Report on MovieLens project

Justyna Piątyszek

2022-04-01

Introduction

This is a report on a movie recommendation system that tries to predict what rating a given user would give a specific movie using regularization and matrix factorization methods.
A movie for which a high rating is predicted for a specific user can then be recommended to that user.

We will use a subset of the MovieLens dataset with 10 million ratings to generate our models.
MovieLens 10M dataset is available on the following websites:
https://grouplens.org/datasets/movielens/10m/
http://files.grouplens.org/datasets/movielens/ml-10m.zip

To compare different models, we will use residual mean squared error (RMSE) as our loss function.
We can interpret RMSE similarly to a standard deviation (a typical error we make when predicting a movie rating).
In the end, we will choose the model with the lowest RMSE for the final evaluation with the validation set.

We start by loading the needed libraries (please note that this process could take a couple of minutes):

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

library(tidyverse)
library(caret)
library(data.table)
library(dlookr)
library(lubridate)
library(recosystem)
```

Loading the file into R (for R 4.0 or upper version):

```
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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

We will create edx set and validation set (final hold-out test set). Validation set will be 10% of MovieLens data:

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

Making sure userId and movieId in validation set are also present in edx set:

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

Adding rows removed from validation set back into edx set:

```
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

Exploratory Data Analysis

We will perform an exploratory data analysis on the edx dataset. The validation set will be used solely to evaluate our final model and therefore we treat it as unknown data and do not include it in the EDA.

We can see that the edx dataset is a tidy data frame with 9,000,055 rows and 6 variables. Each line represents one rating of one movie by one user:

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
```

```
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
class(edx)
```

```
## [1] "data.table" "data.frame"
```

```
dim(edx)
```

```
## [1] 9000055       6
```

The following tibble shows types of all the 6 variables in the edx dataset.
Please note that `timestamp` represents seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
Using dlookr packaage, we can see that the edx dataset has 69,878 unique users and 10,677 unique movies. We can see that there are no missing values for any of the variables:

```
diagnose(edx)
```

```
## # A tibble: 6 x 6
##   variables types     missing_count missing_percent unique_count unique_rate
##   <chr>     <chr>             <int>           <dbl>        <int>       <dbl>
## 1 userId    integer               0               0        69878  0.00776
## 2 movieId   numeric               0               0        10677  0.00119
## 3 rating    numeric               0               0           10  0.00000111
## 4 timestamp integer               0               0      6519590  0.724
## 5 title     character             0               0        10676  0.00119
## 6 genres    character             0               0          797  0.0000886
```

The most given ratings are 4, 3 and 5. No movie has a rating of 0 as the movies are rated from 0.5 to 5.0 in half-star increments. We can also see that full-star ratings are more common than half-star ratings:

```
edx %>% group_by(rating) %>% summarise(n = n()) %>% arrange(desc(n))
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

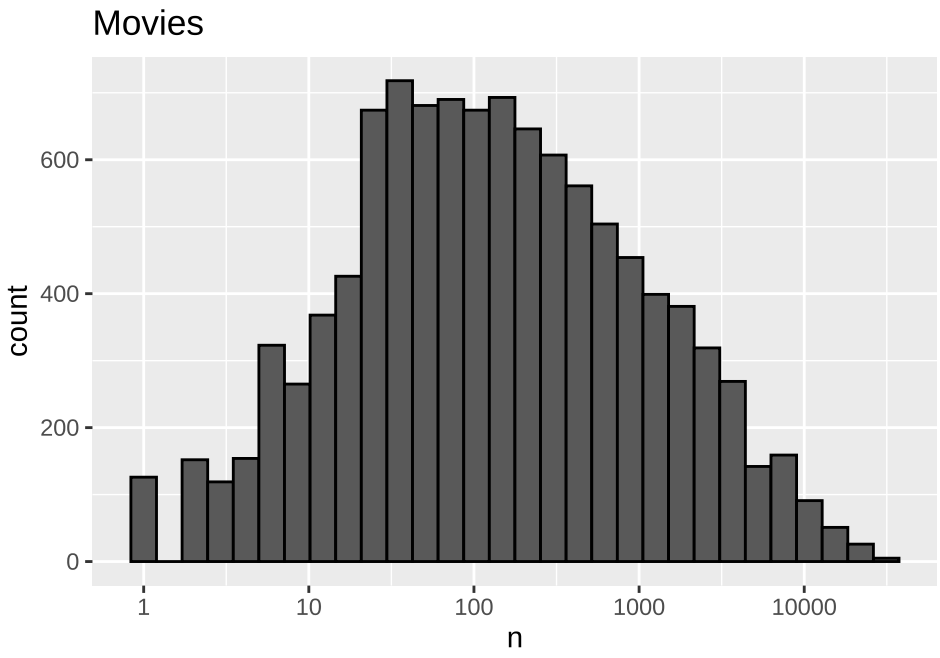
The tibble below shows 10 movie genres with the highest number of ratings received (note that we need to use `str_detect` function as most movies fall into multiple categories and the genres are pipe-separated and combined into one column):

```
genres = c("Action", "Adventure", "Animation", "Children", "Comedy", "Crime",
           "Documentary", "Drama", "Fantasy", "Film-Noir" ,"Horror", "Musical",
           "Mystery", "Romance","Sci-Fi", "Thriller", "War", "Western")
G <- sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
})
G <- as.data.frame(G, optional = TRUE) %>% setNames(., c("Number of ratings"))
G %>% arrange(desc(`Number of ratings`)) %>% top_n(10)
```

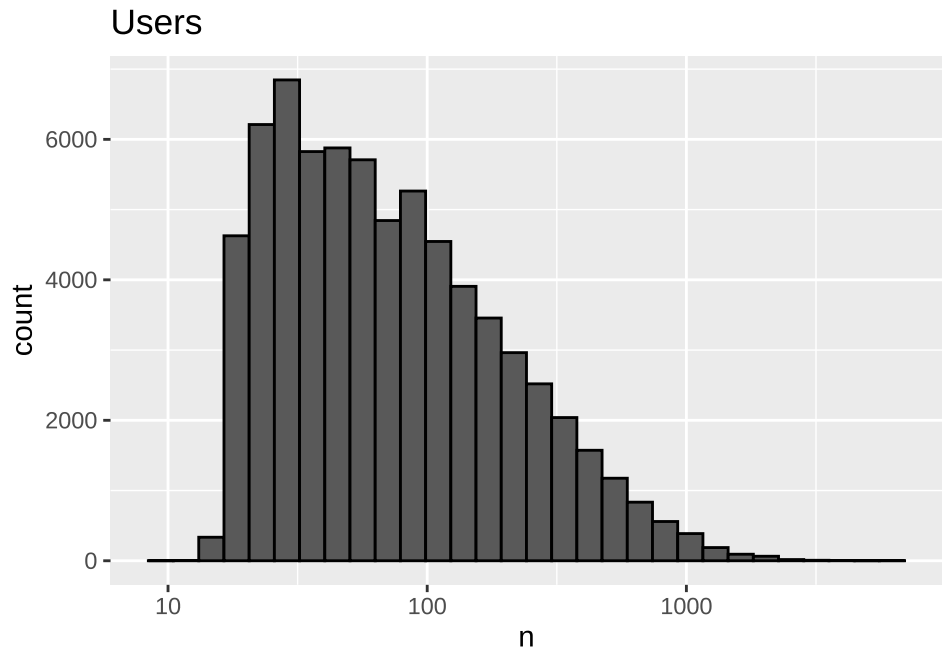
```
##           Number of ratings
## Drama                3910127
## Comedy               3540930
## Action               2560545
## Thriller             2325899
## Adventure            1908892
## Romance              1712100
## Sci-Fi               1341183
## Crime                1327715
## Fantasy               925637
## Children              737994
```

From the following histograms, we can see that some movies are rated more often than others and that some users are far more active than others in giving ratings:

```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")
```



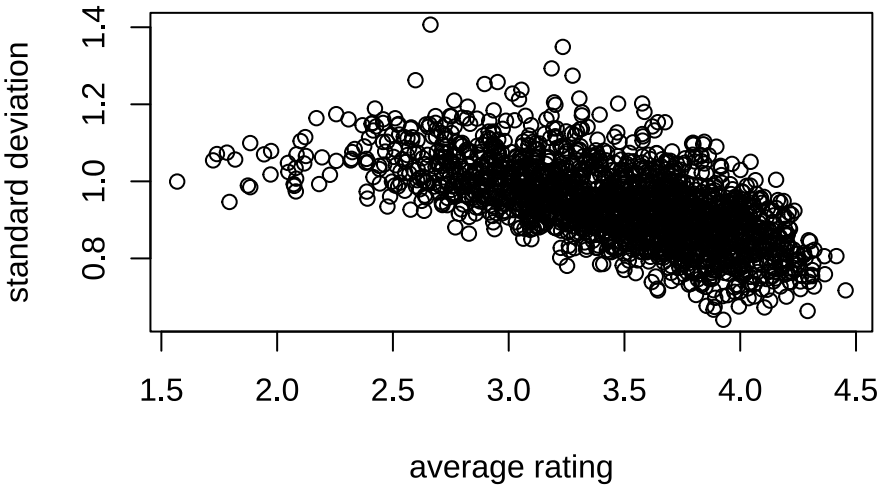
The following tibble shows 10 movies with the greatest number of ratings with the average, median and standard deviation of their ratings:

```
edx %>% group_by(movieId, title) %>%
  summarize(count = n(), average = mean(rating), median = median(rating),
            sd = sd(rating)) %>%
  arrange(desc(count))
```

```
## # A tibble: 10,677 x 6
## # Groups:   movieId [10,677]
##    movieId title                                     count average median    sd
##      <dbl> <chr>                                     <int>   <dbl>  <dbl> <dbl>
##  1     296 Pulp Fiction (1994)                       31362    4.15    4.5 1.00
##  2     356 Forrest Gump (1994)                       31079    4.01    4   0.972
##  3     593 Silence of the Lambs, The (1991)          30382    4.20    4   0.839
##  4     480 Jurassic Park (1993)                      29360    3.66    4   0.938
##  5     318 Shawshank Redemption, The (1994)          28015    4.46    5   0.717
##  6     110 Braveheart (1995)                         26212    4.08    4   0.953
##  7     457 Fugitive, The (1993)                      25998    4.01    4   0.778
##  8     589 Terminator 2: Judgment Day (1991)         25984    3.93    4   0.906
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a~ 25672    4.22    4.5 0.914
## 10     150 Apollo 13 (1995)                          24284    3.89    4   0.852
## # ... with 10,667 more rows
```

We notice that movies with low to moderate rating tend to have higher rating variability than movies with high average rating:

```
T <- edx %>% group_by(movieId, title) %>%
  summarize(count = n(), average = mean(rating), median = median(rating), sd = sd(rating)) %>%
  filter(count > 1000)
plot(T$average, T$sd, xlab="average rating", ylab="standard deviation")
```



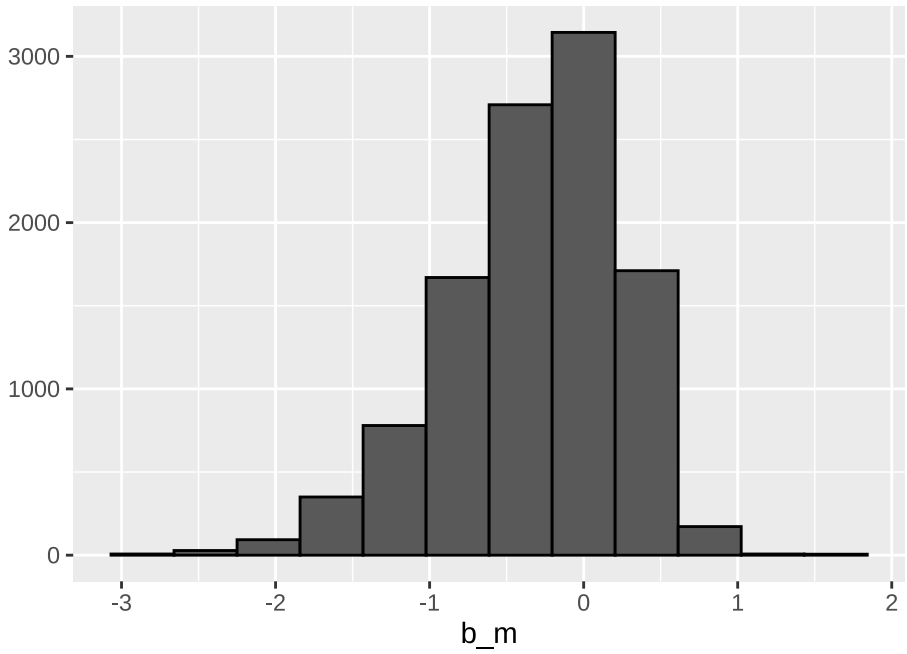
We can also see that some movies are generally rated lower than others, we will mark it as **b_m** - movie bias. To demonstrate this, we calculate the overall average for all the ratings in the dataset:

```
mu <- mean(edx$rating)
print(mu)
```

```
## [1] 3.512465
```

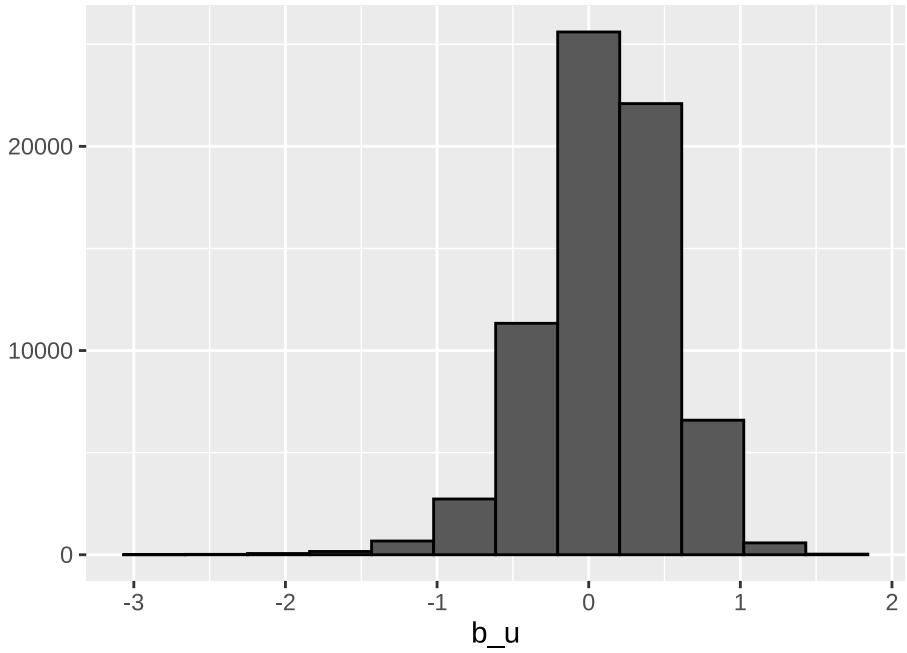
And plot a histogram which demonstrates the number of movies that are on both sides of the overall average mu:

```
edx %>% group_by(movieId) %>% summarize(b_m = mean(rating - mu)) %>%
  qplot(b_m, geom ="histogram", bins = 12, data = ., color = I("black"))
```



We also notice that there is a variability among users - we will mark it as **b_u** - user bias. The following histogram shows the number of users with reference to the average rating:

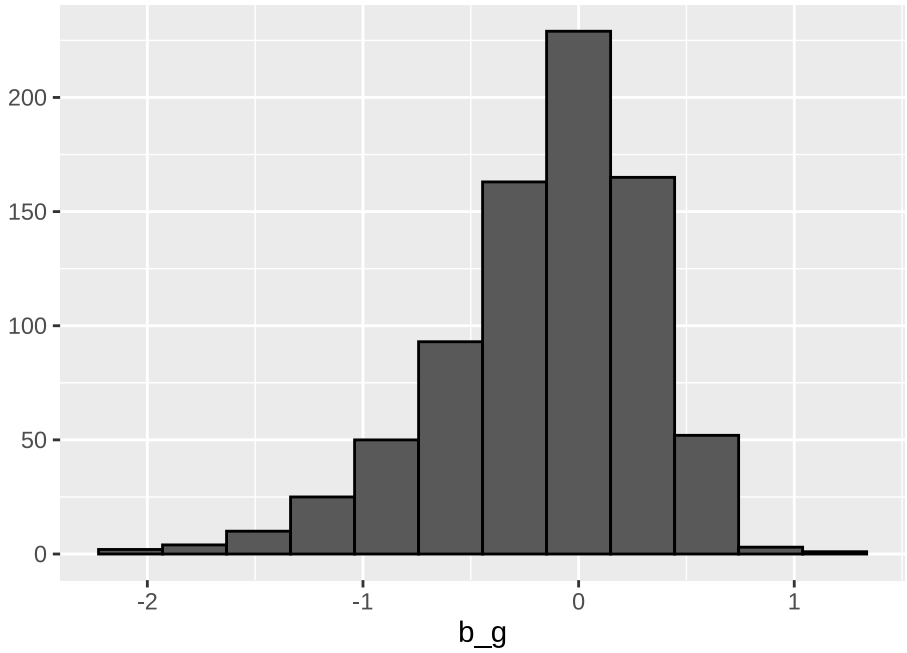
```
edx %>% group_by(userId) %>% summarize(b_u = mean(rating - mu)) %>%
  qplot(b_u, geom ="histogram", bins = 12, data = ., color = I("black"))
```



Most users average ratings are around **3.5** stars, but there is a big group of users who tend to give higher ratings around 4 stars and a group of users who give a movie an average of 3 stars.

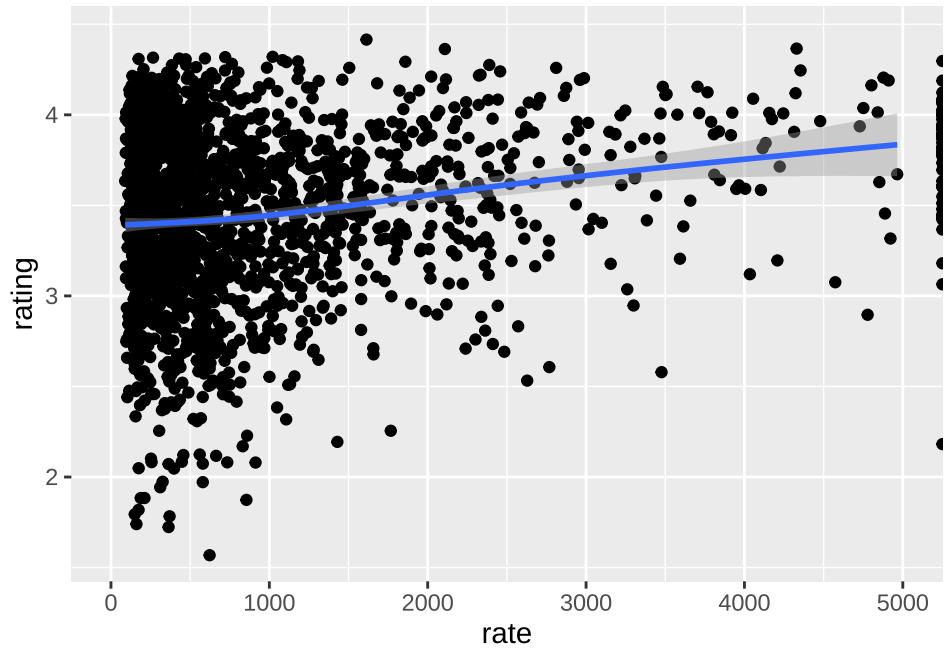
A similar effect can be noticed for different genres. Some of them seem to get a much higher average rating than others. We will add it later to our model as **b_g** - genre bias:

```
edx %>% group_by(genres) %>%
  summarize(b_g = mean(rating - mu)) %>%
  qplot(b_g, geom ="histogram", bins = 12, data = ., color = I("black"))
```



We will also add **b_n** - a number bias, as we can see the trend that movies with more ratings per year (rate) have a slightly higher average rating. We extract the year from the timestamp (movies in our dataset were rated between January 1995 and the first days of 2009):

```
edx %>% mutate(year = year(as_datetime(timestamp))) %>% group_by(movieId) %>%
  mutate(count = n()) %>%
  filter(count > 1000) %>%
  summarize(b_n = n(), years = 2008 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = b_n/years) %>%
  ggplot(aes(rate, rating)) +
  geom_point() +
  geom_smooth() +
  xlim(0, 5000)
```



Method - Prediction models

We will build several prediction models based on the observed biases and compare RMSE of each model.

First, we create train and test sets of ratings from edx:

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

We need to perform semi_join to make sure that all movies and users in the test set are present also in the training set:

```
test_set <- test_set %>% semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

We construct a function calculating RMSE for a vector of ratings and their predictions to be used for evaluating our models:

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

Calculating the average rating in the train set:

```
mu <- mean(train_set$rating)
print(mu)
```

```
## [1] 3.512482
```

Regularization model

To create models accounting for the movie-, user-, genre- and number-specific effects, we will take the following steps for each of the models:

1. Group our `train_set` by the chosen variable (`movieId`, `userId`, `genres` and a new variable `rate` for the number effect).
2. Calculate the given bias by extracting the average rating `mu` from the rating of the given group.
3. Create predictions on the `test_set` by adding the bias variable(s) to the average `mu`.
4. Calculate the error using the `RMSE` function.

These four code sections perform all of the mentioned steps for the models and print out the RMSE:
MOVIE BIAS MODEL

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu))

predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_m

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_1_rmse)
```

```
## [1] 0.9437429
```

MOVIE + USER BIAS MODEL

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

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

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_2_rmse)
```

```
## [1] 0.865932
```

MOVIE + USER + GENRE BIAS MODEL

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
```

```
  summarize(b_g = mean(rating - mu - b_m - b_u))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred2 = mu + b_m + b_u + b_g) %>%
  .$pred2

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_3_rmse)
```

```
## [1] 0.8655941
```

MOVIE + USER + GENRE + NUMBER BIAS MODEL

```
number_avgs <- train_set %>%
  mutate(year = year(as_datetime(timestamp))) %>% group_by(movieId) %>%
  mutate(count = n()) %>%
  mutate(n = count, years = 2008 - first(year),
            title = title[1],
            rating = mean(rating)) %>%
  mutate(rate = n/years) %>%
  ungroup() %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(rate) %>%
  summarize(movieId = movieId, b_n = mean(rating - mu - b_m - b_u - b_g))

number_avgs <- number_avgs[,-1]

number_avgs <- number_avgs %>% distinct(movieId, .keep_all = TRUE)

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(number_avgs, by='movieId') %>%
  mutate(pred3 = mu + b_m + b_u + b_g + b_n) %>%
  .$pred3

model_4_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_4_rmse)
```

```
## [1] 0.865026
```

We can see that by adding the effects of movie, user, genre and number of ratings, we greatly improved our RMSE down to **0.8650**.

REGULARIZED MOVIE + USER + GENRE + NUMBER BIAS MODEL

For our fourth model including all the effects, we will perform the regularization technique. Regularization (also called shrinkage method) penalizes large estimates calculated around small sample sizes as in our case, estimates for movies with very small number of ratings can have a negative effect on our overall RMSE result. Regularization aims at reducing the variance of the effect sizes.

We will use `lambda` as the penalty term.

The following `rmses` function will calculate predictions with regularized movie, user, genre and number biases for the sequence of lambdas between 0 and 5 in 0.25 increments:

```
lambdas <- seq(0, 5, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_m <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu)/(n()+l))
  b_g <- train_set %>%
    left_join(b_m, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_m - b_u- mu)/(n()+l))
  b_n <- train_set %>%
    mutate(year = year(as_datetime(timestamp))) %>% group_by(movieId) %>%
    mutate(count = n()) %>%
    mutate(n = count, years = 2008 - first(year),
           title = title[1],
           rating = mean(rating)) %>%
    mutate(rate = n/years) %>%
    ungroup() %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    group_by(rate) %>%
    summarize(movieId = movieId, b_n = sum(rating - b_m - b_u - b_g - mu)/(n()+l))
  b_n <- b_n[,-1]
  b_n <- b_n %>% distinct(movieId, .keep_all = TRUE)
  predicted_ratings <-
    test_set %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_n, by = 'movieId') %>%
    mutate(pred = mu + b_m + b_u + b_g + b_n) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
lambda <- lambdas[which.min(rmses)]
model_5_rmse <- min(rmses)
print(model_5_rmse)
```

```
## [1] 0.8644904
```

We get the best predictions with the parameter lambda of **4.25**. The regularized model decreases our RMSE to **0.8645**.

Matrix factorization

We will check if we can get even better predictions by using the recosystem package.
The recosystem package contains functions of the recommender system using matrix factorization. The aim of factorization is to predict unknown entries (an unknown movie rating from a user) in the rating matrix based on observed values.

To run matrix factorization using recosystem, we need to reshape our data to a data frame with three columns: `user_id`, `item_id` and `rating` using `data_memory()` function, which creates an object of class `DataSource`:

```
train_data <- data_memory(user_index = train_set$userId,
                          item_index = train_set$movieId,
                          rating = train_set$rating)

test_data <- data_memory(user_index = test_set$userId,
                         item_index = test_set$movieId,
                          rating = test_set$rating)
```

We will use default parameters to train the data (please note that `recommender$train` will create a `recommender` model stored in-memory containing all necessary information for prediction, therefore we do not need to save it into a separate object):

```
recommender <- Reco()
recommender$train(train_data)
```

Returning predicted values in memory using test set and calculating RMSE:

```
predictions <- recommender$predict(test_data, out_memory())
model_6_rmse <- RMSE(predictions, test_set$rating)
print(model_6_rmse)
```

```
## [1] 0.8340639
```

We can see that the matrix factorization method gives us a much lower RMSE and we will use it to evaluate our predictions on the validation set.

Converting the validation set into the recosystem format:

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

Calculating predictions and the final RMSE:

```
predictions_final <- recommender$predict(validation_data, out_memory())
final_model_RMSE <- RMSE(predictions_final, validation$rating)
print(final_model_RMSE)
```

```
## [1] 0.8340526
```

Results

We will store the results of all the models in the following table:

```
rmse_results <- data_frame(method = c("MOVIE BIAS MODEL", "MOVIE + USER BIAS MODEL",
                                      "MOVIE + USER + GENRE BIAS MODEL",
                                      "MOVIE + USER + GENRE + NUMBER BIAS MODEL",
                                      "REGULARIZED MODEL", "MF MODEL", "FINAL MODEL"),
                           RMSE = c(model_1_rmse, model_2_rmse, model_3_rmse, model_4_rmse,
                                    model_5_rmse, model_6_rmse, final_model_RMSE))
rmse_results$RMSE <- format(round(rmse_results$RMSE,8),nsmall=8)
print(rmse_results)
```

```
## # A tibble: 7 x 2
##   method                                RMSE
##   <chr>                                 <chr>
## 1 MOVIE BIAS MODEL                      0.94374291
## 2 MOVIE + USER BIAS MODEL               0.86593196
## 3 MOVIE + USER + GENRE BIAS MODEL       0.86559407
## 4 MOVIE + USER + GENRE + NUMBER BIAS MODEL 0.86502597
## 5 REGULARIZED MODEL                     0.86449037
## 6 MF MODEL                              0.83406394
## 7 FINAL MODEL                           0.83405263
```

Conclusion:

By using the regularization method accounting for different effects observed in the data, we managed to improve our RMSE from **0.9437** down to **0.8645**.
The recosystem package allowed us to effectively build a prediction model using matrix factorization with just a few lines of code. By using default parameters, we improved our RMSE to **0.8340** which means we could use it to make reliable movie recommendations.
Our final model could benefit from tuning parameters (such as the number of factors (`dim`), regularization for P and Q factors (`costp_l2`, `costq_l2`), the number of iterations (`niter`) or learning rate (`lrate`)) since we can see that this method provided a substantially lower RMSE than the regularized model. However, we have to take into account that the calculations would require a lot of time due to the size of the dataset and may be restricted by machine computing capabilities, which can cause the R session to abort.