HarvardX PH125.9x All Learners Capstone Project

Michael Lai

2023-03-08

Introduction & Overview

This report is part of the final submission for the "All Learners Capstone Project" of the Massive Open Online Course (MOOC) HarvardX PH125.9x, which is the last of a series of courses comprising its Data Science Program. Detailed information of the project can be obtained from the overview section of the course, and below is a summary.

In short, we have to build a movie recommendation system from the given dataset called "movielens" which can be extracted from the dslabs R package. Upon processing the dataset into a training and a test set, we are required to use the training set to develop an algorithm to reduce the estimation error, measured in Root Mean Square Error (RMSE), in the final test.

As its names suggests, RMSE is the square root of the Mean Squared Error (MSE), which is in turn defined by the sum of all squared values of the differences between predicted and actual ratings (aka squared errors), divided by the total no. of predictions. In a prefect prediction scenario, there is not such differences and hence the RMSE would be 0. As for this project, an RMSE of below 0.86490 would entail full mark of that particular grading rubric.

The script of that algorithm has been written and saved in another file submitted along with this report, while this report has also knitted the codes and results of the script for demonstration.

The code for pre-processing the original dataset into training and test sets, and the resulting files are provided by the project guidelines. They are shown below:

```
# Prepare two raw files for processing, ratings_file & movies_file
dl <- "ml-10M100K.zip"
if(!file.exists(dl)) download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file)) unzip(dl, ratings_file)
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file)) unzip(dl, movies_file)

# Tidy the ratings file
ratings <- as.data.frame(str_split(read_lines(ratings_file),
                                   fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>% mutate(userId = as.integer(userId), movieId = as.integer(movieId),
                              rating = as.numeric(rating), timestamp = as.integer(timestamp))

# Tidy the movies file
movies <- as.data.frame(str_split(read_lines(movies_file),
                                  fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
```

```
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>% mutate(movieId = as.integer(movieId))

# Create a master database by joining the movies & ratings files
movielens <- left_join(ratings, movies, by = "movieId")

# Separate the master database into training and test sets
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

final_holdout_test <- temp %>% semi_join(edx, by = "movieId") %>% semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into training set
edx <- rbind(edx, anti_join(temp, final_holdout_test))

# Remove redundant files
rm(dl, ratings, movies, test_index, temp, movies_file, ratings_file)
```

The training dataset is stored in the file edx, while the test set is stored in final_handout_test. In the next section, we shall take a brief look at the datasets, estimate the effects of individual characteristics of the movies and users on the ratings, then take a look at the relationship between the residual average ratings and review counts per movie to see if we can further reduce the RMSE with that information.

The RMSE of the test set is compiled along each step, with the value presented at the end of the report being the final RMSE generated by the algorithm.

Data summary

First of all, a preliminary assessment is carried out to find out the basic parameters of the training data set.

```
nrow(edx)
```

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

There are 9,000,055 movie reviews in the training set.

```
n_distinct(edx$movieId)
```

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

There are 10,677 movies.

```
n_distinct(edx$userId)
```

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

There are 69,878 users.

As for the test set,

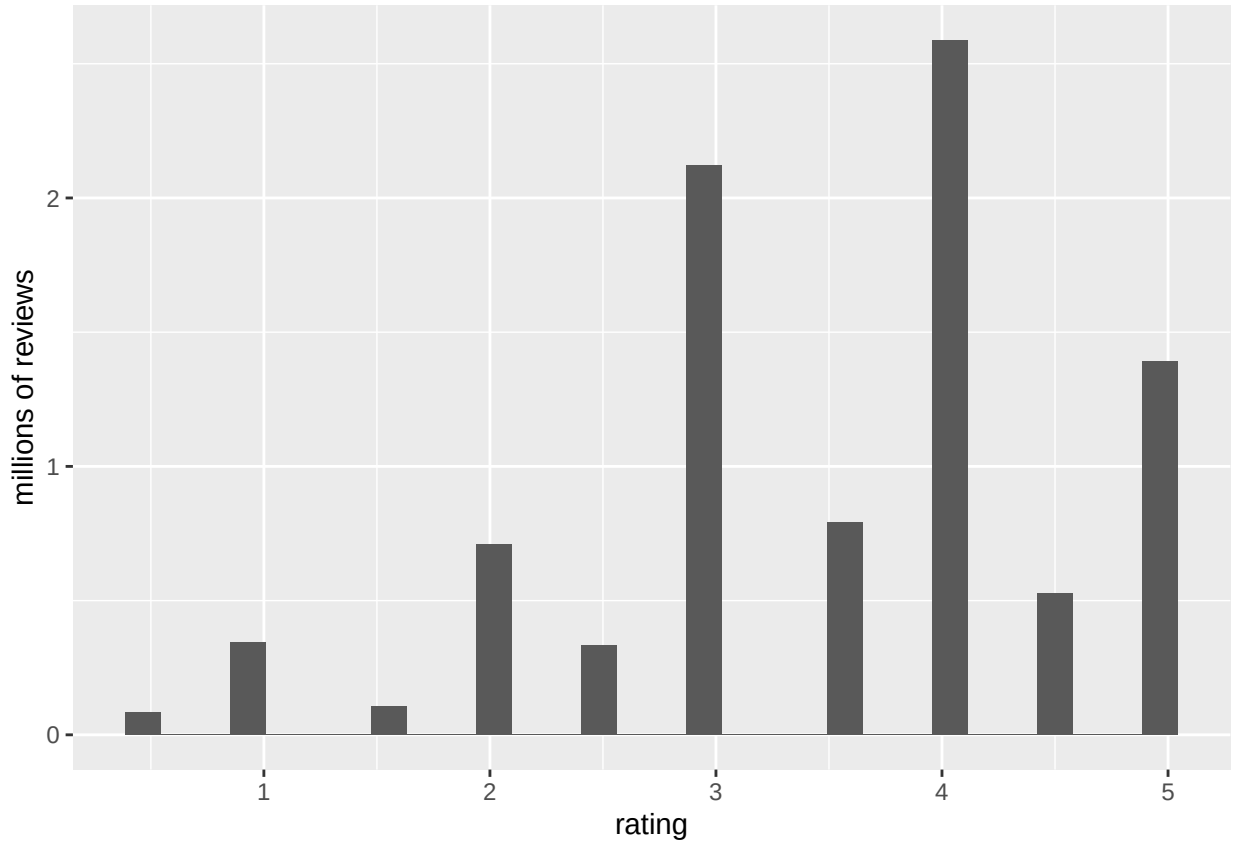
```
head (final_holdout_test, n=5)
```

```
##   userId movieId rating timestamp                 title                              genres
## 1      1     231      5 838983392  Dumb & Dumber (1994)                            Comedy
## 2      1     480      5 838983653  Jurassic Park (1993) Action|Adventure|Sci-Fi|Thriller
## 3      1     586      5 838984068     Home Alone (1990)                   Children|Comedy
## 4      2     151      3 868246450        Rob Roy (1995)         Action|Drama|Romance|War
## 5      2     858      2 868245645 Godfather, The (1972)                     Crime|Drama
```

There are a total of 999,999 records in the test set.

A plot of the ratings shows its overall distribution.

```
ggplot(edx, aes(x = rating, y=..count../1000000)) +
  geom_histogram() + xlab("rating") + ylab("millions of reviews")
```



The ratings range from 0.5 to 5, with 4 being the most common choice. Also, half scores (e.g. 3.5) are much less frequent than full scores.

The average rating across all movies is 3.51.

```
mu <- mean(edx$rating) %>% print
```

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

The naive root mean square error (nRMSE) is the RMSE calculated by simply using the overall rating mean in the training set to predict ratings in the test set, i.e.

```
nRMSE <- sqrt(mean((final_holdout_test$rating - mu)^2)) %>% print
```

```
## [1] 1.0612
```

This is the baseline RMSE on which we are to improve. And by subtracting the mean score from all ratings, we normalize the scores around the mean and hence work on minimizing the residual values only.

```
edx$rating = edx$rating - mu
```

A positive score means an above average rating while a negative score means otherwise. With such basic parameters in mind, in the next section we will compile, in order, the following effects to include in our rating prediction algorithm.

Methods & Analysis

1. The movie effect

The first step of the algorithm is to predict the review ratings simply with the average rating of each movie. It is expected that some movies have generally better reviews than others. The average ratings of each movie, after normalization, are computed.

Nonetheless, as the ratings of unpopular movies - i.e. movies with few review counts - are not as reliable as those with many, the possibility of adding a penalty term in estimating the average ratings to reduce the deviation from the overall mean (mu) of those movies, is explored. The modified rating average for a particular movie is called weighted average rating, and is defined below:

weighted average rating = (sum of ratings of the movie) / (lambda + no. of ratings of the movie)

Where lambda is a SINGLE coefficient that affects the weightings of all penalty terms. When lambda = 0, the weighted averages are just the original average movie ratings; as lambda gets larger the weighted average ratings get smaller, so does the effect any particular movie has on rating estimation. The appropriate choice of lambda that can produce the least RMSE as a whole is calculated by a tuning process, which involves using cross-validation to estimate the average MSE from the training set. This will be further explained in the results section.

After determining the weighted average ratings, the algorithm is represented by

prediction of a review rating = (weighted average rating of the movie)

We will then deduct such estimations from the training set before proceeding to the investigate residual ratings with the second effect.

2. The user effect

The next step of the algorithm is to predict the residual ratings with the average rating of each user. Similar to movie ratings, it is expected that some users have generally higher reviews than others. The average ratings of all users, after normalization and deducting average movie ratings, are computed.

Again, as the ratings of rare users are not as reliable as those with many, the possibility of adding a penalty term is also explored by a similar equation:

weighted average rating = (sum of ratings of the user) / (lambda + no. of ratings of the user)

The appropriate value of lambda is also calculated by the same tuning and cross validation processes of the movie effect. A nd a fter d etermining t he w eighted a verage r atings, t he a lgorithm i s r epresented by

prediction of a review rating = (weighted average rating of the movie) + (weighted average rating of the user)

Yet before deciding to deduct such rating estimations from the training set, we will investigate the third effect - t he g enre e ffect to se e ho w we ca n re fine the estimation.

3. The genre effect

It is speculated that the genres of a movie might tell how users review it, as different users have different tastes. Hence every genre in the genres field of each movie is separated and gathered under a separate row to create a movie genre table, with each row indicating a single genre present for a movie. A total of 19 genres are present across all movies.

Next a user genre chart is created which takes the average rating by genre for each user, so now the residual user ratings are classified i nto d ifferent ge nre av er ages. Wi th su ch in formation th e ra tings ca n be predicted according to the genres to which the movie belongs:

prediction of a review rating = (weighted average rating of the movie) + (average rating of the user for all genre averages to which the movie belongs)

If the user has no ratings for any of the genres of the movie in question, the overall user rating will be used instead.

We will then deduct such estimations from the training set before proceeding to the investigate residual ratings with the fourth effect.

4. The review count effect

For the final fa ctor w e w ill in vestigate t h e r elationship b etween t h e r esidual m ovie r atings a n d t h eir n o . of reviews by computing the correlation between the two factors. If the results show a moderate relationship, we will fit i t w ith a r egression m odel b y t uning t he c omplexity p arameter t hrough cross-validation.

Under such scenario, the final a lgorithm w ill b e r epresented by

prediction of a review rating = (weighted average rating of the movie) +

(average rating of the user for all genre averages to which the movie belongs) OR

(weighted average rating of the user if no ratings for any of the genres) + (average rating by review count)

This will be used to evaluate the RMSE in the test set.

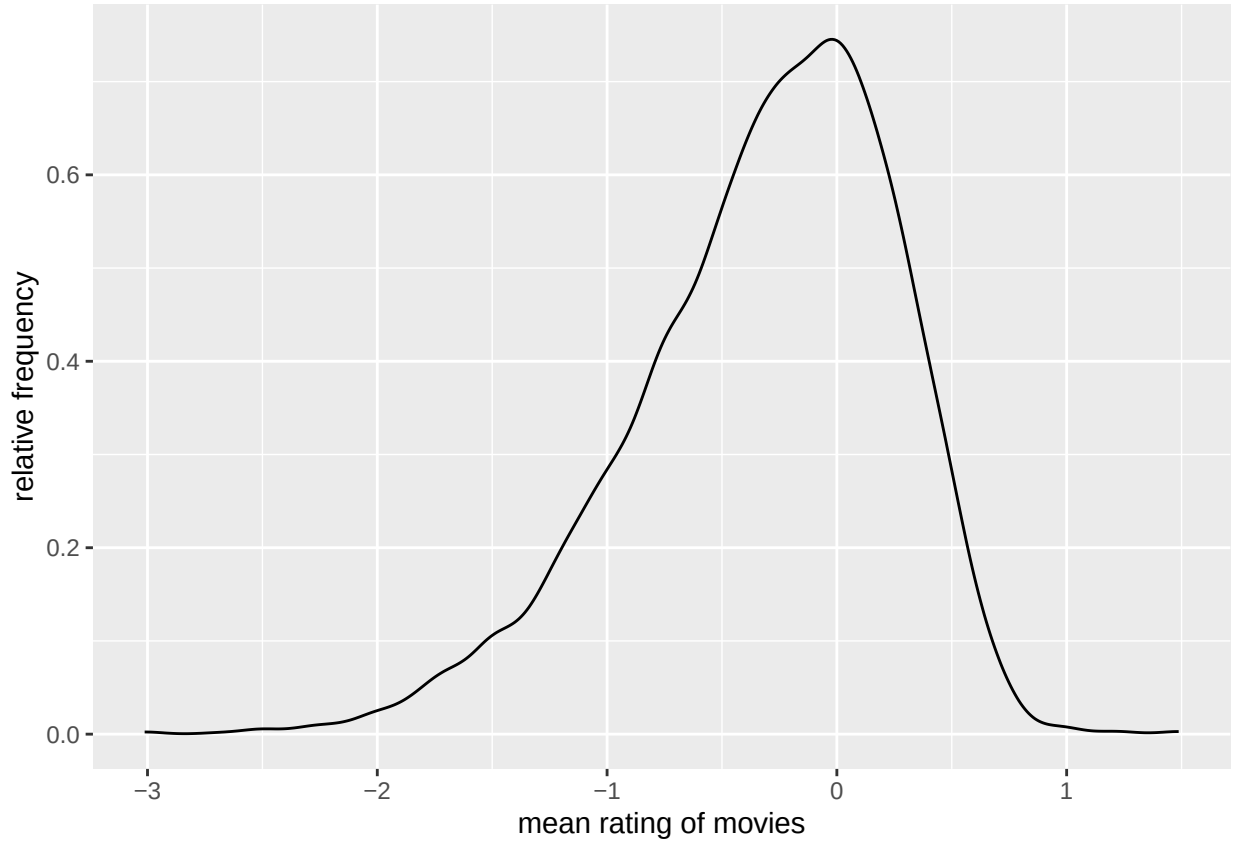
Results

1. The movie effect

The average rating of each movie is computed in the table mov_avg, with its distribution plotted below.

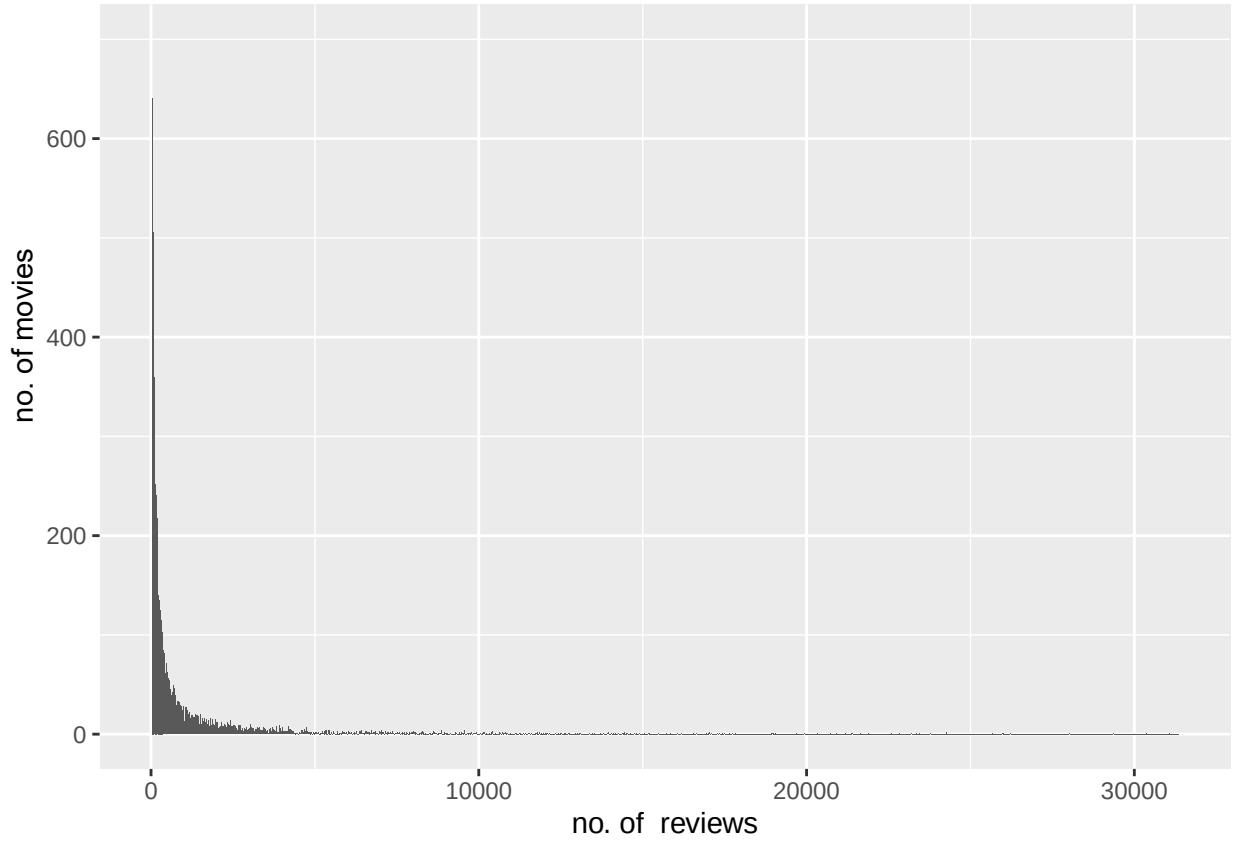
```
mov_avg <- edx %>%
  group_by(movieId) %>%
  summarize(mov_avg = mean(rating), count=n()) %>%
  arrange (movieId)

ggplot(mov_avg, aes(mov_avg)) + geom_density() +
  xlab("mean rating of movies") + ylab("relative frequency")
```



The distribution of the movies by their review count is given below.

```
ggplot(mov_avg, aes(count, ..count..)) + geom_histogram(binwidth=20) +
  xlab("no. of  reviews") + ylab("no. of movies")
  + xlim(0,max(mov_avg$count)) + ylim(0,700)
```



It can be shown that the number of reviews for each movie varies extremely, with most movies receiving only several ratings while the most popular ones receiving tens of thousands. Hence as mentioned before, the possibility of adding a penalty term to reduce the deviation from the overall mean is explored. First the index matrix i is created to partition the training set into 10 equal subsets (k1 to k10) as validation sets to tune the penalty parameter lambda in the cross-validation process.

```
i <- createDataPartition(edx$rating, 10, 0.1, FALSE)
colnames(i) <- paste("k", c(1:10),sep = "")
print(i[1:10,])
```

```
##       k1  k2 k3  k4 k5  k6  k7 k8  k9 k10
##  [1,]  4   1 13   2  1  47   5 15  24  42
##  [2,]  6  21 31  24 13  50   9 19  39  49
##  [3,] 34  26 50  25 16  64  30 22  74  57
##  [4,] 39  34 55  35 34  69  39 24  89  63
##  [5,] 45  90 59  52 37  71  65 31 128  67
##  [6,] 46 102 60  75 49  84  73 44 131  81
##  [7,] 55 103 70  78 62  96  76 51 134  88
##  [8,] 56 111 71  81 65 100  95 60 144  96
##  [9,] 61 136 79  93 72 116 101 68 145  99
## [10,] 62 153 83 115 92 118 105 74 146 101
```

With each subset k1 to k10, its counterpart records are used to determine the weighted mean ratings. Then the validation subset is joined to the counterpart records by movieId, and MSE in each subset is computed. The average MSE across all 10 subsets are in turn computed. Finally, we will feed lambda values between 0

and 4 into the function to see which one gives the lowest RSME. The code below and executes the procedures described above.

```
avg_mov_MSE <- function (lambda) mean(
  sapply(1:10, function(k)
    edx[setdiff(1:nrow(edx),i[,k]),2:3] %>%
      group_by(movieId) %>%
      summarize(mov_avg = sum(rating)/(lambda + n())) %>%
      inner_join(edx[i[,k],2:3], "movieId")%>%
      group_by(movieId) %>%
      summarize(SSE = sum((rating - mov_avg)^2), penalty= sum(mov_avg^2)/n() * lambda,
      count=n()) %>% summarize(MSE = sum(SSE + penalty)/sum(count)) %>% pull(MSE)))

lambda<-seq(0,4)

sapply(lambda, avg_mov_MSE)
```

```
## [1] 0.88965 0.89366 0.89737 0.90083 0.90411
```

Results show that the MSE is the least when lambda is zero. In other words, it is actually not conducive to adding a penalty term, so we will simply stick to the original mean rating for each movie as the rating estimate.

With this parameter set, we now try to evaluate the RMSE in the test set just by using original mean ratings obtained from the training set. Keep in mind the predicted ratings are now standardized, so we have to subtract mu from the ratings in the test set first.

```
RMSE <- final_holdout_test %>%
  select(movieId, rating) %>%
  inner_join(mov_avg, "movieId") %>%
  mutate(SE = (rating - mu - mov_avg)^2) %>%
  pull(SE) %>% mean() %>% sqrt() %>% print
```

```
## [1] 0.94391
```

This RMSE is significantly better than the naive RMSE.

We then subtract the movie averages from the training set ratings to further analyze the residual errors by user effect.

```
edx <- edx %>% inner_join(mov_avg, "movieId") %>% mutate(rating = rating - mov_avg)
```

---

2. The user effect

An average for the user effect, user_avg, similar to that of movie is also determined. Remember now the averages are taken AFTER subtracting the movie mean ratings so they are much smaller.

```
user_avg <- edx[,c(1,2,3)] %>%
  group_by(userId) %>%
  summarize(user_avg = mean(rating), count=n()) %>%
  arrange(userId) %>% print
```

```
## # A tibble: 69,878 x 3
##    userId user_avg count
##     <int>    <dbl> <int>
##  1      1   1.68      19
##  2      2  -0.236     17
##  3      3   0.264     31
##  4      4   0.652     35
##  5      5   0.0853    74
##  6      6   0.346     39
##  7      7   0.0238    96
##  8      8   0.203    727
##  9      9   0.232     21
## 10     10   0.0833   112
## # ... with 69,868 more rows
```

Again we will use cross-validation to test if the RMSE can be further reduced by adding a penalty term. Likewise, the training set is divided into 10 equal subsets and with the indexes stored in i. The counterpart records of each subset are used to determine the weighted mean ratings. Then the validation subset is joined to the counterpart records and again the MSE in each subset is computed. The average MSE across all 10 validations are in turn computed, and lambda values between 0 and 4 are fed into the function.

```
i <- createDataPartition(edx$rating, 10, 0.1, FALSE)

user_avg_MSE <- function (lambda) mean(
  sapply(1:10, function (k)
    edx[setdiff(1:nrow(edx),i[,k]),1:3] %>%
      group_by(userId) %>%
      summarize(user_avg = sum(rating)/(lambda + n())) %>%
      inner_join(edx[i[,k],1:3], "userId") %>%
      group_by(userId) %>%
      summarize(SSE =sum((rating -user_avg)^2), penalty =sum(user_avg^2)/n() *lambda,
      count=n()) %>% summarize(MSE =sum(SSE +penalty)/sum(count)) %>% pull(MSE)))

lambda <- c(0:4)

sapply(lambda, user_avg_MSE)
```

```
## [1] 0.74710 0.75931 0.77047 0.78075 0.79026
```

Again, the average MSE is the least when lambda is zero, so we will just stick to the original user average as the estimate. And with this we try again to evaluate the RMSE. Keep in mind we have to subtract both mu and the movie mean from the ratings in the test set, in addition to the user averages.

```
RMSE <- final_holdout_test %>%
  select(movieId, userId, rating) %>%
  inner_join(mov_avg, "movieId") %>%
  inner_join(user_avg, "userId") %>%
  mutate(SE = (rating - mu - mov_avg - user_avg)^2) %>%
  pull(SE) %>% mean() %>% sqrt() %>% print
```

```
## [1] 0.86535
```

This is another significant improvement. But this time instead of subtracting user averages from the training set ratings, we want to see if we can further pinpoint the preferences of users for different movie genres.

---

3. The genre effect

As mentioned earlier, it is suggested that the genres of a movie might have different effect for different users. However, most movies belong to more than one genre and such genre information is lumped together in one single field:

```
head(edx[1:5, c(2,5,6)])
```

```
##   movieId                          title                        genres
## 1     122              Boomerang (1992)                Comedy|Romance
## 2     185               Net, The (1995)          Action|Crime|Thriller
## 3     292               Outbreak (1995)  Action|Drama|Sci-Fi|Thriller
## 4     316               Stargate (1994)       Action|Adventure|Sci-Fi
## 5     329 Star Trek: Generations (1994) Action|Adventure|Drama|Sci-Fi
```

We have to tidy it up to work on this. As the genres in the field are separated by the symbol "|", the first action is to determine the maximum number of genres a movie will have.

```
max(str_count(edx$genres, "[|]"))
```

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

This means the maximum number of genres a movie can have is 8 (there is no need for the symbol with just one genre). With this information, the next step is to separate the genres column into a maximum of 8 columns and gather each under a separate row. Then the empty columns and the genre with the entry "(no genres listed)" are removed, and the remaining genres grouped together:

```
data.frame(genres = edx[,6]) %>%
  separate(genres, as.character(seq(1:8)),"[|]") %>%
  gather(genre_no, genre, 1:8) %>%
  filter(!is.na(genre) & genre !="(no genres listed)") %>%
  group_by(genre) %>% summarize(count=n())
```

```
## # A tibble: 19 x 2
##    genre         count
##    <chr>         <int>
##  1 Action      2560545
##  2 Adventure   1908892
##  3 Animation    467168
##  4 Children     737994
##  5 Comedy      3540930
##  6 Crime       1327715
##  7 Documentary   93066
##  8 Drama       3910127
##  9 Fantasy      925637
## 10 Film-Noir    118541
```

```
## 11 Horror       691485
## 12 IMAX           8181
## 13 Musical      433080
## 14 Mystery      568332
## 15 Romance     1712100
## 16 Sci-Fi      1341183
## 17 Thriller    2325899
## 18 War          511147
## 19 Western      189394
```

We can see there are a total of 19 genres present for all movies. Next, we create a movie genre data frame (mov_gen) by separating the genres by each movie as above.

```
mov_gen <- edx[,c(2,6)] %>%
  group_by(movieId) %>% slice(1) %>%
  separate(genres, paste("genre", seq(1:8), sep=""),"[|]") %>%
  gather(count, genre, genre1:genre8) %>%
  filter(!is.na(genre) & genre !="(no genres listed)") %>%
  select(movieId, genre) %>%
  arrange(movieId, genre)
```

By assigning 1 to each genre present for each movie, spreading out the genres and filling the NA spaces by 0, a movie genre chart is shown.

```
mutate(mov_gen, review = 1) %>% spread(genre, review, 0)
```

```
## # A tibble: 10,676 x 20
## # Groups:   movieId [10,676]
##    movieId Action Adventure Animat~1 Child~2 Comedy Crime Docum~3 Drama Fantasy Film-~4
##      <int>  <dbl>     <dbl>    <dbl>   <dbl>  <dbl> <dbl>   <dbl> <dbl>   <dbl>   <dbl>
##  1       1      0         1        1       1      1     0       0     0       1       0
##  2       2      0         1        0       1      0     0       0     0       1       0
##  3       3      0         0        0       0      1     0       0     0       0       0
##  4       4      0         0        0       0      1     0       0     1       0       0
##  5       5      0         0        0       0      1     0       0     0       0       0
##  6       6      1         0        0       0      0     1       0     0       0       0
##  7       7      0         0        0       0      1     0       0     0       0       0
##  8       8      0         1        0       1      0     0       0     0       0       0
##  9       9      1         0        0       0      0     0       0     0       0       0
## 10      10      1         1        0       0      0     0       0     0       0       0
## # ... with 10,666 more rows, 9 more variables: Horror <dbl>, IMAX <dbl>, Musical <dbl>,
## #   Mystery <dbl>, Romance <dbl>, Sci-Fi <dbl>, Thriller <dbl>, War <dbl>, Western <dbl>,
## #   and abbreviated variable names 1: Animation, 2: Children, 3: Documentary, 4: Film-Noir
```

Next, a user genre table (user_gen) is created in a similar manner to mov_gen, but it takes the average rating by genre for each user instead.

```
user_gen <- edx[,c(1,2,3,6)] %>%
  separate(genres, paste("genre", seq(1:8), sep=""),"[|]") %>%
  gather(count, genre, genre1:genre8) %>%
  filter(!is.na(genre) & genre !="(no genres listed)")%>%
  group_by(userId, genre) %>%
  summarize(gen_avg = mean(rating)) %>%
  arrange(userId, genre)  %>% print
```

```
## # A tibble: 1,100,981 x 3
## # Groups:   userId [69,878]
##    userId genre     gen_avg
##     <int> <chr>       <dbl>
##  1      1 Action       1.74
##  2      1 Adventure    1.49
##  3      1 Animation    1.40
##  4      1 Children     1.61
##  5      1 Comedy       1.87
##  6      1 Crime        2.06
##  7      1 Drama        1.39
##  8      1 Fantasy      1.94
##  9      1 Musical      1.32
## 10      1 Romance      1.52
## # ... with 1,100,971 more rows
```

So now we have sub-divided the residual average ratings of each user into different genres - if they have watched movies in those genres before. With such information the ratings can be better predicted according to the genres to which the movie belongs.

For example, "Jurassic Park (1993; movieId 480)" belongs simultaneously to Action, Adventure, Sci-Fi & Thriller, and userId 4 has ratings for all these four genres. Hence his/her user rating for this movie will be an average of all these four ratings; "The Godfather (1972); movieId 858)" belongs to both Crime & Drama, but userId 2 has rating on Drama only. Hence his/her rating on Drama will be used solely as the rating for this movie.

If the user has no ratings for any of the genres of the movie in question, his/her overall user rating computed earlier (user_avg) will be used instead.

With this refined model, we compute the RMSE for the test set again.

```
RMSE <- final_holdout_test %>%
  inner_join(mov_avg, "movieId") %>%
  inner_join(user_avg, "userId") %>%
  inner_join(mov_gen, "movieId") %>%
  left_join(user_gen, c("userId", "genre")) %>%
  group_by(movieId, userId, rating, mov_avg, user_avg) %>%
  summarize(gen_avg = mean(gen_avg, na.rm = T)) %>%
  mutate(SE = (rating - mu - mov_avg - ifelse(is.na(gen_avg),user_avg,gen_avg))^2) %>%
  pull(SE) %>% mean() %>% sqrt() %>% print
```

```
## [1] 0.84998
```

This improves the RMSE to below 0.85000, already surpassing the project assessment requirement. Yet there is one more effect we would like to investigate - the no. of movie reviews per movie (the review count).

Before that, however, we have to further subtract the genre/user averages from the training set ratings first. Note seven reviews with the genre "(no genres listed)" are removed from here onward. They all belong to just one movie and hence it does not have much impact of the overall model.

```
edx <- edx %>%
  select(movieId, userId, rating) %>%
  inner_join(mov_gen, "movieId") %>%
  inner_join(user_gen, c("userId", "genre")) %>%
  group_by(movieId, userId, rating) %>%
```

```
  summarize(gen_avg = mean(gen_avg, na.rm = T)) %>%
  mutate(rating = rating - gen_avg)
```

---

4. The review count effect

To study the relationship between the residual movie ratings and the no. of reviews per movie, we first calculate the average ratings for each movie in the table rsd_mov_avg. Then we compute the correlation between the two variables.

```
rsd_mov_avg <- edx %>% group_by(movieId) %>%
  summarize(rsd_mov_avg=mean(rating), no_of_reviews=n())

cor(rsd_mov_avg$no_of_reviews,rsd_mov_avg$rsd_mov_avg)
```

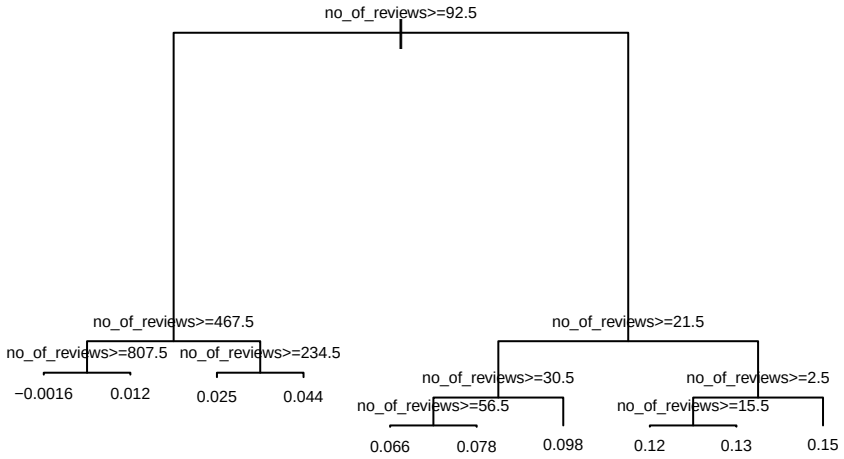
```
## [1] -0.17815
```

Analysis shows a moderately negative relationship between the two variables. Nonetheless, it is not appropriate to use general regression to predict the ratings because the distribution of no. of reviews per movie, as shown earlier in the movie effect section, is far from normal. Therefore, we choose regression tree to predict the residual ratings by partitioning the movies according to their review count, tuning the complexity parameter through cross-validation along the process.

The result is a regression tree with 10 nodes where the residual average rating gradually decreases as the review count increases:

```
rgtr <- train(rsd_mov_avg~no_of_reviews,
              data = rsd_mov_avg,
              method = "rpart",
              tuneGrid = data.frame(cp = seq(0, 0.01, 0.0001)),
              trControl = trainControl(method = "cv", p = 0.9))

plot(rgtr$finalModel, margin=0.1)
text(rgtr$finalModel, cex=0.5)
```



The following table shows the residual rating estimates as per range of review count.

```
filter(rgtr$finalModel$frame, var=="<leaf>") %>%
  select(yval) %>% rename(rsd_estimate=yval) %>%
  cbind("reviews fr" = c(sort(rgtr$finalModel$splits[,4]+0.5, decreasing = T),1),
        "reviews to" = c(max(rsd_mov_avg$no_of_reviews),
                         sort(rgtr$finalModel$splits[,4]-0.5, decreasing = T)))
```

```
##    rsd_estimate reviews fr reviews to
## 1    -0.0016454        808      31362
## 2     0.0121385        468        807
## 3     0.0251567        235        467
## 4     0.0441534         93        234
## 5     0.0656710         57         92
## 6     0.0778165         31         56
## 7     0.0975844         22         30
## 8     0.1153192         16         21
## 9     0.1301997          3         15
## 10    0.1541686          1          2
```

We can plug back the residual rating estimates into the rsd_mov_avg table using the predict function of the rgtr model.

```
rsd_mov_avg <- rsd_mov_avg %>%
  select(movieId, rsd_mov_avg, no_of_reviews) %>%
  cbind(rsd_avg = predict(rgtr))
```

So now we have for each movie a residual rating estimate based on the number of reviews it receives. With this final piece of puzzle, we compute the RMSE for the test set one last time.

```
RMSE <- final_holdout_test %>%
  select(movieId, userId, rating) %>%
  inner_join(user_avg, "userId") %>%
  inner_join(mov_gen, "movieId") %>%
  left_join(user_gen, c("userId", "genre")) %>%
  group_by(movieId, userId, rating, user_avg) %>%
  summarize(gen_avg = mean(gen_avg, na.rm = T)) %>%
  inner_join(mov_avg, "movieId") %>%
  inner_join(rsd_mov_avg, "movieId") %>%
  mutate(SE = (rating - mu - mov_avg
               - ifelse(is.na(gen_avg),user_avg,gen_avg)
               - rsd_avg)^2) %>%
  pull(SE) %>% mean() %>% sqrt() %>% print
```

```
## [1] 0.84982
```

This gives a very slight but hard-earned improvement. We hence take this as our final algorithm for the movie rating recommendation task.

Conclusion

All in all, we have built a movie recommendation system which has successfully reduced the RMSE in the evaluation test set from its nRMSE baseline of 1.0612 to 0.84982, an almost 20% improvement and well surpassing the 0.86490 benchmark required of the grading rubric.

Of these, the movie average has the most significant effect (11.05%), whereas user/genre effect combined further reduces the RMSE by another 9.95% compared to just 8.32% with user effect alone (measured from the reduced RMSE). And the review count has almost negligible effect, reducing it by a mere 0.019% (measured from the reduced RMSE).

However, it may not be fair to conclude the three factors actually do have such varying degrees of influence. Keep in mind as the effect of each factor is subtracted from the ratings before investigating the next factor (hence the ratings become increasingly residual), the order in which the factors are considered matters. In fact, it is speculated that a different RMSE would be produced should this order be different. Future work can explore this issue.

Another interesting point to raise is that despite attempts to introduce the penalty terms during both movie and user average rating estimations, parameter tuning results suggested the opposite. One speculative reason for this is that the record sets used to determine the weighted means are very large compared to the validation sets (9:1), so there are not many cases where a few ratings have to be used to estimate the ratings of many other reviews of the same movie/user. As such, any increase in the penalty term will only serve to increase deviation from the otherwise pretty accurate estimates.

As for the review count, I want to spend a bit effort to decipher what such an intriguing pattern implies - why do the residual movie ratings decrease as the review counts increase? Although popular movies tend to have higher average ratings than non-popular ones and vice versa, any such effect has already been nullified when we first subtract the movie means from the original ratings. At that point the residual averages for each movie are close to zero.

However, when the user/genre averages are also subtracted from those residual ratings, it does not produce a uniform effect! In other words, a popular movie with high acclaim will cause some dissenters to rate it

much lower than they normally would ("I don't know why so many people like this movie - this is crxx!"). So these normally non-critical raters do not have much to add back to the residual ratings (due to their not-so-negative rating history) while the above average ratings by the normally above average raters are subtracted, rendering a slightly negative residual movie average for the popular movies after adjusting for user/genre effect.

The contrary holds true for the movies with few reviews. Some mavericks seem to rate them much higher than they normally would ("This movie is indeed very good; other people just don't get it!"), so after adjusting for the user/genre effect the residual averages are moderately positive. It might indicate there are some "niche movies" for which only certain die-hard fans in particular would appreciate - an interesting phenomenon to observe.

As a matter of fact, I had tried a few other algorithms, including factor analyzing the genres, parsing the timestamps to look for significant difference in the ra tings across months/weekdays, only to no avail after many gruelsome hours. So the tiny improvement made by the movie review count is already a huge achievement in that regard.