Movie Recommendation System

Nady Monir

10/22/2020

Overview

Here we are trying to model the best possible recommendation system that can be trained and tested using a regular PC, the data and the separation between the test set (`validation`) and the training set (`edx`) are already given in course material, and so this script will appear is separate file, just loading the ready data after saving it in the training scipt file, to ease multiple testing

The approach is simple, we will start with a very simple model, and then build a model after model on top of each other, using the residuals of one to train the next model.

The tried models are as follows:

- The Average
- Average + user effect
- Average + user effect + Movie effect
- Average + Regularization of (user_specific + Movie_specific)
- Average + Regularization of (user_specific + Movie_specific) + Matrix Factorization

To tune these models and test their RMSE, we separate the training set (`edx`) into two parts for training and testing. and at the very end we will pick a model and train the whole `edx` dataset on it and then test it aginst the `validation` dataset to get the final RMSE

Analysis

1. Data Preparation

We will start with loading the data and the required libraries

```
load("C:/Users/Home/projects/Capstone Course/Train & Validation ds.RData")

library(tidyverse)
library(caret)
```

Setting a seed, so that the numbers are the same for every body who tries the code

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

and then Partition the training set (`edx`) into two sets so that we can be able to train/test & tune our models without using the final test set (`validation`), the new test set will be 20% of edx dataset

We are also making sure that the new test set will not include any movie or user that doesn't exist at least once in the new training set

```r
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)

train_set <- edx[-test_index,]
temp <- edx[test_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)
```

2. The average model

We will simply assume / predict that all ratings are the same and equal to the average rating exists in the training set

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

and after testing this simple model the RMSE is as follows:

method	RMSE
The Average	1.059904

3. Adding the User effect

we will build a table that Stratify / Group users and measure the average effect on the ratings after removing the average, remember we are building the model on the residuals of the average model

```r
user_specific<-train_set %>% mutate(rating=rating-mu) %>%
  group_by(userId) %>% summarise(user_avg=mean(rating))
```

And then use the average and the prepared user effects to calculate our prediction on the test set

```r
y_hat_user<-test_set %>% left_join(user_specific) %>% mutate(y_hat=mu+user_avg) %>% pull(y_hat)
```

Now the RMSE has improved as follows:

method	RMSE
The Average	1.0599043
Average + user effect	0.9783971

4. Adding the Movie effect

We will group the traing set by movie and calculate the average for each movie after removing the average and the user effect

```
movie_specific<-train_set %>% left_join(user_specific) %>% mutate(rating=rating-mu-user_avg) %>%
  group_by(movieId) %>% summarise(movie_avg=mean(rating))
```

And then we will use the average and the prepared user effects and the movie effect to calculate our prediction on the test set

```
y_hat_user_movie<-test_set %>% left_join(user_specific) %>%
  left_join(movie_specific) %>%
  mutate(y_hat=mu+user_avg+movie_avg) %>% pull(y_hat)
```

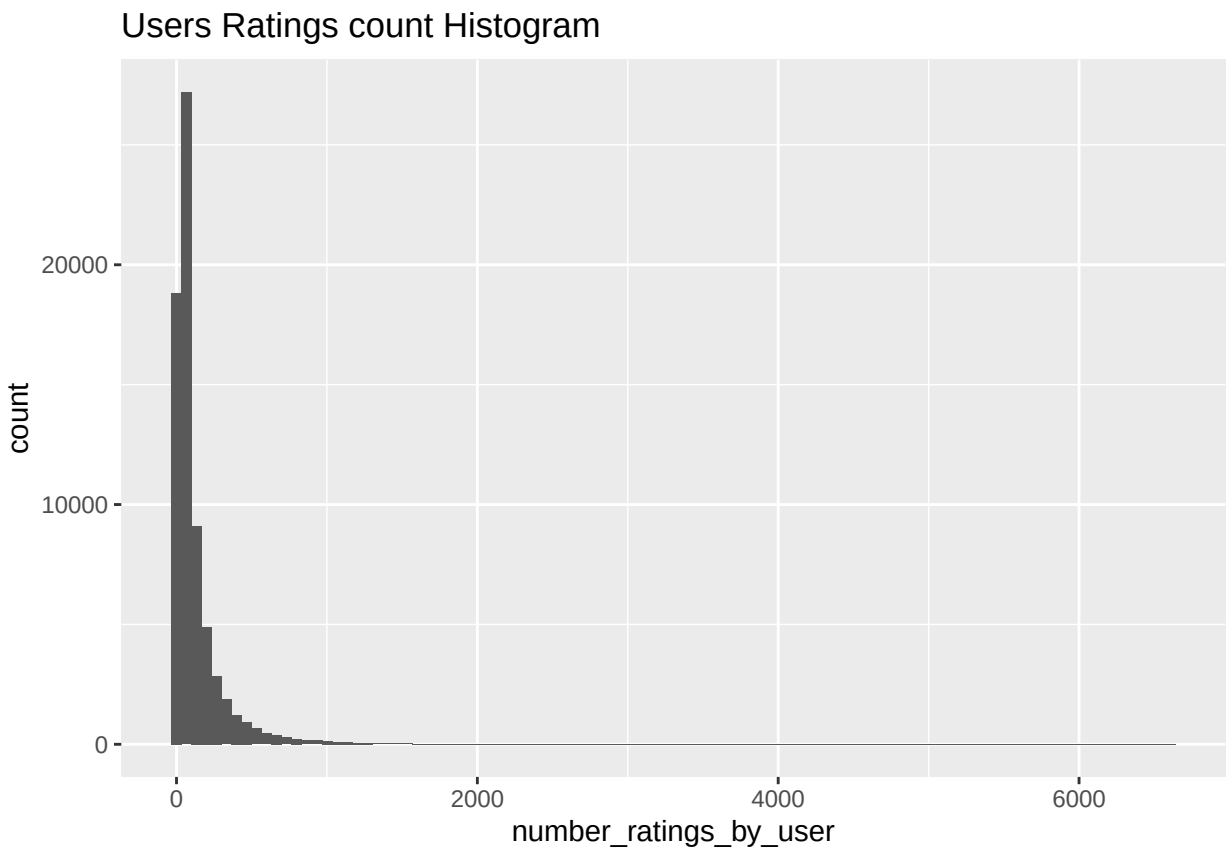
Now the RMSE has improved as follows:

method	RMSE
The Average	1.0599043
Average + user effect	0.9783971
Average + user effect + Movie effect	0.8826307

5. Regularization of users and movies

By exploring the data (`edx`) we can see that some users rated very few movies

```
edx %>% group_by(userId) %>% summarize(number_ratings_by_user=n()) %>%
  ggplot(aes(number_ratings_by_user))+geom_histogram(bins=100)+ggtitle("Users Ratings count Histogram")
```

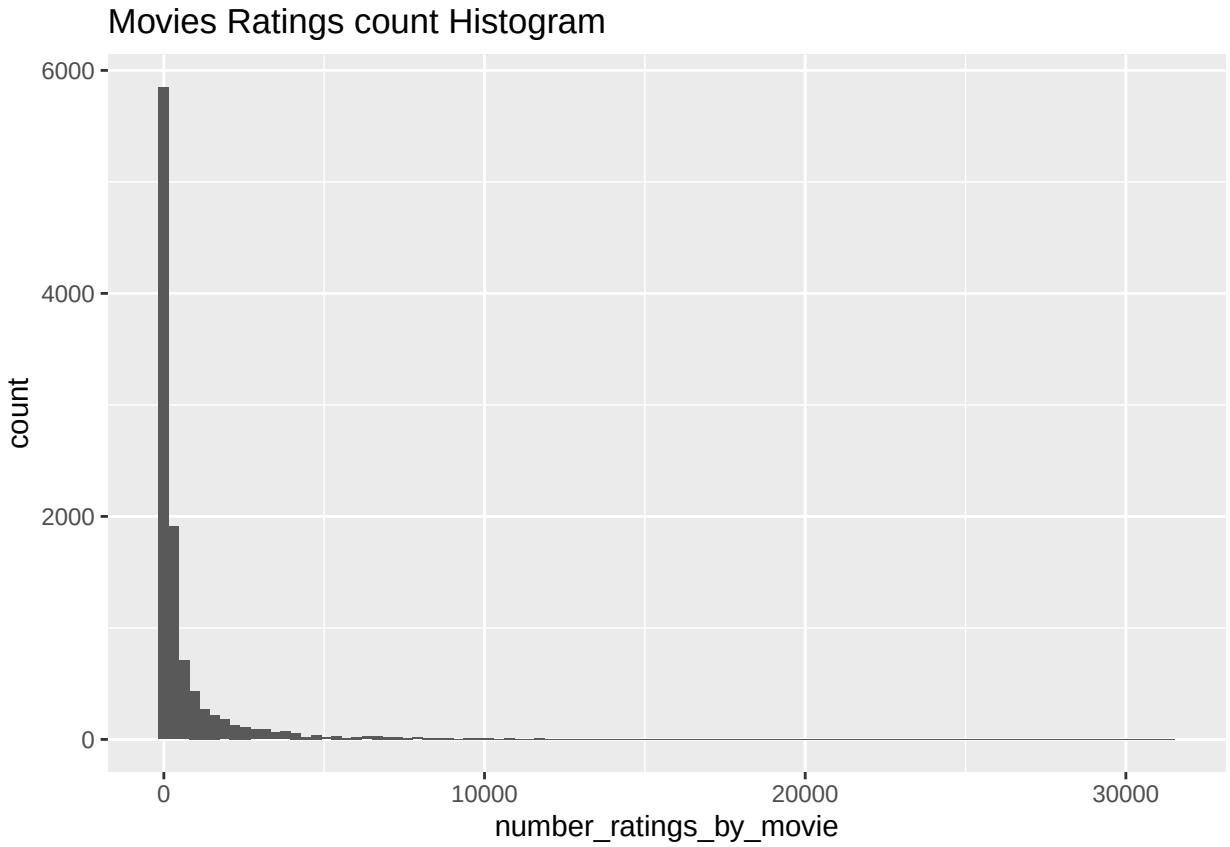


```
edx%>%  group_by(userId) %>% summarise(n=n()) %>% ungroup() %>% arrange(n) %>%  slice(1:10)  %>% knitr:
```

userId	n
62516	10
22170	12
15719	13
50608	13
901	14
1833	14
2476	14
5214	14
9689	14
10364	14

And some movies have been rated very few times, remember the `edx` alone is about **9 M** different ratings

```
edx %>% group_by(movieId) %>% summarize(number_ratings_by_movie=n()) %>%
  ggplot(aes(number_ratings_by_movie))+geom_histogram(bins=100)+ggtitle("Movies Ratings count Histogram
```




```
edx%>%  group_by(movieId) %>% summarise(n=n()) %>% ungroup() %>% arrange(n) %>% slice(1:10) %>% knitr::
```

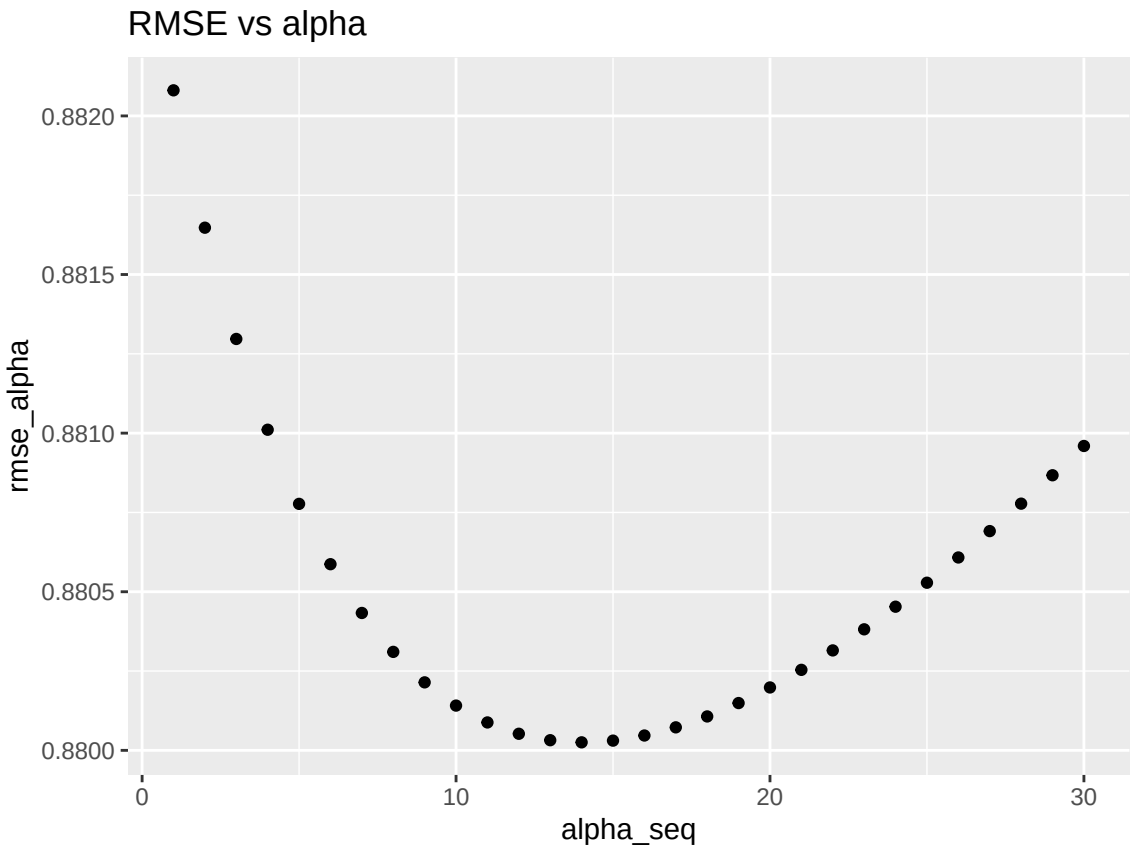
movieId	n
3191	1
3226	1
3234	1
3356	1
3383	1
3561	1
3583	1
4071	1
4075	1
4820	1

and so to decrease the error that can be caused by those few ratings, and because we are measuring RMSE, we would rather be closer to the mean if the number of the ratings are so little, so we will use regularization on users and movies model

we will choose one alpha parameter for the whole model (for users and movies), by selecting the alpha giving the min RMSE

```
alpha_seq=seq(1,30)

rmse_alpha=sapply(alpha_seq, function(a){

  user_specific<-train_set %>% mutate(rating=rating-mu) %>%
    group_by(userId) %>% summarise(user_avg=sum(rating)/(n()+a))

  movie_specific<-train_set %>% left_join(user_specific) %>% mutate(rating=rating-mu-user_avg) %>%
    group_by(movieId) %>% summarise(movie_avg=sum(rating)/(n()+a))

  y_hat_user_movie<-test_set %>% left_join(user_specific) %>%
    left_join(movie_specific) %>%
    mutate(y_hat=mu+user_avg+movie_avg) %>% pull(y_hat)

  RMSE(test_set$rating,y_hat_user_movie)
})
```

that will give us the following plot between the alpha and the RMSE :



Now the RMSE is as follows:

method	RMSE
The Average	1.0599043
Average + user effect	0.9783971
Average + user effect + Movie effect	0.8826307
Average + Regularization of (user_specific + Movie_specific)	0.8800253

Before we go on to the next model, we have to re-prepare the user and movie effects table with the new alpha parameter (that minimized the RMSE), so that we will be able to use the correctly

```
user_specific<-train_set %>% mutate(rating=rating-mu) %>%
  group_by(userId) %>% summarise(user_avg=sum(rating)/(n()+alpha))

movie_specific<-train_set %>% left_join(user_specific) %>% mutate(rating=rating-mu-user_avg) %>%
  group_by(movieId) %>% summarise(movie_avg=sum(rating)/(n()+alpha))
```

6. Matrix Factorization

The last model the we will try is to detect unseen correlations or grouping between user and/or movies, and the Matrix Factorization approach is the one we choose here.

The main challenge here is to do this model on a common PC, Matrix Factorization by default is an intensive memory using method, and it can crash easily with not very large dataset, and it also requires a great deal of proccessing, so we would rather use a multi-thread solution

To solve those main challenges we could use `recosystem` package, it allows multi-threading and out of memory calculations, and it is specialized in building a recommendation system between users and any "item" (here, it is a movie)

We start by cacluating the residuals from our last model

```
residual_training<- train_set %>% left_join(user_specific) %>% left_join(movie_specific) %>%
  mutate(res=rating-mu-user_avg-movie_avg)

residual_test<- test_set %>% left_join(user_specific) %>% left_join(movie_specific) %>%
  mutate(res=rating-mu-user_avg-movie_avg)
```

and then prepare the datasources for the model

```
train_reco <-  with(residual_training, data_memory(user_index = userId,
                                    item_index = movieId,
                                    rating = res))
test_reco  <-  with(residual_test, data_memory(user_index = userId,
                                               item_index = movieId,
                                               rating = res))
```

We will build the model and tune it with the common settings that is being used

```
model <-  recosystem::Reco()
model$tune(train_reco, opts = list(dim      = c(10L, 20L),
                                costp_l1 = c(0, 0.1),
                                costp_l2 = c(0.01, 0.1),
                                costq_l1 = c(0, 0.1),
                                costq_l2 = c(0.01, 0.1),
                                lrate    = c(0.01, 0.1),
                                nthread = 4, niter = 20))
model$train(train_reco, opts = c( nthread = 4, niter = 20))
```

and then we use it to make prediction on the test set, which was also prepared above

```
residual_prediction <-  model$predict(test_reco, out_memory())
y_hat<-test_set %>% left_join(user_specific) %>% left_join(movie_specific) %>%
  mutate(residual_prediction=residual_prediction) %>%
  mutate(y_hat=mu+user_avg+movie_avg+residual_prediction) %>%
  pull(y_hat)
```

Testing...

method	RMSE
The Average	1.0599043
Average + user effect	0.9783971
Average + user effect + Movie effect	0.8826307
Average + Regularization of (user_specific + Movie_specific)	0.8800253
Average + Regularization of (user_specific + Movie_specific) + Matrix Factorization	0.8344657

Results

To get the final result we should use the `edx` dataset to train the model and test it on the `validation` dataset and since the last model that includes the Matrix Factorization have the smallest RMSE, we will implement it on `edx`

Note that we are using the same `alpha` as before, since we can't retune it using `validation` dataset, the `validation` dataset is assumed to be unknown when we train the model

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

user_specific<-edx %>% mutate(rating=rating-mu) %>%
  group_by(userId) %>% summarise(user_avg=sum(rating)/(n()+alpha))

movie_specific<-edx %>% left_join(user_specific) %>% mutate(rating=rating-mu-user_avg) %>%
  group_by(movieId) %>% summarise(movie_avg=sum(rating)/(n()+alpha))

residual_training<- edx %>% left_join(user_specific) %>% left_join(movie_specific) %>%
  mutate(res=rating-mu-user_avg-movie_avg)

residual_test<- validation %>% left_join(user_specific) %>% left_join(movie_specific) %>%
  mutate(res=rating-mu-user_avg-movie_avg)

train_reco <-  with(residual_training, data_memory(user_index = userId,
                                                   item_index = movieId,
                                                   rating = res))
test_reco  <-  with(residual_test, data_memory(user_index = userId,
                                               item_index = movieId,
                                               rating = res))

model <-  recosystem::Reco()

model$tune(train_reco, opts = list(dim      = c(10L, 20L),
                                   costp_l1 = c(0, 0.1),
                                   costp_l2 = c(0.01, 0.1),
                                   costq_l1 = c(0, 0.1),
                                   costq_l2 = c(0.01, 0.1),
                                   lrate    = c(0.01, 0.1),
                                   nthread = 4, niter = 20))

model$train(train_reco, opts = c( nthread = 4, niter = 20))
```

Now we extract the final Prediction

```
residual_prediction <-  model$predict(test_reco, out_memory())

y_hat<-validation %>% left_join(user_specific) %>% left_join(movie_specific) %>%
  mutate(residual_prediction=residual_prediction) %>%
  mutate(y_hat=mu+user_avg+movie_avg+residual_prediction) %>%
  pull(y_hat)
```

and Here is the Final RMSE

method	RMSE
The Average	1.0599043
Average + user effect	0.9783971
Average + user effect + Movie effect	0.8826307
Average + Regularization of (user_specific + Movie_specific)	0.8800253
Average + Regularization of (user_specific + Movie_specific) + Matrix Factorization	0.8344657
Applying on Validation dataset	0.8333468

Conclusion

We started with a very simple model which is just the mean of the observed ratings. From there, we added movie and user effects, then With regularization we added a penalty value for the movies and users with few number of ratings. The model achieved the RMSE of 0.8800253, and it simulate a linear regression model (ratings ~ user + movie)

Finally, we used the recosystem package that implements the Matrix Factorization algorithm, and achieved the RMSE of 0.8333468 on `validation` dataset

Limitations & Future Work

While we still didn't take into account the dates of the ratings and the genres of the movies, the issue year of the movies, we may be able to better predict the ratings, Also the users data (which isn't included here at all), for example his country, may also have some effect on the ratings as well.

The challenge is with the increasing every minute, it is harder / more expensive to compute complex models.