An Algorithm for a Movie Recommendations System

Charlie Oliver Ontoria Gomez

March 20, 2021

Overview

This project is all about finding an effective algorithm for recommending movies to users. I used the 10M version of the movielens dataset that is publicly available at movielens.org. The 10 million entries of the dataset were divided into a training set (`edx`) and a test set (`validation`). My more specific goal is predict the rating for a particular movie and from a particular user and I want to create an effective algorithm by minimizing a chosen loss function: the residual mean squared error(RMSE). Overall, I chose a simple algorithm that can run on my computer. The scope of the project was limited by my hardware capabilities. In the code provided in a separated document, you will notice that all I did was mutate columns in the train set since I cannot run even a simple linear regression with the big dataset that I have but it was possible for my computer to do analysis since the data is sparse. In the end it was enough to create an effective algorithm.

Methods and Analysis

Before I tested any algorithms, I further divided the `edx` dataset into a `test_set` (20%) and a `train_set` (80%) since I did not want to use the actual test set (`validation`) and break the golden rule of machine learning.
I also created an RMSE function with this code:

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

Finding the `mu`

The first step is to find a baseline rating `mu`. This can be found by calculating the mean of the ratings across all movies with this code:

```
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.5125
```

Putting this into the RMSE function gives us 1.0599 which is bad because the predictions are missing by 1 point on average. There is still a big room for improvement.

Finding the `movie_effect`

Next, I factored in the `movie_effect`. Some movies will naturally be rated higher due to various reasons such as quality, and some will be rated lower. The best way is to run a linear regression but as I stated earlier, I do not have the hardware capabilities. I calculated for this by finding the mean of the residuals after subtracting `mu` from the actual rating. It is shown in this code:

```
movie_ave <- train_set %>% group_by(movieId) %>%
  summarize(movie_effect = mean(rating - mu))
```

Putting the values obtained in the RMSE function give us 0.94374.

Finding the `user_effect`

Then, I factored in the `user_effect`.Some people will rate movies higher than most people and some will rate movies lower. I can quantify this bias by finding the mean of the residuals after subtracting `mu` and the `movie_effect` from the actual rating. It is shown in this code:

```
user_ave <- train_set %>%
  left_join(movie_ave, by="movieId") %>% group_by(userId) %>%
  summarize(user_effect = mean(rating - mu - movie_effect))
```

Putting the results into the RMSE function gives us 0.86593. This is already great but can still be improved

Regularizing the `movie_effect` and the `user_effect`

Checking the top 10 best and worst movies produced by the current algorithm

```
movie_titles <- edx  %>%
  select(movieId, title) %>%
  distinct()

#Best Movies
train_set %>% count(movieId) %>%
  left_join(movie_ave) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(movie_effect)) %>%
  select(title, movie_effect, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
```

```
##                                                                                 title
## 1                                                  Hellhounds on My Trail (1999)
## 2  Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
## 3                                      Satan's Tango (SÃfÂ¡tÃfÂ¡ntangÃfÂ³) (1994)
## 4                                           Shadows of Forgotten Ancestors (1964)
## 5                                                      Money (Argent, L') (1983)
## 6                                            Fighting Elegy (Kenka erejii) (1966)
## 7                                                 Sun Alley (Sonnenallee) (1999)
## 8                                                 Aerial, The (La Antena) (2007)
## 9                                       Blue Light, The (Das Blaue Licht) (1932)
## 10                                                                   More (1998)
##    movie_effect n
## 1        1.4875 1
## 2        1.4875 3
## 3        1.4875 2
## 4        1.4875 1
## 5        1.4875 1
## 6        1.4875 1
## 7        1.4875 1
## 8        1.4875 1
## 9        1.4875 1
```

```
## 10       1.4042 6
```

```
#Worst Movies
train_set %>% count(movieId) %>%
  left_join(movie_ave) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange((movie_effect)) %>%
  select(title, movie_effect, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
```

```
##                                       title movie_effect   n
## 1                           Besotted (2001)      -3.0125   1
## 2          Confessions of a Superhero (2007)      -3.0125   1
## 3  War of the Worlds 2: The Next Wave (2008)      -3.0125   2
## 4        SuperBabies: Baby Geniuses 2 (2004)      -2.7500  40
## 5               From Justin to Kelly (2003)      -2.6672 168
## 6                 Legion of the Dead (2000)      -2.6375   4
## 7                     Disaster Movie (2008)      -2.6375  28
## 8                  Hip Hop Witch, Da (2000)      -2.6034  11
## 9                         Criminals (1996)      -2.5125   1
## 10               Mountain Eagle, The (1926)      -2.5125   1
```

We can see that most of the best and worst movies are obscure with only a few ratings. In order to improve the model, I need to penalize movies that have lower number of ratings. I can do this through regularization. Basically I want to divide the movie effect and user effect with the number of ratings plus an added value lambda instead of simply taking the mean. I found out that the optimal lambda is 4.75. Putting this into code:

```
movie_ave_reg <- train_set %>% group_by(movieId) %>%
  summarize(movie_effect = sum(rating - mu)/(n()+4.75))
user_ave_reg <- train_set %>%
  left_join(movie_ave_reg, by="movieId") %>% group_by(userId) %>%
  summarize(user_effect = sum(rating - mu - movie_effect)/(n()+4.75))
predicted_ratings_3 <- test_set %>%
  left_join(movie_ave_reg, by = "movieId") %>%
  left_join(user_ave_reg, by = "userId") %>%
  mutate(predictions = mu + movie_effect + user_effect) %>%
  .$predictions
RMSE(test_set$rating, predicted_ratings_3)
```

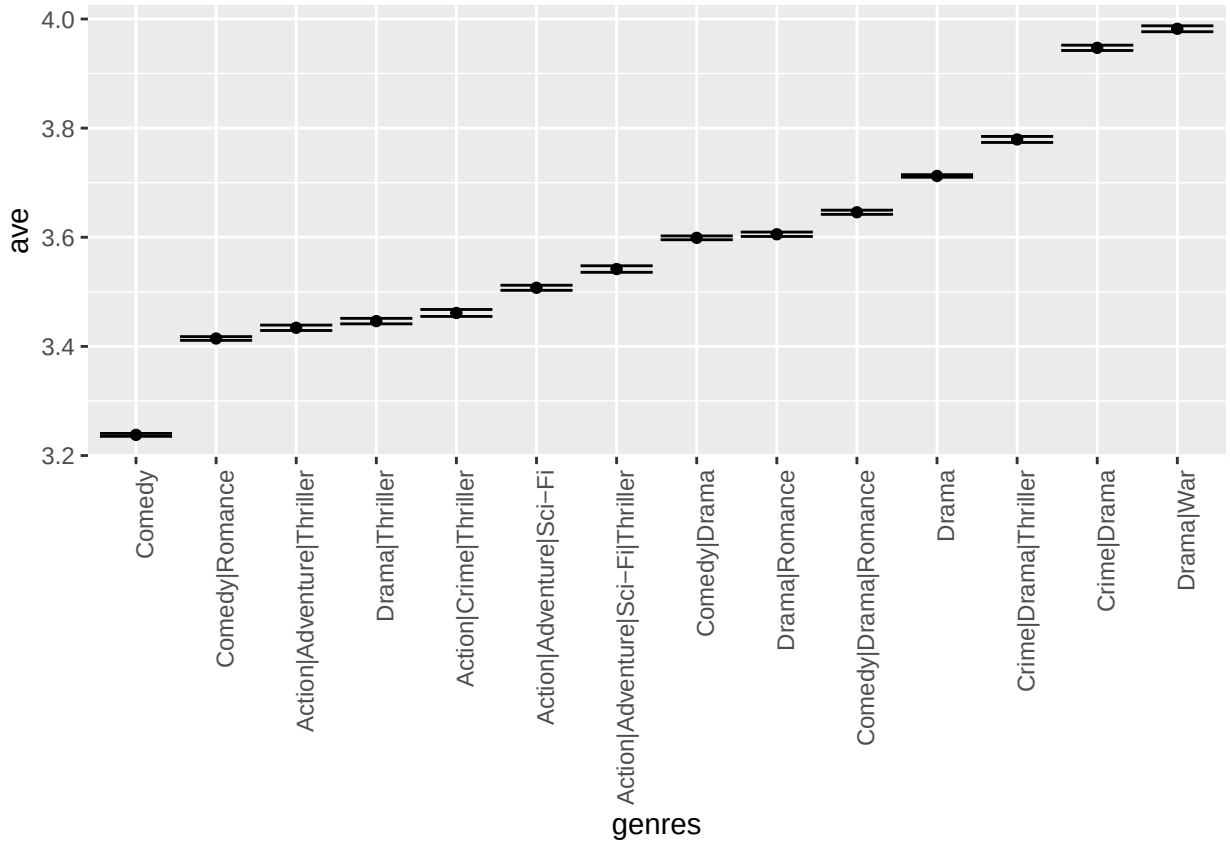
```
## [1] 0.86524
```

This gives us an RMSE of 0.86524 which is only slightly better than 0.86593, but still is a significant improvement.

Personalizing the Prediction

Note that with just these two factors and mu, I have created a great baseline rating that already provides great predictions, but it is not personalized. I tried to calculate for a `user_genre_effect` which would predict the rating of each user for each genre. This is a much simpler way than matrix factorization (the best way to calculate for this personalized effect) but still super effective. Unfortunately, even with such trivial calculations my computer cannot process them so instead I calculated for a `genre_effect`. This does not need to be regularized since its tally is different from the `movie_effect` and `user_effect`. I know there is a genre effect because of this:

```
edx %>% group_by(genres) %>%
  summarize(n = n(), ave = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, ave)) %>%
  ggplot(aes(x = genres, y = ave, ymin = ave - 2*se, ymax = ave + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



As we can see, among the most rated genres, Comedy has the lowest average rating while Drama/War, gets rated the highest.We can calculate for the genre effect through this code:

```
genre_ave <- train_set %>%
  left_join(user_ave_reg, by="userId") %>%
  left_join(movie_ave_reg, by = "movieId") %>%
  group_by(genres) %>%
  summarize(genre_effect = mean(rating - mu - movie_effect - user_effect))

predicted_ratings_4 <- test_set %>%
  left_join(user_ave_reg, by="userId") %>%
  left_join(movie_ave_reg, by="movieId") %>%
  left_join(genre_ave, by= "genres") %>%
  mutate(predictions = mu + movie_effect + user_effect + genre_effect) %>%
  .$predictions

RMSE(test_set$rating,predicted_ratings_4)
```

```
## [1] 0.86494
```

Running the Algorithm to the Entire Training Set (edx)

This algorithm looks good enough. Now I will apply this on the validation set. But first I want to update the algorithm to run on the entire edx set instead of just the train_set. More data is better for an algorithm. This will improve the predictions

```
movie_ave_reg_edx <- edx %>% group_by(movieId) %>%
  summarize(movie_effect = sum(rating - mu)/(n()+4.75))
user_ave_reg_edx <- edx %>%
  left_join(movie_ave_reg_edx, by="movieId") %>% group_by(userId) %>%
  summarize(user_effect = sum(rating - mu - movie_effect)/(n()+4.75))
genre_ave_edx <- edx %>%
  left_join(user_ave_reg_edx, by="userId") %>%
  left_join(movie_ave_reg_edx, by = "movieId") %>%
  group_by(genres) %>%
  summarize(genre_effect = mean(rating - mu - movie_effect - user_effect))

validation_predictions <- validation %>%
  left_join(user_ave_reg_edx, by="userId") %>%
  left_join(movie_ave_reg_edx, by="movieId") %>%
  left_join(genre_ave_edx, by= "genres") %>%
  mutate(predictions = mu + movie_effect + user_effect + genre_effect) %>%
  .$predictions
```

This summarizes my methods.

Results

Now we run the algorithm to the actual test set, `validation`

```
validation_predictions[is.na(validation_predictions)] <- 0 # setting NAs to zero

RMSE(validation$rating,validation_predictions)
```

```
## [1] 0.86445
```

In terms of model performance, this algorithm works well. It is simple yet effective.

Conclusion

Due to the size of the dataset and and the limitations of my computer, I was not able to use more complex algorithms and instead looked for effects that improve the baseline ratings such as the `user_effect` and `movie_effect`. Along with the mean of all the ratings `mu`, these three factors provided a great baseline rating. Regularization helped in lowering the RMSE in a small way but still significant. Personalizing the predicted ratings in this dataset with matrix factorization was not possible. Personalizing the predicted ratings by finding the mean of the residuals after subtracting the `user_effect` and `movie_effect` then grouping by `userId` and `genres` was also not possible with my computer and I instead settled on the `genre` effect. It was crucial for me to run the algorithm again on the entire edx set or I would not be able to surpass the project benchmark for the RMSE. In the end, this simple algorithm was all I needed.