ML-Capstone-Movielens

Becky

2023-11-25

STEPS: 1. Preprocess Data + Create Test and Training Set 2. SVD Matrix Factorization 3. Test result

Setting up - skipping this step after saving the required df to local to save memory

```
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# 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 core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
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
```

```
library(tidyverse)
library(caret)
library(Matrix)
```

```
##
## Attaching package: 'Matrix'
##
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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

write.csv(final_holdout_test, "final_holdout_test.csv", row.names = FALSE)
#write.csv(edx, "edx.csv", row.names = FALSE)
```

MovieLense Machine Learning Project - SVD

```
#load librarys
library(tibble)
library(readr)
library(dplyr)
library(tidyr)
library(recommenderlab)
```

```
## Loading required package: arules

##
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: proxy

##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

##
## Attaching package: 'recommenderlab'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```r
# create a RMSE function to avoid errors caused by NA
# also slightly less computationally intensive...
RMSE <- function(true, predict) {
  differences <- true - predict
  differences <- differences[!is.na(differences)]
  sqrt(mean(differences^2))
}
```

Load and preprocess data

- use edx set userId and movieId to create data partition
- only taking 10% of the edx data because this is the largest data file size my computer can take to perform the pivot_wider table, otherwise I constantly ran into the "memory limit" error

```r
set.seed(2023)
# too bad I don't have more powerful RAMs, can edx sponsor me some Nvidia RAM
#next time it asks me to do ml project with 8million rows of data?
# this is about the max. number of movielens data my computer has memory to
# process for pivot table and matrix, after many hours of testing + sweat & tears
maxRowNum = 170000
# create a subset from edx df to perform testing and training
edx_small <- sample_n(edx, size = maxRowNum/0.8)

test_ind <- createDataPartition(edx_small$userId, times = 1, p=0.2, list = FALSE)
test_subset <- edx_small[test_ind,]
train_subset <- edx_small[-test_ind,]

#clean test and train set
test_subset <- test_subset |>
  semi_join(train_subset, by = "movieId") |>
  semi_join(train_subset, by = "userId")
train_subset <- train_subset |>
  semi_join(test_subset, by = "movieId") |>
  semi_join(test_subset, by = "userId")

#save and load test and train set to save my computer from running out of memory......
#write.csv(train_subset, "train_subset.csv", row.names = FALSE)
#write.csv(test_subset, "test_subset.csv", row.names = FALSE)
#train_subset <- read.csv("train_subset.csv")
#test_subset <- read.csv("test_subset.csv")

rm(edx, edx_small)
```

Transform the train_subset into matrix

```r
# pivot the df to create matrix: user a row, movie as column
rating_matrix <- train_subset |>
  select(movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating) |>
  column_to_rownames(var = "userId")

# convert to matrix
rating_matrix <- as.matrix(rating_matrix)

# calculate all movie avg
```

```
mu <- mean(rating_matrix, na.rm = TRUE)
n <- colSums(!is.na(rating_matrix))
sums <- colSums(rating_matrix - mu, na.rm = TRUE)

#create a fit movie data frame for lambda testing
fit_movies <- data.frame(movieId = as.integer(colnames(rating_matrix)),
                         mu = mu,
                         b_i_reg = 0 )
```

Regularization

*calculate the movie effect b_i_reg & find the estimated user effect b__u

```
# for movie effect, finding lambda to penalize movie with few rating
lambdas = seq(0,10,.1)
lambdas_rmse <- sapply(lambdas, function(lambda){
  b_i <- sums/(n + lambda)
  fit_movies$b_i <-  b_i
  #left join to the test set
  left_join(test_subset, fit_movies, by='movieId') |>
    #create predicted value by adding mu + b_i
    mutate(pred = mu + b_i) |>
    #calculate RMSE
    summarize(rmse = RMSE(rating, pred)) |>
    pull(rmse)
})
#the lambda value that minimize RMSE is 2.3
lambda <- lambdas[which.min(lambdas_rmse)]
```

remove the user effects and movie effects from the matrix

```
b_i_reg <- colSums(rating_matrix - mu, na.rm=TRUE)/(n + lambda)
fit_movies$b_i_reg <- b_i_reg
fit_users <- data.frame(userId = as.integer(rownames(rating_matrix)),
                        b_u = rowMeans(sweep(rating_matrix-mu,2,b_i_reg), na.rm=TRUE))

rating_matrix <- sweep(rating_matrix-mu, 2, fit_movies$b_i_reg) - fit_users$b_u
#rating_matrix[1:10,1:10]

# replace NA with 0 or -1 for the svd model,
# -1 assume user dislikes the movie or movie is unpopular
rating_matrix[is.na(rating_matrix)] <- 0
#rating_matrix[1:10, 1:10]
```

SVD Model Training

```
svd_model <- svd(rating_matrix)
#save(svd_model, file = "svd_model.RData")
#load("svd_model.RData")
```

After running the svd model, I am constructing an approximated matrix I only select 10 features to avoid overfitting

```
#user
U <- svd_model$u
```

```r
#movie
V <- svd_model$v
Sigma <- diag(svd_model$d)

#assign row names and column names for the user and movie matrix U and V
rownames(U) <- rownames(rating_matrix)  # User IDs as row names
colnames(V) <- colnames(rating_matrix)  # Movie IDs as column names

# select number of features to use, avoid overfitting
num_features <- 10

# reconstruct the rating matrix using a subset of features
# U[, 1:num_features] %*%
# Sigma[1:num_features, 1:num_features] %*%
# t(V[, 1:num_features])
approx_rating_matrix <-
  U[, 1:num_features] %*% Sigma[1:num_features, 1:num_features] %*% t(V[, 1:num_features])

# set the proper row and column names
rownames(approx_rating_matrix) <- rownames(rating_matrix)
colnames(approx_rating_matrix) <- colnames(rating_matrix)
```

Testing on test set

Need to transform the test set so that the test rating matrix is in the same structure as the approx_rating_matrix Also transform the approx_rating_matrix to contain only user and movie in test set

```r
# test set pivot table transformation and convert to matrix
test_rating_matrix <- test_subset  |>
  select(movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating) |>
  column_to_rownames(var = "userId")
test_rating_matrix <- as.matrix(test_rating_matrix)

# replace the NA with 0 in the matrix
test_rating_matrix[is.na(test_rating_matrix)] <- 0

# find the common colnames and rownames between approx. matrix and test matrix
common_users <- intersect(row.names(test_rating_matrix), row.names(approx_rating_matrix))
common_movies <- intersect(colnames(test_rating_matrix), colnames(approx_rating_matrix))

# transform the approx_rating_matrix to contain only user and movie in test set
aligned_approx_ratings <- approx_rating_matrix[common_users, common_movies]
aligned_test_ratings <- test_rating_matrix[common_users, common_movies]

#remove unnecessary large matrix to save some memory space...
rm(test_rating_matrix)
```

```r
# Function to calculate RMSE in chunks
calculate_rmse_in_chunks <- function(actual, predicted, chunk_size) {
  n <- nrow(actual)
  rmse_values <- numeric(ceiling(n / chunk_size))
  for (i in seq(1, n, by = chunk_size)) {
    chunk_end <- min(i + chunk_size - 1, n)
    chunk_actual <- actual[i:chunk_end, , drop = FALSE]
    chunk_predicted <- predicted[i:chunk_end, , drop = FALSE]
    rmse_values[ceiling(i / chunk_size)] <- RMSE(chunk_actual, chunk_predicted)
  }

  mean(rmse_values, na.rm = TRUE)
}

# Calculate RMSE in chunks
chunk_size <- 10000

#add mu back to the approx. rating

#aligned_approx_ratings_withMU <- aligned_approx_ratings + mu
#aligned_test_ratings_withMU <- aligned_test_ratings - mu

rmse <- calculate_rmse_in_chunks(aligned_test_ratings, aligned_approx_ratings, chunk_size)

#aligned_test_ratings[1:10, 1:10]
rmse
```

```
## [1] 0.07195849
```

The rmse for the test set before regularization is 0.07508595 The rmse for the test set after regularization is 0.07195849

Testing on final_hold_out set (last required step)

Need to transform the final hold out set so that the test rating matrix is in the same structure as the approx_rating_matrix Also transform the approx_rating_matrix to contain only user and movie in final hold out set

```
# I am creating a small subset of final hold out since my model can take around 170,000
# rows of movielens data at max.

final_holdout_test_small <- final_holdout_test |>
  semi_join(train_subset, by = "movieId") |>
  semi_join(train_subset, by = "userId") |>
  sample_n(size = maxRowNum)

head(final_holdout_test_small)
```

```
##   userId movieId rating  timestamp                                  title
## 1  57784    1187      3  948512874                     Passion Fish (1992)
## 2  30826    3081      3  965337555                    Sleepy Hollow (1999)
## 3  10623    3927      2  975799914                Fantastic Voyage (1966)
## 4  10668   37830      3 1218405034 Final Fantasy VII: Advent Children (2004)
## 5  42723    2427      5 1012189830                  Thin Red Line, The (1998)
## 6  38234    1135      2  966539648                  Private Benjamin (1980)
##                                  genres
## 1                                 Drama
## 2           Fantasy|Horror|Mystery|Romance
## 3                          Adventure|Sci-Fi
## 4 Action|Adventure|Animation|Fantasy|Sci-Fi
## 5                          Action|Drama|War
## 6                                    Comedy
```

```
#remove unnecessary large matrix to save some memory space
#otherwise, my code can not run...
rm(rating_matrix, svd_model, aligned_test_ratings)
```

```
# test set pivot table transformation and convert to matrix
finaltest_rating_matrix <- final_holdout_test_small  |>
  select(movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating) |>
  column_to_rownames(var = "userId")
finaltest_rating_matrix <- as.matrix(finaltest_rating_matrix)

# replace the NA with 0 in the matrix
finaltest_rating_matrix[is.na(finaltest_rating_matrix)] <- 0

# find the common colnames and rownames between approx. matrix and test matrix
common_users <-
  intersect(row.names(finaltest_rating_matrix), row.names(approx_rating_matrix))
common_movies <-
  intersect(colnames(finaltest_rating_matrix), colnames(approx_rating_matrix))

# transform the approx_rating_matrix to contain only user and movie in test set
aligned_approx_ratings <- approx_rating_matrix[common_users, common_movies]
aligned_finaltest_ratings <- finaltest_rating_matrix[common_users, common_movies]
```

```

rmse_final <- calculate_rmse_in_chunks(aligned_finaltest_ratings,
                                       aligned_approx_ratings,
                                       chunk_size)

rmse_final
```

Calculating the rmse with svd model on final hold out set

```
## [1] 0.1533169
```

The rmse_final after regularization is around 0.15, not bad!