Movielens Ratings Prediction

Code

1. Dataset description and project goals

This data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike other MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided. The data are contained in three files, movies.dat, ratings.dat and tags.dat, of which we'll use only the former two. This and other GroupLens data sets are publicly available for download at GroupLens Data Sets.

The goal of the project is to generate ratings prediction with the minimum RMSE and maximum accuracy. Spercifically, the task requires RMSE to be lower (i.e. better) than 0.87750 in order to obtain maximum points.

2. Ingesting and exploring the data

First, let's download relevant libraries:

Hide

```
suppressMessages(if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(tm)) install.packages("tm", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(slam)) install.packages("slam", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(igraph)) install.packages("igraph", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(Matrix)) install.packages("Matrix", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(reshape2)) install.packages("reshape2", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(SparseM)) install.packages("SparseM", repos = "http://cran.us.r-project.org"))
suppressMessages(if(!require(xgboost)) install.packages("xgboost", repos = "http://cran.us.r-project.org"))
```

If the data are not on your hard drive, use the following to download it and create a movielens dataframe for future analysis, including adjusting some of the feature types (commented out since I have the data on my hard drive; uncomment if you need to run):

Hide

```
# dl <- tempfile()
# download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
# ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/rat
ings.dat"))),
#                       col.names = c("userId", "movieId", "rating", "timestamp"))
# movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::",
3)
# colnames(movies) <- c("movieId", "title", "genres")
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[
movieId],
#                                            title = as.character(title),
#                                            genres = as.character(genres))
# movielens <- left_join(ratings, movies, by = "movieId")
```

If the data is on your hard drive, replace the above cell to this one and use your computer’s path:

Hide

```
ratings <- read.table(text = gsub("::", "\t", readLines("ml-10M100K/ratings.dat")),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines("ml-10M100K/movies.dat"), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[mo
vieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

Let’s set the validation portion to be 10% of movielens data:

Hide

```
set.seed(46)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list =
FALSE)
train <- movielens[-test_index,]
temp <- movielens[test_index,]
```

Make sure userId and movieId in validation set are also in train set, and add rows removed from validation set back into train set:

Hide

```
validation <- temp %>%
     semi_join(train, by = "movieId") %>%
     semi_join(train, by = "userId")
removed <- anti_join(temp, validation)
```

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

Hide

```
train <- rbind(train, removed)
```

Let's check the number of genres:

Hide

```
length(unique(movielens$genres))
```

```
[1] 797
```

Hide

```
head(unique(movielens$genres))
```

```
[1] "Comedy|Romance"                "Action|Crime|Thriller"         "Comedy"
"Action|Drama|Sci-Fi|Thriller"
[5] "Action|Adventure|Sci-Fi"       "Action|Adventure|Drama|Sci-Fi"
```

Multiple genres are joined together creating a list of combined "genres", which does not make sense in identifying distinct genres, so need to distill:

Hide

```
corp <- VCorpus(VectorSource(movies$genres))
dtm <- DocumentTermMatrix(corp,
                          control = list(tokenize = function(x)
                            unlist(strsplit(as.character(x), "\\|"))))
dtm$dimnames$Terms
```

```
 [1] "(no genres listed)" "action"             "adventure"          "animation"
"children"           "comedy"
 [7] "crime"              "documentary"        "drama"              "fantasy"
"film-noir"          "horror"
[13] "imax"               "musical"            "mystery"            "romance"
"sci-fi"             "thriller"
[19] "war"                "western"
```

Need to remove “(no genres listed)” as it is not useful for the model:

Hide

```
corp1 <- tm_map(corp, removeWords, "(no genres listed)")
dtm1 <- DocumentTermMatrix(corp1,
                          control = list(tokenize = function(x)
                            unlist(strsplit(as.character(x), "\\|"))))
dtm1$dimnames$Terms
```

```
 [1] "action"      "adventure"   "animation"   "children"    "comedy"      "crime"
"documentary" "drama"       "fantasy"
[10] "film-noir"   "horror"      "imax"        "musical"     "mystery"     "romance
"     "sci-fi"      "thriller"    "war"
[19] "western"
```

We can use these genre names later as additional features if we can not reach the required RMSE.

Let’s understand what genres are considered close to each other, create matrix to be used for adjacency analysis:

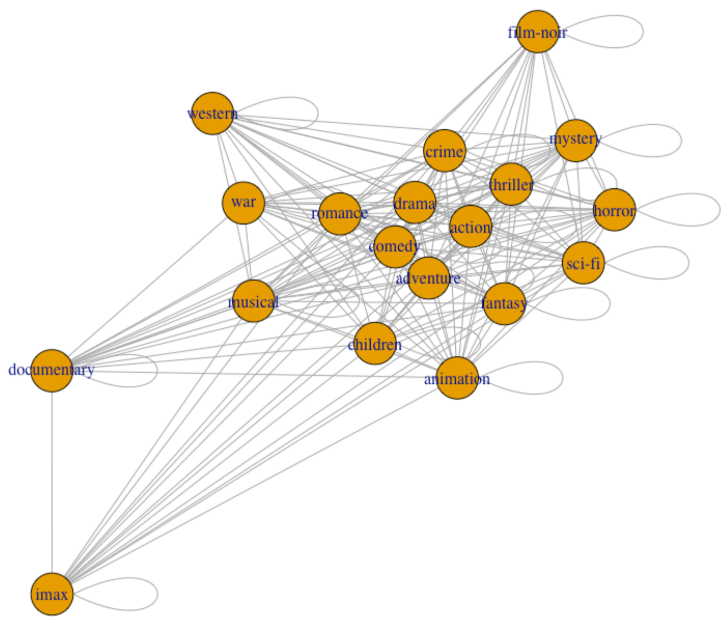
Hide

```
adj <- crossprod_simple_triplet_matrix(dtm1); adj
```

```
             Terms
Terms          action adventure animation children comedy crime documentary drama fa
ntasy film-noir horror imax musical mystery romance sci-fi
  action         1473       452        36       47    314   318           1   483
128         6    108    2       7      49     104    270
  adventure       452      1025       113      220    282    52           6   292
206         3     30    6      35      29     105    194
  animation        36       113       286      187    122     5           2    36
83         2      4    5      56       7      20     47
  children         47       220       187      528    256     8           2    88
144         0      3    3      72      10      27     30
  comedy          314       282       122      256   3703   283          32  1081
209         6    155    1     208      77     847    145
  crime           318        52         5        8    283  1118           4   638
14        74     58    1      11     160      93     24
  documentary       1         6         2        2     32     4         482    27
1         0      7   20      26       1       4      0
  drama           483       292        36       88   1081   638          27  5339
164        92    158    2     146     242    1006    149
  fantasy         128       206        83      144    209    14           1   164
543         4     78    2      38      45      96     81
  film-noir         6         3         2        0      6    74           0    92
4       148      4    0       1      29      11      5
  horror          108        30         4        3    155    58           7   158
78         4   1013    1      10     123      19    203
  imax              2         6         5        3      1     1          20     2
2         0      1   29       3       0       1      1
  musical           7        35        56       72    208    11          26   146
38         1     10    3     436       4     121      9
  mystery          49        29         7       10     77   160           1   242
45        29    123    0       4     509      52     43
  romance         104       105        20       27    847    93           4  1006
96        11     19    1     121      52    1685     33
  sci-fi          270       194        47       30    145    24           0   149
81         5    203    1       9      43      33    754
  thriller        500       150         8        1    167   494           1   768
64        54    384    0       4     277     119    206
  war             131        69         7        3     61    10          22   373
11         0      4    0       6       6      75     12
  western          47        50         2        6     63    11           0    66
4         0      3    0       9       3      27      5
             Terms
Terms          thriller war western
  action            500 131      47
  adventure         150  69      50
  animation           8   7       2
  children            1   3       6
  comedy            167  61      63
```

```
crime             494  10       11
documentary         1  22        0
drama             768 373       66
fantasy            64  11        4
film-noir          54   0        0
horror            384   4        3
imax                0   0        0
musical             4   6        9
mystery           277   6        3
romance           119  75       27
sci-fi            206  12        5
thriller         1706  35        8
war                35 511       16
western             8  16      275
```

We can see that e.g. action is considered relatively close to thriller and adventure. A graph can help us depict the adjacency of different genres better:



We can see how close various genres are to each other which will influence our predictions. E.g. film-noir appears to be category of its own, with the closest (yet quite distant relative to most others) genres being mystery and crime, meanwhile action genre seems to be closely related to thriller, comedy, and adventure.

Also, the data appears to be rather sparse:

Hide

```
UMRmatrix <- sparseMatrix(i = movielens$userId,
                          j = movielens$movieId,
                          x = movielens$rating)
print(UMRmatrix)
```

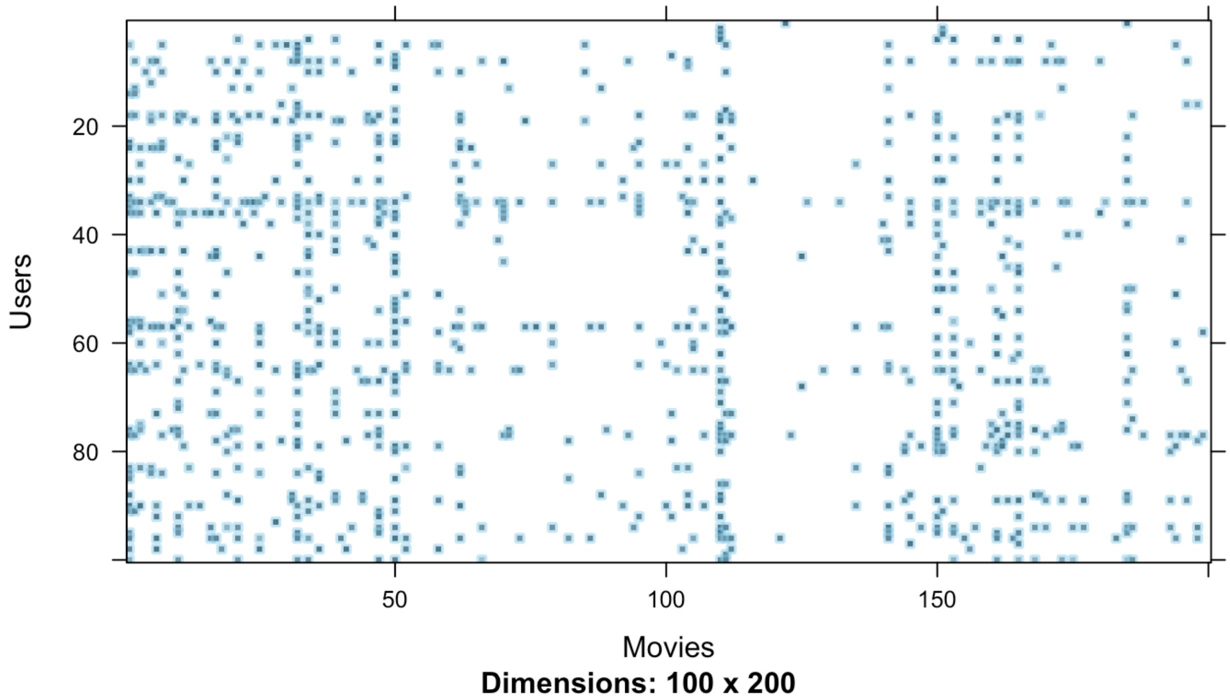
```
71567 x 65133 sparse Matrix of class "dgCMatrix"

[1,] . .   . . . . . . . . . . . . . . . . .   . . .   . . . . . . . . .   . . . .
.   . . .   . . . . . . . . . . . . . . . . . . . . . . . . ......
[2,] . .   . . . . . . . . . . . . . . . . .   . . .   . . . . . . . . .   . . . .
.   . . .   . . . . . . . . . . . . . . . . . . . . . . . . ......
[3,] . .   . . . . . . . . . . . . . . . . .   . . .   . . . . . . . . .   . . . .
.   . . .   . . . . . . . . . . . . . . . . . . . . . . . . ......
[4,] . .   . . . . . . . . . . . . . . . . .   . 3 .   . . . . . . . . .   . . 5 .
.   . . 3.0 . . . . . . . . . . . . . . . . . . . . . . . ......
[5,] 1 .   . . . . 3 . . . . . . . . . . . .   . . .   . . 3 . . 3 . 5 .   5 . . .
.   . . .   . . . . . . . 5 . . . . 4 . . . . 3 3 . . . . . ......
[6,] . .   . . . . . . . . . . . . . . . . .   . . .   . . . . . . . . .   5 . . .
.   . . .   . . . . . . . . . . . . . . . . . . . . . . . . ......
[7,] . .   . . . . . . . . . . . . . . . . .   . . .   . . . . . . . . .   4 . . .
.   . . .   . . . . . . . . . . 4 . . . . . . . . . . . . . ......
[8,] . 2.5 . . 3 4 . . . . . . . . . 3 . . 3.5 . . 2.5 . . . . . . . . 3.5 4 . 3 .
3.5 . . 2.5 . . . . . . . 4 . . 5 . . . . . . . . . . . . . ......

 ..............................
 ........suppressing columns and rows in show(); maybe adjust 'options(max.print= *
, width = *)'
 ..............................


[71561,] . .   .   . . . . . . . . . . . . . . . .   . . . . . . . . . . . . 4 . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
[71562,] . 3.5 .   . . . . . . . . . . . . . . . 2.0 . . . . . . . . . . . . . . .
. . . . 3 . . . . . . . . . . . . . . . . . . . . . . 4 . ......
[71563,] . .   .   . . . . . . . . . . . . . . . .   . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
[71564,] . .   2.5 . . . . . . . . . . . . . . . 1.5 . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . ......
[71565,] . .   .   . . 3 . . . . . . . . . . . . .   . 3 . . . . . . . . . . . . 5
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
[71566,] 5 .   .   . . . . . . . . . . . . . 4 . .   . . 4 . . 4 . . . . . . 4 . 4
. 5 . . . . . . . . . . . . . 4 . . . . . . . 4 . . . 3 . ......
[71567,] . .   .   . . . . . . . . . . . . . . . .   . . . . . . . . . . . . 3 . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
```

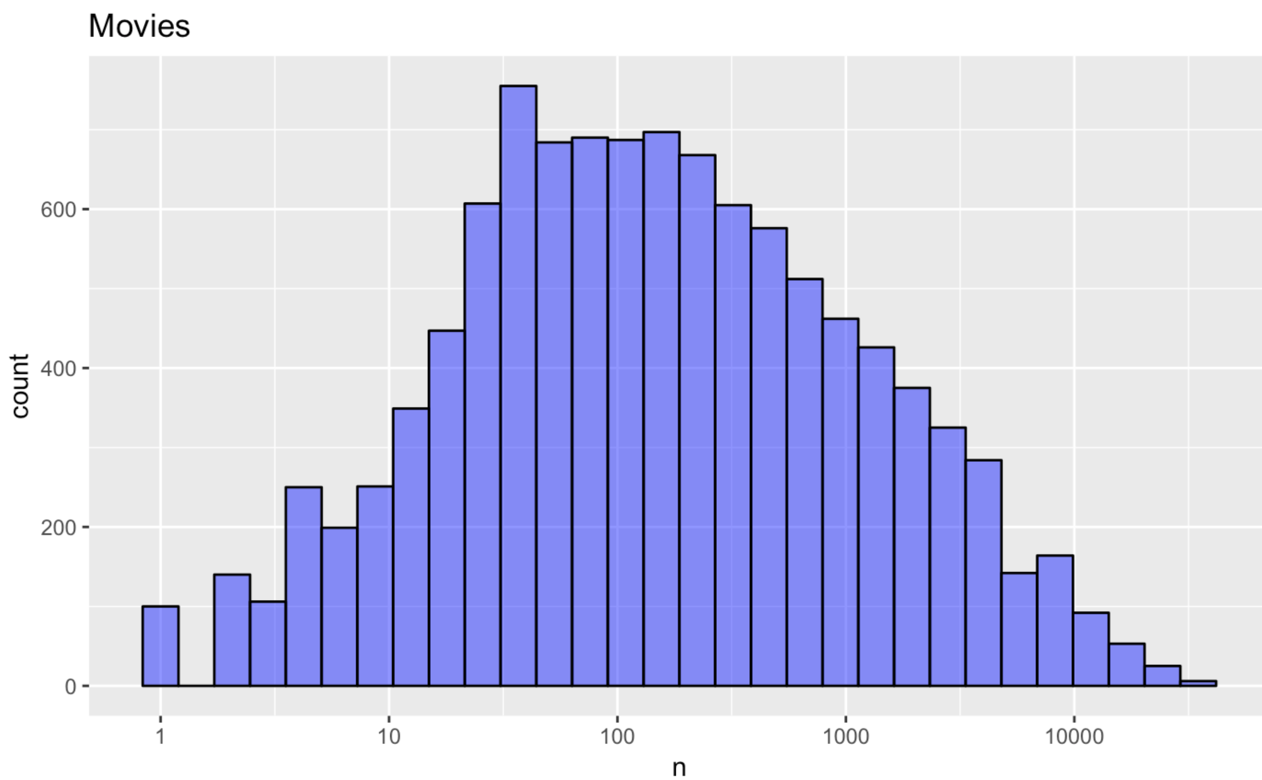
An image of a small sample will help us visualize the sparsity:



What this implies is that some movies get rated more than others…

Hide

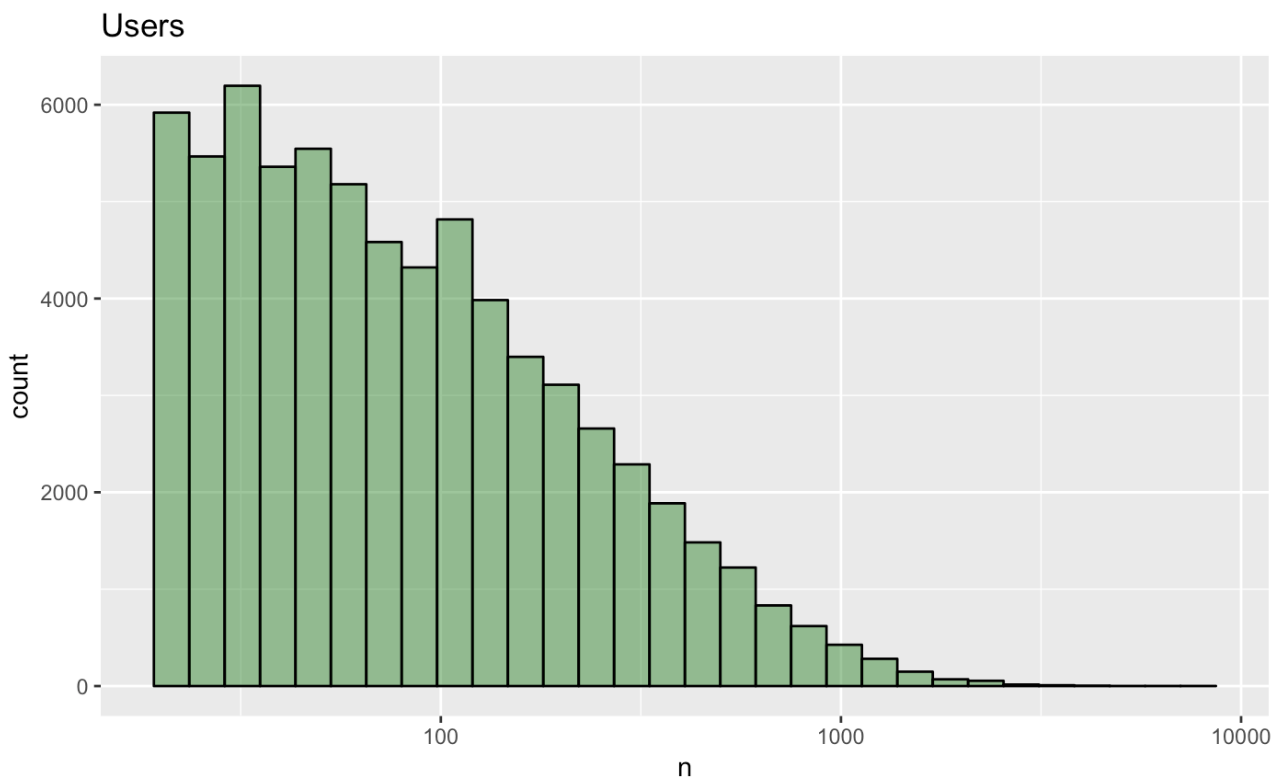
```
movielens %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "blue", alpha = 0.5) +
  scale_x_log10() +
  ggtitle("Movies")
```



…and some users are more active in rating the movies than other:

Hide

```
movielens %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "forestgreen", alpha = 0.5) +
  scale_x_log10() +
  ggtitle("Users")
```

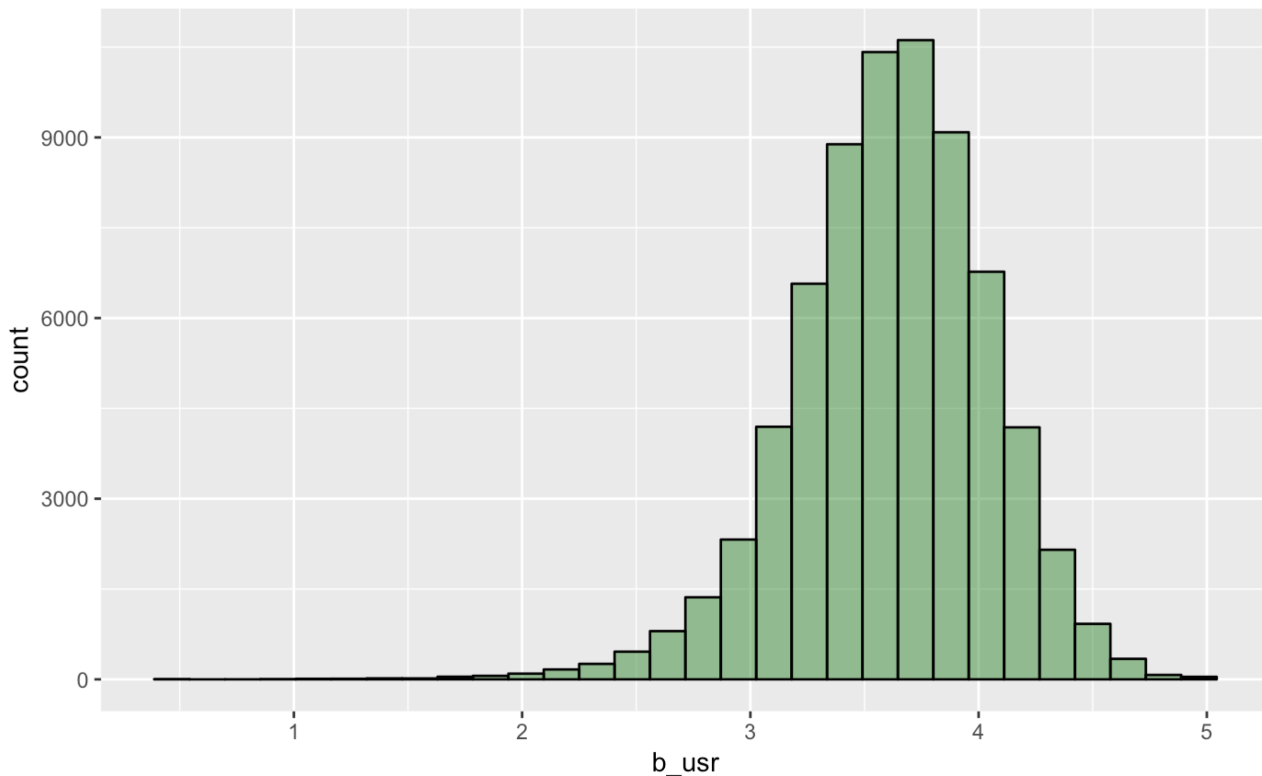


In essence, this is what you'd expect in any ratings database, unless there are tangible incentives to enter ratings (should decrease sparsity) or the ratings are required (should reduce or eliminate sparsity depending on the requirement's enforcement).

Let's also look at how the ratings are distributed (and use "b" following statistics convention for denoting "effect"):

Hide

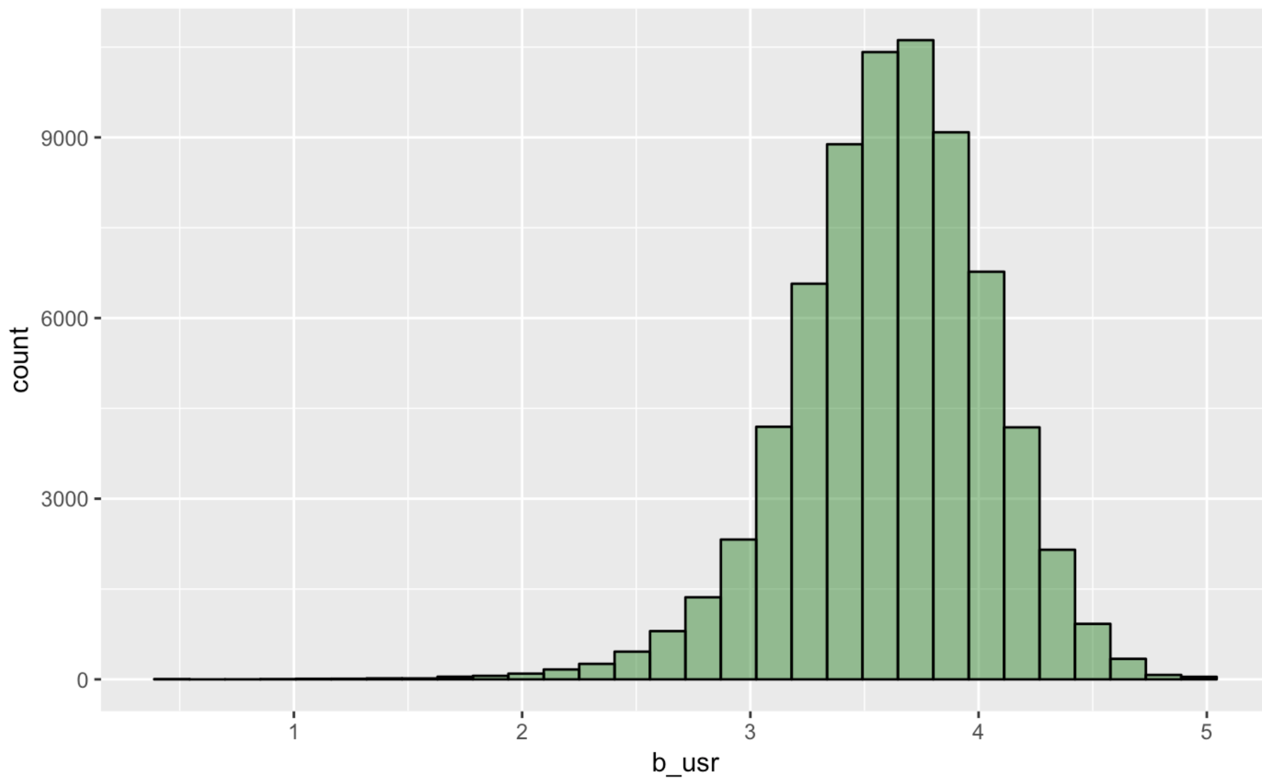
```
train %>%
  group_by(userId) %>%
  summarize(b_usr = mean(rating)) %>%
  ggplot(aes(b_usr)) +
  geom_histogram(bins = 30, color = "black", fill = "forestgreen", alpha = 0.5)
```



How about if we count only those users who rated more than 50 movies?

Hide

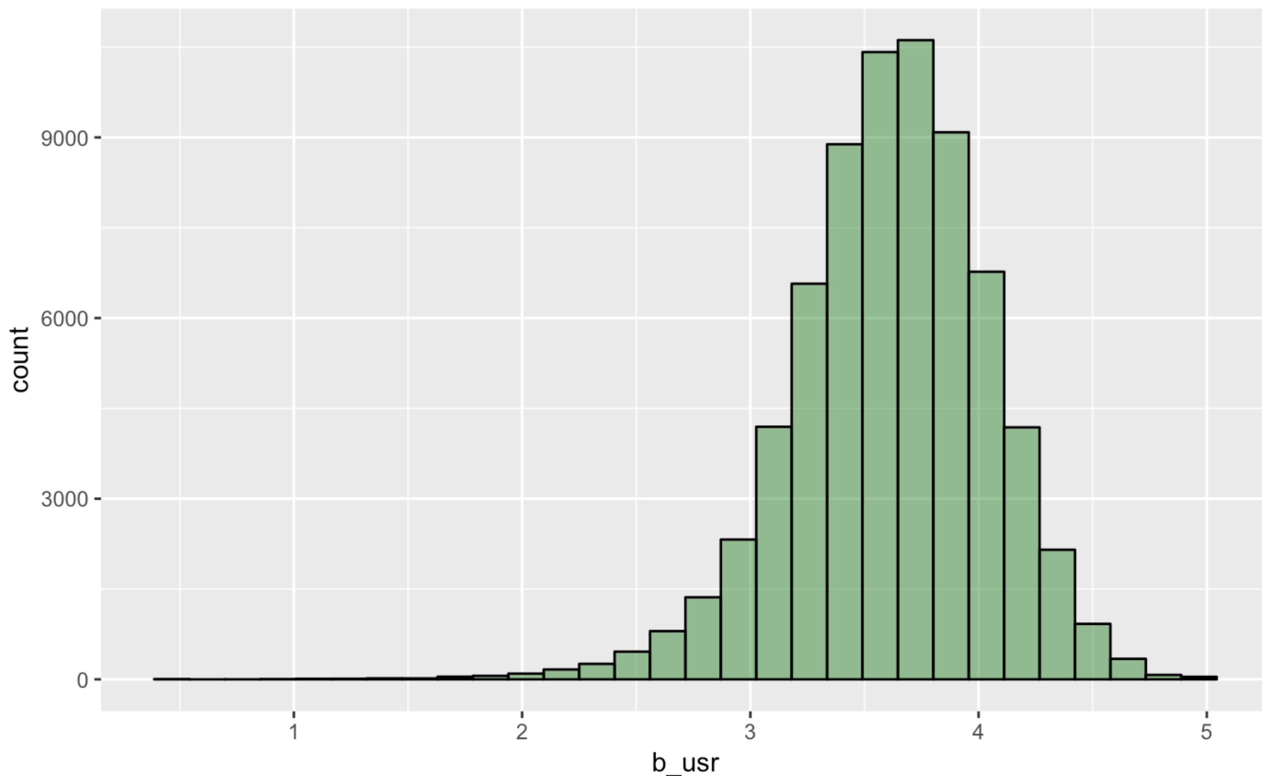
```
train %>%
  group_by(userId) %>%
  summarize(b_usr = mean(rating)) %>%
  filter(n()>=50) %>%
  ggplot(aes(b_usr)) +
  geom_histogram(bins = 30, color = "black", fill = "forestgreen", alpha = 0.5)
```



What about the users who rated more than 100 movies?

Hide

```
train %>%
  group_by(userId) %>%
  summarize(b_usr = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_usr)) +
  geom_histogram(bins = 30, color = "black", fill = "forestgreen", alpha = 0.5)
```



The distributions are largely similar.

3. Analysis and Results

Let’s move into the analysis phase and start by defining RMSE which we’ll use to measure the accuracy of predicted versus actual ratings:

Hide

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

What is the average rating for the training set?

Hide

```
mu <- mean(train$rating); mu
```

```
[1] 3.512452
```

Let’s see what RMSE we would get by using the average rating for predicted ratings in the validation set, and use it as a benchmark for further methods (unless magically we get an RMSE better than the required 0.8775):

Hide

```
naive_rmse <- RMSE(validation$rating, mu)
rmse_results <- data_frame(method = "Simple average", RMSE = naive_rmse); rmse_resu
lts
```

method <chr>	RMSE <dbl>
Simple average	1.060603

1 row

I an unable to run a linear regression with userId and movieId as factors, since my computer's vector memory cannot handle the size. Here, I would like to use Gradient Boosting Machine (GBM), but due to the size of the data and my computer's limited power, I'll try XGB (Extreme Gradient Boosting) which is much faster than GBM:

Hide

```
#XGB requires matrix data, so need to convert the dataframe into matrix before appl
ying the method
set.seed(46)
train_x <- subset(train, select = c(userId, movieId, rating))
train_x <- setDT(train_x)
train_x <- data.table(train_x, keep.rownames = F)
rating <- train_x$rating
new_tr <- model.matrix(~.+0, data = train_x[,-c("rating"), with=F])
train_mx <- xgb.DMatrix(data = new_tr,label = rating)
bst <- xgboost(data = train_mx, label = rating, max.depth = 7, eta = 0.4, nrounds =
40,
               nthread = 2, lambda = 0.7, gamma = 1, objective = "reg:linear")
```

```
xgboost: label will be ignored.
```

```
[14:24:06] Tree method is automatically selected to be 'approx' for faster speed. t
o use old behavior(exact greedy algorithm on single machine), set tree_method to 'e
xact'
[1] train-rmse:2.084944
[2] train-rmse:1.496073
[3] train-rmse:1.213414
[4] train-rmse:1.093425
[5] train-rmse:1.042716
[6] train-rmse:1.023690
[7] train-rmse:1.016785
[8] train-rmse:1.012478
[9] train-rmse:1.008699
[10]    train-rmse:1.007576
[11]    train-rmse:1.006729
[12]    train-rmse:1.005283
[13]    train-rmse:1.004756
[14]    train-rmse:1.003380
[15]    train-rmse:0.999767
[16]    train-rmse:0.998198
[17]    train-rmse:0.997332
[18]    train-rmse:0.996976
[19]    train-rmse:0.996093
[20]    train-rmse:0.995892
[21]    train-rmse:0.995167
[22]    train-rmse:0.995091
[23]    train-rmse:0.994749
[24]    train-rmse:0.993730
[25]    train-rmse:0.992548
[26]    train-rmse:0.991015
[27]    train-rmse:0.990754
[28]    train-rmse:0.990388
[29]    train-rmse:0.989749
[30]    train-rmse:0.988956
[31]    train-rmse:0.987893
[32]    train-rmse:0.987385
[33]    train-rmse:0.987230
[34]    train-rmse:0.986729
[35]    train-rmse:0.986570
[36]    train-rmse:0.986099
[37]    train-rmse:0.984593
[38]    train-rmse:0.983779
[39]    train-rmse:0.982851
[40]    train-rmse:0.982366
```

Let's check RMSE for the predictions:

Hide

```
validation_x <- subset(validation, select = c(userId, movieId, rating))
validation_x <- setDT(validation_x)
validation_x <- data.table(validation_x, keep.rownames = F)
rating <- validation_x$rating
new_val <- model.matrix(~.+0, data = validation_x[,-c("rating"), with=F])
validation_mx <- xgb.DMatrix(data = new_val,label = rating)
xgbpred <- predict(bst,validation_mx)
xgb1_pred <- RMSE(rating, xgbpred)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="XGB 1",
                          RMSE = xgb1_pred))
rmse_results %>% knitr::kable()
```

method	RMSE
Simple average	1.0606032
XGB 1	0.9824392

Improvement in RMSE is significant, but insufficient. Let’s calculate movie and user effects as the next step, and re-apply XGB just using those effects:

Hide

```
mu <- mean(train$rating)
mov_eff <- train %>%
  group_by(movieId) %>%
  summarize(mov_eff = mean(rating - mu))
usr_eff <- train %>%
  left_join(mov_eff, by='movieId') %>%
  group_by(userId) %>%
  summarize(usr_eff = mean(rating - mu - mov_eff))
```

Let’s create a dataframe which will add respective effects to direct movie and user ratings:

Hide

```
train_eff <-
    train %>%
    left_join(mov_eff, by = "movieId") %>%
    left_join(usr_eff, by = "userId")
validation_eff <-
    validation %>%
    left_join(mov_eff, by = "movieId") %>%
    left_join(usr_eff, by = "userId")
```

Select the effect features, convert the resulting dataframe into matrix, and run XGB #2:

Hide

```
set.seed(46)
train_xf <- subset(train_eff, select = c(mov_eff, usr_eff, rating))
train_xf <- setDT(train_xf)
train_xf <- data.table(train_xf, keep.rownames = F)
ratingf <- train_xf$rating
new_trf <- model.matrix(~.+0, data = train_xf[,-c("rating"), with=F])
train_mxf <- xgb.DMatrix(data = new_trf,label = ratingf)
bstf <- xgboost(data = train_mxf, label = ratingf, max.depth = 7, eta = 0.4, nround
s = 40,
               nthread = 2, lambda = 0.7, gamma = 1, objective = "reg:linear")
```

```
xgboost: label will be ignored.
```

```
[14:29:40] Tree method is automatically selected to be 'approx' for faster speed. t
o use old behavior(exact greedy algorithm on single machine), set tree_method to 'e
xact'
[1] train-rmse:2.035350
[2] train-rmse:1.399921
[3] train-rmse:1.083158
[4] train-rmse:0.943233
[5] train-rmse:0.887402
[6] train-rmse:0.866358
[7] train-rmse:0.858588
[8] train-rmse:0.855717
[9] train-rmse:0.854628
[10]    train-rmse:0.854154
[11]    train-rmse:0.853956
[12]    train-rmse:0.853851
[13]    train-rmse:0.853767
[14]    train-rmse:0.853710
[15]    train-rmse:0.853676
[16]    train-rmse:0.853623
[17]    train-rmse:0.853592
[18]    train-rmse:0.853545
[19]    train-rmse:0.853533
[20]    train-rmse:0.853514
[21]    train-rmse:0.853474
[22]    train-rmse:0.853464
[23]    train-rmse:0.853408
[24]    train-rmse:0.853374
[25]    train-rmse:0.853345
[26]    train-rmse:0.853307
[27]    train-rmse:0.853290
[28]    train-rmse:0.853242
[29]    train-rmse:0.853213
[30]    train-rmse:0.853185
[31]    train-rmse:0.853143
[32]    train-rmse:0.853085
[33]    train-rmse:0.853043
[34]    train-rmse:0.853021
[35]    train-rmse:0.853012
[36]    train-rmse:0.853005
[37]    train-rmse:0.852956
[38]    train-rmse:0.852916
[39]    train-rmse:0.852887
[40]    train-rmse:0.852839
```

Let’s check RMSE for the new XGB model:

Hide

```
validation_xf <- subset(validation_eff, select = c(mov_eff, usr_eff, rating))
validation_xf <- setDT(validation_xf)
validation_xf <- data.table(validation_xf, keep.rownames = F)
ratingf <- validation_xf$rating
new_valf <- model.matrix(~.+0, data = validation_xf[,-c("rating"), with=F])
validation_mxf <- xgb.DMatrix(data = new_valf,label = ratingf)
xgbpredf <- predict(bstf,validation_mxf)
xgb2_pred <- RMSE(ratingf, xgbpredf)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="XGB 2",
                          RMSE = xgb2_pred))
rmse_results %>% knitr::kable()
```

method	RMSE
Simple average	1.0606032
XGB 1	0.9824392
XGB 2	0.8609207

4. Conclusion

Using Extreme Gradient Boosting (XGB) algorithm on movie and uaer effect values, we were able to achieve RMSE of 0.86, which is better than the required 0.8775.