Capstone Project Report

Heloise Auger

April 3rd 2019

HarvardX: PH125.9 - Data Science: Capstone

This report presents the methodology and results used as part of HarvardX course PH125.9 - Data Science: Capstone.

Find this project online at: https://github.com/HeloiseA/MovieLens_Project

Introduction

Recommandation systems are what allow companies like Netflix to suggest movies to their audience based on users' past activities. For example, a person who has watched and highly rated many romantic comedies can expect to see this genre of movie appearing in their suggested "watch later" list. Using data science, it is possible to approximate the rating a user might give to a certain movie, and thus help deciding wether to suggest it to them or not.

For this project, a subset of the MovieLens dataset, the MovieLens 10M dataset, will be used to predict the ratings a sample of users would give to certain movies. This dataset may be found at: https://grouplens.org/datasets/movielens/10m/.

After exploring the dataset and possible techniques, a method was chosen to generate ratings predictions. This method rests upon the minimization of a Residual Mean Squared Error (RMSE) loss function by a model composed of various parameters. An iterative approach was taken to add effect parameters and a regularization parameter to the model. The following sections detail this analysis and the results it produced.

Methods

The road to a minimized RMSE value was not a linear one. In this section, we present the exploration and chosen methods used in order to optimize the predictions of movie ratings.

Creating the training and testing datasets

The course instructors provided a segment of code in order to download and clean the MovieLens 10M dataset. Additionally, this code separated the dataset into two subsets: the `edx` dataset, for training, and the `validation` dataset, for testing.

Creating the RMSE function

The evaluation of the predictions was instructed to be made via an RMSE loss function, defined as:

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

with $\hat{y}_{u,i}$ and $y_{u,i}$ being the predicted and actual ratings, and N, the number of possible combinations between user u and movie i.

This function evaluates the square root of the mean of the differences between true and predicted ratings and is defined in the code in order to avoid repeating it for every new model.

```
RMSE <- function(validation, y_hat){
  sqrt(mean((validation - y_hat)^2))
}
```

The variable `validation` will contain the true ratings from `validation$rating`, and the variable `y_hat` will contain our corresponding predictions.

Data exploration and visualization

In order to begin analyzing the data, it is capital to know what format our datasets are in. This section presents steps that do not appear in the final R script, but were integral to the completion of this project. To begin, the following code was used to glance into the training set.

```
nrow(edx)
```

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

```
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
```

The number of rows in the dataset as well as its columns' name are now known. The same step was applied for the `validation` set. After some exploration, it does not seem like the datasets require anymore wrangling to be tidy and ready for analysis. With this insight, a first approach to ratings prediction was attempted.

A linear model was defined which took into account user IDs and movie IDs, converted to factors. The fit returned by the `lm` function was expected to be used within the `predict` function in order to generate the ratings predictions.

```
cols <- c("userId", "movieId")
edx[cols] <- lapply(edx[cols], factor)
validation[cols] <- lapply(validation[cols], factor)
fit <- lm(rating ~ userId + movieId, data = edx)
y_hat <- predict(fit, validation)
```

This approach failed due to the lack of RAM on this student's computer. An attempt was made to use a subset of the original datasets in order to run the code.

```
mini_edx <- edx[sample(nrow(edx), 10000), ]
mini_validation <- validation[sample(nrow(validation), 1000), ]
```

But this approach could not be completed for the same reasons.

Model 1: the simplest recommandation system

From the exploration of the data, it appears that in order to retrieve movie ratings predictions from the voluminous MovieLens 10M dataset, an approach that is less memory-hungry is required.

Knowing this, the first model was designed to be exceedingly simple: the `validation` ratings were approximated by the mean of all ratings in `edx`, which translates to this formula:

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

Where μ is the average of all ratings in `edx` and $\varepsilon_{u,i}$ corresponds to the independent errors sampled from the same distribution. Knowing these errors are centered around zero, the previous equation translates to the following code.

```
y_hat1 <- mean(edx$rating)
rmse1 <- RMSE(validation$rating, y_hat1)
```

Model 2: adding a movie effect

Model 2 builds upon model 1 by adding a parameter accounting for the ratings of indivual movies. This term, centered around zero, increases or decreases the predicted rating based on the mean of the differences between the average rating and a movie's ratings. The model becomes:

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

where b_i is the new movie effect parameter. In the following code, the data frame `movie_avgs` carries the movie IDs and their b_i parameters, to be included in the calculation of new predictions.

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

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

y_hat2 <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

rmse2 <- RMSE(validation$rating, y_hat2)
```

Model 3: adding a user effect

Some movie watchers are more critical than others. A user-dependant term is thus added to the previous model. The process is very similar to that of the movie effect. The new term is again centered around zero, and increases or decreases the prediction based on the mean difference of the average rating mu and movie parameter b_i from each user's rating. The model's equation becomes:

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

where b_u is the user effect parameter. The following code implements this model. Along with `movie_avgs` containing the b_i parameters, we now have `user_avgs` which contains the b_u parameters.

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

y_hat3 <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

rmse3 <- RMSE(validation$rating, y_hat3)
```

Using regularization to improve on Model 3

Results show that the third model performed adequately, but it is obviously not perfect. Our system accounts for every movie, meaning it is in part based on the ratings of obscure movies with few ratings, which introduces uncertainty. How can we modelize the popularity of a movie within our system?

In order to reduce the unpopular movies' effect towards zero, regularization principles will be applied to the third model. A penalty, driven by the number of ratings for a movie and regularization parameter lambda, is added to the model. Instead of b_i, the regularized movie effect is defined as:

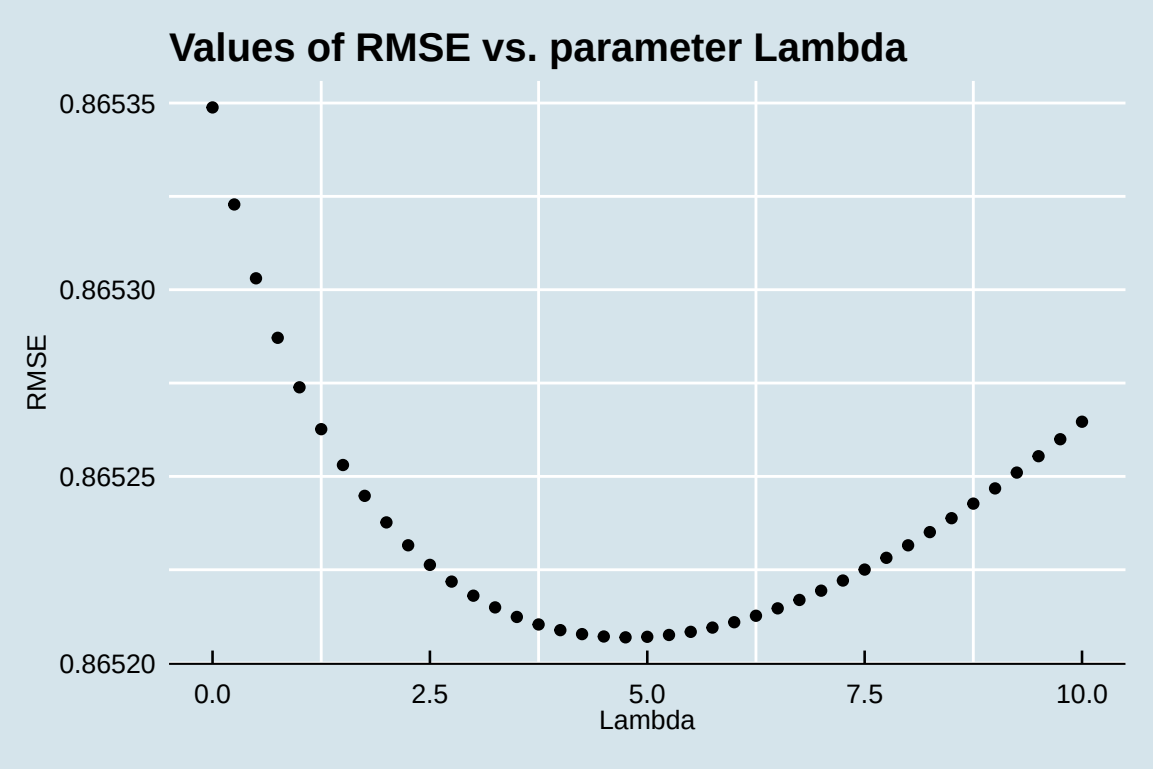
$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

where n_i is the number of ratings per movie. In addition, parameter lambda may be optimized by testing an array of values and keeping the one which minimizes the RMSE. In the code below, the `sapply` function returns a vector of RMSE values in order to determine the optimal value of lambda.

```
lambdas <- seq(0, 10, 0.25)

sums_movie <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  y_hat4 <- validation %>%
    left_join(sums_movie, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(validation$rating, y_hat4))
})
```

To visualize the optimal value for lambda, a plot of its values against their corresponding RMSE was produced.

```
library(ggplot2)
library(ggthemes)
library(grid)

p <- qplot(lambdas, rmses, main="Values of RMSE vs. parameter Lambda",
           xlab="Lambda", ylab="RMSE")
p + theme_economist() +
  theme(panel.grid.major = element_line(color = "white", size = 0.5),
        panel.grid.minor = element_line(color = "white", size = 0.5))
```



Thus, we can see that the RMSE is minimized for a value of lambda of:

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

Results

In this section, the RMSE returned by all of the previously defined models are presented. The optimal value of regularization parameter lambda is also included.

```
cat("RMSE from Model 1: ", rmse1)
```

```
## RMSE from Model 1:  1.061202
```

```
cat("RMSE from Model 2: ", rmse2)
```

```
## RMSE from Model 2:  0.9439087
```

```
cat("RMSE from Model 3: ", rmse3)
```

```
## RMSE from Model 3:  0.8653488
```

```
cat("Minimum RMSE from model 3 with regularized movie effect parameter: ", final_rmse)
```

```
## Minimum RMSE from model 3 with regularized movie effect parameter:  0.8652069
```

As for parameter lambda, a value of:

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

yielded a minimized RMSE.

As expected, the RMSE decreased as the model grew more complex. In future works, other parameters could be added to this system; one based on movie genre comes to mind.

Conclusion

This report aimed to present the process of completing the MovieLens Capstone Project as part of course PH125.9. The goal of this project was to present a method to minimize an RMSE loss function of the true and predicted ratings of a subset of the MovieLens dataset.

The MovieLens 10M dataset was downloaded, cleaned, and separated into training and testing subsets. The RMSE loss function was defined. After some exploration, three models resulting in decreasing RMSE were presented. Finally, a variation on the third model introducing a regularization parameter returned a minimized RMSE value.