MovieLens-Report

Marian Dumitrascu

March 17, 2019

MovieLens Data Analysis Report

Introduction

This document is a report of MovieLens data analysis prepared for Capstone project *PH125.9 Data Science: Capstone*.

The objective is to explore the data, gain insights, and conclude about ways to predict user ratings or make movie recommendations based on previous user activity. For this I used the methods presented in the course *PH125.8x Data Science: Machine Learning*, as well as several other techniques based on previous courses and papers. Generally, I tried to add something new or show I have a solid understanding of what was presented.

In the first sections of the report I prepare data for analysis and show how is organised, then I focus on the variability of rating and popularity over time: both date released and date of rating. Then, I do the same for movie and user effect.

The last section I make an analysis and compute RMSE based on Singular Value Decomposition (SVD) that was introduced in the course.

Initial data loading code was provided online. It partitions data into two chunks: edx - for model fitting, and validation for inferring the predictions. I subsequently divided edx into two parts: edx_train_set - for training, and edx_validation_set - for cross validation.

For this report I used a subset of data in order to make it run faster. In my second report - MovieLens RMSE - I use the full data.

Data Preparation

A short look at the data and we (I use "we" many times in this report, is still me) observe that timestamp is in seconds since 1970 and the genre column is a pipe delimited list of genres. For our analysis we'll convert *timestamp* into a more usable format called *rating_date*, extract year, and create a new column *genre_1* containing just the first genre in the list - in order to simplify the RSMS prediction later.

```
# note: ideally we would heve done this before data spliting,
# but I want to leave thet code intact.

# since we're doing this more than once, create a function
extract_first_genre <- function(a_movie_set)
{
  # get the first genre from list in genres column
  tmp <- as.data.frame(sapply(str_split(a_movie_set$genres, "\\|"), first))

  # set the column name
  names(tmp) <- c("genre_1")

  # bind the new column to the dataset
  a_movie_set <- bind_cols(a_movie_set, tmp)
```

```
  rm(tmp) # clean some memory and the environment

  a_movie_set
}

# do this for both edx and validation sets
edx <- extract_first_genre(edx)
validation <- extract_first_genre(validation)

# additionally, change the genre_1 column to be factor, to make joining easier
genre_levels <- sort(union(levels(edx$genre_1), levels(validation$genre_1))) # get the levels from both

edx <- edx %>% mutate(genre_1 = factor(genre_1, genre_levels))
validation <- validation %>% mutate(genre_1 = factor(genre_1, genre_levels))

# create "rating_date" and "year_rated" columns in edx data set
edx <- edx %>%
  mutate(rating_date = structure(edx$timestamp,class=c('POSIXt','POSIXct'))) %>%
  mutate(year_rated = as.integer(year(rating_date)))

# get rid of "timestamp" and "date_rating", we dont need them anymore
edx <- edx %>% select(-timestamp, -rating_date)

nicetable <- head(edx) %>% knitr::kable()

# some cosmetic settings
# knitr::opts_chunk$set(fig.width=24, fig.height=16)
```

edx dataset looks like this

userId	movieId	rating	title	genres	genre_1	year_rated
1	122	5	Boomerang (1992)	Comedy|Romance	Comedy	1996
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	Action	1996
1	316	5	Stargate (1994)	Action|Adventure|Sci-Fi	Action	1996
1	329	5	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	Action	1996
1	355	5	Flintstones, The (1994)	Children|Comedy|Fantasy	Children	1996
1	356	5	Forrest Gump (1994)	Comedy|Drama|Romance|War	Comedy	1996

We notice that the release year is specified in the title column (Example: "Sense and Sensibility (1995)" year released is 1995). For the analysis sake, we'll extract the year into a new column called *year_released*. Data provided is very consistent and we wont need to make any additional safety coding.

```
edx <- edx %>% mutate(year_released = as.integer(str_sub(title, -5, -2))) %>%
  mutate (title = str_trim(str_sub(title, 1, -7)))

nicetable <- head(edx) %>% knitr::kable()
```

edx dataset looks now like this

userId	movieId	rating	title	genres	genre_1	year_rated	year_relea
1	122	5	Boomerang	Comedy|Romance	Comedy	1996	1

userId	movieId	rating	title	genres	genre_1	year_rated	year_relea
1	292	5	Outbreak	Action|Drama|Sci-Fi|Thriller	Action	1996	1
1	316	5	Stargate	Action|Adventure|Sci-Fi	Action	1996	1
1	329	5	Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	Action	1996	1
1	355	5	Flintstones, The	Children|Comedy|Fantasy	Children	1996	1
1	356	5	Forrest Gump	Comedy|Drama|Romance|War	Comedy	1996	1

For genre analysis we will expand *genres* pipe line delimited field into rows, using *unnest* function from dplyr package. For same movie and userId we will now have a row for each genre. For this purpose we'll create a new dataset called *edx_genres* to store expanded records.

```
# expand "genres"
edx_genres <- edx %>% mutate(genres = str_split(as.character(genres), "\\|")) %>% unnest(genres)

# remove "genres" column from edx, we dont need it
edx <- edx %>% select(-genres)

nicetable <- head(edx_genres) %>% knitr::kable()
```

edx_genres dataset looks like this

userId	movieId	rating	title	genre_1	year_rated	year_released	genres
1	122	5	Boomerang	Comedy	1996	1992	Comedy
1	122	5	Boomerang	Comedy	1996	1992	Romance
1	292	5	Outbreak	Action	1996	1995	Action
1	292	5	Outbreak	Action	1996	1995	Drama
1	292	5	Outbreak	Action	1996	1995	Sci-Fi
1	292	5	Outbreak	Action	1996	1995	Thriller

.

```
# split edx further into edx_train_set and edx_validation_set to be useed in cross-validation
edx_validate_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train_set <- edx[-edx_validate_index, ]
temp <- edx[edx_validate_index, ]

# Make sure userId and movieId in edx_validation_set set are also in edx_train_set set
# remove rows in temp that userId or movieId not found in edx_train_set
edx_validation_set <- temp %>%
  semi_join(edx_train_set, by = "movieId") %>%
  semi_join(edx_train_set, by = "userId")

# get removed rows and put them edx_train_set
removed <- anti_join(temp, edx_validation_set, by = c("movieId", "userId"))
edx_train_set <- rbind(edx_train_set, removed)

rm(temp, edx_validate_index, removed) # clean some memory and the environment
```

Data Exploration

Lets take a quick look at what do we have in our sample.

```
summary <- edx %>% summarize(
  n_users = n_distinct(userId),
  n_movies = n_distinct(movieId),
  n_genres = n_distinct(genre_1),
  min_year_rated = min(year_rated),
  max_year_rated = max(year_rated),
  min_year_released = min(year_released),
  max_year_released = max(year_released))
```

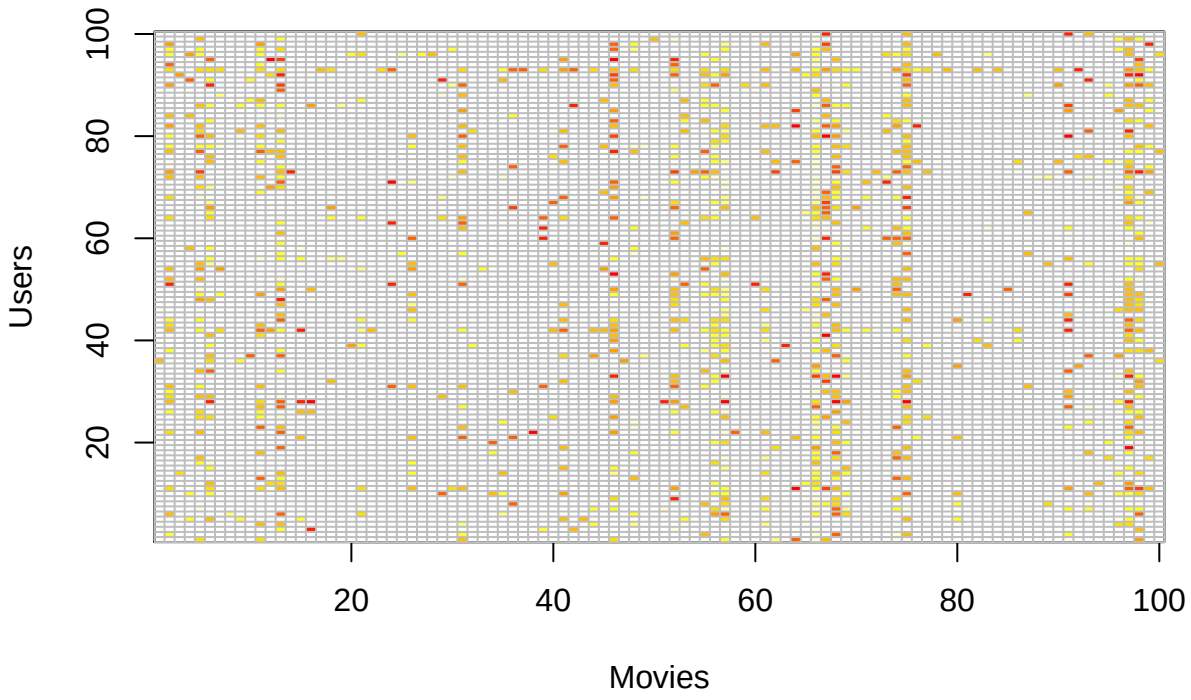
Our training data contains 450120 movie rating records, from 4089 unique users, for 8898 movies, rated between 1996 and 2009, and released between 1915 and 2008.

Data Shape. The Sparse Matrix

The matrix User to Movie rating is a sparse matrix. That is: with a large number of values unfilled or NA. This is a reason for predicting user rating to be difficult. I will replicate findings from the course and also add some thoughts of my own.

First we compute a matrix with users on rows, movies on columns and rating as values (in the course we put 1 as values). We will code the ratings with different color. We also order data by genre and number of votes to see if we can observe any patterns.

```
# select top 100 users by number of ratings they gave
users <- edx %>% group_by(userId) %>%
  summarize(n = n()) %>%
  top_n(100, n) %>%
  arrange(n) %>% # sort by numbeer of ratings to see a patern in the image
  pull(userId)

edx %>% filter(userId %in% users) %>%
  arrange(genre_1) %>% # arrange genre to see if we get any bands corresponding to genres
  select(userId, movieId, rating) %>%
  # mutate(rating = 1) %>%  # put a 1 for each user/movie
  spread(movieId, rating) %>% # create a column for each movieId w rating as values
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t() %>%
  image(1:100, 1:100, . , xlab = "Movies", ylab = "Users")

# add a grid
abline(h=0:100+0.5, v=0:100+0.5, col="grey")
```




```
rm(users)
```

Obviously we notice the data is pretty sparse. But since I sorted data by genre, we can also notice some bands that correspond to genres.

Distribution by Genre, Year Released and Year Rated

Lets show some tables first then I'll dive into visualizations.

For this we use the data set with genres expanded *edx_genres*:

```
edx_genres %>% group_by(genres) %>%
  summarize(
    rating_avg = format(round(mean(rating),2), nsmall = 2)) %>%
  arrange(desc(rating_avg)) %>%
  knitr::kable()
```

genres	rating_avg
Film-Noir	4.03
Documentary	3.85
War	3.80
IMAX	3.75
Drama	3.70
Mystery	3.70

genres	rating_avg
Crime	3.69
Animation	3.61
Musical	3.58
Romance	3.57
Western	3.56
Thriller	3.52
Adventure	3.51
Fantasy	3.50
Comedy	3.45
Action	3.44
Children	3.42
Sci-Fi	3.41
Horror	3.27

We can see that *Film-Noir* is listed first (or among the first, depending of the data sample), this category has only few votes. This makes a strong case for using regularization in our model (penalizing movies that have low amount of ratings).

The highest amount of ratings (popularity) tells a different story:

```
# arrange data by popularity and plot
edx_genres %>% group_by(genres) %>%
  summarize(
    n_ratings = n(),
    n_users = n_distinct(userId),
    n_movies = n_distinct(movieId),
    rating = format(round(mean(rating),2), nsmall = 2)) %>%
  arrange(desc(n_users)) %>%
  knitr::kable()
```

genres	n_ratings	n_users	n_movies	rating
Comedy	177694	4088	3168	3.45
Drama	195253	4087	4353	3.70
Thriller	117057	4070	1559	3.52
Action	128726	4069	1321	3.44
Romance	85938	4068	1432	3.57
Adventure	96314	4067	911	3.51
Crime	66850	4013	974	3.69
Sci-Fi	68165	3976	683	3.41
Fantasy	46576	3878	495	3.50
War	25376	3732	396	3.80
Children	37300	3704	494	3.42
Mystery	28545	3549	450	3.70
Horror	34301	3451	932	3.27
Musical	21699	3374	355	3.58
Animation	23436	3353	266	3.61
Western	9127	2591	196	3.56
Film-Noir	5872	1738	119	4.03
Documentary	4190	1313	350	3.85
IMAX	353	294	20	3.75

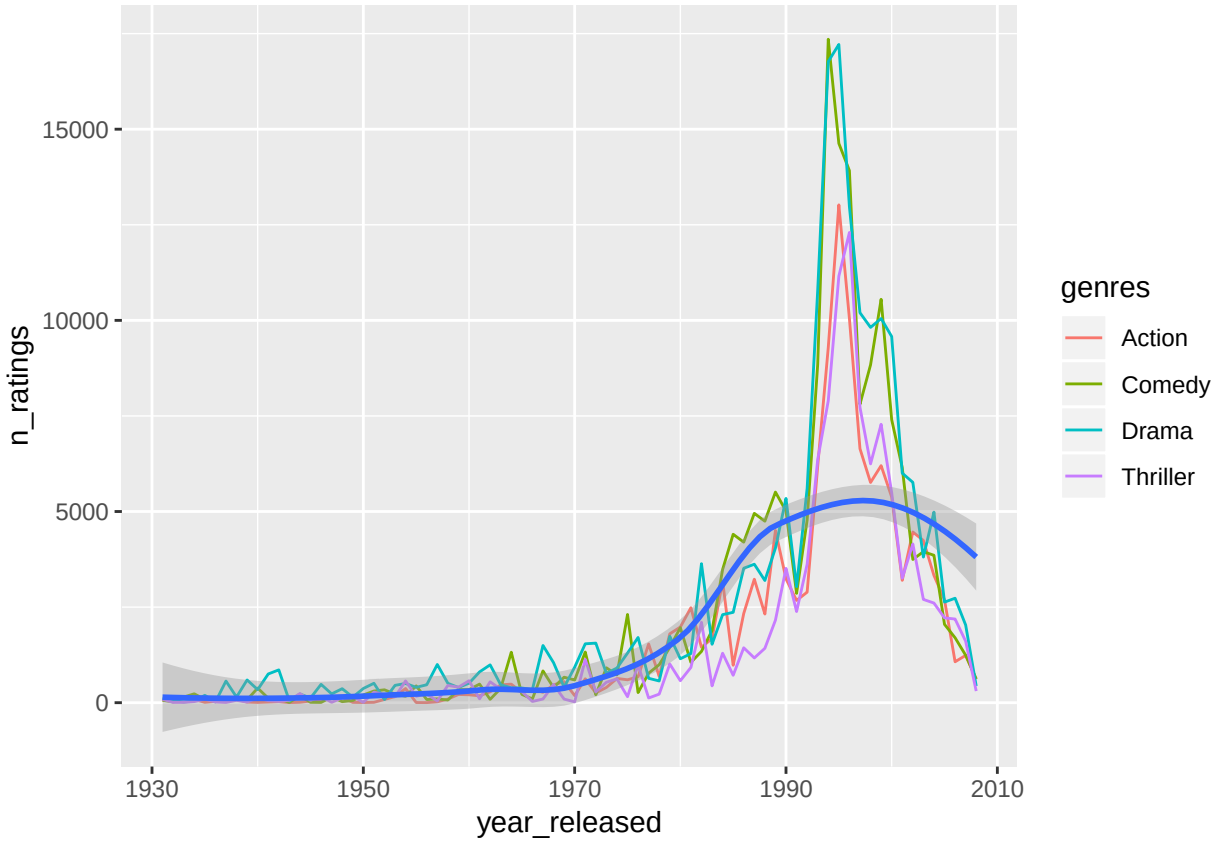
```r
# thet the top 4 genres, we'll usee it later
top_genres <- edx_genres %>% group_by(genres) %>%
  summarize(n_ratings = n()) %>%
  arrange(desc(n_ratings)) %>%
  top_n(4, n_ratings) %>% pull(genres)
```

We notice now that Comedy and Drama are most popular, while Film-Noir is among the last.

Lets see how most popular genres varies in popularity year by year (year of release and year of rating):

```r
genres_popularity <- edx_genres %>%
  select(movieId, year_released, genres) %>% # keep only what we're interested in
  group_by(year_released, genres) %>% # group by year and genre
  summarize(n_ratings = n())


genres_popularity %>%
  filter(year_released > 1930) %>%
  filter(genres %in% top_genres) %>%
  ggplot(aes(x = year_released, y = n_ratings)) +
    geom_line(aes(color=genres)) +
    geom_smooth() +
    scale_fill_brewer(palette = "Paired")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
rm(genres_popularity)
```

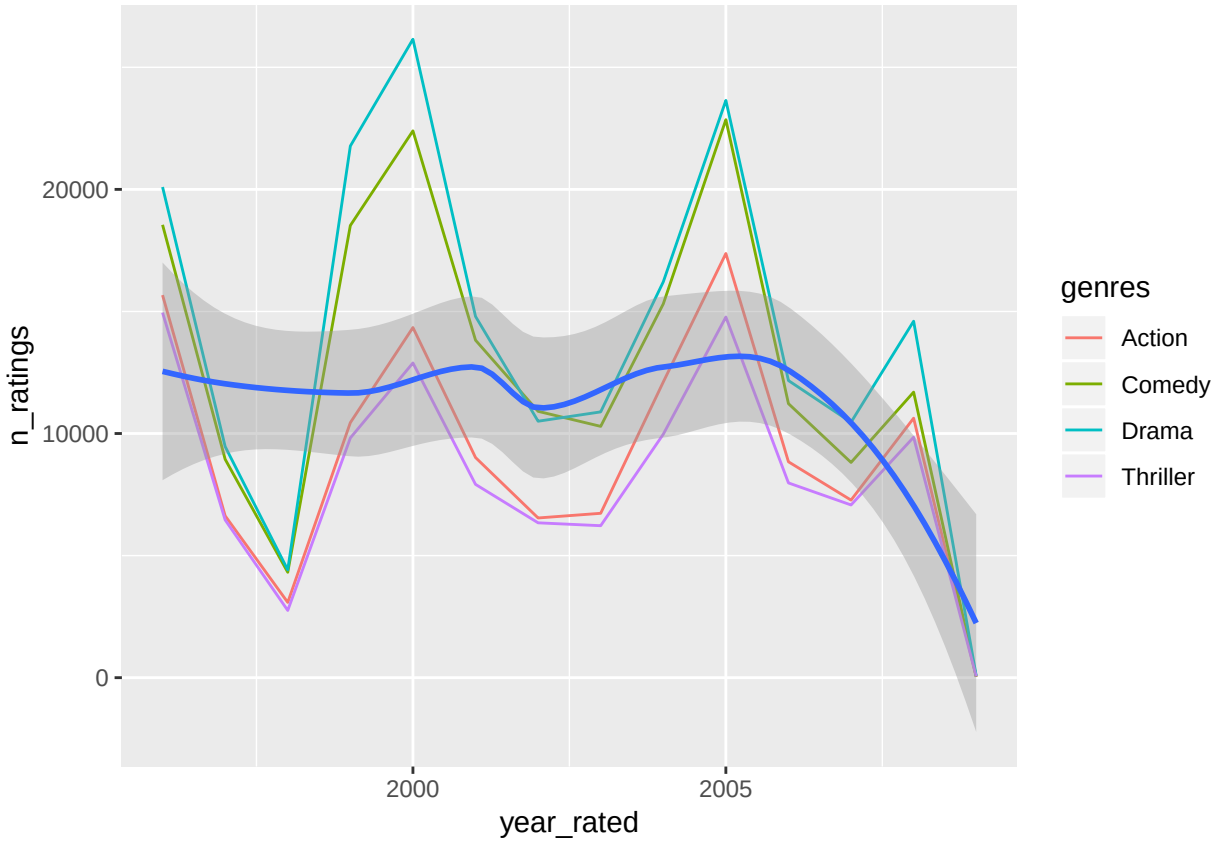
There are at least two things we notice:

1. The popularity decrease abruptly toward the end years of rating period. This can be explained by the fact that the interval between release date and end of survey haven't allowed them to be popular.
2. They are wiggly. I can only speculate at this point that there is a *date of release effect*, that means for example, after the release of a blockbuster comedy movie in a certain year, users want to see more comedies released around same period.

Now, lets do the same thing - popularity - year by year, but for *rating year* (the date when they were rated).

```
genres_popularity <- edx_genres %>%
  select(movieId, year_rated, genres) %>% # keep only what we're interested in
  group_by(year_rated, genres) %>% # group by year and genre
  summarize(n_ratings = mean(n()))


genres_popularity %>%
  filter(genres %in% top_genres) %>%
  ggplot(aes(x = year_rated, y = n_ratings)) +
    geom_line(aes(color=genres)) +
     geom_smooth() +
     scale_fill_brewer(palette = "Paired")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
rm(genres_popularity)
```

This is what we can observe here:

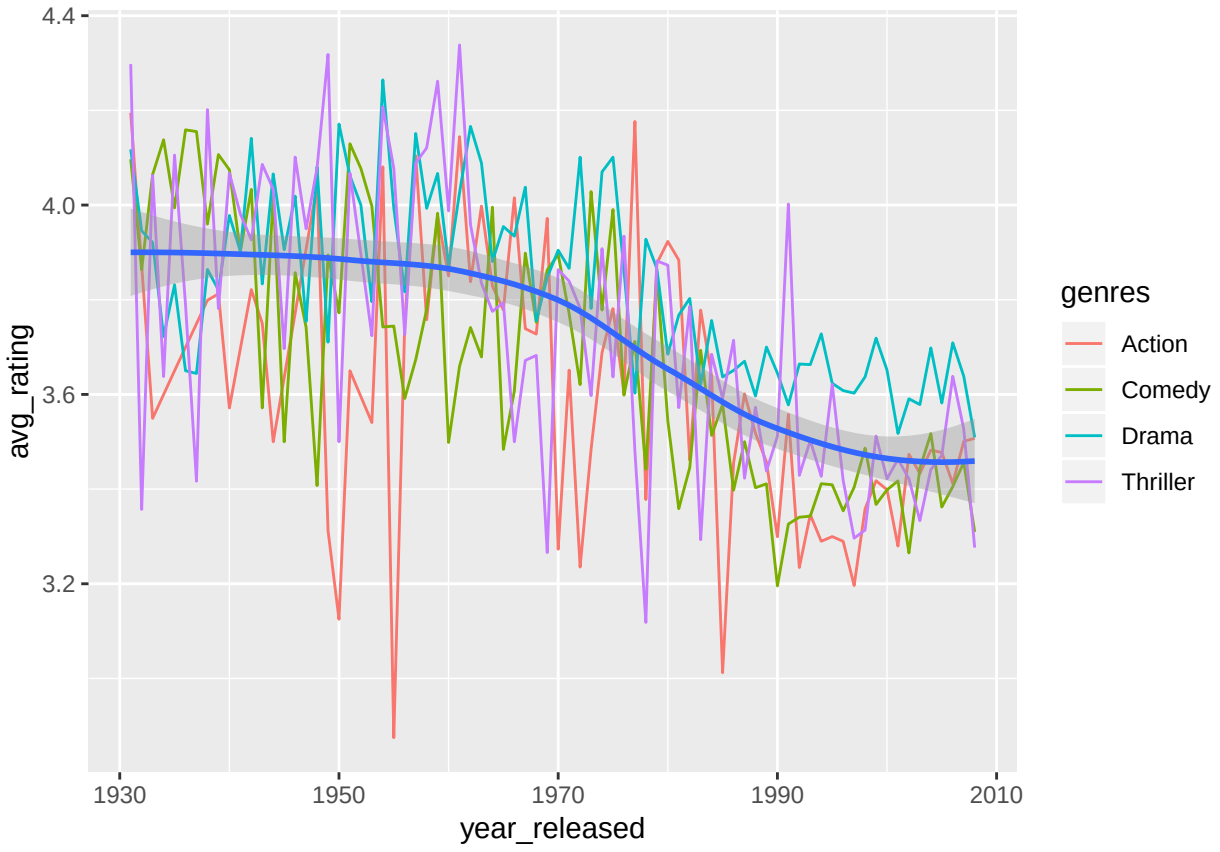
1. Popularity still decreases toward the end of the survey period. This has the same cause as in previous case.
2. There are only two major peaks in popularity. One around 2000, and one around 2005. We can speculate at this point that around those periods, more movies were added to the database, causing the increase in number of ratings. I don't believe this would add a lot of significance in MSRP as is. Perhaps showing a rating date by month would make more sense, but I will not go into that detail in this report.

Lets analyse the average rating over time for major genres. That will give us a sense of users taste in time.

```
# get data from the edx_genres that has the genres list expanded into rows
genres_popularity <- edx_genres %>%
  select(movieId, year_released, genres, rating) %>% # keep only what we're interested in
  group_by(year_released, genres) %>% # group by year and genre
  summarize(avg_rating = mean(rating))


genres_popularity %>%
  filter(year_released > 1930) %>%
  filter(genres %in% top_genres) %>%
```

```
  ggplot(aes(x = year_released, y = avg_rating)) +
    geom_line(aes(color=genres)) +
    geom_smooth() +
    geom_abline() +
    scale_fill_brewer(palette = "Paired")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
rm(genres_popularity)
```

Here is what we notice here:

1. Old movies tend to be rated higher (which is not actually true), but there is also higher variability. Both observations are caused by low number of old movies in the db
2. Average popularity is wiggly, but not all genres are in sync. For example, when Thriller popularity increases, Comedy movies decreases and vice-versa. On the other side Thriller and Action are at times, in sync.

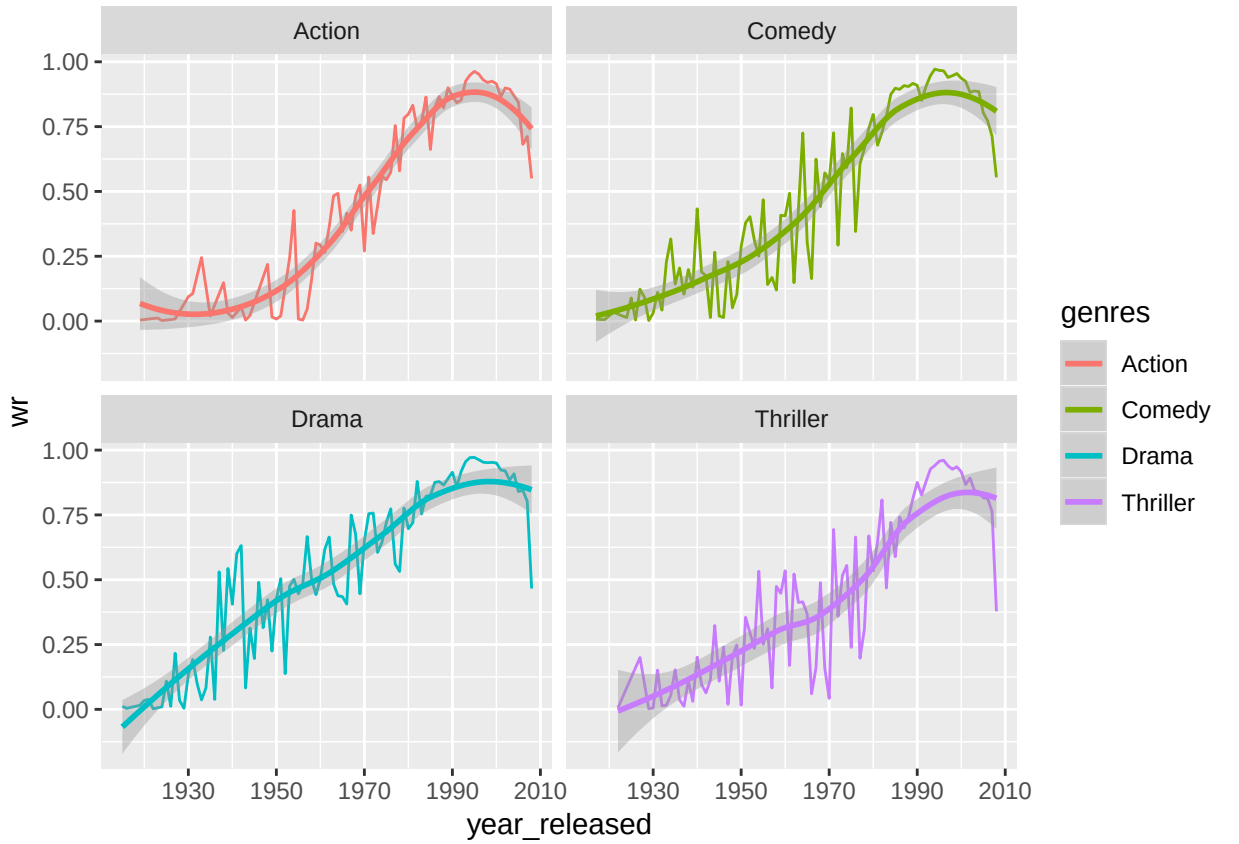
A conclusion here is that users taste profile varies in time (time of the movie release). An accurate model should consider this trend.

We'll show now the best years by genre, based on user ratings. Here I used a *weighted rating* function that I found it in Kaggle article about movie prediction (you can find their excellent analysis here: https://www.kaggle.com/stevelatuidahodotedu/movielens-dataset-analysis). This is another way to account for

regularization, and it gives a higher score to movies that have a larger amount of reviews. The term m in the function can be actually optimized, but I will use a value of 500 for now.

```
# rate average over the whole dataset, we'll use this all over
rate_avg <- mean(edx$rating)

# R = average for the movie (mean) = (Rating)
# v = number of votes for the movie = (votes)
# m = minimum votes required to be listed in the Top 250, this can be optimized, i will use 500
# C = the mean vote across the whole report
weighted_rating <- function(R, v, m, C) {
  return (v/(v+m))*R + (m/(v+m))*C
}

# organise data by decades of year relesed.
genres_rating <- edx_genres %>%
  mutate(decade = year_released %/% 10 * 10) %>%
  group_by(year_released, genres) %>%
  summarize(count = n(), rating_avg = mean(rating)) %>%
  ungroup() %>%
  mutate(wr = weighted_rating(mean, count, 500, mean(mean))) %>%
  arrange(year_released)

# plot a faceted graph of the weighted average by year relesed.
genres_rating %>%
  filter(genres %in% top_genres) %>%
  ggplot(aes(year_released, wr)) +
  geom_line(aes(group = genres, color=genres)) +
  geom_smooth(aes(group = genres, color=genres)) +
  facet_wrap(~genres)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can see that movies are consistently getting better ratings over time. This is in contrast with the observations in the previous section, but is actually more accurate due to the use of weighted average.

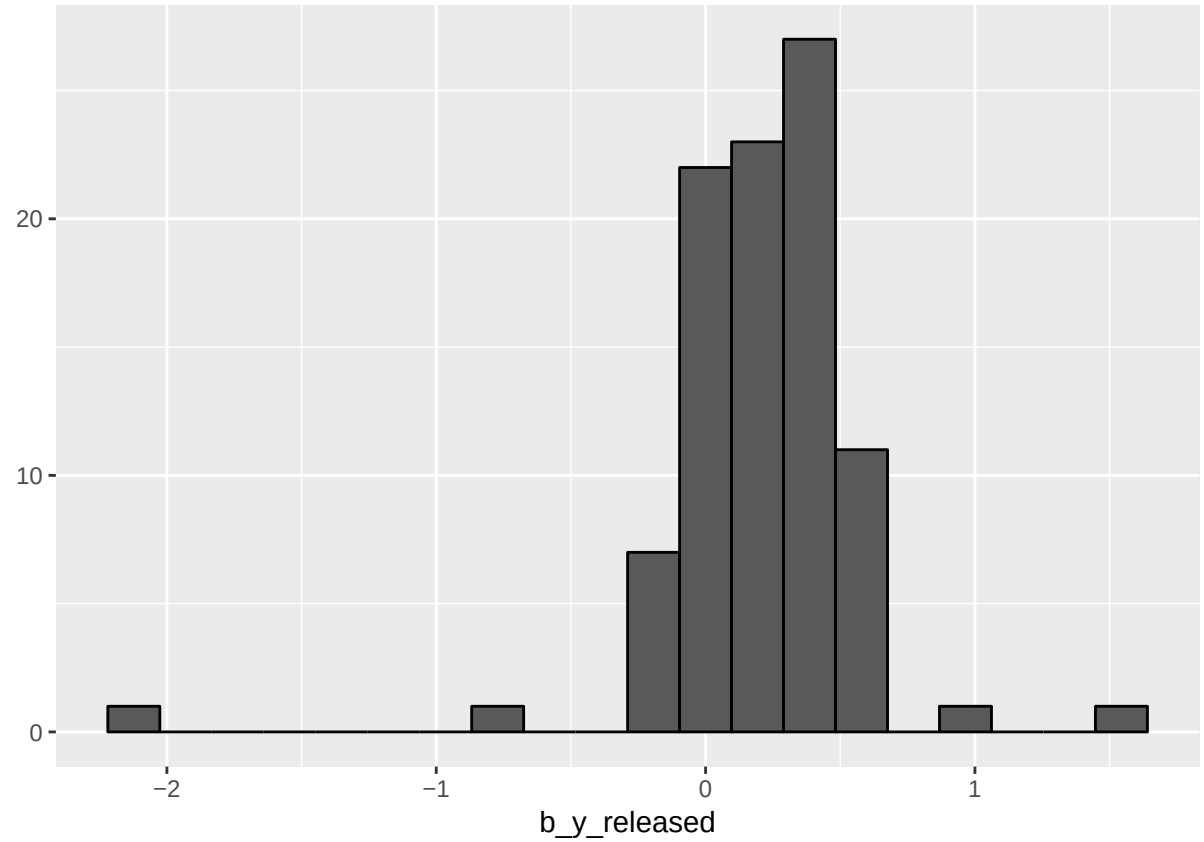
Biases and Effects

In this section I will study various effects (or biases) and decide which ones are good for a prediction model.

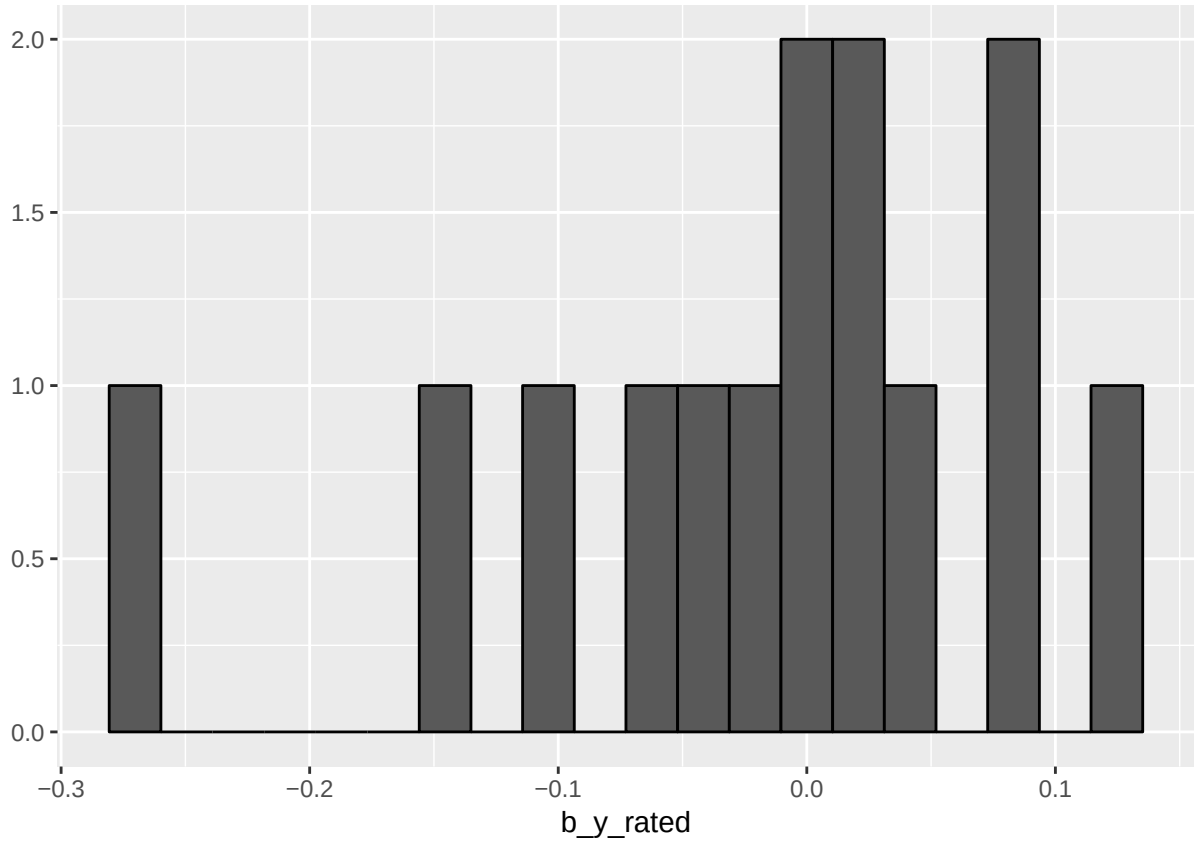
Year Released and Year Rated Effect

To predict a user rating for a movie we can use the ratings given to movies similar with that movie, or ratings given to that movie by users similar with that user (to paraphrase the course). We can also study the year released and year rating effects. I will compute here the *b_y_released* and *b_y_rated effects*, and plot their distribution as histograms.

```
# get the movie effect in b_y_1
movie_avgs <- edx_train_set %>%
  group_by(year_released) %>%
  summarize(b_y_released = mean(rating - rate_avg))

# plot the distribution of b_y around 0
movie_avgs %>%  qplot(b_y_released, geom = "histogram", bins = 20, data = ., color = I("black"))
```




```
# get the user effect in b_y_rating
movie_avgs <- edx_train_set %>%
  group_by(year_rated) %>%
  summarize(b_y_rated = mean(rating - rate_avg))

movie_avgs %>%  qplot(b_y_rated, geom = "histogram", bins = 20, data = ., color = I("black"))
```

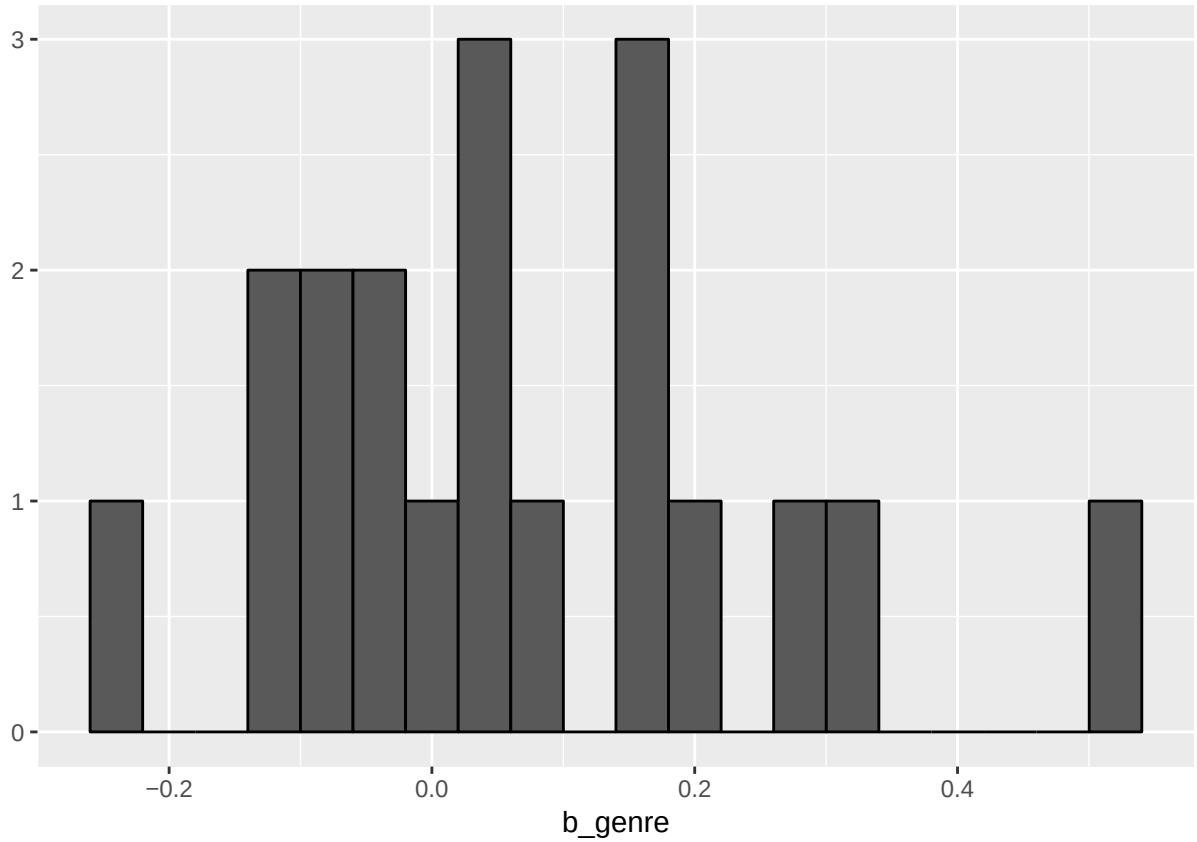


Year released resemble a normal distribution, while the rating year doesn't. I will exclude it from the model.

Genre Effect

I'm also curious to see the distribution of the genre effect. I will compute this into *y_genre* of the *movie_avgs* dataset, and plot the distribution. For this, will use the *edx_genres* dataset that has the genres list expanded into a row for each genre.

```
# get genre effect and store it as b_rating in movie_avgs
movie_avgs <- edx_genres %>%
  group_by(genres) %>%
  summarize(b_genre = mean(rating - rate_avg))

movie_avgs %>%  qplot(b_genre, geom = "histogram", bins = 20, data = ., color = I("black"))
```

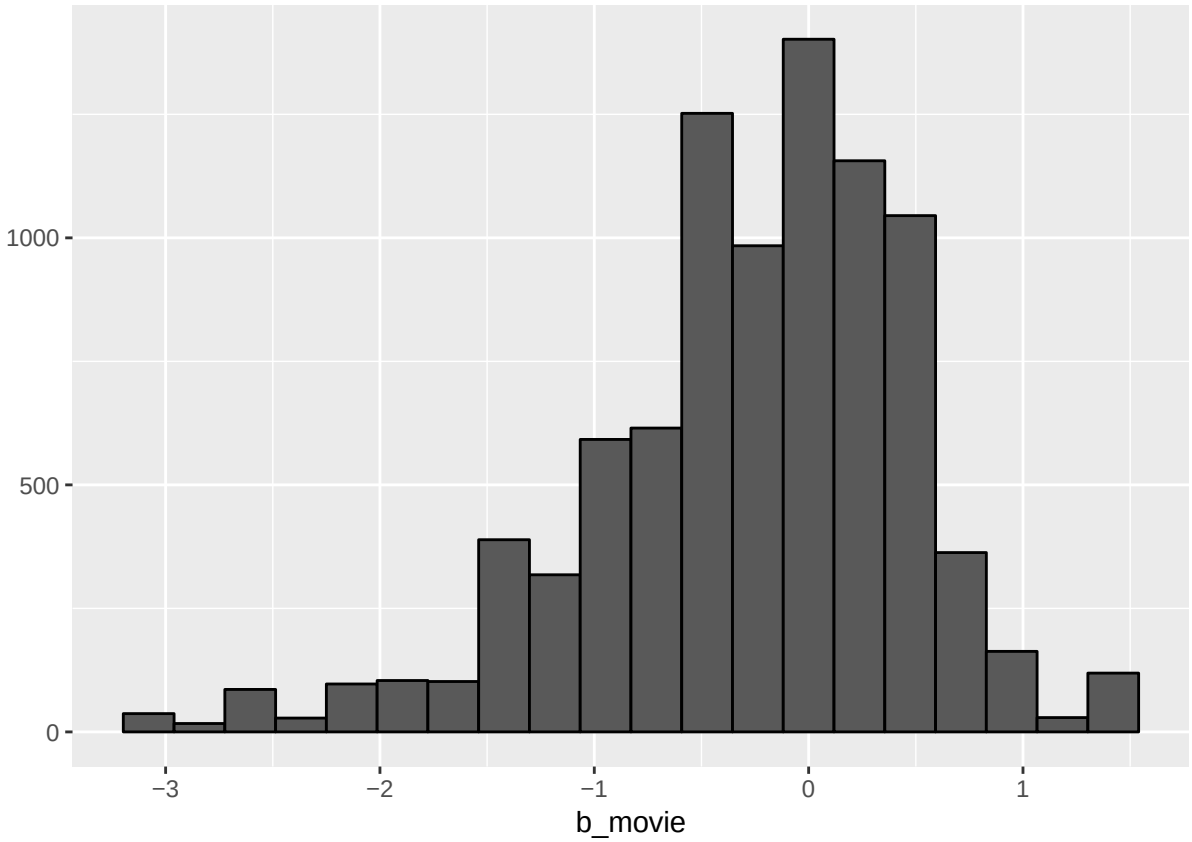


It is also resemble a normal distribution, hardly though. The isolated bars around extremes are caused yet once again by movies with low number of ratings.

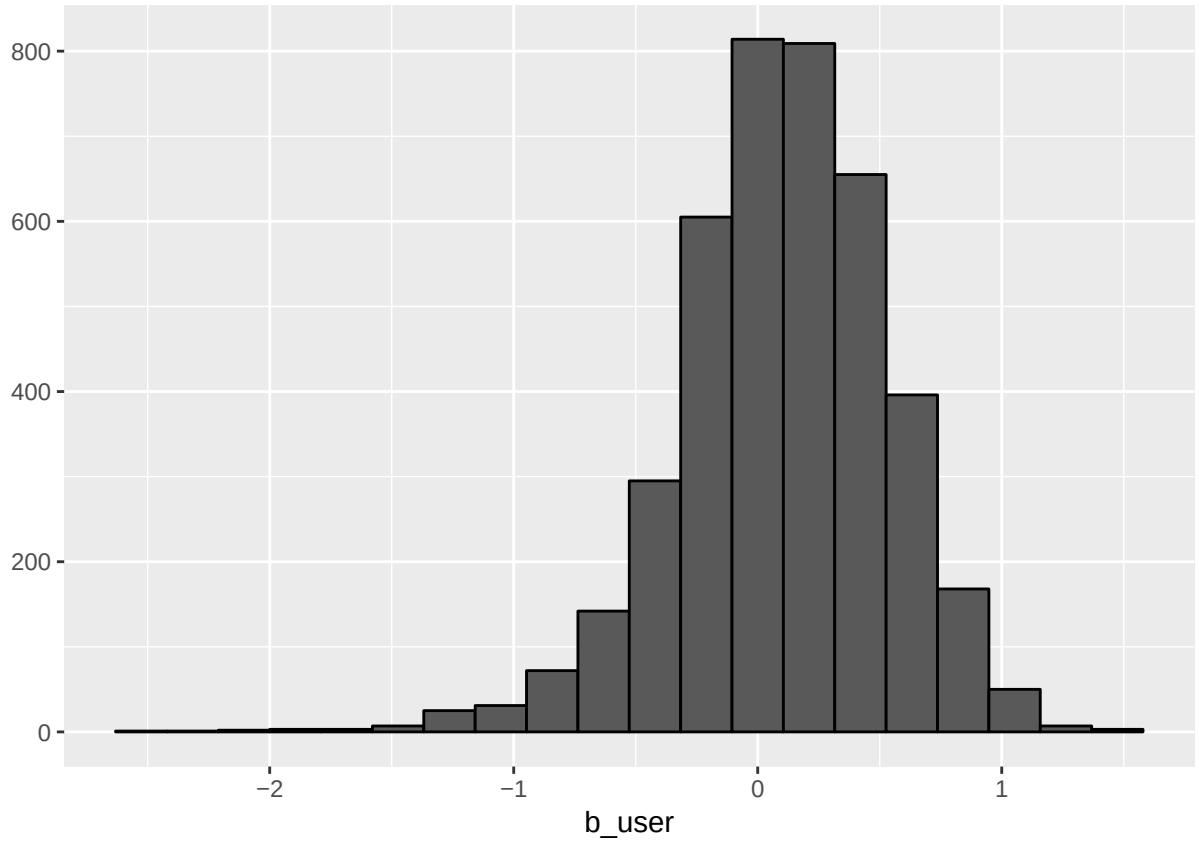
Movie and User Effect

Lets do the same plots for movie and user effects.

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - rate_avg))

movie_avgs %>%  qplot(b_movie, geom = "histogram", bins = 20, data = ., color = I("black"))
```




```
movie_avgs <- edx %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - rate_avg))

movie_avgs %>%  qplot(b_user, geom = "histogram", bins = 20, data = ., color = I("black"))
```



Both movie and user effects show a clear normal distribution which makes them great candidates for my final RMSE calculation.

RMSE Using SVD

The course did a very good introduction to Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). However, it's not going into details about using it in a prediction model for movielens. I thought it would be useful if I try to fill this gap.

In this section I will predict movie ratings and compute RMSE using the decomposition of the matrix y which is a *n_users* by *n_movies* sparse matrix of *rating* values. Simply put this method tries to fill the missing values of this matrix based on existing values.

From what I read, all teams competing for Netflix challenge used SVD in one form or another, but only adding a clever regularization made the winner. I will not account for regularization in my SVD prediction because I'm not that clever. Instead, will select a sub sample from original data with users who voted more than 100 movies and movies that were voted more than 100 times. This should eliminate the need for regularization but still support the case of using SVD.

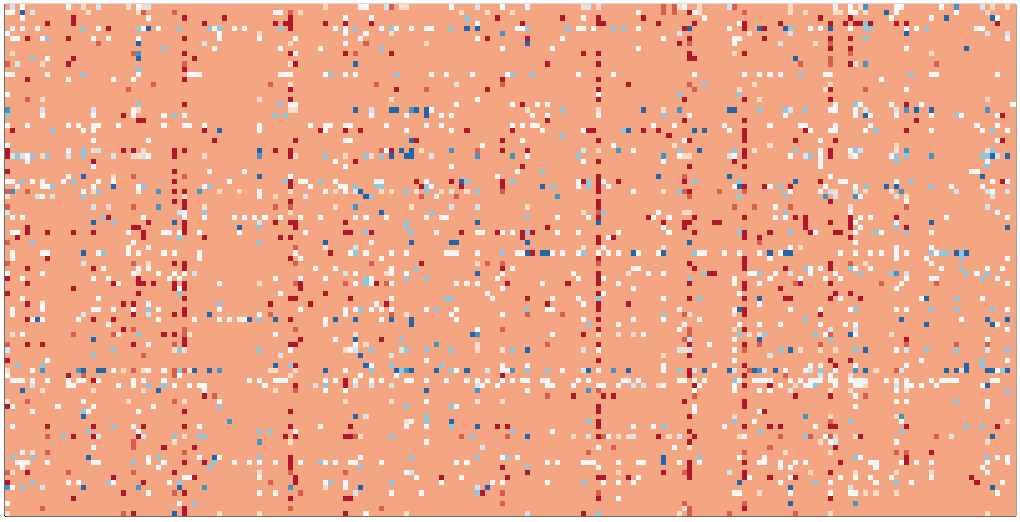
```
# remove some previously used data
rm(edx_train_set, edx_validation_set, edx, validation)

set.seed(1) # to have some replication consistency

# movielens <- movielens[sample(1:nrow(movielens), 100000, replace = FALSE), ]

# select only data that would not require regularization
```

```r
movielens <- movielens %>%
  group_by(movieId) %>%
  filter(n() > 100) %>%  ungroup() %>%
  group_by(userId) %>%
  filter(n() > 100) %>% ungroup()

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)

edx <- movielens[-test_index,] # training set
temp <- movielens[test_index,] # temporary validation set

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
     semi_join(edx, by = "movieId") %>%
     semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed) # add those rows to edx

rm(test_index, temp, removed)
```

Now, convert the training set into a users by movies matrix and compute the SVD of that matrix. Then show an image of a sample of that matrix.

```r
y <- edx %>%
  select(userId, movieId, rating) %>%  # select only columns we are interested in
  spread(movieId, rating) %>%  # expand rmovies row into columns
  as.matrix() # convert to matrix

# set row names to userId
rownames(y) <- y[, 1]

# remove first column which is users column
y <- y[,-1]

# initialize NA values with the global average
y[is.na(y)] <- rate_avg

# compute the residuals, center the rate values around 0
y <- y - rate_avg

# define a function that draws an image ot of a matrix
matrix_to_image <- function(x, zlim = range(x), ...){
  colors <- rev(RColorBrewer::brewer.pal(9, "RdBu"))
  cols <- 1:ncol(x)
  rows <- 1:nrow(x)

  image(cols, rows, t(x[rev(rows),, drop = FALSE]), xaxt = "n", yaxt = "n",
        xlab = "", ylab = "", col = colors, zlim = zlim, ...)
  # abline(h = rows + 0.5, v = cols + 0.5)
  # axis(side = 1, cols, colnames(x), las = 2)
}
```

```r
# compute the SVD using svd() fuction.
svd <- svd(y)

# get the components of svd into U D and V, for readability
# normaly should not do this in order to preserve memory
U <- svd$u
D <- svd$d
V <- svd$v

# choosee the best dimensions for the img
image_x <- min(c(nrow(y), 100))
image_y <- min(c(ncol(y), 200))

# draw the imagee
matrix_to_image(y[1:image_x, 1:image_y])
```



We notice much less sparsity in data due to our filter.

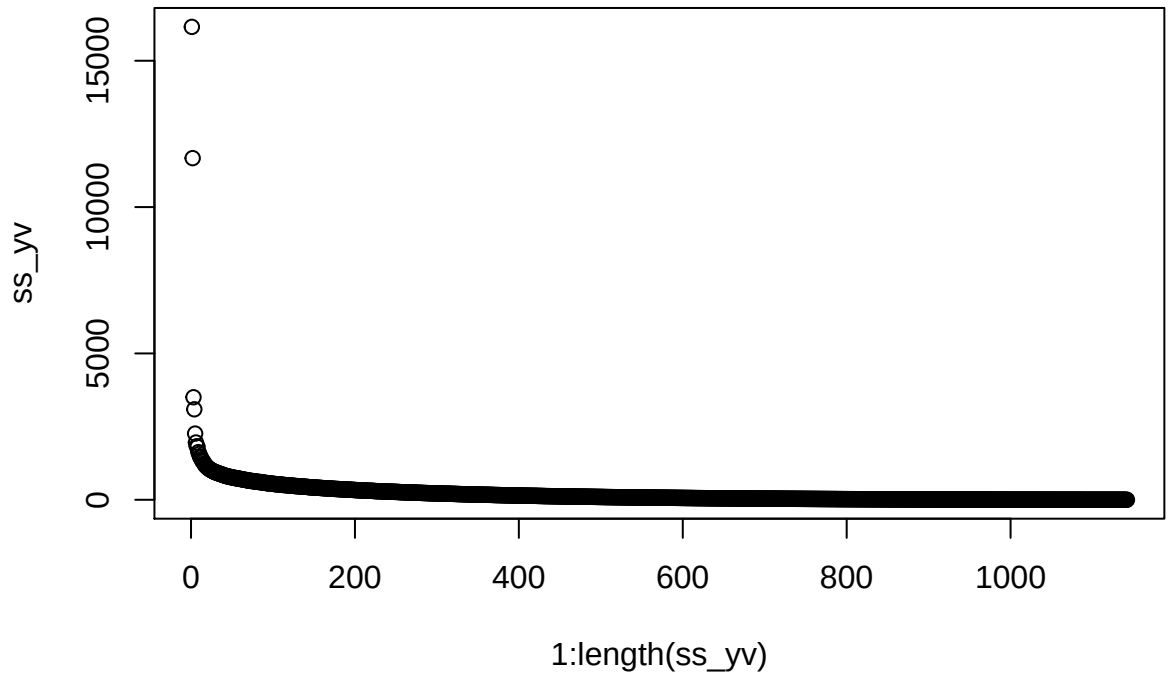
Verify that doing the dot multiplication of U x D x t(V) will produce something very close to the initial matrix

```r
y_svd <- U %*% diag(D) %*% t(V)
```

```r
# the original y and y_svd obtained by dot multiplying the decomposed matrix, are almost identical
max(abs(y_svd - y))
```

```
## [1] 4.574119e-14
```

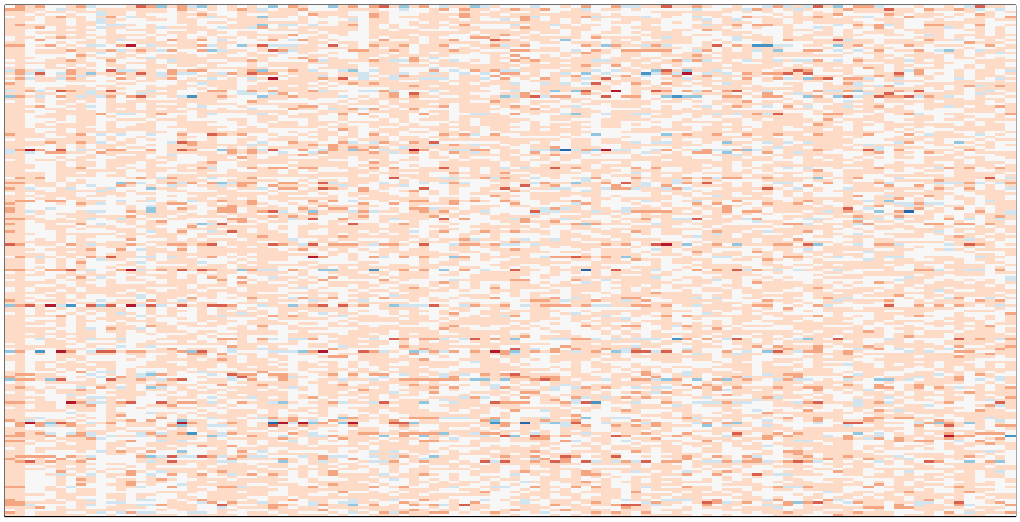
```r
# also plot the sum of square to see how variability varies in the decomposed matrix
ss_yv <- colSums((y_svd %*% V)^2)
plot(1:length(ss_yv), ss_yv)
```



This indicates that most of variability can be explained in the first components of the decomposition matrices.

Lets draw an image of the V matrix. This will give an insight of variability.

```r
image_x <- min(c(nrow(V), 200))
image_y <- min(c(ncol(V), 100))
V_for_image <- V[1:image_x, 1:image_y]
matrix_to_image(V_for_image, zlim = range(V_for_image))
```



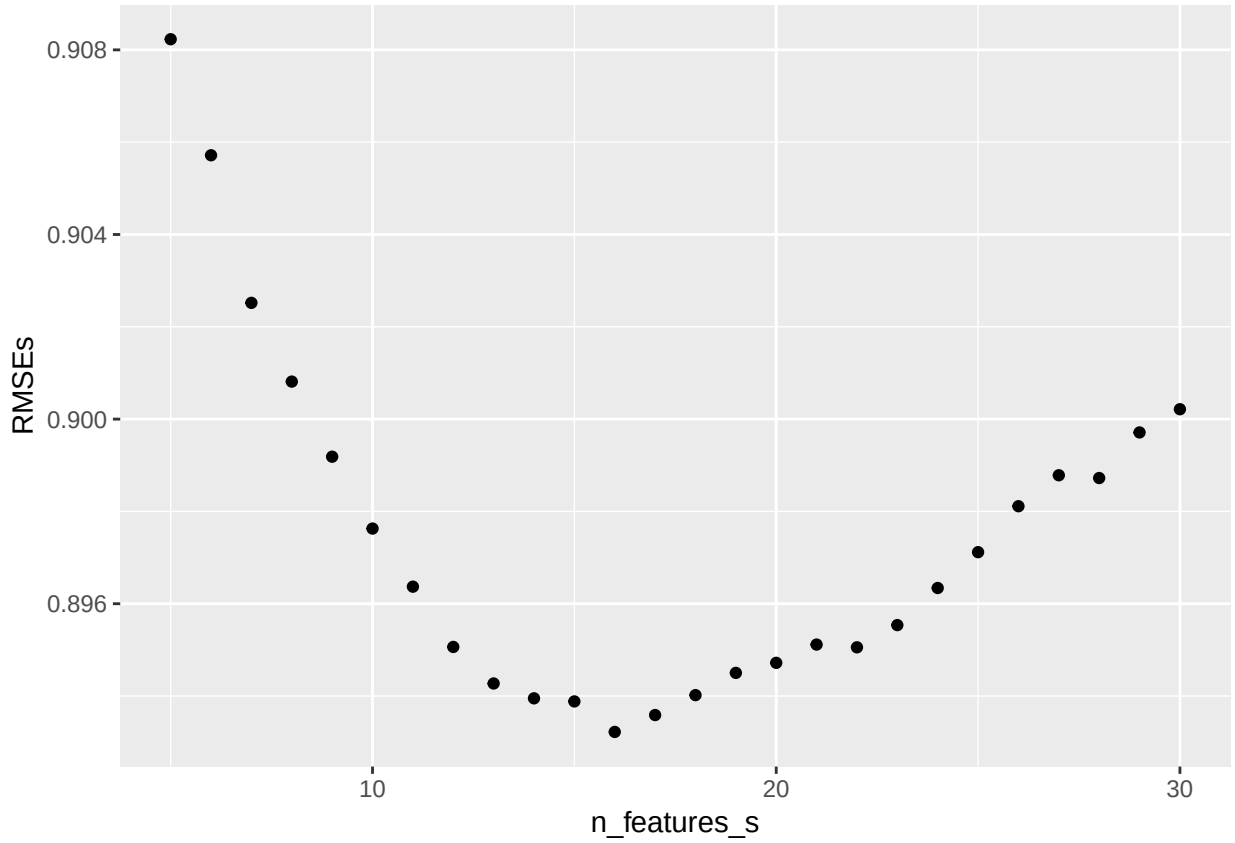
We notice that first lines in V have lower variability than the rest, they correspond to the features detected by our decomposition.

The number of features selected for prediction should be optimized. So I'll loop over an optimal range of values and determine the best value. Function *RMSE_feeat_options* computes *y_hat* of predicted rates, then it converts it back into a frame with userId, movieId, and rating (predicted rating) We'll join it with the training set by userId and movieId and get the predicted ratings for the validation set. (I tried to reference the matrix *y_hat* by the two indices inside a mutate of training dataset, but that didn't work.)

The plot will show how the best number of features are selected and minimum RMSE (This chunk will take a while to execute.)

```
# n_feature is the number of features in D that we choose to fit our model
# in fact this should be optimized
# n_features <- 15

n_features_s <- seq(5, 30)

RMSE_feat_optimize <- function(n_features){

    # compute y_hat by dot mltiplication: U x D x t(V)
    y_hat <- sweep(U[ , 1:n_features], 2, D[1:n_features], FUN = "*") %*% t(V[ , 1:n_features, drop = F

    # set colnames and rownames to movieId and userId respectively, take them from original y
    rownames(y_hat) <- rownames(y)
    colnames(y_hat) <- colnames(y)
```

```
    y_hat <- as.data.frame(y_hat)

    # bring rownames into userId column
    y_hat <- y_hat %>%  mutate(userId = as.integer(rownames(y_hat)))

    # gather movie columns into one column named movieId with rating as values (predicted ratings)
    y_hat <- y_hat %>%
      gather(movieId, rating, -userId)

    # convert movieIds to numeric
    y_hat <- y_hat %>%
      mutate(movieId = as.integer(movieId))

    # create a data set with predictions for the validation set
    # by joining the validation set with y_hat
    rating_predicted_set <- validation %>%
      select(userId, movieId) %>%     # select only what we're intrested in, we especially dont want ra
      left_join(y_hat, by = c("userId", "movieId")) %>%
      mutate(rating_pred = rate_avg + rating)

    RMSE(rating_predicted_set$rating_pred, validation$rating)
}

RMSEs <- sapply(n_features_s, RMSE_feat_optimize)

qplot(n_features_s, RMSEs)
```




```
# number of features detected that produces best RMSE
n_features_s[which.min(RMSEs)]
```

```
## [1] 16
```

```
# best rmse
rmse <- min(RMSEs)
rmse
```

```
## [1] 0.8932227
```

We obtained a RMSE of 0.8932227 which is promising, but this is in context of eliminating data that would require regularization. Nevertheless, SVD is a good method for predicting rates if we figure out also how to apply regularization.

RMSE Using Movie, User and Genre Effects Results

A complete run of the RMSE using the full data set is contained in **MoveLens_RMSE_v09.Rmd** file. I also attached the pdf generated for that file: **MoveLens_RMSE_v09.pdf** I used: movie, user and genre effect applying regularization. Here are the results from that report:

Method	RMSE
Global Rating Average	1.0612019

Method	RMSE
Movie Effect	0.9439087
Movie + User Effect	0.8653488
Movie + User Effect Regularized	0.8648201
Movie/User/Genre Effect Regularized	0.8647167

Conclusion

Recommendation systems is a special chapter in machine learning and has it's own challenges. In this report I tried to tackle some of these challenges and found that release date, genres, movie and user effects are good candidates to be used in a prediction model. I will use them in my final RMSE report.

I also did an evaluation of the RMSE computation using SVD, demonstrating that it's a good choice for a robust model.

One important conclusion is that regularization is the key ingredient that would make a prediction more accurate.

Beside the course, I also studied several papers and introduced a *weighted average* function that accounts for regularization. More information about this can be found here: https://www.kaggle.com/stevelatuidahodotedu/movielens-dataset-analysis and here: https://districtdatalabs.silvrback.com/computing-a-bayesian-estimate-of-star-rating-means

Obviously there are a lot more insights and analysis that can be done, but I hope I covered some of the most important in this report.

Thank you for reading!