Predict movie rating

Mario R. Melchiori

1/5/2019

Introduction

Recommendation systems use rating data from many products and users to make recommendations for a specific user.

Netflix uses a recommendation system to predict your ratings for a specific 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.

In this capstone, we will build our own recommendation system.

Below, we will try out different models. Mainly, we use the methodology presented in the course's book, but adding the Matrix Factorization model, which was only introduced in it.

We keep track of the RMSE we get for each one because we will report the best (lower) on validation set, at the end.

The definition of root mean square error (RMSE) is square root of the average of the residuals squared:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2}$$

where Y_i is the true i-th movie rating and $\hat{Y}_i$ our predicted rating. We can interpret this similarly to a standard deviation. It is the typical error we make when predicting a movie rating.

We write a function called `RMSE()` that takes two numeric vectors (one corresponding to the true movie ratings, and one corresponding to predicted movie ratings) as input, and returns the root mean square error (RMSE) between the two as output.

The definition of root mean square error is square root of the average of the residuals squared:

```r
## RMSE compute root mean square error (RMSE)
RMSE <- function(true_ratings, predicted_ratings){
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

RMSE was the metric used to judge entries in the Netflix challenge. The lower the RMSE was on Netflix's quiz set between the submitted rating predictions and the actual ratings, the better the method was. We will be using RMSE to evaluate our machine learning models in this capstone as well.

The data

We download the MovieLens data and run the following code the staff provided to generate the datasets. We use the 10M version of the MovieLens dataset available [here] (https://grouplens.org/datasets/movielens/10m/).

```
#################################################################
# Create edx set, validation set, and submission file
#################################################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse

## -- Attaching packages -----------------------------------------------------------------------

## v ggplot2 3.1.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0

## -- 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
```

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(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")
```

```r
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)
```

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

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

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

The data set is very long, so we use models that allow the data set can be adjusted and processed within the available RAM in a machine.

We randomly split the `edx` set into training and test data. We use `set.seed(1)` to provide a reproducible example and the `createDataPartition`'s caret function to select your test index to create two separate data frames called: `train` and `test` from the original `edx` data frame. `test` contain a randomly selected 10% of the rows of `edx`, and have `train` contain the other 90%. We will use these data frames to do the rest of the analyses. In the code, we make sure userId and movieId in test set are also in train set and add rows removed from `test` set back into `train` set. After you create `train` and `test`, we remove `removed`,`temp`,`test_index` to save space.

```r
# generate reproducible partition

set.seed(1)

# test set will be 10% of edx data

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

# Make sure userId and movieId in test set are also in train set

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

# Add rows removed from test set back into train set

removed <- anti_join(temp, test)
```

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

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

The code below is to save space.

```
# to save space
rm(removed, temp, test_index)
```

Data Exploration

We can see this table is in tidy format with thousands of rows:

```
# show train set
train %>% as_tibble()
```

```
## # A tibble: 8,100,065 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     292      5 838983421 Outbreak (1995)      Action|Drama|Sci-Fi~
##  3      1     316      5 838983392 Stargate (1994)      Action|Adventure|Sc~
##  4      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  5      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  6      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  7      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  8      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
##  9      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## 10      1     377      5 838983834 Speed (1994)         Action|Romance|Thri~
## # ... with 8,100,055 more rows
```

Each row represents a rating given by one user to one movie. We can see the number of unique users from `train` that provided ratings and how many unique movies were rated. How many users are there in the `train` data set? How many movies are in the `train` data set? What is the lowest rating in the `train` data set? The highest?

```
train %>% summarize(
n_users=n_distinct(userId),# unique users from train
n_movies=n_distinct(movieId),# unique movies from train
min_rating=min(rating),  # the lowest rating
max_rating=max(rating) # the highest rating
)
```

```
##   n_users n_movies min_rating max_rating
## 1   69878    10677        0.5          5
```

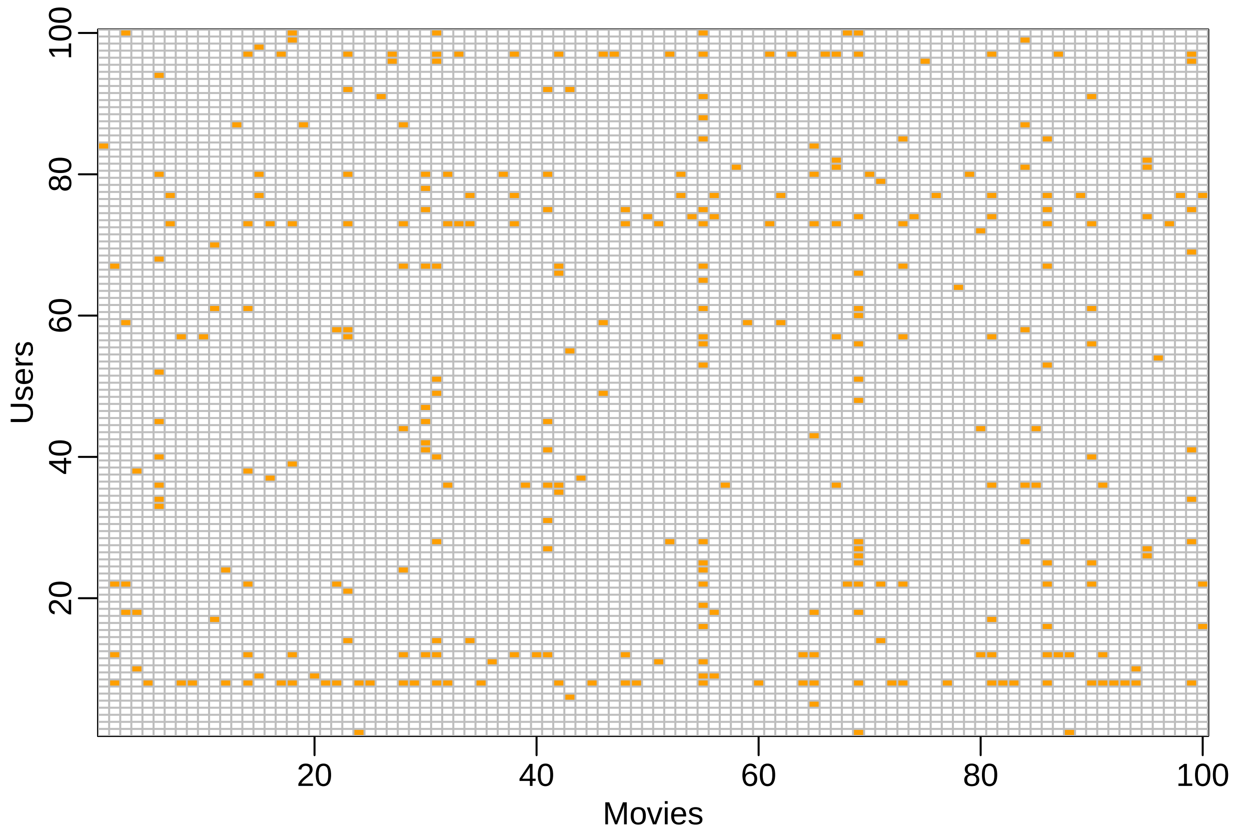
If we multiply `n_users` times `n_movies`, we get a number almost of 750 million, yet our data table has about 8,000,000 rows. This implies that not every user rated every movie. So we can think of these data as a very large matrix, with users on the rows and movies on the columns, with many empty cells. The `gather` function permits us to convert it to this format, but if we try it for the entire matrix, it will crash R. Let's show the matrix for seven users and five movies.

```r
# matrix for 5 movies and 7 users
keep <- train %>%
  count(movieId) %>%
  top_n(5, n) %>%
  .$movieId

tab <- train %>%
  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)	Shawshank Redemption (1994)	Silence of th
13	NA	NA	4	NA	
16	NA	3	NA	NA	
17	NA	NA	NA	NA	
18	NA	3	NA	4.5	
19	4	1	NA	4.0	

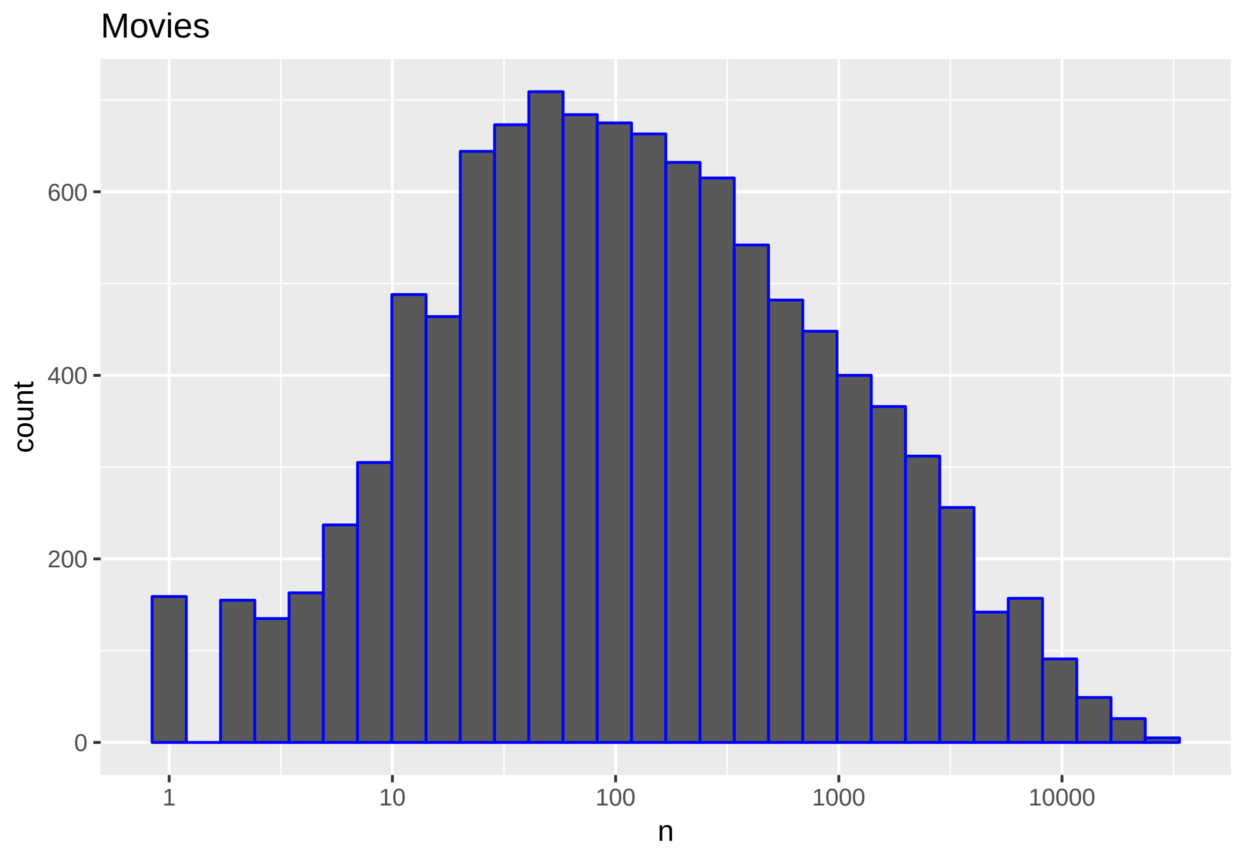
You can think of the task of a recommendation system as filling in the `NA`s in the table above. To see how *sparse* the matrix is, here is the matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating.

```r
# matrix for a random sample of 100 movies and 100 users with yellow
# indicating a user/movie combination for which we have a rating.

users <- sample(unique(train$userId), 100)
rafalib::mypar()
train %>% 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 = "grey")
```



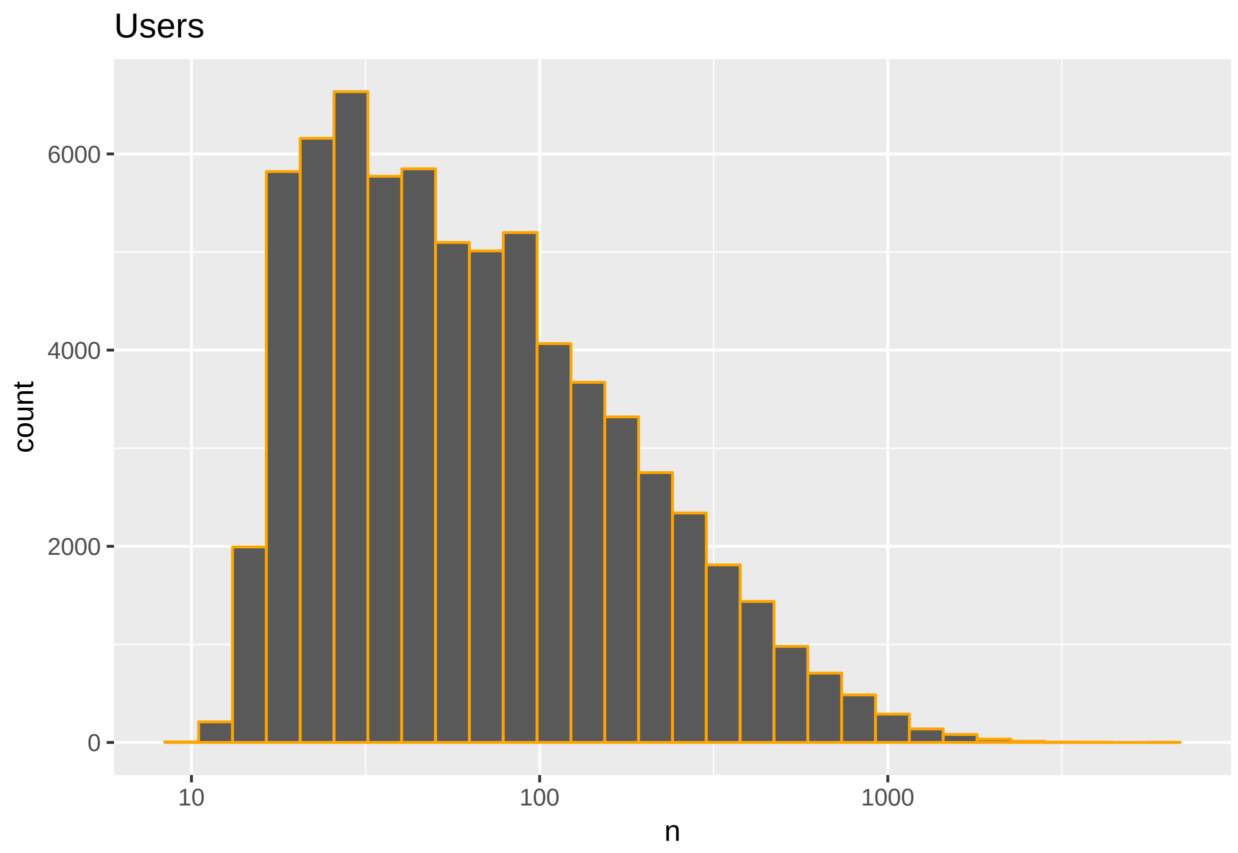
The first thing we notice is that some movies get rated more than others. Here is the distribution:

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



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

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



The code below is to save space.

```
# to save space
rm(tab, keep, users)
```

Methods and Analysis

More simplest model

We start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. What number should this prediction be? 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: Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. What number should this prediction be? 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:

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

```
mu <- mean(train$rating) # compute mean rating
mu
```

```
## [1] 3.512456
```

```r
naive_rmse <- RMSE(test$rating, mu) # compute root mean standard error
naive_rmse
```

```
## [1] 1.060054
```

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. To win the grand prize of $1,000,000, a participating team had to get an RMSE of about 0.857. So we can definitely do much better!

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

```r
# create a results table with this approach

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

method	RMSE
Just the average	1.060054

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:

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

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.

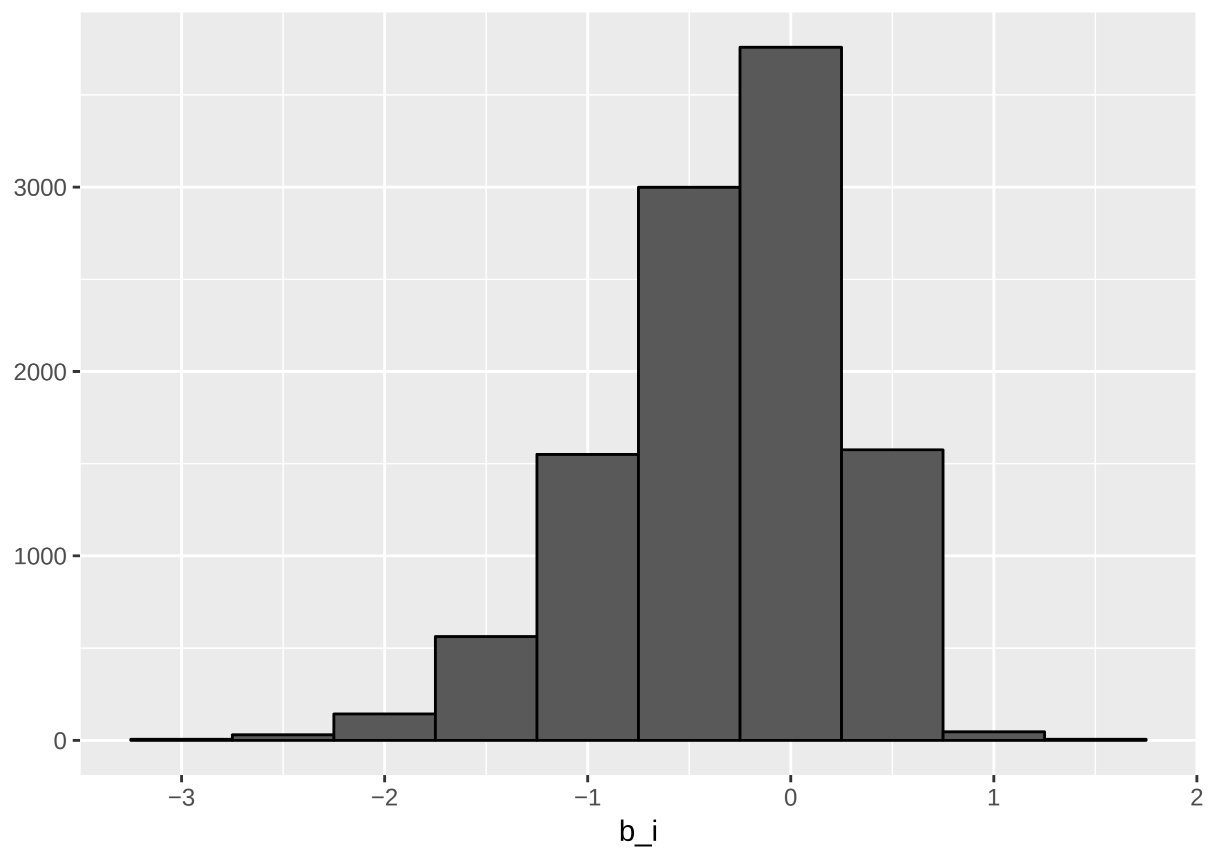
We can again use least squares to estimate the b_i in the following way,

```r
fit <- lm(rating ~ as.factor(movieId), data = train)
```

but because there are thousands of b_i as each movie gets one, the `lm()` function will be very slow here if not imposible to run.

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:

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



Let's see how much our prediction improves once we use $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$:

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

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

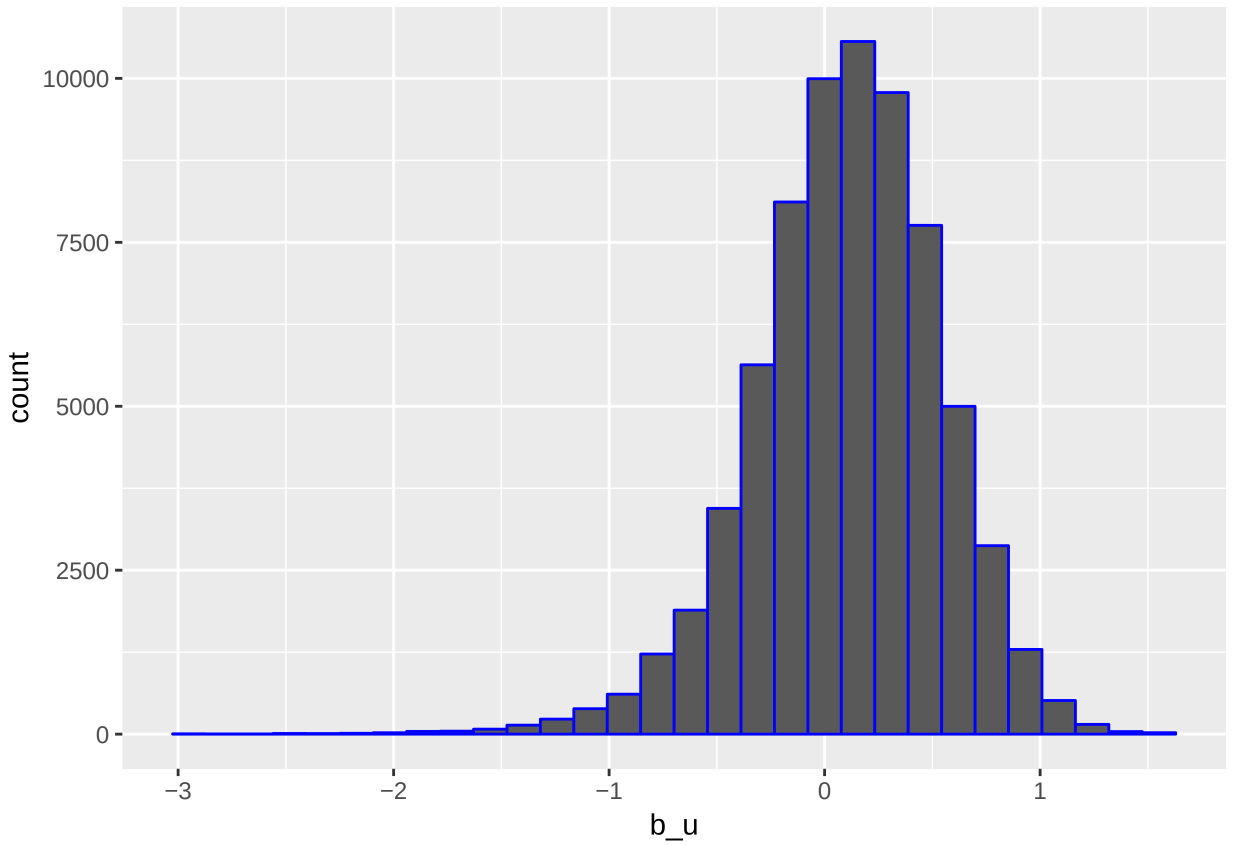
method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615

We already see an improvement. But can we make it better?

Modeling User effects

Let's compute the average rating for user u:

```r
train %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "blue")
```



Notice that 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:

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

where b_u is a user-specific effect. Now if a cranky user (negative b_u) rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

To fit this model, we could again use `lm` like this:

```r
lm(rating ~ as.factor(movieId) + as.factor(userId))
```

but, for the reasons described earlier, we won't. Instead, we will compute an approximation by computing $\hat{\mu}$ and $\hat{b}_i$ and estimating $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$:

```r
# compute user effect b_u
user_avgs <- train %>%
  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:

```r
# compute predicted values on test set
predicted_ratings <- test %>%
  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, test$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	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843

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:

```r
# Regularization

# connect movieId to movie title
movie_titles <- train %>%
  select(movieId, title) %>%
  distinct()
```

Here are the 10 best movies according to our estimate:

```r
# top 10 best movies based on b_i
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487544
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487544
Shadows of Forgotten Ancestors (1964)	1.487544
Fighting Elegy (Kenka erejii) (1966)	1.487544
Sun Alley (Sonnenallee) (1999)	1.487544
Blue Light, The (Das Blaue Licht) (1932)	1.487544
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544
Human Condition II, The (Ningen no joken II) (1959)	1.237544
Human Condition III, The (Ningen no joken III) (1961)	1.237544

And here are the 10 worst:

```r
# top 10 worse movies based on b_i
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Besotted (2001)	-3.012456
Hi-Line, The (1999)	-3.012456
Accused (Anklaget) (2005)	-3.012456
Confessions of a Superhero (2007)	-3.012456
War of the Worlds 2: The Next Wave (2008)	-3.012456
SuperBabies: Baby Geniuses 2 (2004)	-2.767775
Disaster Movie (2008)	-2.745789
From Justin to Kelly (2003)	-2.638139
Hip Hop Witch, Da (2000)	-2.603365
Criminals (1996)	-2.512456

When often the best are rated:

```r
# add number of rating of the "best" obscure movies
train %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

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

title	b_i	n
Hellhounds on My Trail (1999)	1.487544	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487544	1

title	b_i	n
Shadows of Forgotten Ancestors (1964)	1.487544	1
Fighting Elegy (Kenka erejii) (1966)	1.487544	1
Sun Alley (Sonnenallee) (1999)	1.487544	1
Blue Light, The (Das Blaue Licht) (1932)	1.487544	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544	2
Human Condition II, The (Ningen no joken II) (1959)	1.237544	4
Human Condition III, The (Ningen no joken III) (1961)	1.237544	4

When often the worse are rated:

```r
# add number of rating of the "worse" obscure movies

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

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

title	b_i	n
Besotted (2001)	-3.012456	1
Hi-Line, The (1999)	-3.012456	1
Accused (Anklaget) (2005)	-3.012456	1
Confessions of a Superhero (2007)	-3.012456	1
War of the Worlds 2: The Next Wave (2008)	-3.012456	2
SuperBabies: Baby Geniuses 2 (2004)	-2.767775	47
Disaster Movie (2008)	-2.745789	30
From Justin to Kelly (2003)	-2.638139	183
Hip Hop Witch, Da (2000)	-2.603365	11
Criminals (1996)	-2.512456	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.

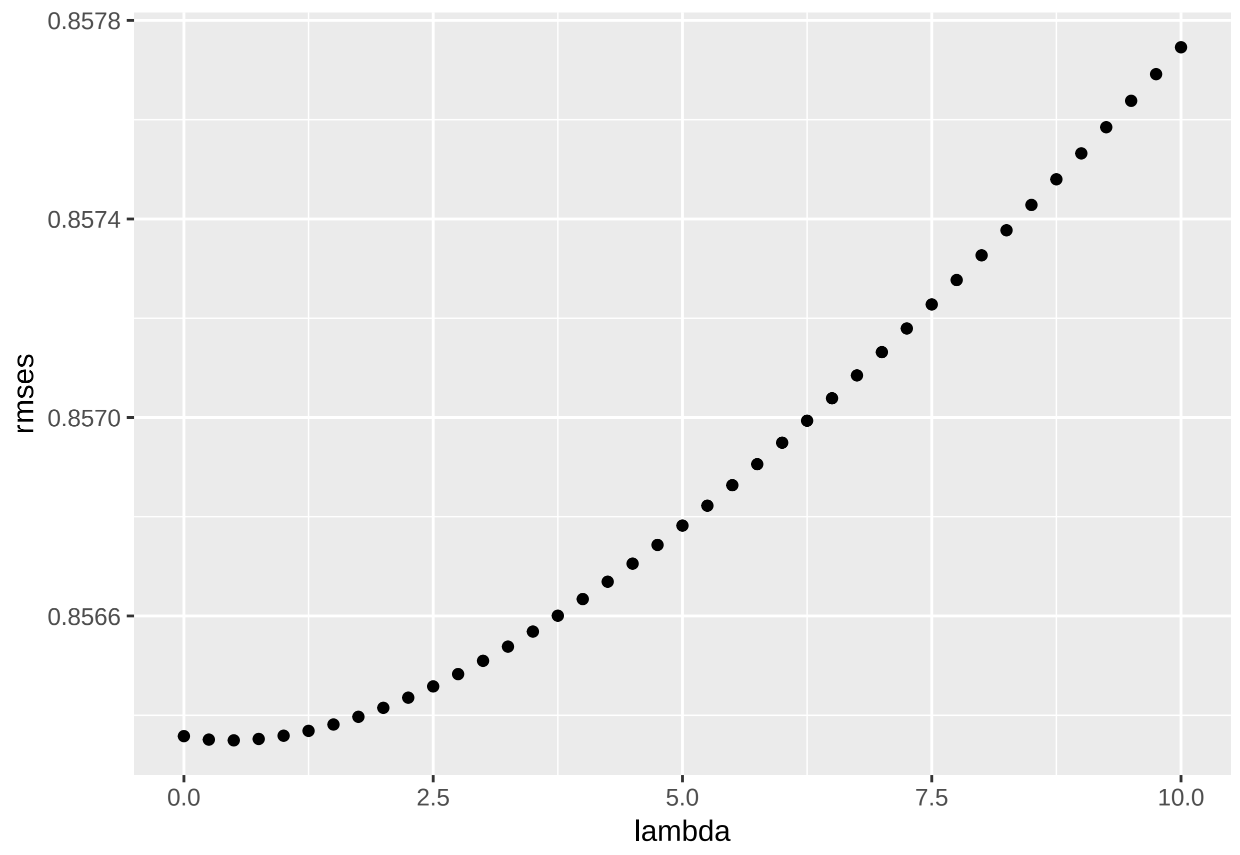
Choosing the penalty terms

We use regularization for the estimate both movie and user effects. We are minimizing:

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

Here we use cross-validation to pick a λ:

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

lambda <- seq(0, 10, 0.25)

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

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

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

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

  return(RMSE(train$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 test set for the final assessment of our third model

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

# compute user effect with regularization on train set
b_u <- train %>%
    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 <-
    test %>%
    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(test$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	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843
Reg Movie and User Effect Model on test set	0.8645518

In this case, the long dataset prevents obscure movies with just a few users increase our RMSE, so the regularization does not produce significant improvements in performance using the RMSE as a metric.

For this reason we discard the model with Reguralization and focus on the movies and users effects to calculate the residuals at next section.

Matrix factorization

We have described how the model:

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

accounts for movie to movie differences through the b_i 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:

$$r_{u,i} = y_{u,i} - \hat{u} - \hat{b}_i - \hat{b}_u$$

We compute the residuals for train and test set:

```
# compute movie effect without regularization on train set
b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# compute user effect without regularization on train set
b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - b_i - mu))

# compute residuals on train set
```

```r
train <- train %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# compute residuals on test set
test <- test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)
```

Matrix Factorization is a popular technique to solve recommender system problem. The main idea is to approximate the matrix $R_{m,n}$ by the product of two matrixes of lower dimension: $P_{k,m}$ and $Q_{k,n}$.

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: $P[,u]'$ x $Q[,i]$. Short and full description of package is available here.

This package works with data saved on disk in 3 columns with no headers.

The usage of recosystem is quite simple, mainly consisting of the following steps:

- Create a model object (a Reference Class object in R) by calling `Reco()`.
- (Optionally) call the `$tune()` method to select best tuning parameters along a set of candidate values.
- Train the model by calling the `$train()` method. A number of parameters can be set inside the function, possibly coming from the result of `$tune()`.
- (Optionally) export the model via `$output()`, i.e. write the factorization matrices P and Q into files or return them as R objects.
- Use the `$predict()` method to compute predicted values.

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

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

```r
# create data saved on disk in 3 columns with no headers
train_data <- data_memory(user_index = train$userId, item_index = train$movieId,
                          rating = train$res, index1 = T)

test_data <- data_memory(user_index = test$userId, item_index = test$movieId, index1 = T)

# create a model object
recommender <- Reco()
```

We use cross validation to tune the model parameters.the parameters are chosen by minimizing RMSE as a loss function. The code does not run here as it takes over one hour to optimize, but we show its output.

```r
# This is a randomized algorithm
set.seed(1)

# call the `$tune()` method to select best tuning parameters
res = recommender$tune(
    train_data,
    opts = list(dim = c(10, 20, 30),
                costp_l1 = 0, costq_l1 = 0,
                lrate = c(0.05, 0.1, 0.2), nthread = 2)
)

# show best tuning parameters
print(res$min)
```

```
## $dim
## [1] 30
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.05
##
## $loss_fun
## [1] 0.798114
```

The following code trains a recommender model. It will read from a training data source and create a model file at the specified location. The model file contains necessary information for prediction.

```
# Train the model by calling the `$train()` method
# some parameters coming from the result of `$tune()`
# This is a randomized algorithm
set.seed(1)
suppressWarnings(recommender$train(train_data, opts = c(dim = 30, costp_l1 = 0,
                                                        costp_l2 = 0.01, costq_l1 = 0,
                                                        costq_l2 = 0.1, lrate = 0.05,
                                                        verbose = FALSE)))
```

We predict unknown entries in the rating matrix on test set.

```
# use the `$predict()` method to compute predicted values
# return predicted values in memory
predicted_ratings <- recommender$predict(test_data, out_memory()) + mu + test$b_i + test$b_u
```

We use the test set for the final assessment

```
# ceiling rating at 5
ind <- which(predicted_ratings > 5)
predicted_ratings[ind] <- 5
```

```r
# floor rating at 0.50
ind <- which(predicted_ratings < 0.5)
predicted_ratings[ind] <- 0.5

# create a results table with this approach
model_4_rmse <- RMSE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User + Matrix Fact. on test set",
                                 RMSE = model_4_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843
Reg Movie and User Effect Model on test set	0.8645518
Movie and User + Matrix Fact. on test set	0.7965118

Lastly, to meet the requirements of the project we calculate the final RMSE on the validation dataset. We will use the edx dataset provided that ensures that it contains the users who are in validation set.

```r
# compute movies effect without regularization on edx set
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# compute users effect without regularization on edx set
b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - b_i - mu))

# compute residuals on edx set
edx <- edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# compute residuals on validation set
validation <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(res = rating - mu - b_i - b_u)

# Using recosystem
# create data saved on disk in 3 columns with no headers

edx_data <- data_memory(user_index = edx$userId, item_index = edx$movieId,
                          rating = edx$res, index1 = T)
```

```
validation_data <- data_memory(user_index = validation$userId, item_index = validation$movieId, index1 

# create a model object
recommender <- Reco()

# Train the model by calling the `$train()` method
# some parameters coming from the result of `$tune()`
# This is a randomized algorithm
set.seed(1)

suppressWarnings(recommender$train(edx_data, opts = c(dim = 30, costp_l1 = 0,
                                                       costp_l2 = 0.01, costq_l1 = 0,
                                                       costq_l2 = 0.1, lrate = 0.05,
                                                       verbose = FALSE)))

# use the `$predict()` method to compute predicted values
# return predicted values in memory

predicted_ratings <- recommender$predict(validation_data, out_memory()) + mu +
                      validation$b_i + validation$b_u

# ceiling rating at 5
ind <- which(predicted_ratings > 5)
predicted_ratings[ind] <- 5

# floor rating at 5
ind <- which(predicted_ratings < 0.5)
predicted_ratings[ind] <- 0.5

# create a results table with this and prior approaches
model_5_rmse <- RMSE(validation$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie and User effects and Matrix Fact. on validation set",
                                 RMSE = model_5_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0600537
Movie Effect Model on test set	0.9429615
Movie and User Effects Model on test set	0.8646843
Reg Movie and User Effect Model on test set	0.8645518
Movie and User + Matrix Fact. on test set	0.7965118
Movie and User effects and Matrix Fact. on validation set	0.7939559

Summary

The data set is very long, so we train models that allow the data set can be adjusted and processed within the available RAM in personal machine.

The long dataset prevents obscure movies with just a few users increase our RMSE, so the regularization does not produce significant improvements in performance using the RMSE as a metric.

We discard the model with Reguralization and focus on the movies and users effects to calculate the residuals.

This residuals were modeled using Matrix Factorization.

We reach a RMSE of 0.794 on `validation` set using the full `edx` set for training. *The model Movie and User effects and Matrix Factorization* achieved this performance using the methodology presented in the course's book without regularization and the R package `recosystem`.