MovieLens Project

Saman Musician

4/24/2021

Introduction

For this project, we have to create a movie recommendation system using the MovieLens dataset. The original MovieLens data set is a very big data set provided by GroupLens Research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, in order to gather research data on personalized recommendations. We will train a machine learning algorithm using a smaller version of this data set, 10M version of the MovieLens dataset, for the ease of computations.

The data set includes about 10 million movie ratings by 69878 different users for 10677 different movies, along with the timestamps of the ratings and the genres of the movies. The goal of this project is to create a recommendation system using this data set and all the tools we have studied throughout the courses in the Data Science series to predict how a movie would be rated by a user based on the information we have with minimum error.

The key steps performed includes, data gathering, data cleaning and exploration, calculating different effects on the output and using regularization for predictors in the data set to predict the ratings. The final algorithm considers the movie effect, user effect, the genre effect and the year effect in the prediction of ratings and a regularization lambda parameter to lower the RMSE calculated.

2. Method

2.1. Data wrangling

This section explains the process and techniques used in this project. First, we should acquire the data and preprocess it to be ready for the analysis. Using the provided script we first download the data and then split it into a training set, "edx", and a hold-out final "validation" set. Note that since the validation set is hold untouched we have to prevent the inclusion of any new movie or user in this set. So, we remove the data in 'validation' set for which we don't have any entries in the 'edx' set and add them back to the 'edx' set.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
dim(edx)
```

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

```
dim(validation)
```

```
## [1] 999999      6
```

Moreover, in the procedure of training the final algorithm, time of each review is used in the format of year. So, a new column should be added to both 'edx' and 'validation' sets named "year". Here we mutate this column in both data sets for our further purpose:

```
benchmark = as.Date('1970-1-1')
edx <- edx %>%
  mutate(year = format(as.Date(timestamp/86400,benchmark),format = "%Y"))
validation <- validation %>%
  mutate(year = format(as.Date(timestamp/86400,benchmark),format = "%Y"))
rm(benchmark)
dim(edx)
```

```
## [1] 9000055       7
```

```
dim(validation)
```

```
## [1] 999999      7
```

Next, because we need to use only the training set for optimization and developing the final algorithm, we split the 'edx' data set again into a training set and a test set. We use 0.1 proportion for the test set. Note that again, we make sure for the data for all the movies and users in the test set there is data in the training set. We add back new data in test set to train set like before.

```
set.seed(1, sample.kind = "Rounding")
index <- createDataPartition(edx$rating, 1, p = 0.1, list = F)
train_set <- edx[-index,]
temp <- edx[index,]

test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(index, temp, removed)
```

2.2 Analysis

The model for this calculation is as follows:

$$Y_{u,i} = \mu + b_i + b_u + b_g + b_y + \epsilon_{u,i}$$

$Y_{u,i}$ predicted rating by user u for movie i
μ average of all ratings
b_i movie effect bias term
b_u user effect bias term
b_g genre effect bias term
b_y year effect bias term
$\epsilon_{u,i}$ error for user u movie i

As the first try, we calculate each bias term the same way as what we have seen in Machine Learning course, by grouping the data and averaging the ratings. We calculate each of the 4 bias terms step by step and in the order of movie, user, genre and year. Then we make predictions using the model above and calculate RMSE on the test set.

```r
# Calculating average rating:
mu <- mean(train_set$rating)

# Movie effect Calculation:
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Calculating User effects:
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Calculating Genres effects:
genre_effect <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

genre_effect[1,2] <- 0   #deleting the effect of "(no genres listed)"

# Calculating year effect:
year_effect <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_effect, by='genres') %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u - b_g))

#--------------------
#making predictions:
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_effect, by='genres') %>%
  left_join(year_effect, by= 'year') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_y) %>% .$pred

try1_rmse <- RMSE(predicted_ratings, test_set$rating, na.rm = T)
results <- data.frame(Method = 'simple average', RMSE = try1_rmse)

results
```

```
##           Method      RMSE
## 1 simple average 0.8643141
```

Although the calculated RMSE seems good, We use regularization to improve the effects of the bias terms by considering lambda. According to the concept of regularization we penalize the bias terms to be smaller, when the numbers of ratings are small. Also the bias term would not be affected much when there is a large numbers of ratings:

$$b_i(\lambda) = \frac{1}{\lambda+n_i}\sum (Y_{u,i} - \mu)$$

According to this formula, instead of averaging all the ratings we consider a lambda parameter for calculating the bias terms. In order to acquire the lambda we create a function. This function takes a value of lambda and calculates bias terms and the RMSE for the predicted values and test set values. The code for the function calculating the bias terms using regularization is as follows:

```r
regularize <- function(l){
  mu <- mean(train_set$rating)

  # Movie effect Calculation
  movie_avgs <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  # User effects Calculation
  user_avgs <- train_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  # Genres effects Calculation
  genre_effect <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

  genre_effect[1,2] <- 0   #deleting the effect of "(no genres listed)"

  # year effect Calculation
  year_effect <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_effect, by='genres') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u - b_g)/(n()+l))

  # Prediction and RMSE
  predicted_ratings <- test_set %>%
    left_join(movie_avgs, by = 'movieId') %>%
    left_join(user_avgs, by = 'userId') %>%
    left_join(genre_effect, by = 'genres') %>%
    left_join(year_effect, by = 'year') %>%
    mutate(pred = mu + b_i + b_u + b_g + b_y) %>% .$pred

  return(RMSE(predicted_ratings, test_set$rating, na.rm = T))
}
```

3. Results

For optimizing the best value for lambda we first give different integer values of lambda to the function, then we try to make it more precise by inputting decimal numbers. Finally the value of lambda resulting in the least RMSE value is chosen. I could simply create a vector for this purpose but because of the memory allocation limit on my computer, I run each line of code individually.

```
l <- c()              #due to memory allocation limits on my PC
l[1] <- regularize(1)
l[2] <- regularize(2)
l[3] <- regularize(3)
l[4] <- regularize(4)
l[5] <- regularize(5)
l[6] <- regularize(6)
l[7] <- regularize(7)
l[8] <- regularize(8)
l[9] <- regularize(9)
which.min(l)
```

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

```
min(l)
```

```
## [1] 0.8637815
```

```
# checking for more improvements in lambda:

options(digits = 10)
regularize(4.8)
```

```
## [1] 0.8637818439
```

```
regularize(5.2)
```

```
## [1] 0.8637823144
```

```
regularize(4.9)
```

```
## [1] 0.8637815314
```

```
# << lambda = 5.0 is optimized >>

results <- bind_rows(results,data_frame(Method="Regularized",RMSE = l[5] ))

results
```

```
##           Method         RMSE
## 1 simple average 0.8643141244
## 2    Regularized 0.8637815093
```

The value of 5.0 is chosen for lambda. Now we can train our final algorithm on the 'edx' data set with this lambda value and calculate the final RMSE value on the 'validation' set.

```
## [1] "The final RMSE:"
```

```
## [1] 0.8644101472
```

The final result of RMSE is 0.8644101, which is below the value that was the goal by this assignment.
It is noteworthy to mention the fact that this final RMSE value is a bit higher than the value for the optimized model on training and test sets, which may be an indication of some overtraining.

4. Conclusion

To wrap up this study, we improved a movie recommendation system by taking the effects of movie, user, genre group and year of rating into consideration and regularizing these effects.
It is highly presumable that the interaction between the users and using the correlation matrix can be effective in predicting the ratings; however, using the subset provided in this project did not result in the desired model. It may be possibly because of the limited number of entries in comparison with the original MovieLens data set.
The effect I assume can make the performance of the model much better is the genre effect. In the presented model the entries are grouped by the whole genre groups. As a result, the effect of an individual genre is not specifically accounted for. This claim was proven to me when the RMSE reduction after adding the presented genre effect to the model was not tangible enough. I tried so hard to divide different genre effects but after adding the year effect I succeeded in reaching to the desired RMSE value. However, I believe the efficacy of considering each single genre would be satisfactory.