Capstone file

Namesh Nazar

12/23/2020

Introduction/overview/executive summary

This is an analysis to develop a movie recommendation model.

Dataset

Movielens dataset is used to conduct the analysis and predict ratings for different movies. The dataset includes a unique ID for both movies and users, it includes the name of the movie, the rating it received by a user, and the timestamp of the rating. It also includes the genres that apply to the movies.

Goal of the project

We use this data to predict the rating that a user is likely to provide to a movie and therefore recommend movies to users that they are likely to rate highly.

Steps that were performed

We use the data set to develop models that will predict expected ratings given by users for movies. The models are optimized to minimized the Root Mean Squared Errors (RMSE) between predicted ratings and actual ratings.

1. Movielens dataset is split into edx and validation data so that models can be developed on edx data and final performance can be evaluated on validation data.
2. Edx data is explored through visualizations to better understand the data.
3. Edx dataset is split into training and test set to optimise the model
4. Features are added to the model one by one to evaluate improvement in RMSE and then the models are regularized.
5. The final model is applied to the validation dataset to evaluate the model performance.

We have limited data with few users rating few movies depending on quality of movie, its age, its genre etc. We therefore use bias associated with users, movies, how old the movie is, and its genre to predict the rating of particular movie for different users. The assumption is that similar movies receive similar ratings from similar people in the same amount of time.

Method/Analysis

Exploratoring the data:

We first explore the structure of the dataset we are using the design the model. Since the edx is a random subset of the movielens data and the part we will actually be designing our model on I am only using this, however we could have used the whole dataset as well.

We see that we have close to 9 million rows and 6 columns - two integers, two numerical, and two characters. From the first few rows of the dataset, we can see that the year of movie release, while a useful variable, is a part of the movie title and not a separate column. This form of initial exploration of data is often useful to better understand the variables.

```
#See dimentions of dataset
dim(edx)
```

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

```
#See structure of dataset
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
#See first few rows of dataset
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
```

Transforming the data

We anticipate that the years since movie release would be a useful variable for our model. We therefore transform the dataset to add the year of release ("year") and the years since release ("old") as separate columns. We use head() function to see the result to make sure that the code worked as expected.

```
# Add two columns: year= year of movie release and old= In 2020, how many years has it been since movie

edx<- edx %>% mutate(year=str_sub(title,-5,-2), old=2020-as.numeric(year))
head(edx)
```

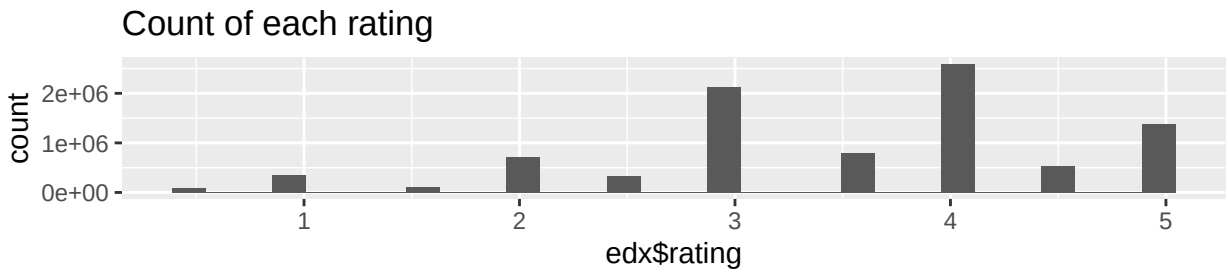
```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres year old
## 1:                Comedy|Romance 1992  28
## 2:          Action|Crime|Thriller 1995  25
## 3:  Action|Drama|Sci-Fi|Thriller 1995  25
## 4:       Action|Adventure|Sci-Fi 1994  26
## 5: Action|Adventure|Drama|Sci-Fi 1994  26
## 6:       Children|Comedy|Fantasy 1994  26
```

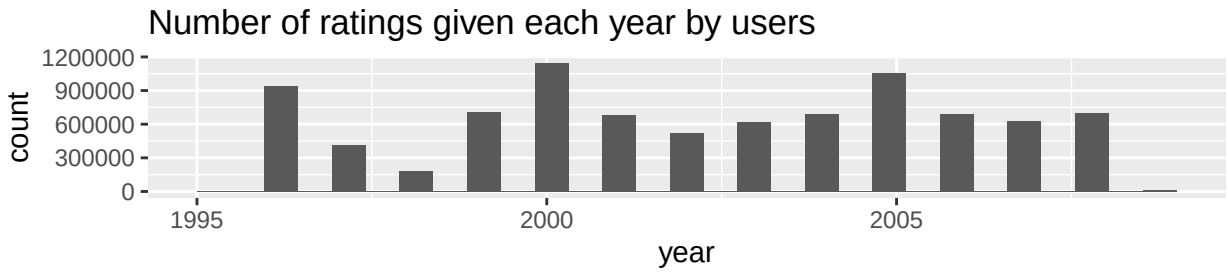
Visualizing the data

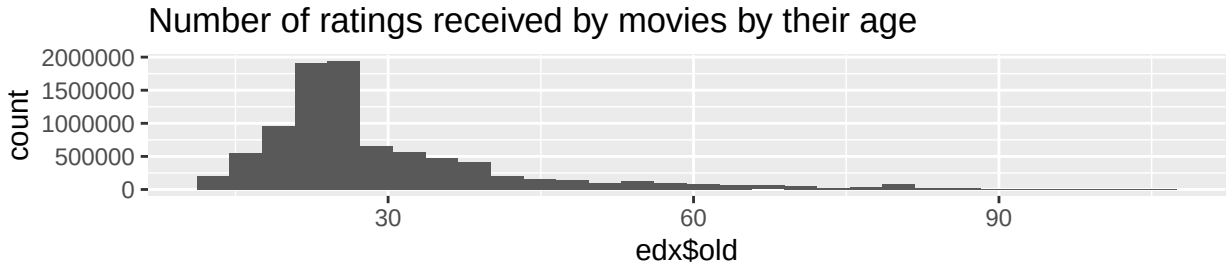
We then visualize the data to better understand it. Following are some major findings from these visualizations:

1. We can see 4 is most frequently given rating and very few ratings of 0 or 1 are given.
2. Over time ratings have increased indicating more activity by users.
3. We observe that old movie and very new movies have fewer ratings than others indicating lack of interest to rate very old movies and insufficient time to rate very new movies.

```
# Histogram of count of different ratings.
a<-ggplot(data=edx, aes(edx$rating))+geom_histogram()+ggtitle("Count of each rating")

# Histogram of count of ratings given each year using the timestamp variable. the time stamp is used to
b<-edx %>% mutate(date= as_datetime(timestamp),
                  year=as.numeric(format(date,'%Y'))) %>% ggplot(aes(year))+geom_histogram()+ggtitle("Numb

# Histogram of the year of age of movies by number of ratings
c<-ggplot(data=edx, aes(edx$old))+geom_histogram()+ggtitle("Number of ratings received by movies by the

# Arranging all graphs in one grid
grid.arrange(a,b,c, nrow=3)
```







Splitting the data

Edx dataset is used for optimizing the model and the performance of the final optimized model will be evaluated using validation dataset. For the model optimization we split edx dataset into training and test sets.

```
# Set seed before creating partition to get consistent result each time code is evaluated
set.seed(755)

# create partition and save index number of subsets in test_index
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)

#Use test_index to split edx into training and test set
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

#To make sure we don't include users and movies in the test set that do not appear in the training set,
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

We then have a look at the two datasets to make sure everything worked.

```
# Have a look at the two dataset to ensure all the variable names, columns, data are as expected.
head(test_set)
```

```
##    userId movieId rating timestamp                               title
## 1:      1     362      5 838984885            Jungle Book, The (1994)
## 2:      1     420      5 838983834       Beverly Hills Cop III (1994)
## 3:      1     466      5 838984679        Hot Shots! Part Deux (1993)
## 4:      2     733      3 868244562                    Rock, The (1996)
## 5:      2     736      3 868244698                      Twister (1996)
## 6:      2    1049      3 868245920 Ghost and the Darkness, The (1996)
##                               genres year old
## 1:        Adventure|Children|Romance 1994  26
## 2:      Action|Comedy|Crime|Thriller 1994  26
## 3:                 Action|Comedy|War 1993  27
## 4:         Action|Adventure|Thriller 1996  24
## 5: Action|Adventure|Romance|Thriller 1996  24
## 6:                  Action|Adventure 1996  24
```

```
head(train_set)
```

```
##    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 year old
## 1:                Comedy|Romance 1992  28
## 2:         Action|Crime|Thriller 1995  25
## 3:  Action|Drama|Sci-Fi|Thriller 1995  25
## 4:       Action|Adventure|Sci-Fi 1994  26
## 5: Action|Adventure|Drama|Sci-Fi 1994  26
## 6:       Children|Comedy|Fantasy 1994  26
```

Modeling approach

After splitting the edx dataset into training and testing dataset we start trying different features in the dataset to make predictions.

1. Our first model will use the average rating in the dataset as the prediction for all missing ratings. By using the average we calculate the RMSE and use this as the baseline. For any model to be worth exploring it needs to perform better than this baseline.
2. We first use movie effects to predict ratings. We assume that movies get similar ratings from different users- blockbuster hits are more likely to get higher ratings than flop movies.
3. We then add user effect to the model since we anticipate similar users give similar ratings to movies- more picky users give lower ratings even to blockbuster hits than other users.
4. We then add the age of the movie since our visualization above shows that movies of similar age have similar count of ratings.
5. We then add the genre of the movies to test if similar kind of movies get similar kind of ratings.
6. We regularize all models to remove any error being added due to few datapoints to make predictions - for instance very old and very new movies that have very few ratings which may skew their results.

7. We identify the best performing model
8. We then evaluate the performance of the best model using validation data.

Results

Step 1: Baseline using average rating

We set the baseline of performance by using the average rating as a predictor of ratings. For any model to be valuable to us it must perform better than this baseline. According to the following output the baseline is a rating prediction off by one star.

```
# Find the mean rating in the training set

mu_hat <- mean(train_set$rating)

#Apply the mean rating as predicted value for all ratings and compare it to actual ratings in test set
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060561
```

Step 2: Movie Effects

We start improving on this prediction by adding features to the model to see if they add more predictive power to the model. We first test for movie effects assuming that same movie probably get similar ratings. We find that in fact the prediction has definitely improved with about 0.94 RMSE:

```
# Find the mean rating in the training set
mu <- mean(train_set$rating)

# Compute least squared estimate using mean of difference of actual rating and average rating
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Calculate Predicted value using least squared estimate of movie effect
predicted_ratings_movie <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

# Calculate RMSE
movie_effect<-RMSE(predicted_ratings_movie, test_set$rating)

movie_effect
```

```
## [1] 0.9439868
```

Step 3: User Effects

While this is better than the baseline, we can further improve on this by combining movie and user effects to see how similar people rate similar movies. The RMSE has indeed improved significantly.

```r
# Find the mean rating in the training set
mu <- mean(train_set$rating)

# Compute least squared estimate using mean of difference of actual rating, average rating and estimate
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Calculate Predicted value using least squared estimate of user and movie effects
  predicted_ratings_usermovie <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

  # Calculate RMSE
movieuser_effect<-RMSE(predicted_ratings_usermovie, test_set$rating)

movieuser_effect
```

```
## [1] 0.8666408
```

Step 4: Time Effects

We further try to see if the time since the movie release can add any predictive power to the previous model. Based on our earlier visualization we know that age of movie is correlated with the count of ratings it receives. Based on the model below we see that while it has improved the model, the incremental improvement is less than before:

```r
# Find the mean rating in the training set
mu <- mean(train_set$rating)

# Compute least squared estimate using mean of difference of actual rating, average rating, estimated m
year_avgs <- train_set %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(old) %>%
  summarize(b_t = mean(rating - mu- b_i - b_u))

# Calculate Predicted value using least squared estimate of user, movie, and time effects
predicted_ratings_umt<-test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='old')%>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  pull(pred)

# Calculate RMSE
movieusertime_effect<-RMSE(predicted_ratings_umt, test_set$rating)

movieusertime_effect
```

```
## [1] 0.8662943
```

Step 5: Genre Effects

We build on the previous model by adding another feature to see if we can get more improvement in the model. This time we add genre to see if similar movies tend to get similar ratings. We do find a marginal improvement in the RMSE but not a lot.

```
# Find the mean rating in the training set
mu <- mean(train_set$rating)

# Compute least squared estimate using mean of difference of actual rating, average rating, estimated m
genre_avgs <- train_set %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(year_avgs, by='old') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu- b_i - b_u - b_t))

# Calculate Predicted value using least squared estimate of user, movie, time and genre effects
predicted_ratings_umtg<-test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(year_avgs, by='old')%>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  pull(pred)

# Calculate RMSE
movieusertimegenre_effect<-RMSE(predicted_ratings_umtg, test_set$rating)

movieusertimegenre_effect
```

```
## [1] 0.8660414
```

Step 6: Regularize data

So far we have only used the variables directly in the model. However, we know that some movies get a lot of ratings while others don't. This is indicated by the visualization earlier that shows that very old and very new movies get lesser ratings than other movies. This could result in noisy estimates that are skewed due to difference in the available data to make predictions for different movies. We therefore use regularization to adjust for this problem. Regularization permits us to penalize large estimates that are formed using small sample sizes.

First we regularlize the best performing model- including movie, user, time and genre effects- to see if we can get the required RMSE. We will also regularize other models just to compare but we anticipate the best performing model without regularlization will perform the best with regularlization as well:

```
# Set a range of lamba values to find the optimal penalty terms
lambdas <- seq(0, 10, 1)

# Function to find RMSE values using different lamda values
```

```
rmses <- sapply(lambdas, function(l){

  # Find the mean rating in the training set
  mu <- mean(train_set$rating)

  # Compute least squared estimate of estimated movie, user, time, genre effects
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

   b_t <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
     group_by(old) %>%
    summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+l))

    b_g <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_t, by="old") %>%
     group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - b_t - mu)/(n()+l))

    # Calculate Predicted value using least squared estimate of user, movie, time and genre effects
  predicted_ratings_regmutg <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "old") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
    pull(pred)

  # Calculate RMSE
    return(RMSE(predicted_ratings_regmutg, test_set$rating))
})

# Find optimal lamda
minlambda <- lambdas[which.min(rmses)]

# Find min RMSE
regmovusertimegenre<-min(rmses)

regmovusertimegenre
```

```
## [1] 0.8654462
```

For comparison below are the results of regularization of other models:

- Movie effect: As expect, the RMSE is better then the RMSE of movies model without regularlization

```
# Set a range of lamba values to find the optimal penalty terms
lambdas <- seq(0, 10, 0.25)

 # Find the mean rating in the training set
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

  # Function to find RMSE values using different lamda values
rmses <- sapply(lambdas, function(l){
  predicted_ratings_regmov <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

    # Calculate RMSE
  return(RMSE(predicted_ratings_regmov, test_set$rating))
})

# Find optimal lamda
lambda <- lambdas[which.min(rmses)]

# Find min RMSE
regmov<- min(rmses)

regmov
```

```
## [1] 0.9439217
```

- Movie and user effect: We then try regularization of the model including both movie and user effects and the results are not only better than user+movie effect without regularization but also better than the model using user, movie, time and genre effect without regulation.

```
# Set a range of lamba values to find the optimal penalty terms
lambdas <- seq(0, 10, 0.25)

# Function to find RMSE values using different lamda values
rmses <- sapply(lambdas, function(l){

 # Find the mean rating in the training set
  mu <- mean(train_set$rating)

  # Compute least squared estimate of estimated movie and user effects
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
```

```
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  # Calculate Predicted value using least squared estimate of user and movie effects
  predicted_ratings_regmovuser <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

   # Calculate RMSE
    return(RMSE(predicted_ratings_regmovuser, test_set$rating))
})
# Find optimal lamda
lambda <- lambdas[which.min(rmses)]

# Find min RMSE
regmovuser<-min(rmses)

regmovuser
```

```
## [1] 0.8659626
```

- Movie, user, time effect: We then try adding time to the regularized model to test for improvement in RMSE and we do find a marginal improvement. The increase in the number of variables the model is evaluating is starting to slow down the code. We therefore reduced the lamdas that the model would have to evaluate to reduce computation time.

```
# Set a range of lamba values to find the optimal penalty terms
lambdas <- seq(0, 10, 1)

# Function to find RMSE values using different lamda values
rmses <- sapply(lambdas, function(l){

  # Find the mean rating in the training set
  mu <- mean(train_set$rating)

  # Compute least squared estimate of estimated movie, user, and time effects
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

   b_t <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
     group_by(old) %>%
    summarize(b_t = sum(rating - b_i - b_u - mu)/(n()+l))
```

```
   # Calculate Predicted value using least squared estimate of user, movie and time effects
  predicted_ratings_regmut <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "old") %>%
    mutate(pred = mu + b_i + b_u + b_t) %>%
    pull(pred)

  # Calculate RMSE
    return(RMSE(predicted_ratings_regmut, test_set$rating))
})

# Find optimal lamda
lambda <- lambdas[which.min(rmses)]

# Find min RMSE
regmovusertime<-min(rmses)

regmovusertime
```

```
## [1] 0.8656733
```

Step 7: Identify best performing model

Following is the RMSE using the test set in edX. We can see that the regularized model using movie, user, time and genre effect performs the best compared to others.

```
# Make a table summarizing all results
rmse_results <- tibble(method = c("Just the average","Movie effect","User+ Movie effect","User+Movie+Ti
rmse_results
```

```
## # A tibble: 9 x 2
##   method                                          RMSE
##   <chr>                                          <dbl>
## 1 Just the average                               1.06
## 2 Movie effect                                   0.944
## 3 User+ Movie effect                             0.867
## 4 User+Movie+Time effect                         0.866
## 5 User+Movie+Time+genre effect                   0.866
## 6 Regularized Movie Effect Model                 0.944
## 7 Regularized Movie + User Effect Model          0.866
## 8 Regularized Movie + User + time Effect Model   0.866
## 9 Regularized Movie + User +time + genre Effect Model 0.865
```

Step 8: Evaluate performance of model on validation data

We now use the best performing model for the final evaluation using validation data set.

```r
# Find the mean rating in the edx set
mu_final <- mean(edx$rating)

# Compute least squared estimate of estimated movie, user, time, genre effects. The minlambda from best
 b_i_final <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_final = sum(rating - mu_final)/(n()+minlambda))

  b_u_final <- edx %>%
    left_join(b_i_final, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_final = sum(rating - b_i_final - mu_final)/(n()+minlambda))

   b_t_final <- edx %>%
    left_join(b_i_final, by="movieId") %>%
    left_join(b_u_final, by="userId") %>%
     group_by(old) %>%
    summarize(b_t_final = sum(rating - b_i_final - b_u_final - mu_final)/(n()+minlambda))

    b_g_final <- edx %>%
    left_join(b_i_final, by="movieId") %>%
    left_join(b_u_final, by="userId") %>%
    left_join(b_t_final, by="old") %>%
     group_by(genres) %>%
    summarize(b_g_final = sum(rating - b_i_final - b_u_final - b_t_final - mu_final)/(n()+minlambda))

# Add columns of 'year' and 'old' in validation dataset
validation<- validation %>% mutate(year=str_sub(title,-5,-2), old=2020-as.numeric(year))

# Calculate Predicted value using least squared estimate of user, movie and time effects
predicted_ratings_val <-
    validation %>%
    left_join(b_i_final, by = "movieId") %>%
    left_join(b_u_final, by = "userId") %>%
    left_join(b_t_final, by = "old") %>%
    left_join(b_g_final, by = "genres") %>%
    mutate(pred = mu_final + b_i_final + b_u_final + b_t_final + b_g_final) %>%
    pull(pred)

# Calculate RMSE
    RMSE(predicted_ratings_val, validation$rating)
```

```
## [1] 0.8642529
```

Conclusion

In summary, we find that the movie recommendations can be improved by using different features while controlling for any bias introduced by few observations resulting in big predictions. However, it is important to evaluate the relative predictive power of different variables. In the above model the incremental improvement using time and genre was less than user and movie IDs. We may be able to improve the predictive power by genre by separating out all the genre tags that are applied to each movie so that there is a bigger pool of data per genre. Similarly, for evaluation of models we can use other evaluation metrics (like specificity,

accuracy, recall etc) for the model to test performance. We can also treat ratings as categorical variables and use more advance models like KNN, random forrest etc however they require higher computation power.