MovieLens Recommender Report

Sougata Ghosh

25/09/2019

Introduction

Recommendation systems use ratings that users have given items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. On October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. In September 2009, the winners were announced.

The Netflix data is not publicly available, but the GroupLens research lab generated their own database, the **MovieLens** dataset, with over 20 million ratings for over 27,000 movies by more than 138,000 users. In this project we will be using the 10M version of the MovieLens Dataset and some of the data analysis strategies discussed in the HarvardX-PH125.8x Machine Learning course to create a movie recommendation system of our own. Specifically we will train a machine learning algorithm using the inputs in the train set to predict movie ratings in the validation set.

Loss Function

The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set. We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y_{u,i}}$. The RMSE is then defined as:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.RMSE is the metric we will use to evaluate the different machine learning models we build below.

Let's write a function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

Let us also install some relevant packages here.

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
## -- Attaching packages ------------------------------------------------------------ tidyverse 1.2.1 --

## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts --------------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE

## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year

## The following object is masked from 'package:base':
##
##     date
```

Create Train And Validation Set

We use the following code to generate the datasets edx and validation. We develop our models using the edx set. For a final test of each model, we predict movie ratings in the validation set as if they were unknown. RMSE will be used to evaluate how close our predictions are to the true values in the validation set.

```r
#################################
# Create edx set, validation set
#################################

# Note: this process could take a couple of minutes


# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
if (!exists("edx")){
  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")
  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")

  # Validation set will be 10% of MovieLens data

  set.seed(1)

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

  # 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)
  #Remove data not required
  rm(dl, ratings, movies, test_index, temp, movielens, removed)
}
```

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

Methods/Analysis

In this section:

1. We undertake data exploration to understand the problem at hand
2. We preprocess and transform the data to get age of movie, time of rating, etc.
3. We undertake data visualization to see how different factors such as movie, user, age of movie might have a bearing on ratings.
4. We discuss our modeling apporach.

Data Exploration

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

```
## # A tibble: 9,000,061 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     231      5 838983392 Dumb & Dumber (1~ Comedy
##  4      1     292      5 838983421 Outbreak (1995)   Action|Drama|Sci-Fi|T~
##  5      1     316      5 838983392 Stargate (1994)   Action|Adventure|Sci-~
##  6      1     329      5 838983392 Star Trek: Gener~ Action|Adventure|Dram~
##  7      1     355      5 838984474 Flintstones, The~ Children|Comedy|Fanta~
##  8      1     356      5 838983653 Forrest Gump (19~ Comedy|Drama|Romance|~
##  9      1     362      5 838984885 Jungle Book, The~ Adventure|Children|Ro~
## 10      1     364      5 838983707 Lion King, The (~ Adventure|Animation|C~
## # ... with 9,000,051 more rows
```

Each row represents a rating given by one user for one movie.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

We can see the number of unique users that provided ratings and how many unique movies were rated:

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

If we multiply **n_users** and **n_movies** we get a number of almost 750 million. However the edx dataset has only a little over 9 million rows. This implies that not every user rates every movie. So we can think of these data as a very large matrix, with users on the rows and movies on the columns, with many empty cells. Let's show the matrix for seven users and five movies.

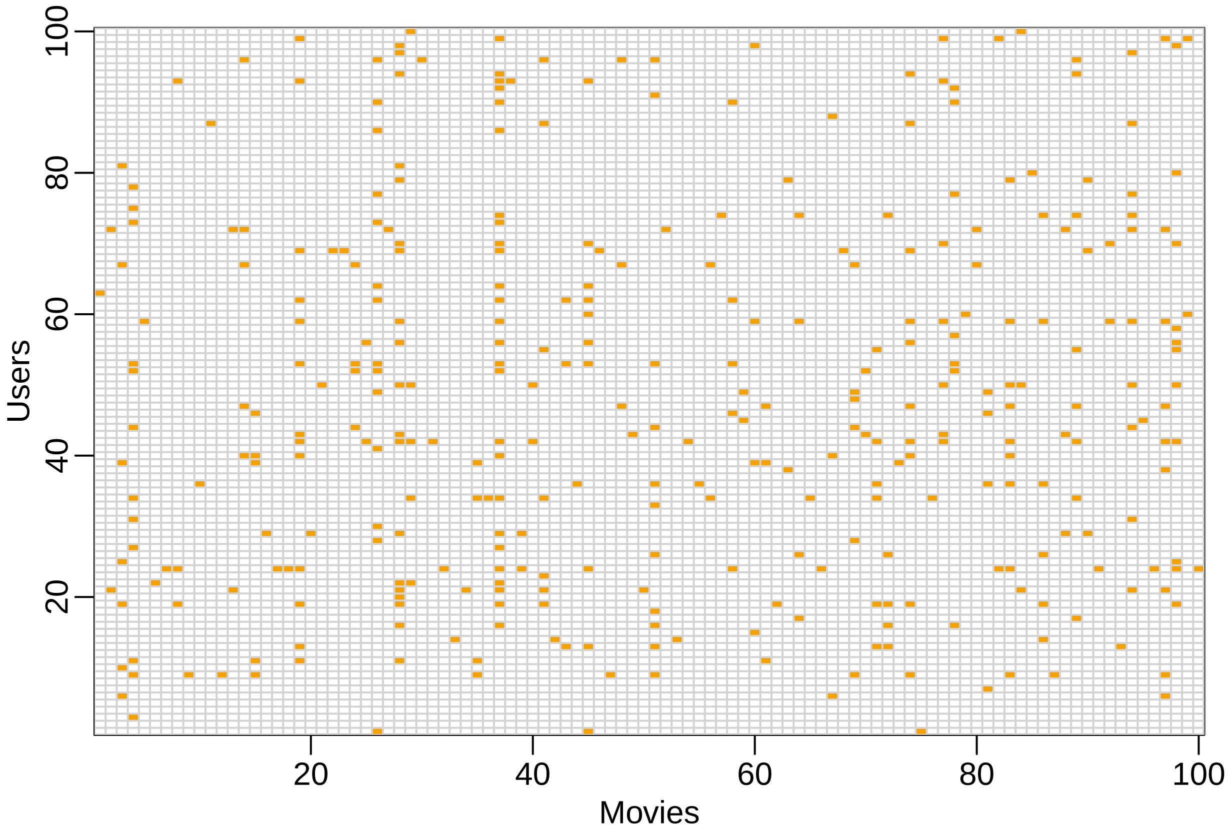
```
keep <- edx %>%
  count(movieId) %>%
  top_n(4, n) %>%
  .$movieId

tab <- edx %>%
  filter(movieId%in%keep) %>%
  filter(userId %in% c(13:20)) %>%
  select(userId, title, rating) %>%
  mutate(title = str_remove(title, ", The"),
         title = str_remove(title, ":.*")) %>%
  spread(title, rating)
tab %>% knitr::kable()
```

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Silence of the Lambs (1991)
13	NA	NA	4	NA
16	NA	3	NA	NA
17	NA	NA	NA	5
18	4.5	NA	5	NA
19	4.0	1	NA	NA

We can think of the task of a recommendation system as filling in the NAs in the table above.

We can get an indication of the sparsity of the data by looking at a matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating. As we can see here, the data is very sparse.

```
if(!require(rafalib)) install.packages("rafalib", repos = "http://cran.us.r-project.org")
library(rafalib)
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users") %>%
  abline(h=0:100+0.5, v=0:100+0.5, col = "lightgrey")
```



Data Preprocessing

```
# Convert timestamp column to date
edx <- edx %>% mutate(date = round_date(as_datetime(timestamp), unit = "week"))
validation <- validation %>% mutate(date = round_date(as_datetime(timestamp), unit = "week"))
edx <- edx %>% select(-timestamp)
validation <- validation %>% select(-timestamp)
```

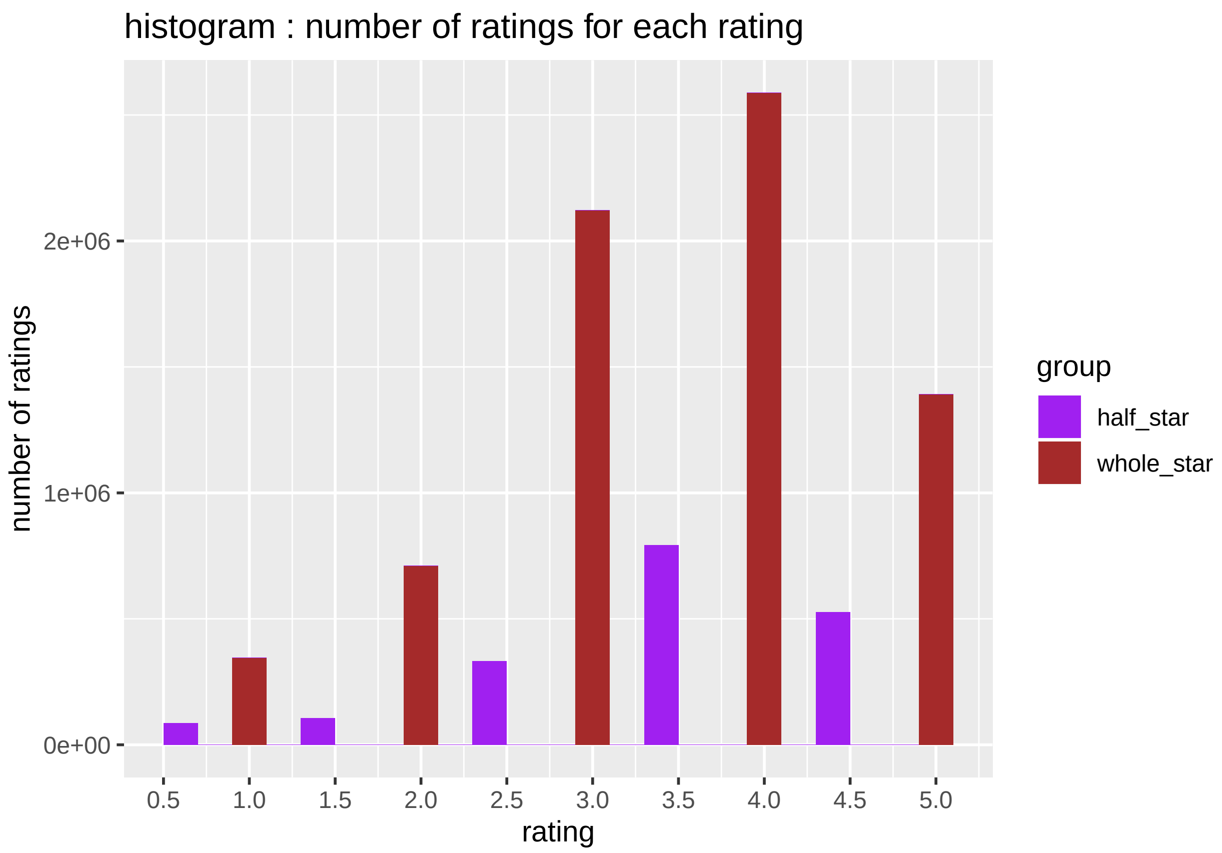
```
# Separate year of release from title and calculate age of movie

edx <- edx %>% mutate(release_year = as.numeric(str_sub(title,-5,-2)),
                      movie_title = str_replace(title, " \\(.*\\)", ""))
edx <- edx %>% mutate(movie_age = 2019 - release_year)
edx <-edx %>%select(-title)
validation <- validation %>% mutate(release_year = as.numeric(str_sub(title,-5,-2)),
                                    movie_title = str_replace(title, " \\(.*\\)", ""))
validation <- validation %>% mutate(movie_age = 2019 - release_year)
validation <- validation %>% select(-title)
```

Data Visualization

Let's look at some of the general properties of the data to better understand the challenges.

As a first step let us take a look at the distribution of ratings

```
#create a dataframe "explore_ratings" which contains half star and whole star ratings  from the edx set

group <-  ifelse((edx$rating == 1 |edx$rating == 2 | edx$rating == 3 |
                  edx$rating == 4 | edx$rating == 5) ,
                   "whole_star",
                   "half_star")

explore_ratings <- data.frame(edx$rating, group)

# histogram of ratings

ggplot(explore_ratings, aes(x= edx.rating, fill = group)) +
  geom_histogram( binwidth = 0.2) +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  scale_fill_manual(values = c("half_star"="purple", "whole_star"="brown")) +
  labs(x="rating", y="number of ratings") +
  ggtitle("histogram : number of ratings for each rating")
```



We notice the following:

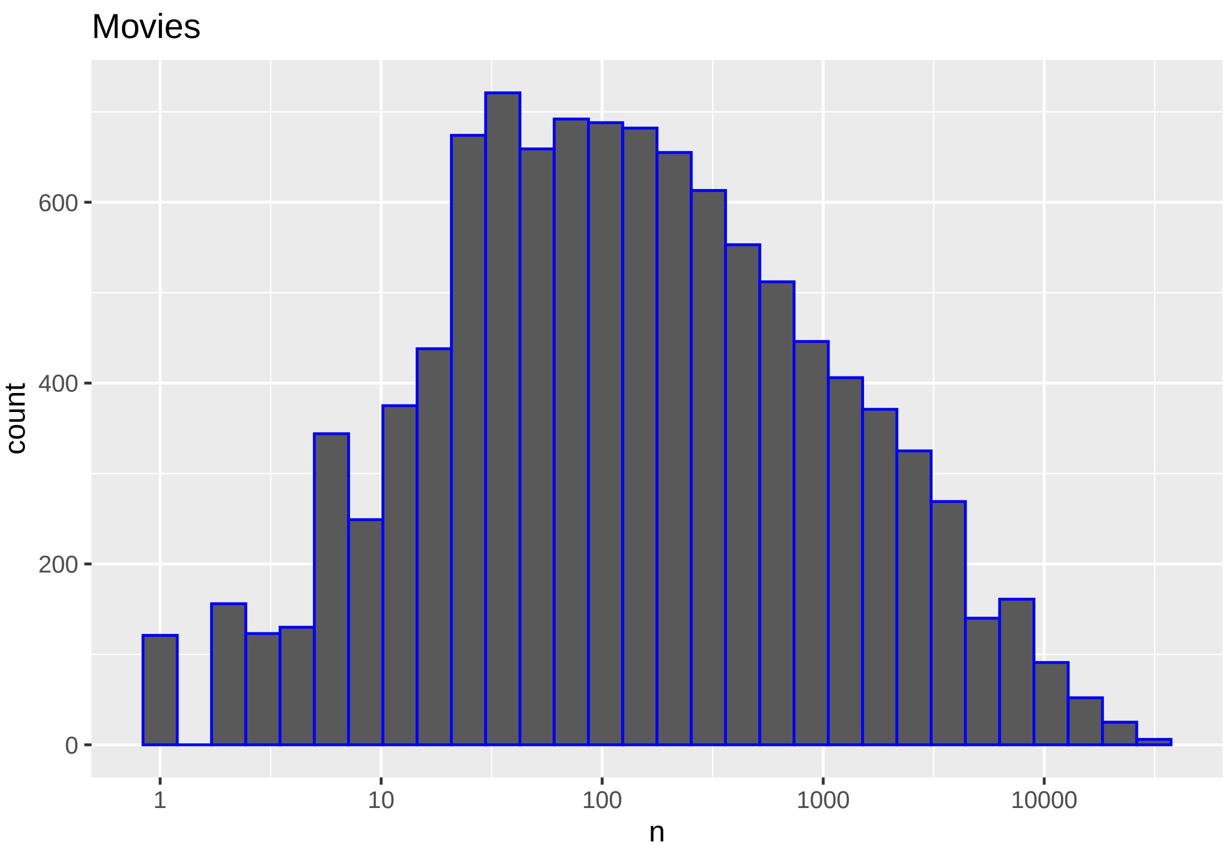
- No user gives zero as a rating
- The top 5 ratings from most to least are : 4, 3, 5, 3.5 and 2
- Half-star ratings are less common than whole-star ratings

Let us now look at the different factors that can gave a bearing on ratings.

Movies

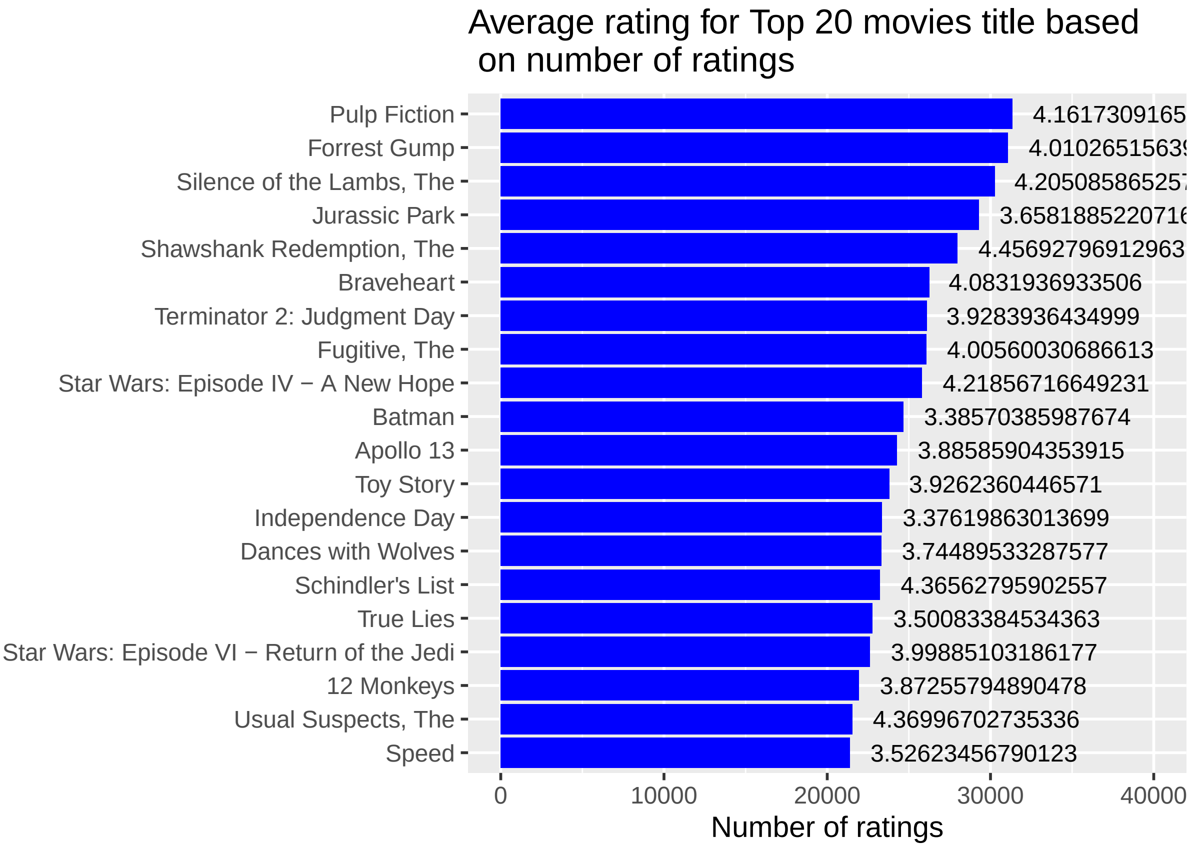
The first thing we notice is that some movies get rated more than others. Here is the distribution:

```
# plot count rating by movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue") +
  scale_x_log10() +
  ggtitle("Movies")
```



Further for the top 20 most rated movie titles we see the average ratings for them are also high. This means that more popular movies tend to be rated more.

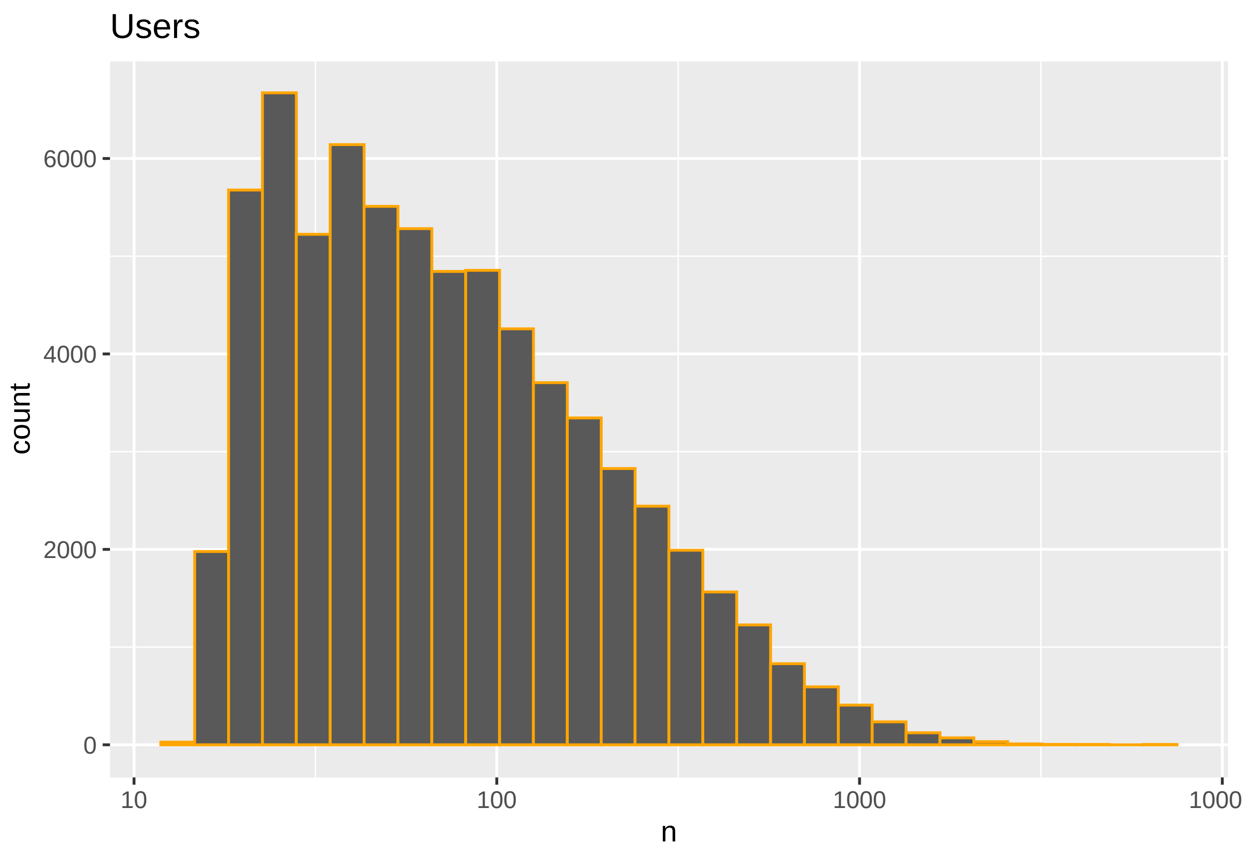
```
top_title <- edx %>%
  group_by(movie_title) %>%
  summarize(count=n(), avg_rating = mean(rating)) %>%
  top_n(20,count) %>%
  arrange(desc(count))

top_title %>%
  ggplot(aes(x=reorder(movie_title, count), y=count)) +
  geom_bar(stat='identity', fill="blue") + coord_flip(y=c(0, 40000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= avg_rating), hjust=-0.1, size=3) +
  labs(title="Average rating for Top 20 movies title based \n on number of ratings")
```



Users

Our second observation is that some users are more active than others at rating movies:

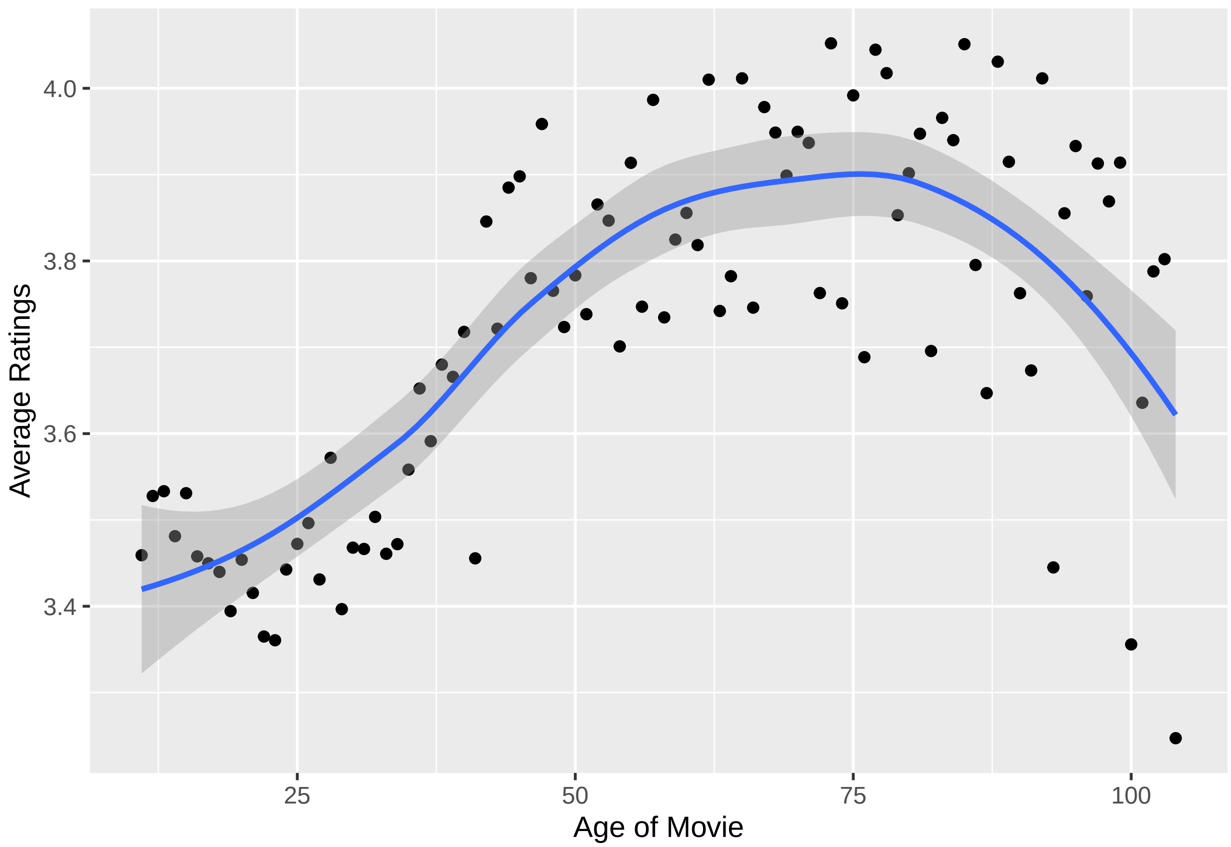
```
# plot count rating by user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "orange") +
  scale_x_log10() +
  ggtitle("Users")
```



Age of movie

```
edx %>% group_by(movie_age) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(movie_age, rating)) +
  xlab("Age of Movie") +
  ylab("Average Ratings") +
  geom_point() +
  geom_smooth()
```

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

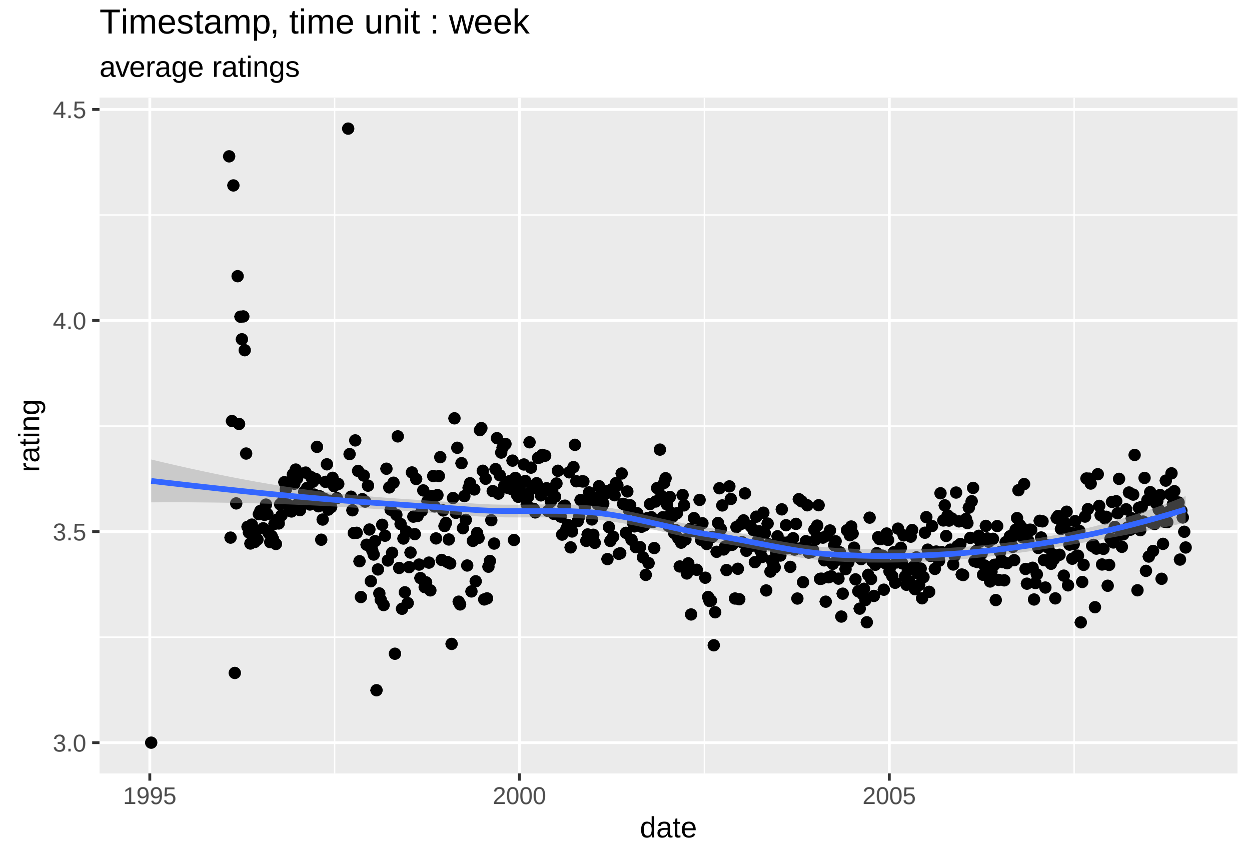


The general trend suggests that newer movies tend to be rated lower.

Time of rating

```
edx %>% group_by(date)%>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Timestamp, time unit : week")+
  labs(subtitle = "average ratings")
```

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



We notice that there is some evidence of a time effect on ratings but not much.

Genres

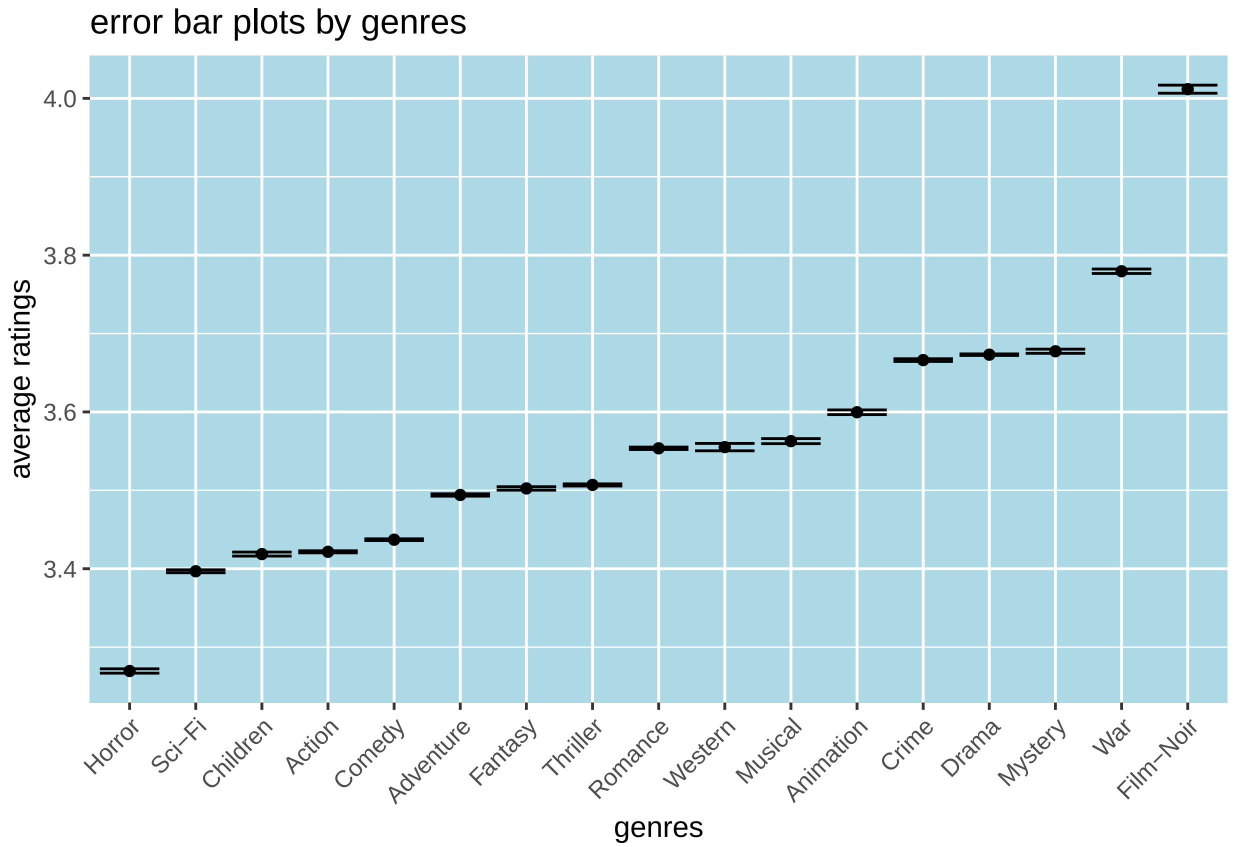
Let us first identify the top individual genres present

```
  split_edx <- edx %>% separate_rows(genres, sep = "\\|")

  top_genres <- split_edx %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))


top_genres %>% as_tibble()
```

```
## # A tibble: 20 x 2
##    genres                 count
##    <chr>                  <int>
##  1 Drama                3909401
##  2 Comedy               3541284
##  3 Action               2560649
##  4 Thriller             2325349
##  5 Adventure            1908692
##  6 Romance              1712232
##  7 Sci-Fi               1341750
##  8 Crime               1326917
##  9 Fantasy              925624
## 10 Children             737851
## 11 Horror               691407
## 12 Mystery              567865
## 13 War                  511330
## 14 Animation            467220
## 15 Musical              432960
## 16 Western              189234
## 17 Film-Noir            118394
## 18 Documentary           93252
## 19 IMAX                   8190
## 20 (no genres listed)        6
```

We notice that the “Drama” genre has the top number of movies ratings, followed by the “Comedy” and the “Action” genres.

```
# We compute the average and standard error for each genre and plot these as error bar plots for genres
  split_edx %>%
  group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  ylab("average ratings") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "error bar plots by genres" ) +
  theme(
    panel.background = element_rect(fill = "lightblue",
                                    colour = "lightblue",
                                    size = 0.5, linetype = "solid"),
    panel.grid.major = element_line(size = 0.5, linetype = 'solid',
                                    colour = "white"),
    panel.grid.minor = element_line(size = 0.25, linetype = 'solid',
                                    colour = "white")
  )
```



```r
rm(split_edx)
```

The generated plot shows evidence of a genre effect. However we will not be using this effect in subsequent modeling here.

Our modeling approach therefore will be to start with a simple baseline model and progressively add possible effects of some of the factors explored above to gauge model performance.

Results

In this section we build various models and discuss their performance.

A First Simple Model - Just The Average

Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. We can use a model based approach to answer this. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

with $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings:

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

```
## [1] 3.512464
```

If we predict all unknown ratings with $\hat{\mu}$ we obtain the following RMSE:

```
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060651
```

From looking at the distribution of ratings, we can visualize that this is the standard deviation of that distribution. We get a RMSE about 1.06.

As we will be comparing different approaches, we create a results table with this naive approach:

```
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.060651

Modeling movie effects

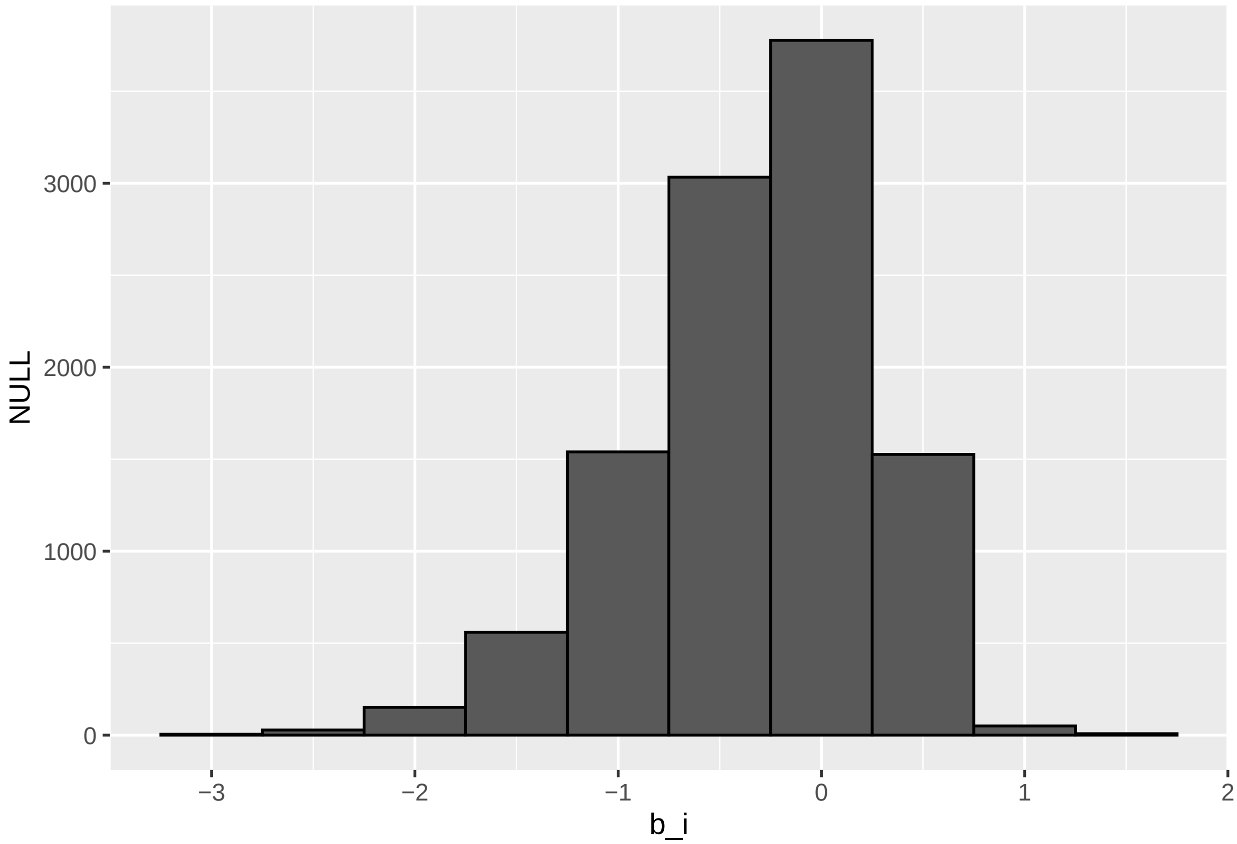
We know from experience that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. We can augment our previous model by adding the term b_i to represent average ranking for movie i :

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

Statistics textbooks refer to to the bs as effects. However, in the Netflix challenge papers, they refer to them as "bias", thus the b notation.

In this particular situation we know that the least square estimate $\hat{b_i}$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. So we can compute them this way:

```
mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



We can see that these estimates vary substantially.

Let's see how much our predictions improve once we use $\hat{y_{u,i}} = \hat{\mu} + \hat{b_i}$:

```
# create a results table with this and prior approaches
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

# create a results table with this and prior approach
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model on test set",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

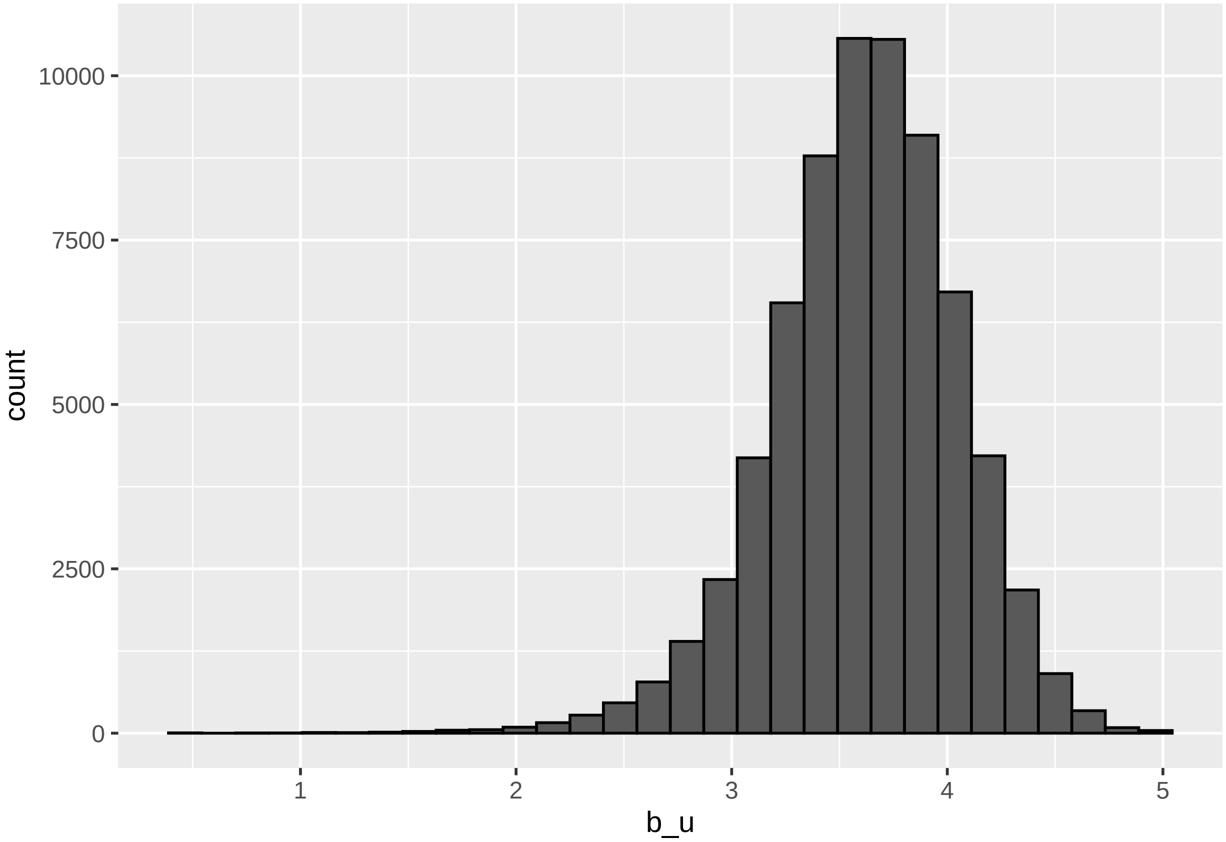
method	RMSE
Just the average	1.0606506
Movie Effect Model on test set	0.9437046

We can already see an improvement.

Modeling movie + user effects

Let's compute the average rating for user u for those that have rated over 100 movies:

```
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



There is variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model may be:

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

where bu is a user-specific effect. For reasons explained earlier, we will incorporate this effect by computing $\hat{\mu}$ and $\hat{b_i}$, and estimating $\hat{b_u}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i}$

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

We can now construct predictors and see how much the RMSE improves:

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# create a results table with this and prior approaches
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User Effects Model on test set",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model on test set	0.9437046
Movie and User Effects Model on test set	0.8655329

We can see further improvement in our predictions by incorporating the user-specific effect.

Regularization

Regularization permits us to penalize large estimates that are formed using small sample sizes.

These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

Let's look at the top 10 worst and best movies based on $\hat{b}_i$. First, let's create a database that connects movieId to movie title:

```
movie_titles <- edx %>%
  select(movieId, movie_title) %>%
  distinct()
```

The 10 best movies according to our estimate:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(movie_title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

movie_title	b_i
Hellhounds on My Trail	1.487536
Satan's Tango	1.487536
Shadows of Forgotten Ancestors	1.487536
Fighting Elegy	1.487536
Sun Alley	1.487536
Blue Light, The	1.487536
Constantine's Sword	1.487536

movie_title	b_i
Human Condition II, The	1.320869
Who's Singin' Over There?	1.237536
Human Condition III, The	1.237536

And the 10 worst:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(movie_title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

movie_title	b_i
Besotted	-3.012464
Hi-Line, The	-3.012464
Grief	-3.012464
Accused	-3.012464
War of the Worlds 2: The Next Wave	-2.762464
SuperBabies: Baby Geniuses 2	-2.698905
Hip Hop Witch, Da	-2.679131
From Justin to Kelly	-2.583171
Disaster Movie	-2.528593
Stacy's Knights	-2.512464

These movies seem obscure. Let us see how often these 'best' and worst' movies have been rated

```
edx %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(movie_title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

movie_title	b_i	n
Hellhounds on My Trail	1.487536	1
Satan's Tango	1.487536	2
Shadows of Forgotten Ancestors	1.487536	1
Fighting Elegy	1.487536	1
Sun Alley	1.487536	1
Blue Light, The	1.487536	1
Constantine's Sword	1.487536	1
Human Condition II, The	1.320869	3
Who's Singin' Over There?	1.237536	4
Human Condition III, The	1.237536	4

```
edx %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(movie_title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

movie_title	b_i	n
Besotted	-3.012464	2
Hi-Line, The	-3.012464	1
Grief	-3.012464	1
Accused	-3.012464	1
War of the Worlds 2: The Next Wave	-2.762464	2
SuperBabies: Baby Geniuses 2	-2.698905	59
Hip Hop Witch, Da	-2.679131	12
From Justin to Kelly	-2.583171	198
Disaster Movie	-2.528593	31
Stacy's Knights	-2.512464	1

The supposed "best" and "worst" movies were rated by very few users, in most cases just 1. These movies were mostly obscure ones. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of b_i negative or positive, are more likely.

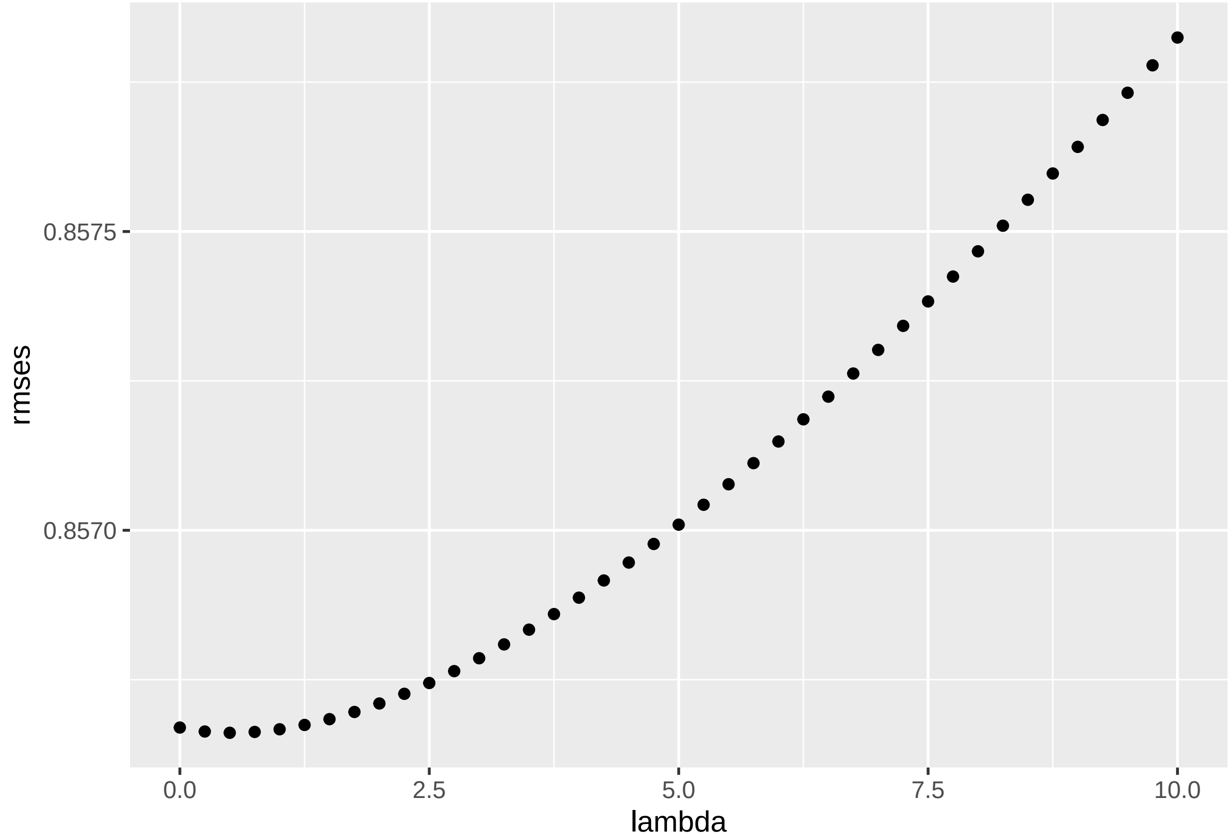
These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

The general idea behind regularization is to constrain the total variability of the effect sizes. Specifically, instead of minimizing the least square equation, we minimize an equation that adds a penalty for large movie and user effects for small sample sizes, effectively shrinking the estimates :

$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

To pick an optimal value for λ, the penalty term, we use cross-validation.

```
# use cross-validation to pick a lambda:

lambda <- seq(0, 10, 0.25)

rmses <- sapply(lambda, function(l){
  mu <- mean(edx$rating)

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

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

  predicted_ratings <-
    edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(edx$rating, predicted_ratings))
})

qplot(lambda, rmses)
```



```
# pick lambda with minimun rmse
lambda <- lambda[which.min(rmses)]
# print lambda
lambda
```

```
## [1] 0.5
```

We use the validation set for final assessment of the regularized model

```r
# compute movie effect with regularization on train set
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# compute user effect with regularization on train set
b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# # compute predicted values on test set
predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

# create a results table with this and prior approaches
model_3_rmse <- RMSE(validation$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Reg Movie and User Effect Model on test set",
                                 RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model on test set	0.9437046
Movie and User Effects Model on test set	0.8655329
Reg Movie and User Effect Model on test set	0.8654111

It appears that regularization helps improve model predictions marginally.

Modeling movie + users + time effect

Here we incorporate the possible effect of the time of rating in addition to movie and user effects.

```r
# Calculate time effects ( b_t) using the training set
temp_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))

# predicted ratings
  predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(temp_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  .$pred

model_4_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie, User and Time Effects Model on test set",
                                     RMSE = model_4_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model on test set	0.9437046
Movie and User Effects Model on test set	0.8655329
Reg Movie and User Effect Model on test set	0.8654111
Movie, User and Time Effects Model on test set	0.8654570

We recognise that there is no improvement in RMSE over the regularized movie + user effect model.

Modeling movie + user + movie_age effect

Here we include the possible effect of the age of a movie on ratings in addition to movie and user effects.

```
# Calculate age effects ( b_a) using the training set
age_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(movie_age) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u))

# predicted ratings
  predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(age_avgs, by='movie_age') %>%
  mutate(pred = mu + b_i + b_u + b_a) %>%
  .$pred

model_5_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie, User and Age Effects Model on test set",
                                     RMSE = model_5_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model on test set	0.9437046
Movie and User Effects Model on test set	0.8655329
Reg Movie and User Effect Model on test set	0.8654111
Movie, User and Time Effects Model on test set	0.8654570
Movie, User and Age Effects Model on test set	0.8652156

There is a marginal improvement in RMSE by incorporating the age of a movie.

Matrix Factorization with stochastic gradient descent

Matrix factorization is a widely used concept in machine learning. It is very much related to factor analysis, singular value decomposition (SVD) and principal component analysis (PCA). Here we describe the concept in the context of movie recommendation systems.

We have described how the model:

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

accounts for movie to movie differences through the bi and user to user differences through the b_u. But this model leaves out an important source of variation related to the fact that groups of movies have similar rating patterns and groups of users have similar rating patterns as well. We will discover these patterns by studying the residuals:

$$r_{u,i} = y_{u,i} - \hat{b_i} - \hat{b_u}$$

If the model above explains all the signals, and the ϵ are just noise, then the residuals for different movies should be independent of each other. However it can be shown by examining correlation plots of residuals, that there is visible correlation between the movie 'The Godfather' and 'The GodFather part II' indicating that users that liked a gangster movie The Godfather more than what the model expects them to, based on the movie and user effects, also liked another gangster movie, The Godfather II more than expected. Similar patterns can be seen with romantic movies like 'You've Got Mail' and 'Sleepless In Seattle'. These results tell us that there is structure in the data beyond the obvious movie and user effects. This structure can be modeled by :

$$r_{u,i} \approx p_u q_i$$

This implies that we can explain more variability by modifying our previous model for movie recommendations to:

$$Y_{u,i} = \mu + b_i + b_u + p_u q_i + \epsilon_{u,i}$$

$r_{u,i}$ is the residual for the uth user and the ith movie and belongs to matrix $R_{u \times i}$. pu is the uth column of a matrix $P_{k \times u}$ and q_i is the ith column of a matrix $Q_{k \times i}$. Matrix P represents latent factors of users. So, each k-elements column of matrix P represents each user. Each k-elements column of matrix Q represents each item . So, to find rating for item i by user u we simply need to compute two vectors: $P'_{,u} \times Q_{,i}$. Further descriptions of this technique and the recosystem package are available here.

To perform our recommender system using parallel Matrix factorization with stochastic gradient descent, we follow the different steps:

i. We created an identical copy of edx and validation set (edx.copy and valid.copy) , selecting only userId, movieId and rating columns. With the recosystem package, the data file for training set needs to be arranged in sparse matrix triplet form, i.e., each line in the file contains three numbers "user_index", "item_index", "rating".

ii. No RAM problem : Unlike most other R packages for statistical modeling that store the whole dataset and model object in memory, recosystem can significantly reduce memory use, for instance the constructed model that contains information for prediction can be stored in the hard disk, and output result can also be directly written into a file rather than be kept in memory. That is why we simply use our whole edx.copy as train set (9,000,055 occurences) and valid.copy as validation set (999,999 occurences).

iii. Finally, we create a model object by calling Reco() , call the tune() method to select best tuning parameters along a set of candidate values , train the model by calling the train() method and use the predict() method to compute predicted values.

```
#Matrix Factorization with parallel stochastic gradient descent

#Create a copy of training(edx) and validation sets and retain only userId, movieId and rating features

edx.copy <-  edx %>%
            select(c("userId","movieId","rating"))

names(edx.copy) <- c("user", "movie", "rating")


valid.copy <-  validation %>%
  select(c("userId","movieId","rating"))

names(valid.copy) <- c("user", "movie", "rating")

#as matrix
edx.copy <- as.matrix(edx.copy)
valid.copy <- as.matrix(valid.copy)


#write edx.copy and valid.copy tables on disk
write.table(edx.copy , file = "trainset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)
write.table(valid.copy, file = "validset.txt" , sep = " " , row.names = FALSE, col.names = FALSE)

rm(edx.copy, valid.copy)
```

```
#  data_file(): Specifies a data set from a file in the hard disk.
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: recosystem
```

```
set.seed(123) # This is a randomized algorithm
train_set <- data_file("trainset.txt")
valid_set <- data_file("validset.txt")


#Next step is to build Recommender object
r = Reco()

# Matrix Factorization :  tuning training set

opts = r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                     costp_l1 = 0, costq_l1 = 0,
                                     nthread = 1, niter = 10))
#opts
```

```
# Matrix Factorization : trains the recommender model

r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9731  1.2056e+007
##    1       0.8706  9.8639e+006
##    2       0.8379  9.1560e+006
##    3       0.8171  8.7472e+006
##    4       0.8020  8.4769e+006
##    5       0.7902  8.2797e+006
##    6       0.7802  8.1297e+006
##    7       0.7719  8.0036e+006
##    8       0.7649  7.9073e+006
##    9       0.7588  7.8229e+006
##   10       0.7537  7.7572e+006
##   11       0.7490  7.7000e+006
##   12       0.7447  7.6480e+006
##   13       0.7410  7.6045e+006
##   14       0.7376  7.5672e+006
##   15       0.7344  7.5333e+006
##   16       0.7316  7.5026e+006
##   17       0.7288  7.4705e+006
##   18       0.7264  7.4466e+006
##   19       0.7242  7.4255e+006
```

```r
#Making prediction on validation set and calculating RMSE:

pred_file = tempfile()

r$predict(valid_set, out_file(pred_file))
```

```
## prediction output generated at C:\Users\sauga\AppData\Local\Temp\RtmpIPtxwB\file3fe46b86323f
```

```r
#valid_set
scores_real <- read.table("validset.txt", header = FALSE, sep = " ")$V3
scores_pred <- scan(pred_file)


model_6_rmse <- RMSE(scores_pred, scores_real)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Matrix Factorization Model on test set",
                                     RMSE = model_6_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0606506
Movie Effect Model on test set	0.9437046
Movie and User Effects Model on test set	0.8655329
Reg Movie and User Effect Model on test set	0.8654111
Movie, User and Time Effects Model on test set	0.8654570
Movie, User and Age Effects Model on test set	0.8652156
Matrix Factorization Model on test set	0.7829723

The Matrix Factorization method yields the best model performance on the test set. This is expected as this approach decomposes users and movies into a set of latent factors that can lead to better predictions. In fact, matrix factorization methods were probably the most important class of techniques for winning the Netflix Prize.

Final RMSE reported : **0.7829**

Conclusion

In this project we have tried to predict user recommendations for the 10M version of the MovieLens Data using various methods as outlined above. Using the provided training(edx) and validation sets, we successively trained different linear regression models and followed it up with a matrix factorization method. The model evaluation performance through RMSE(root mean squared error) showed that the matrix factorization with stochastic gradient descent method worked best with the present dataset.

Future work might involve trying other recommender engines such as **recommenderlab** and **SlopeOne** and also trying ensemble methods such as **Gradient Boosted Decision Trees** to see if it is possible to achieve further improvement in model performance.