edx HarvardX Data Science Project Report - Goodbooks 10k

Li Chen

6/11/2019

1. Introduction

There have been recommender systems with good datasets for movies (Netflix, Movielens) and music (Million Songs) recommendation, and now we will study the books recommendation using the "goodbooks-10k" data from Kaggle (https://www.kaggle.com/zygmunt/goodbooks-10k). In this capstone project we will study the collaborative filtering algorithms and matrix factorization using recommender engines in R: the "recommenderlab" package and "recosystem" package. We are going to train our algorithms using the goodbooks-10k dataset to predict book ratings. RMSE will be used to evaluate the performance of the algorithms.

2. Dataset and Exploratory Analysis

2.1 Kaggle goodbooks 10k data

The Kaggle goodbooks-10k dataset contains ratings for ten thousand popular books whose ratings were found on the internet. Ratings go from one to five. Both book IDs and user IDs are continuous: For books, they are 1-10000, for users, 1-53424. The upzip data include following files:

- ratings.csv: contains ratings (user_id, book_id, rating).
- to_read.csv: provides IDs of the books marked "to read" by each user, as user_id,book_id pairs.
- books.csv: has metadata for each book (goodreads IDs, authors, title, average rating, etc.).
- book_tags.csv: contains tags/shelves/genres assigned by users to books. Tags in this file are represented by their IDs.
- tags.csv: translates tag IDs to names.

First loading library and data:

```
library(tidyverse)
library(caret)
library(readr)
library(knitr)
library(Matrix)
library(data.table)
library(dplyr)
library(tidyr)
library(ggplot2)
```

Install the two recommender engines, recommenderlab and recosystem R packages and load them:

```
#install.packages("recommenderlab")
#install.packages("recosystem")
library(recommenderlab)
library(recosystem)
```

Download and read the goodbooks-10k data from the author's github repository:

```
# The separate data files can be downloaded from my github repository
datafile_books <- "https://github.com/orientalpearls/edx-ds-cyo-goodbooks-10k/raw/master/books.csv"
datafile_book_tags <- "https://github.com/orientalpearls/edx-ds-cyo-goodbooks-10k/raw/master/book_tags.
```

```
datafile_tags <- "https://github.com/orientalpearls/edx-ds-cyo-goodbooks-10k/raw/master/tags.csv"
datafile_ratings <- "https://github.com/orientalpearls/edx-ds-cyo-goodbooks-10k/raw/master/ratings.csv"

# Read the .csv data into R
books_all <- fread(datafile_books)
book_tags_all <- fread(datafile_book_tags)
tags_all <- fread(datafile_tags)
ratings_all <- fread(datafile_ratings)
```

2.2 Data Cleaning

Following shows the number of unique users in the goodbooks-10k dataset that provided ratings and the number of unique books that were rated:

```
ratings_all %>%
  summarize(num_books = n_distinct(book_id),
            num_users = n_distinct(user_id))
```

```
##   num_books num_users
## 1     10000     53424
```

When exploring the data we found that there are multiple ratings for some combinations of user and book, and for the collaborative filtering it is better to have more ratings per user. So we are going to remove the duplicate ratings and to remove users who have rated fewer than 5 books.

First we remove the duplicate ratings:

```
# Remove the duplicate ratings from the same user_id and book_id
ratings_n <- ratings_all %>%
  group_by(user_id, book_id) %>%
  mutate(N=n())
cat('Number of duplicate ratings: ',
    nrow(filter(ratings_n, N > 1)))
```

```
## Number of duplicate ratings:  4487
```

```
ratings <- ratings_n %>%
  filter(N == 1) %>%
  subset(select = -N)
```

Then we remove the users who have rated fewer than 5 books:

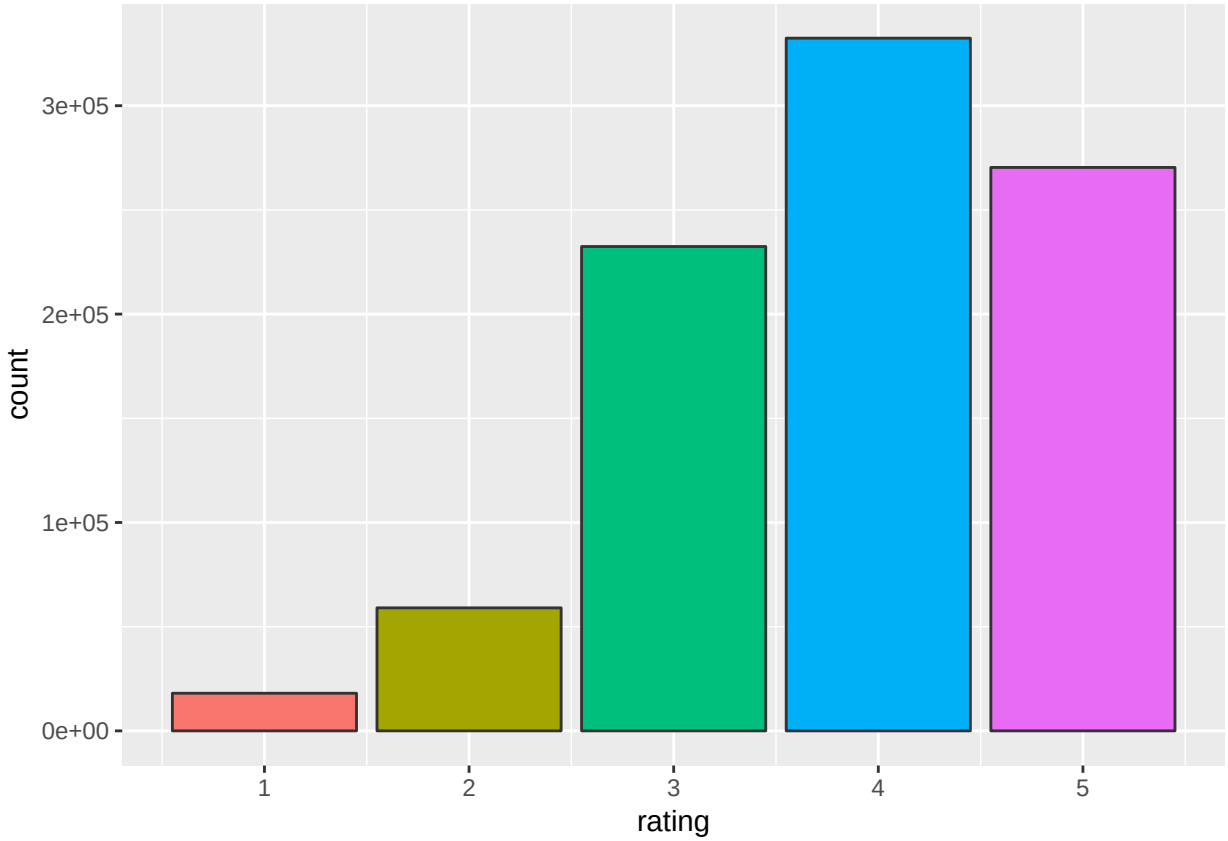
```
# Remove users who rated fewer than 5 books
ratings_users_n <- ratings %>%
  group_by(user_id) %>%
  mutate(N_u = n())
cat('Number of users who rated fewer than 5 books: ',
    uniqueN(filter(ratings_users_n, N_u <= 4)$user_id))
```

```
## Number of users who rated fewer than 5 books:  17771
```

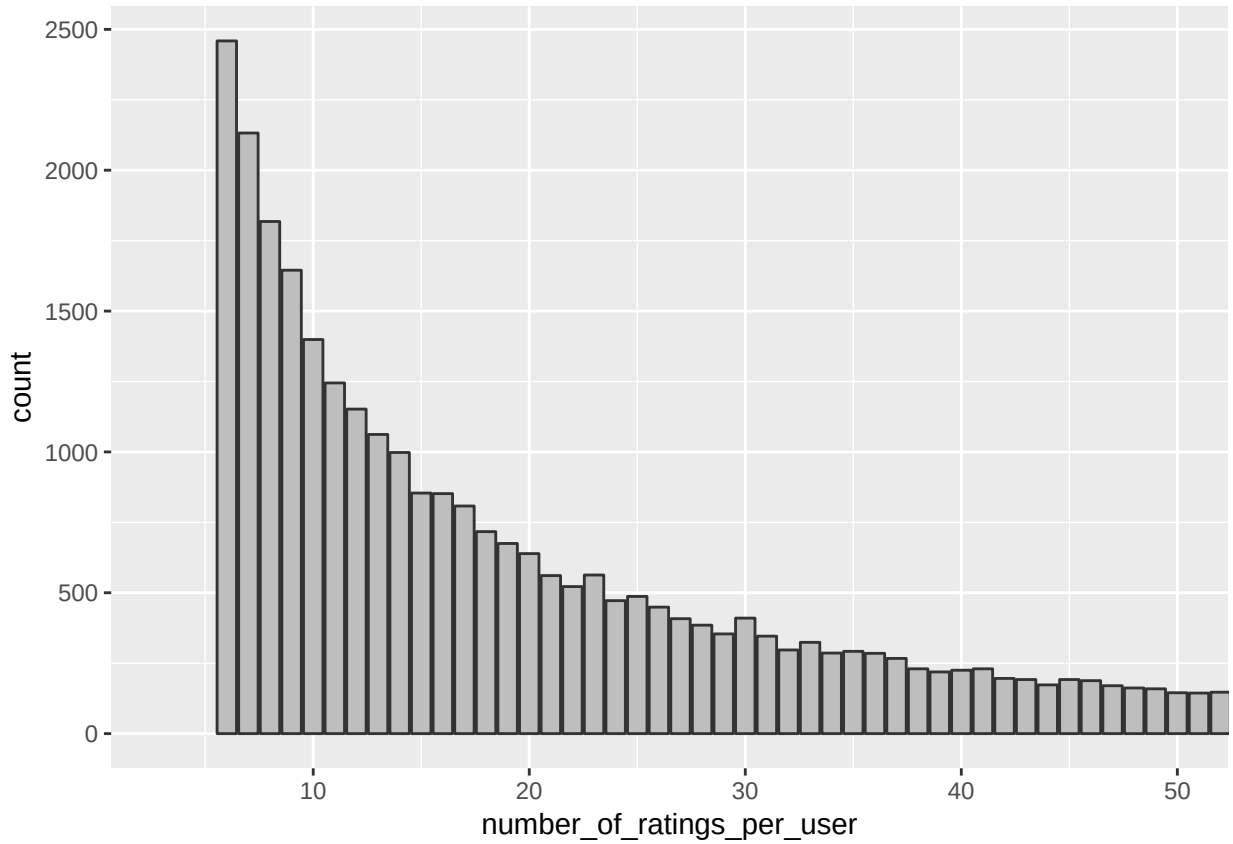
```
ratings <- ratings_users_n %>%
  filter(N_u > 5) %>%
  subset(select = -N_u)
```

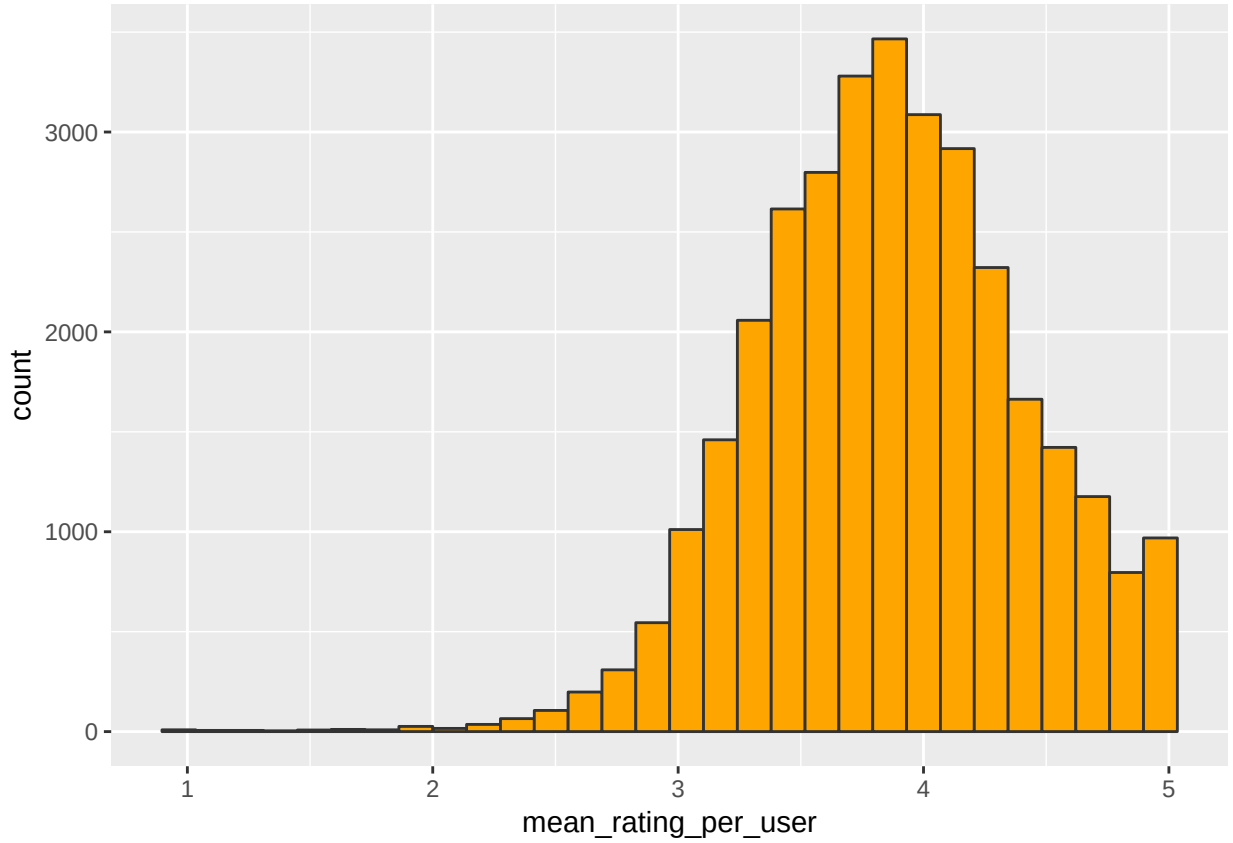
2.3 Data Exploration

Distribution of ratings shows that people tend to give positive ratings to books as most of the ratings are in the 3-5 range, while very few ratings are in the 1-2 range.

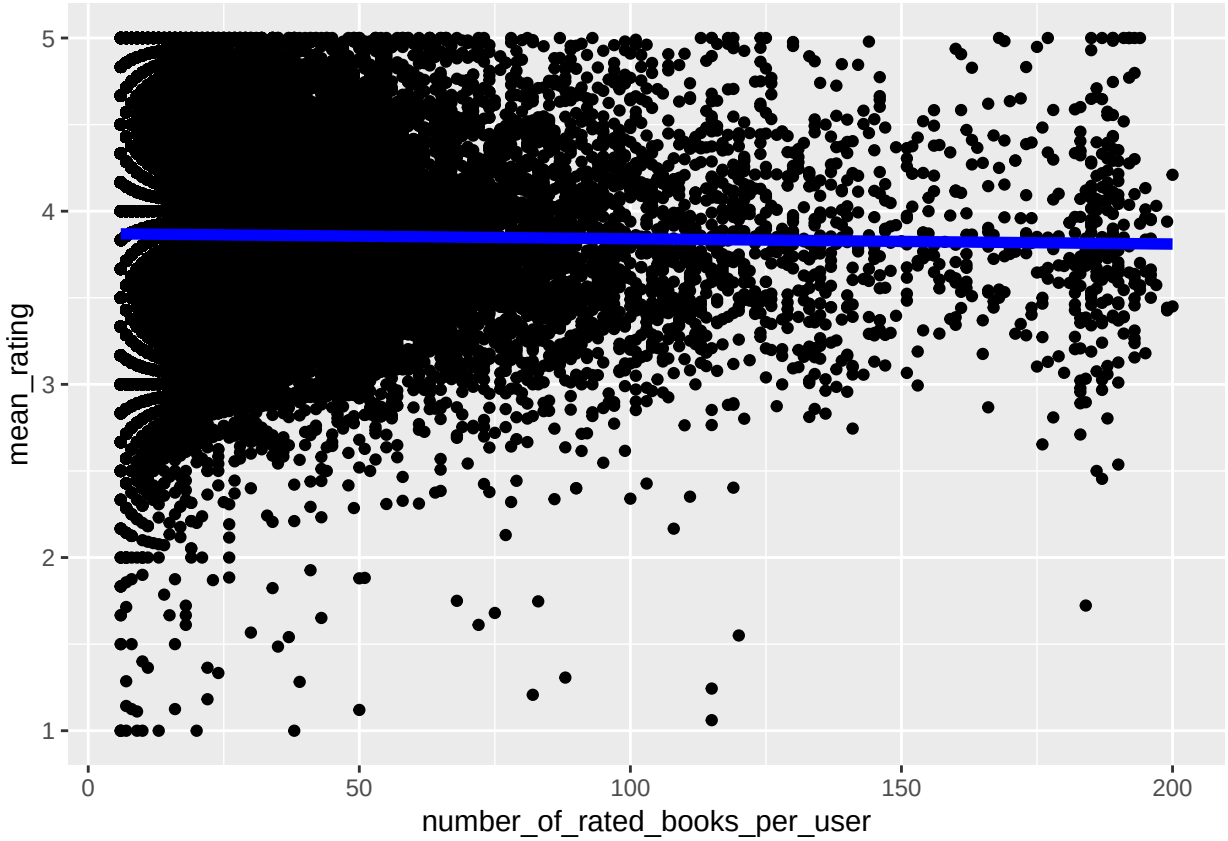


Following figures show the distribution of number of ratings per user and distribution of mean user ratings.

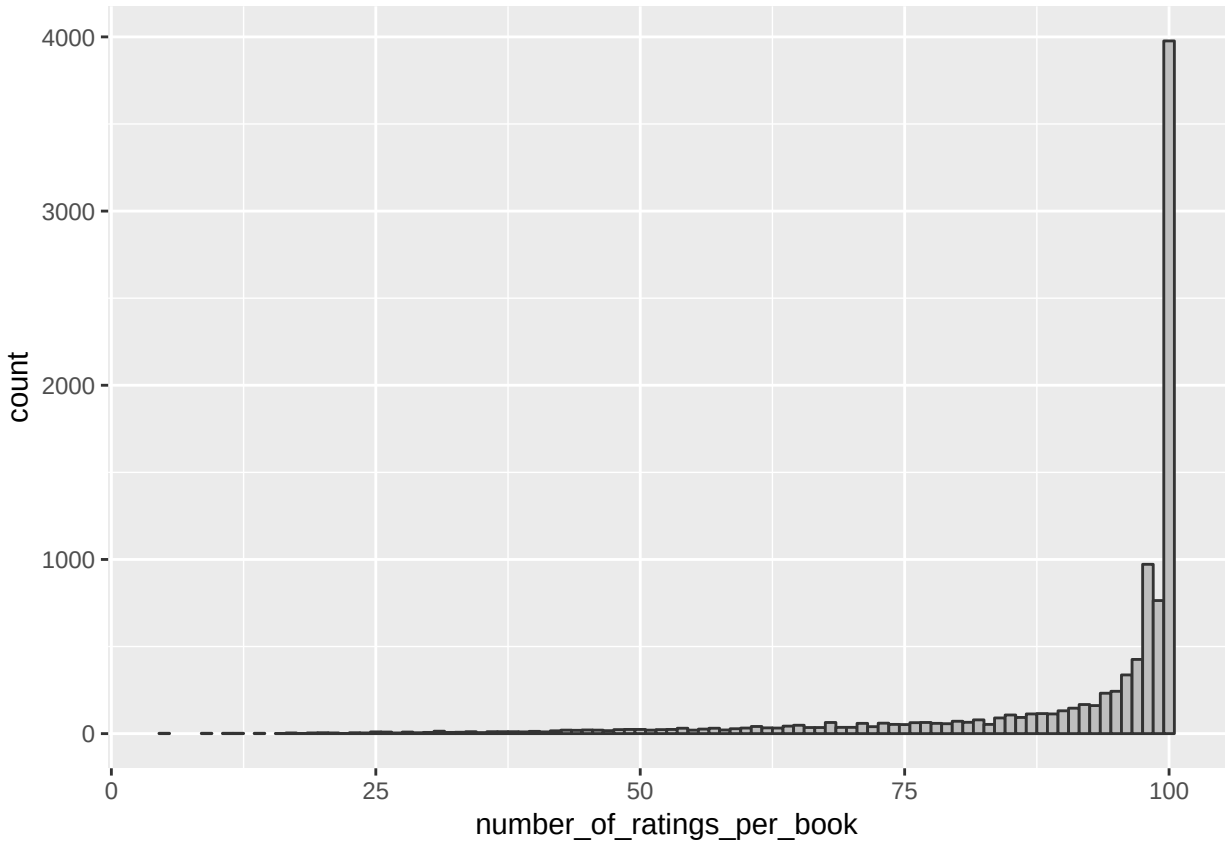

2500
2000
1500
1000
500
0
count
10
20
30
40
50
number_of_ratings_per_user


3000
2000
1000
0
count
1
2
3
4
5
mean_rating_per_user

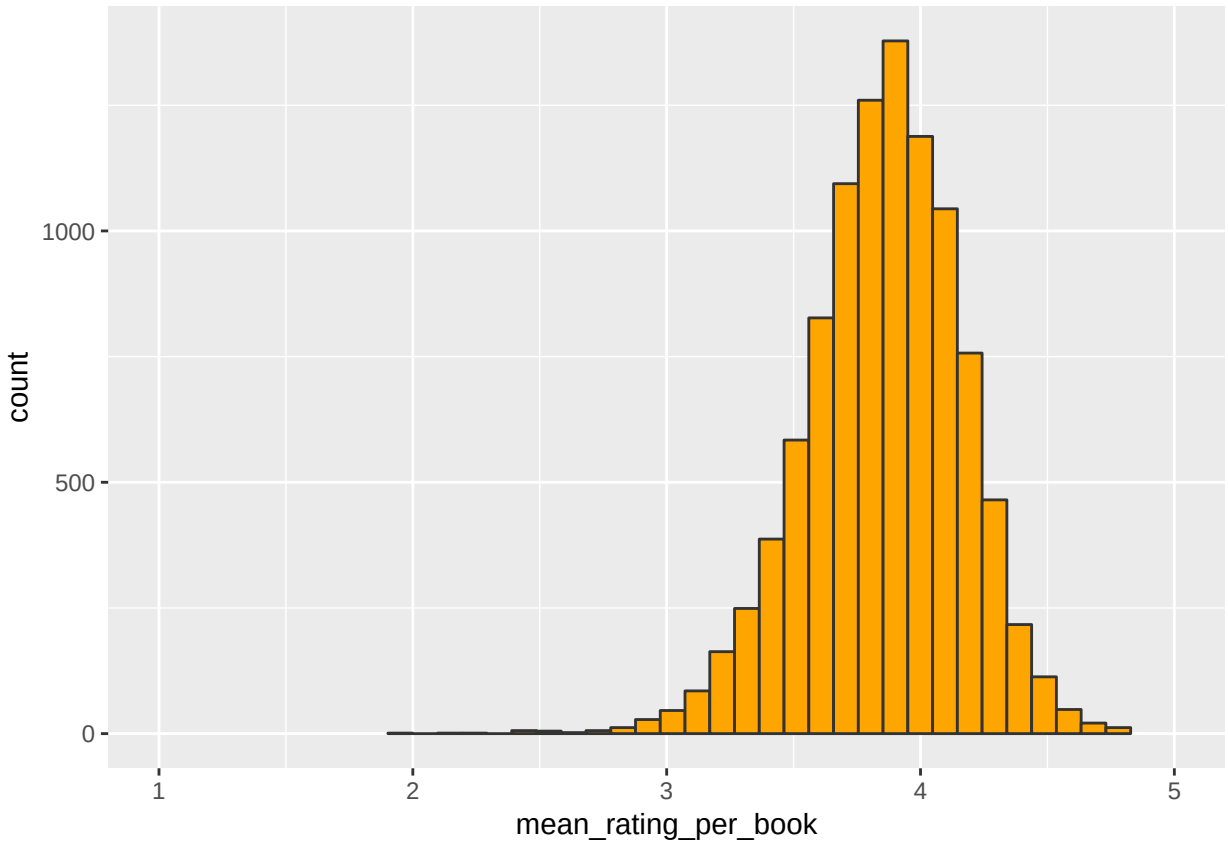
It is possible that users who rate more books tend to rate books differently from less frequent raters hey are/become more critical the more they read and rate. The figure below shows that frequent raters is more likely to give lower ratings to books, but just slightly.



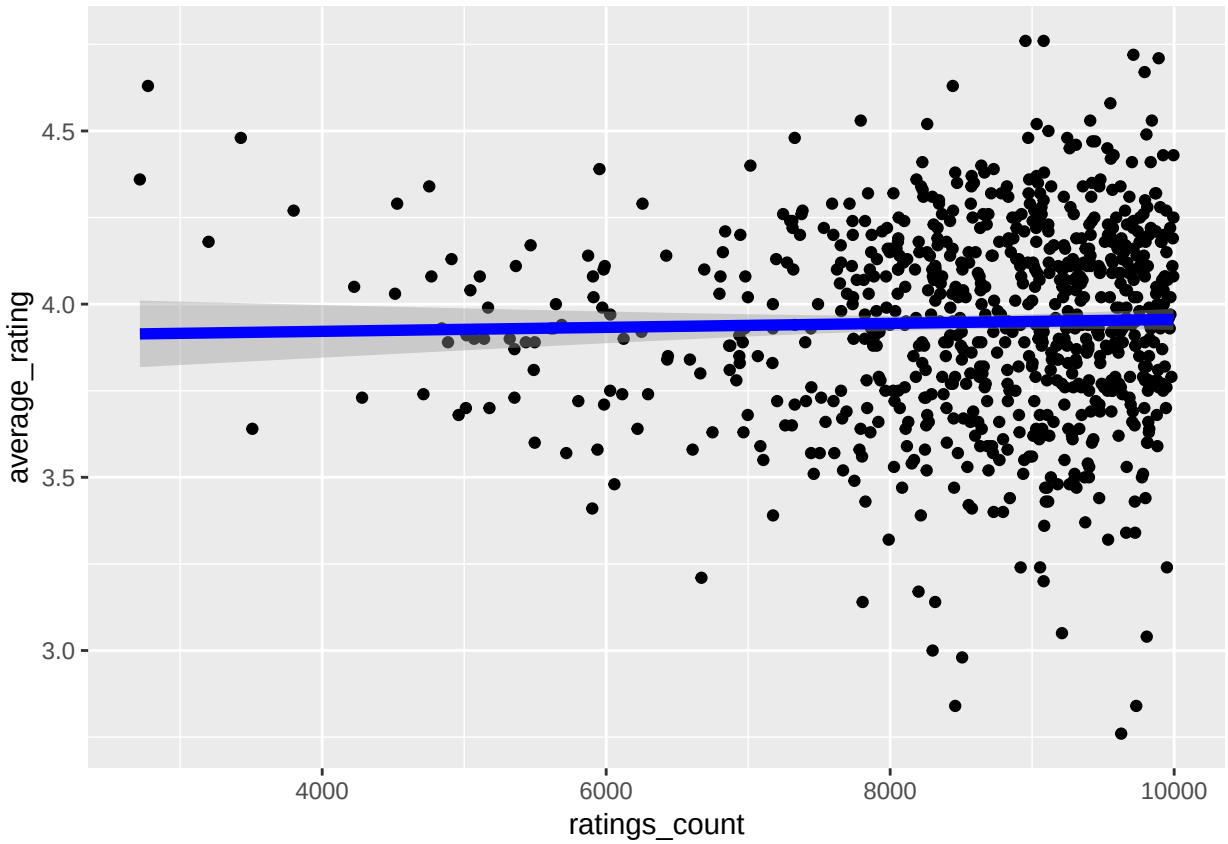
The distribution of the number of ratings per book shows that most of the books in the dataset gets 100 ratings, which is consistent with those explained in the Kaggle dataset website (https://www.kaggle.com/zygmunt/goodbooks-10k) "generally, there are 100 reviews for each book, although some have less - fewer - ratings".



Distribution of mean book ratings doesn't show any peculiarities.



Do popular books (books with large number of ratings) get higher average ratings? It might be that the popularity of a book (in terms of the number of ratings it receives) will be associated with the average rating it receives, such that once a book is becoming popular it gets better ratings. However, our data shows that this is true only to a very small extent.



3. Methods and Analysis

Our project will try the collaborative filtering and matrix factorization methods for recommendation system. In this section we will explain the different machine learning algorithms, data preparation, and parameter selection in the two R packages, recommenderlab and recosystem, for building recommender systems. We will use the RMSE to measure the performance.

3.1 Performance Evaluation Metric - RMSE

One of the typical metrics to measure the effectiveness of the recommendation system is root mean squared error (RMSE). RMSE is defined as the square root of the average square error between the true rating and predicted rating.

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

It is used to evaluate how close the predictions are to the true values in the validation set: the smaller the error, the better the recommendation system is; if the error is larger than 1 star, then the recommendation system is not good. The R code for RMSE function is written as below:

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

3.2 Collaborative Filtering - recommenderlab package

Collaborative filtering is a standard method for recommendation system. The general idea is that if two users share the same interests in the past, e.g. they liked the same book, they will also have similar tastes in the future. Or given a new user, the algorithm considers the user's purchases and recommends similar items.

The most popular two methods are user-based and item-based collaborative filtering. User-based collaborative filtering (UBCF) method is built based on assumption that if two users have similar ratings on some items, they will have similar ratings on the remaining items. The same for item-based collaborative filtering (IBCF) with item perspective. The collaborative filtering approach considers only user preferences and does not take into account the features or contents of the items (books) being recommended.

Recommenderlab is a R-package that provides the infrastructure to evaluate and compare several collaborative-filtering algortihms. Many algorithms are already implemented in the package, and we can use the available ones to save some coding effort, or add custom algorithms and use the infrastructure (e.g. cross validation).

3.2.1 Data processing

The recommenderlab package allows us to use and evaluate recommender systems. But the data must be first converted into a matrix format. Below, the ratings is converted into a recommenderlab format called a realRatingMatrix. Most of the values in the rating matrix are missing, because every user just rated a few of the books. This allows us to represent this matrix in sparse format in order to save memory.

```
#Ratings data first converted to matrix format
dimension_names <- list(user_id = sort(unique(ratings$user_id)),
                        book_id = sort(unique(ratings$book_id)))
ratingmat <- spread(select(ratings, book_id, user_id, rating), book_id, rating) %>%
  subset(select=-user_id)
ratingmat <- as.matrix(ratingmat)
dimnames(ratingmat) <- dimension_names

#Then converted to sparseMatrix format
tmpmat <- ratingmat
tmpmat[is.na(tmpmat)] <- 0
sparse_ratings <- as(tmpmat, "sparseMatrix")
rm(tmpmat)

#Finally converted into a recommenderlab realRatingMatrix format
real_ratings <- new("realRatingMatrix", data = sparse_ratings)
dim(real_ratings)
```

```
## [1] 32394 10000
```

3.2.2 Exploring Parameters of Recommendation Models

Running an algorithm in Recommenderlab is pretty easy. All one has to do is call *Recommender()* and pass the data, select a method (e.g., "UBCF" - user-based collaborative filtering) and pass some params (e.g., the method for calculating similarity, e.g. pearson or cosine similarity, and the number of most similar users used for the predictions, nn, e.g. 5).

The recommenderlab package contains some options for the recommendation algorithm:

```
recommenderRegistry$get_entry_names()
```

```
##  [1] "ALS_realRatingMatrix"            "ALS_implicit_realRatingMatrix"
##  [3] "ALS_implicit_binaryRatingMatrix" "AR_binaryRatingMatrix"
##  [5] "IBCF_binaryRatingMatrix"         "IBCF_realRatingMatrix"
##  [7] "POPULAR_binaryRatingMatrix"      "POPULAR_realRatingMatrix"
```

```
##  [9] "RANDOM_realRatingMatrix"          "RANDOM_binaryRatingMatrix"
## [11] "RERECOMMEND_realRatingMatrix"     "SVD_realRatingMatrix"
## [13] "SVDF_realRatingMatrix"            "UBCF_binaryRatingMatrix"
## [15] "UBCF_realRatingMatrix"
```

We can see that there are many different recommender systems with different algorithms that we can use. In this project we will illustrate the random, popular, user-based collaborative algorithms of the recommenderlab package.

RANDOM

The algorithm ("RANDOM"): randomly predicts a rating for each user. The "POPULAR" method is a method based on item popularity.

POPULAR

It is also called item average approach, and computes average rating for each item based on available ratings and predicts each unknown rating as average for item. Thus missed ratings for each item will be the same for each user. In this lgorithm, first the average rating for each item is calculated, then the missed ratings is predicted in R (rating matrix) as average for item.

UBCF

User-based collaborative filtering works under assumption that users with similar ratings will rate items similarly. It makes predictions based on aggregated ratings from the closest users (nearest neighbors). Nearest neighbors are defined based on similarity between users, which is calculated using available ratings. There are many different similarity measures, which are used for training such recommender. The most popular for collaborative filtering are Pearson correlation and Cosine similarity. For each user, these steps are taken: 1. Calculate similarity between user u and all other users. For this could be used any preferred similarity measure; 2. Select top n users with the highest similarity to users u; 3. Calculate predictions for unknown ratings for user u as average of available ratings from n closest users or as weighed (on similarity distance) ratings of n closest users; 4. Pick the top-rated items.

Following shows some default parameters of these models.

```
recommenderRegistry$get_entries("RANDOM", dataType = "realRatingMatrix")
```

```
## $RANDOM_realRatingMatrix
## Recommender method: RANDOM for realRatingMatrix
## Description: Produce random recommendations (real ratings).
## Reference: NA
## Parameters: None
```

```
recommenderRegistry$get_entries("POPULAR", dataType = "realRatingMatrix")
```

```
## $POPULAR_realRatingMatrix
## Recommender method: POPULAR for realRatingMatrix
## Description: Recommender based on item popularity.
## Reference: NA
## Parameters:
##   normalize
## 1  "center"
##                                                  aggregationRatings
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
##                                                aggregationPopularity
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
```

```
recommenderRegistry$get_entries("UBCF", dataType = "realRatingMatrix")
```

```
## $UBCF_realRatingMatrix
```

```
## Recommender method: UBCF for realRatingMatrix
## Description: Recommender based on user-based collaborative filtering.
## Reference: NA
## Parameters:
##     method nn sample normalize
## 1 "cosine" 25  FALSE  "center"
```

The good thing about recommenderlab is that it offers the possibility to easily evaluate and compare algorithms. In order to do so, one first has to create an evaluation scheme using *evaluationScheme()*. For example, we can chose to do 10-fold cross validation. The given parameter determines how many ratings are given to create the predictions and in turn on how many predictions per user remain for the evaluation of the prediction. In this case -1 means that the predictions are calculated from all but 1 ratings, and performance is evaluated for 1 for each user.

```
# 10-fold cross validation
scheme <- evaluationScheme(real_ratings[1:500,],
                           method = "cross-validation",
                           k = 10, given = -1, goodRating = 5)
```

For algorithm UBCF, we have a tuneable parameter nn, which is the number of most similar users which are used to calculate the predictions. To find the best value of nn, separate validation set or cross-validation could be used. We can vary this parameter from 5 to 50 and plot the RMSE.

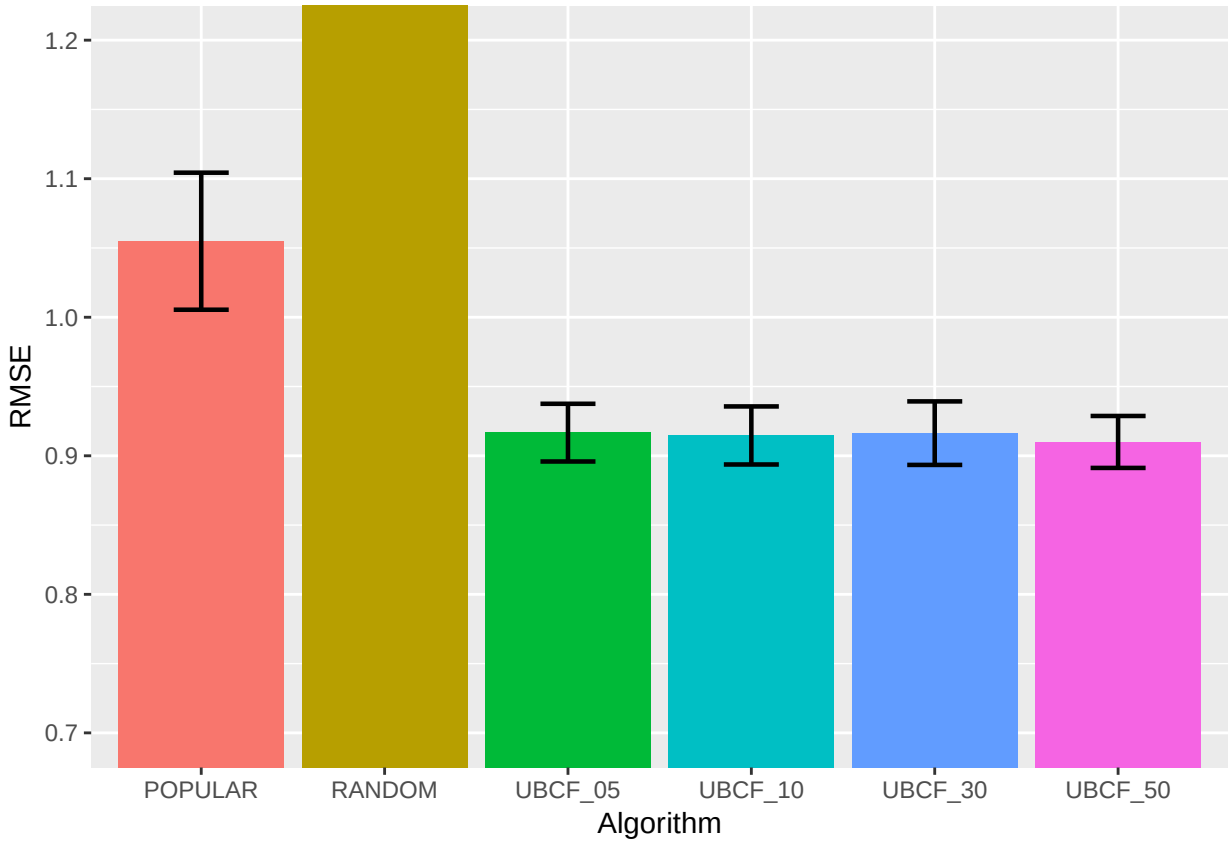
```
# Compare the performance for different algorithms and parameters
algorithms <- list("RANDOM" = list(name = "RANDOM", param = NULL),
                   "POPULAR" = list(name = "POPULAR", param = NULL),
                   "UBCF_05" = list(name = "UBCF", param = list(nn = 05)),
                   "UBCF_10" = list(name = "UBCF", param = list(nn = 10)),
                   "UBCF_30" = list(name = "UBCF", param = list(nn = 30)),
                   "UBCF_50" = list(name = "UBCF", param = list(nn = 50)))

# evaluate the alogrithms with the given scheme
results <- evaluate(scheme, algorithms, type = "ratings")
```

```
## RANDOM run fold/sample [model time/prediction time]
##   1  [0.002sec/0.257sec]
##   2  [0.003sec/0.362sec]
##   3  [0.002sec/0.331sec]
##   4  [0.001sec/0.324sec]
##   5  [0.003sec/0.345sec]
##   6  [0.003sec/0.298sec]
##   7  [0.001sec/1.335sec]
##   8  [0.001sec/0.247sec]
##   9  [0.002sec/0.231sec]
##   10  [0.001sec/0.229sec]
## POPULAR run fold/sample [model time/prediction time]
##   1  [0.029sec/0.083sec]
##   2  [0.011sec/0.081sec]
##   3  [0.012sec/0.08sec]
##   4  [0.013sec/0.078sec]
##   5  [0.01sec/0.106sec]
##   6  [0.015sec/0.087sec]
##   7  [0.01sec/0.087sec]
##   8  [0.013sec/0.093sec]
##   9  [0.012sec/0.114sec]
##   10  [0.017sec/0.124sec]
```

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0.006sec/1.281sec]
##   2  [0.005sec/1.272sec]
##   3  [0.006sec/1.26sec]
##   4  [0.006sec/1.221sec]
##   5  [0.006sec/1.215sec]
##   6  [0.006sec/1.313sec]
##   7  [0.006sec/1.523sec]
##   8  [0.006sec/1.219sec]
##   9  [0.005sec/1.512sec]
##   10  [0.007sec/1.233sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.008sec/1.35sec]
##   2  [0.007sec/1.395sec]
##   3  [0.006sec/2.405sec]
##   4  [0.004sec/1.163sec]
##   5  [0.004sec/1.126sec]
##   6  [0.003sec/1.226sec]
##   7  [0.004sec/1.171sec]
##   8  [0.004sec/1.08sec]
##   9  [0.005sec/1.101sec]
##   10  [0.004sec/1.176sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.004sec/1.1sec]
##   2  [0.007sec/1.158sec]
##   3  [0.005sec/1.162sec]
##   4  [0.005sec/1.19sec]
##   5  [0.005sec/1.133sec]
##   6  [0.007sec/1.264sec]
##   7  [0.006sec/1.166sec]
##   8  [0.005sec/1.262sec]
##   9  [0.004sec/1.229sec]
##   10  [0.008sec/2.617sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.005sec/1.075sec]
##   2  [0.004sec/1.099sec]
##   3  [0.004sec/1.119sec]
##   4  [0.005sec/1.108sec]
##   5  [0.005sec/1.085sec]
##   6  [0.005sec/1.084sec]
##   7  [0.004sec/1.096sec]
##   8  [0.004sec/1.111sec]
##   9  [0.004sec/1.078sec]
##   10  [0.003sec/1.425sec]
```

```r
tmp <- lapply(results, function(x) slot(x, "results"))
res <- tmp %>%
  lapply(function(x) unlist(lapply(x, function(x) unlist(x@cm[ ,"RMSE"])))) %>%
  as.data.frame() %>%
  gather(key = "Algorithm", value = "RMSE")
res %>%
  ggplot(aes(Algorithm, RMSE, fill = Algorithm)) +
  geom_bar(stat = "summary") +
  geom_errorbar(stat = "summary", width = 0.3, size = 0.8) +
```

```
  coord_cartesian(ylim = c(0.7, 1.2)) +
  guides(fill = FALSE)
```



3.3 Matrix Factorization - recosystem package

Matrix Factorization is a popular technique to solve recommender system problem. The main idea is to approximate the matrix $R_{m\times n}$ by the product of two matrixes of lower dimension: $P_{k\times m}$ and $Q_{k\times n}$.

Matrix P represents latent factors of users. So, each k-elements column of matrix P represents each user. Each k-elements column of matrix Q represents each item. So, the rating given by user u on item v would be predicted as $P[,u]' \times Q[,i]$. Short and full description of package is available at https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html.

In recosystem package, we create a model object by calling *Reco()* , call the *tune()* method to select best tuning parameters along a set of candidate values, train the model by calling the *train()* method and use the *predict()* method to compute predicted values.

3.3.1 Data processing

Recosystem can significantly reduce memory use, as the constructed model that contains information for prediction can be stored in the hard disk, and output result can also be directly written into a file rather than be kept in memory. This package works with data saved on disk in 3 columns with no headers. We can create train set (80%) and test set (20%) as follows:

```
#Creating traing and testing set (20% of the ratings dataset)
set.seed(1)
smp_size <- floor(0.20 * nrow(ratings))
in_train <- rep(TRUE, nrow(ratings))
```

```
in_train[sample(1:nrow(ratings), size = smp_size)] <- FALSE

train_set <- ratings[(in_train),]
test_set <- ratings[(!in_train),]

#Setup a test and train set for matrix factorization method for recosystem
train_set_mf <- data_memory(train_set$user_id, train_set$book_id, rating = train_set$rating)
test_set_mf <- data_memory(test_set$user_id, test_set$book_id, rating = NULL)
```

3.3.2 Tuning Parameters

Now we could tune parameters of Recommender: number of latent factors (dim), gradient descend step rate (lrate), and penalty parameter to avoid overfitting (cost). To make this easier and faster to run, cost could be set to some small value:

```
# start a recosystem session
r = Reco()

# tune the hyper parameters
opts = r$tune(train_set_mf, opts = list(dim = c(1:30), lrate = c(0.1, 0.2),
                                        costp_l1 = 0, costq_l1 = 0,
                                        nthread = 1, niter = 10))
```

Atter tuning the optimal parameters is stored in opts$min as below:

```
## $dim
## [1] 1
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.01
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.8497998
```

Next step is using the optimal parameters to train the books rating train dataset and obtained P and Q. Finally, we predict the ratings in the test dataset and calculate the RMSE.

4. Results and Discussion

In this section we will compare performance of the algorithms based on goodbooks-10k data.

4.1 Recommenderlab

We will define the training and test set and create an recommender system. First we need to create the training set and testing set. Training set is set to 80% of the users and 5 ratings of 20% users are set for testing.

```
# Creating training set (80%) and testing set
set.seed(1)
e <- evaluationScheme(real_ratings, method="split", train=0.80, given=-5)
```

Then we use the training and testing set to train the model and predict the ratings for the users in the testing set and calculate the RMSE for the following three algorithms.

RANDOM

```
# Predict ratings for RANDOM algorithm
model <- Recommender(getData(e, "train"), "RANDOM")
prediction <- predict(model, getData(e, "known"), type="ratings")

# Calculation of rmse for RANDOM method
rmse_random <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]
rmse_random
```

```
##     RMSE
## 1.207518
```

POPULAR

```
# Predict ratings for POPULAR algorithm
model <- Recommender(getData(e, "train"), method = "POPULAR", param=list(normalize = "center"))
prediction <- predict(model, getData(e, "known"), type="ratings")

#Calculation of rmse for POPULAR method
rmse_popular <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]
rmse_popular
```

```
##      RMSE
## 0.8944926
```

UBCF

Predicting of ratings and evaluation of UBCF using pearson similarity and selected nn as 50:

```
# Predict ratings for UBCF using pearson similarity
# and selected nn as 50 based on cross-validation
model <- Recommender(getData(e, "train"), method = "UBCF",
                     param=list(normalize = "center", method="pearson", nn=50))
prediction <- predict(model, getData(e, "known"), type="ratings")

#Calculation of rmse for UBCF method
rmse_ubcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]
rmse_ubcf
```

```
##      RMSE
## 0.8927003
```

4.2 Recosystem

Following shows using the optimal parameters to train the books rating train dataset and obtained P and Q.

```
# train it using opts$min
r$train(train_set_mf, opts = opts$min)
```

```
## iter      tr_rmse          obj
##    0       1.4653   1.6259e+06
##    1       0.8473   5.8317e+05
##    2       0.8278   5.5942e+05
##    3       0.8206   5.5082e+05
##    4       0.8166   5.4606e+05
##    5       0.8140   5.4300e+05
##    6       0.8118   5.4036e+05
##    7       0.8105   5.3873e+05
##    8       0.8090   5.3693e+05
##    9       0.8081   5.3579e+05
##   10       0.8072   5.3476e+05
##   11       0.8065   5.3391e+05
##   12       0.8058   5.3303e+05
##   13       0.8051   5.3219e+05
##   14       0.8046   5.3148e+05
##   15       0.8041   5.3091e+05
##   16       0.8036   5.3027e+05
##   17       0.8032   5.2976e+05
##   18       0.8028   5.2927e+05
##   19       0.8024   5.2886e+05
```

```
# res$P and res$Q are the matrix P and Q respectively
res <- r$output(out_memory(), out_memory())
```

Finally, we predict the ratings in the test dataset and calculate the RMSE.

```
#Predicting ratings for test set
predicted_ratings <- r$predict(test_set_mf, out_memory())

#Calculation of rmse for matrix factorization method
rmse_mf <- RMSE(predicted_ratings, test_set$rating)
rmse_mf
```

```
## [1] 0.836229
```

Following shows the final results of RMSE for the above four algorithms:

Method	RMSE
RANDOM	1.2075176
POPULAR	0.8944926
UBCF	0.8927003
MF	0.8362290

5. Conclusion and Discussion

Collaborative filtering is a branch of recommendation that takes account of the information about different users. As expected, our results show that UBCF has lower RMSE than the random or popular algorithms, and matrix factorization by recosystem has the lowest RMSE of 0.836229. For the goodbooks-10k data Matrix factorization using the package "recosystem" performs better than the other three collaborative filtering algorithms in package "recommenderlab". Another advantage of using recosystem is that it is much faster.

There are other algorithms such as item-based collaborative filtering (IBCF) we didn't evaluate due to the calculation time needed. IBCF approach is very similar to user-based, but similarity is computed between items, not users. Assumption is that users will prefer items similar to other items they like. If we have more powerful computer at home, we may try these other algorithms to see if we can achieve lower RMSE on test set than UBCF for given dataset.

Reference

- Michael Hahsler (2019). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-4. https://github.com/mhahsler/recommenderlab
- Yixuan Qiu. recosystem: Recommender System Using Parallel Matrix Factorization. 2017-09-01. https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html
- Package 'recommenderlab'. March 2019. https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf
- Package 'recosystem'. September 2017. https://cran.r-project.org/web/packages/recosystem/recosystem.pdf
- Rafael A. Irizarry; Introduction to Data Science - Data Analysis and Prediction Algorithms with R; 2019-04-22 https://rafalab.github.io/dsbook/