movielens10m capstone

Andy Haas

December 30, 2018

INTRODUCTION:

We are going to inspect data consisting of over 9 million movie ratings. We will investigate trends in the data and try to make predictions on how different users will rate different movies. We will use accuracy to measure the quality of our algorithm because that is what we are graded on; however, we will also look at the RMSE (root mean squared error) Here is the code used to load and prepare the data.

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################

# Note: this process could take a couple of minutes


if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
```

```
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data

set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set

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

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Now lets look at the structure of the data.

```
str(edx)
```

```
## '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  "Bird of Prey (1996)" "Bad Moon (1996)" "Arsenic and Old Lace (1944)" "Some Kind
##  $ genres   : chr  "Action" "Action|Adventure|Horror" "Comedy|Mystery|Thriller" "Drama|Romance" ...
```

We have 9000055 observations and 6 variables. We see that each row is a rating given to one movie by one user. The title includes the release year, which we will need to seperate into a new variable if we plan to use release year. We also see that some movies have more than one genre listed under the genres category. If we choose to use genres as a predictor, we will need to find a way to parse this.

Let's take a look at how many different users and movies there are.

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

Something unusual happens if we look at the number of movie titles. You would expect it to be the same number as movieIds, but it is not.

```
edx %>% summarise(n_titles = n_distinct(title))
```

```
##   n_titles
## 1     8316
```

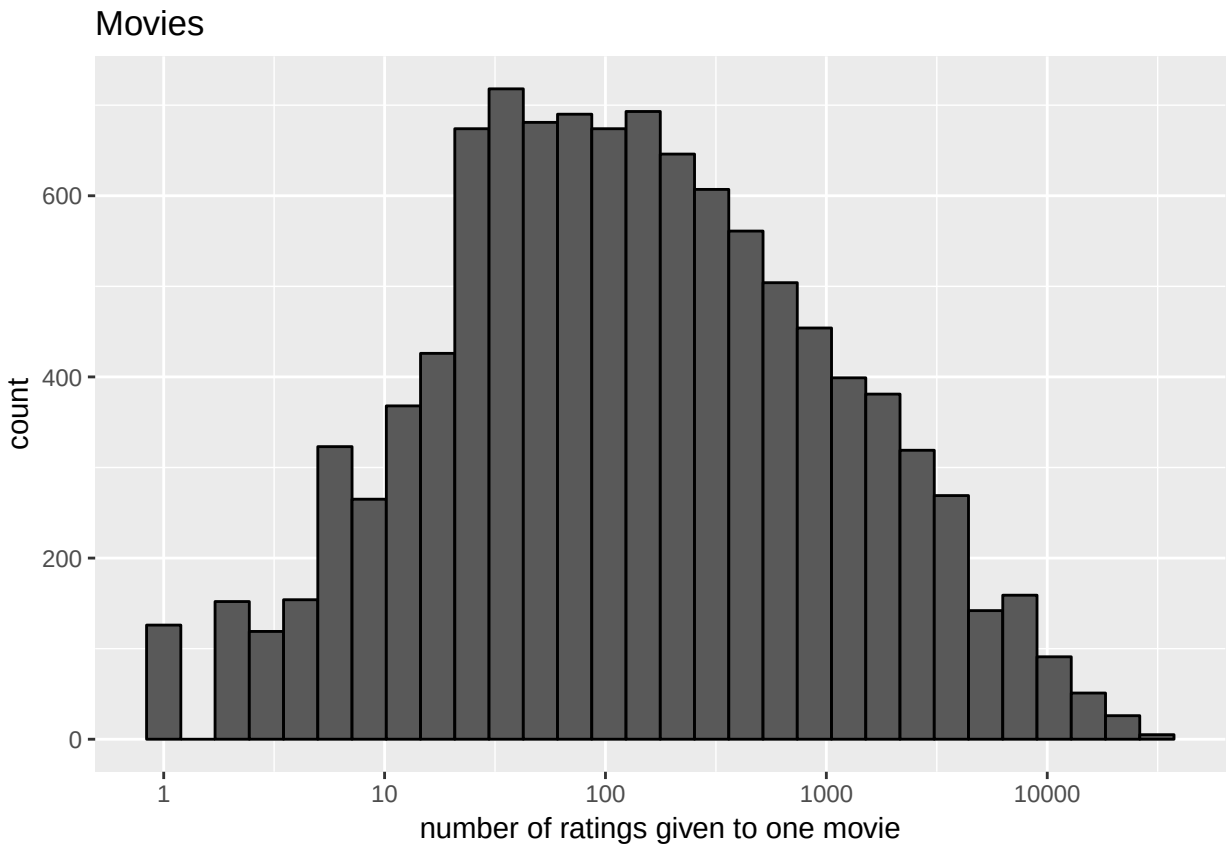
After some investigation, we find that there are 2 different movieId's for War of the Worlds (2005), but also 2361 movieIds with 'NA' in the title column.

```
edx[!duplicated(edx[,c('movieId')]),] %>% group_by(title) %>% count(title) %>% filter(n>1)
```

```
## # A tibble: 2 x 2
## # Groups:   title [2]
##   title                        n
##   <chr>                    <int>
## 1 War of the Worlds (2005)     2
## 2 <NA>                      2361
```

There are 410972 ratings given to movies with NA in the title.

Now let's look at the distribution of the number of ratings given to movies. Some movies are rated much more than other movies, While some movies are rated only once. The average number of ratings per movie is 843.



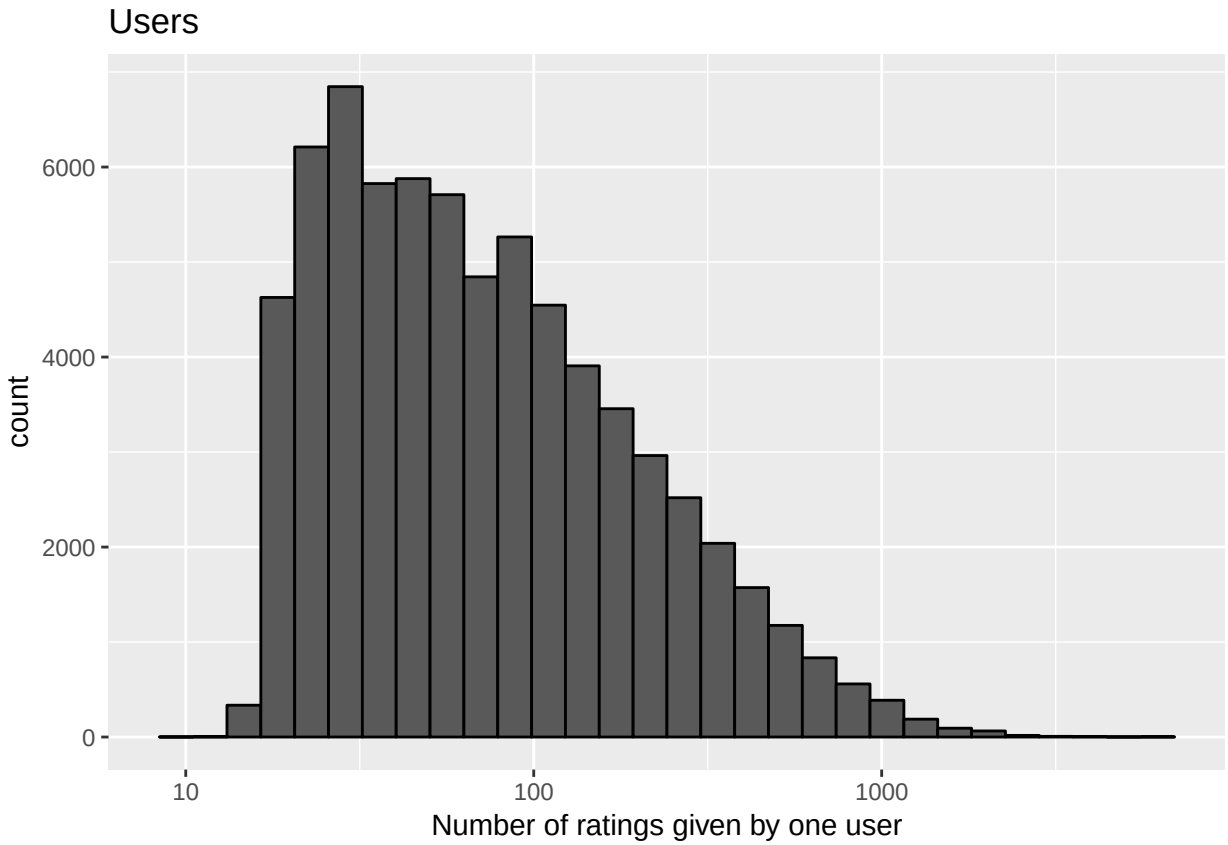
Let's look at the top 10 most rated movies. Patton is the most rated movie with 31362 ratings.

```
edx$n<- as.numeric(ave(edx$movieId,edx$movieId ,FUN = length))
edx[!duplicated(edx[,c('movieId')]),] %>% arrange(desc(n)) %>% select(title,n) %>% slice(1:10)
```

```
##                                          title     n
## 1                                Patton (1970) 31362
## 2                 Amityville Horror, The (1979) 31079
## 3                       Blue in the Face (1995) 30382
## 4                           Guantanamera (1994) 29360
## 5                            Being There (1979) 28015
## 6                 American President, The (1995) 26212
## 7                           Murder at 1600 (1997) 25998
## 8                 Speed 2: Cruise Control (1997) 25984
## 9  Star Maker, The (L'Uomo delle stelle) (1995) 25672
## 10       Monty Python and the Holy Grail (1975) 24284
```

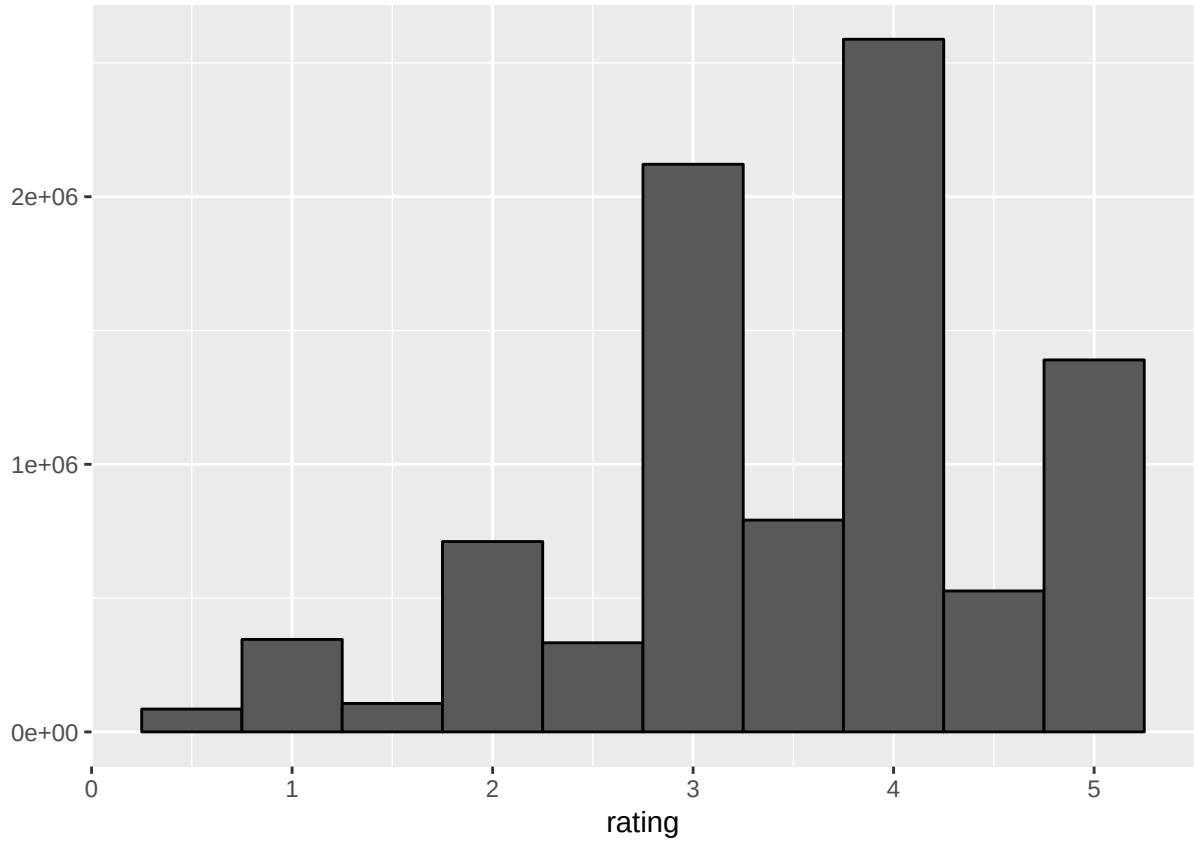
```
edx$n <- NULL
```

Now let's look at the user distribution.



It looks like most users rate somewhere between 50 to 150 movies. The average number of ratings per user is 129, and the most number of ratings given by one user is a whopping 6616 ratings.

The next thing to look at is the actual rating. Ratings are on a scale of from 1 to 5. Zero cannot be given as a rating. All ratings are also multiples of 0.5. i.e. 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5. Let's look at the distribution of ratings.



Now this is very interesting. We see whole numbers are rated much more often than half ratings. This could be the key to making a good guessing algorithm. We also see that 3 and 4 are have the highest occurence, and by a large margin.

Last, let's look at the effect of genre. Does one genre receive better ratings than another? First lets find what the different genres are.

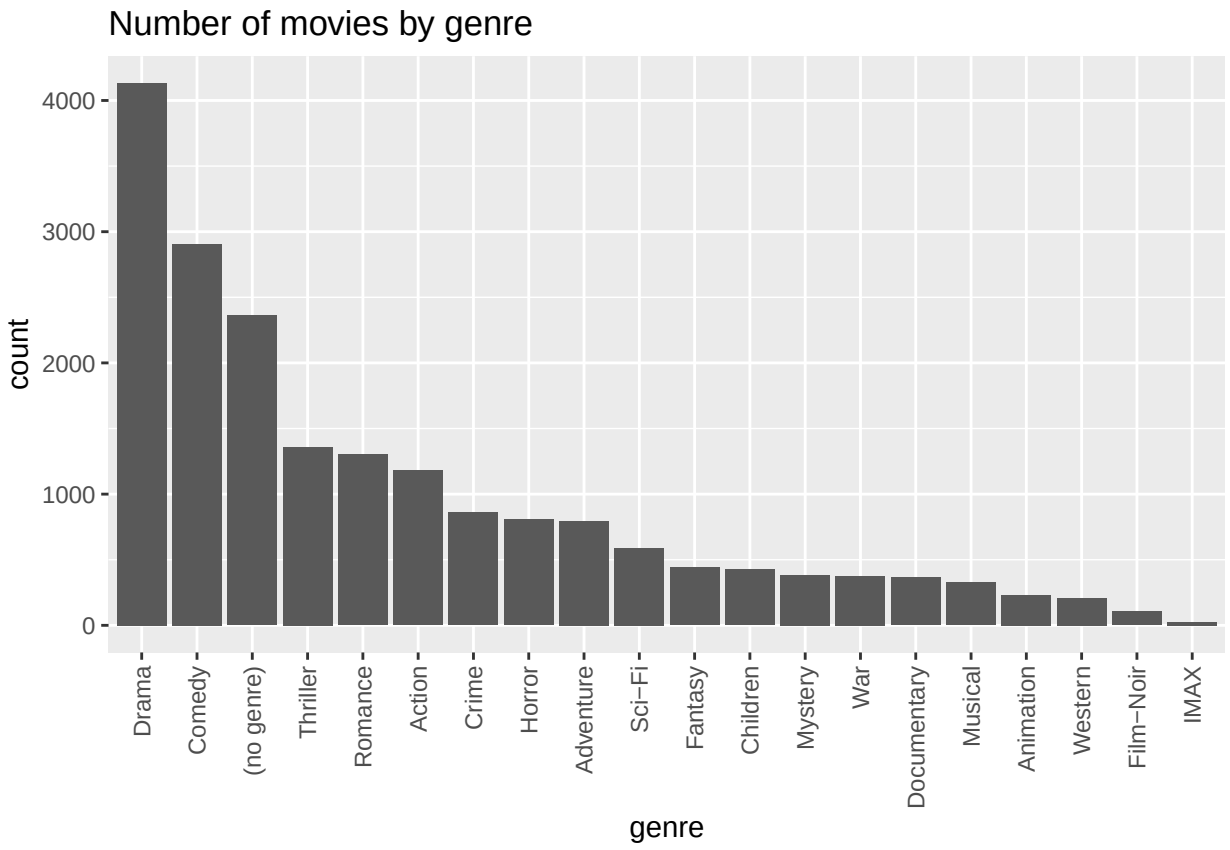
```
split_genres <- function(g){strsplit(g,"|",fixed = TRUE)}
genre_names<- lapply(as.character(edx$genres),split_genres)
genre_names <- unique(unlist(genre_names))
genre_names[is.na(genre_names)] <- "(no genre)"
genre_names
```

```
##  [1] "Action"      "Adventure"   "Horror"      "Comedy"      "Mystery"
##  [6] "Thriller"    "Drama"       "Romance"     "Sci-Fi"      "Children"
## [11] "War"         "Crime"       "Animation"   "Musical"     "Documentary"
## [16] "(no genre)"  "Fantasy"     "Film-Noir"   "Western"     "IMAX"
```

There are 20 genres, one of which is NA. We will change this to a string "(no genre)" in the whole edx dataframe. Let's look at the distribution of the genres.

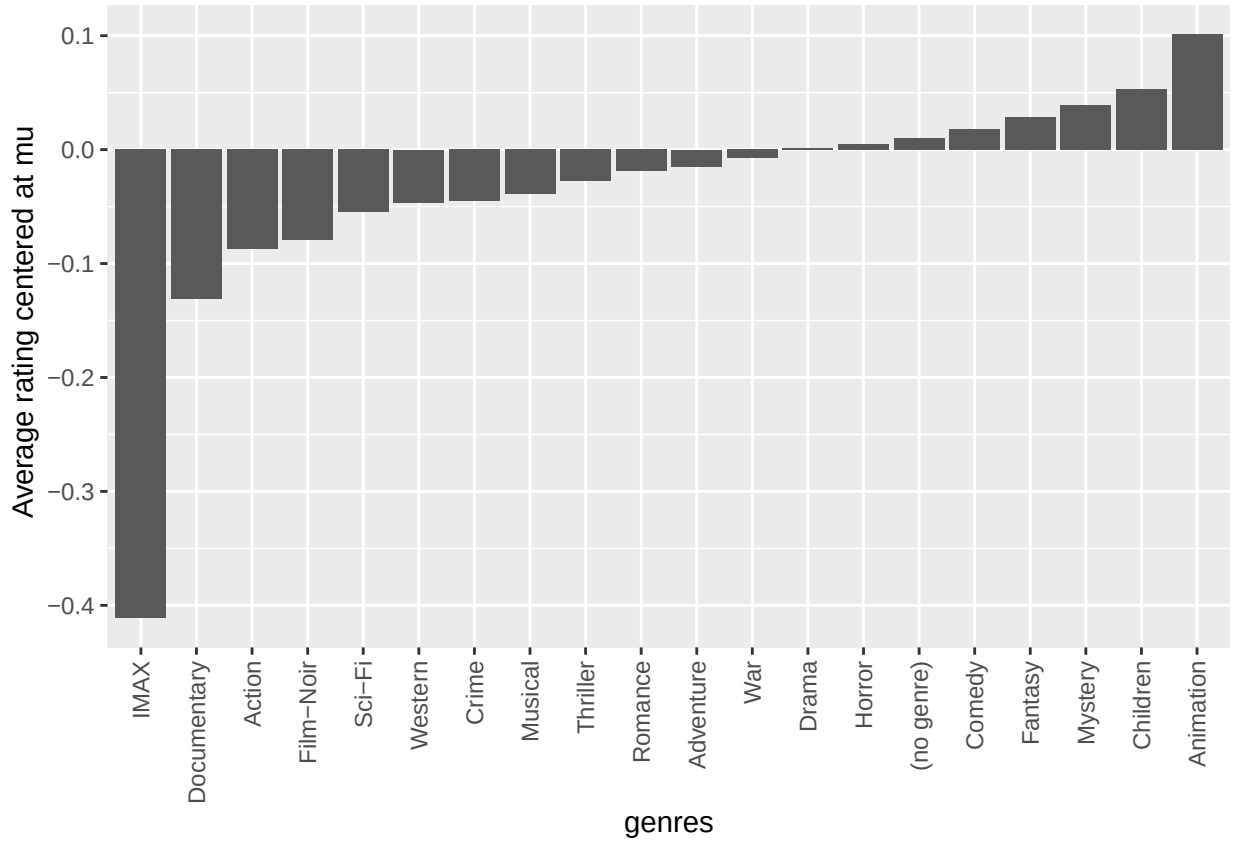
```
edx$genres[is.na(edx$genres)] <- "(no genre)"
count_genres <- function(genre){
  edx %>% distinct(movieId,.keep_all = TRUE) %>% filter(grepl(genre,genres)) %>% nrow()
}
genre_counts <- lapply(genre_names, count_genres)
genre_df <- data.frame(genres = genre_names, count = unlist(genre_counts), row.names = NULL)
```

```
genre_df %>% ggplot(aes(x = reorder(genres, -count),count)) +
  geom_bar(stat = "identity")+
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
  ggtitle("Number of movies by genre") +
  xlab("genre")
```



Now lets look at the average rating per genre. We will center the averages at mu, the average for all movies.

```
mu <- mean(edx$rating)
get_genre_averages <- function(genre){
  edx %>% filter(grepl(genre,genres)) %>% summarise(mean(rating-mu))
  }
genre_averages <- sapply(genre_names,get_genre_averages)
genre_dr <- data.frame(genre = genre_names, b_g = unlist(genre_averages),row.names = NULL)
genre_dr %>% ggplot(aes(x = reorder(genre, b_g),b_g)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x=element_text(angle=90,hjust=1,vjust=0.5)) +
  xlab("genres") +
  ylab(" Average rating centered at mu")
```



We can see that IMAX has the lowest average rating, but it is only a half-star below average. The best rated genre is Animation, but it is only .1 star above average. Sadly, I would say using a genre bias to make predictions will not be so helpful.

Methods and Analysis

We will evaluate a few different methods, and analyse the results. We will keep the accuracy and RMSE for each method stored in a dataframe called 'results'.

Let's start by partitioning our edx data. 90% will be used for train and 10% for test.

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-test_index,]
temp <- edx[test_index,]
edx_test <- temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
removed <- anti_join(temp, edx_test)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx_train <- rbind(edx_train, removed)
rm(test_index, temp, removed)
```

The mean and the mode prediction

The first two methods will be very basic. We will look at what our accuracy and RMSE are when using just the mean of all ratings, 3.5, and using the mode of all ratings, 4.

```
results <- NULL
four_accuracy <- mean(edx_test$rating==4)
four_RMSE <- RMSE(edx_test$rating,rep(4,times=nrow(edx_test)))
results <- data_frame(method = "Just the mode", Accuracy = four_accuracy, RMSE = four_RMSE)

mu_accuracy <- mean(edx_test$rating==3.5)
mu_RMSE <- RMSE(edx_test$rating,rep(3.5,times=nrow(edx_test)))
results <- bind_rows(results, data_frame(method = "Just the mean", Accuracy = mu_accuracy, RMSE = mu_RMS
results %>% knitr::kable()
```

method	Accuracy	RMSE
Just the mode	0.2880416	1.167939
Just the mean	0.0875156	1.061204

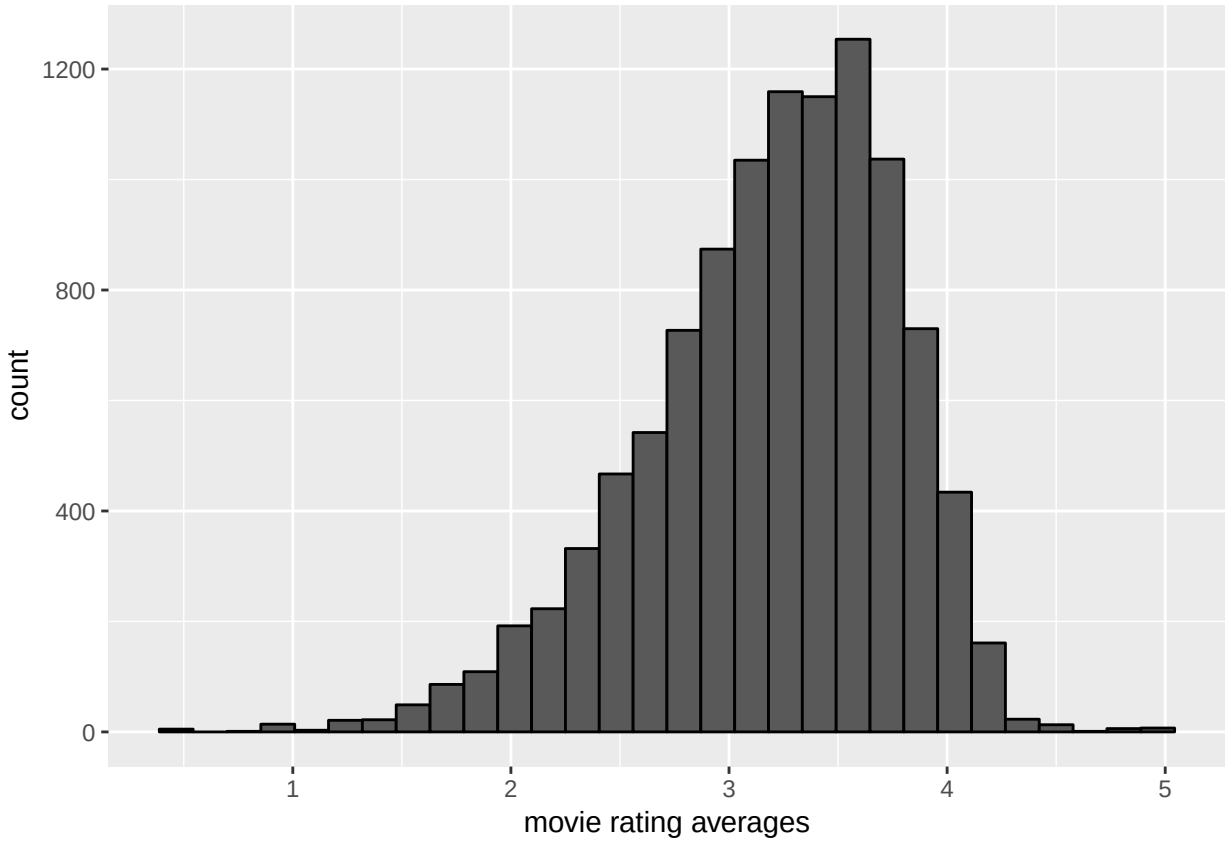
We see that the mode is substantially better than the mean, but still not very good. The accuracy is higher for the mode, but the RMSE is higher for the mean.

The Movie Effect model

Another method we have learned is user and movie bias. Some movies are rated much higher on average than others, while some are very often given low ratings. Let's calculate the average rating for all movies, mu, and the average rating given by all users to movie i, we will call this bias b_i and use this formula to calculate a new prediction.

$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$

We will also round all predictions to the nearest 0.5 for accuracy, but keep the decimal prediction for calculating the RMSE. Here is a histogram of movie rating averages.



We can see that there is definitely some variance in movie averages. This could help us change our prediction from just the mean or just the mode.

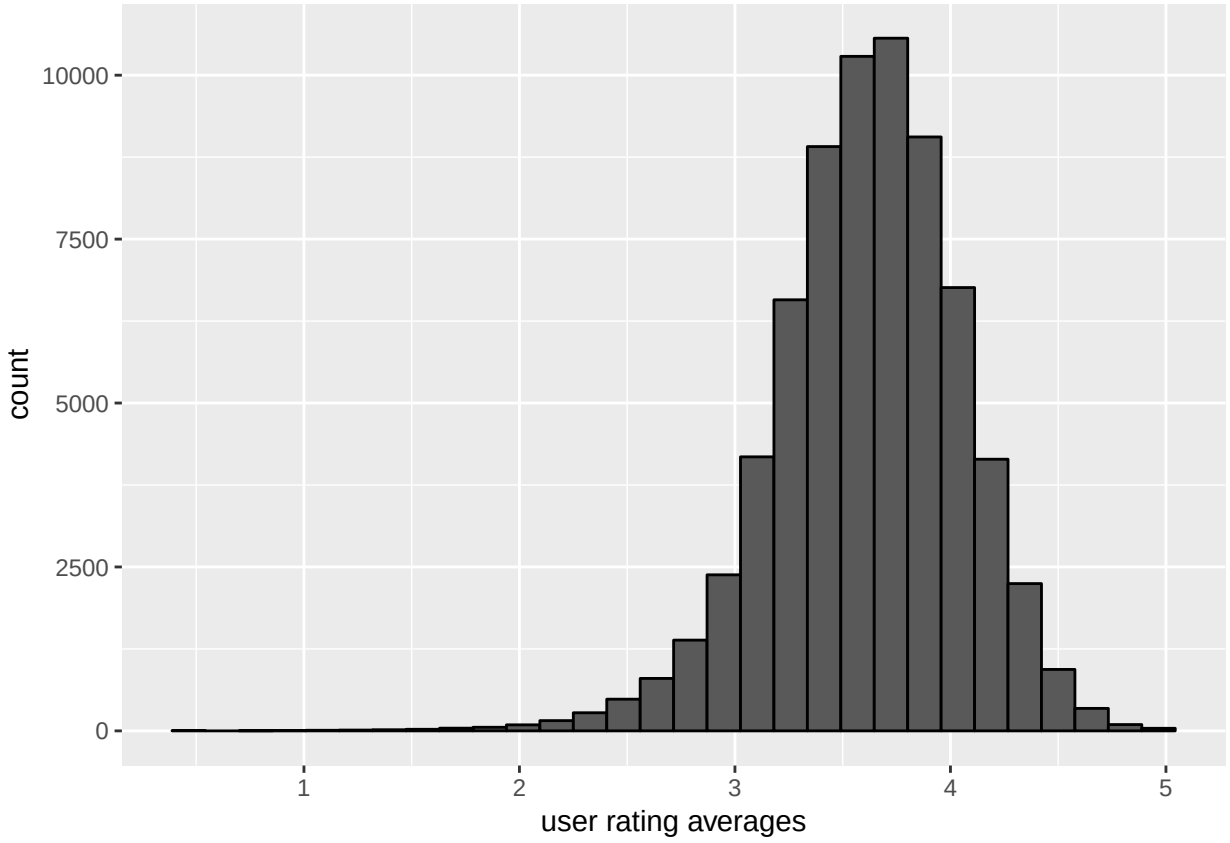
```
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings1 <- mu + edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
predicted_ratings1r <- round(predicted_ratings1/.5)*.5
model_1 <- mean(predicted_ratings1r == edx_test$rating)
model_1_RMSE <- RMSE(edx_test$rating,predicted_ratings1)
results <- bind_rows(results, data_frame(method="Movie Effect Model", Accuracy = model_1, RMSE = model_
results %>% knitr::kable()
```

method	Accuracy	RMSE
Just the mode	0.2880416	1.1679394
Just the mean	0.0875156	1.0612036
Movie Effect Model	0.2236352	0.9441568

Our accuracy actually got worse from just guessing a 4 for all movies. The RMSE is still improving though.

The Movie and User effect model

Some users give 5 stars to nearly every movie they see, and other movies are tough to please. By calculating user averages, we can apply a user bias to predict higher or lower depending on the user. Here is the code and results with the movie and user effect.



We see from the plot that there are indeed some users that requre us to take a user bias into account when making predictions.

```
user_avgs <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings2 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
predicted_ratings2r <- round(predicted_ratings2/0.5)*.5
model_2 <- mean(predicted_ratings2r == edx_test$rating)
model_2_RMSE <- RMSE(edx_test$rating,predicted_ratings2)
results <- bind_rows(results, data_frame(method="mu + b_i + b_u", Accuracy = model_2, RMSE = model_2_RMS
results %>% knitr::kable()
```

method	Accuracy	RMSE
Just the mode	0.2880416	1.1679394

method	Accuracy	RMSE
Just the mean	0.0875156	1.0612036
Movie Effect Model	0.2236352	0.9441568
mu + b_i + b_u	0.2637357	0.8262583

Our accuracy is getting a little better, but still very far from where I think it should be. The RMSE got much lower. That is a good sign. In fact, the RMSE is quite a bit lower than where we started.

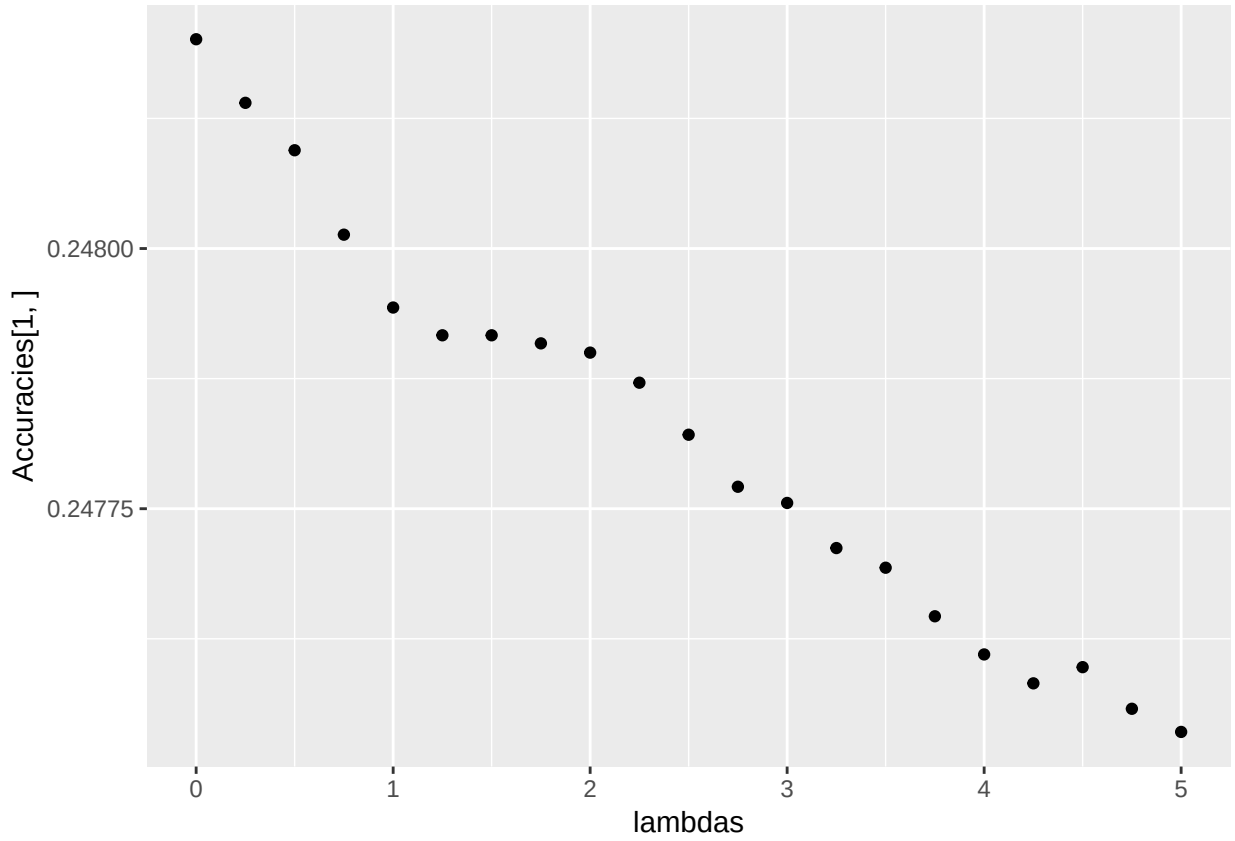
Regularization model

If we look at where our biggest mistakes are, we see it is from movies that have very few ratings, but are given a very low or very high rating. To give these movies less weight, we can introduce a lambda in the denominater when calculating the average rating. This is called regularization.

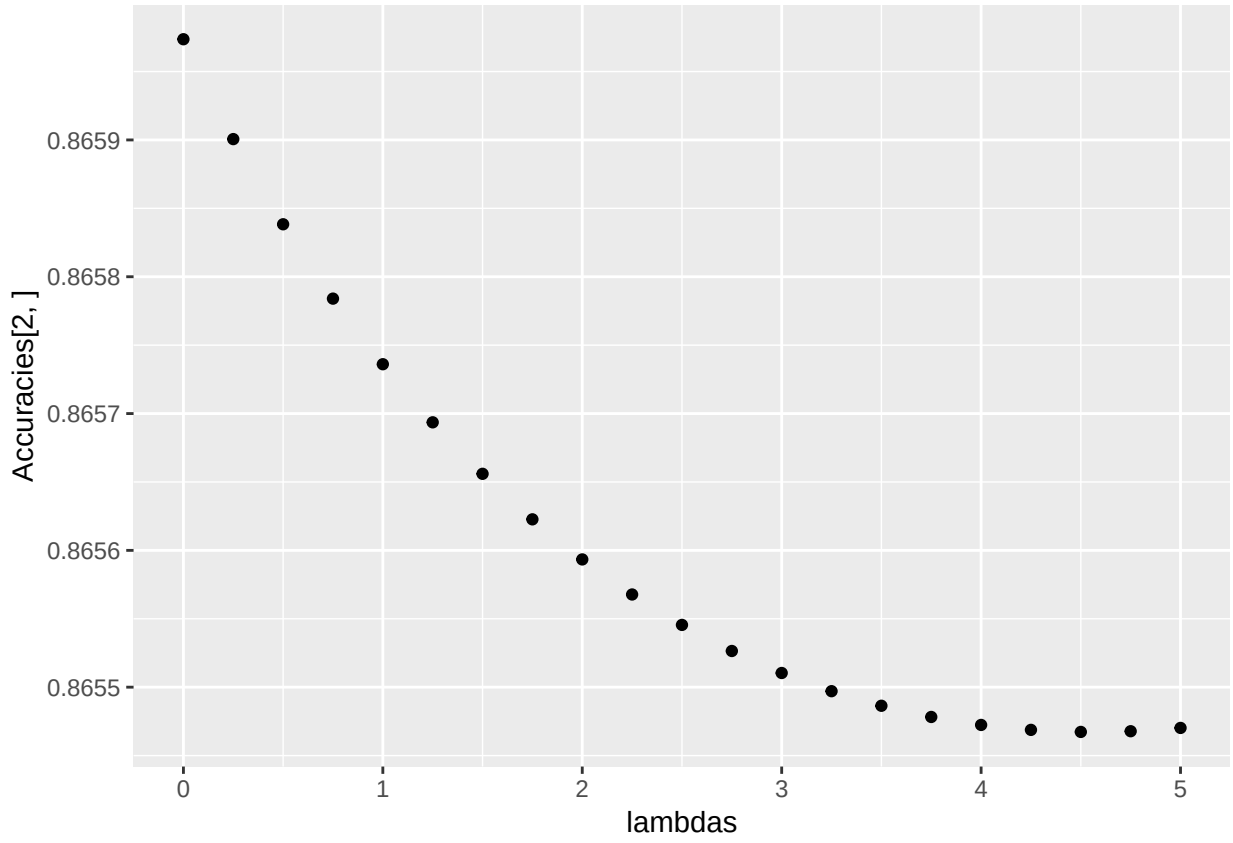
$\frac{1}{N}\sum_{u,i}\left(y_{i,u}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$

We will test over a seqeuence of different values of lambda and see which gives us the highest accuracy.

```
lambdas <- seq(0, 5, 0.25)

Accuracies <- sapply(lambdas, function(l){

  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings3 <-
    edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = (mu + b_i + b_u)) %>%
    .$pred

  return( c(mean(round(predicted_ratings3/0.5)*0.5==edx_test$rating), RMSE(edx_test$rating,predicted_ra
})

qplot(lambdas, Accuracies[1,])
```




```
qplot(lambdas, Accuracies[2,])
```




```
lambdas[which.max(Accuracies[1,])]
```

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

```
results <- bind_rows(results, data_frame(method="Regularization", Accuracy = max(Accuracies[1,]),
                                         RMSE = min(Accuracies[2,])))
results %>% knitr::kable()
```

method	Accuracy	RMSE
Just the mode	0.2880416	1.1679394
Just the mean	0.0875156	1.0612036
Movie Effect Model	0.2236352	0.9441568
mu + b_i + b_u	0.2637357	0.8262583
Regularization	0.2482011	0.8654673

Our accuracy got worse again, and a lambda of 0 yields the best accuracy, but a lambda of 4 yields the best RMSE.

recommenderlab models

I also did some testing using the recommenderlab package. tried a few different models available. Here are the results from the UBCF model (user-based collaborative filtering), IBCF model (item-based collaborative filtering), and SVD model (single variable decomposition).

The run time is quite long with a dataet of 100,000 entries, so I don't think we will be able to use it for our 10 million entry database, so we will use a smaller dataset for testing purposes. We will filter the edx

database to use only users and movies with over 800 ratings. This gives us 457,708 ratings given by 465 users of 2,195 different movies. We will then split it into a train and test set.

We will also create a matrix of dimension 465 x 2,195 where each row represents one user and each column represents each movie. Creating this matrix with many users and movies is what can potentialy break R. When I tried to create it using the full edx data, I received an error saying there is not enough memory to create the 6 gig matrix.

```
edx800<- edx %>%
  group_by(movieId) %>%
  filter(n() >= 800) %>% ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 800) %>% ungroup()
#make test and train set
test_index <- createDataPartition(y = edx800$rating, times = 1, p = 0.1, list = FALSE)
edx800_train <- edx800[-test_index,]
edx800_test<- edx800

y <- edx800_train %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
# rownames are the userId
rownames(y)<- y[,1]
y <- y[,-1]
movie_titles <- edx800_train%>%
  select(movieId, title) %>%
  distinct()
# colnames are the movieId
colnames(y) <- with(movie_titles, movieId[match(colnames(y), movieId)])
```

Now we will use the Recommender() function and predict() functions of the recommenderlab package to get ratings for our smaller test set, then calculate accuracy and RMSE.

```
library(recommenderlab)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

```
## Loading required package: arules
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
## Loading required package: proxy
```

```
##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry

##
## Attaching package: 'recommenderlab'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```
r <- as(y, "realRatingMatrix")
#UBCF
ubcf_rec=Recommender(r[1:nrow(r)],method="UBCF")
ubcf <- predict(object = ubcf_rec,r[1:nrow(r)],type="ratings")
ubcf <- as(ubcf,"matrix")

get_UBCF_test_ratings <- function(u){
  ubcf[which(rownames(ubcf)==edx800_test$userId[u]), which(colnames(ubcf)==edx800_test$movieId[u])]
}
ubcf_test_pred <- sapply(1:nrow(edx800_test),get_UBCF_test_ratings)

ubcf_test_pred[which(is.na(ubcf_test_pred))] <- 4
ubcf_test_predr <- round(ubcf_test_pred/.5)*.5
ubcf_test_pred_accuracy <- mean(ubcf_test_predr==edx800_test$rating)
ubcf_test_pred_RMSE <- RMSE(edx800_test$rating,ubcf_test_pred)

results <- bind_rows(results, data_frame(method="UBCF", Accuracy = ubcf_test_pred_accuracy,
                                         RMSE = ubcf_test_pred_RMSE))

#IBCF
ibcf_rec=Recommender(r[1:nrow(r)],method="IBCF")
ibcf <- predict(object = ibcf_rec,r[1:nrow(r)],type="ratings")
ibcf <- as(ibcf,"matrix")

get_IBCF_test_ratings <- function(u){
  ibcf[which(rownames(ibcf)==edx800_test$userId[u]), which(colnames(ibcf)==edx800_test$movieId[u])]
}
ibcf_test_pred <- sapply(1:nrow(edx800_test),get_IBCF_test_ratings)

ibcf_test_pred[which(is.na(ibcf_test_pred))] <- 4
ibcf_test_predr <- round(ibcf_test_pred/.5)*.5
ibcf_test_pred_accuracy <- mean(ibcf_test_predr==edx800_test$rating)
ibcf_test_pred_RMSE <- RMSE(edx800_test$rating,ibcf_test_pred)
```

```
results <- bind_rows(results, data_frame(method="IBCF", Accuracy = ibcf_test_pred_accuracy,
                                         RMSE = ibcf_test_pred_RMSE))

#SVD
svd_rec=Recommender(r[1:nrow(r)],method="SVD")
svd<- predict(object = svd_rec,r[1:nrow(r)],type="ratings")
svd<- as(svd,"matrix")

get_SVD_test_ratings <- function(u){
  svd[which(rownames(svd)==edx800_test$userId[u]), which(colnames(svd)==edx800_test$movieId[u])]
}
svd_test_pred <- sapply(1:nrow(edx800_test),get_SVD_test_ratings)

svd_test_pred[which(is.na(svd_test_pred))] <- 4
svd_test_predr <- round(svd_test_pred/.5)*.5
svd_test_pred_accuracy <- mean(svd_test_predr==edx800_test$rating)
svd_test_pred_RMSE <- RMSE(edx800_test$rating,svd_test_pred)

results <- bind_rows(results, data_frame(method="SVD", Accuracy = svd_test_pred_accuracy,
                                         RMSE = svd_test_pred_RMSE))
results %>% knitr::kable()
```

method	Accuracy	RMSE
Just the mode	0.2880416	1.1679394
Just the mean	0.0875156	1.0612036
Movie Effect Model	0.2236352	0.9441568
mu + b_i + b_u	0.2637357	0.8262583
Regularization	0.2482011	0.8654673
UBCF	0.2570656	1.2087653
IBCF	0.2479812	1.2459049
SVD	0.2567204	1.2087378

Results

I am very sad to report that the best accuracy was simply from predicting a 4 for all ratings. The accuracy was .288. I don't want to submit a vector of all 4's for my capstone, so I will use the method including both movie and user bias on the validation set and submit that.

I think the recommenderlab has potential to increase accuracy, possibly using a combination of user and movie effect, bias, and svd, but my computer doesn't have enough memory for creating a "realRatingMatrix" for the movielens10m data to be used by the recommenderlab package.

Conclusion

After thoroughly exploring the movielens dataset, I have learned a lot about movie ratings and predictions. This assignment served as an excellent challenge for me to test my R skills in many different ways. I learned to reshape dataframes, make plots, parse data using regular expressions, and probably the most useful skill, how to read documentation.

Thank you for taking the time to review my report. I hope you learned something, I sure did.