Movie Recommendation

Brian Searles

11/18/2019

```
## Loading required package: tidyverse

## -- Attaching packages -----------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

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

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

## Loading required package: 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

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

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

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")

## Warning in rm(dl, ratings, movies, test_index1, test_temp, edx, removed1):
## object 'dl' not found

## Warning in rm(dl, ratings, movies, test_index1, test_temp, edx, removed1):
## object 'ratings' not found

## Warning in rm(dl, ratings, movies, test_index1, test_temp, edx, removed1):
## object 'movies' not found
```

Introduction

The purpose of this project was to develope a machine learning program that could predict the rating of a movie with as small of a root mean squared error, hencefore called the RMSE, as possible, with a goal of less than 0.8649. This program uses the movielens data set developed by the University of Minnesota. This data set consists of 10 million ratings of 10 thousand different movies by 72 thousand unique users. Every entry in the data set is comprised of a movie id, a user id, a rating given by a signle user out of 5 stars, a timestamp describing when the movie was released, the title of the movie, and a list of genres that the movie falls under. This data is split into three catagories, train/test/validate. The train set is used to train the machine learning algorithm, while the test set is used to test the improved accuracy of the added bias. Finally, the validate set is used to test the final accuracy of the model, and is what will be used to judge the final RMSE. Using this information, this program attempts to predict the average rating that a movie recieves by a linear model method with regularization. The modeling equation takes the form $Y = \mu + \sum b_x$, henceforth called the prediction equation, where μ is the average rating of all movies and b_x refers to the multiple "biases" that exist within the data set. This program uses 3 biases, movie bias, user bias, and genre bias, to calculate the predicted rating. Then, once the biases have been determined, a regularization technique in which a penalty term λ is added. The optimum value of the penalty term is calculated via applying a series of different values of λ and picking the value which minimized the RMSE.

Once the average, biases, and penalty terms have been found, the program tests itself by measuring the predicted values versus the actual values using the RMSE method.

Analysis

Firstly, let us analyze the components of the data set.

```r
head(train)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
## 8      1     356      5 838983653           Forrest Gump (1994)
##                          genres
## 1                Comedy|Romance
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
## 8      Comedy|Drama|Romance|War
```

As stated in the introduction, the data set is comprised of 6 catagories (columns), userId, movieId, rating, timestamp, title, and genre. Each individual who participated in providing data in the data set is given a unique userId. Every movie rated is also given a unique Id, called movieId. Each user can only rate a movie once, so we can see that there are no repeating entries with the same userId and movieId. The rating is given as an integer on the scale from 1 to 5. The other 3 catagories are information about the movie, such as the movies title, genre, and timestamp. The timestamp category provides the time difference between when the movie was released and midnight of the date of 1/1/1970 in seconds.

This type of data implies a linear relationship for predicting the rating of a movie. This leads us to believe that the rating of a movie is a function of multiple biases, which each influence the rating in their own way.

The first bias we will analyze is the movie bias, which we will call b_i. This bias originates due to the fact that some movies are better than others and therefore obtain a higher rating. To account for this bias in our model, we need to find its value. We can define this bias as the difference between the movies rating and the average rating of all movies. In code, this takes the form:

```r
movie_avg <- train %>% group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
predicted_ratings <- mu + test%>%
  left_join(movie_avg, by = 'movieId') %>% pull(b_i)
```

Plugging this bias into our prediction equation, we get $Y = \mu + b_i$. See the Results section for influence of this bias.

The second bias we will consider is the user bias, which we will call b_u. This bias considers how individual users tend to rate certain types of movies higher or lower. We find this value in a similar method to the movie bias, where the bias is equal to the mean of the difference of an individual rating, the average rating, and the movie bias. The code takes the form:

```r
user_avg <- train %>% left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>% summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- test %>% left_join(movie_avg, by = 'movieId') %>%
  left_join(user_avg, by='userId') %>% mutate(pred = mu +b_i + b_u) %>% pull(pred)
```

Plugging this bias into our prediction equation, we get $Y = \mu + b_i + b_u$. See the Results section for influence of this bias.

The third bias that effects the model is the genre bias, which we will call b_g. This bias focuses on the relationship between the genre of a movie and its rating. The assumption is that movies of a certain genre, or a combination of genres, tend to rate higher than movies of other genres. By taking this into account we can predict either a higher rating or lower rating based off its genre classification. The code that does this is this:

```r
genre_avg <- train %>% left_join(movie_avg, by = 'movieId') %>%
  left_join(user_avg, by = 'userId') %>% group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
predicted_ratings4 <- test %>% left_join(movie_avg, by = 'movieId') %>%
  left_join(user_avg, by='userId')%>% left_join(genre_avg, by ='genres') %>%
  mutate(pred = mu +b_i+b_u+b_g)%>% pull(pred)
```

This gives us a prediction equation of $Y = \mu + b_i + b_u + b_g$. Again, see the Results section for influence of this bias.

We can continue to add bias terms to our prediction equation in a similar matter if we suspect they may improve predicting accuracy. We will stop at 3 bias terms for now but the implimenation of additional bias terms is fairly straight forward.

Introducing a penalty term into our prediction equation will also help to minimize the RMSE. To do this, we must first find the optimal value of this penalty term, which we will call λ. This can be done using this code:

```r
lambdas <- seq(0,10,0.2)
rmses <- sapply(lambdas, function(l){
  b_i1 <- train %>% group_by(movieId) %>% summarize(b_i1 = sum(rating-mu)/(n()+l))
  b_u1 <- train %>% left_join(b_i1, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u1 = sum(rating -b_i1 - mu)/(n()+l))
  b_g1 <- train %>% left_join(b_i1, by = "movieId") %>% left_join(b_u1, by = "userId") %>%
    group_by(genres) %>% summarize(b_g1 = sum(rating-b_i1-b_u1-mu)/(n()+l))
  predicted_ratings5 <- test %>% left_join(b_i1, by = "movieId") %>%
    left_join(b_u1, by = "userId") %>% left_join(b_g1, by = "genres") %>%
    mutate(pred = mu+b_i1+b_u1+b_g1) %>% pull(pred)
  return(RMSE(predicted_ratings5, test$rating))
})
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.8
```

Our optimal value of λ is 4.8. To input this into our code, λ must modify each of our biases:

```r
b_i1 <- train %>% group_by(movieId) %>%
  summarize(b_i1 = sum(rating-mu)/(n()+lambda))
b_u1 <- train %>% left_join(b_i1, by = "movieId") %>%
  group_by(userId) %>% summarize(b_u1 = sum(rating -b_i1 - mu)/(n()+lambda))
b_g1 <- train %>% left_join(b_i1, by = 'movieId') %>%
  left_join(b_u1, by = 'userId') %>% group_by(genres) %>%
  summarize(b_g1 = sum(rating - mu - b_i1 - b_u1)/(n()+lambda))
```

It should be noted that there are additional techniques that could be used to analyze this data in in addition to constructing a linear model, such as using matrix factorization. However, Seeing as the linear model method is sufficient in achieving an RMSE that is below our threshold, we will ignore it for this report.

Results

The RMSE will be used to judge the accuracy of the algorithm. This equation takes the form:

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

Where true ratings are the actual ratings taken from the test data set, and predicted ratings are the rating predicted by the machine learning algorithm.

To analyze the results, we will observe how the RMSE changes with the addition of each term in the prediction equation. Firstly, let us look at how only predicting the ratings to be equal to the mean rating works out. In this example all predicted ratings will be equal to μ and will be plugged into the RMSE equation.

```
test_mean_only <- RMSE(test$rating, mu)
test_mean_only
```

```
## [1] 1.060054
```

We see that the RMSE of only the mean results in an accuracy of 1.060054.

Next, we will add the movie bias term. This gives us a prediction equation of $Y = \mu + b_i$

```
test_movie_bias <- RMSE(predicted_ratings2, test$rating)
test_movie_bias
```

```
## [1] 0.9429615
```

We observe that the RMSE decreases to a value of 0.9429615. This is a difference of 0.1170925 compared to the μ only accuracy. This leads us to believe that by adding addition biases, we will continue to see a decrease in RMSE.

Now we will add the user bias term. This gives us a prediction equation of $Y = \mu + b_i + b_u$

```
test_user_bias <- RMSE(predicted_ratings3, test$rating)
test_user_bias
```

```
## [1] 0.8646843
```

The RMSE decreases further to a value of 0.8646843. This gives us a difference of 0.0782772 in RMSE between this user bias test and the previous movie bias test.

We now add the genre bias term. This gives us a prediction equation of $Y = \mu + b_i + b_u + b_g$

```
test_genre_bias <- RMSE(predicted_ratings4, test$rating)
test_genre_bias
```

```
## [1] 0.8643241
```

The RMSE is now 0.8643241. We get a difference of 0.0003602 when comparing the RMSE of this test to the previous test. We see that the difference in RMSE is decreasing as we add more biases. This leads us to believe that as more biases are added, the influence each bias has on the overall RMSE decreases.

While this is a good value, by adding a penalty value we can decrease the RMSE even more.

```
test_penalty <- RMSE(predicted_ratings5, test$rating)
test_penalty
```

```
## [1] 0.8641364
```

This finally gives us our smallest RMSE of 0.8641364.

Testing against the validate data set

But these tests have only be focusing on the train and test data sets. For our final test, we will put our predicted values against the validate data set that we created at the beginning of the program.

```r
b_i_final <- train %>% group_by(movieId) %>%
  summarize(b_i_final = sum(rating-mu)/(n()+lambda))
b_u_final <- train %>% left_join(b_i_final, by = "movieId") %>%
  group_by(userId) %>% summarize(b_u_final = sum(rating -b_i_final - mu)/(n()+lambda))
b_g_final <- train %>% left_join(b_i_final, by = 'movieId') %>%
  left_join(b_u_final, by = 'userId') %>% group_by(genres) %>%
  summarize(b_g_final = sum(rating - mu - b_i_final - b_u_final)/(n()+lambda))
predicted_ratings_final <- validation %>% left_join(b_i_final, by = "movieId") %>%
  left_join(b_u_final, by = "userId") %>% left_join(b_g_final, by ="genres") %>%
  mutate(pred = mu+b_i_final+b_u_final+b_g_final) %>% pull(pred)

test_final <- RMSE(predicted_ratings_final, validation$rating)
test_final
```

```
## [1] 0.8648541
```

This gives us an RMSE of 0.86485 which is less than our goal of 0.8649. This value is greater than the value we got when using the test/train data set. This discrepency could be due to overtraining on the train set.

Conclusion

In conclusion, this program was developed with the intention of minimizing the RMSE by using a machine learning model. The model was based upon a linear equation and altered with regularization techniques. The final RMSE was 0.86485, which was below our goal. The biggest issue currently with this program is that the program does require a long time to run, and adding any more techniques or biases to the program will only slow it down further. Additional techniques, such as matrix factorization, could be added to improve the accuracy of the program but this will only add to the total run time of the program.