HarvardX PH125.9x - Movielens: Movie recommendation system

Stefania_Ravazzini

2021/11/14

1. Executive Summary

The project is based on the data coming from the online movie recommender service **MovieLens**. We will use information about real-world movie ratings: each row represents the rating given to a movie by a specific user. Users were selected at random for inclusion. All users selected had rated at least 20 movies. Each user is represented by an id, and no other information is provided.
[Citation: F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872]

The aim of the project is to build a machine learning algorithm that predicts the movie ratings with the lowest RMSE achievable. RMSE is a metric of how well the model performs: the lowest the RMSE, the smallest the error of the model. RMSE is calculated as the square root of the average through all the observations of the difference squared between actual rating and predicted rating, for each combination of user and movie.

At the very beginning of the analysis, the zip file containing the data is downloaded from the source site. This file contains two separate datasets.
The columns - or "features" - are as follows:
For the `ratings` dataset:
- `userId` : the id of the single user
- `movieId` : the id of the movie
- `rating` : the rating given to the single movie by the specific user
- `timestamp` : time when the review was written

For the `movies` dataset:
- `movieId` : the id of the movie - this key is in common with the `ratings` dataset
- `title` : title of the movie
- `genres` : genre of the movie

In some steps of data preparation, we will merge these two datasets in a single one, by using the `movieId` common key field.

After some data exploration / visualization process, we will have some data cleaning and data improvement section. Finally, the machine learning steps will be performed. The unique dataset will be split into two separate datasets, i.e. the **training set** and the **test set**. After that, various machine learning algorithms are developed and tested during the analysis: the best performing one will be chosen as the final model. According to machine learning standards, the development and training of the algorithms is made on the training set, while the final RMSE is evaluated on the test set.

In the last chapter of the analysis, some future work is suggested.

2. Analysis

2.a Data preparation

The zip file containing the data is downloaded; the two dat files contained in the zip file are extracted and loaded into R:

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.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")

## if using R 3.6 or earlier:
## movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
##                                            title = as.character(title),
##                                            genres = as.character(genres))
## if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
```

We now explore the two datasets `ratings` and `movies`.

```
head(ratings)
```

```
##    userId movieId rating timestamp
## 1:      1     122      5 838985046
## 2:      1     185      5 838983525
## 3:      1     231      5 838983392
## 4:      1     292      5 838983421
## 5:      1     316      5 838983392
## 6:      1     329      5 838983392
```

```
head(movies)
```

```
##   movieId                          title
## 1       1               Toy Story (1995)
## 2       2                 Jumanji (1995)
## 3       3        Grumpier Old Men (1995)
## 4       4       Waiting to Exhale (1995)
## 5       5 Father of the Bride Part II (1995)
## 6       6                    Heat (1995)
##                                         genres
## 1 Adventure|Animation|Children|Comedy|Fantasy
## 2                  Adventure|Children|Fantasy
## 3                              Comedy|Romance
## 4                        Comedy|Drama|Romance
## 5                                      Comedy
## 6                       Action|Crime|Thriller
```

We create a unique dataset, called `movielens`, by using the key field `movieId` which is in common for the two datasets:

```
movielens <- left_join(ratings, movies, by = "movieId")
```

Let's have a quick look at the first rows of the dataset and check info on its structure:

```
head(movielens)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

```
str(movielens)
```

```
## Classes 'data.table' and 'data.frame':   10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
ncol_movielens <- ncol(movielens)
nrow_movielens <- nrow(movielens)
```

The dataset has 10000054 rows and 6 columns.

2.b Data exploration / visualization

Let's have a look at all the features of `movielens` dataset, one by one, except `timestamp` which is not relevant and `title` which is just a text translation of the `movieId` feature.

userId

```
ml_users <- unique(movielens$userId)
length_ml_users <- length(ml_users)
```

`userId` has 69878 unique users.

```
ml_users_detail <- movielens %>%
  mutate(userId = as.factor(userId)) %>%
  group_by(userId) %>%
  summarize(count=n()) %>%
  arrange(desc(count))
```

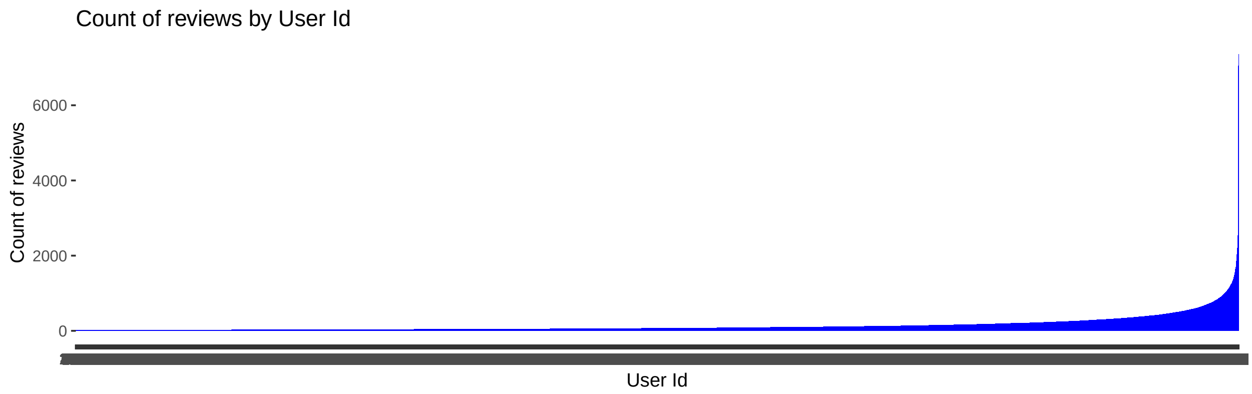
```
kable(ml_users_detail%>%
  top_n(10), caption = "Top 10 User Ids for number of reviews")
```

Table 1: Top 10 User Ids for number of reviews

userId	count
59269	7359
67385	7047
14463	5169
68259	4483
27468	4449
3817	4165
19635	4165
63134	3755
58357	3697
27584	3479

Top 10 users made at least 3400 reviews.

```
ml_users_detail %>%
  mutate(userId = fct_reorder(userId, count)) %>%
  ggplot(aes(x= userId, y = count)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(x = "User Id", y = "Count of reviews") +
  ggtitle("Count of reviews by User Id")
```



The distribution of reviews by user id is really skewed and it is clear that some users are more active than others.

```
kable(ml_users_detail %>%
      summarize(min_review = min(count),
```

```
            max_review = max(count),
            mean_review = mean(count),
            sd_review = sd(count)), caption = "User Id - stats")
```

Table 2: User Id - stats

min_review	max_review	mean_review	sd_review
20	7359	143.1073	216.7126

On average, a single user gave 143 reviews.

movieId

```
ml_movieid <- unique(movielens$movieId)
length_ml_movieid <- length(ml_movieid)
```

`movieId` has 10677 unique movies.

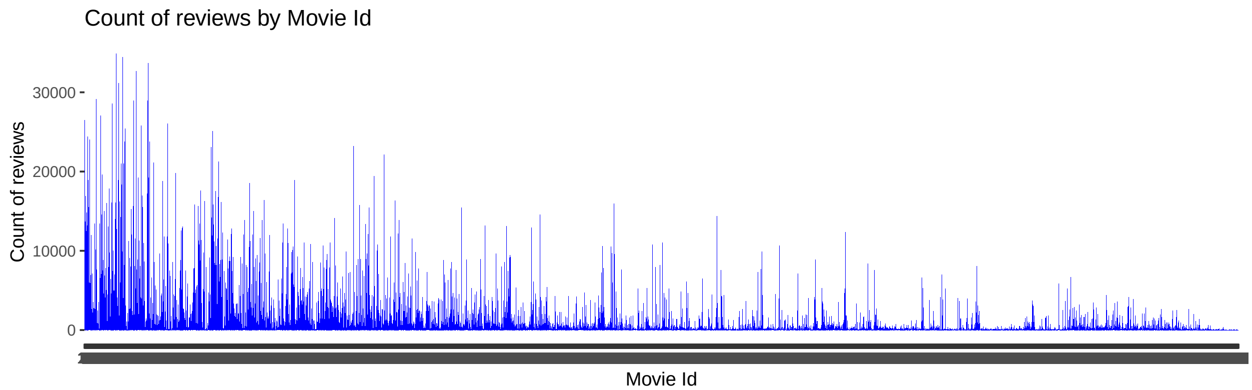
```
ml_movieid_detail <- movielens %>%
  mutate(movieId = as.factor(movieId)) %>%
  group_by(movieId, title) %>%
  summarize(count=n()) %>%
  arrange(desc(count))
```

```
head(ml_movieid_detail, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   movieId [10]
##    movieId title                                                   count
##    <fct>   <chr>                                                   <int>
##  1 296     Pulp Fiction (1994)                                     34864
##  2 356     Forrest Gump (1994)                                     34457
##  3 593     Silence of the Lambs, The (1991)                        33668
##  4 480     Jurassic Park (1993)                                    32631
##  5 318     Shawshank Redemption, The (1994)                        31126
##  6 110     Braveheart (1995)                                       29154
##  7 457     Fugitive, The (1993)                                    28951
##  8 589     Terminator 2: Judgment Day (1991)                       28948
##  9 260     Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 28566
## 10 150     Apollo 13 (1995)                                        27035
```

Top 10 movies are really famous titles and they have at least 27000 reviews each.

```
ml_movieid_detail %>%
  mutate(movieId = fct_reorder(movieId, count)) %>%
  ggplot(aes(x= movieId, y = count)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(x = "Movie Id", y = "Count of reviews") +
  ggtitle("Count of reviews by Movie Id")
```



The distribution of reviews by movie id shows that some movies receive very little reviews while others are more frequently reviewed.

```
ml_movieid_detail_1 <- ml_movieid_detail %>%
  select(movieId, count) %>%
  ungroup()
```

```
kable(ml_movieid_detail_1 %>%
      summarize(min_review = min(count),
            max_review = max(count),
            mean_review = mean(count),
            sd_review = sd(count)), caption = "Movie Id - stats")
```

Table 3: Movie Id - stats

min_review	max_review	mean_review	sd_review
1	34864	936.5977	2487.328

On average, a single movie receives 937 reviews.

genres

```
ml_genres <- unique(movielens$genres)
length_ml_genres <- length(ml_genres)
```

`genres` has 797 unique movie genres.

```
ml_genres_detail <- movielens %>%
  mutate(genres = as.factor(genres)) %>%
  group_by(genres) %>%
  summarize(count=n()) %>%
  arrange(desc(count))
```

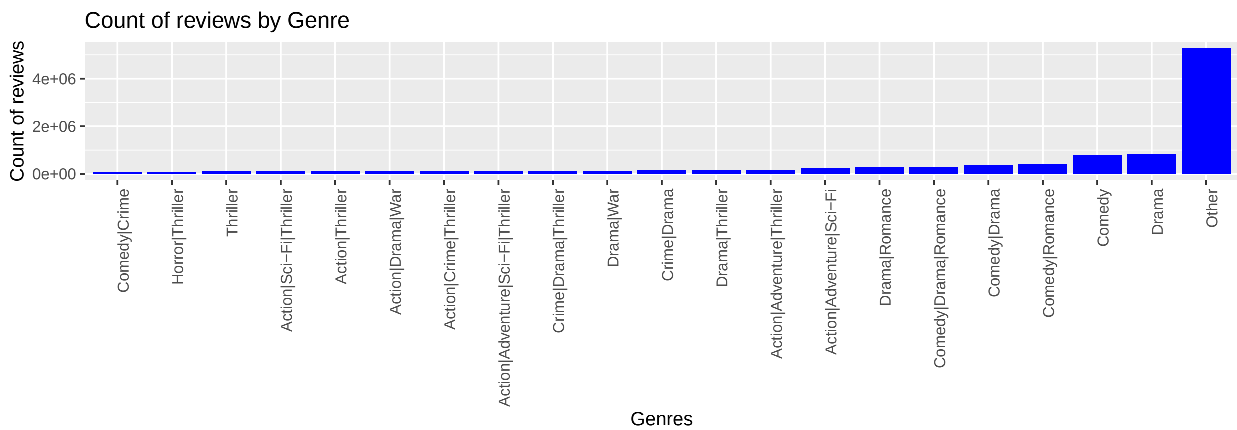
```r
head(ml_genres_detail, 10)
```

```
## # A tibble: 10 x 2
##    genres                     count
##    <fct>                      <int>
##  1 Drama                     815084
##  2 Comedy                    778596
##  3 Comedy|Romance            406061
##  4 Comedy|Drama              359494
##  5 Comedy|Drama|Romance      290231
##  6 Drama|Romance             288539
##  7 Action|Adventure|Sci-Fi   244586
##  8 Action|Adventure|Thriller 165671
##  9 Drama|Thriller            161609
## 10 Crime|Drama               152827
```

Top 10 genres are more or less split between drama, comedy and action.
We group the genres with less than 80K reviews into a single category, that we call `Other`. This is made for data visualization purpose.

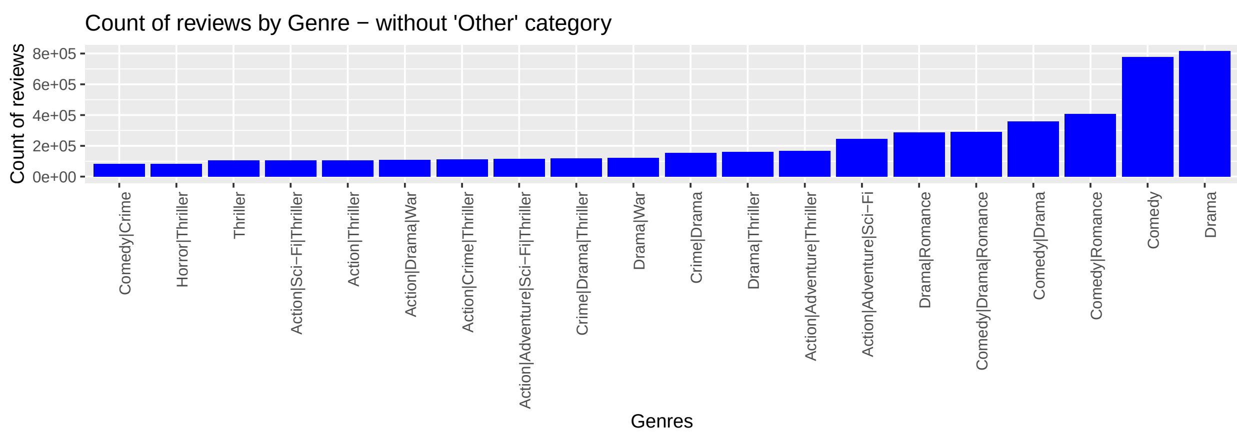
```r
ml_genres_detail_1 <- ml_genres_detail %>%
  mutate(genres = as.character(genres))

ml_genres_detail_1 <- ml_genres_detail_1 %>%
  mutate(genres = case_when(count < 80000 ~ "Other",
                            TRUE ~ genres))

ml_genres_detail_1 <- ml_genres_detail_1 %>%
  group_by(genres) %>%
  summarize(count=sum(count))
```

```r
ml_genres_detail_1 %>%
  mutate(genres = fct_reorder(genres, count)) %>%
  ggplot(aes(x= genres, y = count)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(x = "Genres", y = "Count of reviews") +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  ggtitle("Count of reviews by Genre")
```



The genres plotted here have at least 80000 reviews.

```
ml_genres_detail_1 %>%
  mutate(genres = fct_reorder(genres, count)) %>%
  filter(genres != "Other") %>%
  ggplot(aes(x= genres, y = count)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(x = "Genres", y = "Count of reviews") +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  ggtitle("Count of reviews by Genre - without 'Other' category")
```



This focus on genres with at least 80000 reviews where `Other` category is not displayed helps to see more in detail which genres are most reviewed.

```
kable(ml_genres_detail %>%
      summarize(min_review = min(count),
              max_review = max(count),
              mean_review = mean(count),
              sd_review = sd(count)) , caption = "Genre - stats")
```

Table 4: Genre - stats

min_review	max_review	mean_review	sd_review
2	815084	12547.12	50276.36

On average, a single genre receives 12547 reviews.

rating

```
ml_rating <- unique(movielens$rating)
length_ml_rating <- length(ml_rating)
```

`rating` has 10 unique ratings.

```
ml_rating_detail <- movielens %>%
  mutate(rating = as.factor(rating)) %>%
  group_by(rating) %>%
  summarize(count=n())
```

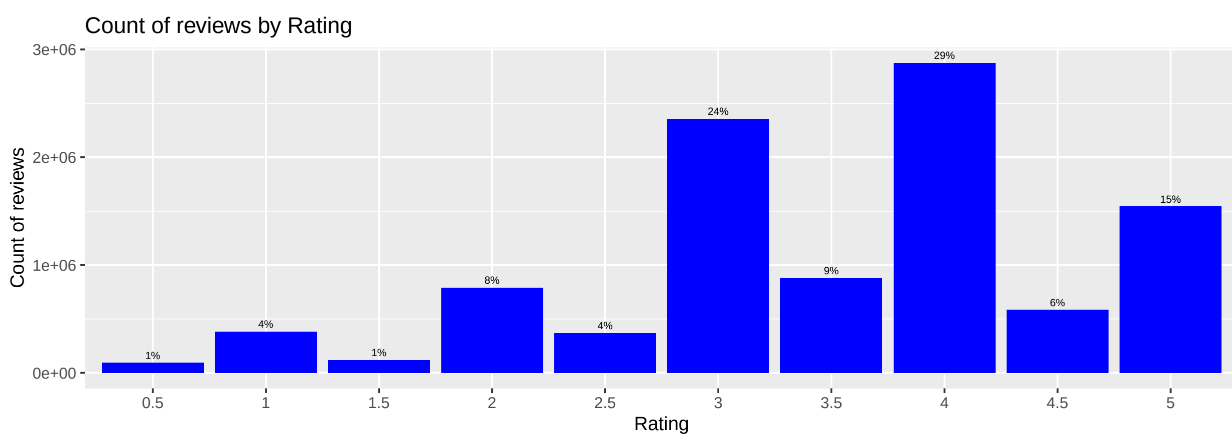
```r
kable(ml_rating_detail, caption = "Ratings")
```

Table 5: Ratings

rating	count
0.5	94988
1	384180
1.5	118278
2	790306
2.5	370178
3	2356676
3.5	879764
4	2875850
4.5	585022
5	1544812

Ratings range from 0.5 to 5; ratings 3 and 4 are the most frequent ones.

```r
ml_rating_detail %>%
  ggplot(aes(x= rating, y = count)) +
  geom_bar(stat = "identity", fill = "blue") +
  geom_text(aes(label=paste0(round(count/nrow_movielens, 2)*100,"%")), position =position_dodge(width =
            vjust = -0.5, size = 2) +
  labs(x = "Rating", y = "Count of reviews") +
  ggtitle("Count of reviews by Rating")
```



More than a half of total ratings are either `3` or `4`; non-integer ratings like `0.5`, `1.5` etc. are not frequently used.

Wrap up: Users that were selected in the dataset have at least 20 reviews each, this means that users who rarely give reviews are not included. At the same time, there's a high variability in this feature because some users are way more active than others. Few movies receive a lot of reviews (see top 10 movies), and genres of drama, comedy and action are the most popular. The reviews range from `0.5` to `5` and the most frequent ratings are `3` and `4`.

2.c Data cleaning and improvement

Let's see if some missing value is detected:

```
sum(is.na(movielens))
```

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

The database doesn't include any missing value. During the data preparation phase, we saw that the `title` column contains the year of release of the movie between brackets. We then build a new column, the `movieYear` column, that we can create from a string-split on `title` column. This will be used as an additional feature in the machine learning section.

```
movielens <- movielens %>%
  mutate(movieYear = str_sub(title, -5, -2))
```

Let's explore the database after this addition:

```
head(movielens)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres movieYear
## 1:                Comedy|Romance      1992
## 2:         Action|Crime|Thriller      1995
## 3:                        Comedy      1994
## 4:  Action|Drama|Sci-Fi|Thriller      1995
## 5:       Action|Adventure|Sci-Fi      1994
## 6: Action|Adventure|Drama|Sci-Fi      1994
```

```
str(movielens)
```

```
## Classes 'data.table' and 'data.frame':   10000054 obs. of  7 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  $ movieYear: chr  "1992" "1995" "1994" "1995" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
sum(is.na(movielens))
```

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

The new column doesn't bring in any NAs.
Let's explore the new column.
movieYear

```r
ml_movieyear <- unique(movielens$movieYear)
length_ml_movieyear <- length(ml_movieyear)
```

We have movies released in 94 different years.

```r
ml_movieyear_detail <- movielens %>%
  mutate(movieYear = as.factor(movieYear)) %>%
  group_by(movieYear) %>%
  summarize(count=n()) %>%
  arrange(desc(count))
```

```r
kable(ml_movieyear_detail%>%
  top_n(10), caption = "Top 10 Movie years for number of reviews")
```

```
## Selecting by count
```

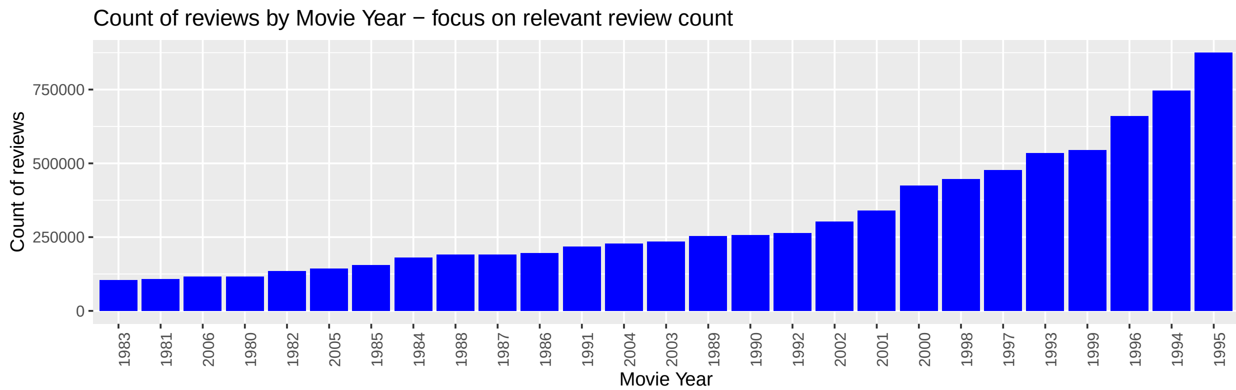
Table 6: Top 10 Movie years for number of reviews

movieYear	count
1995	874436
1994	746042
1996	659425
1999	543990
1993	534899
1997	477463
1998	446739
2000	425218
2001	339508
2002	302452

Top 10 years range from the 90s to 2002 and they have at least 300000 reviews each.
We filter to retain only movie years with relevant count of reviews. This is made for data visualization purpose.

```r
ml_movieyear_detail_1 <- ml_movieyear_detail %>%
  filter(count > 100000)
```

```r
ml_movieyear_detail_1 %>%
  mutate(movieYear = fct_reorder(movieYear, count)) %>%
  ggplot(aes(x= movieYear, y = count)) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(x = "Movie Year", y = "Count of reviews") +
  theme(axis.text.x=element_text(angle=90,hjust=1)) +
  ggtitle("Count of reviews by Movie Year - focus on relevant review count")
```



The focus on movie years having relevant review counts shows the concentration of reviews on specific years.

```
kable(ml_movieyear_detail %>%
  summarize(min_review = min(count),
            max_review = max(count),
            mean_review = mean(count),
            sd_review = sd(count)), caption = "Movie Year - stats")
```

Table 7: Movie Year - stats

min_review	max_review	mean_review	sd_review
36	874436	106383.6	172558.9

On average, on a single movie release-year, we find 100000 reviews.

Wrap up: we added a new column `movieYear` and we'll see if this has an explanatory effect in our machine learning models.

2.d Training and test set

For building our machine learning models, we split the `movielens` dataset into training set called `edx` - this will contain 90% of observations - and test set called `validation` - this will contain 10% of rows.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

We make sure that `userId` and `movieId` in `validation` set are also in `edx` set.

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

We then add rows removed from `validation` set back into `edx` set.

```
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

We perform a data consistency test. We check that the number of rows of training and test set is consistent with split of the source database.

```
nrow_edx <- edx %>%
  summarize(n())
nrow_movielens * 0.9 - nrow_edx
```

```
##    n()
## 1 -6.4
```

```
nrow_validation <- validation %>%
  summarize(n())
nrow_movielens * 0.1 - nrow_validation
```

```
##   n()
## 1 6.4
```

The checks performed are satisfying.

2.e Machine learning models

We build up machine learning models that will predict the rating that a specific user would give to a specific movie. In other words, rating is our dependent variable.
The other features will be used as the independent variables.
`edx` will be our training set, that we use for training the algorithm.
`validation` will be our test set, that we use for testing the "goodness of fit" of our predictions.

First, we declare a RMSE function, which is equal to the square root of the average through all the observations of the difference squared between actual ratings and predicted ratings. This RMSE function will help us evaluate the average error of our predictions.

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

We start from what we saw in **HarvardX PH125.8x** course and then we add some improvements.

`Model 1.0`
We define the following:
Prediction for rating of user u of movie i = Yu,i
Average rating = mu_hat
Random error of prediction Yu,i = Eu,i
`Model 1.0`: Yu,i = mu_hat + Eu,i

```r
mu_hat <- mean(edx$rating)
```

```r
model1_rmse <- RMSE(validation$rating, mu_hat)
```

RMSE of `Model 1.0`: 1.0612018.

`Model 2.0`
We define the following:
Movie effect = bi
`Model 2.0`: Yu,i = mu_hat + bi + Eu,i

```r
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

```r
model2_predictions <- mu_hat + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
```

```r
model2_rmse <- RMSE(validation$rating, model2_predictions)
```

RMSE of `Model 2.0`: 0.9439087.

`Model 3.0`
We define the following:
User effect = bu
`Model 3.0`: Yu,i = mu_hat + bi + bu + Eu,i

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

```r
model3_predictions <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

```r
model3_rmse <- RMSE(validation$rating, model3_predictions)
```

RMSE of `Model 3.0`: 0.8653488.

`Model 4.0`
We define the following:
Genre effect = bg
`Model 4.0`: Yu,i = mu_hat + bi + bu + bg + Eu,i

```r
genre_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))
```

```r
model4_predictions <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  pull(pred)
```

```r
model4_rmse <- RMSE(validation$rating, model4_predictions)
```

RMSE of `Model 4.0`: 0.8649469.

`Model 5.0`
We define the following:
Release year effect = by
`Model 5.0`: Yu,i = mu_hat + bi + bu + bg + by + Eu,i

```r
year_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(movieYear) %>%
  summarize(b_y = mean(rating - mu_hat - b_i - b_u - b_g))
```

```r
model5_predictions <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_avgs, by = 'movieYear') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_y) %>%
  pull(pred)
```

```r
model5_rmse <- RMSE(validation$rating, model5_predictions)
```

RMSE of `Model 5.0`: 0.8647606.

`Tuning parameter session`
We define the following:

Penalty term lambda = l
The penalty term reduces the importance of rare observations and can be applied to all the effects bi, bu, bg, by.
Number of movies = n_i
Number of users = n_u
Number of genres = n_g
Number of release years = n_y
bi(l) = 1 / (lambda + n_i) * sum(on all n_i ratings)(Yu,i - mu_hat)
bu(l) = 1 / (lambda + n_u) * sum(on all n_u ratings)(Yu,i - mu_hat - bi(l))
bg(l) = 1 / (lambda + n_g) * sum(on all n_g ratings)(Yu,i - mu_hat - bi(l) - bu(l))
by(l) = 1 / (lambda + n_y) * sum(on all n_y ratings)(Yu,i - mu_hat - bi(l) - bu(l) - bg(l))
We perform a tuning parameter session, in order to find the penalty term l that minimizes the RMSE.

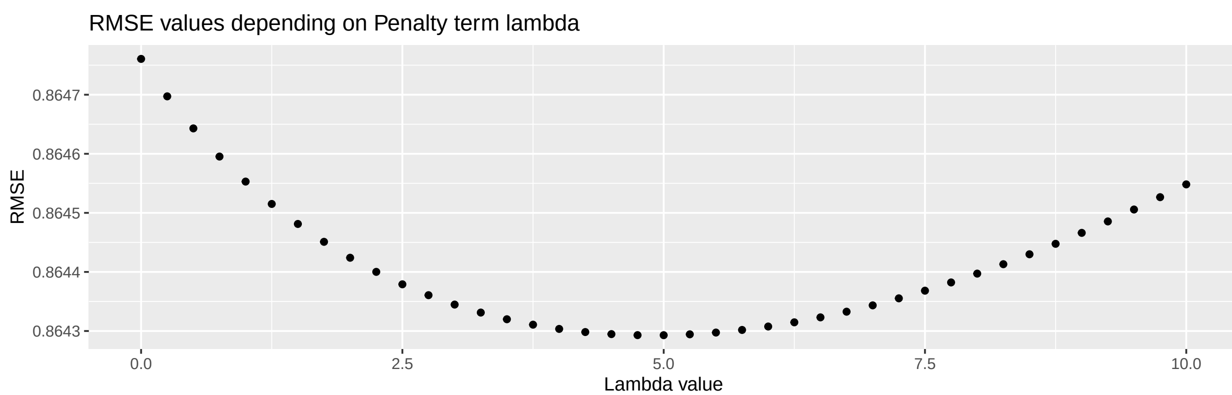
```r
l <- seq(0, 10, 0.25)
```

```r
rmses <- sapply(l, function(lambda){

  mu_hat <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+lambda))

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda))

  b_g <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu_hat - b_i - b_u)/(n()+lambda))

  b_y <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(movieYear) %>%
    summarize(b_y = sum(rating - mu_hat - b_i - b_u - b_g)/(n()+lambda))

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "movieYear") %>%
    mutate(pred = mu_hat + b_i + b_u + b_g + b_y) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
```

```r
l_best <- l[which.min(rmses)]
```

l = 5 is the penalty term that minimizes the RMSE. We can see it by exploring the following graph.

```r
lambdas <- data.frame(l, rmses)

ggplot(data = lambdas, aes(x = l, y = rmses)) +
  geom_point() +
  xlab("Lambda value")+
  ylab("RMSE") +
  ggtitle("RMSE values depending on Penalty term lambda")
```



`Model 6.0`
Having the penalty term l = `l_best`, we come to define the final model:
`Model 6.0`: Yu,i = mu_hat + bi(l) + bu(l) + bg(l) + by(l) + Eu,i

```r
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l_best))

b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_i)/(n()+l_best))

b_g <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu_hat - b_i - b_u)/(n()+l_best))

b_y <- edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(movieYear) %>%
    summarize(b_y = sum(rating - mu_hat - b_i - b_u - b_g)/(n()+l_best))
```

```r
model6_predictions <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "movieYear") %>%
    mutate(pred = mu_hat + b_i + b_u + b_g + b_y) %>%
    pull(pred)
```

```r
model6_rmse <- RMSE(validation$rating, model6_predictions)
```

RMSE of the final model `Model 6.0`: 0.8642929.

3. Model results and model performance

Let's have a look at the model results.
We embed the RMSEs of the various models into a single table:

```
models <- c("Model 1.0 - Yu,i = mu_hat + Eu,i",
            "Model 2.0 - Yu,i = mu_hat + bi + Eu,i",
            "Model 3.0 - Yu,i = mu_hat + bi + bu + Eu,i",
            "Model 4.0 - Yu,i = mu_hat + bi + bu + bg + Eu,i",
            "Model 5.0 - Yu,i = mu_hat + bi + bu + bg + by + Eu,i",
            "Model 6.0 - Yu,i = mu_hat + bi(l) + bu(l) + bg(l) + by(l) + Eu,i")

rmse_models <- c(model1_rmse,
                 model2_rmse,
                 model3_rmse,
                 model4_rmse,
                 model5_rmse,
                 model6_rmse)

kable(data.frame(models = models, RMSE = rmse_models), caption = "Models performance")
```

Table 8: Models performance

models	RMSE
Model 1.0 - Yu,i = mu_hat + Eu,i	1.0612018
Model 2.0 - Yu,i = mu_hat + bi + Eu,i	0.9439087
Model 3.0 - Yu,i = mu_hat + bi + bu + Eu,i	0.8653488
Model 4.0 - Yu,i = mu_hat + bi + bu + bg + Eu,i	0.8649469
Model 5.0 - Yu,i = mu_hat + bi + bu + bg + by + Eu,i	0.8647606
Model 6.0 - Yu,i = mu_hat + bi(l) + bu(l) + bg(l) + by(l) + Eu,i	0.8642929

In the exercise of machine learning, we first started from a simple prediction where ratings are estimated through the simple average of all ratings, `mu_hat` (`Model 1.0`). This is not really a precise model because RMSE is higher than 1: this means that we would miss the correct prediction by 1 rating, on average.
Then we added the movie feature `bi`, so our linear model is the average rating adjusted with the movie effect (`Model 2.0`). This is the step lowers RMSE below 1. Then we add, in sequence, user effect `bu` (`Model 3.0`), genre effect `bg` (`Model 4.0`) and finally release-year effect `by` (`Model 5.0`). The additions that help lower RMSE the most are those of movie effect and user effect, suggesting that - as one would expect - the quality of the movie and the specificity of the user are the greatest drivers for ratings prediction. Genre and release-year help to refine the RMSE a bit, but these features have a limited impact.
In the final step we applied to our explanatory terms a penalty parameter `l` - chosen after a dedicated tuning-parameter session, where `l_best` was elected as the best among many tries. The aim of this parameter is to lower the importance of the feature and shrink its coefficient towards zero if the feature is not enough stable (i.e., if its size is small). Given that our features are all quite helpful in lowering RMSE, the final step of `Model 6.0` where all the features are adjusted by the penalty term gives an RMSE which is not too different from that of the previous `Model 5.0`. All the same, it is the lowest RMSE, therefore `Model 6.0` is our best model.

4. Conclusion

In this analysis, we gained insights into the variables that can predict movie ratings.
We tried a simple linear model, each time adding a new feature as explanatory variable, so to see, step by step, how the performance metric RMSE would update. As already said, each of the variables of movie effect, user effect, genre effect and movie release-year have a positive impact in refining the model performance, but especially the movie effect and user effect are somewhat the most significant variables that can be observed.
The dataset has 10 million rows, 90% of them being in the training set: this means that the models were trained on a very big sample-size and this is the strength of the method. The higher the number of observations, the higher the chance that the model algorithm is precise.
At the same time, just because the number of observations was so big, we couldn't test complex models because of run-time that is required by such an amount of rows. More complex models could have grasped some relationships between dependent and independent variables way better than our model. The simplicity of the final model is the real limitation of this analysis.
For future work, all the same, other models could be tested, starting from the preliminary insights gained with this analysis. For example, the principal component analysis could allow to reduce the features into the few, most explanatory ones. This could solve run-time issues and allow for the application of more complex models, such as the generalized additive models, where relations other than linear could be detected.