MovieLens Project Submission

Nir Levy, January 2022

Summary

This report describes my work on the MovieLens project, within the Capstone course of Harvardx's Data Science Professional Certificate.
The goal of the project was to predict ratings provided by users to movies. Both users and movies in the test set were included in the training set as well. After downloading, arranging and exploring the data (section 2), I evaluated the performance of several prediction algorithms on the training set (section 3). The Funk Singular Value Decomposition model (SVDF) performed best, but it took a long time to train, so I decided to use the Popular model which performed
reasonably well and ran quickly. I applied the Popular model to the test set and received a Root Mean Squared Error (RMSE) of 0.859 (section 4). My main conclusion is that it is important to be aware of the trade-off between performance and speed. (section 5) Although the SVDF model produced the lowest RMSE, due to my limited processing power it was impractical. The 'Popular' model was much faster and produced a reasonable RMSE, so this seemed like a better choice.

Structure of the report

The report is structured as follows: the first section describes the goal of the analysis. The second section presents the creation of training and test sets, as well as some exploratory analysis. The third section presents some prediction algorithms that I checked out using the training set, before choosing the one that I used for predicting the test set scores (the 'Popular' model). The fourth section presents the results of applying this model on the test set.
Finally, the fifth section presents the conclusion.

Table of contents

1. Introduction

The assignment

The goal of the project was to predict ratings provided by users to movies. Both users and movies in the test set were included in the training set as well. Therefore, the hypothetical scenario is that we already have information about a set of users and a set of movies. However, the information does not cover the ratings given by each user to each movie. It only covers ratings given by each user to a small amount of movies (the matrix in which the rows are users and the columns are movies is very sparse). Based on the ratings that we observe, we wish to predict other ratings within the same set of users and movies.

The analysis

The key steps of the analysis were as follows: 1. Downloading, arranging and exploring the data (section 2) 2. Evaluating the performance of various prediction algorithms on the training set (section 3) 3. Applying the chosen model ('Popular') to the test set and calculating the RMSE (section 4)

The following sections present these steps and the conclusion.

2. Creating the data and exploring it

I begin by downloading the data, using the code provided by the course. Since it is provided by the course, I have hidden it in the report. See the R script or the Rmd file for the code.

Next, I save the files.

```
### saving the training and test sets ###
saveRDS(movies, file="movies")
saveRDS(ratings, file="ratings")
saveRDS(edx, file="edx")
saveRDS(validation, file="validation")
saveRDS(movielens, file="movielens")
```

Now I examine the data by running some analyses and creating some plots.

```
Sys.time() # recording the start time
```

```
## [1] "2022-01-10 07:03:30 CET"
```

```
### Exploring the training set ###
dim(edx)
```

```
## [1] 9000055       6
```

```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```r
sum(is.na(edx)) # counting missing values
```

```
## [1] 0
```

```r
# counting the unique values
n_distinct(edx$userId) # users
```

```
## [1] 69878
```

```r
n_distinct(edx$movieId) # movies
```

```
## [1] 10677
```

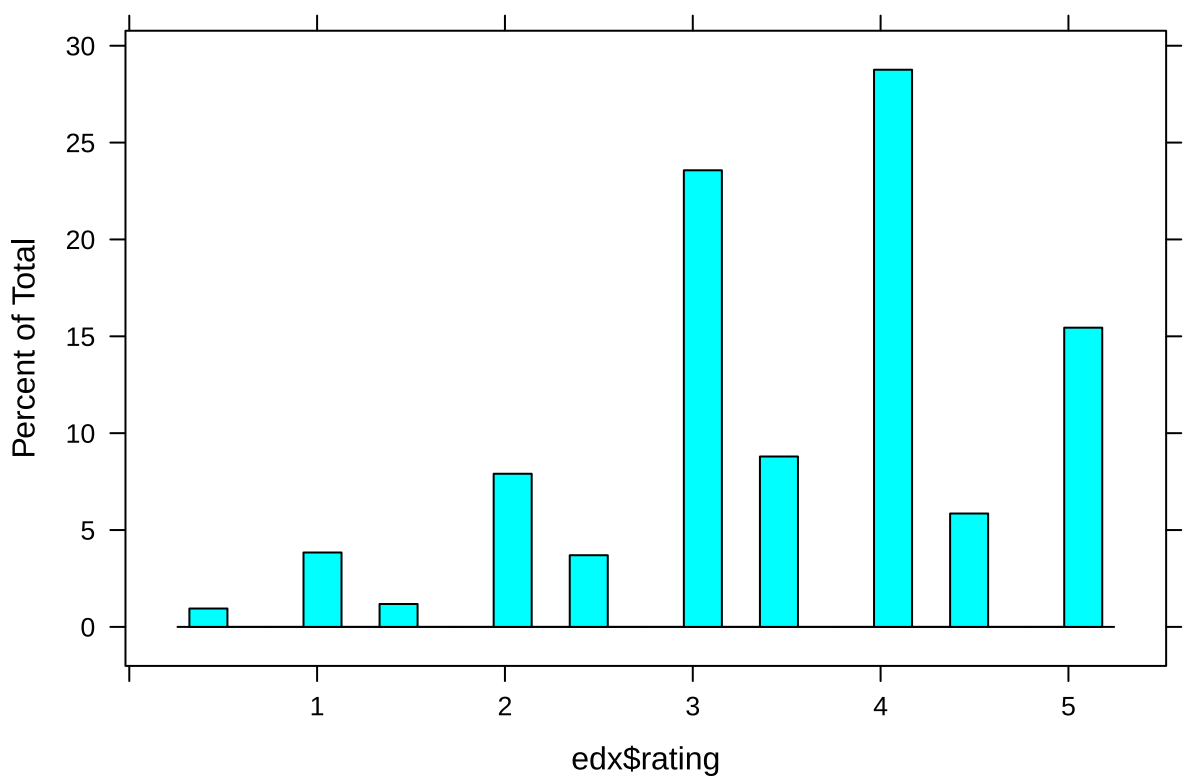
```r
n_distinct(edx$rating) # ratings
```

```
## [1] 10
```

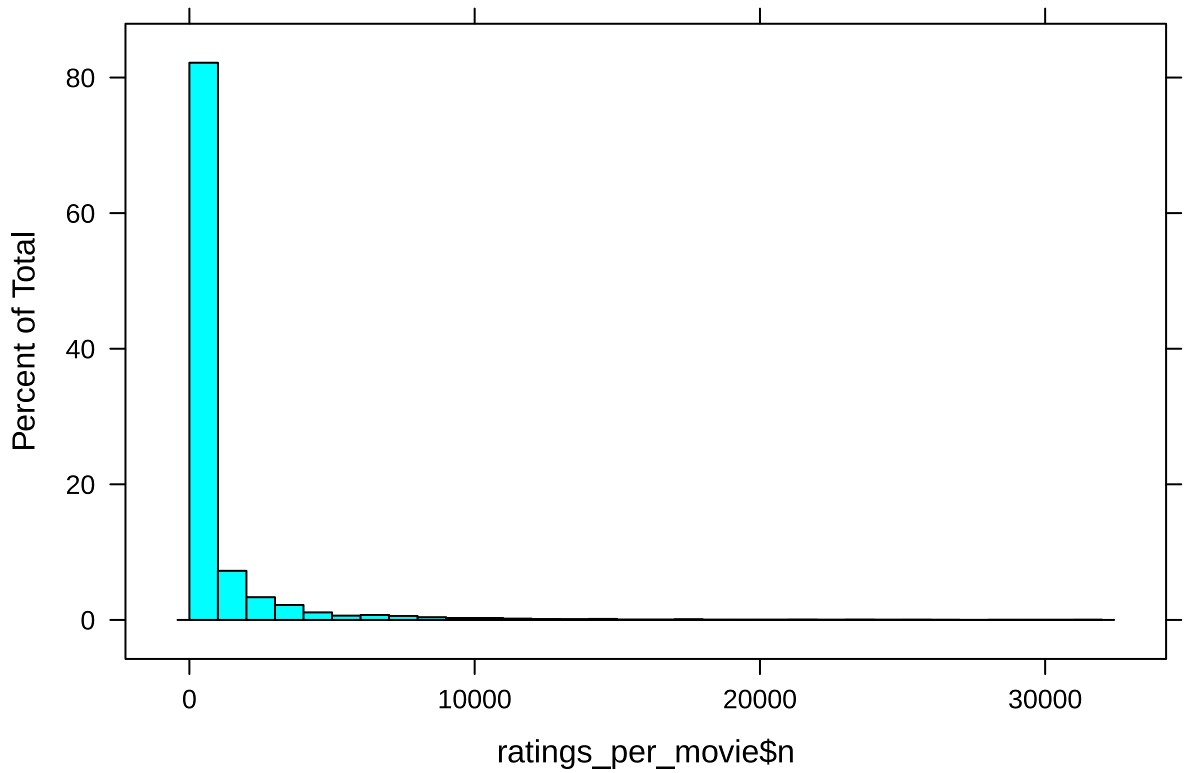
```r
n_distinct(edx$genres) # genres
```

```
## [1] 797
```

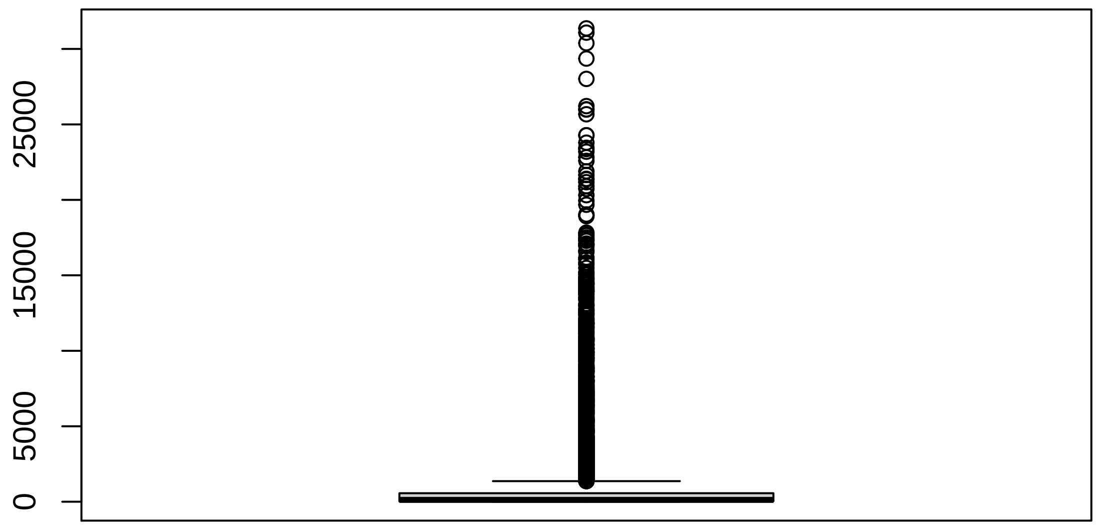
```r
# plotting the distributions of the ratings
histogram(edx$rating)
```



```r
# plotting the distribution of the ratings per movie
ratings_per_movie<-edx %>%
  count(movieId)
histogram(ratings_per_movie$n, breaks=30)
```




```r
boxplot(ratings_per_movie$n)
```



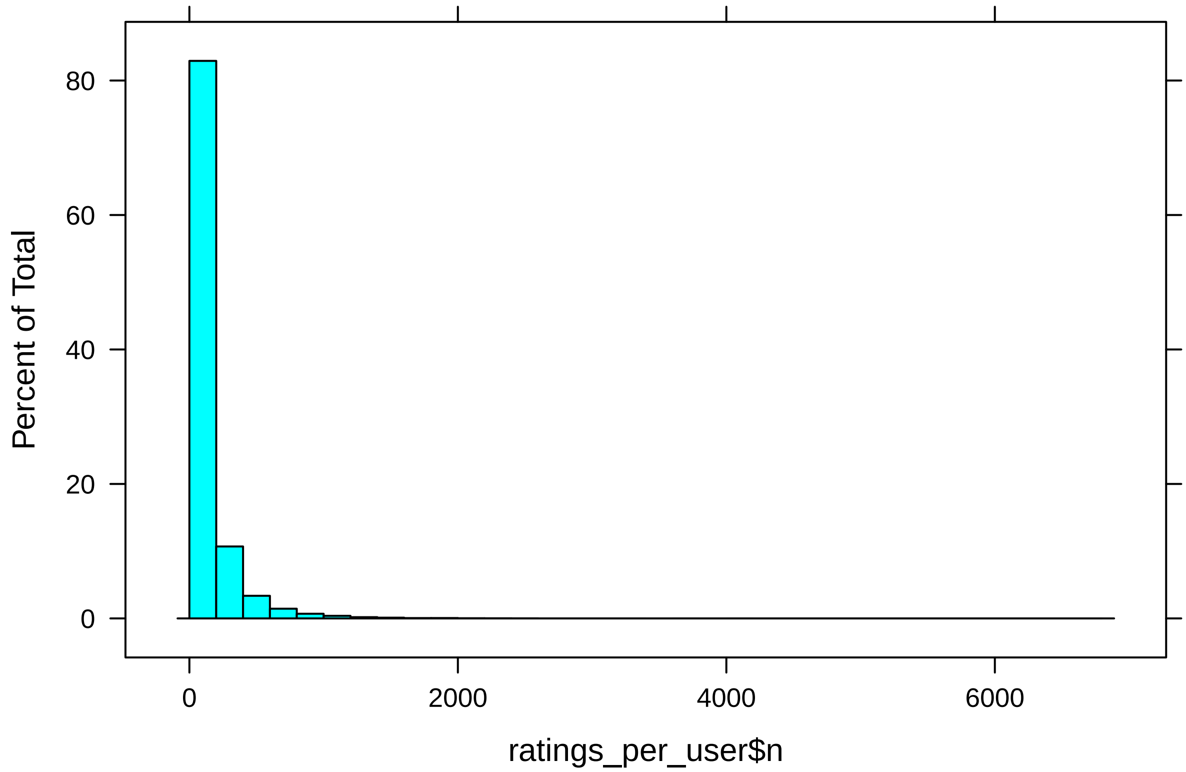

```
summary(ratings_per_movie$n)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##      1.0     30.0    122.0    842.9    565.0 31362.0
```

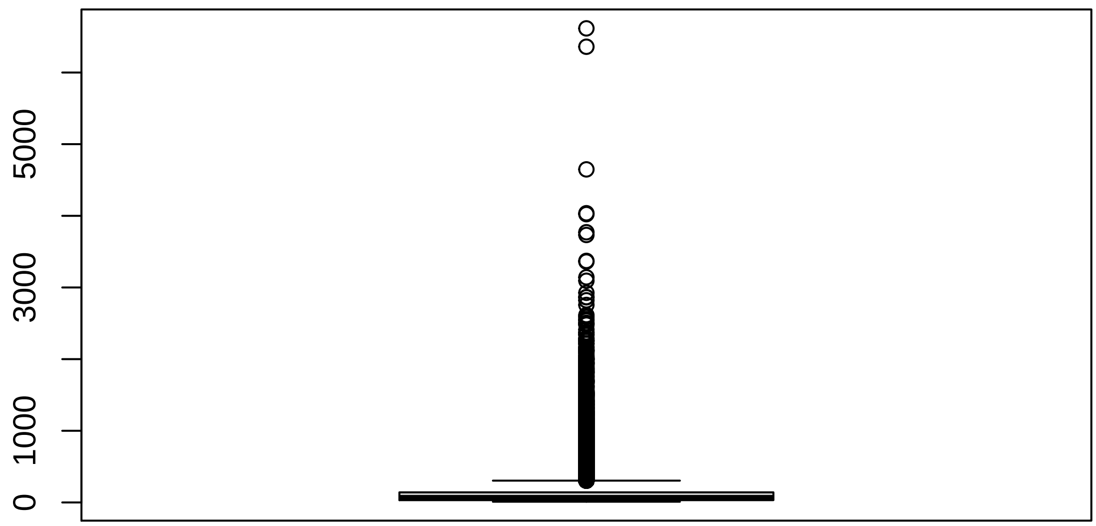
```
rm(ratings_per_movie) # removing the variable

# plotting the distribution of ratings per user
ratings_per_user<-edx %>%
  filter(!is.na(rating))  %>%
  count(userId)

histogram(ratings_per_user$n, breaks=30)
```




```r
boxplot(ratings_per_user$n)
```




```r
summary(ratings_per_user$n)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    32.0    62.0   128.8   141.0  6616.0
```

```r
rm(ratings_per_user) # removing the variable
```

3. Choosing a prediction algorithm

Now I evaluate several prediction models, using the training set. I apply the Leave One Out Cross Validation (LOOCV) method, due to a processing power limitation. Other methods took too much time to run. This section is structured as follows: a. Evaluating the 'movie and user fixed effects' model b. Adding genre fixed effects to the model c. Evaluating various algorithms: Popular, IBCF, UBCF, SVD, SVDF d. Choosing a model to apply to the test set, based on the results

a. Evaluating the 'movie and user fixed effects' model

I begin by evaluating the 'movie and user fixed effects' model that was presented in the machine learning course.

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:03:35 CET"
```

```r
# clearing memory
invisible(gc())

# Increasing memory size
memory.limit(size = 10^10)
```

```
## [1] 1e+10
```

```r
### The validation method is "Leave One Out Cross Validation (LOOCV)" ###

### creating a subset of the training set, for testing the model ###
# The core training set will be called 'core' and the subset of the training set
# for validation will be called 'sub'.
# (I am avoiding the term `validation set`, since edx called the test set 'validation set'.)

# The 'sub' set will be 15% of the training set
 sub_index <- createDataPartition(y = edx$rating, times = 1, p = 0.15, list = FALSE)
 core <- edx[-sub_index,]
 temp <- edx[sub_index,]

# Making sure userId and movieId in sub set are also in core set
sub <- temp %>%
  semi_join(core, by = "movieId") %>%
  semi_join(core, by = "userId")

# Adding rows removed from sub set back into core set
removed <- anti_join(temp, sub)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```r
core <- rbind(core, removed)

# removing unnecessary objects
rm(sub_index, temp, removed)

### The first model ###
# Predicting only according to the average rating in the dataset ###
mu <- mean(core$rating)
mu
```

```
## [1] 3.512513
```

```r
# checking the root mean squared error
sub$average<-mu
naive_rmse <- RMSE(sub$rating, sub$average)
naive_rmse
```

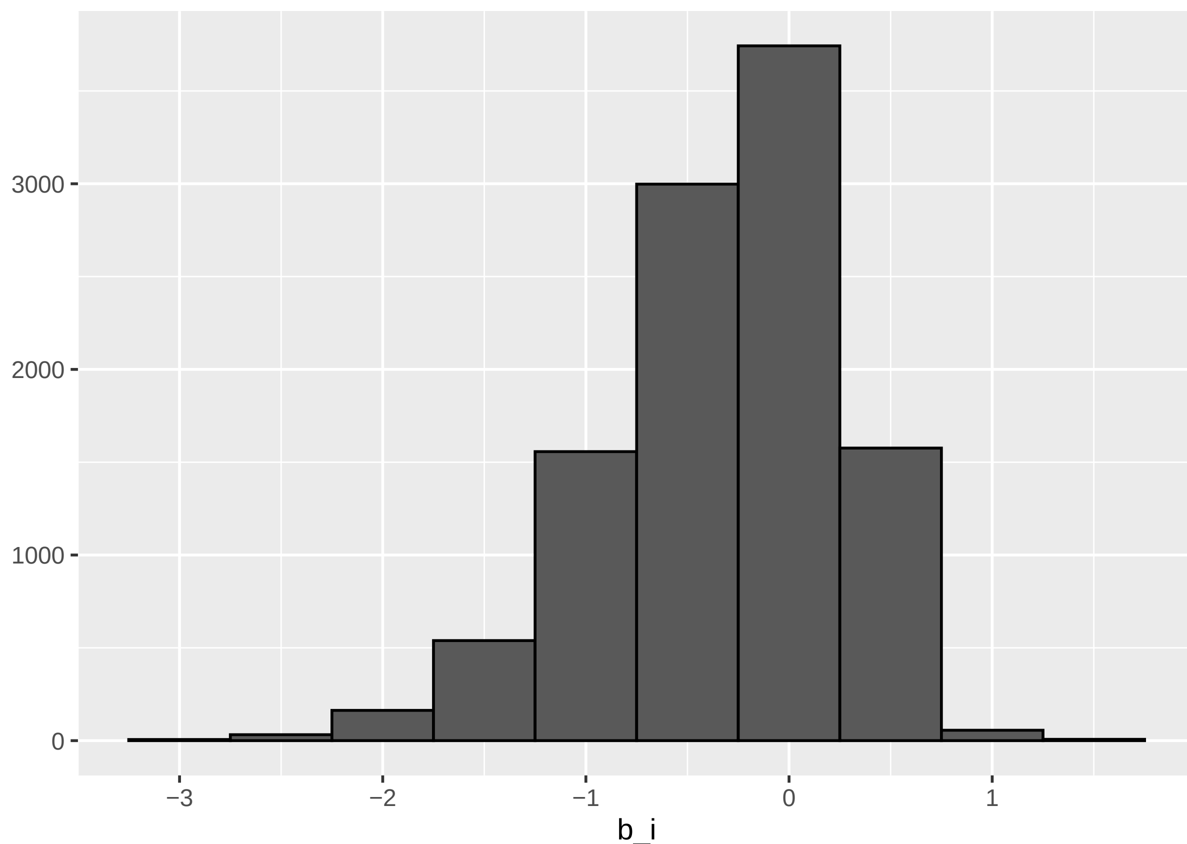
```
## [1] 1.061102
```

```r
# creating a results table
suppressWarnings(rm(rmse_results)) # removing this object if I already created it when running the code
rmse_results <- c("Just the average", round(naive_rmse,5))
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##             method               RMSE
## "Just the average"           "1.0611"
```

```r
### adding movie effects ###
movie_avgs <- core %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# examining the distributions of constant movie effects
movie_effects_plot<-qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
movie_effects_plot
```

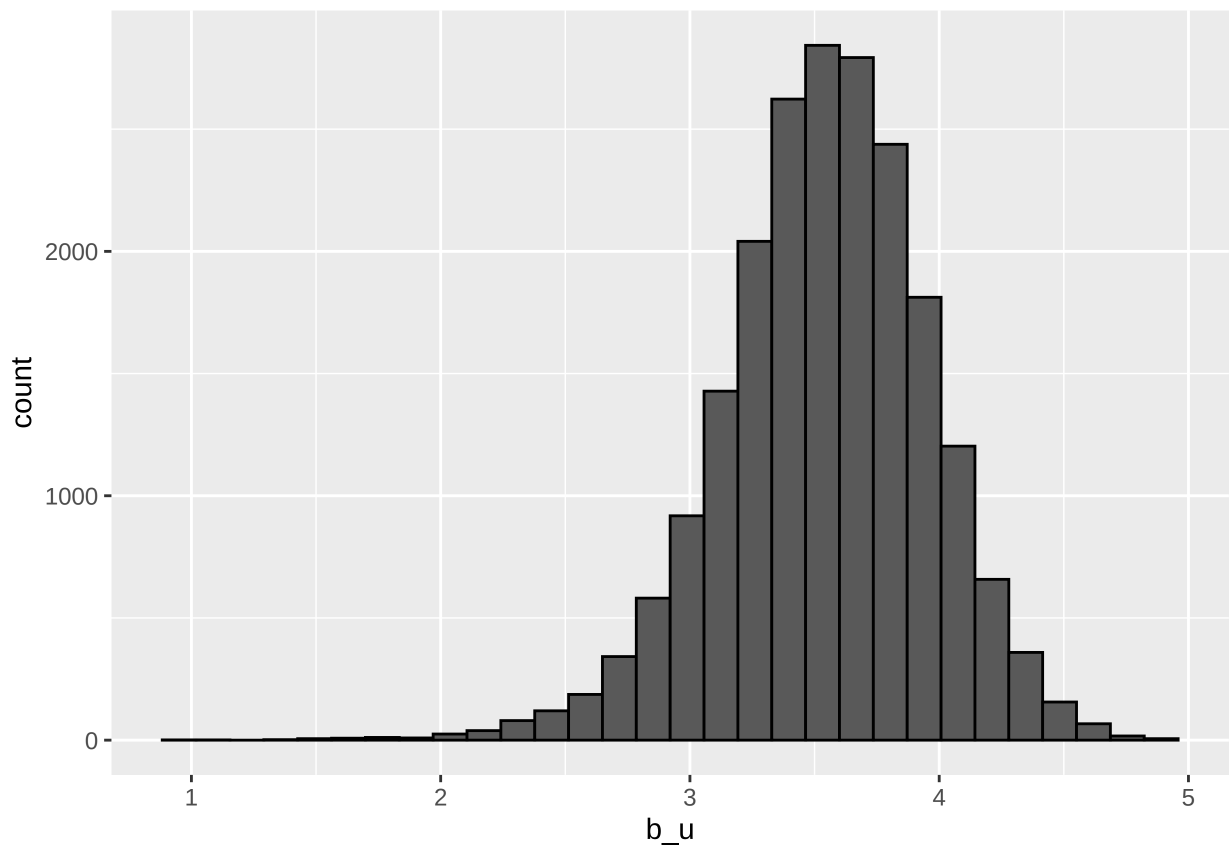



```r
# checking the rmse
predicted_ratings <- mu + sub %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
movie_effects_rmse<-RMSE(predicted_ratings, sub$rating)
```

```r
rmse_results <- rbind.data.frame(rmse_results, c("With unregularized fixed movie effects", round(movie_
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##                                   method    RMSE
## 1                       Just the average  1.0611
## 2 With unregularized fixed movie effects 0.94397
```

```r
### adding user effects ###
# examining the average rating per user
core %>%
 group_by(userId) %>%
 filter(n()>=100) %>%
 summarize(b_u = mean(rating)) %>%
 ggplot(aes(b_u)) +
 geom_histogram(bins = 30, color = "black")
```




```r
# estimating the user effects, by computing
# the overall average and the item effect, and then
# the user effect is the average of the remainder, after they
# are substacted from the rating (user effect= average of (rating - overall average - item effect)
user_avgs <- core %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
```

```r
  summarize(b_u = mean(rating - mu - b_i))

# checking the rmse
predicted_ratings <- sub %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
movie_and_user_effects_rmse<-RMSE(predicted_ratings, sub$rating)

rmse_results <- rbind.data.frame(rmse_results, c("With unregularized movie and user fixed effects", rou
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##                                            method    RMSE
## 1                                Just the average  1.0611
## 2          With unregularized fixed movie effects 0.94397
## 3 With unregularized movie and user fixed effects  0.8659
```

```r
# adding regularization
# Regularizing with lambda = 3
lambda <- 3

movie_avgs <- core %>%
  left_join(user_avgs, by='userId') %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu - b_u)/(n()+lambda), n_i = n())

### Calculating RMSE ###
predicted_ratings <- sub %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
movie_and_user_effects_with_regularization_rmse<-RMSE(predicted_ratings, sub$rating)

# checking the RMSE
rmse_results <- rbind.data.frame(rmse_results,c("Movie and user effects with regularization", round(movi
# rmse_results <- rbind(rmse_results, c("Movie and user effects with regularization", round(movie_and_u
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##                                            method    RMSE
## 1                                Just the average  1.0611
## 2          With unregularized fixed movie effects 0.94397
## 3 With unregularized movie and user fixed effects  0.8659
## 4      Movie and user effects with regularization 0.86464
```

Interim learnings, based on the RMSE table: 1. Adding user fixed effects significantly improves the RMSE 2. Regularization does not improve the RMSE in our case

b. Adding genre fixed effects to the model

In addition to movie and user effects, genres might also have fixed effects. That is, some genres may have a higher or lower rating than the average rating of all movies. In order to account for this, I add genre fixed effects to the model. From here on, until the end of the analysis, we depart from the code that was provided in the Machine Learning course.

```
# adding genre fixed effects with regularization

# genre fixed effects with regularization
# creating numeric genre column
core$genresnum<-as.numeric(as.factor(core$genres))

genre_avgs <- core %>%
  left_join(user_avgs) %>%
  left_join(movie_avgs) %>%
  group_by(genresnum) %>%
  summarize(b_g = mean(rating - mu-b_u-b_i)/(n()+lambda), n_i = n())
```

```
## Joining, by = "userId"

## Joining, by = "movieId"
```

```
# creating a numeric user_genres column in the
# sub dataset
sub$genresnum<-as.numeric(as.factor(sub$genres))

### Calculating RMSE ###
predicted_ratings <- sub %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genresnum') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
movie_user_genre_effects<-RMSE(predicted_ratings, sub$rating)

# checking the RMSE
rmse_results <- rbind(rmse_results, c("Movie, user and genre effects with regularization", round(movie_u
names(rmse_results)<-c("method", "RMSE")
rmse_results
```

```
##                                             method    RMSE
## 1                                 Just the average  1.0611
## 2            With unregularized fixed movie effects 0.94397
## 3   With unregularized movie and user fixed effects  0.8659
## 4          Movie and user effects with regularization 0.86464
## 5 Movie, user and genre effects with regularization 0.86466
```

Interim learning: adding genre fixed effects does not improve the RMSE

c. Evaluating various algorithms: Popular, IBCF, UBCF, SVD, SVDF

Now let us evaluate more advanced preduction models, namely: 1. Popular 2. Item Based Collaborative Filtering (IBCF) 3. User Based Collaborative Filtering (UBCF) 4. Singular Value Decomposition (SVD) 5. Funk Singular Value Decomposition (SVDF)

```
# Increasing memory size
memory.limit(size = 10^10)
```

```
## [1] 1e+10
```

```
# cleaning memory
invisible(gc())

### Preparing the data ###
# removing movies in the training set that do not appear in the test set
edx_reduced <- edx %>%
  semi_join(validation, by = "movieId")

# exploring the datasets
dim(edx)
```

```
## [1] 9000055       6
```

```
dim(edx_reduced)
```

```
## [1] 8993159       6
```

```
# saving
saveRDS(edx_reduced, file="edx_reduced")

# converting the training set into a matrix
users_and_ratings_train_set<-cbind.data.frame(edx_reduced$userId, edx_reduced$movieId, edx_reduced$rati
dim(users_and_ratings_train_set)
```

```
## [1] 8993159       3
```

```
# converting the matrix into a "realRatingMatrix")
trainmat_reduced <- as(users_and_ratings_train_set, "realRatingMatrix")
dim(trainmat_reduced)
```

```
## [1] 69878  9809
```

```
# saving
saveRDS(trainmat_reduced, file="trainmat_reduced")

# creating a regular matrix
trainmat_reduced_reg<-as(trainmat_reduced, "matrix")

# saving
```

```r
saveRDS(trainmat_reduced_reg, file="trainmat_reduced_reg")

# exploring the matrix #
class(trainmat_reduced_reg)
```

```
## [1] "matrix" "array"
```

```r
n_missing<-sum(is.na(trainmat_reduced_reg)) # counting missing values
n_missing
```

```
## [1] 676440143
```

```r
all<-nrow(trainmat_reduced_reg)*ncol(trainmat_reduced_reg) # counting all values
all
```

```
## [1] 685433302
```

```r
p_missing<-n_missing/all # calculating the percentage of missing values
p_missing
```

```
## [1] 0.9868796
```

```r
rm(trainmat_reduced_reg, n_missing, all) # removing the matrix and other unnecessary objects
```

Since over 98% percent of the values are missing, I add SVD and SVDF to the models that I evaluate. These model are supposed to deal with sparse matrices well.

Since my processing power is limiited, I evaluate the models on only 10% of the users in the training set. Otherwise the evaluation takes too much time. I choose the 10% of the users who gave the most ratings.

```r
# Increasing memory size
memory.limit(size = 10^10)
```

```
## [1] 1e+10
```

```r
# cleaning memory
invisible(gc())

# Calculating the 90th percentile of the number of ratings per user
min_n_users <- quantile(rowCounts(trainmat_reduced), 0.9)
min_n_users
```

```
## 90%
## 301
```

```r
# Keeping only users who gave many ratings (the top 10% of the number of ratings)
# to reduce computation time. Otherwise the analysis runs
# very slowly on my computer

trainmat_final_10 <- trainmat_reduced[rowCounts(trainmat_reduced) > min_n_users,]
dim(trainmat_final_10)
```

```
## [1] 6976 9809
```

```r
# making sure that 10% of the users are indeed in the final training set
nrow(trainmat_reduced)
```

```
## [1] 69878
```

```r
# checking number of ratings per item
number_of_ratings<-colCounts(trainmat_final_10)
min(number_of_ratings)
```

```
## [1] 0
```

```r
max(number_of_ratings)
```

```
## [1] 5501
```

```r
# exploring a sample of the matrix
trainmat_final_10@data[1500:1510, 2001:2009]
```

```
## 11 x 9 sparse Matrix of class "dgCMatrix"
##       2117 2118 2119 2120 2121 2122 2123 2124 2125
## 16029    4    .    .    .    .    .  .    .      .
## 16043    .    .    .    .    .    .  .    .      .
## 16050    .    .    .    .    .    .  .    .      .
## 16056    .    .    .    .    .    .  .    .      .
## 16058    .    .    .    .    .    .  3.0  4.0    3
## 16067    .    .    .    .    .    .  .    .      .
## 16069    .    .    .    .    .    .  .    .      .
## 16073    .    .    .    .    .    .  .    .      .
## 16077    .    .    .    .    .    .  .    .      .
## 16078    .    .    .    .    .    .  4.5  4.5    .
## 16082    .    .    .    .    .    .  .    .      .
```

```r
# normalizing the values
normalize(trainmat_final_10, method = "Z-score")
```

```
## 6976 x 9809 rating matrix of class 'realRatingMatrix' with 3999667 ratings.
## Normalized using z-score on rows.
```

```r
# saving
saveRDS(trainmat_final_10, file="trainmat_final_10")

# Setting up the evaluation scheme (leave one out cross validation (LOOCV),
# reserving 10% of the training set for validation)

Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:06:27 CET"
```

```r
scheme_10 <- trainmat_final_10 %>%
  evaluationScheme(method = "split",
                   k=1,
                   train  = 0.9,  # 90% data train
                   given  = -8,
                   goodRating = 3.5
  )

Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:07:45 CET"
```

```r
# saving
saveRDS(scheme_10, file="scheme_10")

Sys.time()
```

```
## [1] "2022-01-10 07:07:50 CET"
```

```r
##################### Evaluating the models ################################

### Popular ###
# evaluating the "popular" model

Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:07:50 CET"
```

```r
result_rating_popular_10 <- evaluate(scheme_10,
                                     method = "popular",
                                     parameter = list(normalize = "Z-score"),
                                     type   = "ratings"
)
```

```
## popular run fold/sample [model time/prediction time]
##   1  [0.71sec/1.29sec]
```

```r
# examining the results
result_rating_popular_10@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))
```

```
##        RMSE       MSE       MAE
## 1 0.8588175 0.7375674 0.6530973
```

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:07:54 CET"
```

```r
# saving
saveRDS(result_rating_popular_10, file="result_rating_popular_10")

### svd ###
# evaluating the svd model
result_rating_svd_10 <- evaluate(scheme_10,
                              method = "svd",
                              parameter = list(normalize = "Z-score", k = 5),
                              type  = "ratings"
)
```

```
## svd run fold/sample [model time/prediction time]
##   1  [5.07sec/3.62sec]
```

```r
# examining the results
result_rating_svd_10@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))
```

```
##        RMSE       MSE       MAE
## 1 0.9191795 0.8448909 0.7176927
```

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:08:04 CET"
```

```r
### ibcf ###

# evaluating
result_rating_ibcf_10 <- evaluate(scheme_10,
                                  method = "ibcf",
                                  parameter = list(normalize = "Z-score"),
                                  type  = "ratings"
)
```

```
## ibcf run fold/sample [model time/prediction time]
##   1  [2397.95sec/1.81sec]
```

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:48:05 CET"
```

```r
# saving
saveRDS(result_rating_ibcf_10, file="result_rating_ibcf_10")

# examining the results
result_rating_ibcf_10@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))
```

```
##       RMSE      MSE      MAE
## 1 1.248455 1.558641 0.950424
```

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:48:05 CET"
```

```r
### ubcf ###

# evaluating
result_rating_ubcf_10 <- evaluate(scheme_10,
                                  method = "ubcf",
                                  parameter = list(normalize = "Z-score", k = 5),
                                  type   = "ratings"
)
```

```
## ubcf run fold/sample [model time/prediction time]
##   1
```

```
## Warning: Unknown parameters: k
```

```
## Available parameter (with default values):
## method    =  cosine
## nn    =  25
## sample    =  FALSE
## weighted  =  TRUE
## normalize     =  center
## min_matching_items    =  0
## min_predictive_items  =  0
## verbose   =  FALSE
## [0.53sec/275.9sec]
```

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 07:52:43 CET"
```

```r
# saving
saveRDS(result_rating_ubcf_10, file="result_rating_ubcf_10")

# examining the results
result_rating_ubcf_10@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))
```

```
##        RMSE       MSE       MAE
## 1 0.8653869 0.7488945 0.6472406
```

```r
# saving
saveRDS(result_rating_svd_10, file="result_rating_svd_10")

### svdf ###

# evaluating
result_rating_svdf_10 <- evaluate(scheme_10,
                                  method = "svdf",
                                  parameter = list(normalize = "Z-score", k = 5),
                                  type  = "ratings"
)
```

```
## svdf run fold/sample [model time/prediction time]
##   1  [1329.49sec/47.08sec]
```

```r
Sys.time() # recording the time in order to see how long each step takes
```

```
## [1] "2022-01-10 08:15:45 CET"
```

```r
# saving
saveRDS(result_rating_svdf_10, file="result_rating_svdf_10")

# examining the results
result_rating_svdf_10@results %>%
  map(function(x) x@cm) %>%
  unlist() %>%
  matrix(ncol = 3, byrow = T) %>%
  as.data.frame() %>%
  summarise_all(mean) %>%
  setNames(c("RMSE", "MSE", "MAE"))
```

```
##        RMSE      MSE       MAE
## 1 0.7986319 0.637813 0.6053268
```

The ‘Popular’ algorithm seems like a reasonable choice, since it’s run time is short and it is relatively accurate (only the SVDF algorithm produces a lower RMSE, but SVDF takes a long time to run).

Now I train the Popular algorithm on the same 10% of the data that I used for evaluation.

```r
# training the algorithm on one tenth of the training set
# (training it on the full set took too much time)
recommendations_pop_10 <- Recommender(trainmat_final_10, method = "popular")

Sys.time() # noting the time in order to measure the time the next command takes
```

```
## [1] "2022-01-10 08:15:46 CET"
```

```r
# saving
saveRDS(recommendations_pop_10, file="recommendations_pop_10")
```

4. Applying the model to the test set In this final part of the analysis I predict the ratings in the test set, using the trained Popular algorithm. Running the analysis on the whole test set took too much time, so I break it up into ten parts.

```r
# Increasing memory size
memory.limit(size = 10^10)
```

```
## [1] 1e+10
```

```r
# cleaning memory
invisible(gc())

### converting the testing set into a reaRatingmatrix ###
# creating userId,  movieId and rating columns
users_and_ratings_test_set<-cbind.data.frame(validation$userId, validation$movieId, validation$rating)

# exploring users_and_ratings_test_set
dim(users_and_ratings_test_set)
```

```
## [1] 999999      3
```

```r
head(users_and_ratings_test_set)
```

```
##   validation$userId validation$movieId validation$rating
## 1                 1                231                 5
## 2                 1                480                 5
## 3                 1                586                 5
## 4                 2                151                 3
## 5                 2                858                 2
## 6                 2               1544                 3
```

```r
# creating the matrix
testmat <- as(users_and_ratings_test_set, "realRatingMatrix")
dim(testmat)
```

```
## [1] 68534  9809
```

```r
# checking the number of ratings per item
number_of_ratings<-colCounts(testmat)
min(number_of_ratings)
```

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

```r
max(number_of_ratings)
```

```
## [1] 3502
```

```r
# creating an index of rows in the test matrix
index<-seq(1:nrow(testmat))
length(index) # making sure that the index was created properly
```

```
## [1] 68534
```

```r
### Splitting the test matrix intro 10 equal parts ###
### to shorten processing time ###

# splitting the index into 10 equal parts
splitted_index<-split(index,              # Applying split() function
                      cut(seq_along(index),
                          10,
                          labels = FALSE))
# splitted_index
tenth_1<-as.vector(splitted_index[[1]])
tenth_2<-as.vector(splitted_index[[2]])
tenth_3<-as.vector(splitted_index[[3]])
tenth_4<-as.vector(splitted_index[[4]])
tenth_5<-as.vector(splitted_index[[5]])
tenth_6<-as.vector(splitted_index[[6]])
tenth_7<-as.vector(splitted_index[[7]])
tenth_8<-as.vector(splitted_index[[8]])
tenth_9<-as.vector(splitted_index[[9]])
tenth_10<-as.vector(splitted_index[[10]])

# verifying that all of the 10ths together comprise the total number of users
# in the test set
total<-length(tenth_1)+length(tenth_2)+length(tenth_3)+length(tenth_4)+length(tenth_5)+
 length(tenth_6)+length(tenth_7)+length(tenth_8)+length(tenth_9)+length(tenth_10)
total
```

```
## [1] 68534
```

```r
nrow(testmat)
```

```
## [1] 68534
```

```r
# splitting the test set into ten parts
testmat_10_1<-testmat[tenth_1]
testmat_10_2<-testmat[tenth_2]
testmat_10_3<-testmat[tenth_3]
testmat_10_4<-testmat[tenth_4]
testmat_10_5<-testmat[tenth_5]
testmat_10_6<-testmat[tenth_6]
testmat_10_7<-testmat[tenth_7]
testmat_10_8<-testmat[tenth_8]
testmat_10_9<-testmat[tenth_9]
testmat_10_10<-testmat[tenth_10]

# examining some of the tenths, to make sure that
# they were created properly
dim(testmat_10_1)
```

```
## [1] 6854 9809
```

```r
dim(testmat_10_7)
```

```
## [1] 6854 9809
```

```r
dim(testmat_10_9)
```

```
## [1] 6853 9809
```

```r
dim(testmat_10_10)
```

```
## [1] 6854 9809
```

```r
# removing unnecessary files from the workspace
rm(tenth_1, tenth_2, tenth_3, tenth_4, tenth_5)
rm(tenth_6, tenth_7, tenth_8, tenth_9, tenth_10)

# turning the tenths of the test set into matrices
testmat_10_1_matrix<-as(testmat_10_1, "matrix")
saveRDS(testmat_10_1_matrix, file="testmat_10_1_matrix")

testmat_10_2_matrix<-as(testmat_10_2, "matrix")
saveRDS(testmat_10_2_matrix, file="testmat_10_2_matrix")

testmat_10_3_matrix<-as(testmat_10_3, "matrix")
saveRDS(testmat_10_3_matrix, file="testmat_10_3_matrix")

testmat_10_4_matrix<-as(testmat_10_4, "matrix")
saveRDS(testmat_10_4_matrix, file="testmat_10_4_matrix")

testmat_10_5_matrix<-as(testmat_10_5, "matrix")
saveRDS(testmat_10_5_matrix, file="testmat_10_5_matrix")
```

```
testmat_10_6_matrix<-as(testmat_10_6, "matrix")
saveRDS(testmat_10_6_matrix, file="testmat_10_6_matrix")

testmat_10_7_matrix<-as(testmat_10_7, "matrix")
saveRDS(testmat_10_7_matrix, file="testmat_10_7_matrix")

testmat_10_8_matrix<-as(testmat_10_8, "matrix")
saveRDS(testmat_10_8_matrix, file="testmat_10_8_matrix")

testmat_10_9_matrix<-as(testmat_10_9, "matrix")
saveRDS(testmat_10_9_matrix, file="testmat_10_9_matrix")

testmat_10_10_matrix<-as(testmat_10_10, "matrix")
saveRDS(testmat_10_10_matrix, file="testmat_10_10_matrix")

# Creating the predictions, using the 'popular' method

# Predicting the ratings in the validation set, one tenth at a time:
predictions_pop_10_1 <- predict(recommendations_pop_10, testmat_10_1, type="ratingMatrix")
saveRDS(predictions_pop_10_1, file="predictions_pop_10_1") # saving the file

predictions_pop_10_2 <- predict(recommendations_pop_10, testmat_10_2, type="ratingMatrix")
saveRDS(predictions_pop_10_2, file="predictions_pop_10_2") # saving the file

predictions_pop_10_3 <- predict(recommendations_pop_10, testmat_10_3, type="ratingMatrix")
saveRDS(predictions_pop_10_3, file="predictions_pop_10_3") # saving the file

predictions_pop_10_4 <- predict(recommendations_pop_10, testmat_10_4, type="ratingMatrix")
saveRDS(predictions_pop_10_4, file="predictions_pop_10_4") # saving the file

predictions_pop_10_5 <- predict(recommendations_pop_10, testmat_10_5, type="ratingMatrix")
saveRDS(predictions_pop_10_5, file="predictions_pop_10_5") # saving the file

predictions_pop_10_6 <- predict(recommendations_pop_10, testmat_10_6, type="ratingMatrix")
saveRDS(predictions_pop_10_6, file="predictions_pop_10_6") # saving the file

predictions_pop_10_7 <- predict(recommendations_pop_10, testmat_10_7, type="ratingMatrix")
saveRDS(predictions_pop_10_7, file="predictions_pop_10_7") # saving the file

predictions_pop_10_8 <- predict(recommendations_pop_10, testmat_10_8, type="ratingMatrix")
saveRDS(predictions_pop_10_8, file="predictions_pop_10_8") # saving the file

predictions_pop_10_9 <- predict(recommendations_pop_10, testmat_10_9, type="ratingMatrix")
saveRDS(predictions_pop_10_9, file="predictions_pop_10_9") # saving the file

predictions_pop_10_10 <- predict(recommendations_pop_10, testmat_10_10, type="ratingMatrix")
saveRDS(predictions_pop_10_10, file="predictions_pop_10_10") # saving the file

Sys.time()
```

```
## [1] "2022-01-10 08:21:30 CET"
```

```r
# removing files to free up memory
rm(testmat_10_1, testmat_10_2, testmat_10_3, testmat_10_4, testmat_10_5)
rm(testmat_10_6, testmat_10_7, testmat_10_8, testmat_10_9, testmat_10_10)
rm(splitted_index, testmat)

# cleaning memory
invisible(gc())

# turning the results into matrices
predictions_pop_10_1<-readRDS("predictions_pop_10_1")
predmat_pop_10_1<-as(predictions_pop_10_1, "matrix")
saveRDS(predmat_pop_10_1, file="predmat_pop_10_1")
rm(predictions_pop_10_1, predmat_pop_10_1) # removing files to free up memory

predictions_pop_10_2<-readRDS("predictions_pop_10_2")
predmat_pop_10_2<-as(predictions_pop_10_2, "matrix")
saveRDS(predmat_pop_10_2, file="predmat_pop_10_2")
rm(predictions_pop_10_2, predmat_pop_10_2) # removing files to free up memory

predictions_pop_10_3<-readRDS("predictions_pop_10_3")
predmat_pop_10_3<-as(predictions_pop_10_3, "matrix")
saveRDS(predmat_pop_10_3, file="predmat_pop_10_3")
rm(predictions_pop_10_3, predmat_pop_10_3) # removing files to free up memory

predictions_pop_10_4<-readRDS("predictions_pop_10_4")
predmat_pop_10_4<-as(predictions_pop_10_4, "matrix")
saveRDS(predmat_pop_10_4, file="predmat_pop_10_4")
rm(predictions_pop_10_4, predmat_pop_10_4) # removing files to free up memory

predictions_pop_10_5<-readRDS("predictions_pop_10_5")
predmat_pop_10_5<-as(predictions_pop_10_5, "matrix")
saveRDS(predmat_pop_10_5, file="predmat_pop_10_5")
rm(predictions_pop_10_5, predmat_pop_10_5) # removing files to free up memory

predictions_pop_10_6<-readRDS("predictions_pop_10_6")
predmat_pop_10_6<-as(predictions_pop_10_6, "matrix")
saveRDS(predmat_pop_10_6, file="predmat_pop_10_6")
rm(predictions_pop_10_6, predmat_pop_10_6) # removing files to free up memory

predictions_pop_10_7<-readRDS("predictions_pop_10_7")
predmat_pop_10_7<-as(predictions_pop_10_7, "matrix")
saveRDS(predmat_pop_10_7, file="predmat_pop_10_7")
rm(predictions_pop_10_7, predmat_pop_10_7) # removing files to free up memory

predictions_pop_10_8<-readRDS("predictions_pop_10_8")
predmat_pop_10_8<-as(predictions_pop_10_8, "matrix")
saveRDS(predmat_pop_10_8, file="predmat_pop_10_8")
rm(predictions_pop_10_8, predmat_pop_10_8) # removing files to free up memory

predictions_pop_10_9<-readRDS("predictions_pop_10_9")
predmat_pop_10_9<-as(predictions_pop_10_9, "matrix")
saveRDS(predmat_pop_10_9, file="predmat_pop_10_9")
rm(predictions_pop_10_9, predmat_pop_10_9) # removing files to free up memory
```

```
predictions_pop_10_10<-readRDS("predictions_pop_10_10")
predmat_pop_10_10<-as(predictions_pop_10_10, "matrix")
saveRDS(predmat_pop_10_10, file="predmat_pop_10_10")
rm(predictions_pop_10_10, predmat_pop_10_10) # removing files to free up memory

# cleaning memory
invisible(gc())

# loading the prediction files that were created in the code above
predmat_pop_10_1<-readRDS("predmat_pop_10_1")
predmat_pop_10_2<-readRDS("predmat_pop_10_2")
predmat_pop_10_3<-readRDS("predmat_pop_10_3")
predmat_pop_10_4<-readRDS("predmat_pop_10_4")
predmat_pop_10_5<-readRDS("predmat_pop_10_5")
predmat_pop_10_6<-readRDS("predmat_pop_10_6")
predmat_pop_10_7<-readRDS("predmat_pop_10_7")
predmat_pop_10_8<-readRDS("predmat_pop_10_8")
predmat_pop_10_9<-readRDS("predmat_pop_10_9")
predmat_pop_10_10<-readRDS("predmat_pop_10_10")

# calculating the differences
diffmat_10_1<-predmat_pop_10_1-testmat_10_1_matrix
saveRDS(diffmat_10_1, file="diffmat_10_1")

diffmat_10_2<-predmat_pop_10_2-testmat_10_2_matrix
saveRDS(diffmat_10_2, file="diffmat_10_2")

diffmat_10_3<-predmat_pop_10_3-testmat_10_3_matrix
saveRDS(diffmat_10_3, file="diffmat_10_3")

diffmat_10_4<-predmat_pop_10_4-testmat_10_4_matrix
saveRDS(diffmat_10_4, file="diffmat_10_4")

diffmat_10_5<-predmat_pop_10_5-testmat_10_5_matrix
saveRDS(diffmat_10_5, file="diffmat_10_5")

diffmat_10_6<-predmat_pop_10_6-testmat_10_6_matrix
saveRDS(diffmat_10_6, file="diffmat_10_6")

diffmat_10_7<-predmat_pop_10_7-testmat_10_7_matrix
saveRDS(diffmat_10_7, file="diffmat_10_7")

diffmat_10_8<-predmat_pop_10_8-testmat_10_8_matrix
saveRDS(diffmat_10_8, file="diffmat_10_8")

diffmat_10_9<-predmat_pop_10_9-testmat_10_9_matrix
saveRDS(diffmat_10_9, file="diffmat_10_9")

diffmat_10_10<-predmat_pop_10_10-testmat_10_10_matrix
saveRDS(diffmat_10_10, file="diffmat_10_10")

# combining the matrices of the differences
diffmat_all<-rbind(
```

```
  diffmat_10_1,
  diffmat_10_2,
  diffmat_10_3,
  diffmat_10_4,
  diffmat_10_5,
  diffmat_10_6,
  diffmat_10_7,
  diffmat_10_8,
  diffmat_10_9,
  diffmat_10_10
)

# saving
saveRDS(diffmat_all, file="diffmat_all")

# cleaning the working space to free up memory
rm(diffmat_10_1, diffmat_10_2, diffmat_10_3, diffmat_10_4, diffmat_10_5)
rm(diffmat_10_6, diffmat_10_7, diffmat_10_8, diffmat_10_9, diffmat_10_10)

# making sure that the dimensions of the matrix
# fit the number of users in the validation set
dim(diffmat_all)
```

```
## [1] 68534  9809
```

```
length(unique(validation$userId))
```

```
## [1] 68534
```

```
### calculating RMSE
# calculating the number of non-empty cells in the test set
number_of_ratings_in_test_set<-sum(!is.na(diffmat_all))
number_of_ratings_in_test_set
```

```
## [1] 999990
```

```
# saving
saveRDS(number_of_ratings_in_test_set, file="number_of_ratings_in_test_set")

# calculating the squared differences
squared_differences_all<-diffmat_all^2

# saving
saveRDS(squared_differences_all, file="squared_differences_all")

# dividing the sum of the squared differences by the number of ratings
rmse<-sqrt(sum(squared_differences_all, na.rm=T)/number_of_ratings_in_test_set)
rmse
```

```
## [1] 0.85875
```

```
# saving
saveRDS(rmse, file="rmse")
```

The RMSE above is based on my own way of calculation. In order to make sure that the calculation is correct, in the following section I calculate the RMSE using the existing RMSE function.

```
### validating by calculating RMSE in an alternative way ###

# removing all files from the working space, to free up memory
rm(list=ls())

# clearing memory
invisible(gc())

# Increasing memory size
memory.limit(size = 10^10)
```

```
## [1] 1e+10
```

```
# loading the predmap files
predmat_pop_10_1<-readRDS("predmat_pop_10_1")
predmat_pop_10_2<-readRDS("predmat_pop_10_2")
predmat_pop_10_3<-readRDS("predmat_pop_10_3")
predmat_pop_10_4<-readRDS("predmat_pop_10_4")
predmat_pop_10_5<-readRDS("predmat_pop_10_5")
predmat_pop_10_6<-readRDS("predmat_pop_10_6")
predmat_pop_10_7<-readRDS("predmat_pop_10_7")
predmat_pop_10_8<-readRDS("predmat_pop_10_8")
predmat_pop_10_9<-readRDS("predmat_pop_10_9")
predmat_pop_10_10<-readRDS("predmat_pop_10_10")

# creating one unified matrix of predictions
predmat_pop_all<-rbind(
  predmat_pop_10_1,
  predmat_pop_10_2,
  predmat_pop_10_3,
  predmat_pop_10_4,
  predmat_pop_10_5,
  predmat_pop_10_6,
  predmat_pop_10_7,
  predmat_pop_10_8,
  predmat_pop_10_9,
  predmat_pop_10_10
)

# removing the predmap files to free up memory
rm(predmat_pop_10_1, predmat_pop_10_2, predmat_pop_10_3, predmat_pop_10_4, predmat_pop_10_5)
rm(predmat_pop_10_6, predmat_pop_10_7, predmat_pop_10_8, predmat_pop_10_9, predmat_pop_10_10)

# clearing memory
invisible(gc())

# saving
```

```r
saveRDS(predmat_pop_all, file="predmat_pop_all")

# loading the validation set
validation<-readRDS("validation")

# making sure that all users in the validation set are included
dim(predmat_pop_all)
```

```
## [1] 68534  9809
```

```r
length(unique(validation$userId))
```

```
## [1] 68534
```

```r
# clearing the working space to free up memory
rm(list=ls())

# clearing memory
invisible(gc())

# loading the tents of the validation set in matrix form
testmat_10_1_matrix<-readRDS("testmat_10_1_matrix")
testmat_10_2_matrix<-readRDS("testmat_10_2_matrix")
testmat_10_3_matrix<-readRDS("testmat_10_3_matrix")
testmat_10_4_matrix<-readRDS("testmat_10_4_matrix")
testmat_10_5_matrix<-readRDS("testmat_10_5_matrix")
testmat_10_6_matrix<-readRDS("testmat_10_6_matrix")
testmat_10_7_matrix<-readRDS("testmat_10_7_matrix")
testmat_10_8_matrix<-readRDS("testmat_10_8_matrix")
testmat_10_9_matrix<-readRDS("testmat_10_9_matrix")
testmat_10_10_matrix<-readRDS("testmat_10_10_matrix")

# creating one unified matrix of real ratings
testmat_all<-rbind(
  testmat_10_1_matrix,
  testmat_10_2_matrix,
  testmat_10_3_matrix,
  testmat_10_4_matrix,
  testmat_10_5_matrix,
  testmat_10_6_matrix,
  testmat_10_7_matrix,
  testmat_10_8_matrix,
  testmat_10_9_matrix,
  testmat_10_10_matrix
)

# cleaning the working space to free up memory
rm(testmat_10_1_matrix, testmat_10_2_matrix, testmat_10_3_matrix, testmat_10_4_matrix, testmat_10_5_mat
rm(testmat_10_6_matrix, testmat_10_7_matrix, testmat_10_8_matrix, testmat_10_9_matrix, testmat_10_10_ma

# cleaning memory
invisible(gc())
```

```r
# saving
saveRDS(testmat_all, file="testmat_all")

# loading the validation set
validation<-readRDS("validation")

# making sure that all users in the validation set are included
dim(testmat_all)
```

```
## [1] 68534  9809
```

```r
length(unique(validation$userId))
```

```
## [1] 68534
```

```r
# removing the validation set to free up memory
rm(validation)

# cleaning memory
invisible(gc())

# loading the prediction file
predmat_pop_all<-readRDS("predmat_pop_all")

# verifying that all observations are included
dim(predmat_pop_all)
```

```
## [1] 68534  9809
```

```r
dim(testmat_all)
```

```
## [1] 68534  9809
```

```r
# calculating RMSE through the built-in function
rmse_alternative<-RMSE(testmat_all, predmat_pop_all, na.rm=T)

# loading the original RMSE
rmse<-readRDS("rmse")

# comparing the two RMSEs
rmse
```

```
## [1] 0.85875
```

```r
rmse_alternative
```

```
## [1] 0.85875
```

```
difference<-rmse-rmse_alternative
difference
```

```
## [1] 0
```

There is no difference between the two methods of calculating the RMSE. This validates the calculation.

5. Conclusion

I learned a lot from this exercise, most importantly how to work with the Recommender package in R. My main conclusion is that it is important to be aware of the trade-off between performance and speed. Although the SVDF model produced the lowest RMSE, due to my limited processing power it was impractical. The 'Popular' model was much faster and produced a reasonable RMSE, so this seemed like a better choice. The 'fixed effects' model that we learned in the machine learning course performed almost as well as the Popular model in terms of RMSE, and was faster, so that could also be a viable option.