CapstoneProject

Hans M. Rupp

26 5 2022

Introduction

The aim of this project is to predict ratings for movies in the open source MovieLens dataset. This is inspired by a 2006 competition by Netflix to improve its recommondation system. This project is part of the capstone project of HarvardX PH125.9x Data Science. It is based on https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems The main benchmark is achieving a RMSE < 0.86490 on the validation data set.

Recommender systems can divided into two categories [1]

The **content filtering** approach creates a profile for each user or product. Some profiles are manually curated.

Collaborative filtering relies only on past user behavior.

The two primary areas of collaborative filtering are the neighborhood methods and latent factor models A product's neighbors are other products that get similar reatings by the same user. Latent factor models explain the ratings by factors characterizing both items and users. We will use a latent factor model here.

Loading the data

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("https://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 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
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)
```

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

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Data Analysis and Cleanup

The data structure of the dataset:

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

The edx dataset contains 9000055 observations (rows) with 6 features (columns). It will be used for training the model. The validation dataset contains 999999 observations (rows) with 6 features (columns). It will be used for validating the model

Checking for missing values

```
nas <- function(x) { any(is.na(x)) }
edx %>% summarise_all(nas)
```

```
##   userId movieId rating timestamp title genres
## 1  FALSE   FALSE  FALSE     FALSE FALSE  FALSE
```

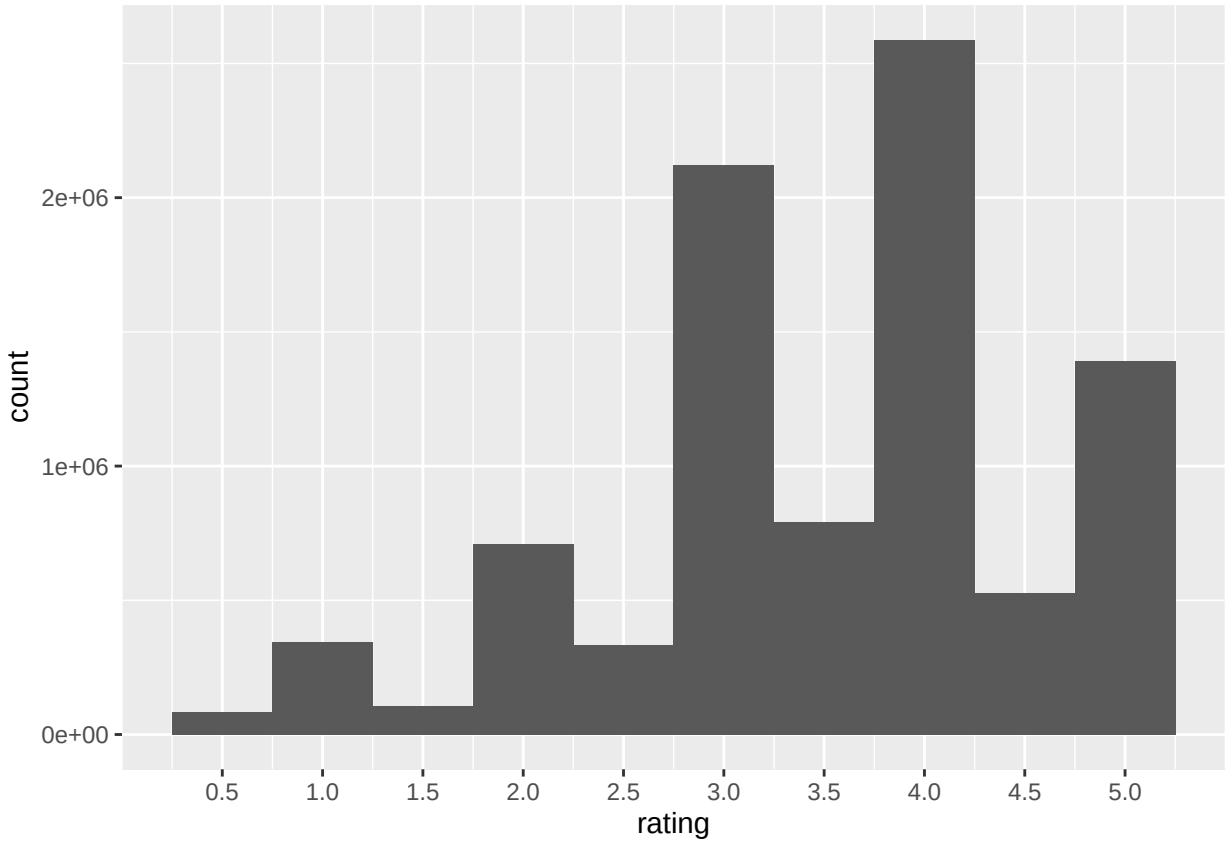
```
validation %>% summarise_all(nas)
```

```
##   userId movieId rating timestamp title genres
## 1  FALSE   FALSE  FALSE     FALSE FALSE  FALSE
```

There are no columns with NAs in the edx or the validation data. So no clean-up is necessary

Overall distribution of the ratings

```
edx %>% ggplot(aes(x=rating)) +
  geom_histogram(binwidth = 0.5) +
  scale_x_continuous(breaks = seq(min(edx$rating), max(edx$rating), by = 0.5))
```



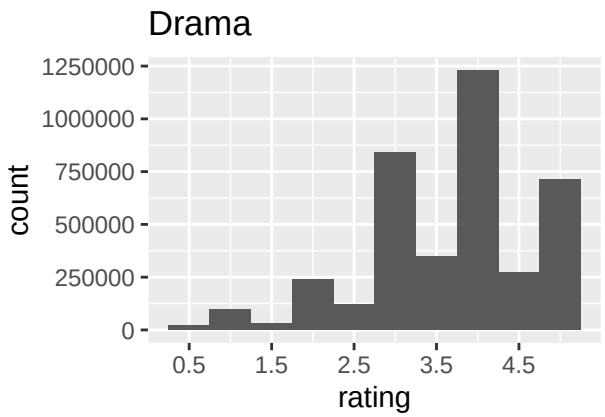
The **different genres** are

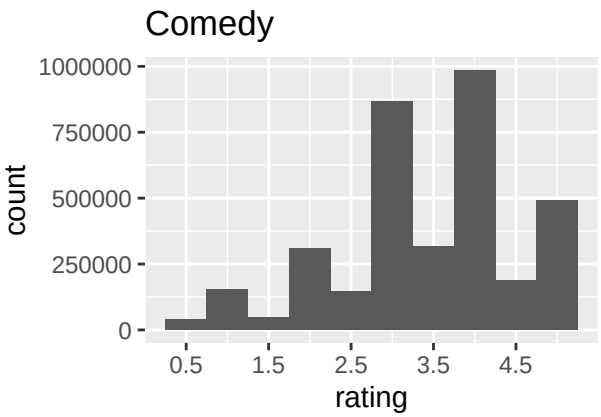
```
dist_genres <- edx %>% distinct(genres)
dg <- as.vector(str_split(dist_genres$genres, "\\|", simplify = T))
unique(dg[dg != ""])
```

```
##  [1] "Comedy"             "Action"             "Children"
##  [4] "Adventure"          "Animation"          "Drama"
##  [7] "Crime"              "Sci-Fi"             "Horror"
## [10] "Thriller"           "Film-Noir"          "Mystery"
## [13] "Western"            "Documentary"        "Romance"
## [16] "Fantasy"            "Musical"            "War"
## [19] "IMAX"               "(no genres listed)"
```

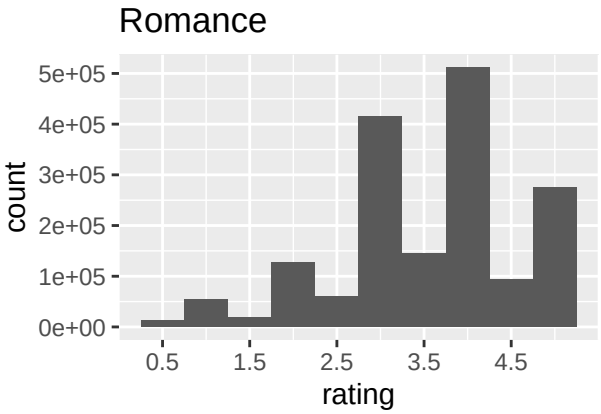
Distribution of ratings for different genres

```
hist_drama <- edx %>% filter(str_detect(genres, "Drama")) %>%
  ggplot(aes(x=rating)) + geom_histogram(binwidth = 0.5) +
  scale_x_continuous(breaks = seq(0.5, 5, by = 1)) + labs(title="Drama")

hist_comedy <- edx %>% filter(str_detect(genres, "Comedy"))  %>%
  ggplot(aes(x=rating)) + geom_histogram(binwidth = 0.5) +
  scale_x_continuous(breaks = seq(0.5, 5, by = 1)) + labs(title="Comedy")

hist_thriller <- edx %>% filter(str_detect(genres, "Thriller"))  %>%
  ggplot(aes(x=rating)) + geom_histogram(binwidth = 0.5) +
  scale_x_continuous(breaks = seq(0.5, 5, by = 1)) + labs(title="Thriller")

hist_romance <- edx %>% filter(str_detect(genres, "Romance"))  %>%
  ggplot(aes(x=rating)) + geom_histogram(binwidth = 0.5) +
  scale_x_continuous(breaks = seq(0.5, 5, by = 1)) + labs(title="Romance")

grid.arrange(hist_drama, hist_comedy, hist_thriller,hist_romance, ncol = 2)
```


Drama
1250000
1000000
750000
500000
250000
0
count
0.5
1.5
2.5
3.5
4.5
rating

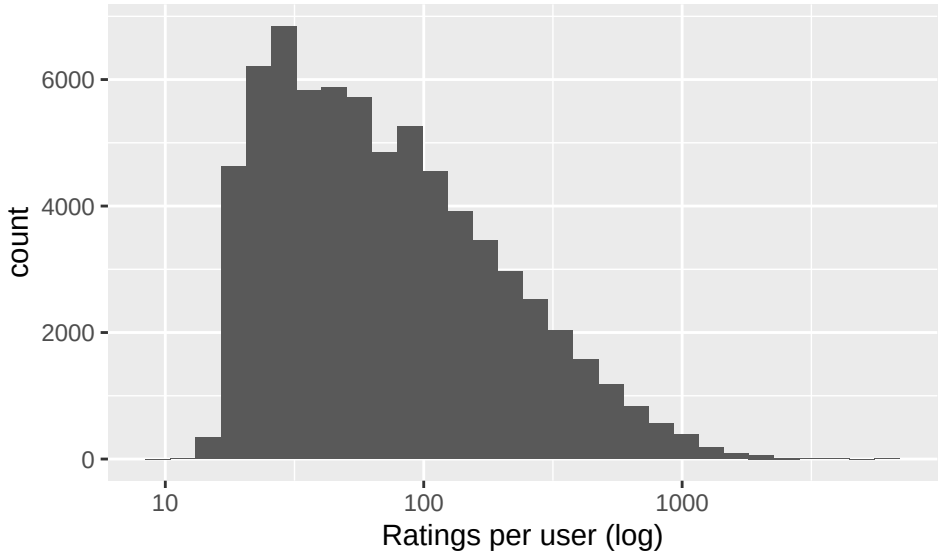

Comedy
1000000
750000
500000
250000
0
count
0.5
1.5
2.5
3.5
4.5
rating


Thriller
6e+05
4e+05
2e+05
0e+00
count
0.5
1.5
2.5
3.5
4.5
rating


Romance
5e+05
4e+05
3e+05
2e+05
1e+05
0e+00
count
0.5
1.5
2.5
3.5
4.5
rating

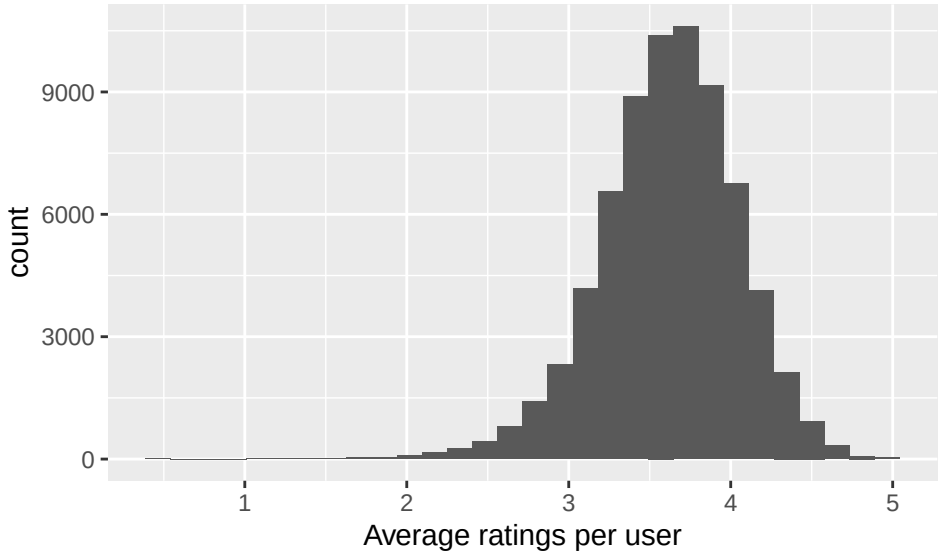
Different **users** have widely different **numbers of ratings**

```
edx %>% count(userId) %>% ggplot(aes(x=n)) + geom_histogram() + scale_x_log10() +
  xlab("Ratings per user (log)")
```



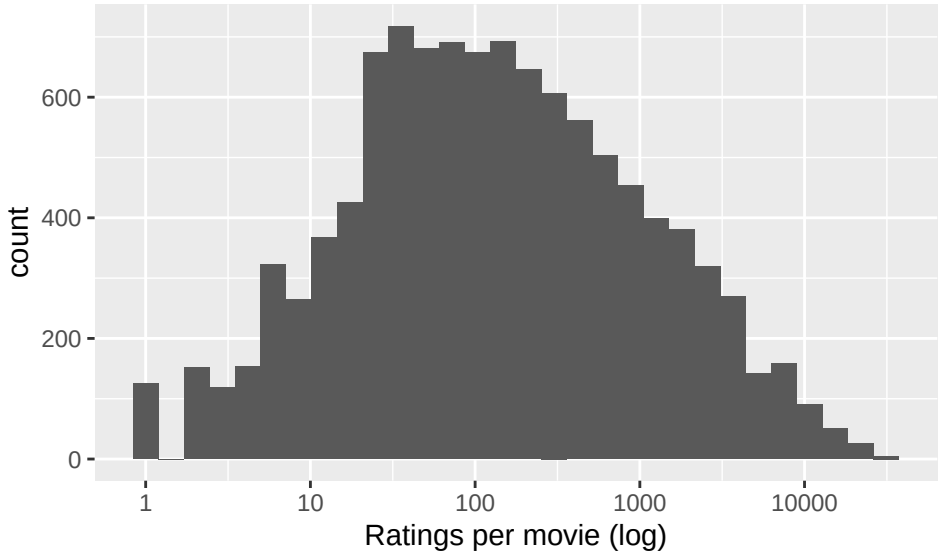
Different **users** have different **rating means**

```
edx %>% group_by(userId) %>% summarise(user_mean=mean(rating)) %>% ggplot(aes(x=user_mean)) +
  geom_histogram()  + xlab("Average ratings per user")
```



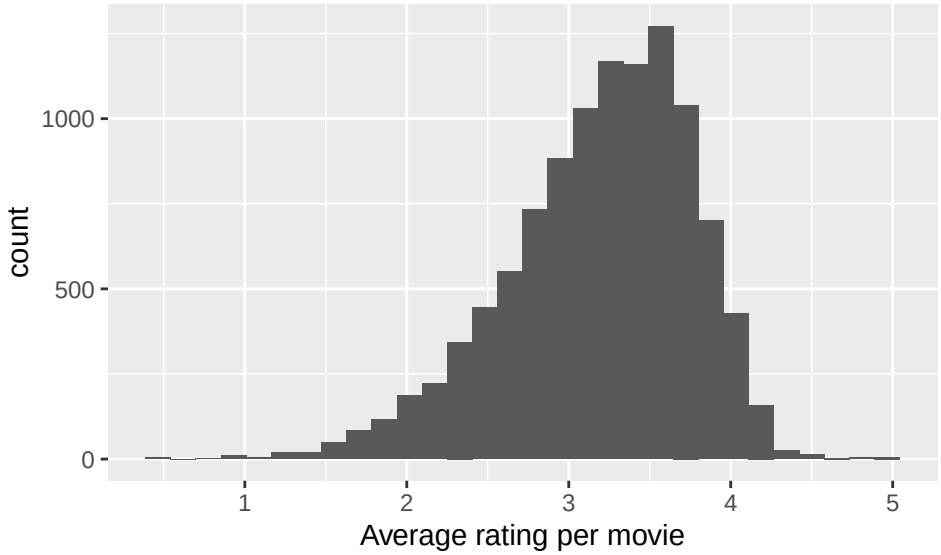
Different **movies** have different **numbers of ratings**

```
edx %>% group_by(movieId) %>% summarise(n=n()) %>% ggplot(aes(x=n)) + geom_histogram() +
  scale_x_log10() + xlab("Ratings per movie (log)")
```



Different **movies** have different **rating means**

```
edx %>% group_by(movieId) %>% summarise(movie_mean = mean(rating)) %>% ggplot(aes(x=movie_mean)) +
  geom_histogram() + xlab("Average rating per movie")
```



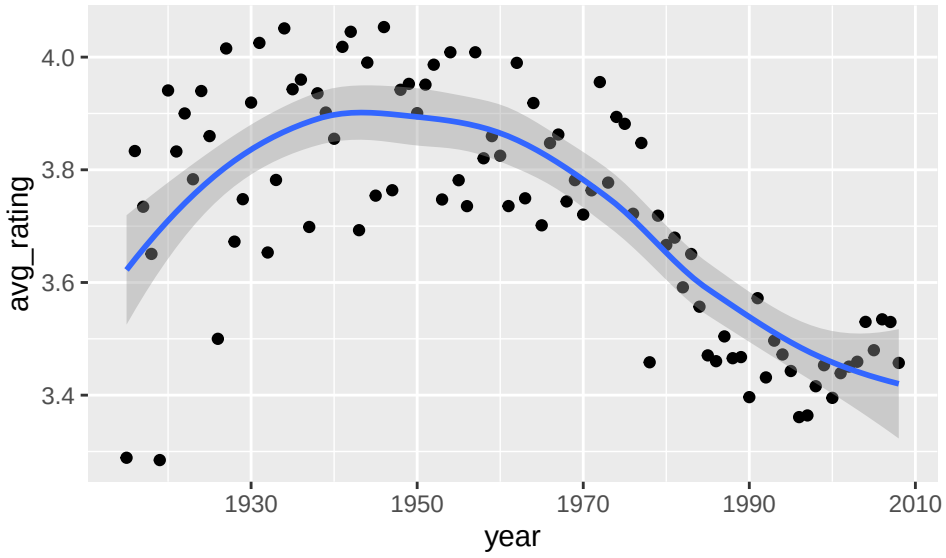
Release year has some influence on **average rating**

Extract the release date from the title

```
release_year_edx <- as.numeric(str_sub(edx$title, start=-5, end=-2))
edx <- edx %>% mutate(year=release_year_edx)
validation <- validation %>% mutate(year=as.numeric(str_sub(title, start=-5, end=-2)))
```

Plot of the average rating for each release year

```
edx %>% group_by(year) %>% summarise(avg_rating = mean(rating)) %>%
  ggplot(aes(x=year, y=avg_rating)) +
  geom_point() + geom_smooth()
```



Matrix Factorization

We will use Matrix Factorization to build a model to predict movie ratings

movie i user u year y genre g

RMSE

The Root Mean Square Error will be used as the Loss function

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

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

Naive model: simply using the mean ratings

This model assumes the same rating μ for all movies i and users u with an error $\epsilon_{u,i}$
$Y_{u,i} = \mu + \epsilon_{u,i}$

```
mu_rating <- mean(edx$rating)
rmse_mean <- RMSE(validation$rating, mu_rating)
rmse_mean
```

```
## [1] 1.061202
```

This gives a RMSE of 1.0612018

We build a table of the RMSEs for the different models:

```
models <- tibble(model="Mean", rmse=rmse_mean)
```

Effect of different movies

As we saw above different movies are rated differently.

We can add the term b_i for the average ranking for movie i

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

Theoretically we could calculate b_i using linear regression. But with this amount of data that would be very costly.

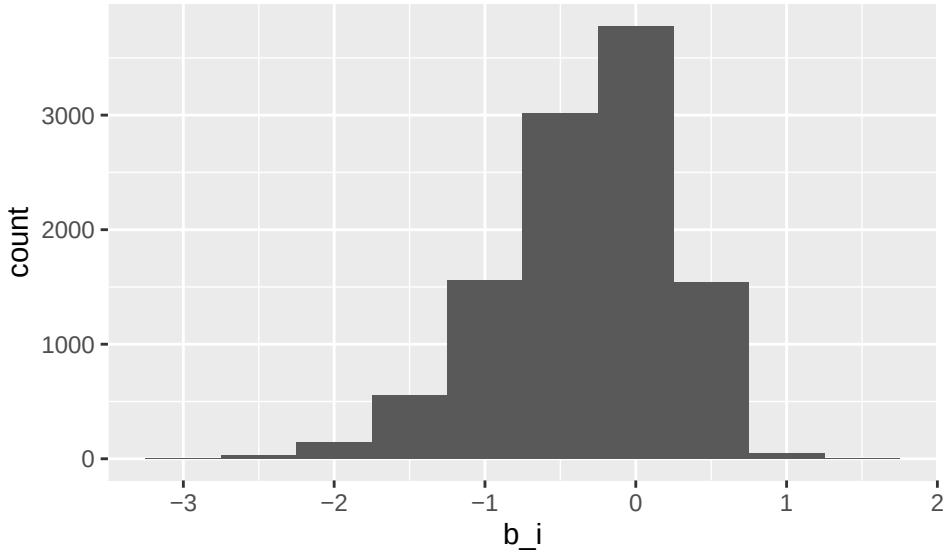
Instead we can just use $b_i = Y_{u,i} - \mu$

Calculate the average rating b_i for each movie minus the overall average rating mu_rating = 3.5124652:

```
movie_avgs <- edx %>% group_by(movieId) %>% summarise(b_i = mean(rating - mu_rating))
```

Plot them:

```
movie_avgs %>% ggplot(aes(b_i)) + geom_histogram(bins = 10)
```



Join the movie_avg data.frame to the edx (training) and validation set:

```
edx_movies <- edx %>% left_join(movie_avgs, by="movieId")
validation_movies <- validation %>% left_join(movie_avgs, by="movieId")
```

Calculating the predicted ratings $Y_{u,i} = \mu + b_i$

```
predicted_ratings_movies <- mu_rating + validation_movies$b_i
```

Calculating the new RMSE

```
rmse_movies <- RMSE(predicted_ratings_movies, validation$rating)
models <- rbind(models, c("movies", rmse_movies))
knitr::kable(models)
```

model	rmse
Mean	1.06120181029262
movies	0.943908662806309

The RMSE has improved to 0.943908662806309

Effect of different users

As we saw above different users have different rating means.

We can add the term b_u for the average rating for user u

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

Calculate the average rating b_u for each movie minus the overall average rating mu_rating and b_i:

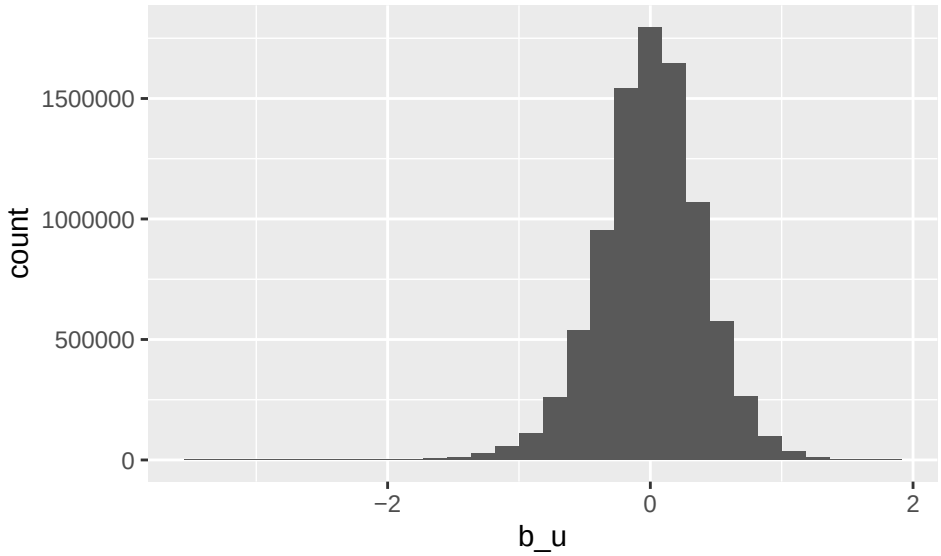
```
user_avg <- edx_movies %>% group_by(userId) %>% summarise(b_u = mean(rating - mu_rating - b_i))
```

Join the user_avg data.frame to the edx_movies (training) and validation_movies set:

```
edx_movies_users <- edx_movies %>% left_join(user_avg, by="userId")
validation_movies_users <- validation_movies %>% left_join(user_avg, by="userId")
```

Plot the distribution of user effects b_u

```
edx_movies_users %>% ggplot(aes(x=b_u)) + geom_histogram()
```



Calculating the predicted ratings $Y_{u,i} = \mu + b_i + b_u$

```
predicted_ratings_movies_users <- mu_rating + validation_movies_users$b_i +
  validation_movies_users$b_u
rmse_movies_users <- RMSE(predicted_ratings_movies_users, validation$rating)
models <- rbind(models, c("movies users", rmse_movies_users))
knitr::kable(models)
```

model	rmse
Mean	1.06120181029262
movies	0.943908662806309
movies users	0.865348824577316

The RMSE has improved to 0.865348824577316

Effect of the release year

We can add the term b_y for the average rating for movies released in year y

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

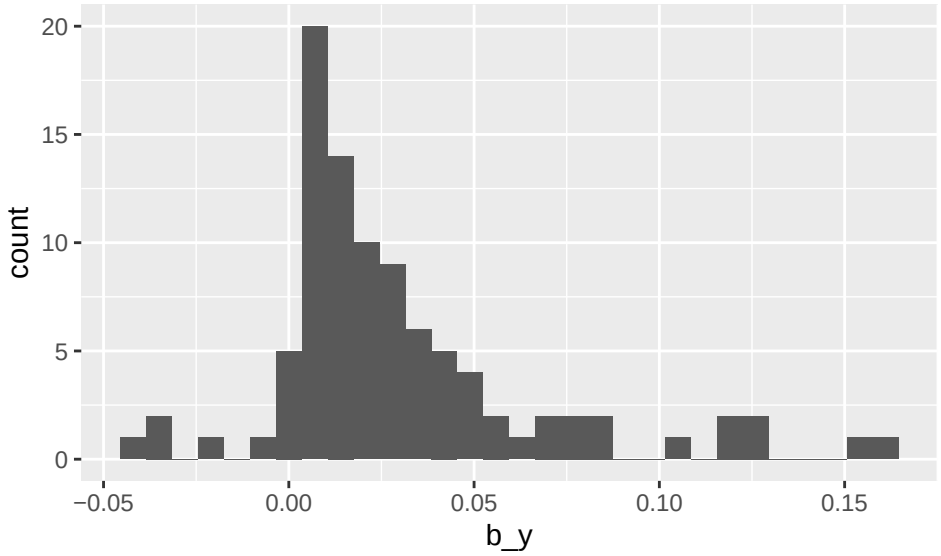
Calculate the average rating b_u for each movie minus the overall average rating mu_rating and b_i and b_u:

```
year_avg <- edx_movies_users %>% group_by(year) %>% summarise(b_y = mean(rating - mu_rating - b_i - b_u
```

Plot the effect of the release year b_y

```
year_avg %>% ggplot(aes(x=b_y)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Join the year_avg data.frame to the edx_movies_users (training) and validation_movies_users set:

```
edx_movies_users_years <- edx_movies_users %>% left_join(year_avg, by="year")
validation_movies_users_years <- validation_movies_users %>% left_join(year_avg, by="year")
```

Calculating the predicted ratings $Y_{u,i} = \mu + b_i + b_u + b_y$

```
predicted_ratings_movies_users_years <- mu_rating + validation_movies_users_years$b_i + validation_movi

rmse_movies_users_years <- RMSE(predicted_ratings_movies_users_years, validation$rating)
rmse_movies_users_years
```

```
## [1] 0.8650043
```

```
models <- rbind(models, c("movies users years", rmse_movies_users_years))
knitr::kable(models)
```

model	rmse
Mean	1.06120181029262

model	rmse
movies	0.943908662806309
movies users	0.865348824577316
movies users years	0.865004340330318

The RMSE has only marginally improved to 0.865004340330318

Effect of the genre

We can add the term b_g for the average rating for movies of genre g

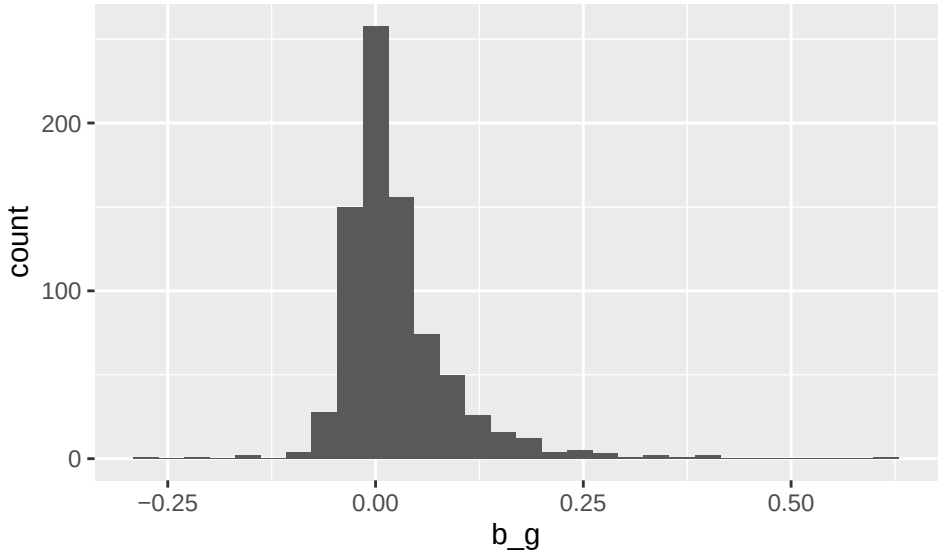
$Y_{u,i,y,g} = \mu + b_i + b_u + b_y + b_g + \epsilon_{u,i}$

Calculate the average rating b_g for each movie minus the overall average rating mu_rating and b_i and b_u and b_y:

```
genre_avg <- edx_movies_users_years %>%
  group_by(genres) %>% summarise(b_g = mean(rating - mu_rating - b_i - b_u - b_y))
```

Plot the effect of the genre b_g

```
genre_avg %>% ggplot(aes(x=b_g)) + geom_histogram()
```



Join the genre_avg data.frame to the edx_movies_users_year (training) and validation_movies_users_year set:

```
edx_movies_users_years_genres <- edx_movies_users_years %>%
  left_join(genre_avg, by="genres")
validation_movies_users_years_genres <- validation_movies_users_years %>%
  left_join(genre_avg, by="genres")
```

Calculating the predicted ratings $Y_{u,i} = \mu + b_i + b_u + b_y + b_g$ and the RMSE

```
predicted_ratings_movies_users_years_genres <- mu_rating +
  validation_movies_users_years_genres$b_i + validation_movies_users_years_genres$b_u  +
  validation_movies_users_years_genres$b_y +  validation_movies_users_years_genres$b_g

rmse_movies_users_years_genres <- RMSE(predicted_ratings_movies_users_years_genres, validation$rating)
```

This model has improved the RMS to 0.8647135

```
models <- rbind(models, c("movies users years genres", rmse_movies_users_years_genres))
knitr::kable(models)
```

model	rmse
Mean	1.06120181029262
movies	0.943908662806309
movies users	0.865348824577316
movies users years	0.865004340330318
movies users years genres	0.864713456045103

Regularization

We have seen that some movies have only a few ratings. Those can have a disproportionate effect on variability. Regularization shrinks the coefficient estimates for small samples towards zero. We introduce a penalty term λ. A greater λ shrinks the coefficient more.

For example for the movie effect:

$\hat{b}_i(\lambda) = \frac{1}{\lambda+n_i} \sum_{u=2}^{n_i} (Y_{u,i} - \hat{\mu})$

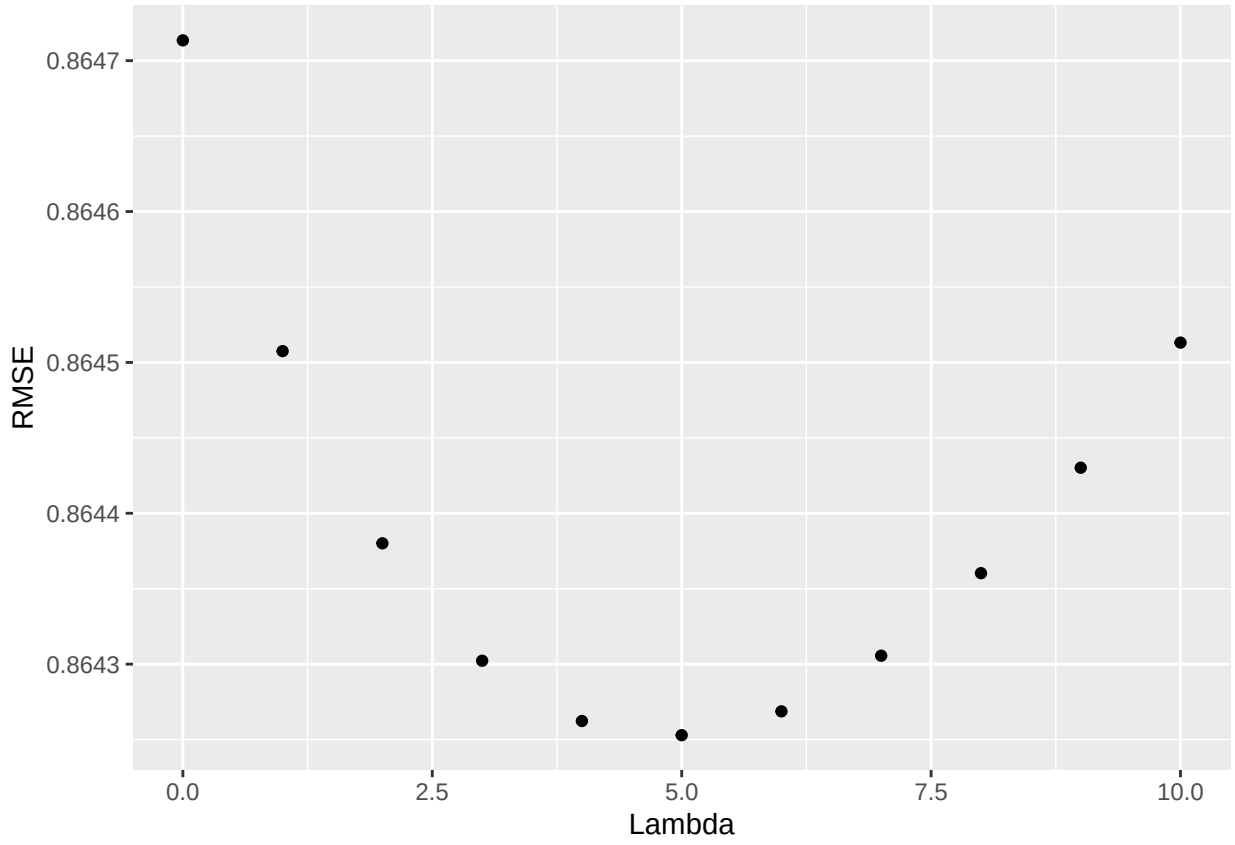
If the sample size n_g is large, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$

λ is a tuning parameter. We can use cross-validation to choose it.

Choosing an optimal λ

```
test_lambdas <- function(l) {
  movie_avgs_r <- edx %>% group_by(movieId) %>% summarise(b_i = sum(rating - mu_rating) /(n()+l))
  edx_movies_r <- edx %>% left_join(movie_avgs_r, by="movieId")
  validation_movies_r <- validation %>% left_join(movie_avgs_r, by="movieId")

  user_avg_r <- edx_movies_r %>% group_by(userId) %>%
    summarise(b_u = sum(rating - mu_rating - b_i)/(n()+l))
  edx_movies_users_r <- edx_movies_r %>% left_join(user_avg_r, by="userId")
  validation_movies_users_r <- validation_movies_r %>% left_join(user_avg_r, by="userId")

  year_avg_r <- edx_movies_users_r %>% group_by(year) %>%
    summarise(b_y = sum(rating - mu_rating - b_i - b_u)/(n()+l))
  edx_movies_users_years_r <- edx_movies_users_r %>% left_join(year_avg_r, by="year")
  validation_movies_users_years_r <- validation_movies_users_r %>%
    left_join(year_avg_r, by="year")

  genre_avg_r <- edx_movies_users_years_r %>% group_by(genres) %>%
    summarise(b_g = sum(rating - mu_rating - b_i - b_u - b_y)/(n()+l))
  edx_movies_users_years_genres_r <- edx_movies_users_years_r %>%
```

```
    left_join(genre_avg_r, by="genres")
  validation_movies_users_years_genres_r <- validation_movies_users_years_r %>%
    left_join(genre_avg_r, by="genres")

  predictions <- mu_rating + validation_movies_users_years_genres_r$b_i +
    validation_movies_users_years_genres_r$b_u +
    validation_movies_users_years_genres_r$b_y +
    validation_movies_users_years_genres_r$b_g
  rmse_movies_users_years_genres_reg <- RMSE(predictions, validation$rating)
  rmse_movies_users_years_genres_reg
}

lambdas <- seq(0,10, 1)
rmses <- sapply(lambdas, test_lambdas)

dfr <- data.frame(lambdas, rmses)
dfr %>% ggplot(aes(x=lambdas, y=rmses)) + geom_point() + labs(x="Lambda", y="RMSE")
```



Using the optimal λ gives the final model

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

movie_avgs_r <- edx %>% group_by(movieId) %>% summarise(b_i = sum(rating - mu_rating) /(n()+l))
edx_movies_r <- edx %>% left_join(movie_avgs_r, by="movieId")
validation_movies_r <- validation %>% left_join(movie_avgs_r, by="movieId")
```

```
user_avg_r <- edx_movies_r %>% group_by(userId) %>%
  summarise(b_u = sum(rating - mu_rating - b_i)/(n()+l))
edx_movies_users_r <- edx_movies_r %>% left_join(user_avg_r, by="userId")
validation_movies_users_r <- validation_movies_r %>% left_join(user_avg_r, by="userId")

year_avg_r <- edx_movies_users_r %>% group_by(year) %>%
  summarise(b_y = sum(rating - mu_rating - b_i - b_u)/(n()+l))
edx_movies_users_years_r <- edx_movies_users_r %>%
  left_join(year_avg_r, by="year")
validation_movies_users_years_r <- validation_movies_users_r %>% left_join(year_avg_r, by="year")

genre_avg_r <- edx_movies_users_years_r %>% group_by(genres) %>%
  summarise(b_g = sum(rating - mu_rating - b_i - b_u - b_y)/(n()+l))
edx_movies_users_years_genres_r <- edx_movies_users_years_r %>%
  left_join(genre_avg_r, by="genres")
validation_movies_users_years_genres_r <- validation_movies_users_years_r %>%
  left_join(genre_avg_r, by="genres")

predictions <- mu_rating + validation_movies_users_years_genres_r$b_i +
  validation_movies_users_years_genres_r$b_u +
  validation_movies_users_years_genres_r$b_y +
  validation_movies_users_years_genres_r$b_g
rmse_movies_users_years_genres_reg <- RMSE(predictions, validation$rating)
rmse_movies_users_years_genres_reg
```

```
## [1] 0.8642529
```

```
models <- rbind(models, c("movies users years genres regular", rmse_movies_users_years_genres_reg))
```

Comparing the different models:

```
knitr::kable(models)
```

model	rmse
Mean	1.06120181029262
movies	0.943908662806309
movies users	0.865348824577316
movies users years	0.865004340330318
movies users years genres	0.864713456045103
movies users years genres regular	0.864252901402938

Conclusion

The final model has a RMSE of 0.8642529 $<$ 0.86490. So we have achieved our goal.

[1] https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf