algorithms     RMSE      MSE       MAE
## 1     RANDOM 1.589662 2.527025 1.2159645
## 2       IBCF 1.027738 1.056246 0.7735532
## 3       UBCF 1.140242 1.300152 0.8665191
```

Approach 4: Popularity

The popularity approach is slightly less complex than the previous two approaches, it instead recommends the most popular anime series to users. This makes sense when you consider seminal series where there is widespread agreement on their acclaim. The RMSE shows an improvement on the UBCF approach, but is not as accuracte as IBCF.

```
##   Algorithms     RMSE      MSE       MAE
## 1     RANDOM 1.589662 2.527025 1.2159645
## 2       IBCF 1.027738 1.056246 0.7735532
## 3       UBCF 1.140242 1.300152 0.8665191
## 4    POPULAR 1.118648 1.251373 0.8559868
```

Approach 5: Singular Value Decomposition (SVD)

SVD looks at both items and users within the data and attempts to define latent factors that drive the recommendations made - for example these latent factors could be genre, or anime studio. The RMSE shows an improvement on the UBCF and Popular approaches but IBCF still seems to be the most effective.

```
##   Algorithms     RMSE      MSE       MAE
## 1     RANDOM 1.589662 2.527025 1.2159645
## 2       IBCF 1.027738 1.056246 0.7735532
## 3       UBCF 1.140242 1.300152 0.8665191
## 4    POPULAR 1.118648 1.251373 0.8559868
## 5        SVD 1.062058 1.127967 0.8089538
```

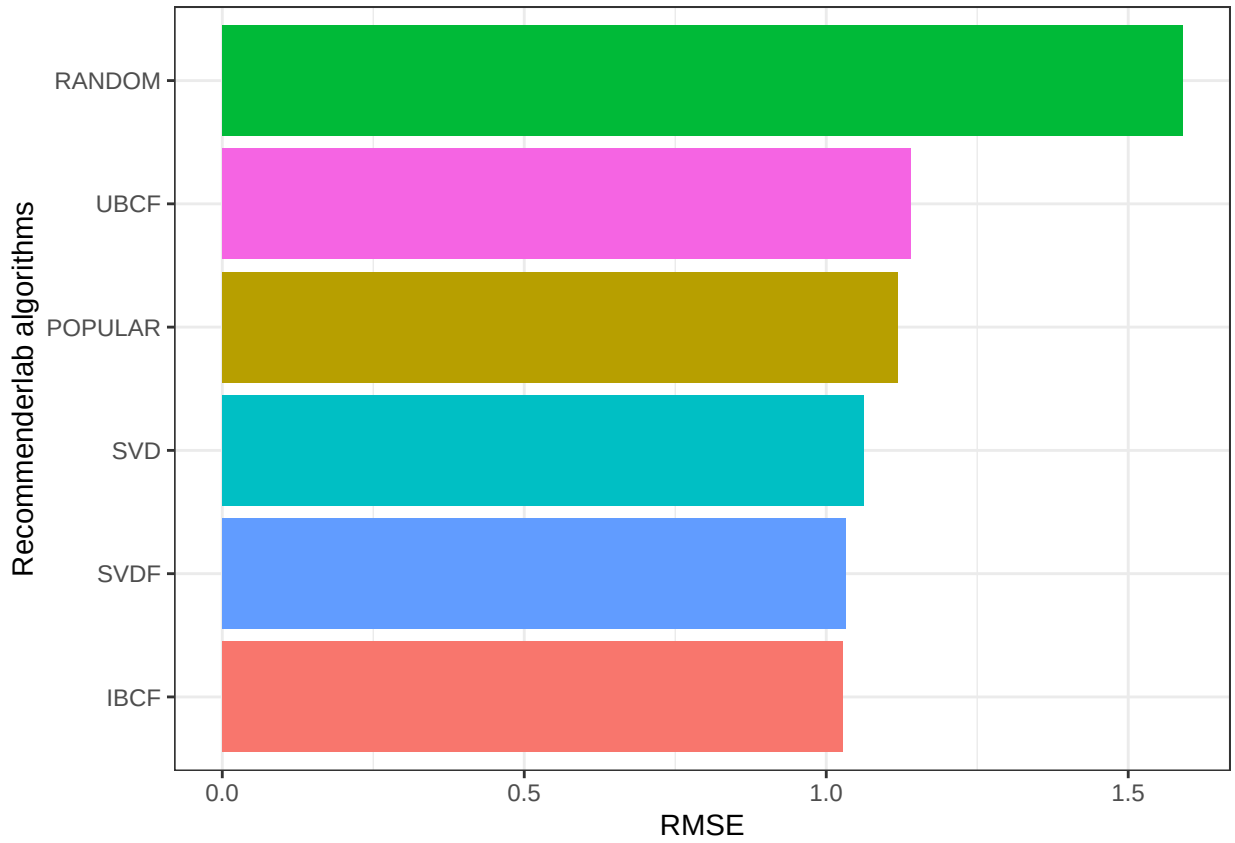
Approach 6: Funk Singular Value Decomposition (SVD)

SVD is best optimised for ratings matrices where there are relatively few missing values. In much recommender data there are large numbers of NAs - i.e. instances where a user has not rated an item. As this is the case within this data the Funk SVD approach has a marginal improvement on SVD, and performs similarly (though slightly worse) than IBCF.

```
##   Algorithms     RMSE      MSE       MAE
## 1     RANDOM 1.589662 2.527025 1.2159645
## 2       IBCF 1.027738 1.056246 0.7735532
## 3       UBCF 1.140242 1.300152 0.8665191
## 4    POPULAR 1.118648 1.251373 0.8559868
## 5        SVD 1.062058 1.127967 0.8089538
## 6       SVDF 1.031875 1.064766 0.7883938
```

The RMSE minimising approach:

Plotting all of the RMSE results against each other allows us to identify which of these was the RMSE minimising approach. The lowest RMSE was for IBCF, with the Funk SVD approach coming a close second.



Conclusion:

Recommendation systems are an intuitive demonstration of the power of machine learning. In this report I built an anime series recommender and evaluated this based on its ability to predict ratings (evaluated via RMSE). Rather than consider content based approaches (as were considered in the movielens project), I looked at Collaborative Filtering (CF) approaches. These use similarity and latent factors to make ratings predictions. I found that the IBCF approach for my dataset was the most effective - though there is scope for this to be different if the approach was instead applied on a larger dataset (I reduced the size of my dataset so that I was able to use the recommenderlab tools on my personal laptop). In addition, I chose an expansive value for "given", it is likely if this number was reduced the IBCF approach would perform less well relative to SVD (and SVDF) - I would expect SVDF to be more robust to small ratings samples.

This report was limited in part by the memory of my hardware that required me to reduce the size of my dataset prior to starting evaluation. This subset was however limited to users who gave many reviews and hence might actually be considered a dataset of anime "experts".

The report considered the use of CF approaches in isolation - further work would be useful to also consider hybrid approaches. Either to combine multiple CF approaches, or more interestingly, to combine CF approaches with content based approaches. Averaging is a powerful tool and different weighting schemes are likely to improve the RMSE relative to non-hybrid approaches.

References:

Michael Hahsler (2016). recommenderlab: A Framework for Developing and Testing Recommendation Algorithms, R package. https://CRAN.R-project.org/package=recommenderlab