Movielens Project

HarvardX: PH125.9x Capstone

Andres Gomez

May 31, 2020

Contents

Introduction

The goal of this project is the following: to provide a prediction for movie ratings given a set of features. In other words, it will give an answer to the question: "what rating would a user give to a certain movie?", or what it is known as a "recommendation system".

To do this, we are given a dataset, which we split into training and test sets. We use the training set to develop an algorithm that will answer the question, and we use the test set to validate the results. In summary this is done to get an accurate measure of the performance of our machine learning algorithm, which can't be done if we use only one dataset for both training and validation.

More specifically, we will use the RMSE on the test set. RMSE stands for residual mean squared error, and we can interpret it as a standard deviation, meaning, it is the typical error we make when predicting a movie rating.

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

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

If the RMSE is larger than 1, then it means our typical error is larger than 1 "star", which is not good (ratings are measured in "stars" from 0.5 to 5, with increments of 0.5).

The "movielens" dataset (the data we are given), contains 6 variables: userid, movieid, rating, timestamp, title of the movie, and genre. So, a row in the dataset, is the rating that one user gave to one movie. One user can rate more than one movie, and it's expected that in general one movie is rated by several users.

The dataset contains 10,000,054 observations. We will use 90% of it to train the model and the remaining 10% to validate. We'll reference them as edx dataset (with 9,000,055 observations) and validation dataset (with 999,999 observations). These partitions of the datasets were done just using the code provided by the instructors, as shown in the next section.

This document has 4 parts:

1. Data preparation
2. An analysis section that explains the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and the modeling approach
3. A results section that presents the modeling results and discusses the model performance
4. A conclusion section that gives a brief summary of the report, its limitations and future work

Data Preparation

```
####################################
# Create edx set and validation set
####################################

# 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
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
```

```
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## -- Attaching packages ------------------------------------------------------------------- tidyverse 1
```

```
## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ---------------------------------------------------------------------- tidyverse_conflic
## 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 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")
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) # if using version lower than R 3.6.0:
set.seed(1, sample.kind = "Rounding")
```

```
## 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)
```

edx dataset is for training and validation dataset is for test, so to avoid any type of confusion I will rename them:

```
train_set <- edx
test_set <- validation
```

Methods/Analysis

After running the code provided in the Appendix, we get two datasets: "edx" used to train the models, and "validation" used to test or validate the results of the models.

First of all, let's do some data exploration over the "edx" dataset, to see how the data that we are going to use for training the models looks like.

Data Exploration

As we already shared, the edx dataset has 9,000,055 observations and 6 variables. We can get that with the following command in R:

```
dim(edx)
```

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

How many times was 3 given as a rating?

```
edx %>% filter(rating == 3) %>% tally()
```

```
##         n
## 1 2121240
```

How many different movies and different users are there in the edx dataset?

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

We also know that Pulp Fiction is the movie with the greatest number of ratings and that 4 is the most often given rating

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

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                     count
##      <dbl> <chr>                                                     <int>
##  1     296 Pulp Fiction (1994)                                       31362
##  2     356 Forrest Gump (1994)                                       31079
##  3     593 Silence of the Lambs, The (1991)                          30382
##  4     480 Jurassic Park (1993)                                      29360
##  5     318 Shawshank Redemption, The (1994)                          28015
##  6     110 Braveheart (1995)                                         26212
##  7     457 Fugitive, The (1993)                                      25998
##  8     589 Terminator 2: Judgment Day (1991)                         25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (19~ 25672
## 10     150 Apollo 13 (1995)                                          24284
## # ... with 10,667 more rows
```

```
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5) %>%
    arrange(desc(count))
```

```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

What is the average of ratings?

```r
mean(edx$rating)
```

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

... and the histogram (frequency) of ratings looks like this:

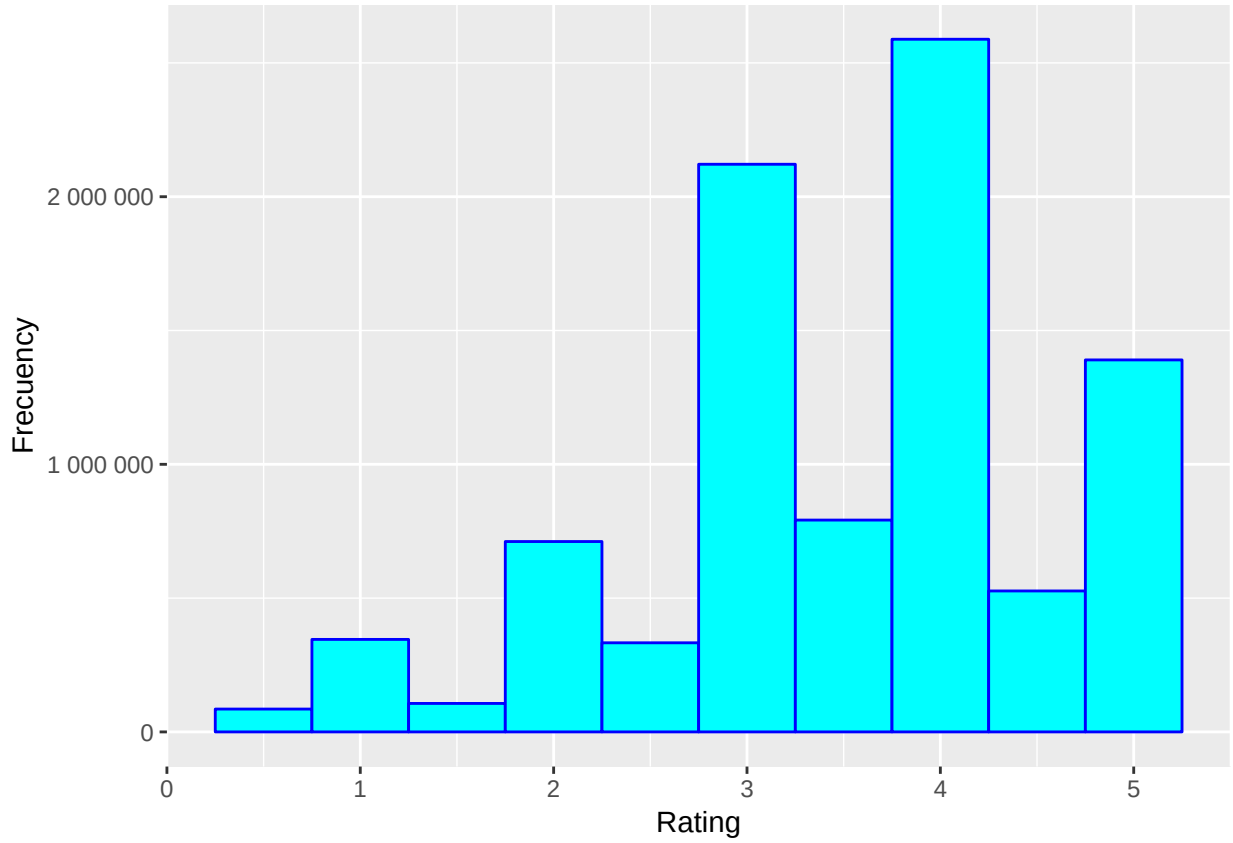
```r
library(tidyverse)
library(dslabs)
library(scales)
```

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```r
edx %>% ggplot()+
  geom_histogram(mapping=aes(x=rating), binwidth = 0.5, color="blue", fill="cyan")+
  xlab("Rating")+ylab("Frecuency")+scale_y_continuous(labels = number)
```



Modeling Approach

The approach we will use is going to be trying different model options, measuring the RMSE and keeping the model with RMSE low enough that keeps us satisfied with the results. In this academic exercise we have some guidance: our goal will be to have a model with RMSE < 0.86490. If we reach that measure, we'll consider the goal achieved.

Results

Let's start building a model that predicts ratings.

We define the RMSE in R as follows:

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

First Approach: a "naive" model

For our first try, let's use a very simple approach: we will predict the same rating for all movies.

Our model would be something like this:

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

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

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

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

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

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

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

Remember that we are evaluating the model on the validation set (test). As we can see, this result is far away from our goal/threshold, but it will help us building the model.

Second Approach: movie bias

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

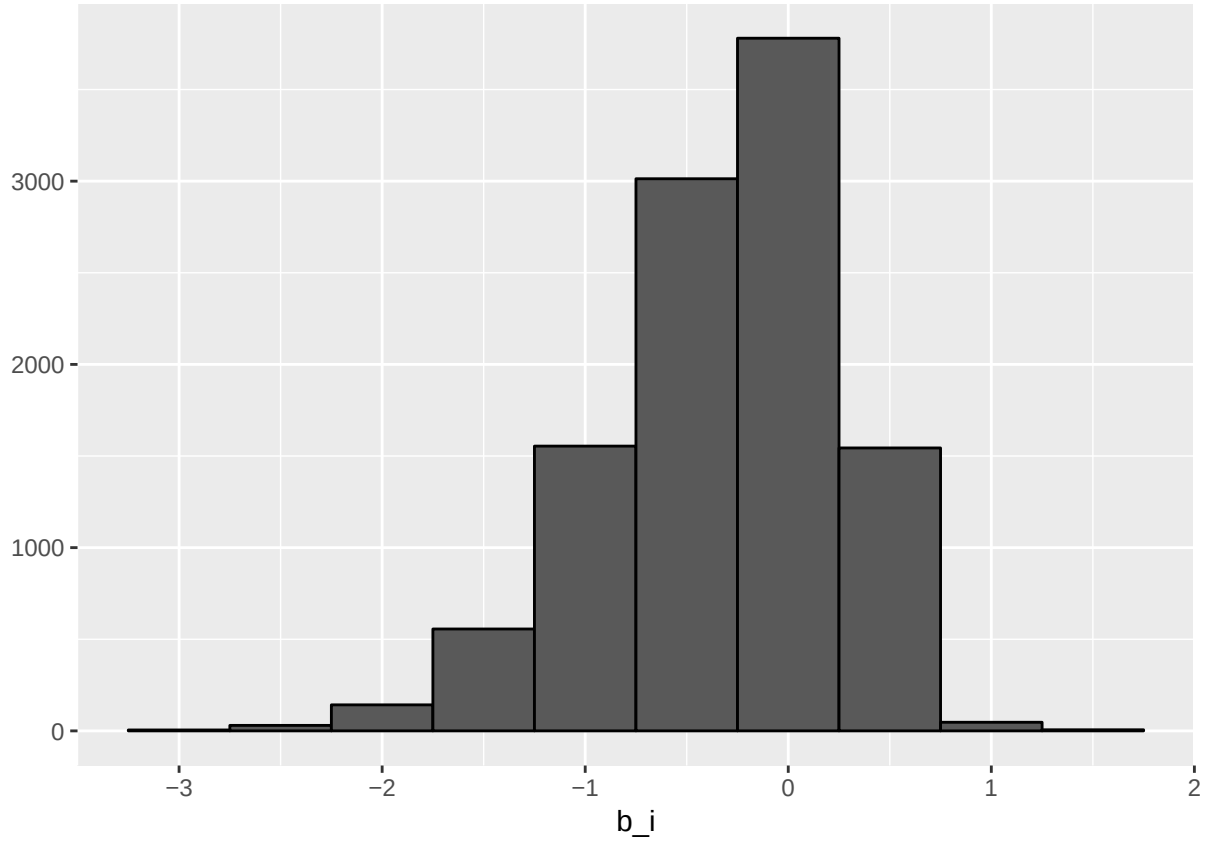
$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

So now let's model it and see if how results reduce our RMSE.

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

We can see that these estimates vary substantially:

```
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Remember $\hat{\mu} = 3.5$ so a $b_i = 1.5$ implies a perfect five star rating.

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

```
predicted_ratings <- test_set %>%
left_join(movie_avgs, by = "movieId") %>%
mutate(pred = mu + b_i) %>%
pull(pred)
Movie_Effect_rmse <- RMSE(predicted_ratings, test_set$rating)
Movie_Effect_rmse
```

```
## [1] 0.9439087
```

There is improvement, but we are still far from our goal.

Third Approach: user effects (with previous movie effects)

We can see that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model can be:

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

where b_u is a user-specific effect

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

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

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
RMSE_Movie_Users <- RMSE(predicted_ratings, test_set$rating)
RMSE_Movie_Users
```

```
## [1] 0.8653488
```

We are close to our goal (RMSE<0.8649), but still we need some improvement in our model.

Penalized least squares (Regularization)

The idea behind regularization is to constrain the total variability of the effect sizes. Then, instead of minimazing the least squares, we will minimize the following, adding a penalty λ.

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

Using calculus we know that the values of b_i that minimize this equation are:

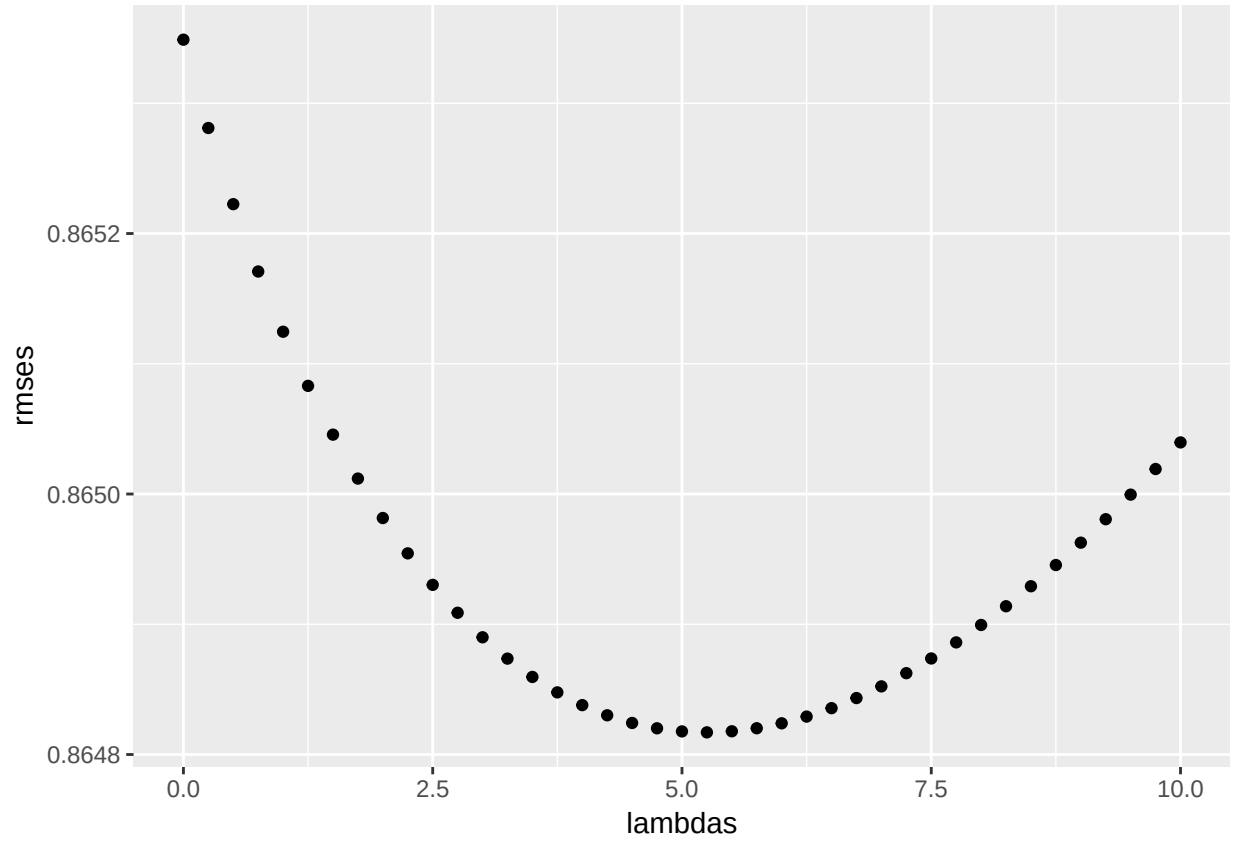
$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}\left(Y_{u,i} - \hat{\mu}\right)$$

We will use cross-validation to pick a λ:

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
```

```
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-test_set %>%
                      left_join(b_i, by = "movieId") %>%
                      left_join(b_u, by = "userId") %>%
                      mutate(pred = mu + b_i + b_u) %>%
                      pull(pred)
return(RMSE(predicted_ratings, test_set$rating))
})
```

```
qplot(lambdas, rmses)
```



For this model, the optimal λ is:

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

```
## [1] 5.25
```

With corresponding RMSE:

```
min(rmses)
```

```
## [1] 0.864817
```

We have reached our goal of obtaining a model that predicts movies ratings, with an RMSE lower than 0.8649.

Conclusions

In an effort to build a recommendation system for movies, we have tried different models to predict movie ratings.

Starting with a "naive" approach, we introduced movie effects, as well as user effects in a linear model looking to minimize the RMSE. We have found that using "Penalized least squares" we can reach an RMSE = 0.864817, which is slightly lower than our target 0.8649.