Movielens Project

Zak Khayat - khayat61

10/31/2020

Overview

This report documents the details of creating a movie recommendation system including the best possible RMSE value, using the MovieLens dataset.The recommendation system was designed using the 10M version (a small subset) of the Movielens dataset, and the tools we have learned throughout the courses in this series.

The main focus of this project is to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

After setting up the dataset, the algorithm is developed using the edx set, and then validate it using the validation set (the final hold-out test set) as if they were unknown. The validation set is only used to measure the RMSE for the final algorithm.

To avoid over training, the edx set was split into separate training and test sets to design and test the algorithm and to come up with best possible optimized parameters.

RMSE will be used to evaluate how close the predictions are to the true values in the validation set (the final hold-out test set).

The final algorithm will be built using the optimized parameters, edx dataset and validation set.

Method/Analysis

After setting the split dataset that will be used, for each set of algorithm parameters being considered, there will be an estimate of the RMSE. the RMSE value will be calculated for each parameter considered. Then, the parameters with the smallest RMSE will be chosen to be used in the final model.

The process steps that were used for this project were as follows:

Step 1: Document the edx set and validation set that was created per the course exercise:

```
library(tidyverse)
```

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

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

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

```r
library(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
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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

```r
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
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)
```

Step 2: split the edx(training set) into training and test sets:

```
edx_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-edx_index,]
edx_test <- edx[edx_index,]

edx_validation <- edx_test %>%
   semi_join(edx_train, by = "movieId") %>%
   semi_join(edx_train, by = "userId")
removed <- anti_join(edx_test, edx_validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx_train <- rbind(edx_train, removed)
```

Step 3: Develop the algorithm model and enhance it through various parameters optimization including Regularization technique, as shown in the Results section

Step 4: analyze and determine the best possible parameters that achieve the best RMSE value for the final model

Step 5: Select the best lambda value to use to determine the final mode and its RMSE value as shown in the Results section

Results

This section shows the results of developing the model through adding/testing various parameters to reach the most optimized RMSE value for the final model

Using the average mean(mu)

y(u,i) = mu + e(u,i)

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

```
## [1] 3.512509
```

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

```
## [1] 1.061135
```

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

```
## Warning: ‘data_frame()‘ is deprecated as of tibble 1.1.0.
## Please use ‘tibble()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_warnings()‘ to see where this warning was generated.
```

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

method	RMSE
Just the average	1.061135

adding the movie avg (b_i) factor to the model and calculate the RMSE

y(u,i) = mu + b_i + e(u,i)

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

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

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

model_bi_rmse <- RMSE(predicted_ratings, edx_validation$rating)
RMSE_results <- bind_rows(RMSE_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_bi_rmse ))
RMSE_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0611352
Movie Effect Model	0.9441568

adding the user avg(b_u) factor to the model and calculate the RMSE

y(u,i) = mu + b_i + b_u + e(u,i)

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

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

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


model_bu_rmse <- RMSE(predicted_ratings, edx_validation$rating)
RMSE_results <- bind_rows(RMSE_results,data_frame(method="Movie + User Effects Model",
     RMSE = model_bu_rmse ))

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

method	RMSE
Just the average	1.0611352
Movie Effect Model	0.9441568
Movie + User Effects Model	0.8659736

The above estimated parameters do not take into account some of data that has large estimates, which are formed using small sizes of ratings for a movie. To reduce these obseure and noisy etimates, we want to use Regularization technique to estimate the b_i and b_u parameters.

Regularization permits penalizing large estimates that are formed using small sizes. The penalty, Lambda, is effective when the number of ratings (n) for movie(i) is small, then we want the estimate of b_i_lambda to shrink toward 0. The larger the lambda, the more we shrink as per the following equation:

b_i_lambda_hat = 1/lambda+n(i) X sum(Y(u,i)- mu)

Let's select a value for lambda(say 4) to estimate b_reg_i and see if we can achieve improvement of RMSE.

```
lambda <- 4
mu <- mean(edx_train$rating)
movie_reg_avgs <- edx_train %>%
      group_by(movieId) %>%
summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

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

model_lambda_rmse <- RMSE(predicted_ratings, edx_validation$rating)
RMSE_results <- bind_rows(RMSE_results,
                  data_frame(method="Regularized Movie Effect Model",
                                  RMSE = model_lambda_rmse ))
RMSE_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0611352
Movie Effect Model	0.9441568

method	RMSE
Movie + User Effects Model	0.8659736
Regularized Movie Effect Model	0.9441383

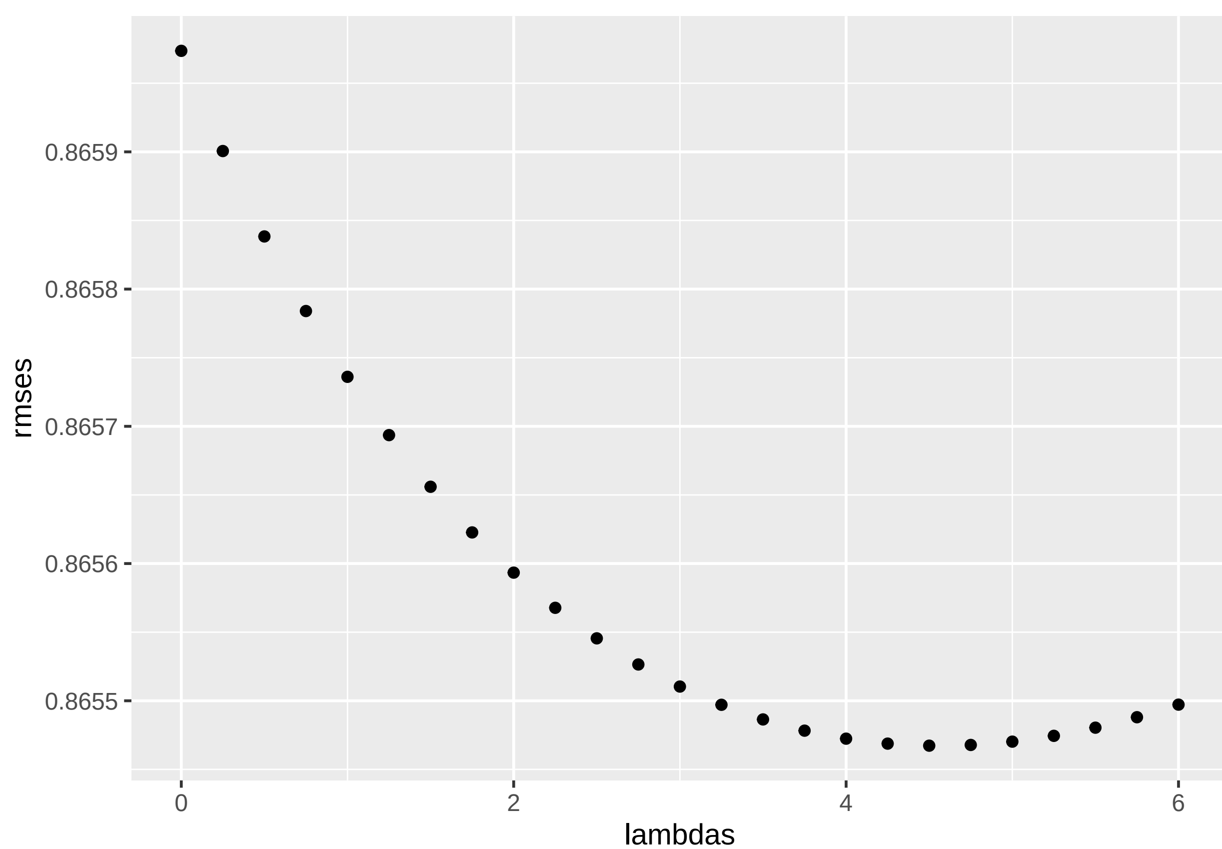
So, lambda is basically a tuning parameter that we need to select to give us the best RMSE value for our final model.Based on our learning, we will use Cross Validation to select the best lambda value for best RMSE value:

```
lambdas <- seq(0, 6, 0.25)
rmses <- sapply(lambdas, function(l){
    mu <- mean(edx_train$rating)
    b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
     group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
 predicted_ratings <-
edx_validation %>%
left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>%
 mutate(pred = mu + b_i + b_u) %>%
  .$pred
  return(RMSE(predicted_ratings, edx_validation$rating))})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)
```

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



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

```
## [1] 4.5
```

```
RMSE_results <- bind_rows(RMSE_results,
                    data_frame(method="Regularized Movie + User Effect Model",
                               RMSE = min(rmses)))
RMSE_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0611352
Movie Effect Model	0.9441568
Movie + User Effects Model	0.8659736
Regularized Movie Effect Model	0.9441383
Regularized Movie + User Effect Model	0.8654673

Based on the Cross Validation for lambda as shown above, Lambda=4.5, will give us the best value for lowest RMSE. So, let's use 4.5 for Lambda to design our final model using the edx training set and test set (Validation set) to see the best RMSE value for the Final Model

```
lambdas <- 4.5
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))
 })
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
RMSE_results<-bind_rows(RMSE_results,data_frame(method="Final Model",RMSE=rmses))
RMSE_results%>%knitr::kable()
```

method	RMSE
Just the average	1.0611352
Movie Effect Model	0.9441568

method	RMSE
Movie + User Effects Model	0.8659736
Regularized Movie Effect Model	0.9441383
Regularized Movie + User Effect Model	0.8654673
Final Model	0.8648242

Conclusion

In conclusion, the movie and user effects helped bringing the RMSE below 1, but the Regularization technique helped getting the most optimized parameters for our final model. RMSE value of 0.8648242 is well under 1, and it meets the target value of below 0.8649 for this project.

Other options or techniques we could use in the future is to include/try genres parameter estimates to further lower the RMSE value.