MovieLens Recommendation System Project

Mohammed Chihab

5/5/2020

Executive Summary

As a brief summary, the main goal of this project is to create a movie recommendation system by developping an algorithm that will yield better prediction. The steps perfomed throughout this project are as follows: First, datasets were generated. second, data rows were made sure not to duplicate in each both sets. Third, the algorithm was started in its development; here, we focus on how much our predictions are affected by different variables in our dataset, then, the total variability of the effect sizes was regularized by a penalty term. Cross validation and the calculated root mean squared errors (RMSEs) are used to select the penalty term, a single penalty term was decided upon and chosen to fit the model. Finally, the model was applied to predict the test set and the root mean squared error (RMSE) was used as an evaluation to those results.

Methods/Analysis

1.01 seting up the main datasets

The project will make use of two datasets that will be generated using a pre-developped code by the course staff. The datasets are named **edx** and **validation** .

```r
#################################
# Create edx set, validation set
#################################

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

# 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 <- 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(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, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

1.02 Exploring the edx dataset

edx dataset as of 5/9/2020

```r
# first three rows in the edx dataset
head(edx, 3)
```

```
##   userId movieId rating timestamp            title                        genres
## 1      1     122      5 838985046 Boomerang (1992)                Comedy|Romance
## 2      1     185      5 838983525  Net, The (1995)         Action|Crime|Thriller
## 4      1     292      5 838983421  Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
```

```r
# last three rows in the edx dataset
tail(edx, 3)
```

```
##     userId movieId rating  timestamp                         title
## 61   59269   63141    2.0 1226443318 Rockin' in the Rockies (1945)
## 77   60713    4820    2.0 1119156754  Won't Anybody Listen? (2000)
## 834  64621   39429    2.5 1201248182                Confess (2005)
##                       genres
## 61   Comedy|Musical|Western
## 77              Documentary
## 834          Drama|Thriller
```

```r
# edx dimentions:
dim(edx)
```

```
## [1] 9000055       6
```

```
# The different movies in the edx dataset
n_distinct(edx$movieId)
```

```
## [1] 10677
```

```
# The different users in the edx dataset
n_distinct(edx$userId)
```

```
## [1] 69878
```

```
# The different genres combinations in the edx dataset
n_distinct(edx$genres)
```

```
## [1] 797
```

```
# The top 2 ratings in the edx dataset
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(2) %>%
    arrange(desc(count))
```

```
## Selecting by count
```

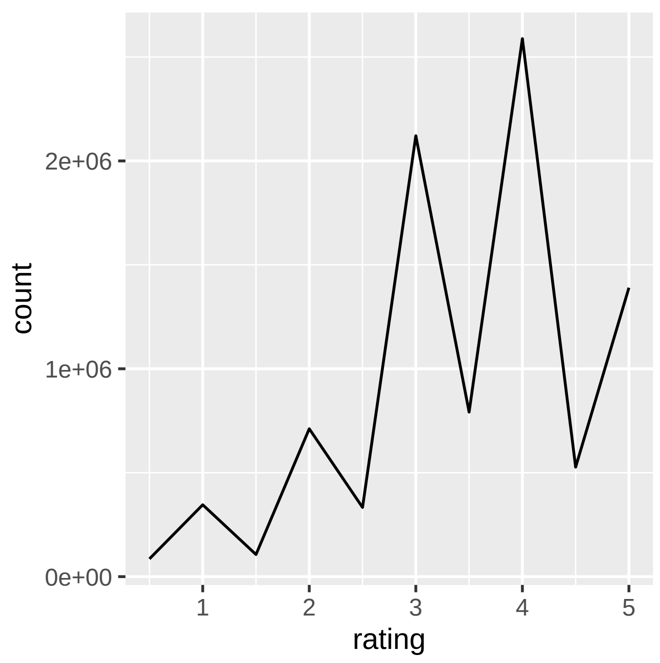
```
## # A tibble: 2 x 2
##   rating   count
##    <dbl>   <int>
## 1      4 2588430
## 2      3 2121240
```

```
# The lowest 2 ratings in the edx dataset
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(-2) %>%
    arrange(count)
```

```
## Selecting by count
```

```
## # A tibble: 2 x 2
##   rating  count
##    <dbl>  <int>
## 1    0.5  85374
## 2    1.5 106426
```

```
# The number of counts per ratings plotted
edx %>%
    group_by(rating) %>%
    summarize(count = n()) %>%
    ggplot(aes(x = rating, y = count)) +
    geom_line()
```



1.03 Deciding on the main training set and main validation test set.

The datasets, **edx** and **validation** , will represent the **training set** and the **test set** respectively.

```
# Renaming edx and validation data sets as train and test sets respictively.

train_set <- edx
test_set <- validation
```

2.00 Splitting the edx dataset into a training set and a test set, For the pupose of training

Once both datasets are prepared, the next step is to reserve the **validation** dataset for final testing, and split the **edx** dataset into two sets, a **training training set** and a **training test set** respectively. This will give the model saught to be developped more credibility.

As the process will generate random datasets, it is best to set the seed.

```
# Set the seed to 1. If  you are using R 3.5 or earlier, use `set.seed(1)` instead

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

Now create the **training training set** and a **training test set** by splitting the **train_set** which is the **edx** dataset.

```
# Split the train_set into train_train_set and train_test_set

train_test_index <- createDataPartition(y = train_set$rating , times = 1, p = 0.1, list = FALSE)
train_train_set <- train_set[-train_test_index,]
train_test_set <- train_set[train_test_index,]
```

Use semi-join to make sure data rows do not duplicate in both sets.

```r
# Avoid duplication

train_test_set <- train_test_set %>%
  semi_join(train_train_set, by = "movieId") %>%
  semi_join(train_train_set, by = "userId")
```

3.01 Fitting and testing a model through the train__train__set and the train__test__set

In this section, to fit and build our model, the **test__set** which is the **validation set** will **not** be used. It will be left for final assessment on which our model will be used upon, it will mainly serve as a stranger data that simulates reality.

Here, the variables that will be taken into account are: the **mean** of the ratings in the training training set, the **average rating for each movie** and the **effect of each user** on the ratings as well as the **effect of the genres combinations** on the ratings .

For better results, While fitting some few different models, it has appeared that **regularizing** the total variability of the effect sizes of our variables by adding a penalty term based on a tuning parameter (call it: **lambda**), **i.e.** to calculate the (average rating for movie), the (user-specific effect) and the (effect of genres vombinations) variables by restraining them. makes the model better and yields a somewhat satisfying results.

Cross validation is used here to decide on the tuning perimeter lambda.

This sequence of numbers from 1 to 8 by 2 step has been used.

```r
# set up the tuning perimeter lambdas

lambdas <- seq(1, 8, 2)
lambdas
```

```
## [1] 1 3 5 7
```

Once the perimeters are set up, and since the root mean squared error (RMSE) can be interpreted similarly to the standrad deviation,it will be used as our loss function to choose the best lambda for our model fitting and to test our predictions.

```r
# set up the RMSE function

RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

To choose lambda through rmses, a function will be created that will iterate over each perimeter and then the one that generates the minimum rmse will be chosen as our tuning parameter.

The variables that will be used are the ones mentioned before and they are as follows:

- **lambda**: our tuning perimeter for our penalty term.
- **mu**: The **mean** of the ratings.
- **e__i**: The **effect of the average rating for each movie**.
- **e__u**: The **effect of each user** on the ratings, while taking into consideration **e__i**.

- **e__g**: The **effect of genres combinations** on the ratings, , while taking into consideration **e__i** and **e__u**.
- **pred** from The **predicted__ratings** variable, which will represent the sum of **mu**, **e__i**, **e__u** and **e__g**.

NOTE: This will also serve as our model if the best tuning perimeter is the only perimeter taken into account to calculate the penalty term.

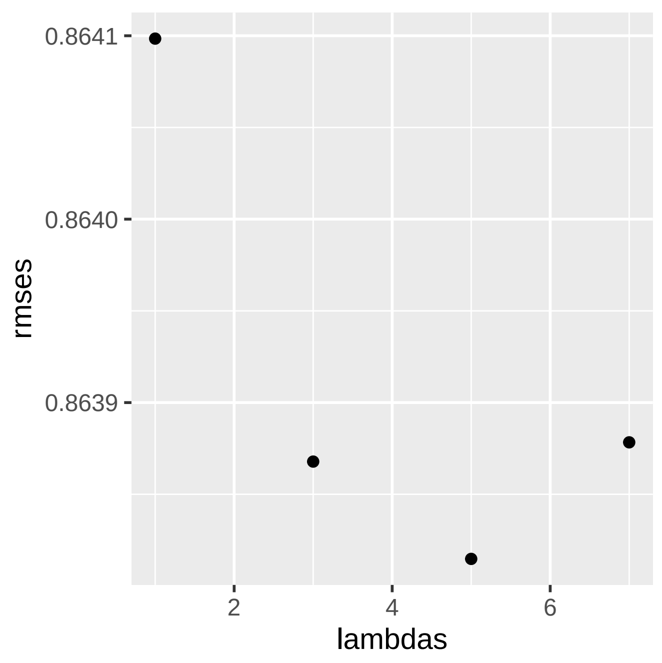
```
# set up the RMSES function

rmses <- sapply(lambdas, function(lambda){
  mu <- mean(train_train_set$rating)
  e_i <- train_train_set %>%
    group_by(movieId) %>%
    summarize(e_i = sum(rating - mu)/(n()+lambda))
  e_u <- train_train_set %>%
    left_join(e_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(e_u = sum(rating - e_i - mu)/(n()+lambda))
  e_g <- train_train_set %>%
    left_join(e_i, by="movieId") %>%
    left_join(e_u, by="userId") %>%
    group_by(genres) %>%
    summarize(e_g = sum(rating - e_i - e_u - mu)/(n()+lambda))
  predicted_ratings <-
    train_test_set %>%
    left_join(e_i, by = "movieId") %>%
    left_join(e_u, by = "userId") %>%
    left_join(e_g, by = "genres") %>%
    mutate(pred = mu + e_i + e_u + e_g) %>%
    .$pred
  return(RMSE(predicted_ratings, train_test_set$rating))
})
```

plot the rmses against the lambdas to see how each lambda performs.

```
# plot the rmses against the lambdas

qplot(lambdas, rmses)
```



pick the best performing lambda to be used as our tuning perimeter.

```
# find out the minimum RMSE

min(rmses)
```

```
## [1] 0.8638148
```

```
# find out the corresponding lambda for the minimum RMSE

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

```
## [1] 5
```

The minimum rmse (**0.8638148**) calculated on the edx test set (**train_test_set**) is lower than the rmse requested in the guidelines on the project best rmses (**0.86490**) by **0.0010852**, which shows that our model is performing well as requested on the **train_test_set**. So, let's see how it performs **elsewhere**, exactly on the **validation set**.

3.02 Model implementation on a simulated foreign data (validation dataset)

In this section, we think of the **validation dataset**, which we renamed **test_set**, as a foreign dataset by simulation. This dataset has **NOT BEEN USED yet**. For training our model, we instead used the **edx** dataset, which we renamed **training_set** which is split into training set, named **training_training_set**, and test set,named **training_test_set**, to avoid the use of the **validation set**.

Let's, we will be using our **model** that we have created and tested before on this **novel data**: the **validation dataset** to **predict** ratings of movies.

```r
# Model implementation on a simulated novel foreign data (the validation dataset)

lambda <- 5

mu <- mean(train_set$rating)

e_i <- train_set %>%
  group_by(movieId) %>%
  summarize(e_i = sum(rating - mu)/(n()+lambda))

e_u <- train_set %>%
  left_join(e_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(e_u = sum(rating - e_i - mu)/(n()+lambda))

e_g <- train_set %>%
  left_join(e_i, by="movieId") %>%
  left_join(e_u, by="userId") %>%
  group_by(genres) %>%
  summarize(e_g = sum(rating - e_i - e_u - mu)/(n()+lambda))

predicted_ratings <-
  test_set %>%
  left_join(e_i, by = "movieId") %>%
  left_join(e_u, by = "userId") %>%
  left_join(e_g, by = "genres") %>%
  mutate(pred = mu + e_i + e_u + e_g) %>%
  .$pred
```

Here we create a data frame of the first 10 true ratings and predicted ratings to get a glimps of how our model has performed. Next, in the results we shall apply a better method, the rmse function, to get a better sense of the performance of our model.

```r
# display the first 10 entries of the true ratings and the predicted ratings

df <- data.frame(true_ratings = head(test_set$rating, 10),
                 prediced_ratings = head(predicted_ratings, 10))
df
```

```
##    true_ratings prediced_ratings
## 1           5.0         4.259621
## 2           5.0         4.972631
## 3           5.0         4.363739
## 4           3.0         3.354043
## 5           2.0         4.256574
## 6           3.0         2.747183
## 7           3.5         3.930071
## 8           4.5         4.167796
## 9           5.0         4.242787
## 10          3.0         3.274467
```

Results Using RMSE

In this section, we shall use the RMSE, a second time but here on the validation test set to compare it with the minimum one chosen based on the edx test set in order to present our result and model performence on a novel foreign simulated dataset. We know that the best rmse would be (RMSE < 0.86490) as per the guidelines, and that what we have achieved using the edx datasets (RMSE was 0.8638148).

```r
# rmse on validation dataset (test_set)

rmse_on_validation_set <- RMSE(predicted_ratings, test_set$rating)
rmse_on_validation_set
```

```
## [1] 0.8644501
```

The rmse(**0.8644501**) calculated on the validation set, though it is larger than the rmse (**0.8638148**) calculated on the edx test set that is used in the training and building of our model by **0.0006353**, it is lower than the rmse requested in the guidelines on the project best rmses **0.86490** by **0.0004499**, which shows that our model is performing well as requested .

conclusion

To conclude, our model does well as requested. However improvment is possible given the resources and the tools are avaible to work on large datasets. Possibly, in large data sets there would be more room to get as closer with the predictions to the true datasets as possible.