MovieLens Report

Carlos Barraza

6/25/2021

Introduction - the MovieLens dataset

The **MovieLens** dataset was collected and made available by GroupLens Research, a research lab in the Department of Computer Science and Engineering at the **University of Minnesota**.

The dataset is comprised of a number of film reviews by various users, including information like *userId, movieId, movie title, rating, timestamp*, and *genres* associated with each movie. The dataset used for this project is the **MovieLens 10M Dataset**, which includes 10,000,000 movie ratings by 72,000 users, and was released in January 2009.

The aim of this project is to generate a machine learning algorithm that will efficiently predict the value of a movie's rating using predictors, while minimizing the value of the *root mean squared error (RMSE)* between predicted values and real values.

The key steps performed to achieve this goal were:

- **Cleaning the data.** The *MovieLens* database had to be formatted in order to be efficiently manageable in R.
- **Algorithm exploration.** Some basic machine learning algorithms were applied to the data to test the time it would take to perform these calculations, using the `train` function.
- **Generation of predictions.** Once the appropiate algorithm was chosen, the necessary calculations were performed in order to obtain the fit, the predictions of the `validation` set and the RMSE value.

Methods and analysis

Data cleanup

A number of packages were installed in order to have access to the neccesary functions to perform all the calculations.

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
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
```

```r
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
```

```r
library(tidyverse)
library(caret)
library(data.table)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

The file containing the 10M MovieLens dataset was then downloaded from the internet and read into R, as `ratings` and `movies`. Both datasets were then joined to form the `movielens` variable.

```
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")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

The `movielens` data was then partitioned to create an `edx` set and a `validation`set. The `edx` set would be used for training and testing, while the `validation` set would be used to obtain the final result of the RMSE.

```
# 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)`
```

```
## 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)
```

Algorithm exploration

A number of different training methods using the `train` function were tested in the `edx` dataset, including `glm`, `knn`, `naive_bayes`, and the `randomForest` function as well.

```
fit_rf <- randomForest(rating ~ ., data = train)
predict_rf <- predict(fit_rf, validation)
```

```
fit_glm <- train(rating ~ ., method = "glm", data = edx)
predict_glm <- predict(fit_glm, validation)
```

```
fit_knn <- train(rating ~ ., method = "knn", data = edx)
predict_knn <- predict(fit_knn, validation)
```

However, these calculations took an incredibly high amount of time, taking hours to compute, and failing to do so due to a lack of computing power from the laptop being used.

Therefore, it was decided that it would be best to take a more rudimentary approach to the problem, like a custom linear model would be.

Generation of predictions

The linear model was started. First, `mu`, the average of the rating values in the training set, needed to be calculated.

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

To implement regularization, a lambda that would minimize the RMSE had to be chosen amongst a range of values.

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

A function was developed that would take each lambda and perform calculations with it. This function would create the `b_i` and `b_u` terms for the linear model, accounting for the movie and user effect, respectively.

```
rmses <- sapply(lambdas, function(l) {

  # Compute b_i term, the coefficient for the movie effect.
  movie_effects <- edx %>% group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + l))

  # Compute b_u term, the coefficient for the user effect.
  user_effects <- edx %>% left_join(movie_effects, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n() + l))

  # Compute the rating predictions in the validation set.
  test_pred <- validation %>% left_join(movie_effects, by = "movieId") %>%
    left_join(user_effects, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  #Return the RMSE value between predictions and real values.
  return(RMSE(test_pred, validation$rating))

})
```

A plot was constructed in order to visualize the behavior of the lambdas and the RMSE values obtained.

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

The value of lambda that would minimize the RMSE was determined using the following code.

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

Finally, the whole process in the function was repeated, now using the ideal lambda value, to obtain the funal value of the RMSE.

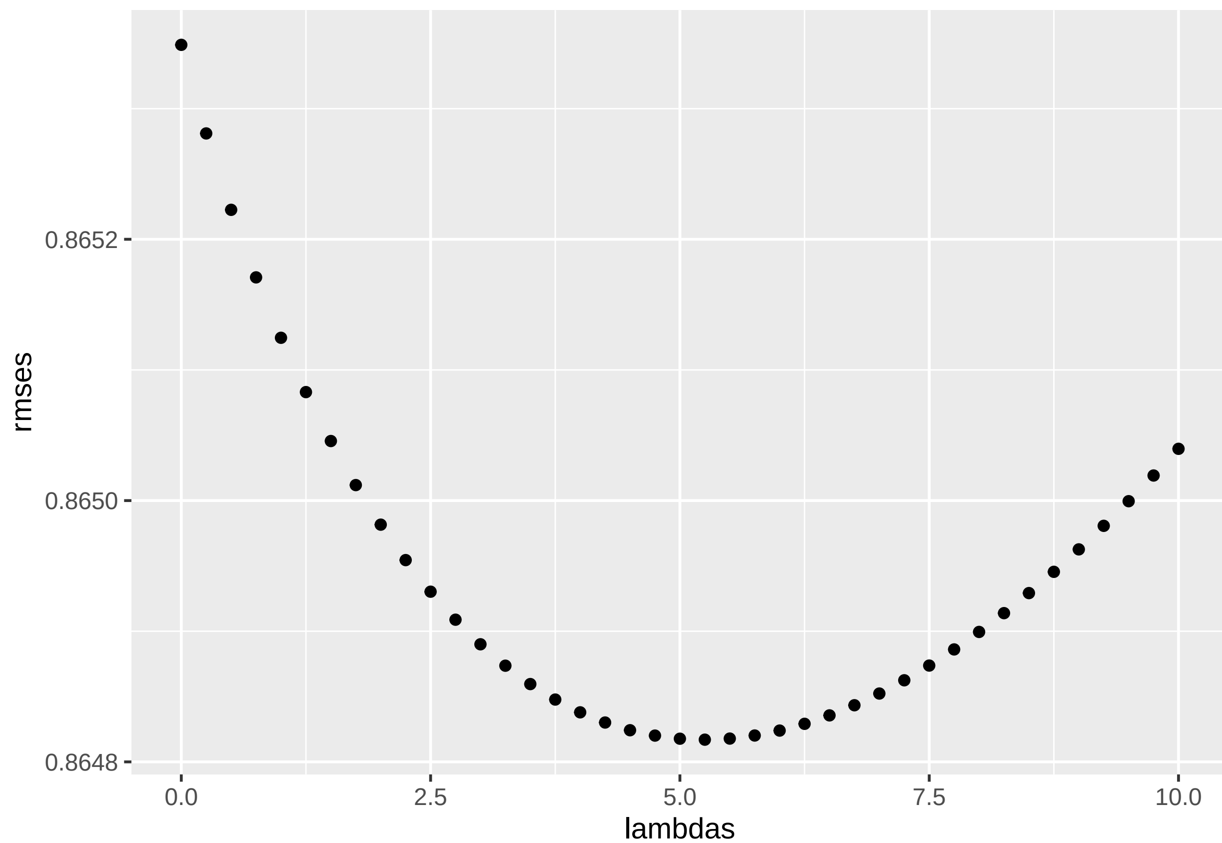
```
# Repeat the process using the ideal lambda.
lf <- 5.25

# Compute b_i term, the coefficient for the movie effect.
movie_effects <- edx %>% group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + lf))

# Compute b_u term, the coefficient for the user effect.
user_effects <- edx %>% left_join(movie_effects, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + lf))

# Compute the rating predictions in the test set.
test_pred <- validation %>% left_join(movie_effects, by = "movieId") %>%
  left_join(user_effects, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

```
RMSE(test_pred, validation$rating)
```

Results

The following figure describes the values of lambdas against the RMSEs obtained with the resulting linear model.

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



The final value of the RMSE obtained with the applied linear model was the following.

```
RMSE(test_pred, validation$rating)
```

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

This value indicates that, on average, our machine learning algorithm will be 0.864817 rating points away from the true value.

Conclusion

This report describes the steps neccesary for the development process of a machine learning algorithm that will be useful to predict movie rating values using the data in the **10M MovieLens dataset** as predictors, as well as the final results obtained when comparing the predicted values with the real values as a RMSE.

Some limitations encountered while doing this project include the insufficient computing power that the laptop possessed. This made it impossible to test out other more powerful functions in the `caret` package, such as the `train` function, cross-validation and using `tuneGrid` and `trControl` parameters.

This report will serve as a basis to record the knowledge obtained in the Data Science courses, and hopefully to aid other people in case they turn to this report as a source of knowledge.