Rating Prediction with MovieLens 10M Dataset

Danilo Ferreira de Oliveira

04/12/2021

Contents

1 Introduction

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Introduction to Data Science

For this project, you will be creating a movie recommendation system using the MovieLens dataset. The version of movielens included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. You can find the entire latest MovieLens dataset here. You will be creating your own recommendation system using all the tools we have shown you throughout the courses in this series. We will use the 10M version of the MovieLens dataset to make the computation a little easier.

Second, you will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

You will download the MovieLens data and run code we will provide to generate your datasets.

Recommendation systems are more complicated machine learning challenges because each outcome has a different set of predictors. For example, different users rate a different number of movies and rate different movies.

To compare different models or to see how well we're doing compared to a baseline, we will use root mean squared error (RMSE) as our loss function. We can interpret RMSE similar to standard deviation.

2 Analysis

2.1 Getting the data

```
library(caret)
library(tidyverse)
library(ggplot2)
library(lubridate)
library(stringr)
library(recosystem)
library(data.table)
library(kableExtra)
library(tibble)
```

2.2 Exploratory Data Analysis

2.2.1 Exploring the variables

```
edx %>% as_tibble()
```

```
>> # A tibble: 9,000,055 x 6
>>    userId movieId rating timestamp title                  genres
>>     <int>   <dbl>  <dbl>     <int> <chr>                  <chr>
>>  1      1     122      5 838985046 Boomerang (1992)       Comedy|Romance
>>  2      1     185      5 838983525 Net, The (1995)        Action|Crime|Thriller
>>  3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
>>  4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
>>  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
>>  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
>>  7      1     356      5 838983653 Forrest Gump (1994)    Comedy|Drama|Romance|~
```

```
>>  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
>>  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
>> 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
>> # ... with 9,000,045 more rows
```

We should analyze the quantity of unique values for each column variable.

```
edx %>% summarize(n_userIds = n_distinct(userId), n_movieIds = n_distinct(movieId),
                  n_titles = n_distinct(title), n_genrecomb = n_distinct(genres)) %>%
  kbl(booktabs = TRUE) %>% kable_styling(latex_options = c("striped", "hold_position"))
```

n_userIds	n_movieIds	n_titles	n_genrecomb
69878	10677	10676	797

Note that there is one more distinct `movieId` than `title`. We need to investigate why.

```
edx %>% group_by(title) %>% summarize(n_movieIds = n_distinct(movieId)) %>%
  filter(n_movieIds > 1) %>% kbl(booktabs = TRUE) %>%
  kable_styling(latex_options = c("striped","hold_position"))
```

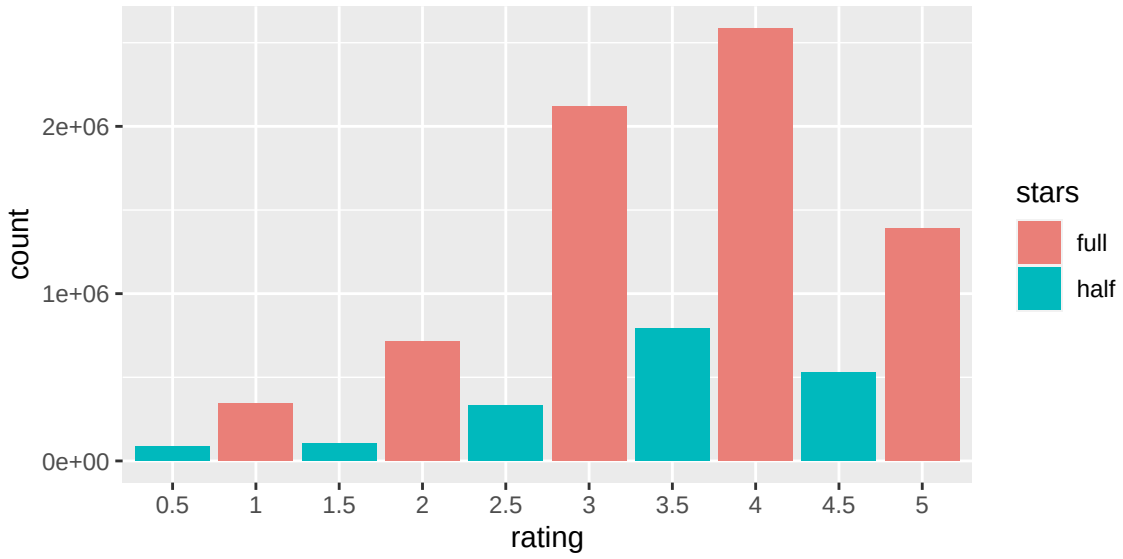
title	n_movieIds
War of the Worlds (2005)	2

So we found out that *War of the Worlds (2005)* has two distinct `movieIds`. We then see how many reviews for each of them there are.

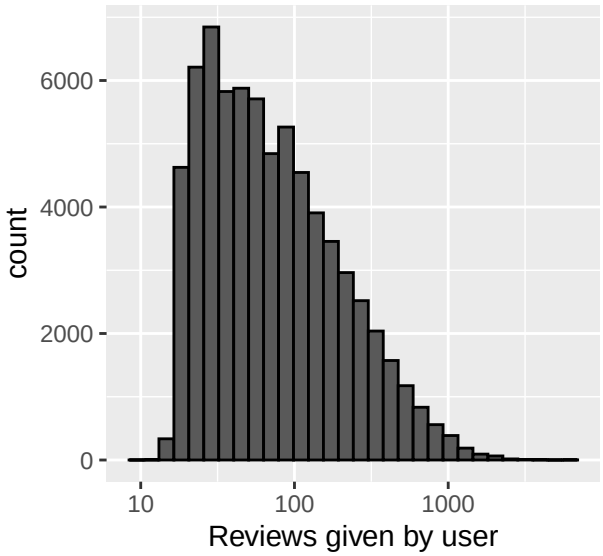
movieId	title	genres	n
34048	War of the Worlds (2005)	Action|Adventure|Sci-Fi|Thriller	2460
64997	War of the Worlds (2005)	Action	28

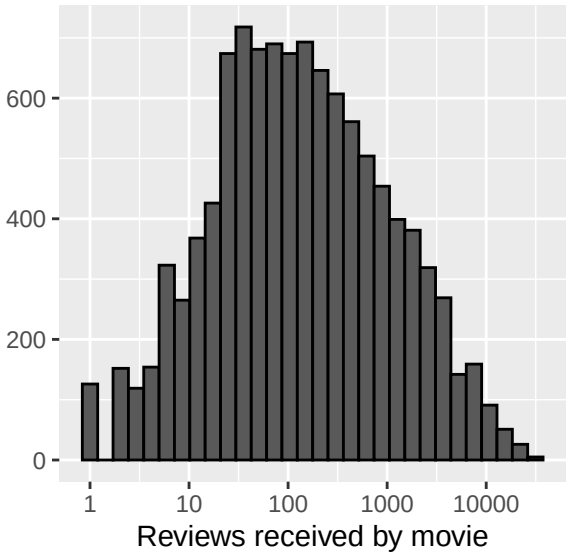
2.2.2 Visualizing the data

We can now visualize the data. Let's see how is configured the ratings distribution.



So we see that full star ratings are more common than those with half stars, since 4, 3 and 5 are the most common ones.

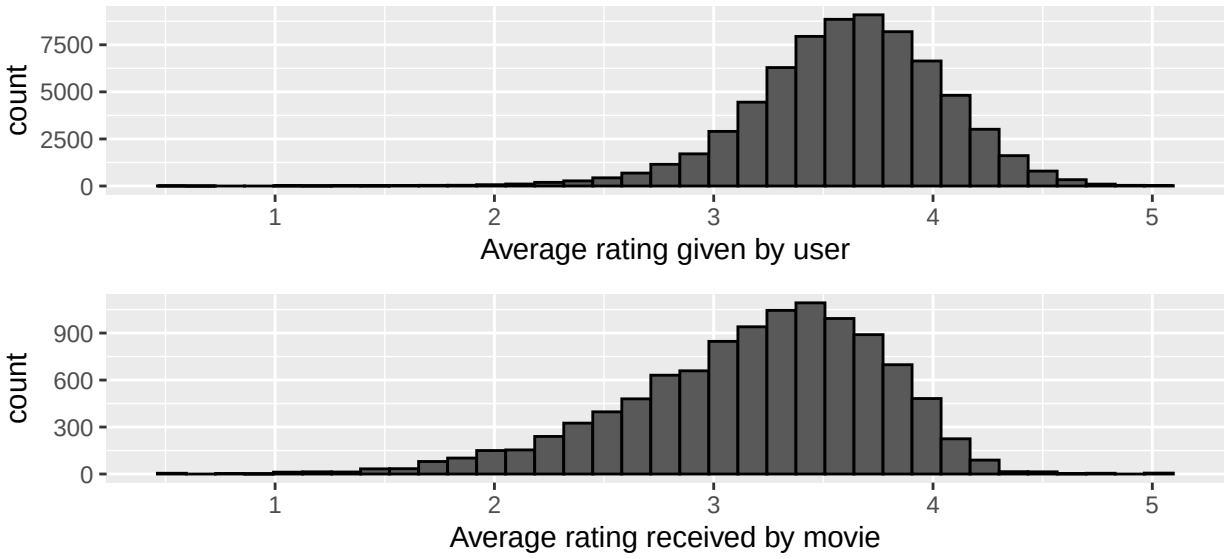





```
>> [1] 49
```

```
>> [1] 61
```

```
>> [1] 57
```

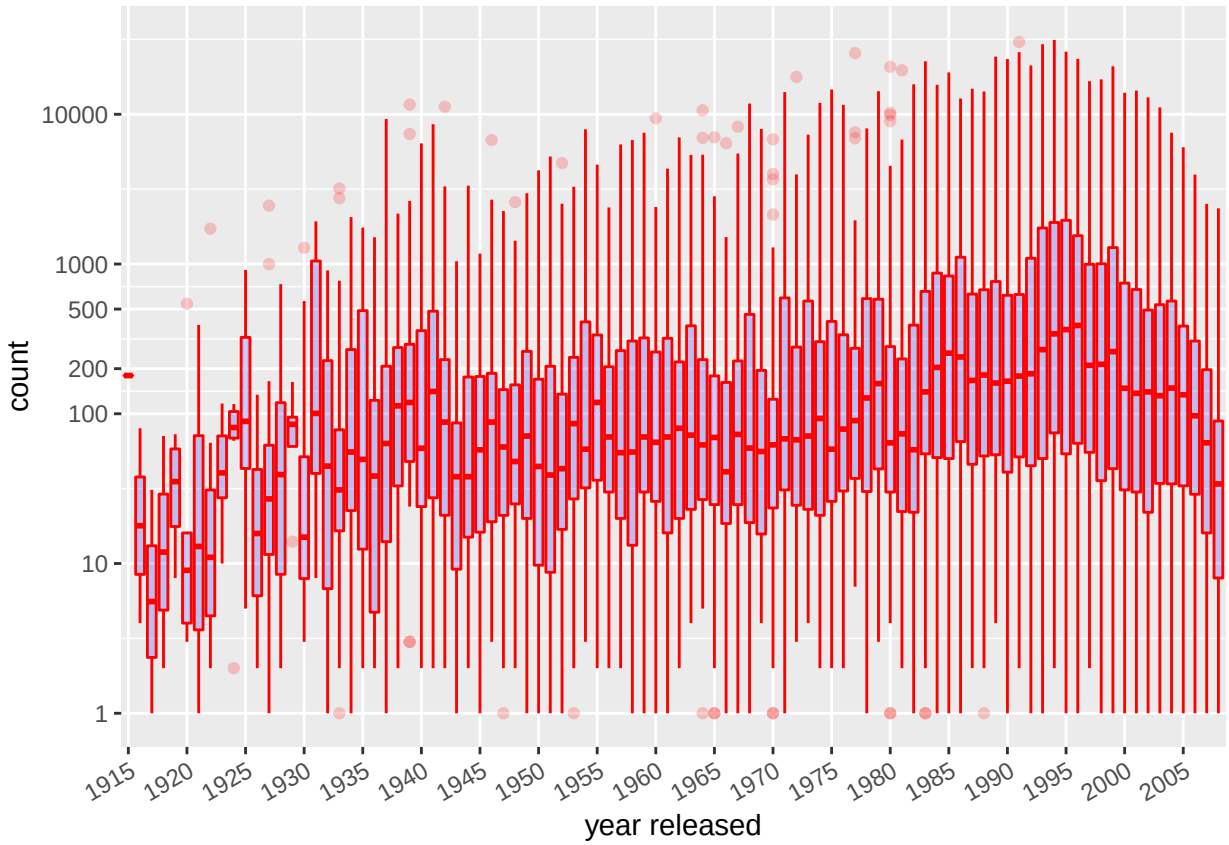


Compute the number of ratings for each movie and then plot it against the year the movie came out using a boxplot for each year. Use the square root transformation on the y-axis (number of ratings) when creating your plot.

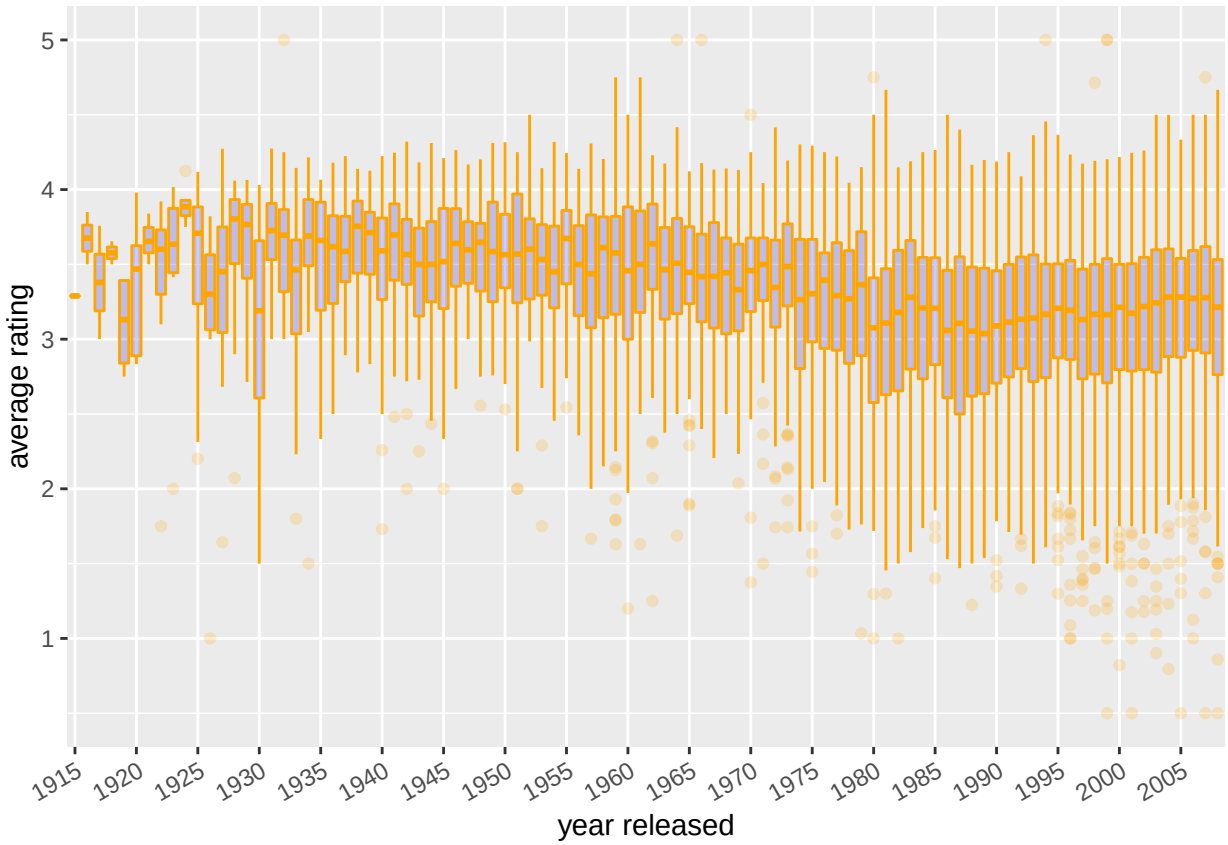
```
release_date <- edx$title %>% str_extract('\\([0-9]{4}\\)') %>%
  str_extract('[0-9]{4}') %>% as.integer()
summary(release_date)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>    1915    1987    1994    1990    1998    2008
```

movieId	title	genres	n
7065	Birth of a Nation, The (1915)	Drama|War	180



We see that, on average, movies that came out after 1993 get more ratings. We also see that with newer movies, starting in 1993, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

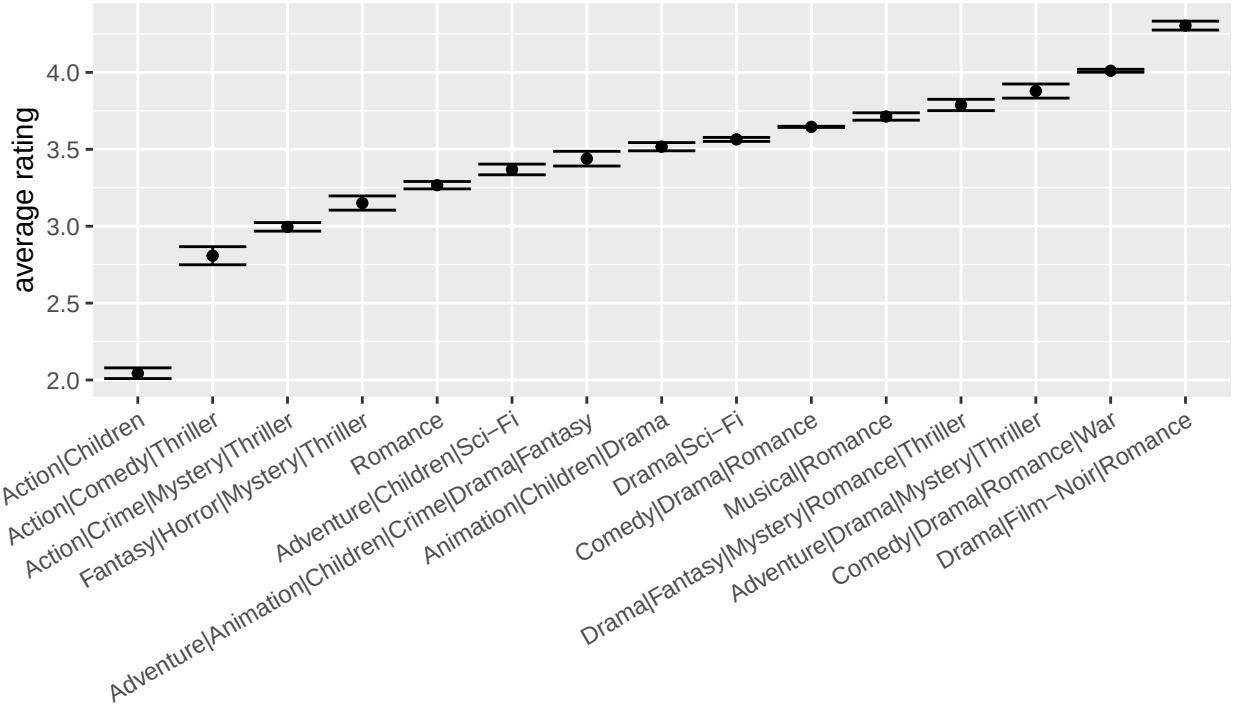


The genres contained in the dataset actually represent the combination of a series of unique genres. We count all the different genre combinations that appear in our dataset, arrange them in descending order for the number of reviews each. The table below show, as an example, the top 7 most reviewed genre combinations.

pos	genres	count
1	Drama	733296
2	Comedy	700889
3	Comedy|Romance	365468
4	Comedy|Drama	323637
5	Comedy|Drama|Romance	261425
6	Drama|Romance	259355
7	Action|Adventure|Sci-Fi	219938

Genre effect on the ratings

```
gcomb_20 <- c(1,seq(31,444,32),444)
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n > 1000) %>% arrange(desc(avg)) %>%  .[gcomb_20,] %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg,
             ymin = avg - 2*se, ymax = avg + 2*se)) + geom_point() +
  geom_errorbar() + theme(axis.text.x=element_text(angle = 30, hjust = 1)) +
  labs(x='', y= 'average rating')
```



There are 20 different genres in this dataset, and they are the following:

Comedy	Drama	War	Film-Noir
Romance	Sci-Fi	Animation	Horror
Action	Adventure	Musical	Documentary
Crime	Children	Western	IMAX
Thriller	Fantasy	Mystery	(no genres listed)

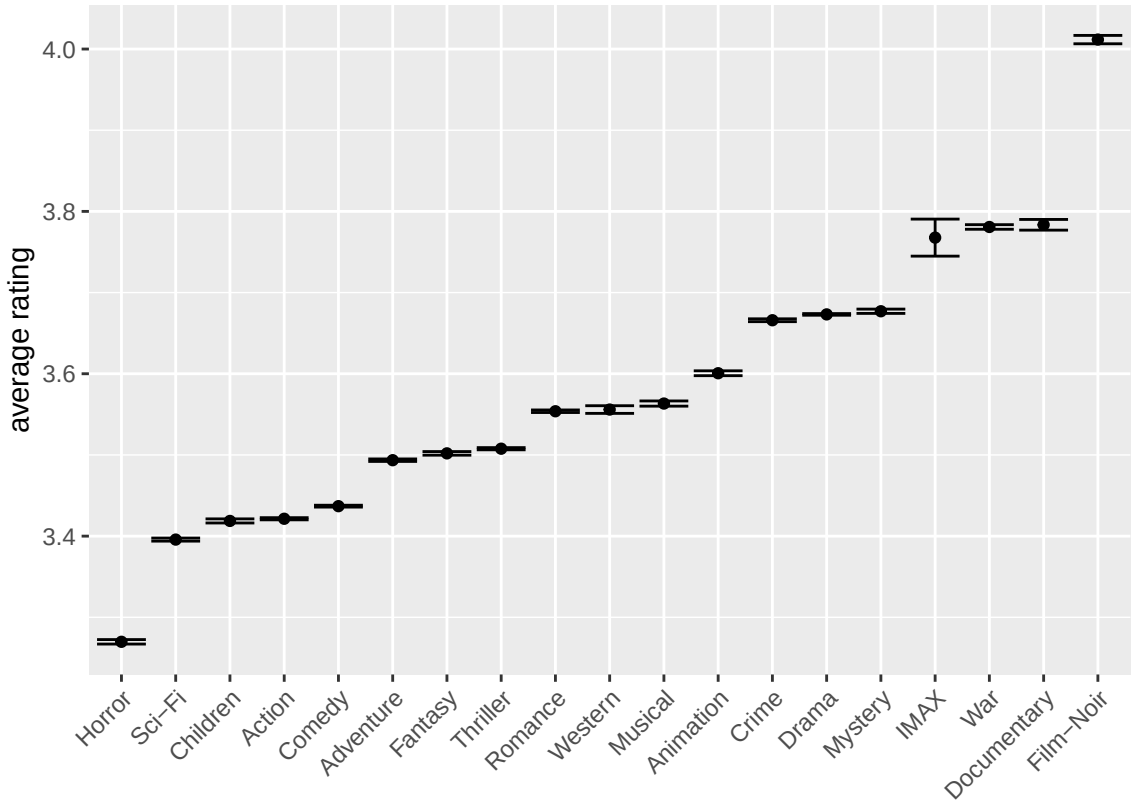
```
all_genres_count <- sapply(all_genres, function(g) {
  sum(str_detect(edx$genres, g))
}) %>% data.frame(review_count=.) %>%
  arrange(desc(.))
```

all_genres_count[1:10]		all_genres_count[11:20]	
Genre	Count	Genre	Count
Drama	3910127	Horror	691485
Comedy	3540930	Mystery	568332
Action	2560545	War	511147
Thriller	2325899	Animation	467168
Adventure	1908892	Musical	433080
Romance	1712100	Western	189394
Sci-Fi	1341183	Film-Noir	118541
Crime	1327715	Documentary	93066
Fantasy	925637	IMAX	8181
Children	737994	(no genres listed)	7

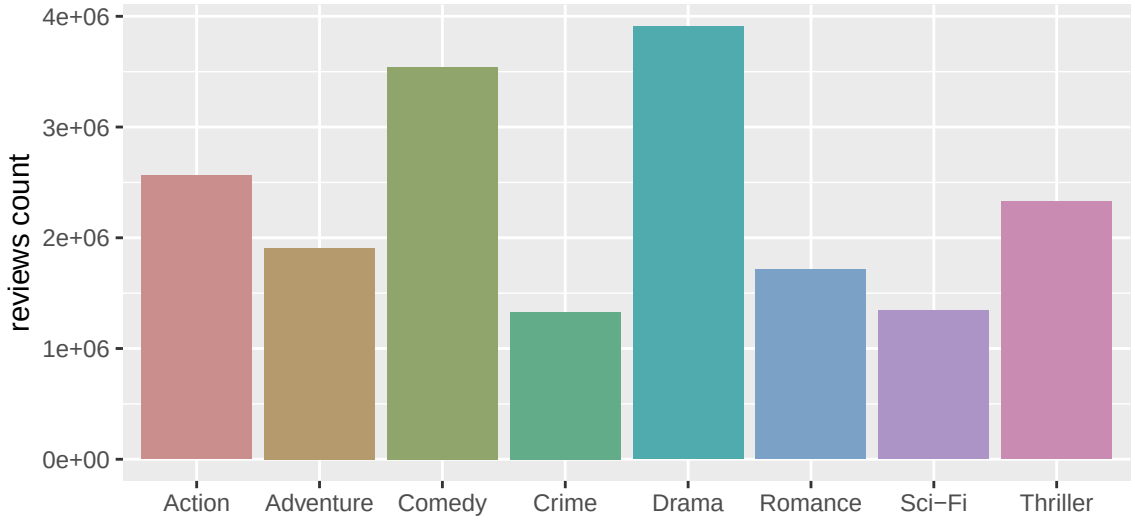
why is there one genre missing in the plot? Here's why

```
edx[which(edx$genres=='(no genres listed)'),] %>%
  summarize(movieId=movieId[1], userId=userId[1], title=title[1], genres=genres[1],
            n = n(), avg = mean(rating), se = sd(rating)/sqrt(n()))
```

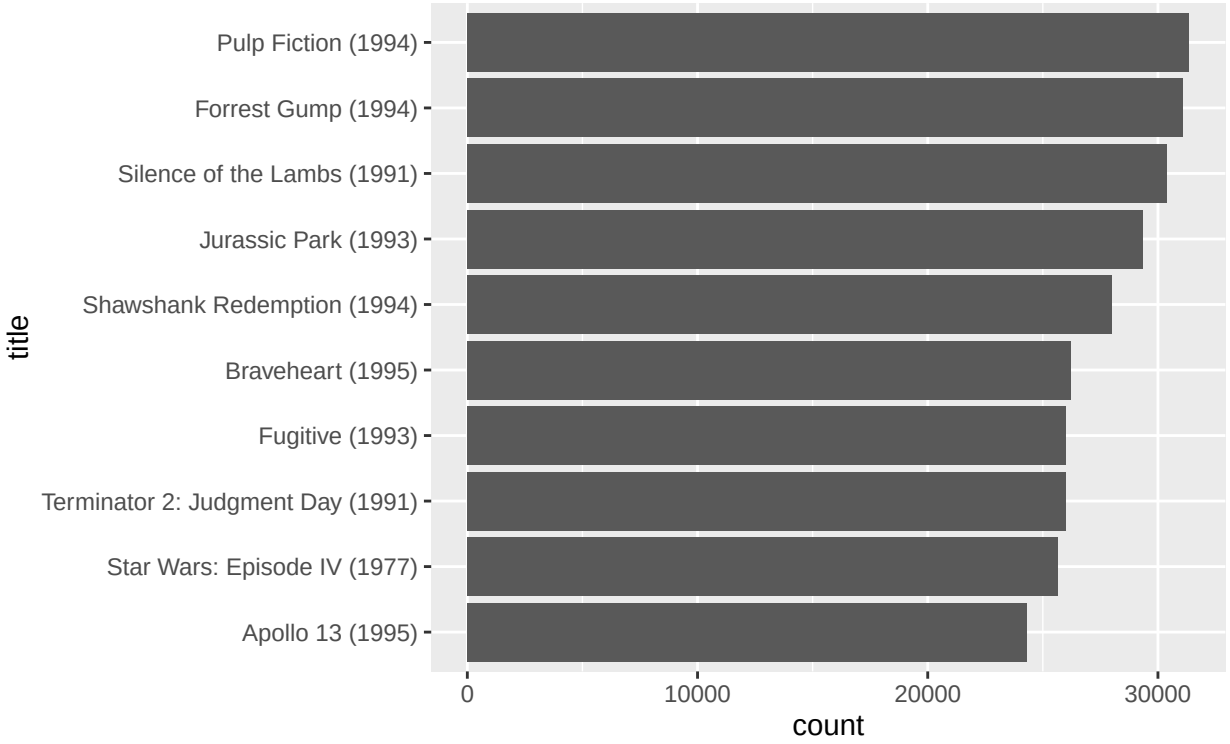
```
>>   movieId userId                title             genres n      avg        se
>> 1    8606   7701 Pull My Daisy (1958) (no genres listed) 7 3.642857 0.4185332
```



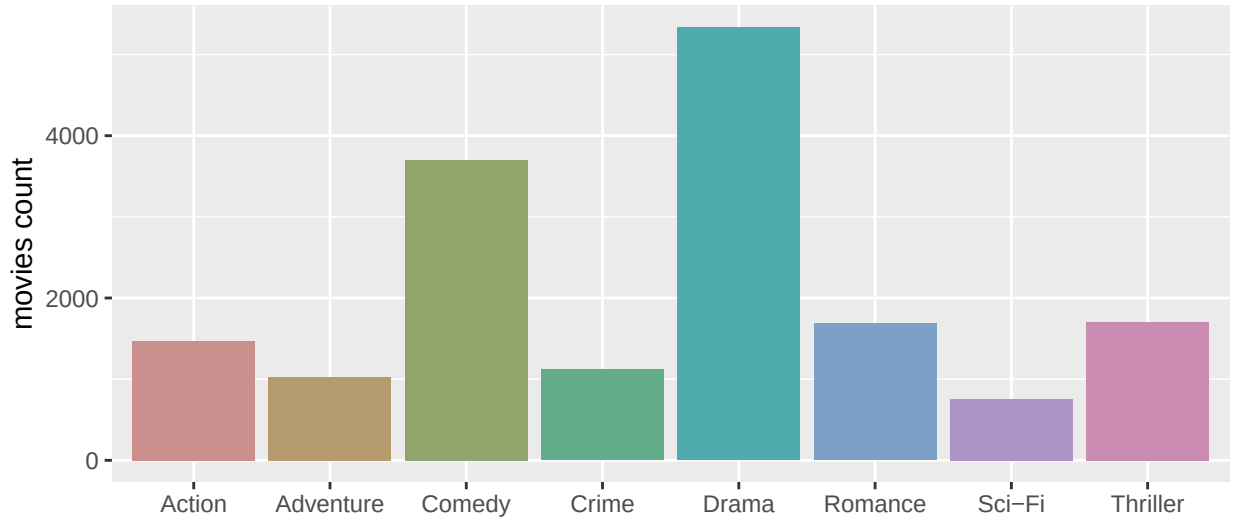
We will then observe the top genres reviewed in this dataset, and how many reviews each one of them have. We accounted for the genres that were reviewed over a million times, resulting in 8 distinct possibilities.



Let's see the unique movies that are in the dataset, how many times they were reviewed, and the number of distinct movies for each of the 8 top genres that were determined previously.



We now plot how many unique movies where reviewed for each of the top 8 genres.



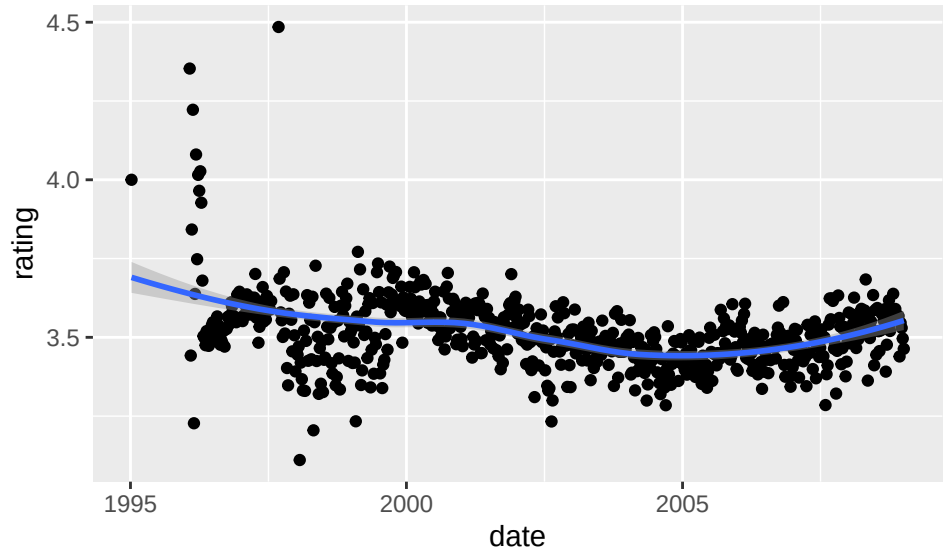
Turning timestamp into date and time

```
>> # A tibble: 6 x 7
>>   userId movieId rating timestamp title         genres       date
>>    <int>   <dbl>  <dbl>     <int> <chr>         <chr>        <dttm>
>> 1      1     122      5 838985046 Boomerang (1~ Comedy|Roma~ 1996-08-02 11:24:06
```

```
>> 2      1     185      5 838983525 Net, The (19~ Action|Crim~ 1996-08-02 10:58:45
>> 3      1     292      5 838983421 Outbreak (19~ Action|Dram~ 1996-08-02 10:57:01
>> 4      1     316      5 838983392 Stargate (19~ Action|Adve~ 1996-08-02 10:56:32
>> 5      1     329      5 838983392 Star Trek: G~ Action|Adve~ 1996-08-02 10:56:32
>> 6      1     355      5 838984474 Flintstones,~ Children|Co~ 1996-08-02 11:14:34
```

Time effect on the dataset

```
edx %>% mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```



2.3 Feature Engineering

```
for (var in all_genres) {
  edx <- edx %>% mutate(genre=ifelse(str_detect(genres,as.character(var)),1,0))
  colnames(edx)[ncol(edx)] <- as.character(var)
}
rm(var)
edx <- edx[,-(4:7),drop=FALSE]
edx[,4:23] %>% as_tibble(.rows=6)
```

```
>> # A tibble: 6 x 20
>>   Comedy Romance Action Crime Thriller Drama `Sci-Fi` Adventure Children Fantasy
>>    <dbl>   <dbl>  <dbl> <dbl>    <dbl> <dbl>    <dbl>     <dbl>    <dbl>   <dbl>
>> 1      1       1      0     0        0     0        0         0        0       0
>> 2      0       0      1     1        1     0        0         0        0       0
>> 3      0       0      1     0        1     1        1         0        0       0
>> 4      0       0      1     0        0     0        1         1        0       0
>> 5      0       0      1     0        0     1        1         1        0       0
```

```
>> 6      1       0      0     0        0     0        0         0        1       1
>> # ... with 10 more variables: War <dbl>, Animation <dbl>, Musical <dbl>,
>> #   Western <dbl>, Mystery <dbl>, Film-Noir <dbl>, Horror <dbl>,
>> #   Documentary <dbl>, IMAX <dbl>, (no genres listed) <dbl>
```

2.4 Creating training and test sets

```
set.seed(123, sample.kind = "Rounding")
index <- createDataPartition(edx$rating, times=1, p=0.2, list= FALSE)
test <- edx[index,]
train <- edx[-index,]
```

```
test <- test %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

If N is the number of user-movie combinations, yu,i is the rating for movie i by user u, and y^u,i is our prediction, then RMSE is defined as follows: RMSE function

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

3 Results

3.1 Prediction models

3.1.1 Naive RMSE

We start with a model that assumes the same rating for all movies and all users, with all the differences explained by random variation: If represents the true rating for all movies and users and represents independent errors sampled from the same distribution centered at zero, then:

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

In this case, the least squares estimate of — the estimate that minimizes the root mean squared error — is the average rating of all movies across all users.

```
mu <- mean(train$rating)
RMSE(test$rating, mu)
```

```
>> [1] 1.060013
```

method	RMSE
Just the Average	1.060013

3.1.2 Movie Effects

We can improve our model by adding a term, bi, that represents the average rating for movie i

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

bi is the average of Yu,i minus the overall mean for each movie i.

Note that because there are thousands of b's, the lm() function will be very slow or cause R to crash, so we don't recommend using linear regression to calculate these effects.

```
movie_avgs <- train %>% group_by(movieId) %>% summarize(b_i = mean(rating-mu))

predicted_ratings <- mu + test %>% left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>   0.500   3.217   3.586   3.512   3.878   4.750

>> [1] 0.9436204
```

method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204

3.1.3 Movie + User Effects

We can further improve our model by adding bu, the user-specific effect:

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

```
user_avgs <- train %>%  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test %>%
  left_join(movie_avgs, by='movieId') %>%  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%  pull(pred)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>> -0.7161  3.1353  3.5651  3.5127  3.9450  6.0547
```

208 e 4149

```
>> [1] 208

>> [1] 4149

>> [1] 0.8662966
```

method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966

3.1.4 Movie + User + Genre Effects

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + \varepsilon_{u,i}$$

with $x_{u,i}^k = 1$ if $g_{u,i}$ is genre k

df_beta [1:10]		df_beta [11:20]	
genre	beta_k	genre	beta_k
Comedy	-0.0022244	War	0.0017305
Romance	-0.0035617	Animation	-0.0153201
Action	-0.0126152	Musical	-0.0102387
Crime	0.0079932	Western	-0.0066510
Thriller	-0.0047380	Mystery	0.0138908
Drama	0.0108468	Film-Noir	0.0306395
Sci-Fi	-0.0119931	Horror	0.0066441
Adventure	-0.0148381	Documentary	0.0624559
Children	-0.0241908	IMAX	-0.0016830
Fantasy	-0.0055682	(no genres listed)	0.1164495

beta equation

$$\sum_{k=1}^{K} x_{u,i}\beta_k = x_{u,i}^{\{1\}}\beta_1 + x_{u,i}^{\{2\}}\beta_2 + ... + x_{u,i}^{\{20\}}\beta_{20}$$

```
sum_x_beta <- as.matrix(test[,4:23])%*%as.matrix(df_beta)
sum_x_beta <- data.frame(sum_x_beta=sum_x_beta[,1])

test <- data.frame(test, sum_x_beta)
rm(df_beta,beta_k,sum_x_beta,all_genres)

predicted_ratings <- test %>%
  mutate(pred = mu + b_i + b_u + sum_x_beta) %>%
  pull(pred)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>> -0.7183  3.1260  3.5581  3.5058  3.9406  6.0652

>> [1] 216

>> [1] 4157

>> [1] 0.8661932
```

method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966
Movie + User + Genre Effects Model	0.8661932

3.1.5 Regularized Movie + User Effects

penalized least squares

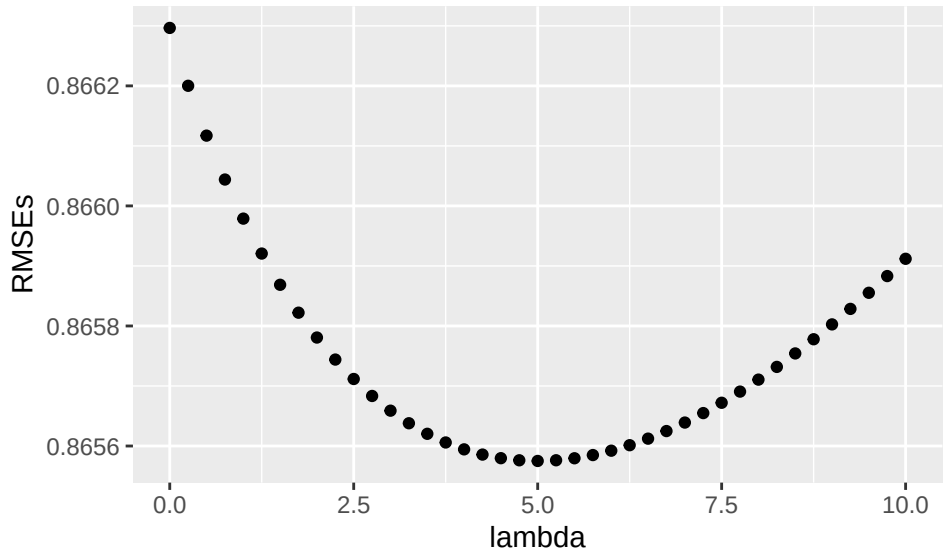
$$\sum_{u,i} \left(y_{u,i} - \mu - b_i - b_u\right)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2\right)$$

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu}\right)$$

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{b}_i - \hat{\mu}\right)$$

```
train <- train %>% select(userId, movieId, rating)
test <- test %>% select(userId, movieId, rating)

lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(test$rating, predicted_ratings))
})
qplot(lambdas, rmses, xlab='lambda', ylab='RMSEs')
```



optimal lambda is 5 and its corresponding RMSE is 0.8655751.

```
>> [1] 5
```

```
>> [1] 0.8655751
```

method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966
Movie + User + Genre Effects Model	0.8661932
Regularized Movie + User Effects Model	0.8655751

3.1.6 Matrix Factorization with *recosystem* package

```
train_dm <- data_memory(user_index = train$userId, item_index = train$movieId,
                        rating = train$rating, index1 = TRUE)
rm(train)

test_dm <- data_memory(user_index = test$userId, item_index = test$movieId,
                       index1 = TRUE)
test_rating <- test$rating
rm(test)

set.seed(123, sample.kind = "Rounding")
r <- Reco()
params = r$tune(train_dm, opts = list(dim = c(15, 20),
                                      costp_l1 = 0, #c(0, 0.1),
                                      costp_l2 = c(0.01, 0.1),
                                      costq_l1 = 0, #c(0, 0.1),
                                      costq_l2 = c(0.01, 0.1),
                                      lrate = c(0.075, 0.1), nthread = 2))
```

```
optimal_params = params$min

r$train(train_dm, opts = c(optimal_params, nthread = 1, niter = 20))
```

with the trained model, it is possible to predict the ratings for the test set.

```
predicted_ratings = r$predict(test_dm, out_memory())

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>  -1.594   3.057   3.543   3.472   3.963   6.300

>> [1] 859

>> [1] 6221

>> [1] 0.7951871
```

method	RMSE
Just the Average	1.0600131
Movie Effect Model	0.9436204
Movie + User Effects Model	0.8662966
Movie + User + Genre Effects Model	0.8661932
Regularized Movie + User Effects Model	0.8655751
Matrix Factorization with SGD Model	0.7951871

3.2 Training with full edx dataset

```
train_dm <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                        rating = edx$rating, index1 = TRUE)
rm(edx)

r$train(train_dm, opts = c(optimal_params, nthread = 1, niter = 20))
```

predicting for validation dataset

```
validation_dm <- data_memory(user_index = validation$userId,
                             item_index = validation$movieId, index1 = TRUE)
validation_rating <- validation$rating

predicted_ratings = r$predict(validation_dm, out_memory())
rm(train_dm,validation)

summary(predicted_ratings)
```

```
>>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
>>  -1.883   3.058   3.545   3.473   3.965   6.159
```

```
>> [1] 433
```

```
>> [1] 3586
```

```
>> [1] 0.7887703
```

method	RMSE
Matrix Factorization with SGD Model	0.7887703

4 Conclusion