DSCapstone Report

Carlos Aguilar

2/11/2020

Introduction

This project builds a movie recommendation on the 10M MovieLens Dataset using five different approaches. The dataset contains data collected by the Movie Lens project and has around 10 million movie ratings given by anonymized users, the movie's genres, title, and release year.

The objective of the project is to produce a system that predicts the rating that a user will give to a particular movie with an RMSE (Root Mean Square Error) below to 0.86490. To achieve this, we will analyze five methods that incrementally consider more information.

The first method would be to predict the simple mean for every movie. The second will add a variable to handle movie bias. The third will add a control for user bias. The last two methods will consider movie and user bias, respectively, but regularizing the results to penalize higher residuals from few ratings.

We are going to use the following packages to perform the analysis

1. Tidyverse
2. Caret
3. Data.table

Methods

First, we are going to download and process the data using the following code (provided by the Harvard X'sData Science Profesional Certificate)

```
# 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 ---------------------------------------------------------------------------
```

```
## <U+2713> ggplot2 3.2.0     <U+2713> purrr   0.3.2
## <U+2713> tibble  2.1.3     <U+2713> dplyr   0.8.3
## <U+2713> tidyr   0.8.3     <U+2713> stringr 1.4.0
## <U+2713> readr   1.3.1     <U+2713> 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 from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
# 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 <- 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(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, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

The result from the code above is two datasets: **edx** and **validation**, the first one will be used to test the different approaches. Once we have a satisfactory result, we will use the validation data set to compute the RMSE.

We will calculate the RMSE with the following function:

```
# Function for calculating the RMSE error
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Also, to compare the results, we are going to create a results table and a helper function to record the results quickly

```
library(tidyverse)
results <- tibble(method=character(), RMSE=double())
add_result <- function(method, RMSE) {
  results <- bind_rows(results, tibble(method = method, RMSE = RMSE))
}
```

We are going to divide the **edx** dataset into a training and a testing partition to check the accuracy before actually testing the results on the **validation** dataset. We use a seed to get consistent results on the process and use the createPartition method from the caret package to create a train set with 90% of the records on the edx data set. Finally, we remove all the entries on the test set that didn't make the cut on the train set.

```
library(caret)
set.seed(2, sample.kind="Rounding") # please use set.seed(2) if using RStudio 3.5 or below

test_index <- createDataPartition(edx$rating, times=1, p=0.1,list=FALSE)
test_set <- edx[test_index, ]
train_set <- edx[-test_index, ]

# Remove movies that don't appear on the train set from the test set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Now we are ready to train our models. The first one will be to calculate the mean of the train set and predict that rating for all the movies on the test set.

```r
first_prediction <- mean(train_set$rating)
average_rmse <- RMSE(test_set$rating, first_prediction)

results <- add_result("Average Model", average_rmse)
average_rmse
```

```
## [1] 1.06031
```

As expected, the result is way above the 0.86490 checkpoint. So, we will move forward and account that some movies are rated differently than others. For this, we need to add a new parameter to the model to consider the movie bias. The parameter `b_i` averages the difference between a rating and the overall mean for every movie. Now, we predict the movie ratings by adding the overall mean mu to the new parameter `b_i`.

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

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

movie_effect_rmse <- RMSE(second_prediction, test_set$rating)

results <- add_result("Movie Effect Model", movie_effect_rmse)
movie_effect_rmse
```

```
## [1] 0.9436226
```

The movie effect got us closer to the aimed outcome. Next, we are introducing a new parameter that takes into account that some users tend to rate movies higher than others. For each user, we will add the parameter `b_u` as the average rate the user gives minus the movie bias `b_i`. Therefore, our prediction will be the overall average `mu`, plus the movie effect `b_i`, plus the user effect `b_u`.

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

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

user_effect_rmse <- RMSE(user_predicted_ratings, test_set$rating)

results <- add_result("User + Movie Effect Model", user_effect_rmse)
user_effect_rmse
```

```
## [1] 0.8663835
```

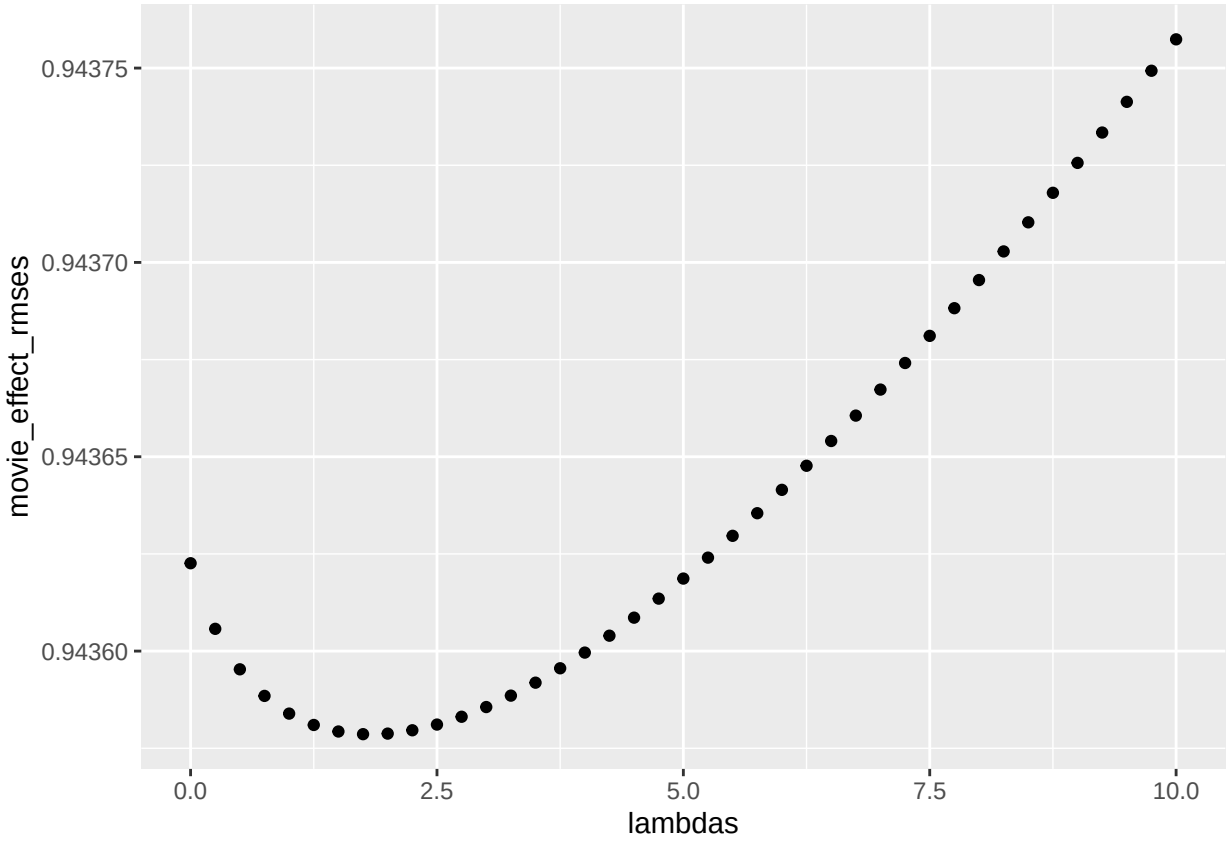
Adding the user effect improved the model considerably, but we need to get a little bit better before we try our methods on the **validation** set. To get this push forward, we will explore the concept of regularization.

As analyzed on the Data Science: Machine Learning course, some of our more significant errors come from movies with few ratings and users that rated few movies. Therefore, we are going to add a penalizing term to reduce the impact of those observations over the prediction model. To simplify this concept, we are applying it first to the movie models, and then we will use it to the user model.

The main idea is to instead of dividing `b_i` by the number of ratings, when calculating the mean, we divide the sum the ratings by the number of ratings plus a control parameter `lambda`. The `lambda` parameter will reduce the value of `b_i` when the number of ratings is small and will be negligible when the number of ratings is high. To find the best possible `lambda` we will use cross-validation. The plot confirms that we are picking an optimized value for `lambda`.

```
# 4. Adding Regularization for the movie effect
lambdas <- seq(0, 10, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
movie_effect_rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, movie_effect_rmses)
```




```
lambdas[which.min(movie_effect_rmses)]
```

```
## [1] 1.75
```

```
results <- add_result("Regularized Movie Effect Model", min(movie_effect_rmses))
min(movie_effect_rmses)
```

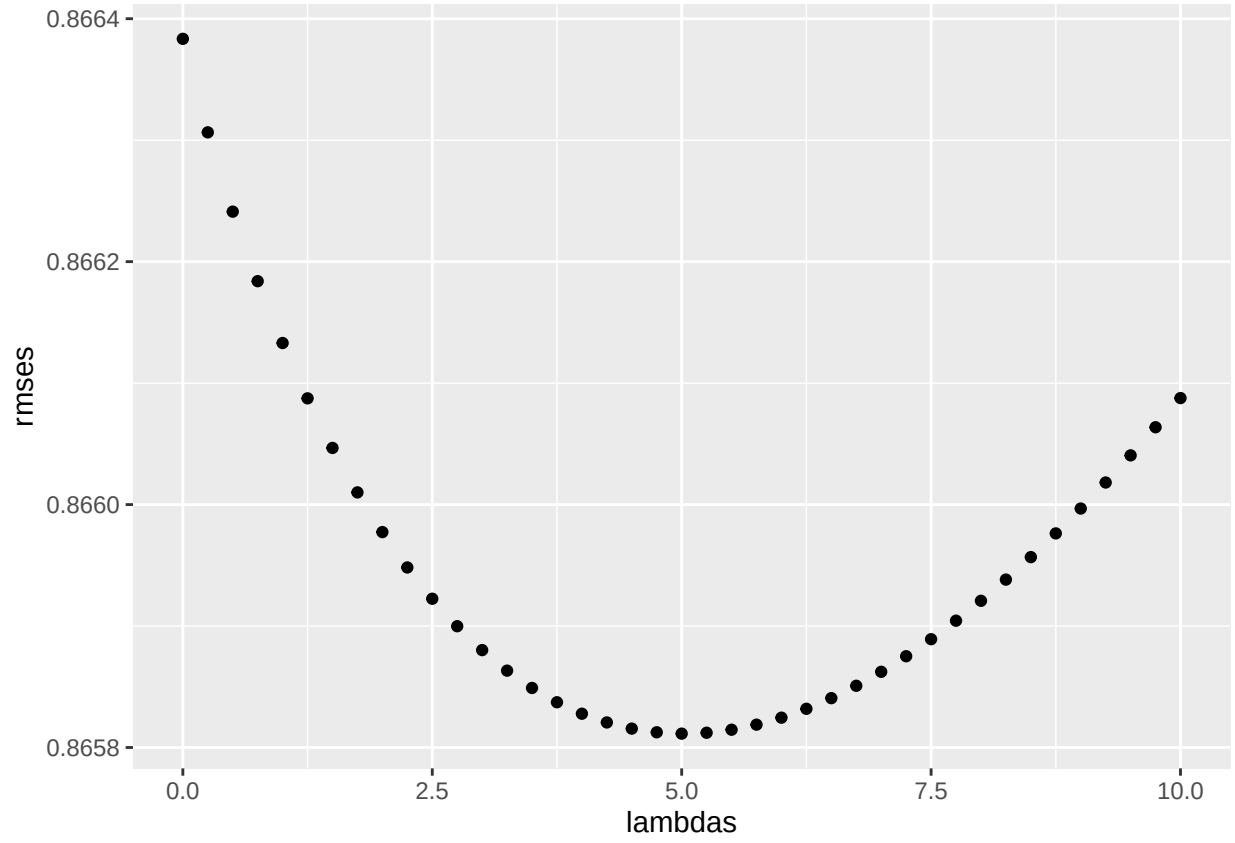
```
## [1] 0.9435786
```

We got a small improvement from the second approach. The next step is to regularize the user effect parameter `b_u` by adding a `lambda` parameter at the mean calculation. Once again, we use cross-validation to find the best value for the `lambda` parameter.

```
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) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```




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

results <- add_result("Regularized User + Movie Effect Model", min(rmses))
min(rmses)
```

```
## [1] 0.8658114
```

The regularization got us pretty close to our aimed threshold; it is time to test the approach on the **validation** set. We will use the same `lambda` calculated on the previous step.

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

```r
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) %>%
  .$pred
final_rmse <- RMSE(predicted_ratings, validation$rating)

results <- add_result("Final Model RMSE", final_rmse)
final_rmse
```

```
## [1] 0.8648177
```

Results

The following table resumes the results we got from the five approaches and the RMSE run on the **validation** set:

```
## # A tibble: 6 x 2
##   method                                 RMSE
##   <chr>                                 <dbl>
## 1 Average Model                         1.06
## 2 Movie Effect Model                    0.944
## 3 User + Movie Effect Model             0.866
## 4 Regularized Movie Effect Model        0.944
## 5 Regularized User + Movie Effect Model 0.866
## 6 Final Model RMSE                      0.865
```

This table shows us that the final model regularizing both the movie effect `b_i` and the user effect `b_u` is enough to achieve the desire RMSE.

Conclusions

The process of getting a smaller RMSE showed us that a better understanding of the data allows us to get better results predicting its behavior. The knowledge that some movies may have similar rates and that users tend to rate similarly improved our model from a random prediction algorithm to a stable predicting model. For this particular project, we achieved the goal by adding particularities of the dataset to the model, which worked for our dataset but may not hold true for different information.

Some limitations of the techniques used are that they fail to take into account some structure of the data that may lead us to better results. Some users may rate a particular type of movie better than others and maybe even reluctant to assess positively other kinds of films. Some users may like a specific actor and may bias their ratings depending on the movies in which that actor appears.

For future work, a Principal Component Analysis can be included in the final model, extracting valuable information about the patterns in the users scoring. Also, comparing the results with recommender packages like the **recommenderlab** could provide some insights into how to improve the prediction model.