Movielens_RMSE

Marian Dumitrascu

March 17, 2019

Movielens RMSE Calculation

Introduction

This document contains the script for computing RMSE for Movielens dataset. It was prepared for Capstone project *PH125.9 Data Science: Capstone*. The results are published in **MovieLens_Report_v08.Rmd** report, in **RMSE Results** section.

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################

# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)
library(tidyr)
library(dplyr)

############################################################################
# data loading using a local file. I commented this before submitting
############################################################################

# movielens <- read_csv("ml-10M100k/edx_full.dat", col_names = TRUE, n_max = 1000000)
# movielens <- read_csv("ml-10M100k/edx_full.dat", col_names = TRUE)

#############################################################################################
# original data loading. I uncomment this for final submission and PDF generation
#############################################################################################

# 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 <- read.table(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)
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)
edx <- rbind(edx, removed)

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

Create an extra validation set out of edx where will perform cross validation

```
edx_validate_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train_set <- edx[-edx_validate_index, ]
temp <- edx[edx_validate_index, ]

# Make sure userId and movieId in edx_validation_set set are also in edx_train_set set
# remove rows in temp that userId or movieId not found in edx_train_set
edx_validation_set <- temp %>%
  semi_join(edx_train_set, by = "movieId") %>%
  semi_join(edx_train_set, by = "userId")

# get removed rows and put them edx_train_set
removed <- anti_join(temp, edx_validation_set, by = c("movieId", "userId"))
edx_train_set <- rbind(edx_train_set, removed)

# remove varriables to clean up memory and not polute the environment
rm(edx_validate_index, temp, removed)
```

RMSE Using Movie Effects Only

```
# global average rating for all movies
rate_avg <- mean(edx_train_set$rating)

# compute RMSE using just the global average, as a reference to the next computations of RMSE
# attach rate_avg to validation dataset and create predicted_ratings
```

```
predicted_ratings <- validation %>%
  left_join(edx, by=c("userId", "movieId")) %>%
  summarize(rating_pred = rate_avg)

# compute RMSE
rmse <- RMSE(predicted_ratings$rating_pred, validation$rating)

# create a data frame to store RMSE, and store rmse
model_results <- data_frame(Method = "Global Rating Average", RMSE = rmse)

# predicted_ratings is a  data frame with average rating by movie (b_movie) centered around rate_avg
predicted_ratings_movies <- edx %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - rate_avg))


# attach *b_movie* determined in the training set edx to the validation set and compute the RMSE
predicted_ratings <- validation %>%
  left_join(predicted_ratings_movies, by = "movieId") %>%
  mutate(rating_pred = rate_avg + b_movie)

rmse <- RMSE(predicted_ratings$rating_pred, validation$rating)


# remove in case we run this chunk multiple times
if (any(model_results$Method =="Movie Effect"))
{
  model_results <- model_results[-which(model_results$Method =="Movie Effect"),]
}

model_results <- bind_rows(model_results,
                           data_frame(
                             Method = "Movie Effect",
                             RMSE = rmse))

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

Method	RMSE
Global Rating Average	1.0612019
Movie Effect	0.9439087

RMSE Using User Effect and Movie Effect

b_user is average rating by user centered around *rate_avg* Take into consideration the movies effect.

```
# determine user efects (b_user), centered around total rating mean (rate_avg)
# take into consideration movie effect (b_movie)
predicted_ratings_users <- edx %>%
  left_join(predicted_ratings_movies, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_user = mean(rating - rate_avg - b_movie))
```

```r
# compute predicted rating by attaching the two average tables (users and movies)
# to the validation set

predicted_ratings <- validation %>%
  left_join(predicted_ratings_movies, by = "movieId") %>%
  left_join(predicted_ratings_users, by = "userId") %>%
  mutate(rating_pred = rate_avg + b_movie + b_user)

rmse <- RMSE(predicted_ratings$rating_pred, validation$rating)

# remove in case we run this chunk multiple times
if (any(model_results$Method =="Movie + User Effect"))
{
  model_results <- model_results[-which(model_results$Method =="Movie + User Effect"),]
}

model_results <- bind_rows(model_results,
                           data_frame(
                             Method = "Movie + User Effect",
                             RMSE = rmse))

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

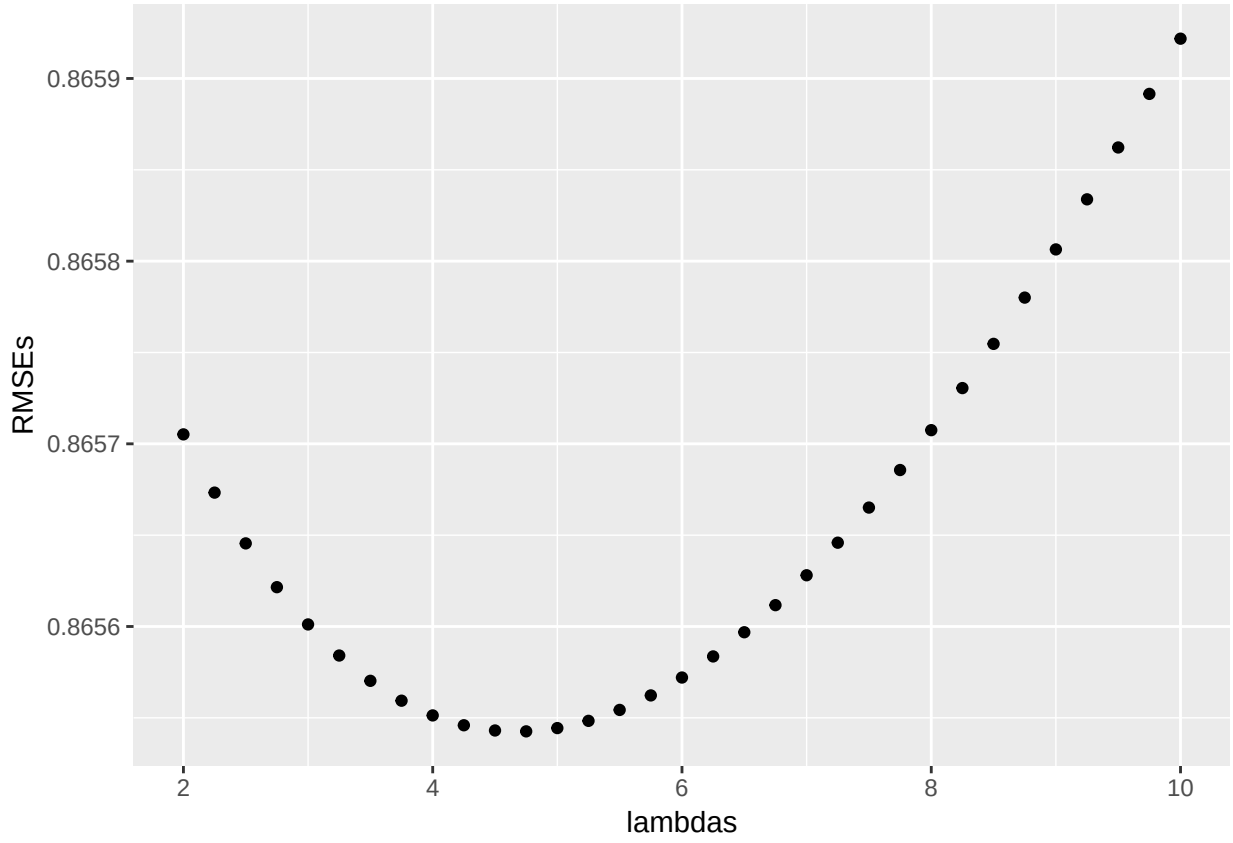
Method	RMSE
Global Rating Average	1.0612019
Movie Effect	0.9439087
Movie + User Effect	0.8653488

RMSE With Regularization

If we select only movies that have a large number of ratings, we observe that RMSE improves. The conclusion is that we would need to apply regularization, that would penalize estimates based on sample size. We define the function *RMSE_regularized* that compute the RMSE for a given *lambda* and a pair of trainning/validation data sets. We will optimize *lambda* using *edx_training_set* and *edx_validation_set*, then compute the final RMSE agains *edx* and *validation*

```r
lambdas <- seq(2, 10, 0.25)

# function used for lambda optimization.
RMSE_regularized <- function(lambda, a_train_set, a_validation_set){

    # movie effect regularized
    b_movies <- a_train_set %>%
      group_by(movieId) %>%
      summarize(b_movie = sum(rating - rate_avg)/(n() + lambda))

    # user effect regularized, also considering the movie effect regularized
    b_users <- a_train_set %>%
      left_join(b_movies, by="movieId") %>%
      group_by(userId) %>%
```

```
      summarize(b_user = sum(rating - b_movie - rate_avg)/(n() + lambda))

    # join movie and user effects tables with the test and compute predicted rate
    predicted_ratings_set <- a_validation_set %>%
      left_join(b_movies, by = "movieId") %>%
      left_join(b_users, by = "userId") %>%
      mutate(rating_pred = rate_avg + b_movie + b_user)

    # compute RMSE
    RMSE(predicted_ratings_set$rating_pred, a_validation_set$rating)

}

# run the optimization against edx_train_set and edx_validation_set prepared earlieer
# store the results of optimization in *RMSEs* vector
RMSEs <-  sapply(lambdas, RMSE_regularized, edx_train_set, edx_validation_set)

# plot lambda agains RMSE
qplot(lambdas, RMSEs)
```




```
# choose the best lambda
lambda_best <- lambdas[which.min(RMSEs)]
lambda_best
```

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

```
rmse <- min(RMSEs)
rmse
```

```
## [1] 0.8655426
```

```
# run RMSE_regularized agains edx and validation for the final RMSE usinglambda_best
rmse <- RMSE_regularized(lambda_best, edx, validation)

# add the rmse to the results table
# remove in case we run this chunk multiple times
if (any(model_results$Method =="Movie + User Efect Regularized"))
{
  model_results <- model_results[-which(model_results$Method =="Movie + User Effect Regularized"),]
}

# add current rmse result to reporting table
model_results <- bind_rows(model_results,
                           data_frame(
                             Method = "Movie + User Effect Regularized",
                             RMSE = rmse))


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

Method	RMSE
Global Rating Average	1.0612019
Movie Effect	0.9439087
Movie + User Effect	0.8653488
Movie + User Effect Regularized	0.8648201

RMSE Include Genres

```
# for testing we consider only the first genre in the list for now
# extract_first_genre adds a column with the first genre in the list.
extract_first_genre <- function(a_movie_set)
{

  tmp <- as.data.frame(sapply(str_split(a_movie_set$genres, "\\|"), first))
  names(tmp) <- c("genre_1")
  a_movie_set <- bind_cols(a_movie_set, tmp)
  # a_movie_set <- a_movie_set %>% mutate(genre_1 = factor(genre_levels))
  a_movie_set
}


# define RMSE_genre_regularized to compute RMSE
RMSE_genre_regularized <- function(lambda, a_train_set, a_validation_set){
    # movie effect applying regularization
    b_movies <- a_train_set %>%
```

```
      group_by(movieId) %>%
      summarize(b_movie = sum(rating - rate_avg)/(n() + lambda))

    # user effect with regularization, also considering the movie effect regularized
    b_users <- a_train_set %>%
      left_join(b_movies, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_user = sum(rating - b_movie - rate_avg)/(n() + lambda))

    # genre effect regularized, also considering the user and movie effect regularized
    b_genres <- a_train_set %>%
      left_join(b_movies, by = "movieId") %>%
      left_join(b_users, by = "userId") %>%
      group_by(genre_1) %>%
      summarize(b_genre = sum(rating - b_movie - b_user - rate_avg)/(n() + lambda))
      #summarize(b_genre = sum(rating - b_movie - b_user - rate_avg)/n())

    # join movie and user effects tables with the test and compute predicted rate
    predicted_ratings <- a_validation_set %>%
      left_join(b_movies, by = "movieId") %>%
      left_join(b_users, by = "userId") %>%
      left_join(b_genres, by = "genre_1") %>%
      mutate(rating_pred = rate_avg + b_movie + b_user + b_genre)

    # compute RMSE
    RMSE(predicted_ratings$rating_pred, a_validation_set$rating)
}


edx <- extract_first_genre(edx)
validation <- extract_first_genre(validation)

# compute RMSE against edx and validation sets
rmse <- RMSE_genre_regularized(lambda_best, edx, validation)
```

```
## Warning: Column `genre_1` joining factors with different levels, coercing
## to character vector
```

```
# remove in case we run this chunk multiple times
if (any(model_results$Method =="Movie/User/Genre Effect Regularized"))
{
  model_results <- model_results[-which(model_results$Method =="Movie/User/Genre Effect Regularized"),]
}


# add current rmse result to reporting table
model_results <- bind_rows(model_results,
                           data_frame(
                             Method = "Movie/User/Genre Effect Regularized",
                             RMSE = rmse))

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

Method	RMSE
Global Rating Average	1.0612019
Movie Effect	0.9439087
Movie + User Effect	0.8653488
Movie + User Effect Regularized	0.8648201
Movie/User/Genre Effect Regularized	0.8647167

```
# genre improves very little the RMSE, needs more study
```