Report - HarvardX:PH125.8x - Data Science: Machine Learning

Jose Carlos Prado Jr

31/01/2019

Introduction

Recommendation systems use ratings that users have given items to make specific recommendations to users.

Companies like Amazon that sell many products to many customers and permit these customers to rate their products are able to collect massive data sets that can be used to predict what rating a given user will give a specific item. Items for which a high rating is predicted for specific users are then recommended to that user.

Netflix uses recommendation systems to predict how many stars a user will give a specific movie.

Here we provide the basics of how these recommendations are predicted, motivated by some of the approaches taken by the winners of the Netflix challenge.

On October 2006, *Netflix* offered a challenge to the data science community. Improve our recommendation algorithm by 10% and win a $1 million.

For this project, the goal is creating a movie recommendation system using the MovieLens dataset. The version of movielens included in the dslabs package is just a small subset of a much larger dataset with millions of ratings. It is posible to find the entire latest MovieLens dataset with **dslabs** package. It was created our own recommendation system using all the tools we have shown you throughout the courses in this series. It was used the 10M version of the MovieLens dataset http://files.grouplens.org/datasets/movielens/ml-10m.zip to make the computation a little easier.

First, the 10M movielens subset file was downloaded. For this purpose it was created temp file. The file downloaded was zipped file, so it was necessary unzip the file and preparing the *movielens* datase, joinning the separate files and setting the column names.

Bellow we can download this subset and unzip the file. The zipped file has two separate files **ratings.dat** and **movies.dat**.

```r
setwd("~/Library/Mobile Documents/com~apple~CloudDocs/R/curso/capstone")
Sys.setenv(PATH = paste(Sys.getenv("PATH"), "~/Aplicativos/Tex/", sep=.Platform$path.sep))
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")

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("ml-10M100K/movies.dat"), "\\::", 3)
```

The *movies* dataset has 3 columns. We set the column names respectively and convert the first column type from *character* to *numeric*.

```r
colnames(movies) <- c("movieId", "title", "genres")
typeof(movies[1,1])
```

```
## [1] "character"
```

```r
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId)[movieId]))
```

Movielens is now the dataset joining *movies* and *ratings* and has the columns *userId*, *movieId*, *rating*, *timestamp*, *title*, *genres*.

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

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

We can see that the movie lens table is tidy formant and contains thousands of rows.Each row represents a rating given by one user to one movie.

We use *set.seed* function for setting the seed of R's random number generator, which is useful for creating simulations or random objects that can be reproduced.

```r
set.seed(1)
```

So let's create create two datasets, the *training* and the *test* ones. We call the training dataset as **edx** with 90% of the data and the test dataset as **validation** with the 90% data. The test set is used to assess the accuracy of the models we implement, just like in other machine learning algorithms.

We use the caret package using this code.

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

To make sure we don't include users and movies in the test set that do not appear in the training set, we removed these using the semi_join function

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

Add rows removed from validation set back into edx set

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

The next steps were inspect the *training* and *test* data sets, build the recommendation system, calculate the residual mean squared error (RMSE) for each model and choose the best RMSE for predicting the rating system in the validation dataset.

Methods

It is posible inspect the edx dataset with some functions and plots.

```
head(edx)
```

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

```
dim(edx)
```

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

edx has 9000055 lines and 6 columns, wich represents 90% of the *movielens* 10M subset.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##
##                              title                             genres
##  Pulp Fiction (1994)              :  31362   Drama                :  733296
##  Forrest Gump (1994)              :  31079   Comedy               :  700889
##  Silence of the Lambs, The (1991):  30382   Comedy|Romance       :  365468
##  Jurassic Park (1993)             :  29360   Comedy|Drama         :  323637
##  Shawshank Redemption, The (1994):  28015   Comedy|Drama|Romance:  261425
##  Braveheart (1995)                :  26212   Drama|Romance        :  259355
##  (Other)                          :8823645   (Other)              :6355985
```

Rating has Min. 0.5, mean 4 and Max 5 ratings.

We can see the number of unique users that provide ratings and for how many unique movies they provided them using this code

```
edx %>%
    summarize(n_users = n_distinct(userId),
              n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

If we multiply those two numbers, we get 746.087.406 movies but the data set has 9.000.063 rows.

```r
n_distinct(edx$movieId)* n_distinct(edx$userId)
```

```
## [1] 746087406
```

This implies that not every user rated every movie. So we can think of this data as a very large matrix with users on the rows and movies on the columns with many empty cells.

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

```
## # A tibble: 5 x 2
##   title                            count
##   <fct>                            <int>
## 1 Pulp Fiction (1994)              31362
## 2 Forrest Gump (1994)              31079
## 3 Silence of the Lambs, The (1991) 30382
## 4 Jurassic Park (1993)             29360
## 5 Shawshank Redemption, The (1994) 28015
```

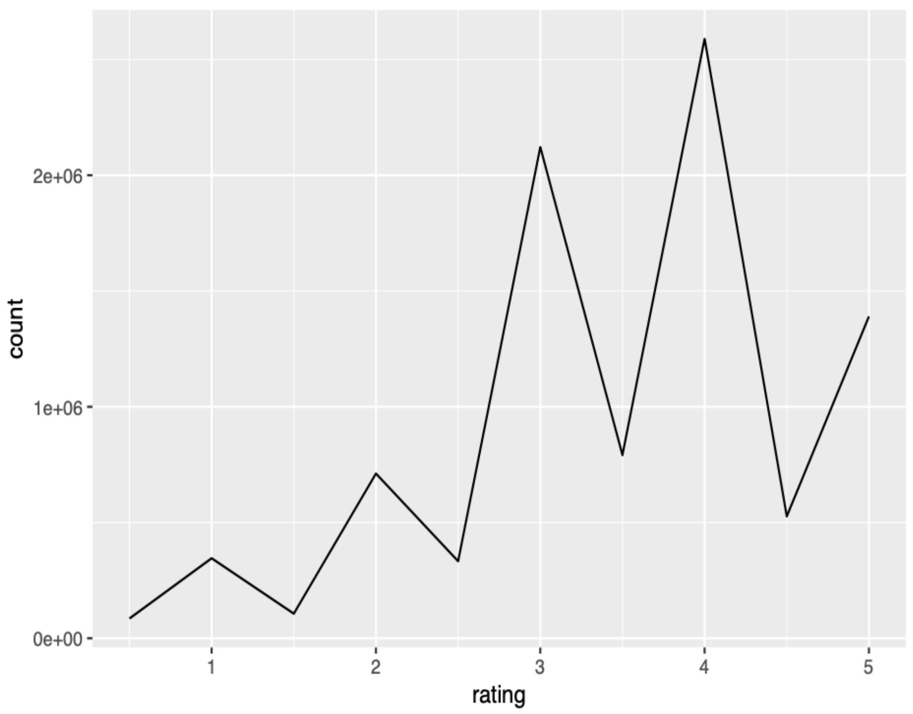
As we can see, *Pulp Fiction (1994)* has the greatest number of ratings [31362 ratings], followed by *Forrest Gump (1994)* [31079 ratings].

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

The five most given ratings in order from most to least are *4*, *3*, *5*, *3.5* and *2*.

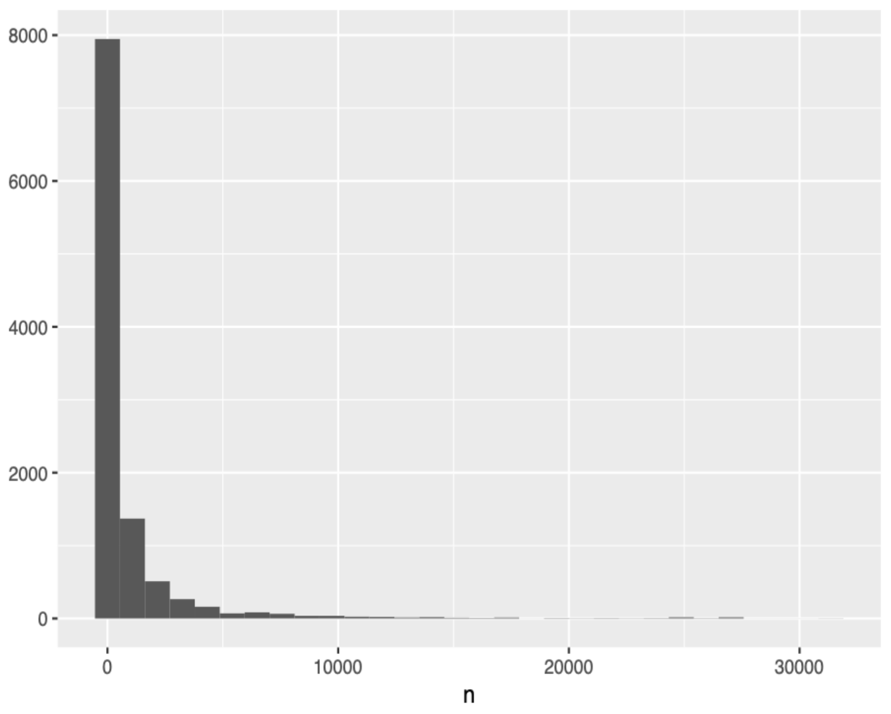
```r
edx %>% group_by(rating) %>%
    summarize(count=n()) %>%
    ggplot(aes(x=rating, y=count))+
    geom_line()
```



In general, half star ratings are less common than whole star ratings (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.)

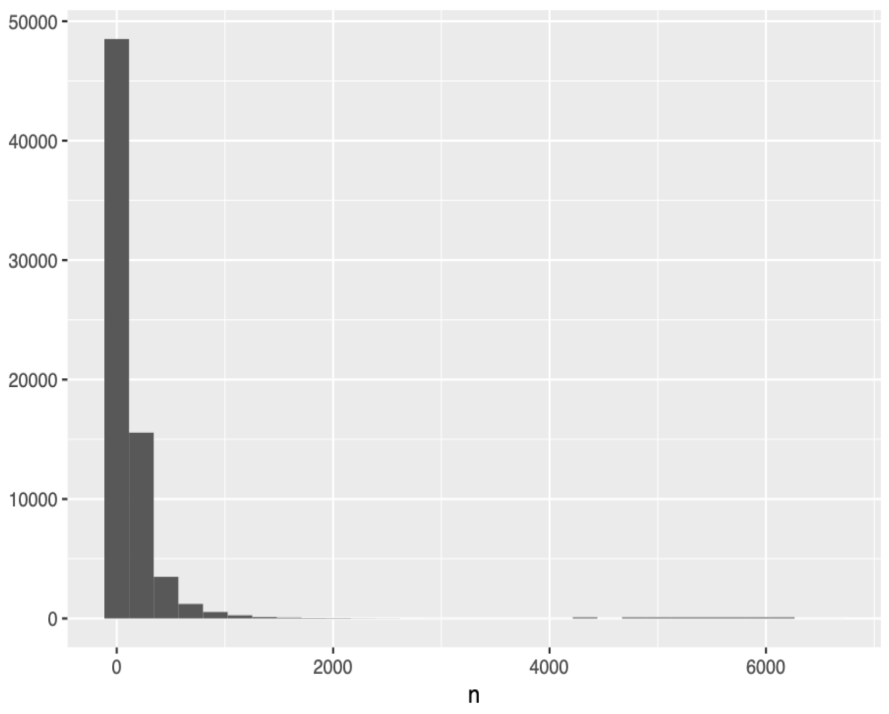
Let's look at some of the general properties of the data to better understand the challenge. The first thing we notice is that some movies get rated more than others. Here's the distribution.

```
edx %>% group_by(movieId) %>%
  summarize(n=n()) %>%
  qplot(n, data= ., geom="histogram")
```



This should not surprise us given that there are blockbusters watched by millions and artsy independent movies watched by just a few. A second observation is that some users are more active than others at rating movies.

```
edx %>% group_by(userId) %>%
  summarize(n=n()) %>%
  qplot(n, data= ., geom="histogram")
```



To compare different models or to see how well we're doing compared to some baseline, we need to quantify what it means to do well. We need a loss function. The Netflix challenge used the typical error and thus decided on a winner based on the residual mean squared error on a test set.

So if we define yui as the rating for movie i by user u and y hat ui as our prediction, hen the residual mean squared error is defined as follows.

Here n is a number of user movie combinations and the sum is occurring over all these combinations. We can interpret the residual mean squared error similar to standard deviation.

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

It is the typical error we make when predicting a movie rating. If this number is much larger than one, we're typically missing by one or more stars rating which is not very good. So let's quickly write a function that computes this residual means squared error for a vector of ratings and their corresponding predictors. It's a simple function that looks like this.

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

If RMSE>1, so it is not a good prediction. The smallest number of RMSE, better is the prediction.

Building the Recommendation System

The Netflix challenge winners implemented two general classes of models. One was similar to k-nearest neighbors, where you found movies that were similar to each other and users that were similar to each other. The other one was based on an approach called matrix factorization.

That's the one we we're going to focus on here. So let's start building these models. Let's start by building the simplest possible recommendation system.

We're going to predict the same rating for all movies, regardless of the user and movie. So what number should we predict? We can use a model-based approach. A model that assumes the same rating for all movies and all users, with all the differences explained by random variation would look something like this.

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

Here epsilon represents independent errors sampled from the same distribution centered at zero, and mu represents the true rating for all movies and users.

We know that the estimate that minimizes the residual mean squared error is the least squared estimate of mu.

And in this case, that's just the average of all the ratings. We can compute it like this.

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

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

That average is 3.51.So that is the average rating of all movies across all users. So let's see how well this movie does. We compute this average on the training data. And then we compute the residual mean squared error on the test set data.

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

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

So we're predicting all unknown ratings with this average. We get a residual mean squared error of about 1.05. That's pretty big. Now note, if you plug in any other number, you get a higher RMSE. That's what's supposed to happen, because we know that the average minimizes the residual mean squared error when using this model.And you can see it with this code.

```
predictions <- rep(2.5, nrow(validation))
RMSE(validation$rating, predictions)
```

```
## [1] 1.46641
```

Now because as we go along we will be comparing different approaches, we're going to create a table that's going to store the results that we obtain as we go along. We're going to call it RMSE results. It's going to be created using this code.

```
rmse_results <- data_frame(method="Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

Now the rmse result is 1.06 for 'Just the average'. We know from experience that some movies are just generally rated higher than others. So we can augment our previous model by adding a term, b i, to represent the average rating for movie i.In statistics, we usually call these b's, effects.

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

However, in this particular situation, we know that the least squared estimate, b hat i, is just the average of yui minus the overall mean for each movie, i. So we can compute them using this code.

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

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

We can use this code and see that our residual mean squared error did drop a little bit.

```r
predicted_ratings <- mu + validation %>%
    left_join(movie_avgs, by='movieId')%>%
    .$b_i

model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087

We already see an improvement. Now can we make it better? How about users? Are different users different in terms of how they rate movies? We include a term, bu, which is the user-specific effect.

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

we will compute our approximation by computing the overall mean, u-hat, the movie effects, b-hat i, and then estimating the user effects, b u-hat, by taking the average of the residuals obtained after removing the overall mean and the movie effect from the ratings yui.

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

And now we can see how well we do with this new model by predicting values and computing the residual mean squared error.

```r
predicted_ratings <- validation %>%
    left_join(movie_avgs, by='movieId')%>%
    left_join(user_avgs, by='userId')%>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8292477

We see that now we obtain a further improvement. Our residual mean squared error dropped down to about 0.82.

Regularization permits us to penalize large estimates that come from small sample sizes. It has commonalities with the Bayesian approaches that shrunk predictions. The general idea is to add a penalty for large values of b to the sum of squares equations that we minimize.

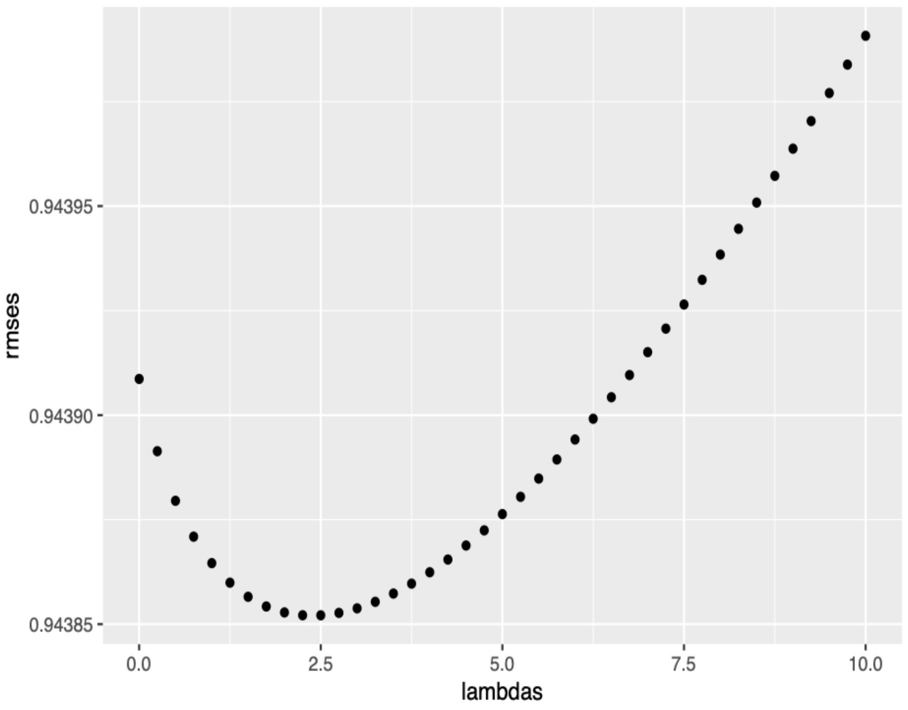
```
lambda <- 3
mu <- mean(edx$rating)
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i=sum(rating-mu)/(n()+lambda), n_i = n())

predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred=mu + b_i) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse))

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8292477
Regularized Movie Effect Model	0.9438538

Now note that lambda is a tuning parameter. We can use cross-fertilization to choose it. We can use this code to do this.

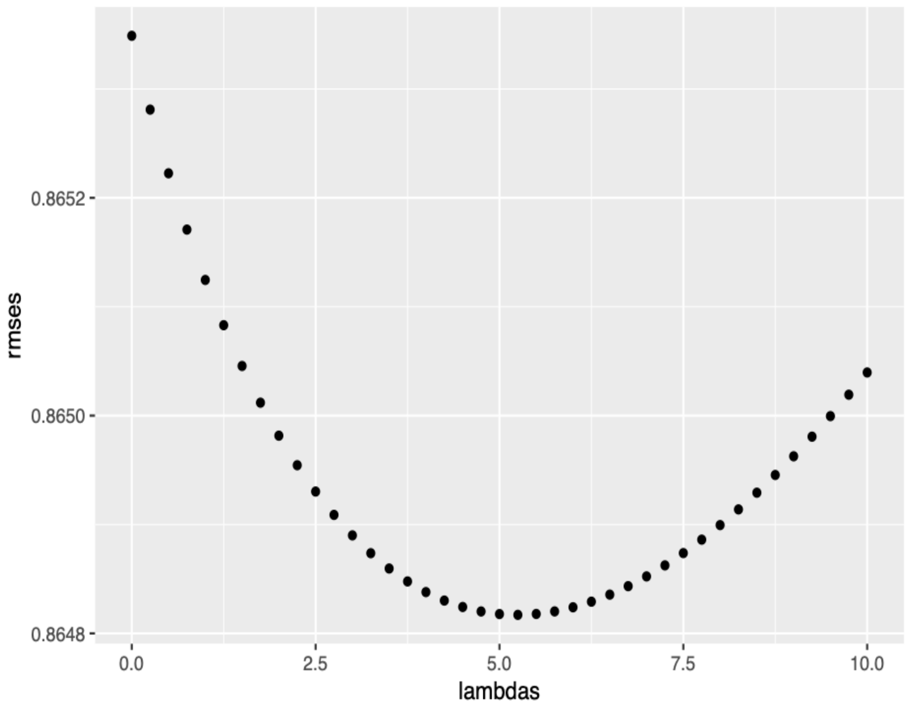
```
lambdas <- seq(0, 10, 0.25)
mu <- mean(edx$rating)
just_the_sum <-edx %>%
    group_by(movieId) %>%
    summarize(s=sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
    predicted_ratings <- validation %>%
      left_join(just_the_sum, by='movieId') %>%
      mutate(b_i = s/(n_i+l)) %>%
      mutate(pred = mu + b_i) %>%
      .$pred
    return(RMSE(predicted_ratings, validation$rating))
```

```
  })

qplot(lambdas,rmses)
```



The estimates that minimizes can be found similarly to what we do previously. Here we again use cross-validation to pick lambda. The code looks like this, and we see what lambda minimizes our equation.

```
lambdas <- seq(0, 10, 0.25)
 rmses <- sapply(lambdas, function(l){
   mu <- mean(edx$rating)

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

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

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

   return(RMSE(predicted_ratings, validation$rating))
})

qplot(lambdas,rmses)
```



For the full model including movie and user effects, the optimal lambda is 5.25.

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

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

Results

Now we know which is the best y for the model. Let's calculate the *Regularized Movie Effect Model - adjusted*.

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

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

b_u <- edx %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

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

model_3_rmse_adjusted <- RMSE(predicted_ratings, validation$rating)
model_3_rmse_adjusted
```

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

Now the RMSE for the *Regularized Movie Effect Model - adjusted* is **0.86**

Conclusion

For this project, it was posible to create a movie recommendation system using the MovieLens dataset. It was posible to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. We could try some machine learning techniques. With the final RMSE, we can use this predicted ratings and export the file `submission.csv` with this code:

```
validation2 <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    select(userId, movieId, pred) %>%
    mutate(rating=pred)%>%
    select(-pred)

write.csv(validation2,
            "submission.csv", na = "", row.names=FALSE)

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

Note that the *rating* column was replaced by the *pred* one which menas that now we have the predicted rating acoording to the final model.