Movie Prediction Report

Introduction and Overview

This project uses a subset of a publically-available dataset (movielens) that provides user ratings of movies. The data used in this project is divided into 2 parts, one set for modeling/optimization (denoted edx in the code) and one set reserved for validation purposes (denoted validation).

The edx dataset has 9,000,055 observations (movie ratings), with 10,677 distinct movies and 69,878 users. All optimization and modeling decisions are made based on the edx dataset, to reserve the validation set for final assessment of the algorithm. The validation data set has 999,999 observations, with 9,809 distinct movies and 68,534 users.

Each movie can be rated from 0.5 to 5, in increments of 0.5. The data also labels the genre (listed as an aggregate category, such as Action|Adventure|Thriller), title of the movie, and time stamp of the rating.

The purpose of this project is to explore some of the factors that influence the rating of each movie, and to predict the ratings in the validation set. Our metric for a successful prediction is an RMSE no higher than 0.86490 for the validation set. Although many factors may have an influence on the rating, this analysis surpasses that criterion using only movie bias, user bias, and genre effects.

The steps performed are as follows:

1. Divide dataset into edx and validation.
2. Visualize and explore the data to get a feel for the influence of various predictors, and mutate the data set with binary genre labels to prepare it for analysis.
3. Analyse various effects using edx to tune the model's movie and user bias, then add in user-specific genre predictions for the top 4 most popular genres.
4. Assess the final model on the validation set.

Our final model has the following form:

$$y_{u,mov} = \bar{y} + b_{mov} + b_u + \sum_{k=1}^{4} x_{mov}^k b_{u,k},$$

where $\bar{y}$ is the average movie rating, b_{mov} is each movie's bias relative to the mean, b_u is each user's bias relative to $\bar{y} + b_{mov}$. Finally, each movie has a set of binary variables x_{mov}^k for each of the top 4 genres (labeled k).

Each of these binary variables (denoted in the data set as *action*, *drama*, *thriller* and *comedy*) is set to 0 or 1 depending on whether the observation's genre labels contains that particular genre. For example, a movie labeled "Action|IMAX|Thriller" would have *action* equal 1, *thriller* equal 1, *drama* equal 0 and *comedy* equal 0. The genre bias $b_{u,k}$ is user-specific, since individual movie-watchers have different preferences for genres.

Methods / Analysis

Step 1: Data cleaning

To create the data set, the movielens data file is downloaded and column names are applied to the data. Using CreateDataPartition from the caret package, the resulting dataframe is split so that 90% goes into *edx* for training and optimization, leaving 10% for validation. Since we cannot predict the rating of a movie or user that doesn't appear in the training set, any movies / users that appear in the validation set, but not in *edx*, are removed from *validation* and put into *edx*.

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

library(tidyverse)
library(caret)
library(data.table)

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

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

ratings <- fread(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")

# Since I am using R 4.0 or later:
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, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

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

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

Step 2: Data exploration and visualization, insights gained

In this step, we will explore potential predictors. The stringr package provides string processing, which will be used to peel off individual genre labels from the aggregate genre categories.

We start by examining the dimensions of *edx* and *validation*. The *edx* dataset, for example, has 9000055 observations, with 10,677 distinct movies and 69,878 users.

```
library(stringr)
dim(edx) #How many rows and columns are there in the edx dataset?
```

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

```
count(edx,rating) #how many of each rating?
```

```
##     rating       n
##  1:    0.5   85374
##  2:    1.0  345679
##  3:    1.5  106426
##  4:    2.0  711422
##  5:    2.5  333010
##  6:    3.0 2121240
##  7:    3.5  791624
##  8:    4.0 2588430
##  9:    4.5  526736
## 10:    5.0 1390114
```

```
nrow(distinct(edx,movieId)) #how many distinct movies
```

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

```
nrow(distinct(edx,userId)) #how many distinct users
```

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

```
nrow(distinct(validation,movieId)) #how many distinct movies
```

```
## [1] 9809
```

```
nrow(distinct(validation,userId)) #how many distinct users
```

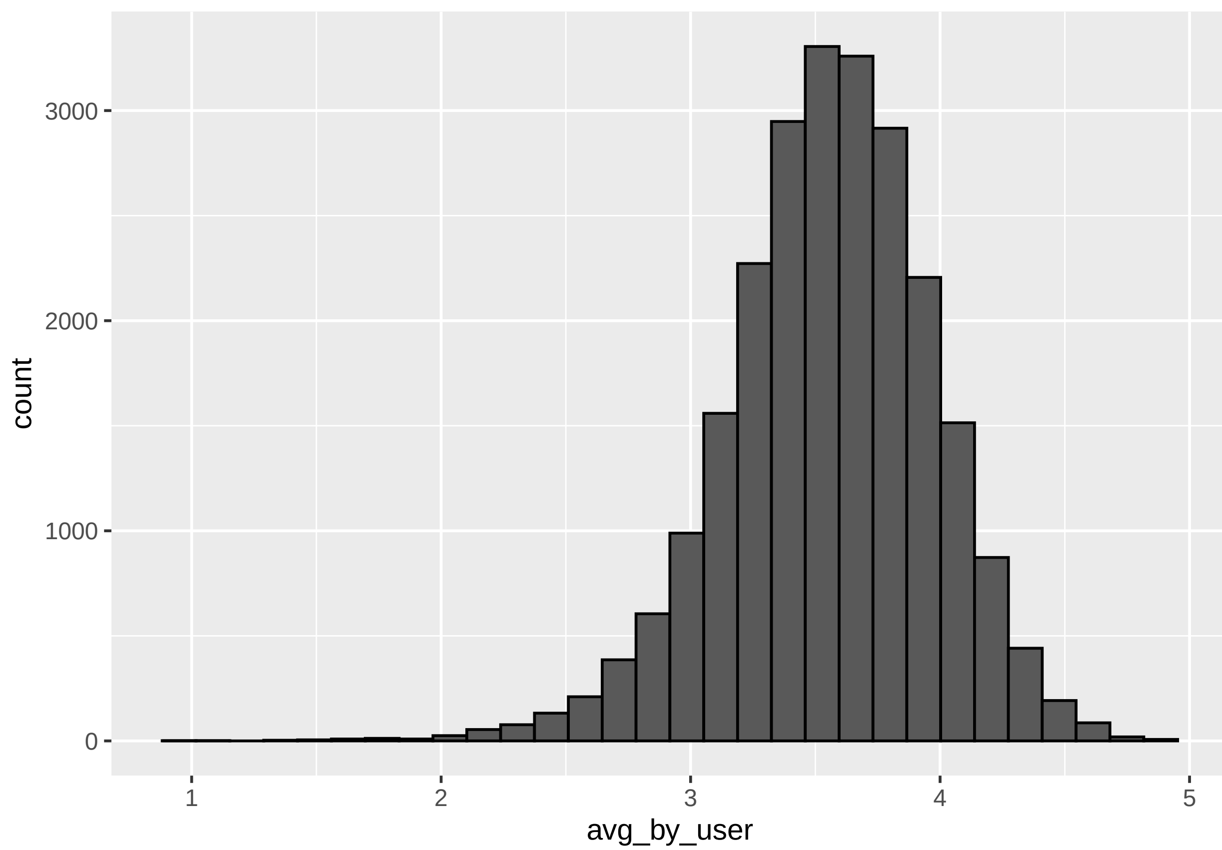
```
## [1] 68534
```

```
dim(validation)
```

```
## [1] 999999      6
```

One now explores whether the ratings seem to be affected by the user, the movie, and the genre. Restricting to users who have rated at least 100 movies, we see that users' averages have a broad range:

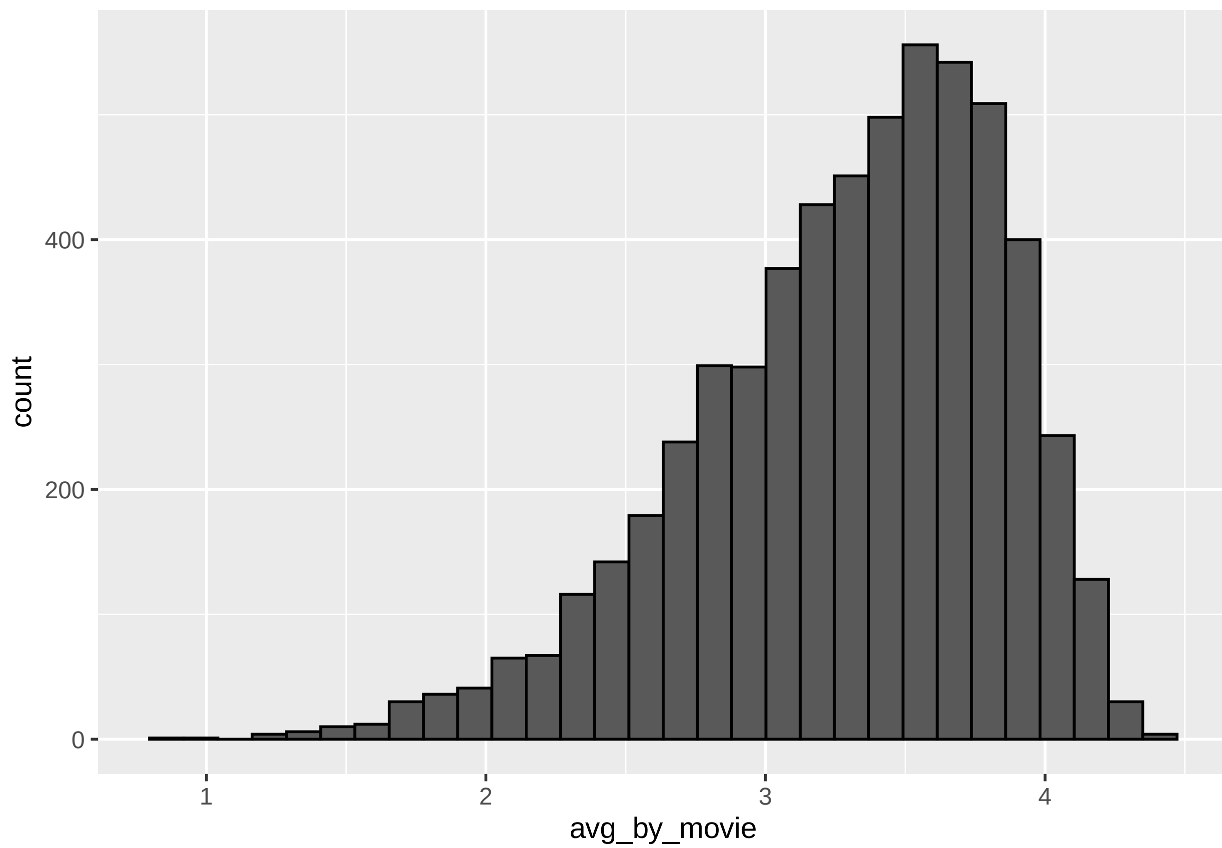
```
edx  %>%
  group_by(userId) %>%  #explore user influence
  filter(n()>=100) %>% #select users with at least 100 ratings
  summarize(avg_by_user = mean(rating)) %>% #calculate average rating
  ggplot(aes(avg_by_user)) +
  geom_histogram(bins = 30, color = "black")
```



From the histogram, we see a lot of spread among users; some are biased high and others low. Thus, we should definitely use user effects to help predict the ratings.

What about individual movies - do some average higher than others? To find out, we do a similar histogram for movie ratings:

```
edx  %>%
  group_by(movieId) %>%  #explore movie influence
  filter(n()>=100) %>% #select movies with at least 100 ratings
  summarize(avg_by_movie = mean(rating)) %>% #calculate average rating
  ggplot(aes(avg_by_movie)) +
  geom_histogram(bins = 30, color = "black")
```



From the histogram, we see a lot of spread among movies; many are biased high but others are very low. The movie bias will be calculated to help predict the ratings.

To analyze the influence of genre, a category will be defined as whatever combination appears in genre (eg, "Adventure|Romance"); we will later use string detection to divide aggregate categories into single genres ("Adventure," "Romance").

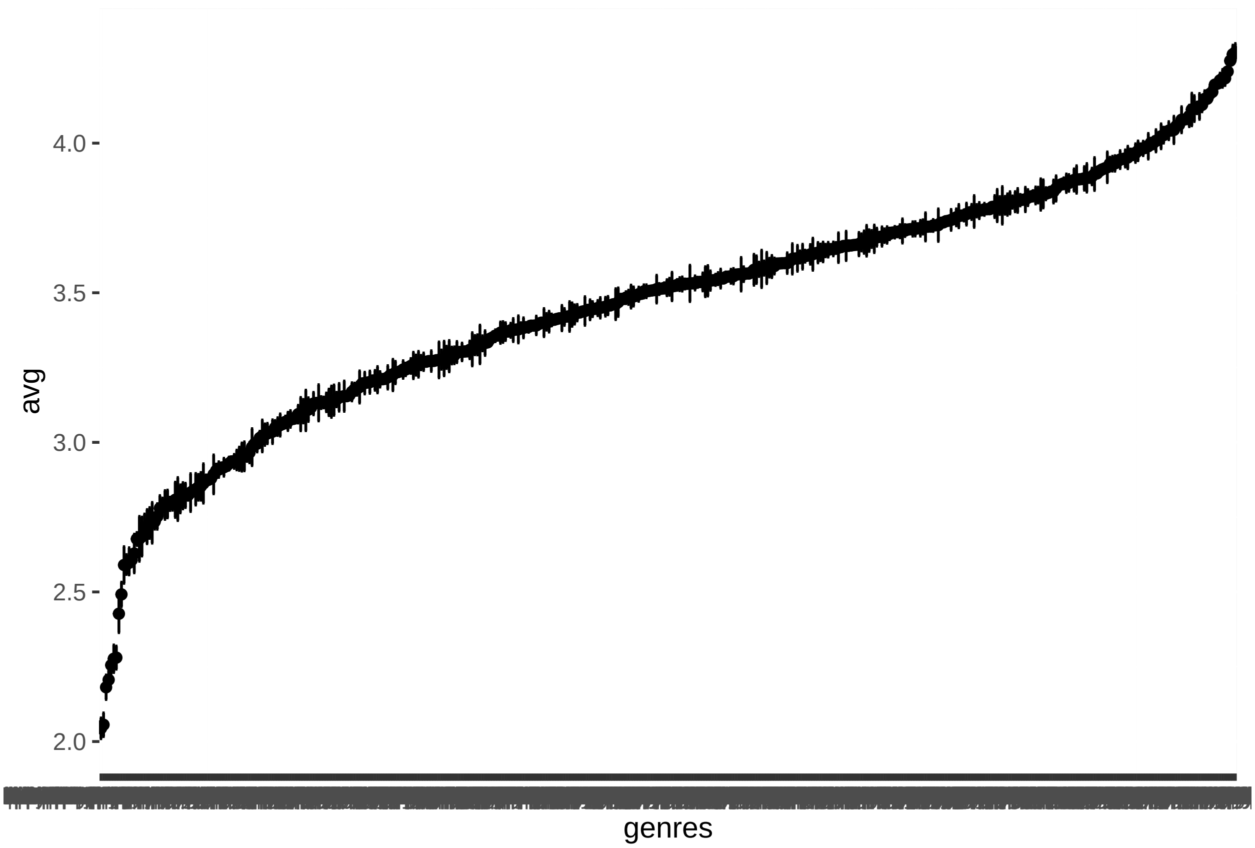
```
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
categories<- edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000)  #use only categories with more than 1,000 ratings.
head(categories)
```

```
## # A tibble: 6 x 4
##   genres                                        n   avg      se
##   <chr>                                     <int> <dbl>   <dbl>
## 1 Action                                    24482  2.94 0.00692
## 2 Action|Adventure                          68688  3.66 0.00409
## 3 Action|Adventure|Animation|Children|Comedy  7467  3.96 0.00901
## 4 Action|Adventure|Animation|Comedy|Drama    1902  3.51 0.0232
## 5 Action|Adventure|Animation|Drama|Fantasy   4333  3.95 0.0143
## 6 Action|Adventure|Animation|Horror|Sci-Fi   4087  3.38 0.0168
```

```r
  #Plot with error bars:
genr <- categories %>% mutate(genres = reorder(genres, avg))
genr %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar()
```



It appears that categories have a strong influence on ratings, since the average per category ranges from 2 to above 4.5. Furthermore, the small error bars show that the differences are highly significant.

Let's try to get a sense of the highest and lowest-rated categories:

```r
#lowest 10
genr %>% group_by(genres) %>% summarize(mu=mean(avg)) %>% top_n(-10,mu)
```

```
## # A tibble: 10 x 2
##    genres                                          mu
##    <fct>                                        <dbl>
##  1 Action|Children                               2.04
##  2 Action|Adventure|Children|Comedy|Fantasy|Sci-Fi  2.06
##  3 Crime|Sci-Fi|Thriller                         2.18
##  4 Action|Adventure|Fantasy|Thriller             2.21
##  5 Action|Adventure|Comedy|Fantasy|Sci-Fi|Western  2.26
##  6 Action|Children|Fantasy                       2.28
##  7 Children|Comedy|Sci-Fi                        2.28
##  8 Action|Comedy|Musical                         2.43
```

```
##   9 Action|Crime|Fantasy                              2.49
## 10 Fantasy|Horror|Thriller                             2.59
```

```
#highest10
genr %>% group_by(genres) %>% summarize(mu=mean(avg)) %>% top_n(10,mu)
```

```
## # A tibble: 10 x 2
##    genres                                    mu
##    <fct>                                  <dbl>
##  1 Crime|Thriller|War                      4.17
##  2 Action|Adventure|Comedy|Fantasy|Romance  4.20
##  3 Crime|Mystery|Thriller                  4.20
##  4 Crime|Film-Noir|Thriller                4.21
##  5 Film-Noir|Romance|Thriller              4.22
##  6 Crime|Film-Noir|Mystery                 4.22
##  7 Film-Noir|Mystery                       4.24
##  8 Animation|Children|Comedy|Crime         4.28
##  9 Action|Crime|Drama|IMAX                 4.30
## 10 Drama|Film-Noir|Romance                 4.30
```

The lowest rated is "Action|Children" at 2.04 and the highest is "Drama|Film-Noir|Romance" at 4.30.

Now let's see if any single genre label, such as "Action," has an effect. To obtain the genre labels, we split the category strings on the "|" character.

```
# Extract all the different label possibilities
vector_of_labels <- unique(flatten(str_split(genr$genres,"\\Q|\\E")))

# Go through categories and see which label has the most ratings
head(vector_of_labels)
```

```
## [[1]]
## [1] "Action"
##
## [[2]]
## [1] "Adventure"
##
## [[3]]
## [1] "Animation"
##
## [[4]]
## [1] "Children"
##
## [[5]]
## [1] "Comedy"
##
## [[6]]
## [1] "Drama"
```

```
length(vector_of_labels)
```

```
## [1] 19
```

```r
# Determine which ones to pick by computing how many ratings appear for each individual genre.
sum_ratings_in_genre<- rep(0,length(vector_of_labels))
ratings_per_genre<- rep(0,length(vector_of_labels))
j=1
for (genre_name in vector_of_labels){ #loop over every label
    i=1
    while(i<=nrow(genr)){ #loop over every category
      # add the ratings for this category to the running total in ratings_per_genre
      n_ratings<- ifelse(str_detect(genr$genres[i],genre_name),genr$n[i],0)
      ratings_per_genre[j] <- ratings_per_genre[j]+n_ratings

      i = i+1
    }
    #track the sum of the ratings so we can get an average later
    sum_ratings_in_genre[j] <- sum_ratings_in_genre[j]+ratings_per_genre[j]*genr$avg[j]
    j=j+1
}
#put results into a data frame
genres_df<- as.data.frame(cbind(vector_of_labels,ratings_per_genre))
genres_df
```

```
##    vector_of_labels ratings_per_genre
## 1            Action           2538262
## 2         Adventure           1882688
## 3         Animation            449766
## 4          Children            720020
## 5            Comedy           3514031
## 6             Drama           3877850
## 7           Fantasy            899693
## 8            Horror            674548
## 9            Sci-Fi           1316549
## 10         Thriller           2302236
## 11            Crime           1312210
## 12          Romance           1692857
## 13              War            500878
## 14          Mystery            551223
## 15          Western            182887
## 16          Musical            422626
## 17        Film-Noir            115093
## 18             IMAX              7386
## 19      Documentary             89943
```

```r
#which ones have over 2 million ratings?
genres_df%>%filter(ratings_per_genre>2000000)
```

```
##   vector_of_labels ratings_per_genre
## 1           Action           2538262
## 2           Comedy           3514031
## 3            Drama           3877850
## 4         Thriller           2302236
```

The most frequently-rated genres are Action, Comedy, Drama, and Thriller, with over 2 million ratings each. For each of these 4, a binary variable, such as *action,* will be set to 1 if the genre contains the label "Action" and 0 otherwise.

```r
# Create four binary variables and assign 0 or 1 to each based on
# whether the category contains that genre (1) or not (0):
genr <- categories %>%
  mutate(action = ifelse(str_detect(genres,"Action")==1,1,0))%>%
  mutate(comedy = ifelse(str_detect(genres,"Comedy")==1,1,0))%>%
  mutate(drama = ifelse(str_detect(genres,"Drama")==1,1,0))%>%
  mutate(thriller = ifelse(str_detect(genres,"Thriller")==1,1,0))
tail(genr)
```

```
## # A tibble: 6 x 8
##   genres                n   avg      se action comedy drama thriller
##   <chr>             <int> <dbl>   <dbl>  <dbl>  <dbl> <dbl>    <dbl>
## 1 Romance|Thriller   1967  3.26 0.0216       0      0     0        1
## 2 Sci-Fi            10125  2.93 0.0109       0      0     0        0
## 3 Sci-Fi|Thriller   40129  3.56 0.00523      0      0     0        1
## 4 Thriller          94662  3.53 0.00314      0      0     0        1
## 5 War                2300  3.67 0.0181       0      0     0        0
## 6 Western           15300  3.54 0.00793      0      0     0        0
```

```r
# compute the mean rating and standard error for each individual genre:
genr%>%filter(action==1) %>%summarize(mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 2
##   `mean(avg)` `sd(avg)/sqrt(n())`
##         <dbl>               <dbl>
## 1        3.33              0.0346
```

```r
genr%>%filter(comedy==1) %>%summarize(mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 2
##   `mean(avg)` `sd(avg)/sqrt(n())`
##         <dbl>               <dbl>
## 1        3.40              0.0344
```

```r
genr%>%filter(drama==1) %>%summarize(mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 2
##   `mean(avg)` `sd(avg)/sqrt(n())`
##         <dbl>               <dbl>
## 1        3.63              0.0213
```

```r
genr%>%filter(thriller==1) %>%summarize(mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 2
##   `mean(avg)` `sd(avg)/sqrt(n())`
##         <dbl>               <dbl>
## 1        3.48              0.0352
```

The genres with the highest rating, Drama (avg rating = 3.63) is significantly different than the others. It seems possible that these individual genre labels, rather than category labels, could be used as predictors.

However, these ratings average over users, and we know that some people have specific genre preferences. For example, one might hate drama and love action, or vice-versa. To be sensitive to that effect, our model will instead implement user-specific genre bias based on the individual genre labels.

Meanwhile, can one use the total number of ratings for a genre as a predictor? Might the genres with the fewest ratings also be the lowest-rated? Let's find out.

```
# what genres have the least frequent ratings?
genres_df%>%filter(ratings_per_genre<200000)
```

```
##   vector_of_labels ratings_per_genre
## 1          Western            182887
## 2        Film-Noir            115093
## 3             IMAX              7386
## 4      Documentary             89943
```

```
# The least-frequently rated genres are Western, Film Noir, IMAX and Documentary.
genr_least_common <- genr %>%
  mutate(western= ifelse(str_detect(genres,"Western")==1,1,0))%>%
  mutate(noir = ifelse(str_detect(genres,"Film-Noir")==1,1,0))%>%
  mutate(imax = ifelse(str_detect(genres,"IMAX")==1,1,0))%>%
  mutate(doc = ifelse(str_detect(genres,"Documentary")==1,1,0))
#Do they also have the lowest ratings?
genr_least_common%>%filter(western==1) %>%summarize(n(), mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 3
##   `n()` `mean(avg)` `sd(avg)/sqrt(n())`
##   <int>       <dbl>               <dbl>
## 1    27        3.56              0.0760
```

```
genr_least_common%>%filter(noir==1) %>%summarize(n(), mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 3
##   `n()` `mean(avg)` `sd(avg)/sqrt(n())`
##   <int>       <dbl>               <dbl>
## 1    22        3.97              0.0451
```

```
genr_least_common%>%filter(imax==1) %>%summarize(n(), mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 3
##   `n()` `mean(avg)` `sd(avg)/sqrt(n())`
##   <int>       <dbl>               <dbl>
## 1     4        3.79               0.171
```

```
genr_least_common%>%filter(doc==1) %>%summarize(n(), mean(avg),sd(avg)/sqrt(n()))
```

```
## # A tibble: 1 x 3
##   `n()` `mean(avg)` `sd(avg)/sqrt(n())`
##   <int>       <dbl>               <dbl>
## 1     6        3.74              0.0298
```

The sample sizes for these least-frequently rated movies is small; there are only 22 film noir movies, for example. However, the average rating is high (3.79 for IMAX, 3.97 for Film-noir!) so it doesn't seem to be the case that the least-watched movies are also low-rated.

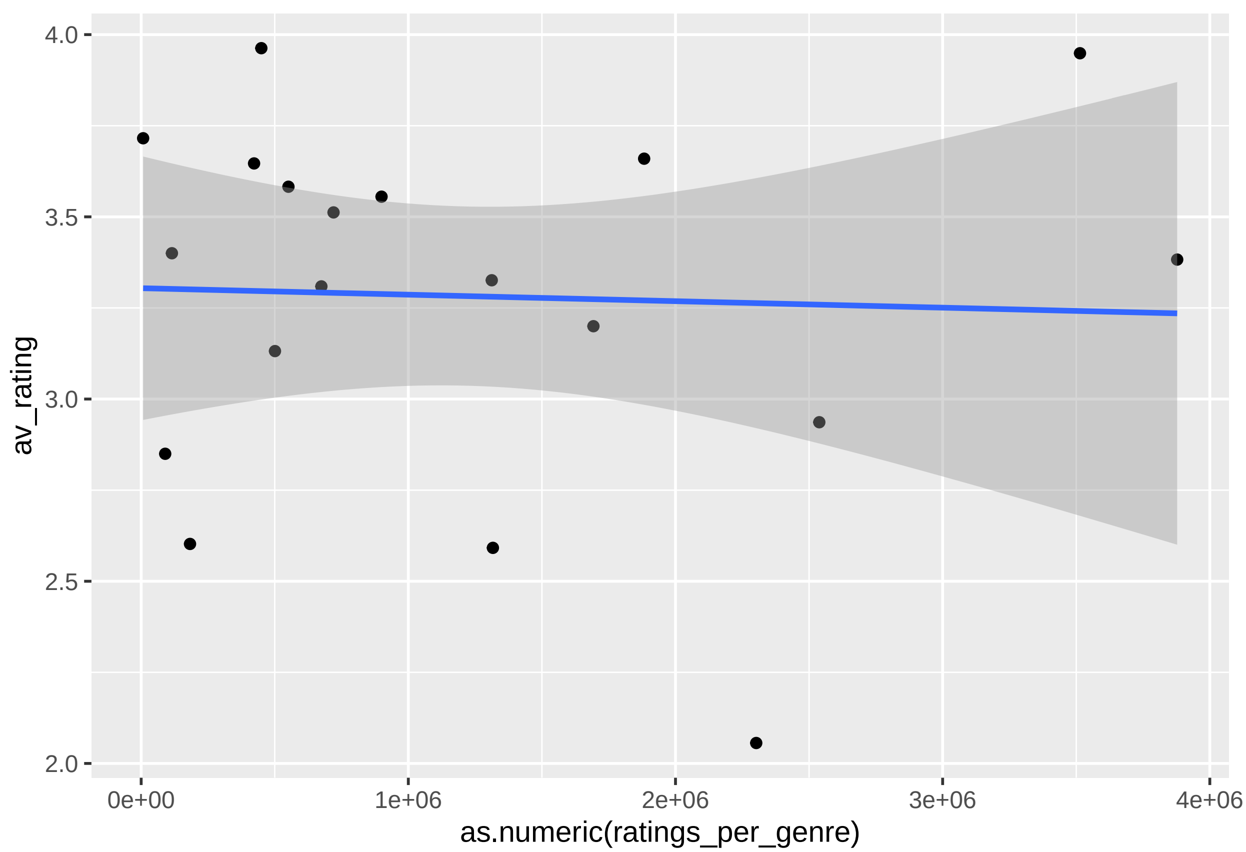
Let's plot the average rating per genre versus the number of ratings for movies in the genre to see if there's a relationship overall:

```
genres_df<- genres_df %>% mutate(av_rating=as.numeric(sum_ratings_in_genre)/as.numeric(ratings_per_genre
head(genres_df)
```

```
##   vector_of_labels ratings_per_genre av_rating
## 1           Action           2538262  2.936321
## 2        Adventure           1882688  3.659569
## 3        Animation            449766  3.962770
## 4         Children            720020  3.512093
## 5           Comedy           3514031  3.948881
## 6            Drama           3877850  3.382677
```

```
genres_df %>%
  ggplot(aes(as.numeric(ratings_per_genre),av_rating)) + geom_point()+geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



From the plot, there is no evidence that the number of ratings in the genre predicts the rating. So we won't use number of ratings in the genre as a predictor.

Sticking with the idea of a user-specific genre effect, we'll assume that the genres with the most influence on accuracy of predictions are going to be those that are the most frequent. Thus, we mutate our dataset to include binary variables for each of the 4 highest-frequency labels. This expanded dataframe will be called *edx_plus*, and it will be used for training and optimization from this point forward.

```
edx_plus <-  edx %>%   mutate(action = ifelse(str_detect(genres,"Action")==1,1,0))%>%
  mutate(comedy = ifelse(str_detect(genres,"Comedy")==1,1,0))%>%
  mutate(drama = ifelse(str_detect(genres,"Drama")==1,1,0))%>%
  mutate(thriller = ifelse(str_detect(genres,"Thriller")==1,1,0))
names(edx_plus)
```

```
##  [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
##  [7] "action"    "comedy"    "drama"     "thriller"
```

From this exploratory analysis, it appears that a useful model will contain an estimate based on the overall average but also adjustments based on the movie, the user, and the user's individual genre preferences.

Step 3: Modeling approaches

In this section, four models are computed. The final model will be the following, as described previously:

$$y_{u,mov} = \bar{y} + b_{mov} + b_u + \sum_{k=1}^{4} x_{mov}^k b_{u,k},$$

This model will be implemented iteratively, in 4 stages. The first stage will use a "default" model; the movie ratings will be estimated as simply the average value of the rating: $y_{u,mov} = \bar{y}$.

The second stage will have the movie bias added ($y_{u,mov} = \bar{y} + b_{mov}$). The third stage will incorporate user effects ($y_{u,mov} = \bar{y} + b_{mov} + b_u$). Finally, user-specific genre effects will complete the final model. This will give us 4 models that we can compare to each other. The RMSE will decrease with each successive stage.

To start, the *edx* set is partitioned into a training set and a test set. The RMSE will be computed using the test set, but averages and regularization parameters will be obtained from the training set.

```
# Step 3: Modeling with the Training Set (edx_plus)
y <- edx_plus$rating #desired outcome to predict

# Start by splitting edx_plus into train/test sets
# The training set is larger by a factor of 3
set.seed(2)
test_index <- createDataPartition(y, times = 1, p = 0.25, list = FALSE)
train_set <- edx_plus %>% slice(-test_index)
test_set <- edx_plus %>% slice(test_index)
dim(train_set)
```

```
## [1] 6750040      10
```

```
dim(test_set)
```

```
## [1] 2250015      10
```

```r
# Estimating the overall average rating: test vs. train
train_average <- train_set %>% summarize(pi=mean(rating)) %>% pull(pi)
train_average
```

```
## [1] 3.512435
```

```r
test_set %>% summarize(pi=mean(rating)) %>% pull(pi)
```

```
## [1] 3.512556
```

```r
# Consistency between test and train
# looks reasonable, as it should for such a large data set.

RMSE = function(predicted, actual){    #define RMSE function
  sqrt(mean((predicted - actual)^2))
}
```

Default Model

Let's create a default value for the RMSE for comparison purposes. The default model will "predict" that all ratings are just equal to the average of the training set.

```r
# use training set average to produce predictions
avg_vector<- rep(train_average,nrow(test_set))
default_model<- RMSE(avg_vector,test_set$rating) #test against the test set
default_model
```

```
## [1] 1.060279
```

As expected, the RMSE using the average is terrible (>1), so we will now test a model with predictors to see if we can do better.

Movie Bias Model

```r
# Bias by movie ID
movie_avgs <- train_set %>% group_by(movieId) %>%
  summarize(b_mov = mean(rating - train_average))

# Calculate RMSE on the test set
# You can only make predictions for movies you have in the
# training set, so we get some NA in the predicted ratings
# that will need to be removed
pred<- test_set %>% left_join(movie_avgs, by='movieId') %>%
  select(movieId,rating,b_mov)

pred_list_mov<- pred[!is.na(b_mov)] %>% #remove the NA's
  mutate(pred_rating=train_average+b_mov) #add movie bias

RMSE(pred_list_mov$pred_rating,pred_list_mov$rating) #test against the test set
```

```
## [1] 0.9440389
```

This result (0.9440389) is better than the default model.

To correct for movies with low numbers of ratings, we introduce regularization with a tuning parameter, lambda.

Five-fold cross validation is used to select the best lambda value. To start, we slice the training set into 5 samples, and choose 1 of the 5 to be a temporary test set. There are 5 choices we can make for the test set, so we create 5 train/test sets by making the choice of the test set different each time. Using all 5 training sets, we average the RMSEs (for each value of lambda) to find the lambda that minimizes the error on the test sets.

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

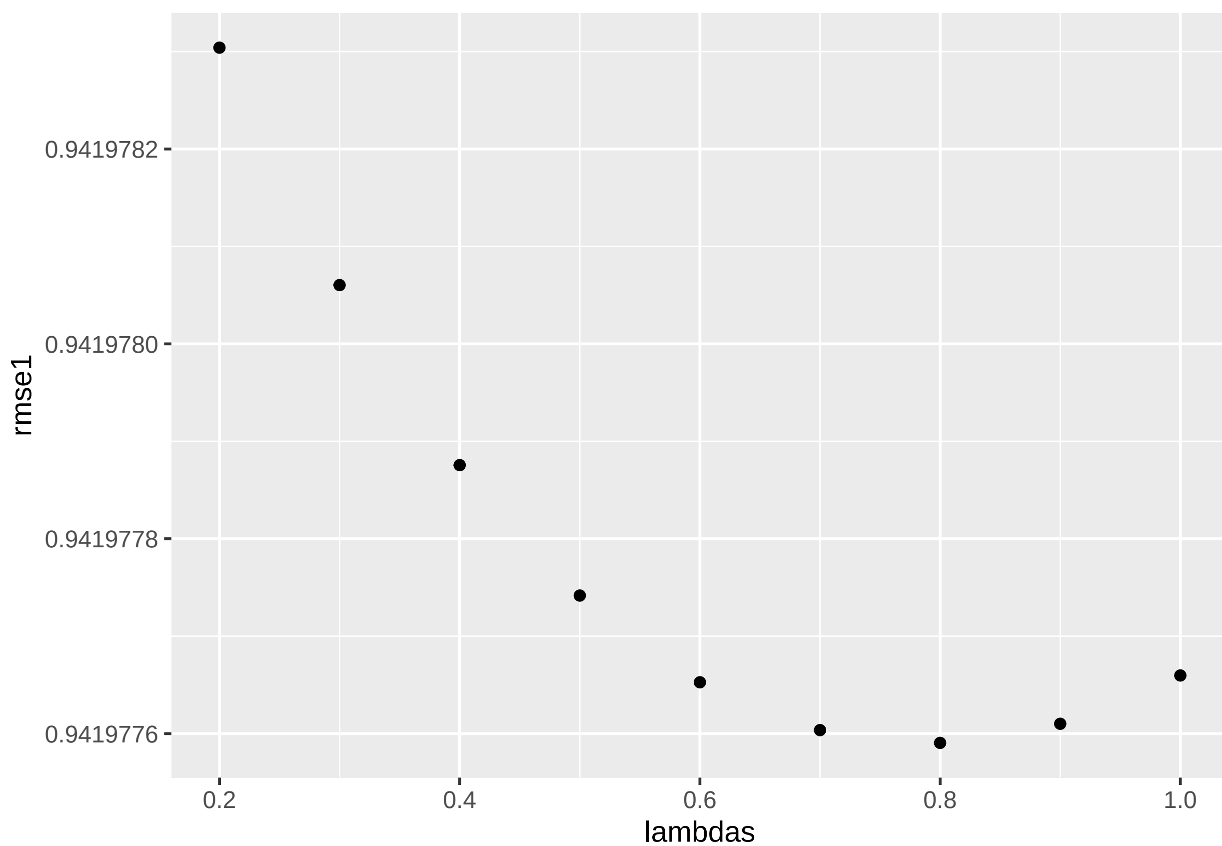
```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
indexes <- createResample(train_set$rating, 5) #create the 5 folds
ind1<- indexes$Resample1
ind2<- indexes$Resample2
ind3<- indexes$Resample3
ind4<- indexes$Resample4
ind5<- indexes$Resample5
# Create 5 test and training sets
train5<-train_set[c(ind1,ind2,ind3,ind4)]
test5<- train_set[ind5]
train1<-train_set[c(ind5,ind2,ind3,ind4)]
test1<- train_set[ind1]
train2<-train_set[c(ind1,ind5,ind3,ind4)]
test2<- train_set[ind2]
train3<-train_set[c(ind5,ind2,ind1,ind4)]
test3<- train_set[ind3]
train4<-train_set[c(ind5,ind2,ind1,ind3)]
test4<- train_set[ind4]

#free up some memory
remove(edx)
gc()
```

```
##              used    (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    2647628   141.4    7788138   416.0   22344112  1193.4
## Vcells 1746615814 13325.7 2196952012 16761.5 1760147554 13428.9
```

```r
# For each training set, we compute the error for each value of lambda.
# Using all 5 training sets, we average the error to find the lambda
# that minimizes the average error across all 5

lambdas <- seq(0.2,1,0.1) # tuning parameter

# Define a function that takes a training and test set and
# outputs a set of RSMEs for several choices of lambda
run_rmse<- function(training,testing){
```

```
sapply(lambdas, function(l){
  just_the_sum <- training %>%
    group_by(movieId) %>%
    summarize(s = sum(rating - train_average), n_i = n())
  predicted_rating <- testing %>% select(rating,movieId)%>%
    inner_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l))%>%  #compute the movie bias
    mutate(pred = train_average + b_i) #apply the bias
  rmse<- RMSE(predicted_rating$pred,predicted_rating$rating)
  return(rmse)
})}

#Apply the function to each test/training set to get 5 sets of output
rmse1<- run_rmse(train1,test1)
qplot(lambdas,rmse1) #check that range for lambda is reasonable
```



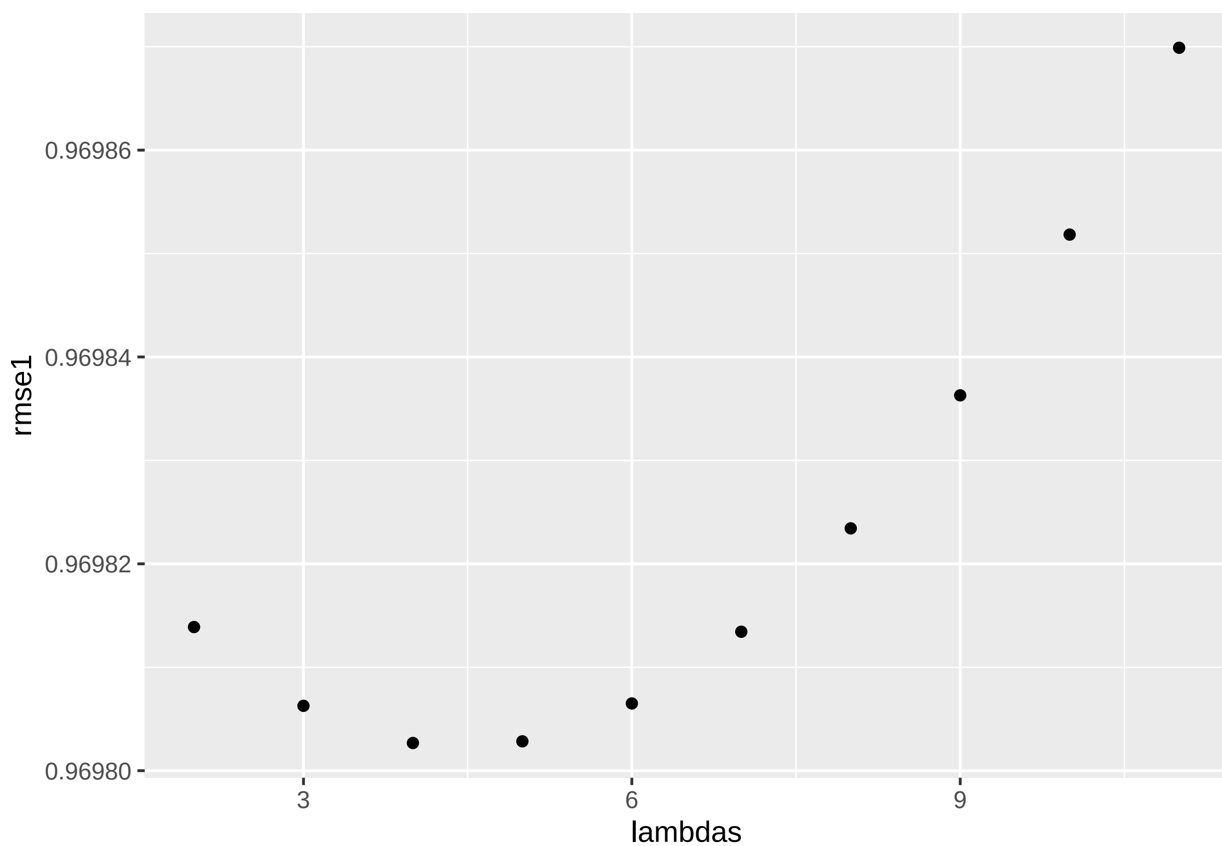

```
rmse2<- run_rmse(train2,test2)
rmse3<- run_rmse(train3,test3)
rmse4<- run_rmse(train4,test4)
rmse5<- run_rmse(train5,test5)

#calculate the average RMSE for each value of lambdas
rmse_list<- as.matrix(rmse1,rmse2,rmse3,rmse4,rmse5)
#  determine which one is the minimal one and select that lambda
l_best <- lambdas[which.min(rowMeans(rmse_list))]
```

```
l_best #<- 0.8
```

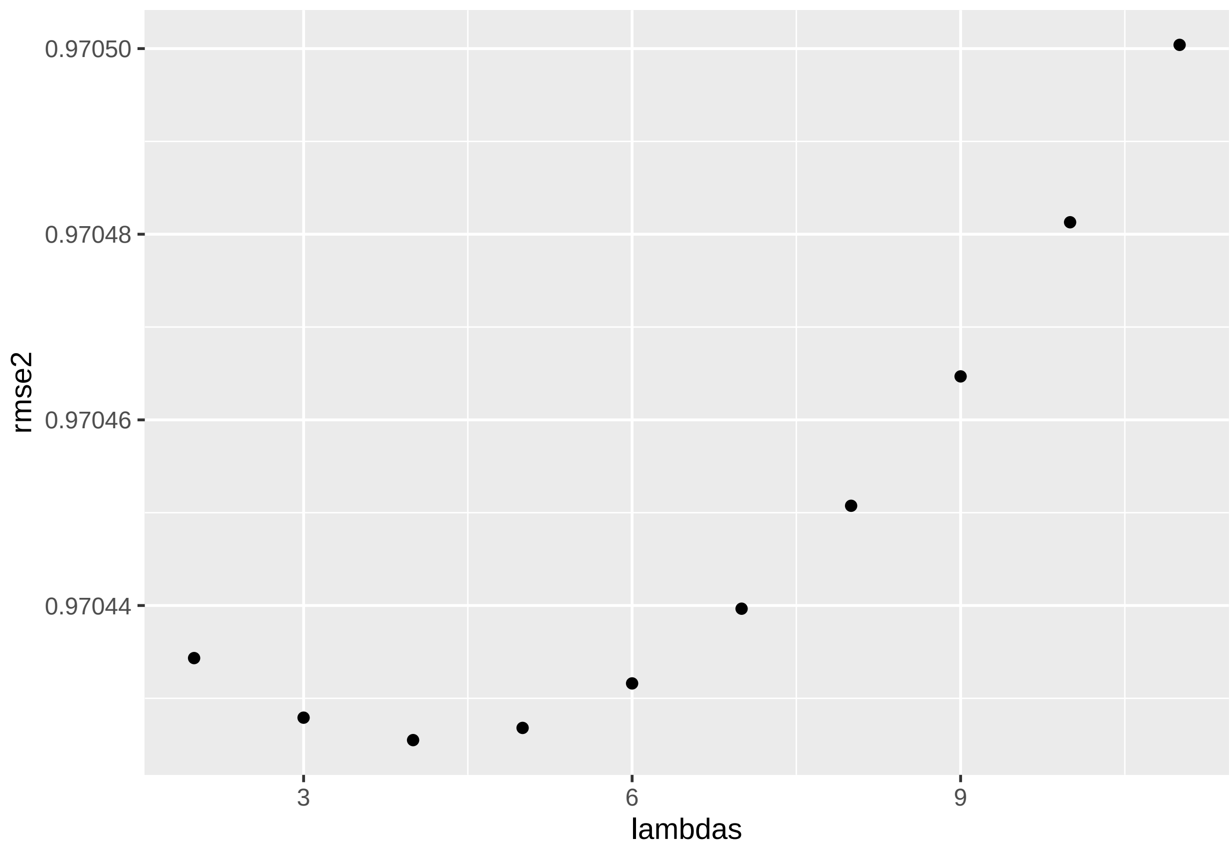
```
## [1] 0.8
```

The same cross-validation process is repeated to get a lambda value for user effects.

```
lambdas <- seq(2,11,1)
run_rmse<- function(training,testing){
  sapply(lambdas, function(l){
    sum_u <- training %>%
      group_by(userId) %>%
      summarize(s_u = sum(rating - train_average), n_u = n())
    predicted_rating <- testing %>% select(rating,userId)%>%
      inner_join(sum_u, by='userId') %>%
      mutate(b_u = s_u/(n_u+l))%>%
      mutate(pred = train_average + b_u)
    rmse<- RMSE(predicted_rating$pred,predicted_rating$rating)
    return(rmse)
  })}
rmse1<- run_rmse(train1,test1)
qplot(lambdas,rmse1) #check that range for lambda is reasonable
```




```
rmse2<- run_rmse(train2,test2)
qplot(lambdas,rmse2) #check that range for lambda is reasonable
```




```
rmse3<- run_rmse(train3,test3)
rmse4<- run_rmse(train4,test4)
rmse5<- run_rmse(train5,test5)

# Calculate the average RMSE for each value of lambdas
rmse_list<- as.matrix(rmse1,rmse2,rmse3,rmse4,rmse5)
rmse_list
```

```
##            [,1]
##  [1,] 0.9698139
##  [2,] 0.9698063
##  [3,] 0.9698027
##  [4,] 0.9698028
##  [5,] 0.9698065
##  [6,] 0.9698134
##  [7,] 0.9698234
##  [8,] 0.9698363
##  [9,] 0.9698518
## [10,] 0.9698699
```

```
# Determine which one is the minimal one and select that lambda
l_user<- lambdas[which.min(rowMeans(rmse_list))]
l_user # l_user = 4 minimizes the RMSE
```

```
## [1] 4
```

```
# We obtain a lambda of 4.
```

Now that the lambdas have been optimized, these regularizations are applied on the test set that wasn't used in the cross-validation process. The entire training set is used to compute the biases, then predictions are made on the test set.

```
sum_mov <- train_set %>%  #movie effects
  group_by(movieId) %>%
  summarize(s_mov = sum(rating - train_average), n_i = n())

sum_u <- train_set %>%   #user effects
  left_join(sum_mov, by='movieId') %>%
  mutate(b_mov = s_mov/(n_i+l_best))%>%
  group_by(userId) %>%
  summarize(s_u = sum(rating - train_average - b_mov), n_u = n())

# Start by modeling with movie effect only on the test set
predicted_rating <- test_set %>% select(rating,movieId)%>%
  left_join(sum_mov, by='movieId') %>%
  mutate(b_mov = s_mov/(n_i+l_best))%>%
  mutate(pred = train_average + b_mov)
predicted_rating<-predicted_rating[!is.na(pred)]

#Compare predicted to actual ratings for the regularized effect of movieId
movie_reg<- RMSE(predicted_rating$pred,predicted_rating$rating)

movie_reg
```

```
## [1] 0.9439934
```

We calculate an RMSE = 0.9439934 for the movie effect model. This is for the test set of edx, not the validation set. We reserve the validation data to use only for final assessment, after we have decided our model is sufficient based on the analysis of the *edx* data.

To reduce the RMSE further, one implements user effects with regularization, using the lambda found above. Predictions are made on the test set of *edx*:

```
# Model with movie effect and user effects
predicted_rating <- test_set %>% select(rating,movieId,userId)%>%
  left_join(sum_mov, by='movieId') %>%
  left_join(sum_u, by='userId') %>%
  mutate(b_mov = s_mov/(n_i+l_best))%>%
  mutate(b_u = s_u/(n_u+l_user))%>%
  mutate(pred = train_average + b_mov+b_u)
predicted_rating<-predicted_rating[!is.na(pred)] #remove NA values

#Compare predicted ratings to actual ratings for the regularized effects
user_movie_reg<- RMSE(predicted_rating$pred,predicted_rating$rating)
user_movie_reg
```

```
## [1] 0.8662249
```

The RMSE has been reduced further to 0.8662249. This is not yet good enough based on our metric. Introducing individual genre bias will improve this.

The bias values are be obtained for the 4 most popular genres. To ensure that the values are user-specific, the ratings are grouped by user before the biases are computed. As before, the biases are computed using the training subset of *edx*.

```
# Start with the user and movie effects (already optimized)
predicted_rating <- train_set %>%
  left_join(sum_mov, by='movieId') %>%
  left_join(sum_u, by='userId') %>%
  mutate(b_mov = s_mov/(n_i+l_best))%>%
  mutate(b_u = s_u/(n_u+l_user))%>%
  select(userId,rating,b_u,b_mov,drama,action,thriller,comedy)

# every user has their own value of bias for each genre

#add in bd variable for drama bias
drama_avgs <- predicted_rating %>% group_by(userId)   %>%
     summarize(bd=mean(drama*(rating - train_average-b_mov-b_u)))

#add in ba variable for action bias
action_avgs <- predicted_rating %>% group_by(userId)   %>%
  summarize(ba=mean(action*(rating - train_average-b_mov-b_u)))

#add in bt variable for thriller bias
thriller_avgs <- predicted_rating %>% group_by(userId)   %>%
  summarize(bt=mean(thriller*(rating - train_average-b_mov-b_u)))

#add in bc variable for comedy bias
comedy_avgs <- predicted_rating %>% group_by(userId)   %>%
  summarize(bc=mean(comedy*(rating - train_average-b_mov-b_u)))
```

```
# Now test on the test set (still part of edx_plus, not the validation set)

# Start with user and movieId effects
predicted_rating_test <- test_set %>%
  select(rating,movieId,userId,drama,action,thriller,comedy)%>%
  left_join(sum_mov, by='movieId') %>%
  left_join(sum_u, by='userId') %>%
  mutate(b_mov = s_mov/(n_i+l_best))%>%
  mutate(b_u = s_u/(n_u+l_user))%>%
  mutate(pred = train_average + b_mov+b_u)%>%
  select(rating,movieId,userId,drama,action,thriller,comedy,pred)
 #this prediction (pred) does not yet have genre effects

#Adjust the predictions for genre biases
#To save memory, I do this sequentially, mutating the prediction each time

adj_rating1 <- predicted_rating_test[!is.na(pred)]%>%
  left_join(drama_avgs, by='userId')%>%
  mutate(pred = pred+bd*drama)  #adds drama bias
head(adj_rating1) #check that bd is specific to the user
```

```
##    rating movieId userId drama action thriller comedy     pred         bd
## 1:      5     122      1     0      0        0      1 4.051824 0.04942056
## 2:      5     185      1     0      1        1      0 4.318464 0.04942056
## 3:      5     329      1     1      1        0      0 4.579037 0.04942056
## 4:      5     355      1     0      0        0      1 3.686610 0.04942056
## 5:      5     420      1     0      1        1      1 3.949734 0.04942056
## 6:      3     539      2     1      0        0      1 3.470012 0.07759699
```

```
adj_rating2 <- adj_rating1  %>%
  select(rating,userId,movieId,comedy,action, pred,thriller)  %>%
  left_join(action_avgs, by='userId') %>%
  mutate(pred = pred+ba*action)  #adds action bias
remove(adj_rating1) #free up memory
gc()
```

```
##              used    (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    2650404   141.6    7788138   416.0   22344112  1193.4
## Vcells 1831596929 13974.0 3163786896 24137.8 2080871973 15875.8
```

```
adj_rating3 <- adj_rating2 %>%
  select(rating,userId,movieId,comedy,pred,thriller) %>%
  left_join(thriller_avgs, by='userId') %>%
  mutate(pred = pred+bt*thriller)%>%  #adds thriller bias
  select(rating,userId,movieId,comedy,pred)

head(adj_rating3)
```

```
##    rating userId movieId comedy     pred
## 1:      5      1     122      1 4.051824
## 2:      5      1     185      0 4.562664
## 3:      5      1     329      0 4.775778
## 4:      5      1     355      1 3.686610
## 5:      5      1     420      1 4.193934
## 6:      3      2     539      1 3.470012
```

```
adjusted_pred<- adj_rating3 %>%
  left_join(comedy_avgs, by='userId') %>%
  mutate(pred = pred+bc*comedy)%>%  #adds comedy bias
  select(rating,userId,movieId,pred)

check_RMSE<- RMSE(adjusted_pred$pred,adjusted_pred$rating)
check_RMSE # We achieve an RMSE of 0.8603188 for the test set
```

```
## [1] 0.8603188
```

Let's do one more simple adjustment by rounding down predictions above 5. We know that none of the actual ratings will exceed 5, so rounding down to 5 will improve accuracy:

```
adjusted_pred<- adjusted_pred %>% mutate(pred=ifelse(pred>5,5,pred))
check_RMSE<- RMSE(adjusted_pred$pred,adjusted_pred$rating)
check_RMSE
```

```
## [1] 0.8601547
```

The new RMSE of 0.8601547 looks promising; it appears that selecting the top 4 genres to use for binary variables is enough to meet the criterion for success. Having made this decision, we are ready to validate our model.

Results

This section presents the modeling results and discusses the model performance. The results are judged on the validation set. We start by mutating the data set to pull out the top 4 genre labels:

```
# Mutate validate data frame to pull out individual genres
# "validate_plus" includes binary variables: action, comedy, drama and thriller
validate_plus <- validation %>%
  mutate(action = ifelse(str_detect(genres,"Action")==1,1,0)) %>%
  mutate(comedy = ifelse(str_detect(genres,"Comedy")==1,1,0)) %>%
  mutate(drama = ifelse(str_detect(genres,"Drama")==1,1,0)) %>%
  mutate(thriller = ifelse(str_detect(genres,"Thriller")==1,1,0))

gc() #clean up memory; this dataframe is big
```

```
##              used    (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    2650310   141.6    7788138   416.0   22344112  1193.4
## Vcells 1853593739 14141.8 3163786896 24137.8 2080871973 15875.8
```

```
# add in user and movie bias
predicted_rating_v <- validate_plus  %>%
  select(rating,movieId,userId,drama,action,comedy,thriller) %>%
  left_join(sum_mov, by='movieId') %>%
  left_join(sum_u, by='userId') %>%
  mutate(b_mov = s_mov/(n_i+l_best)) %>%
  mutate(b_u = s_u/(n_u+l_user)) %>%
  mutate(pred = train_average + b_mov+b_u) %>%
  select(movieId,userId,pred,rating,drama,action,comedy,thriller) %>% na.omit()

# Let's see where we are at with just user and movie effects
user_movie_reg_v<-RMSE(predicted_rating_v$pred,predicted_rating_v$rating)
```

We are at 0.8661695 without any genre effects. Now one adjusts for genre bias. This algorithm uses individual genre preferences of the user (action, thriller, comedy, drama), not the aggregated categories (such has 'Action|Thriller')

```
adj_rating1 <- predicted_rating_v[!is.na(pred)]%>%
  left_join(drama_avgs, by='userId')%>%
  mutate(pred = pred+bd*drama)  #adds drama bias
head(adj_rating1) #check for plausibility
```

```
##    movieId userId     pred rating drama action comedy thriller         bd
## 1:     231      1 4.128710      5     0      0      1        0 0.04942056
## 2:     480      1 4.855608      5     0      1      0        1 0.04942056
```

```
## 3:     586      1 4.243099      5      0       0       1         0 0.04942056
## 4:     151      2 3.453945      3      1       1       0         0 0.07759699
## 5:     858      2 4.335384      2      1       0       0         0 0.07759699
## 6:    1544      2 2.795578      3      0       1       0         1 0.07759699
```

```
adj_rating2 <- adj_rating1  %>%
  select(rating,userId,movieId,comedy,action, pred,thriller)  %>%
  left_join(action_avgs, by='userId') %>%
  mutate(pred = pred+ba*action)  #adds action bias
remove(adj_rating1) #free up memory
gc()
```

```
##               used    (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    2650517   141.6    7788138   416.0   22344112  1193.4
## Vcells 1851723813 14127.6 3163786896 24137.8 2080871973 15875.8
```

```
adj_rating3 <- adj_rating2 %>%
  select(rating,userId,movieId,comedy,pred,thriller) %>%
  left_join(thriller_avgs, by='userId') %>%
  mutate(pred = pred+bt*thriller)%>%  #adds thriller bias
  select(rating,userId,movieId,comedy,pred)

head(adj_rating3)
```

```
##    rating userId movieId comedy     pred
## 1:      5      1     231      1 4.128710
## 2:      5      1     480      0 5.099808
## 3:      5      1     586      1 4.243099
## 4:      3      2     151      0 3.396199
## 5:      2      2     858      0 4.335384
## 6:      3      2    1544      0 2.558556
```

```
adjusted_pred<- adj_rating3 %>%
  left_join(comedy_avgs, by='userId') %>%
  mutate(pred = pred+bc*comedy)%>%  #adds comedy bias
  select(rating,userId,movieId,pred)

adjusted_pred<- adjusted_pred %>% mutate(pred=ifelse(pred>5,5,pred))
final_RMSE<- RMSE(adjusted_pred$pred,adjusted_pred$rating)
final_RMSE
```

```
## [1] 0.860049
```

This is the final RMSE value. For comparison purposes, the default model and movie-only model are also run on the validation set.

```
#re-run the default model on the validation set
avg_vector_v<- rep(train_average,nrow(validation))
default_model_v<- RMSE(avg_vector_v,validation$rating) #test against the test set

# Re-run movie effect on the validation set
```

```
predicted_rating <- validation %>% select(rating,movieId)%>%
  left_join(sum_mov, by='movieId') %>%
  mutate(b_mov = s_mov/(n_i+l_best))%>%
  mutate(pred = train_average + b_mov)
predicted_rating<-predicted_rating[!is.na(pred)]

#Compare predicted to actual ratings for the regularized effect of movieId
movie_reg_v<- RMSE(predicted_rating$pred,predicted_rating$rating)
movie_reg_v
```

```
## [1] 0.9440426
```

Conclusion and Limitations

In this analysis, a model has been developed to successfully predict the rating of a movie if the user, movie, and genre are known. Each additional effect included significantly improved the predictive power, as shown in the following table comparing all 4 models:

1: default model (no bias)

2: movie bias only (with regularization)

3: movie bias and user bias (with regularization)

4: final model: movie & user bias (with regularization) plus genre

```
results <-  as.table(cbind(default_model_v, movie_reg_v, user_movie_reg_v, final_RMSE))
row.names(results)<-  'RMSE'
colnames(results)<- c('Default ',' Movie ', ' User+Movie ', ' Final Model ')
results
```

```
##       Default     Movie   User+Movie   Final Model
## RMSE 1.0612018 0.9440426    0.8661695     0.8600490
```

Slight improvements were gained by regularization of the movie bias and user bias, which corrects for the outsize influence of movies or users with low numbers of ratings. The regularization parameters were optimized using 5-fold cross validation. Very slight improvement is also gained by rounding down predicted ratings above 5.

The predicted values of the RMSE on the various models, when computed with the validation set, agreed closely with the values computed from the *edx* data set, since overtraining was avoided.

The final model was limited to predictions based only on the 4 most popular genres. With 19 possible individual genre labels, it would be possible to create a more detailed model with 19 binary predictors. Furthermore, some information in the original data set, such as timestamp and movie title, were not used to make predictions. For possible future work, it would be interesting to see if movie title could be used to analyze whether sequels are more or less popular than originals. For example, one could see if the appearance of a 2 or a II in the title increases or decreases the rating (eg, "Toy Story" vs. "Toy Story 2").