Movie Recommendation System

Brandon Rufino

05/12/2019

SECTION 1: Overview

Recommender systems are some of the most important machine learning applications to date. Specifically, recommender systems are relied on heavily by companies to cater great consumer experiences. This can be seen through streaming services such as Netflix as well as retail companies such as Nike. Thus, in this project I will create a movie recommender system that will predict movie ratings for a user.

This document will go through several machine learning stages. First, we will state the problem presented by the movielens dataset. We will extrapolate the data from the movielens database into a friendly format which contains information such as users, movies, genres, and their ratings. Second, we will explore the data-set and look at interesting trends. Once data exploration is finished, we will preprocess our data to extrapolate features that we may require for model creation. With our features in hand, we will create several models and predict movie ratings. Lastly, we will choose the best model to run against unseen data and report the final RMSE value.

1.1 Problem statement

For this project we are creating a movie recommendation system using MovieLens dataset. This dataset is a small subset of a much larger dataset with millions of ratings. We are given code to create two datasets: (1) edx and (2) validation. The edx dataset is the dataset in which we will train and test our models. The validation dataset is our 'unseen' data and will be used to report our final RMSE value. The goal of this project is to achieve an RMSE value less than or equal to 0.8649.

1.2 SETUP: Load edx set, validation set

In this step we set up our data. We load our edx data set which we will use to train and tune our parameters for our model selection. We will use the validation dataset to report our final RMSE value.

1.3 QUIZ: Data exploration

Let us explore some properties of our edx dataset. This is so we can better understand our data at hand (and also to ace the quiz).

Let us take a look at the dimensions of edx:

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

Wow! 9000055 entries in our edx dataset alone. As data scientist we should be thrilled to see how many observations we are able to work with.

Let us see how many '0' and '3' we were given as ratings:

```
## [1] "number of 0 ratings:"
```

```
## [1] 0
```

```
## [1] "number of 3 ratings:"
```

```
## [1] 2121240
```

Interesting that we have 0 '0' ratings (pun intended).

Let us see how many unique movies and users we have in the dataset.

```
## [1] "number of unique movies:"
```

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

```
## [1] "number of unique users:"
```

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

Almost 70,000 people!

One may wonder how many ratings there are for genres. Let us take a look:

```
## [1] "drama ratings:"
```

```
## [1] 3910127
```

```
## [1] "comedy ratings:"
```

```
## [1] 3540930
```

```
## [1] "thriller ratings:"
```

```
## [1] 2325899
```

```
## [1] "romance ratings:"
```

```
## [1] 1712100
```

Now let us see which movie has the greatest number of ratings:

```
edx %>% group_by(movieId,title) %>% summarize(count = n()) %>%
    arrange(desc(count))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                  count
##      <dbl> <chr>                                                  <int>
##  1     296 Pulp Fiction (1994)                                    31362
##  2     356 Forrest Gump (1994)                                    31079
##  3     593 Silence of the Lambs, The (1991)                       30382
##  4     480 Jurassic Park (1993)                                   29360
##  5     318 Shawshank Redemption, The (1994)                       28015
##  6     110 Braveheart (1995)                                      26212
##  7     457 Fugitive, The (1993)                                   25998
##  8     589 Terminator 2: Judgment Day (1991)                      25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                       24284
## # ... with 10,667 more rows
```

5 best rated movies? Let us find out:

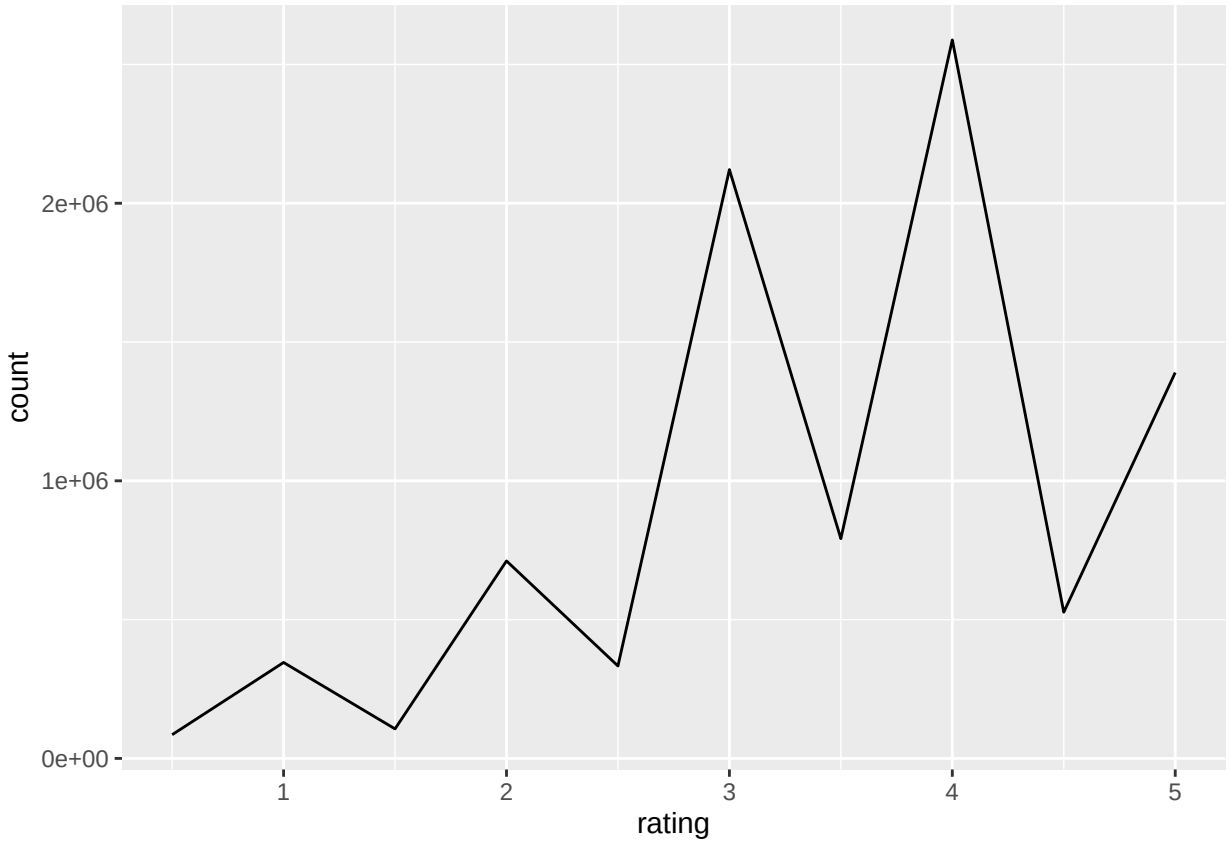
```
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5) %>%
    arrange(desc(count))
```

```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

Fun fact: half star ratings are less common then full-star ratings. Take a look:

```
edx %>%
    group_by(rating) %>%
    summarize(count = n()) %>%
    ggplot(aes(x = rating, y = count)) +
    geom_line()
```



SECTION 2: Methods

In this section we will briefly explore data outside of the mandatory quiz questions. We will then preprocess our data to extrapolate features that we may use in our model training. Lastly, we will describe the different models we will train and test.

2.1: Data exploration continued (outside of QUIZ questions)

This for the most part was done through the given code by the course instructor and mandatory quiz questions that we saw in section 1.2/1.3. Recall we have built a dataset called 'edx' with 6 properties: (1) userId, (2) movieId, (3) rating, (4) timestamp, (5) title, and (6) genres.

Recall, genres such as drama and comedy were frequently rated (refer to the output below). We could parse all genres and count their ratings however, due to long run times given the current data set this was avoided. That said, the appendix contains code on how one would see a list of genres with their rating count.

```
## [1] "drama ratings:"
```

```
## [1] 3910127
```

```
## [1] "comedy ratings:"
```

```
## [1] 3540930
```

```
## [1] "thriller ratings:"
```

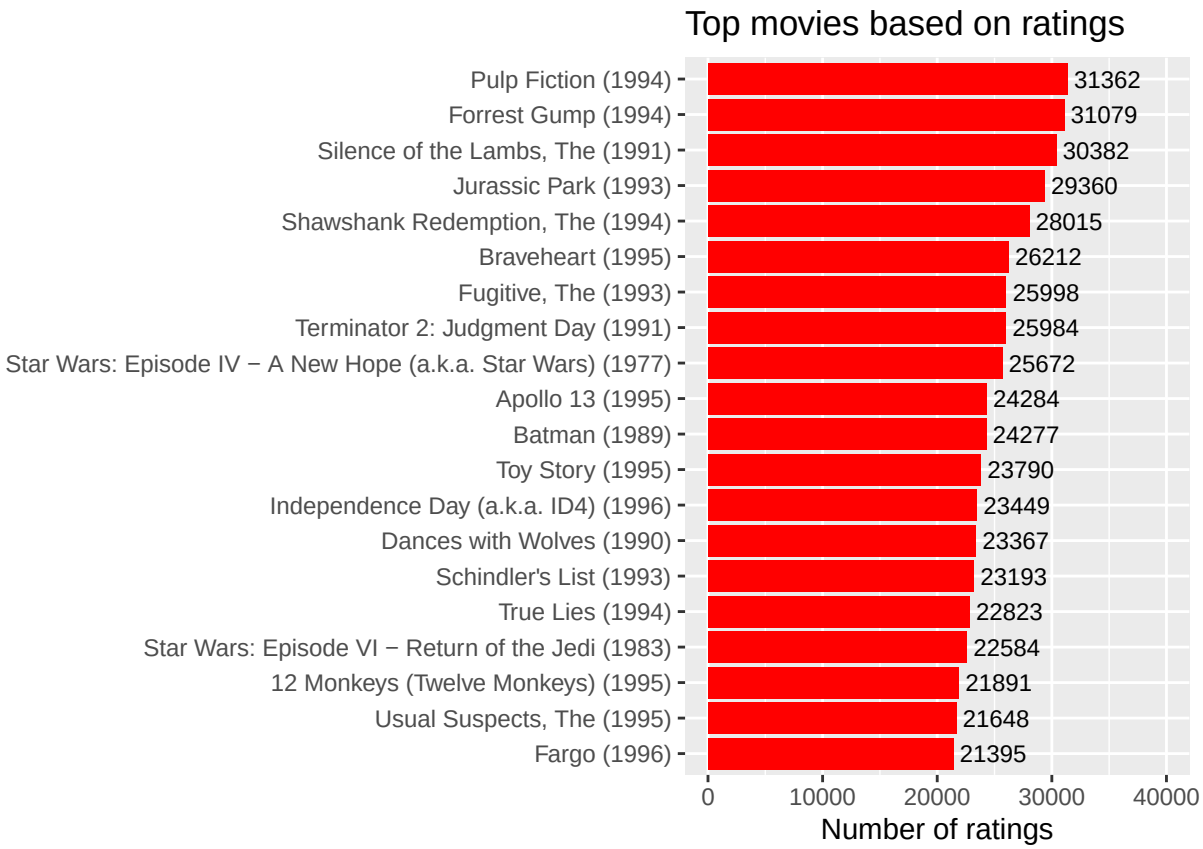
```
## [1] 2325899
```

```
## [1] "romance ratings:"
```

```
## [1] 1712100
```

Lets take a better look at the movies with the top number of ratings via a bar plot:

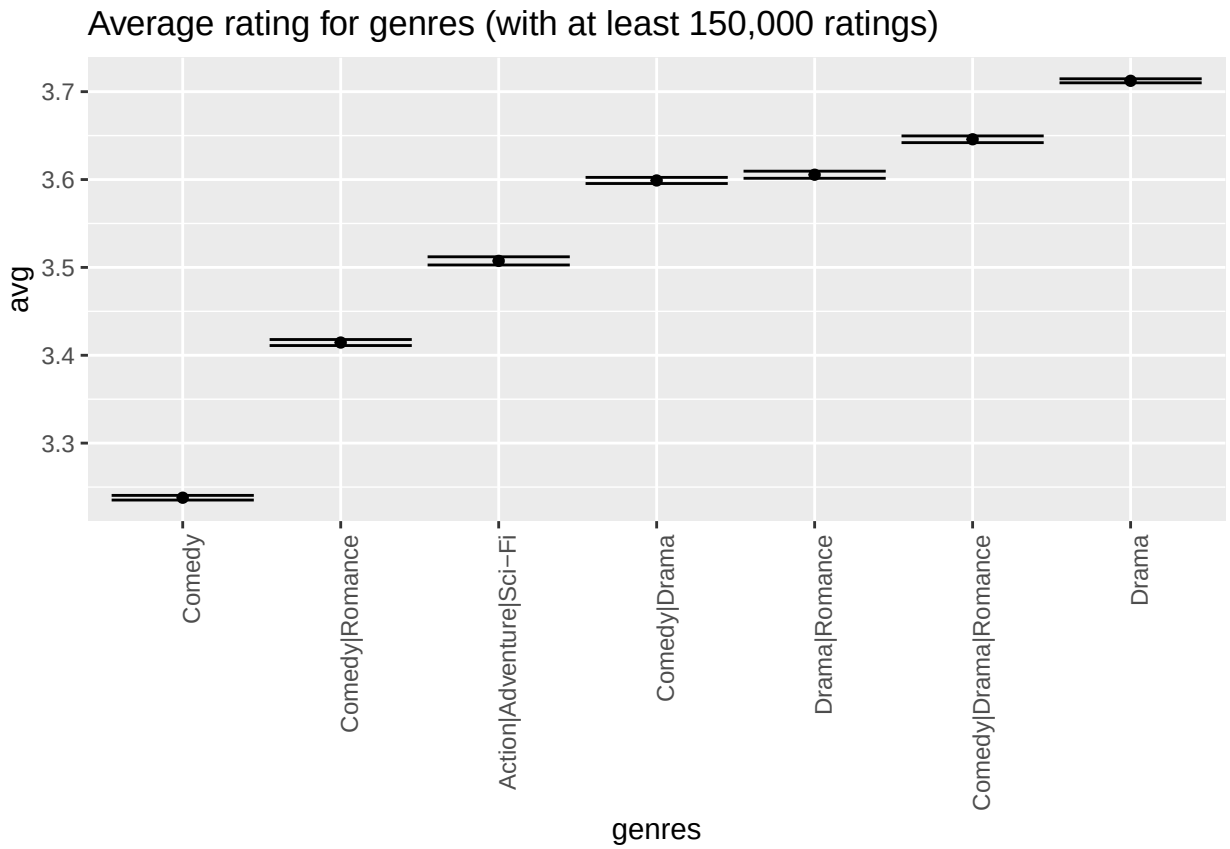
```
# group the top 20 most rated movies
top_titles<-edx %>% group_by(title) %>% summarize(count = n()) %>% top_n(20,count)%>%
    arrange(desc(count))

# plot the top rated titles with the amount of times they were rated
top_titles %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', fill="red") + coord_flip(y=c(0, 40000)) +
  labs(x="", y="Number of ratings") +
  geom_text(aes(label= count), hjust=-0.1, size=3) +
  labs(title="Top movies based on ratings")
```



Alright so we see that the top movies based on number of ratings agree with the genres with the most ratings. For example, Pulp Fiction has the most ratings and it belongs to the Drama category which had the highest number of ratings for genres.

More interesting, let us examine the average rating per genre. To prevent displaying to many genres we will set a minimum requirement of at least 150,000 ratings for a particular genre:

```
#Group genres and count how often they were rated and what the average and sd are
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 150000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "Average rating for genres (with at least 150,000 ratings)")
```



Seems like Comedy is on average one of the lower rating genres. Also note, 'Drama' has a high average rating (which is interesting seeing that it also had the most ratings).

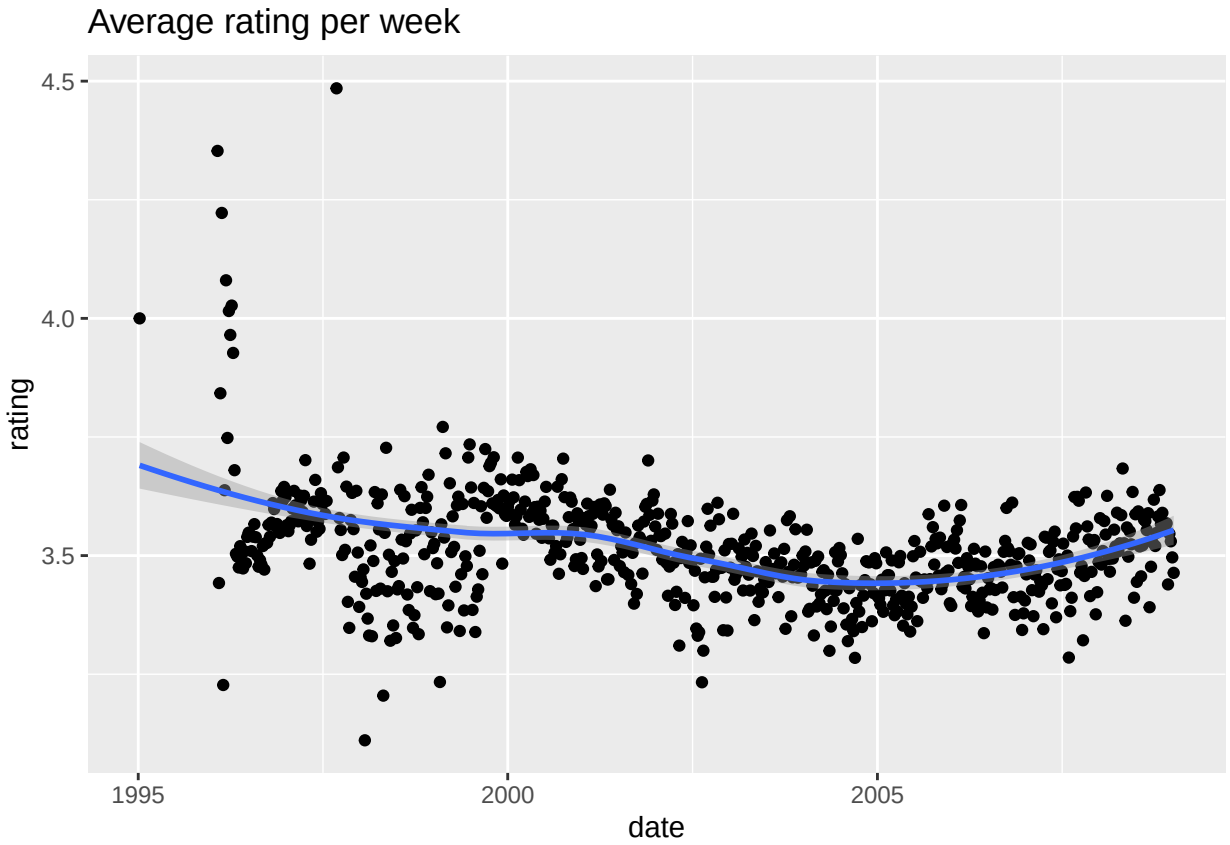
Notice that the edx dataset includes a timestamp. This variable represents the time and data in which the rating was provided. The units are seconds since January 1, 1970. Lets create an additional column called date:

```
# adding a column which converts timestamp to a year
edx_plus <- edx %>% mutate(date = round_date(as_datetime(timestamp), unit = "year"))
```

With an additional data column let us take the average rating per week:

```
# convert timestamp to the week and find the average rating per week
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
    group_by(date) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(date, rating)) +
    geom_point() +
    geom_smooth()+
  ggtitle("Average rating per week")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

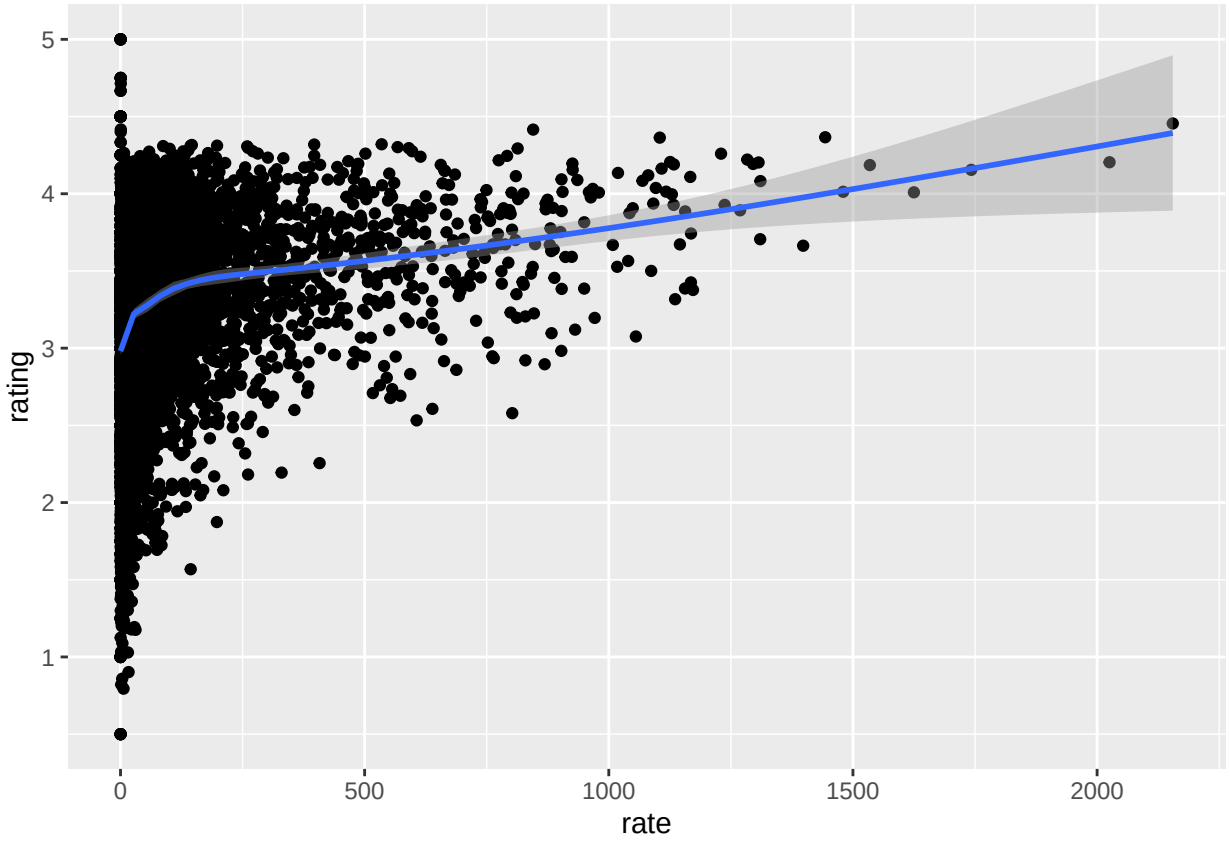


The plot above does not show a strong relationship between average rating and the year. This makes sense because even though movies have increased budgets, peoples expectations of movie quality also increases. Thus, the average ratings generally do not change much over time.

Some may wonder if a movie is being rated more frequently will it average a higher rating? Let us take a look:

```
# group movies, find the rate in which it is rated, find average rating
edx_plus  %>%
    group_by(movieId) %>%
    summarize(n = n(), years = 2018-first(year(date)),
              title = title[1],
              rating = mean(rating)) %>%
    mutate(rate = n/years) %>%
    ggplot(aes(rate, rating)) +
    geom_point() +
    geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Interesting enough, the movies with a higher rate of being rated tend to have higher ratings then movies that are not frequently rated.

2.2: Data preprocessing

Up to this point we have: (1) obtained our movie rating data, and (2) explored interesting findings in relation to the data. That said, the next step to our machine learning workflow is to preprocess our data in order to create meaningful entries for our machine learning models to work with.

Take for example audio data. We would perhaps preprocess audio data by extracting frequency and signal entropy characteristics. In doing so, we can hope that frequency and entropy characteristics between different audio sources differ enough such that a machine learning algorithm can distinguish this.

For a movie recommendation system we have several features we may wish to use. Features such as: ratings, genres, user average rating, and time are all ratings we should consider. Lucky for us, our data has already been processed in a manner which makes these features easy to access. However, we should note our genres can be improved. That is, we could use one-hot encoding to represent which genre each movie belongs to (i.e. replace the one column of genres with the X number of unique genres; placing a zero if the movie does not belong and a one if it does). The code to accomplish a one-hot encoding for genre was attempted. However, due to the long run time with the dataset we will try to create models with how the current data is situated. Refer to the appendix for the code used to attempt one-hot genre encoding (this idea of one-hot encoding our genres would make for a great next step).

Lastly, we should also explore the similarity in movie ratings and users. That is, are there users who have strong correlation with eachother? Are there movies in which were rated very similarily? As we have hinted through other approaches our dataset is very large making some conventional commands hard to perform. Let us create a sparse matrix of our ratings to allow for more computationally efficient commands (this code is credited to the blog posted by Louis Mono):

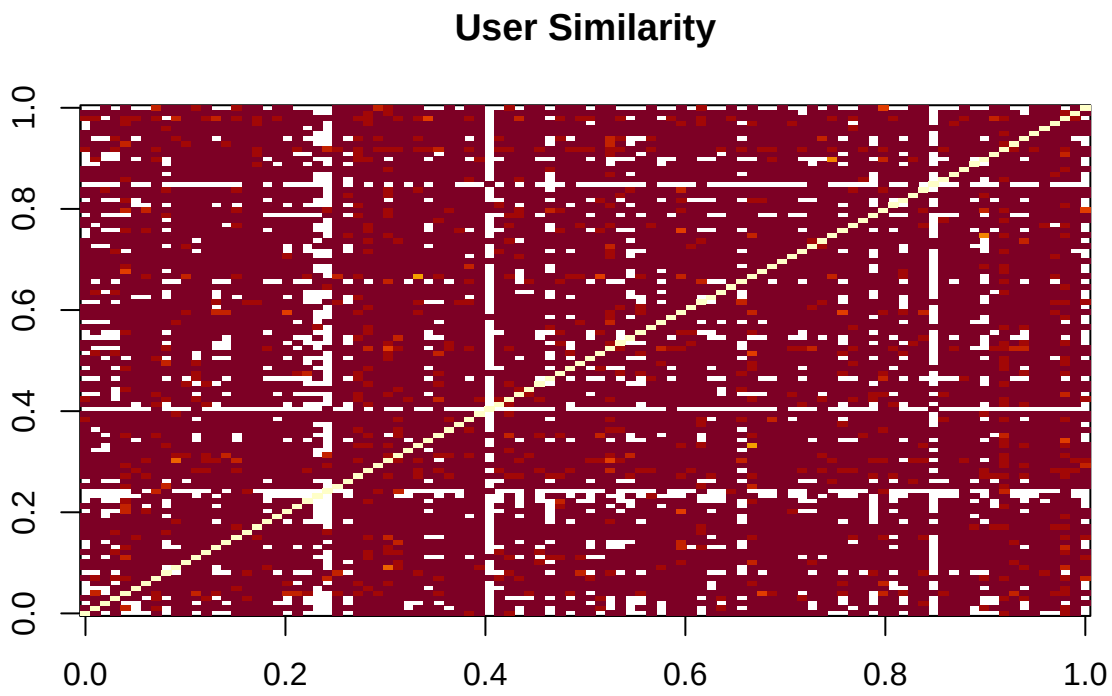
```r
#create a copy of edx
edx.copy <- edx

#sparsematrix function requires movieId and userId to be type factor -> type numeric
edx.copy$userId <- as.factor(edx.copy$userId)
edx.copy$movieId <- as.factor(edx.copy$movieId)

#sparsematrix function requires movieId and userId to be type numeric

edx.copy$userId <- as.numeric(edx.copy$userId)
edx.copy$movieId <- as.numeric(edx.copy$movieId)

#i,j specifying the location of the non-zero entries of the matrix
# x optional value of matrix entries
sparse_ratings <- sparseMatrix(i = edx.copy$userId,
                         j = edx.copy$movieId ,
                         x = edx.copy$rating,
                         dims = c(length(unique(edx.copy$userId)), #dimension of only unique values
                                  length(unique(edx.copy$movieId))),
                         dimnames = list(paste("user", 1:length(unique(edx.copy$userId)), sep = ""),
                                         paste("movie", 1:length(unique(edx.copy$movieId)), sep = "")))


# remove the copy created
rm(edx.copy)

#let us see how the sparse_ratings look
sparse_ratings[1:10,1:10]
```

```
## 10 x 10 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 10 column names 'movie1', 'movie2', 'movie3' ... ]]

##
## user1  . .   . . . . . . . .
## user2  . .   . . . . . . . .
## user3  . .   . . . . . . . .
## user4  . .   . . . . . . . .
## user5  1 .   . . . . 3 . . .
## user6  . .   . . . . . . . .
## user7  . .   . . . . . . . .
## user8  . 2.5 . . 3 4 . . . .
## user9  . .   . . . . . . . .
## user10 . .   . . . . 3 . . .
```

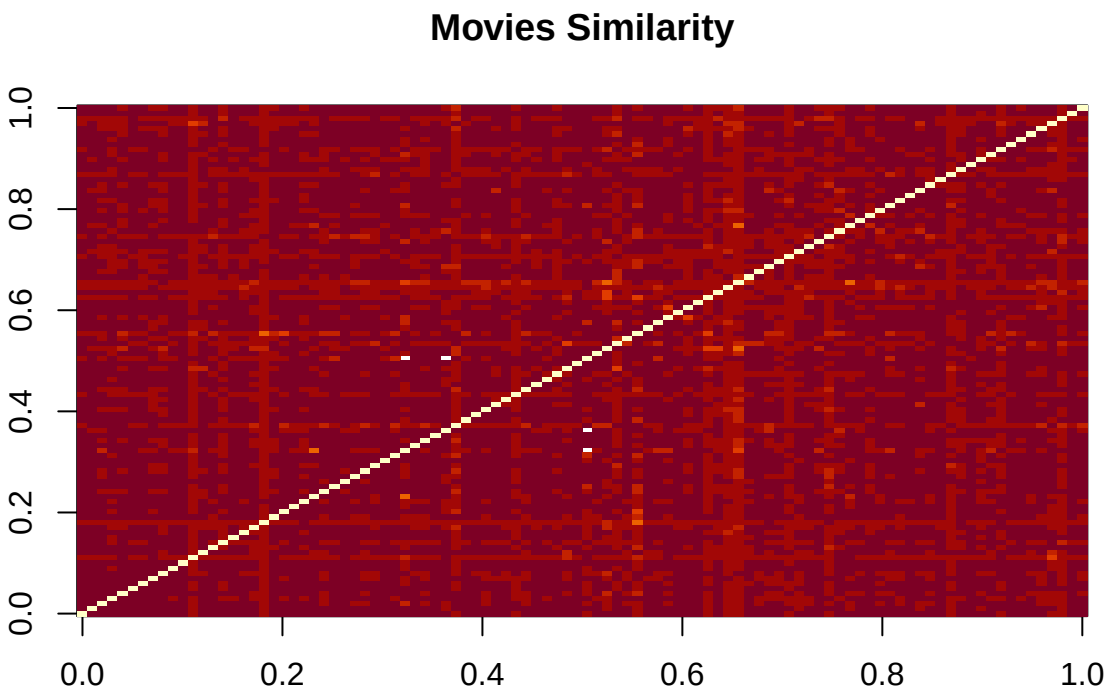
The sparse matrix above stores the users and movies and displays the rating made (if there is one).

Now comes the fun part: let us see if there is similarity in ratings for users and similarities in the ratings for different movies. For example, if user1 likes to watch action films and so does user2 they will have a high similarity. This way, in the future, any action movie that is rated well by user1 may be recommended to user2 since they have similar ratings and taste in movies. This thought can be extended to movies as well. If movie1 and movie2 have a high similarity in ratings then if movie1 is watched and liked by a particular user, movie2 should be recommended to them since it is similar. From referencing the DATAFLAIR TEAM blog and Louis Mono blog we will try cosine similarity measures to accomplish this analysis:

```r
#recommenderlab makes computing similarities easy

#Convert rating matrix into a recommenderlab sparse matrix
ratingMatrix <- new("realRatingMatrix", data = sparse_ratings)


#Calculate similarity using the cosine similarity

similarity_users <- similarity(ratingMatrix[1:100,],
                               method = "cosine",
                               which = "users")

image(as.matrix(similarity_users), main = "User Similarity")
```




```r
#Compute similarity between  movies.

similarity_movies <- similarity(ratingMatrix[,1:100],
                               method = "cosine",
                               which = "items")

image(as.matrix(similarity_movies), main = "Movies Similarity")
```



Each column and row for the images above represents the correlation between two users or two movies. The more yellow/light in color the entry is the more similar the users or movies are to eachother. The diagonal lines represents the correlation between itself which is why it is brightest yellow in color.

Note that the white represents a lack of data in the correlation (i.e. the users did not rate the same movies so we cannot correlate them). However, we can see some users have more similarity between each other then others. The movie similarity effect is more notable through seeing more entries with bright red to yellow color.

To summarize this section we have seen: (1) the data given to us is processed in a friendly format such that most features we are concerned about are easily accesible (i.e. rating, user, movie genre), (2) for future exploration, splitting the genre into a one-hot encoding format would be useful, and (3) similarity measures between different users and movies can be found providing some additional features for movie recommendations. With our data processed we are now ready to explore different types of models.

2.3: Model exploration: regression

Before we begin discussing our model it is first important to note the process of choosing the best model. In machine learning it is important to hold out a test set (our data-set called validation) for very last. We will split our edx data-set into a training and test set. The training set will be used to train the model and the test set will be used to see the performance of the model. We will choose the best model according to the performance (lowest RMSE) against the test set.

With the best model chosen from our training-test set we will report the final RMSE value. That is, we will report how our model works against unseen data - which is our validation dataset. Let us split our training and test set into a 80%/20% split:

```r
# set seed is used to ensure we get the same training and test set upon each run
set.seed(755, sample.kind = "Rounding")
```

```
## Warning in set.seed(755, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

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

Great! Now that we have partioned our data-sets for training and testing, we will follow the same approach that was taught to us by Professor Irizarry for model creating. That is we will use regression models to try and minimize our RMSE.

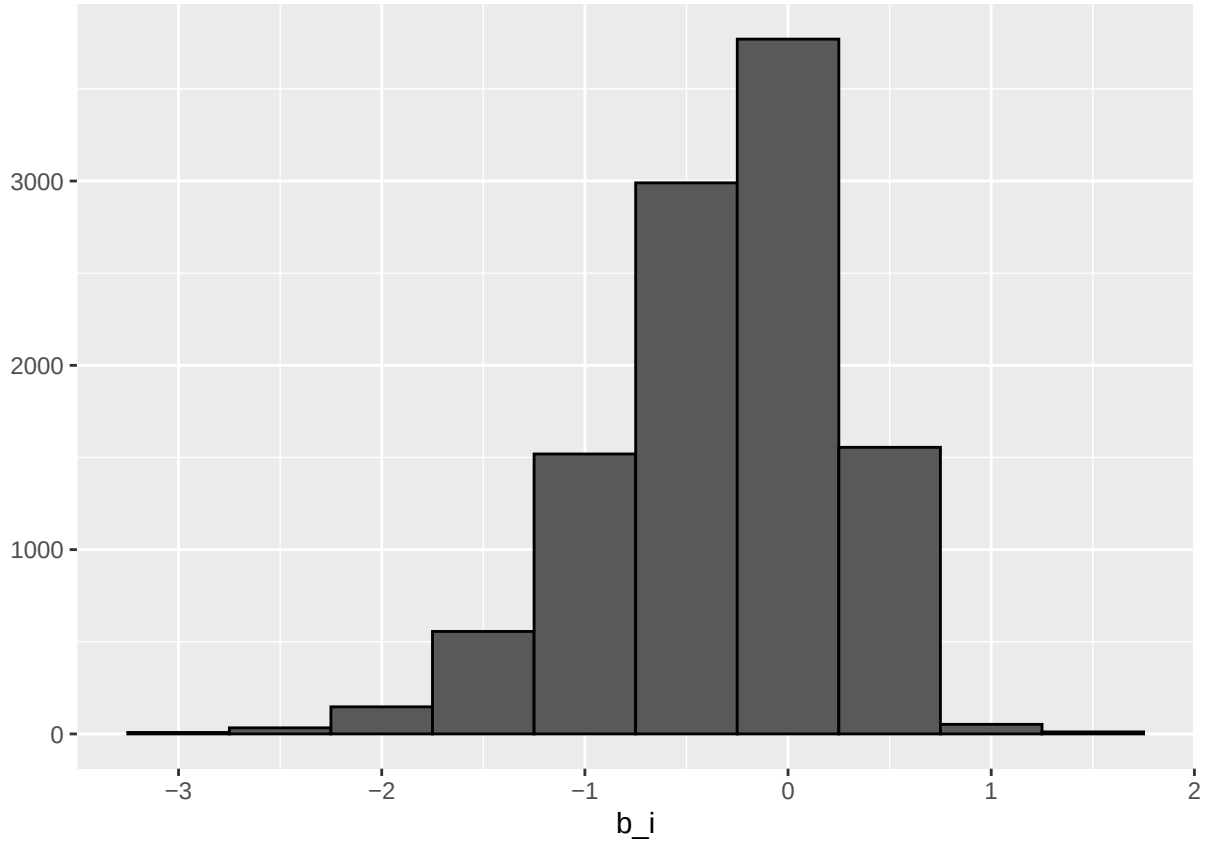
Model 1: Movie rating effect

The first model we would like to try is the movie rating effect. That is our predicted label y_hat, will be equal to the average rating of all movies (mu) and the average rating for each movie (b_i). Remember all models will have some error associated with it. Therefore: * y_hat <- mu+b_i+error

```r
# calculate the average of all ratings of the edx set
mu <- mean(train_set$rating)

#calculate b_i on the training set
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# predicted ratings
predicted_ratings_bi <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

# set any movies who did not have a rating in training_set to the average
predicted_ratings_bi[is.na(predicted_ratings_bi)]<-mu
```

Lets take a look at our movie averages:

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



Model 2: Movie rating + user-specific effect

The second model we would like to try is the user-specific effect. That is, we will take our Model 1 and add to it the average rating of a particular user (b_u): * y_hat<- mu+b_i+b_u+error

```
# calculate b_u using the training set
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#predicted ratings
predicted_ratings_bu <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# set any movies who did not have a rating in the training_set to the average
predicted_ratings_bu[is.na(predicted_ratings_bu)]<-mu
```

Model 3: Movie rating + user-specific + genre rating

We saw in data exploration that there was strong evidence of a genre effect. We will define g_u_i as the genre for user u's rating of movie i. We will calculate the average ratings for each genre. Thus, given any movieId we will add the avg rating for its respective genre as well. Therefore we have: * y_hat = mu + b_i + b_u + b_g + error

```
# calculate b_g using the training set
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

#predicted ratings
predicted_ratings_bg <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

# set any movies who did not have a rating in the training_set to the average
predicted_ratings_bg[is.na(predicted_ratings_bg)]<-mu
```

Model 4: Movie rating + user-specific + regularization

To better our models above and ensure we are converging to the best solution let us add a regularization constant for our movie rating and user specific effect. The purpose of this is to penalize large estimates that come from small sample sizes. Therefore, our estimates will try its best to guess the correct rating while being punished if the movie rating, user-specific, or genre effect is too large.

This method is a little more complicated than the prior two methods. That is we want to also validate how much we want to regularize the movie-rating, genre, and user-specific effects. We will try several regularization models with our regularization constant (lambda) at different values. We will define the function here and apply it in our results section:

```
lambdas <- seq(0, 10, 0.25)

regularize<- function(l){

    mu_reg <- mean(train_set$rating)

    b_i_reg <- train_set %>%
      group_by(movieId) %>%
      summarize(b_i_reg = sum(rating - mu_reg)/(n()+l))

    b_u_reg <- train_set %>%
      left_join(b_i_reg, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+l))

    b_g_reg <- train_set %>%
```

```
    left_join(b_i_reg, by="movieId") %>%
    left_join(b_u_reg, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g_reg = sum(rating - b_i_reg - b_u_reg - mu_reg)/(n()+1))

  predicted_ratings_b_i_u_g <-
    test_set %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_g_reg, by= 'genres') %>%
    mutate(pred = mu_reg + b_i_reg + b_u_reg + b_g_reg) %>%
    .$pred

  predicted_ratings_b_i_u_g[is.na(predicted_ratings_b_i_u_g)]<-mu

  return(RMSE(predicted_ratings_b_i_u_g, test_set$rating))
}
```

SECTION 3: Results

In this section we will see how well our models worked to our test set. We will then validate our model to unseen data in the Validation set.

Before beginning let us define our RMSE function:

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

3.1 Model 1: Movie rating effect

```
rmse_model1 <- RMSE(predicted_ratings_bi,test_set$rating)
rmse_model1
```

```
## [1] 0.9439937
```

3.2 Model 2: Movie rating + user-specific effect

```
rmse_model2 <- RMSE(predicted_ratings_bu,test_set$rating)
rmse_model2
```

```
## [1] 0.8666503
```

3.3 Model 3: Movie rating + user-specific + genre effect

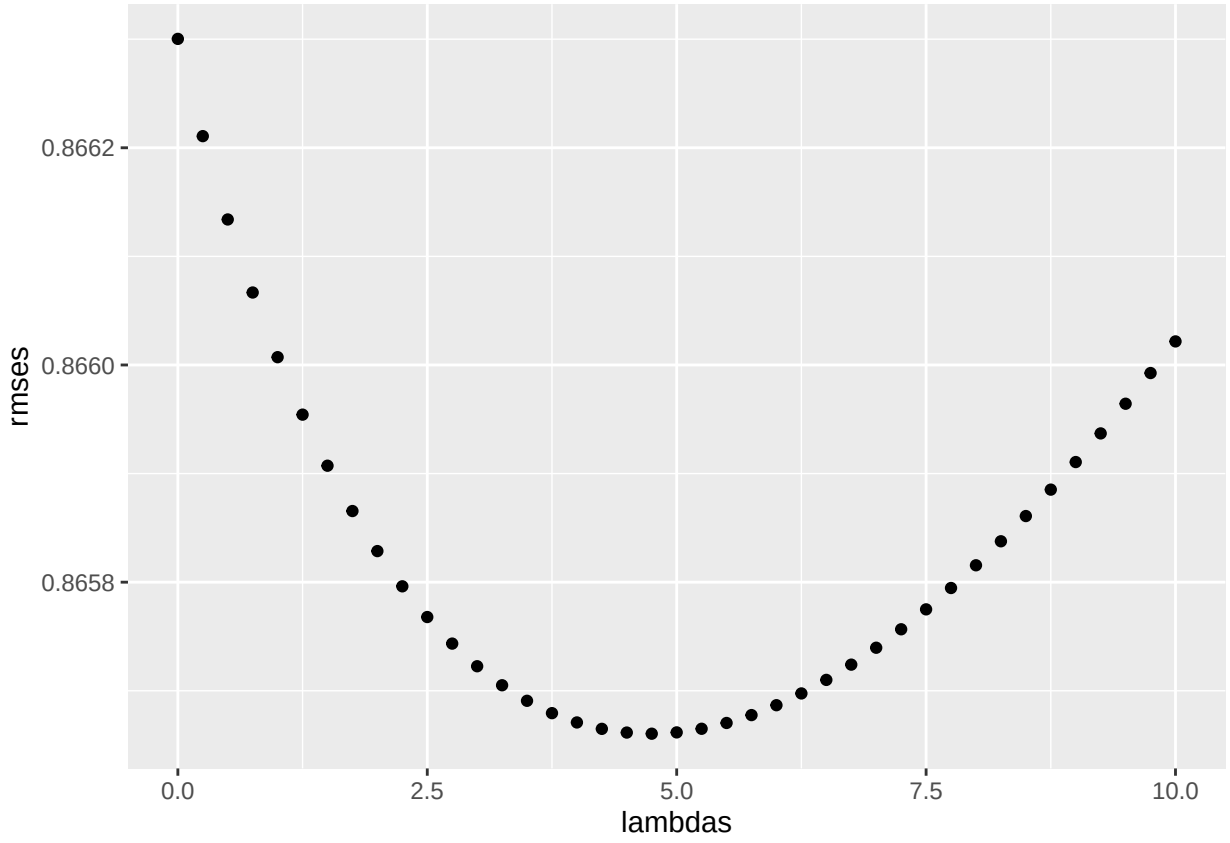
```
rmse_model3 <- RMSE(predicted_ratings_bg,test_set$rating)
rmse_model3
```

```
## [1] 0.8663003
```

3.4 Model 4: Movie rating + user-specific + genre effect + regularization

This result is a little more complicated than the prior two methods. That is we want to also validate how much we want to regularize the movie-rating and user-specific effects. We will try several regularization models with our regularization constant (lambda) at different values:

```
rmses <- sapply(lambdas, regularize)


qplot(lambdas, rmses)
```



Our minimum error is found with a lambda value of 4.75:

```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

```
rmse_model4 <- min(rmses)
rmse_model4
```

```
## [1] 0.8656605
```

3.4 Model Summary

The table below summarizes our RMSE values given our models explored:

```
#summarize all models RMSE values against the test-set
rmse_results <- data.frame(methods=c("movie effect","movie + user effects","move + user + genre effect"

#create table
kable(rmse_results) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="#D7261E")
```

methods	rmse
movie effect	0.9439937
movie + user effects	0.8666503
move + user + genre effect	0.8663003
regularized movie + user effect	0.8656605

As we can see the regularization model performmed best when lambda was set to 4.75 with an RMSE of 0.8657. One way to further drop our RMSE is to explore more models (i.e. perhaps try introducing a time feature). However, let us see what happens when we train our best model to our edx data-set (no need to separate a training and test set now that we have tuned and chosen our best model). Spoiler alert: our edx tuned regression model performs exceptional well to unseen data such as the validation dataset.

3.5 Validation: reporting our final RMSE

As mentioned we will train our regularized model (which takes into account (1) average movie rating, (2) movie-rating, (3) user-rating, and (4) genre-rating effects) with a regularization parameter, lambda, of 4.75.

```
mu_reg <- mean(edx$rating)
b_i_reg <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating - mu_reg)/(n()+lambda))

b_u_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+lambda))

b_g_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
```

```
    left_join(b_u_reg, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g_reg = sum(rating - b_i_reg - b_u_reg - mu_reg)/(n()+lambda))

predicted_ratings_b_i_u_g <-
    validation %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_g_reg,by='genres') %>%
    mutate(pred = mu_reg + b_i_reg + b_u_reg+ b_g_reg) %>%
    .$pred


predicted_ratings_b_i_u_g[is.na(predicted_ratings_b_i_u_g)]<-mu

rmse_validation<-RMSE(predicted_ratings_b_i_u_g, validation$rating)
rmse_validation
```

```
## [1] 0.8644514
```

As shown above our RMSE value against unseen data (our validation dataset) is 0.8644514 which meets our project objective of an RMSE <= 0.8649.

```
rmse_results <- data.frame(methods=c("Final evaluation (validation):"),rmse = c(rmse_validation))

kable(rmse_results) %>%
  kable_styling(bootstrap_options = "striped" , full_width = F , position = "center") %>%
  kable_styling(bootstrap_options = "bordered", full_width = F , position ="center") %>%
  column_spec(1,bold = T ) %>%
  column_spec(2,bold =T ,color = "white" , background ="#D7261E")
```

methods	rmse
Final evaluation (validation):	**0.8644514**

SECTION 4: Conclusion

To conclude this project we have achieved an exceptional RMSE of 0.8644514 against our validation dataset. Given more time I would like to try one-hot encoding on the genres and explore how a model such as KNN would work on this data.

Thank you for reading and I hope you found reading this project as fufilling as I found completing it.

SECTION 5: References

All material in this project is credited to Professor Rafael Irizarry and his team at HarvardX's Data Science course. Most material was learned through his course, book, and github: *Irizarry,R 2018 Recommender

systems,github page,accessed 4 December 2019, https://rafalab.github.io/dsbook/recommendation-systems.html

Another great resource was prior movie recommendation projects. Specifically, two projects were referenced:

(1) The work done by the DATAFLAIR TEAM (how I was inspired to try one-hot encoding for genres) *https://data-flair.training/blogs/data-science-r-movie-recommendation/ and,

(2) The work done by Louis Mono (how I was inspired to try data exploration techniques such as similarity measures) *http://rpubs.com/Jango/486734

SECTION 6: Appendix

Supplementary code

Below is some supplementary code that was talked about through the report. Given better processing units and more time exploring these methods would be valuable.

List of genres and their ratings:

Below is code for finding all unique genres and summarize the amount of ratings for them. Due to an extremely long run time and poor computing this code was avoided. That said, this code was not critical and was simply a step in our data exploration.

```
# top_genres <- edx %>% separate_rows(genres, sep = "\\|") %>%
#   group_by(genres) %>%
#   summarize(count = n()) %>%
#   arrange(desc(count))
#
# top_genres
```

One-hot genre encoding

Below is the code that was created to execute genre encoding. Due to the very large size of data set edx and limited computing power this took to long to execute and thus was not further explored. Given more time I would have liked to see if this genre encoding would allow use of algorithms such as K-Nearest Neighbors.

```
# edx_genre <- as.data.frame(edx$genres, stringsAsFactors=FALSE)
# library(data.table)
# edx_genre_final <- as.data.frame(tstrsplit(edx_genre[,1], '[|]',
#                                    type.convert=TRUE),
#                            stringsAsFactors=FALSE)
# colnames(edx_genre_final) <- c(1:8)
# list_genre <- c("Action", "Adventure", "Animation", "Children",
#                 "Comedy", "Crime","Documentary", "Drama", "Fantasy",
#                 "Film-Noir", "Horror", "Musical", "Mystery","Romance",
#                 "Sci-Fi", "Thriller", "War", "Western")
# genre_matrix <- matrix(0,9000056,18)
# genre_matrix[1,] <- list_genre
# colnames(genre_matrix) <- list_genre
# for (index in 1:nrow(edx_genre_final)) {
```

```
#   for (col in 1:ncol(edx_genre_final)) {
#     gen_col = which(genre_matrix[1,] == edx_genre_final[index,col]) #Author DataFlair
#     genre_matrix[index+1,gen_col] <- 1
# }
# }
# genre_matrix_final <- as.data.frame(genre_matrix[-1,], stringsAsFactors=FALSE) #remove first row, whi
# for (col in 1:ncol(genre_matrix_final)) {
#   genre_matrix_final[,col] <- as.integer(genre_matrix_final[,col]) #convert from characters to intege
# }
# str(genre_matrix_final)
```