Movie recommendation system

Ana Radić

06.05.2020.

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, 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. In this project, we will be creating a *movie recommendation system* using the 10M version of the MovieLens dataset. You can find the entire latest MovieLens dataset here.

10M version of the MovieLens dataset contains 10000054 rows and 6 columns, which is divided into **edx** set (containing 90% of MovieLens) and **validation set** (containing 10% of MovieLens). Edx set will be used to train different predicting models, and for that purpose will be divided into train and test set. We will develop an algorithm using **only** the train set. Once we are done developing the algorithm, we will freeze it and evaluate it using the test set. Decision on a winner model we based on the *root mean squared error (RMSE)* obtain from the test set. We define $y_{\mathrm{u,i}}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$. The RMSE is then defined as:

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

The model that produces the smallest RMSE is the best model. When one model stands out as the best one, we apply that model on the validation set to obtain the final RMSE. **The validation set ONLY be used to test final model**.

ANALYSIS

Data exploration

Let's look at some of the general properties of the **edx** data to better understand the challenges. Edx data set contains 9000055 rows and 6 columns. We can see the first few lines of **edx** dataset:

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Each row represents a rating given by one user to one movie, and here are some details for columns:

- **userId**: Unique ID for the user.
- **movieId**: Unique ID for the movie.
- **rating**: A rating between 0 and 5 for the movie.
- **timestamp**: Date and time the rating was given (in seconds from 1970-01-01).
- **title**: Movie title.
- **genres**: Genres associated with the movie.

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000055	Length:9000055
1st Qu.:18124	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
Median :35738	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35870	Mean : 4122	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

In this project, we will use columns **movieId** and **userId** to predict the rating of a particular user for a particular movie. We will show that the genres column also has an effect on the average rating as well as the movie release time.

We are interested in how many unique users provided ratings and how many unique movies were rated.

```
kable(edx %>% summarize(users = n_distinct(userId),
                        movies = n_distinct(movieId)))
```

users	movies
69878	10677

Also, genres column includes every genre that applies to the movie. Some movies fall under several genres. There is a 797 different movie genre combinations. We have pulled out a few genres to see which one is most commonly evaluated.

```
g <- c("Drama","Comedy","Thriller","Romance", "Horror", "Action")
tab <- sapply(g, function(z)
       sum(str_detect(edx$genres, z)))
       kable(t(tab))
```

Drama	Comedy	Thriller	Romance	Horror	Action
3910127	3540930	2325899	1712100	691485	2560545

It might be interesting to see which movies have been most rated and which movie have the highest average rate:

```
kable(edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>% head())
```

movieId	title	count
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212

```
kable(edx %>% group_by(movieId, title) %>%
              summarise(avg = mean(rating), n = n()) %>%
              arrange(desc(avg)) %>% head())
```

movieId	title	avg	n
3226	Hellhounds on My Trail (1999)	5	1
33264	Satan's Tango (SÃ˘tÃ˘ntangÃł) (1994)	5	2
42783	Shadows of Forgotten Ancestors (1964)	5	1
51209	Fighting Elegy (Kenka erejii) (1966)	5	1
53355	Sun Alley (Sonnenallee) (1999)	5	1
64275	Blue Light, The (Das Blaue Licht) (1932)	5	1

We've noticed that movies that have been rated one or few times have the highest rating, so we don't get a real picture of the best movie. For this reason, we will only consider movies that have been rated more than 100 times. And here are the top 6 movies based on average rating:

```
kable(edx %>% group_by(movieId, title) %>%
        summarise(a = mean(rating), n = n()) %>%
        filter(n > 100) %>%
        arrange(desc(a)) %>% head())
```

movieId	title	a	n
318	Shawshank Redemption, The (1994)	4.455131	28015
858	Godfather, The (1972)	4.415366	17747
50	Usual Suspects, The (1995)	4.365854	21648
527	Schindler's List (1993)	4.363493	23193
912	Casablanca (1942)	4.320424	11232
904	Rear Window (1954)	4.318651	7935

Data preprocessing

The title column includes the name of the movie and released year.

```
head(edx$title)
```

```
## [1] "Boomerang (1992)"              "Net, The (1995)"
## [3] "Outbreak (1995)"               "Stargate (1994)"
## [5] "Star Trek: Generations (1994)" "Flintstones, The (1994)"
```

We will separate the title from the year to leave space for some further analysis. For this purpose we will use a **stringr** package. First, we will check that each title has the same structure, the year of released in parentheses at the very end of the title.

```
sum(str_detect(edx$title,"\\(\\d{4}\\)$")) == length(edx$title)
```

```
## [1] TRUE
```

Now we will define a new edx table that has one column more. That column is the year when the movie was released.

```
# cutting off the year
year <- str_extract(edx$title,"\\(\\d{4}\\)$") %>% str_extract("\\d{4}")

# removing year from the title
title <- str_remove(edx$title,"\\(\\d{4}\\)$") #

# new edx set with separate title and year released
new_edx <- edx %>%
            select(-title) %>%
            mutate(year = year, title = title)
```

The movielens dataset also **includes a timestamp** column. This variable represents the time and data in which the rating was provided. The units are seconds since January 1, 1970. For a more informative view, we will transform that column into a date and time form. We will need a package **lubridate** for this purpose.

```
new_edx <- new_edx %>% mutate(timestamp = as_datetime(timestamp))
```

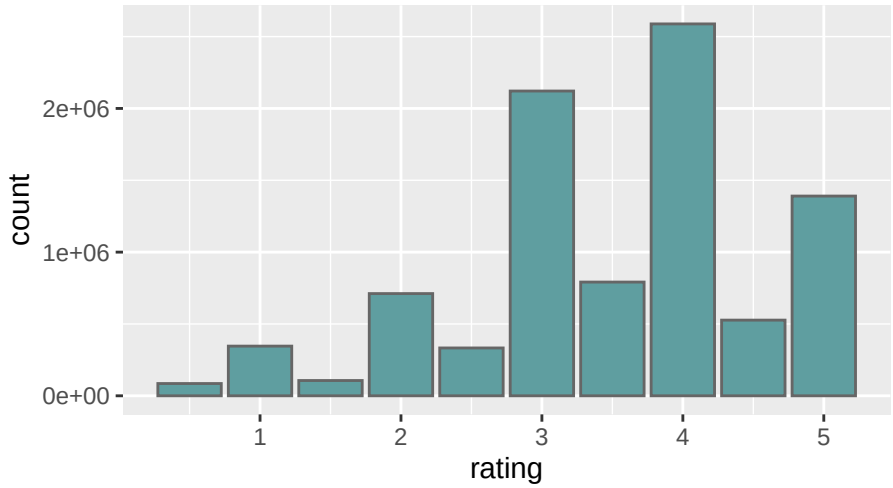
This is a few lines of new edx date set that we will continue to work with.

userId	movieId	rating	timestamp	genres	year	title
1	122	5	1996-08-02 11:24:06	Comedy|Romance	1992	Boomerang
1	185	5	1996-08-02 10:58:45	Action|Crime|Thriller	1995	Net, The
1	292	5	1996-08-02 10:57:01	Action|Drama|Sci-Fi|Thriller	1995	Outbreak
1	316	5	1996-08-02 10:56:32	Action|Adventure|Sci-Fi	1994	Stargate
1	329	5	1996-08-02 10:56:32	Action|Adventure|Drama|Sci-Fi	1994	Star Trek: Generations
1	355	5	1996-08-02 11:14:34	Children|Comedy|Fantasy	1994	Flintstones, The

Exploratory visuelization

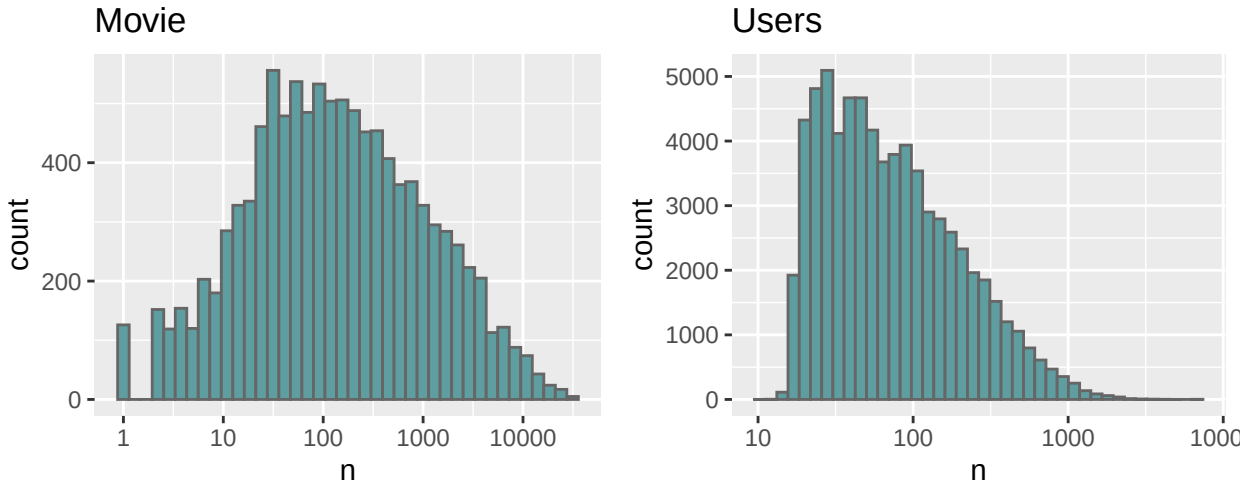
The average rating of all movies is 3.5124652, and the following plot and table shows distribution of ratings.

```
new_edx %>% ggplot(aes(x = rating) ) +
  geom_bar(fill = "cadetblue", color = "grey40")
```



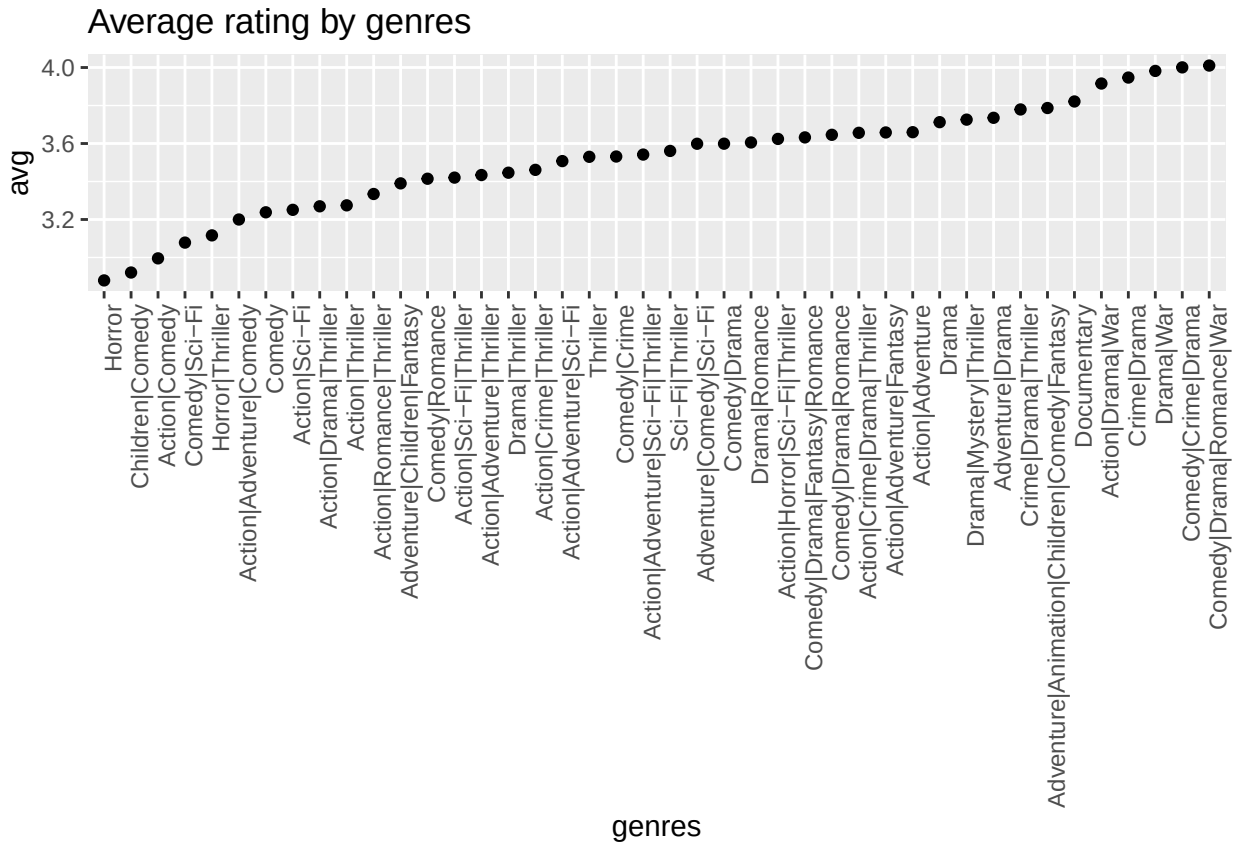
We notice that some movies get rated more than others, and that some users are more active than others at rating movies.

```
p1 <- new_edx %>% group_by(movieId) %>% summarise(n=n()) %>% ggplot(aes(n)) +
        geom_histogram(bins = 40, color = "grey40", fill = "cadetblue") +
        scale_x_log10() + ggtitle("Movie ")

p2 <- new_edx %>% group_by(userId) %>% summarise(n=n()) %>% ggplot(aes(n)) +
        geom_histogram(bins = 40, color = "grey40", fill = "cadetblue")  +
        scale_x_log10() + ggtitle("Users")

grid.arrange(p1, p2, ncol = 2)
```



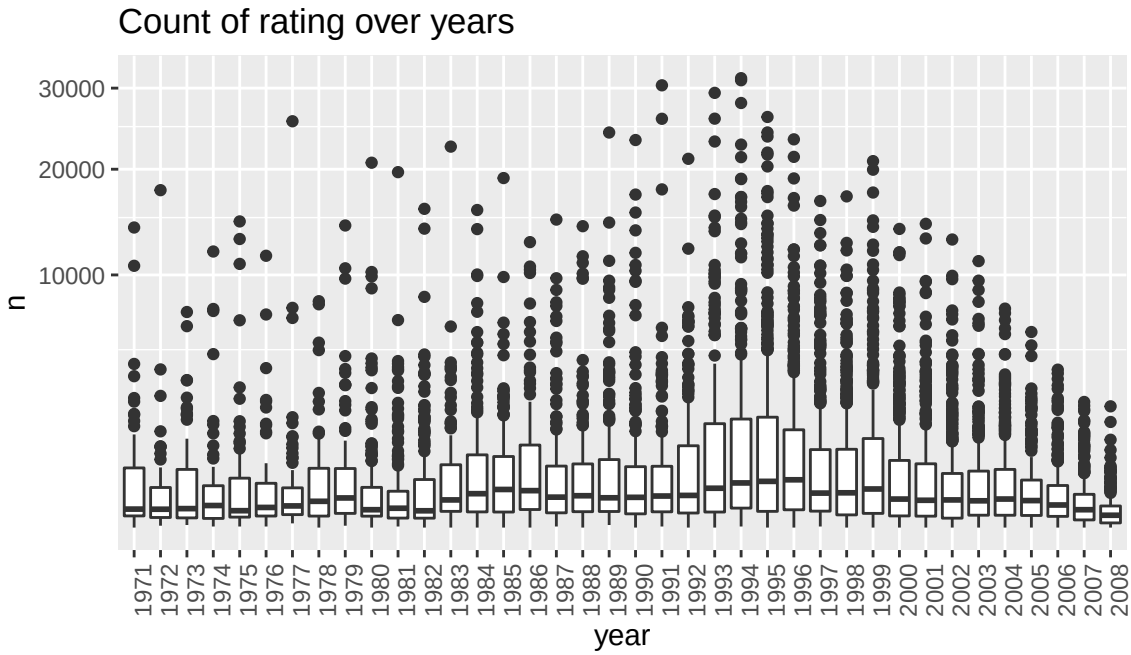
The average rating also depends on the genre. We see that some genres get rated more than others. The next plot shows strong evidence of a genre effect.

```
new_edx %>% group_by(genres) %>%
  summarize(n = n(),avg = mean(rating)) %>%
  filter(n >= 40000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg)) +  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Average rating by genres")
```



On average, movies that came out after 1993 get more ratings. We also see that with newer movies, starting in 1993, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

```
new_edx %>%  group_by(movieId) %>% filter(year > "1970") %>%
  summarize(n = n(), year = first(year))%>%
  qplot(year, n, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  ggtitle("Count of rating over years")
```



In order to predict the rating for movie i by the user u, we will consider the movie and user effect. Our exploratory analysis leads us to conclusive that *regularization* will help us achieve better results. Regularization permits us to penalize large estimates that are formed using small sample sizes. To compare models we use RMSE defined at the beginning.
First we will divide **new_edx** set into train and test set using `caret` package, through several lines of code:

```
ind <- createDataPartition(new_edx$rating, 1, 0.2, list=FALSE)
temp <- new_edx[ind,]
train <- new_edx[-ind,]

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

For practical reasons we will use the function that gives us the RMSE for the given predicted values, obtained from the train set and real values from the test set:

```
RMSE <- function(true, predicted){
  sqrt(mean((true - predicted)^2))
}
```

RESULTS

Basic model

The simplest model, without pure speculation, would be to replace every unknown rating with the average of all movies rating. Basic model look like this:

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

where $\epsilon_{u,i}$ is independent errors sampled from the same distribution centered at 0 and μ is the true rating for all movies. This model assumes the same rating for all movies and users with all the differences explained by random variation. Estimate for μ we denote as $\hat{\mu}$:

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

Let's see what RMSE we obtain with basic model on the test set:

```
error1 <- RMSE(test$rating, mu_hat)

tbl <- tibble(model = "basic model (train/test)",
              RMSE = error1)
kable(tbl)
```

model	RMSE
basic model (train/test)	1.060704

Our error iz about 1, that means about one star, which is not good.

Movie effect

Some movies have higher ratings on average and some less so we can improve our model by adding a movie effect. Our upgraded model now looks like this:

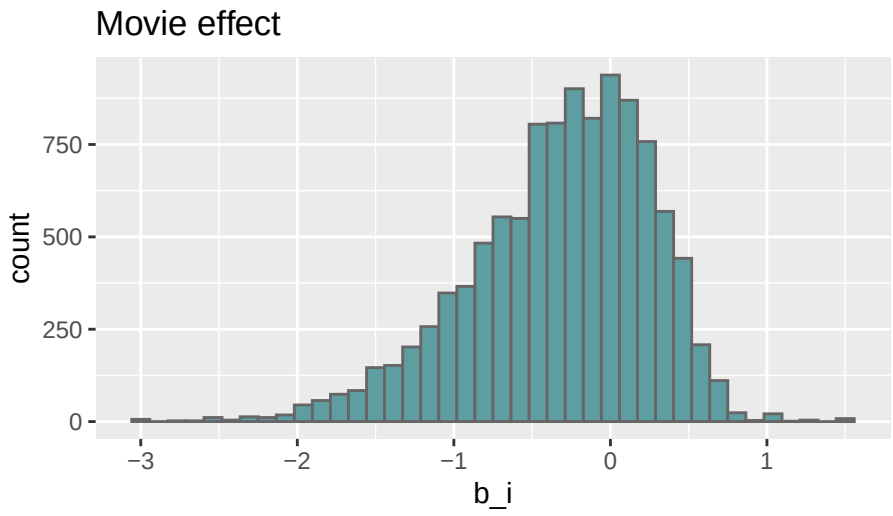
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Estimate for b_i is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i, and for the sake of convenience, we will use the same mark for estimate, b_i.
Let's compute how much, on average, the rating of a movie differs from the average rating of all movies.

```
f <- train %>%
       group_by(movieId) %>%
       summarize(b_i = mean(rating-mu_hat))
```

The following graph shows movie effect on the rating:

```
f %>% ggplot(aes(b_i)) +
      geom_histogram(bins = 40, color = "grey40", fill = "cadetblue") +
      ggtitle("Movie effect")
```



We can now predict new ratings for movies from the test set using an upgraded model, and the RMSE we obtain is already better .

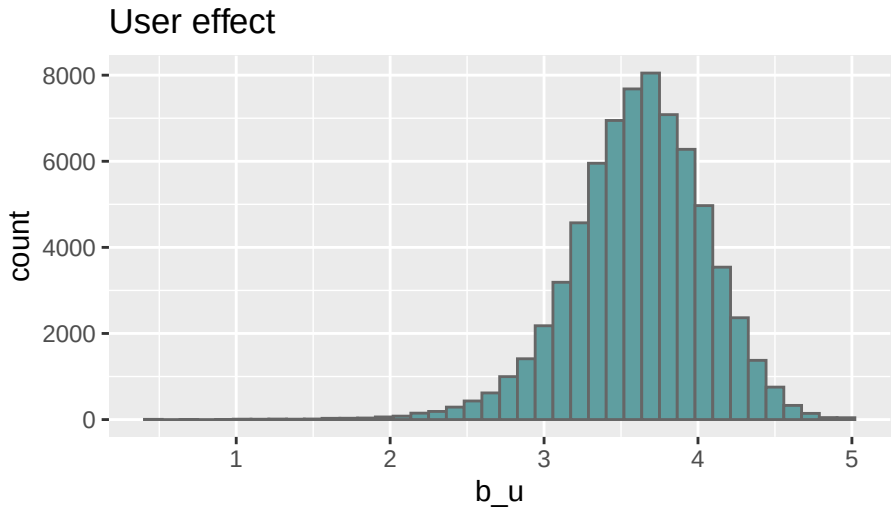
```
predict_movie <- test %>%
                    left_join(f, by="movieId") %>%
                    mutate(pred= mu_hat + b_i) %>%
                    pull(pred)

error2 <- RMSE(test$rating, predict_movie)

tbl <- bind_rows(tbl, tibble(model = "movie effect (train/test)", RMSE = error2))
kable(tbl)
```

model	RMSE
basic model (train/test)	1.0607045
movie effect (train/test)	0.9437144

User-movie effect

Let's see how much effect users have on movie ratings.

```
train %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 40, color = "grey40", fill = "cadetblue") +
  ggtitle("User effect")
```



Some users love every movie they watch and others generally give lower rate. This implies that a further improvement to our model may be:

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

where b_u is a user effect. Now, we add positive b_u to the predicted rating if the user most often gives a higher rating, and negative b_u if the user rarely likes the movie they watch. Estimate for b_u we compute as the average of $y_{u,i} - (\hat{\mu} + \hat{b}_i)$. Let's predict new rating for the movies from the test set and look at the latest RMSE:

```
k <- train %>%
       left_join(f, by = "movieId") %>%
       group_by(userId) %>%
       summarise(b_u = mean(rating-(mu_hat + b_i)))

predict_fk <- test %>%
                left_join(f, by = "movieId") %>%
                left_join(k, by = "userId") %>%
                mutate(pred = mu_hat  + b_i + b_u) %>%
                pull(pred)

error3 <- RMSE(test$rating, predict_fk)

tbl <- bind_rows(tbl, tibble(model = "movie-user effect (train/test)", RMSE = error3))
kable(tbl)
```

model	RMSE
basic model (train/test)	1.0607045
movie effect (train/test)	0.9437144
movie-user effect (train/test)	0.8661625

We're doing well, as RMSE gets smaller and smaller.

Regularization

The last thing we will do in this project to make RMSE smaller is to apply **regularization** to penalize estimates that are formed using small sample sizes. As we said in the previous section, the supposed "best" and "worst" movies were rated by very few users, in most cases just 1. This is because with just a few users, we have more uncertainty. The new $\hat{b}_i$ depends on the lambda parameter (a parameter that penalize the results from small samples):

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

where n_i is the number of ratings made for movie i.This approach will have our desired effect: when our sample size n_i is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink, so λ is the tuning parametar. We use cross validation (bootstrap version) to choose the best parameter.

```
# create 5 new pairs test-train sets and choose the best penalty parameter
i <- createDataPartition(train$rating, 5, 0.2, list=FALSE)
N <- 1:5

# finding the best penalty parameter (using bootstramp)
best_lambde <- sapply(N,function(n){

  # creatin test1 set and train1 set from train set
  temp <- train[i[,n],]
  train1 <- train[-i[,n],]

  # Make sure userId and movieId in test1 set are also in train1 set
  test1 <- temp %>%
    semi_join(train1, by = "movieId") %>%
    semi_join(train1, by = "userId")

  # Add rows removed from test1 set back into train1 set
  removed <- anti_join(temp, test1)
  train1 <- rbind(train1, removed)

  # tuning parameter on train1 set and test1 set
  avg1 <- mean(train1$rating)
  lambda <- seq(2,8,0.25)

  # errors for every parameter
  errors <- sapply(lambda, function(l){
    f <- train1 %>%
           group_by(movieId) %>%
           summarise(b_f = sum(rating-avg1)/(n()+l))

    k <- train1 %>% left_join(f, by = "movieId") %>% group_by(userId) %>%
           summarise(b_k=sum(rating-(avg1 + b_f))/(n()+l))

    pred_fk <- test1 %>%
                 left_join(f, by="movieId") %>%
                 left_join(k, by = "userId") %>%
```

```
                mutate(pred = avg1 + b_f + b_k) %>% pull(pred)

    RMSE(pred_fk, test1$rating)
  })

  c(err=min(errors), lamb=lambda[which.min(errors)])

})

# the best parameter after 5-fold cross validation
best_lambdas <- data.frame(t(best_lambde))
l <- best_lambdas %>% arrange(err) %>% pull(lamb) %>% .[1]
l
```

```
## [1] 4.75
```

We apply the selected lambda on our regularization model and use penalty term (lambda) for user effect as well. The final model looks like this:

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

Now we can predict rating using final model. Do we improve our results?

```
# apply the best parameter on trainig set and test set
f <- train %>%
       group_by(movieId) %>%
       summarise(b_i = sum(rating - mu_hat) / (n() + l))

k <- train %>%
       left_join(f, by = "movieId") %>%
       group_by(userId) %>%
       summarise(b_u = sum(rating - (mu_hat + b_i)) / (n() + l))

# constructing predictors
pred_fk <- test %>%
             left_join(f, by = "movieId") %>%
             left_join(k, by = "userId") %>%
             mutate(pred = mu_hat + b_i + b_u) %>% pull(pred)

# RMSE for regularization model
error4 <- RMSE(pred_fk, test$rating)

#table of errors for comparation
tbl <- bind_rows(tbl, tibble(model="regularization (train/test)", RMSE = error4))
tbl %>% knitr::kable()
```

model	RMSE
basic model (train/test)	1.0607045
movie effect (train/test)	0.9437144
movie-user effect (train/test)	0.8661625
regularization (train/test)	0.8655426

Since we are satisfied with the error produced, we will apply the final model on the entire new_edx set to obtain predictions for the ratings of the given movies and compare the final predicted values with the actual values from the validation set.

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

  f <- new_edx %>%
           group_by(movieId) %>%
           summarise(b_i = sum(rating - mu_hat)/(n()+l))

  k <- new_edx %>%
           left_join(f, by = "movieId") %>%
           group_by(userId) %>%
           summarise(b_u = sum(rating - (mu_hat + b_i)) / (n() + l))

  pred_fk <- validation %>%
              left_join(f, by = "movieId") %>%
              left_join(k, by = "userId") %>%
              mutate(pred = mu_hat + b_i + b_u) %>%
              pull(pred)

  final_error <-  RMSE(pred_fk, validation$rating)

  tbl <- bind_rows(tbl, tibble(model = "regularization (edx/validation)", RMSE = final_error))
  tbl %>% knitr::kable()
```

model	RMSE
basic model (train/test)	1.0607045
movie effect (train/test)	0.9437144
movie-user effect (train/test)	0.8661625
regularization (train/test)	0.8655426
regularization (edx/validation)	0.8648201

We were able to get a satisfactory RMSE of 0.8648201.

CONCLUSION

In this project we create a *movie recomendation system* using 10M version of the MovieLens dataset. The final model we used is:

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

where $Y_{u,i}$ is rating for movie i by user u, $\epsilon_{u,i}$ is independent errors sampled from the same distribution centered at 0, μ is the true rating for all movies and b_i and b_u is the movie and user effect, respectively. Thus we achieved an RMSE of 0.8648201. Further work on the model may take into account the effects of genre and time, or applied factors analysis.