Movie Recommendation System

Jon Wayland

January 16, 2019

Problem Overview

The Data

The `movielens` dataset is a compilation of movie ratings provided by movielens.org. The data contains the following fields:

- `userId`: ID assigned to the unique user who is rating the film
- `movieId`:ID assigned to the unique film
- `rating`:rating given by the user on the scale of 0.5 to 5.0 in 0.5 intervals
- `timestamp`:the point in time when the rating was given
- `title`:the name of the film, inclusive of the year of the film's release
- `genres`:pipe-separated genre categories

As a part of the **PH125.9x** capstone course offered through Harvard on edX, the data is split into two sets:

1. `edx` set: contains 9,000,055 ratings
2. `validation` set: contains 999,999 ratings

The algorithm in this project is built and tested using the `edx` set, while the `validation` set is submitted for predictive performance assessment and will not be used in this report. For purposes of building and iteratively testing the algorithm, the `edx` set is split into a training and a testing set, which are named `movieTrain` and `movieTest`, respectively.

The Goal

The goal of this project is to achieve as low as possible RMSE (root mean squared error) when predicting film ratings. This project is similar to the Netflix Challenge where $1 Million was awarded to the team who was able to improve the performance of Netflix's existing algorithm by 10.06%. In The BellKor Solution to the Netflix Grand Prize, Yehuda Koren explains the team's methodology in winning the competition. Many of their techniques are leveraged in this project, though through use of various R libraries.

Methods & Analysis

Prior to analysis, the following libraries are loaded as they will be used:

```
library(tidyverse) # Loading the tidyverse suite of packages, including dplyr and ggplot2
library(caret) # Loading caret for data splitting and potential training
library(data.table) # Loading data.table to leverage certain structured data.table functions
library(stringr) # Loading stringr to access string manipulation capabilities
```

Data Pre-Processing

As the goal is to achieve the lowest possible RMSE on the `validation` set, the focus from here on out is minimizing the RMSE on the `movieTest` dataset which will be defined as 10% of the `edx` dataset. Before really digging into the data, We believe that the pipe-separated `genres` field will be an important predictor. Because we want to test this hypothesis, we will first separate this column into flags for each genre.

Since we know that each `movieId` represents a single movie, and that each movie has its own unique combination of genres, we can accomplish this by first grabbing the unique `movieId` and `genres` list:

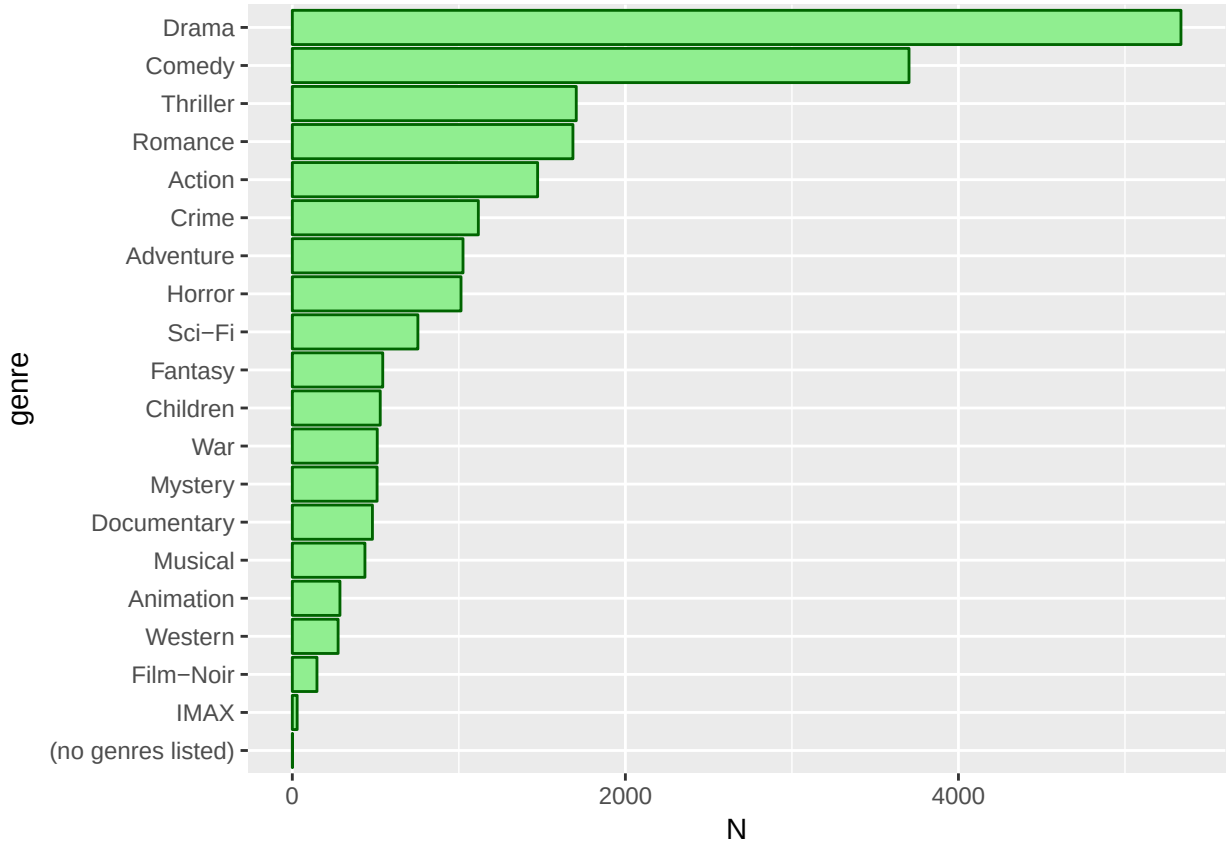
```
genres <- edx %>% select(movieId, genres) %>% unique()
```

Next, we can separate the genres using the `stringr` function `strsplit` and denoting our pipe separator. With `unlist()` we can repetatively match the individual genres with their respective `movieId`, and then re-structure as a `data.frame`.

```
s <- strsplit(genres$genres, split = "\\|")

genres<-data.frame(genre = unlist(s),
                   movieId = rep(genres$movieId, sapply(s, length)))
```

Now we can see how many unique genres that we have and how many films exist within each:

```
genres %>%
  group_by(genre) %>%
  summarize(N=n()) %>%
  mutate(genre = reorder(genre, N))%>%
  ggplot(aes(x=genre,y=N))+
  geom_bar(stat="Identity", fill = "lightgreen", color = "darkgreen")+
  coord_flip()
```



Okay, so we have 20 unique genres, and only a single film that does not have any genres listed (which we will probably ignore). Because we want to spread this `data.frame` out from long to wide, we'll need a value to indicate when the genre is present. We'll also have to fill in the `NA`s where there isn't a genre present with a value after spreading, and so to keep things simple, we will use 1 and 0 respectively:

```
genres$value<-1
genres <- spread(genres, genre, value)
genres[is.na(genres)]<-0
```

Since we used 1 and 0 to indicate the presence of a genre for a particular movieId, we can sum the rows to determine how many genres are associated to the film and store it as a new column named `genreCount`:

```
genres$genreCount <- rowSums(genres[,2:ncol(genres)])
```

Finally, we can rename some of the columns who have hyphens in their names to make them easier to work with:

```
genres <- genres %>% rename(No_Genre_Listed = `(no genres listed)`,
                            Film_Noir = `Film-Noir`,
                            Sci_Fi = `Sci-Fi`)
```

A very imporant note is that all users and movies in the `validation` dataset are also in the `edx` dataset. Given this information, we can join the `genres` dataset to both the `edx` (which is was built off of) and the `validation` sets:

```
edx <- edx %>% inner_join(genres, by = c("movieId" = "movieId"))
validation <- validation %>% inner_join(genres, by = c("movieId" = "movieId"))
# Save space
rm(genres)
```

In addition to genres, we can extract the year the film was released from the `title` field and store as a separate column from each dataset. To do this, we can create a helper function to use `substr` from the right-hand side as opposed to the left:

```
substrRight <- function(x, n){
  substr(x, nchar(x)-n+1, nchar(x))
}
edx$year <- as.integer(gsub(")","",substrRight(edx$title, 5)))
validation$year <- as.integer(gsub(")","",substrRight(validation$title, 5)))
```

With our new fields, we can take a look at what the dataset looks like:

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  28 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 83898
##  $ title          : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ..
##  $ genres         : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Ac
##  $ No_Genre_Listed: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Action         : num  0 1 1 1 1 0 0 0 0 1 ...
##  $ Adventure      : num  0 0 0 1 1 0 0 1 1 0 ...
##  $ Animation      : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Children       : num  0 0 0 0 0 1 0 1 1 0 ...
##  $ Comedy         : num  1 0 0 0 0 1 1 0 0 1 ...
##  $ Crime          : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Documentary    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Drama          : num  0 0 1 0 1 0 1 0 1 0 ...
##  $ Fantasy        : num  0 0 0 0 0 1 0 0 0 0 ...
##  $ Film_Noir      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Horror         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ IMAX           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Musical        : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Mystery        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Romance        : num  1 0 0 0 0 0 1 1 0 0 ...
##  $ Sci_Fi         : num  0 0 1 1 1 0 0 0 0 0 ...
##  $ Thriller       : num  0 1 1 0 0 0 0 0 0 0 ...
##  $ War            : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ Western        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ genreCount     : num  2 3 4 3 4 3 4 3 5 2 ...
##  $ year           : int  1992 1995 1995 1994 1994 1994 1994 1994 1994 1994 ...
```

Splitting into Training and Testing

Using the `caret` library, we can create the `movieTest` and `movieTrain` datasets. To follow the fact that `edx` and `validation` have the same set of users and movies (i.e. there are no new users or movies in `validation`), we force `movieTest` and `movieTrain` to be the same. This will lose a total of 17 ratings in our analysis, but will not effect our final model as it will be built using the entire `edx` set.

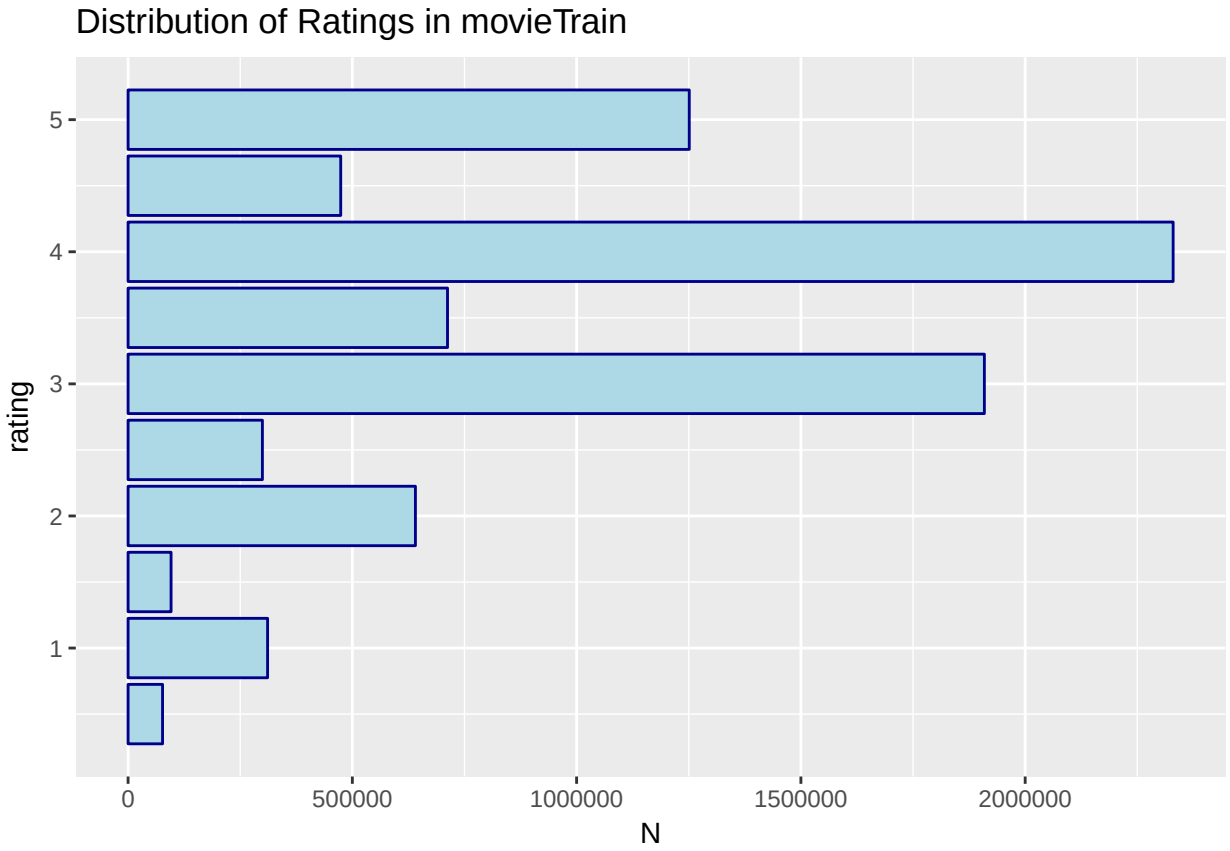
```
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
movieTrain <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in movieTest set are also in movieTrain
movieTest <- temp %>%
  semi_join(movieTrain, by = "movieId") %>%
  semi_join(movieTrain, by = "userId")

# Removing the temp file
rm(temp)

# Checking how many get lost
nrow(edx)-nrow(movieTrain)-nrow(movieTest)
```

```
## [1] 17
```

First Improvement: Using Course Materials

Again, the goal of the project is to achieve the lowest possible RMSE. To start, its a good idea for us to visualize how `rating` is distributed:

```
movieTrain %>%
  group_by(rating) %>%
  summarize(N=n()) %>%
  ggplot(aes(x=rating,y=N))+
  geom_bar(stat="Identity",fill="lightblue",color="darkblue")+
  ggtitle("Distribution of Ratings in movieTrain")+
  coord_flip()
```




```
summary(movieTrain$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

As we can see, the mean rating is 3.512. Since we are predicting on a continuous scale and are attempting to minimize RMSE, we know that the mean is a good starting point. Based on this, we have established our baseline model: `prediction = mu`. If we get worse RMSE than this simple guess, then we know we are in trouble. Likewise, if this were an attempt to predict accuracy, a simple baseline approach would be to have `prediction = 4` as it has the bulk of the volume (see above).

```
mu <- mean(movieTrain$rating)
results<-data.frame(Model = "Baseline",
                    RMSE = RMSE(movieTest$rating, mu))
results %>% knitr::kable()
```

Model	RMSE
Baseline	1.060054

We can surely improve from 1.0601. As per the Netflix Challenge, we can actually adjust from the mean by introducing additional variables. Adjusting according to the effect of the movie, we can center each rating by subtracting the mean. To assess our findings, we will also store all unique movie titles from our original `edx` set.

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

movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()
```

Using this estimate, we can see the top 5 best and worst movies:

```
# Top 5 Best
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:5) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487543
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487543
Shadows of Forgotten Ancestors (1964)	1.487543
Fighting Elegy (Kenka erejii) (1966)	1.487543
Sun Alley (Sonnenallee) (1999)	1.487543

```
# Top 5 Worst
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:5) %>%
  knitr::kable()
```

title	b_i
Besotted (2001)	-3.012457
Hi-Line, The (1999)	-3.012457
Confessions of a Superhero (2007)	-3.012457
War of the Worlds 2: The Next Wave (2008)	-3.012457
SuperBabies: Baby Geniuses 2 (2004)	-2.767776

It seems a bit odd that the top 5 best movies are ones that are not widely popular. Given what we know, we could probably guess that they have such high estimates due to a low volume of ratings. To test this, we can simply check their counts:

```
movieTrain %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:5) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487543	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487543	1
Shadows of Forgotten Ancestors (1964)	1.487543	1
Fighting Elegy (Kenka erejii) (1966)	1.487543	1
Sun Alley (Sonnenallee) (1999)	1.487543	1

So we were right. These films have all had a single rating. Let's see if this is normal by checking the average number of ratings per movie:

```
movieTrain %>%
  group_by(movieId) %>%
  summarize(N = n()) %>%
  summary() %>% knitr::kable()
```

movieId	N
Min. : 1	Min. : 1.0
1st Qu.: 2750	1st Qu.: 28.0
Median : 5428	Median : 110.0
Mean :13064	Mean : 759.8
3rd Qu.: 8703	3rd Qu.: 511.0
Max. :65133	Max. :28168.0

On average, movies have roughly 760 ratings, with a median of 110 ratings. Thus, movies with very few ratings should be penalized. We can store the results of these average movie effect to compare how much of a difference penalizing makes based on the number of ratings the film has (the following step):

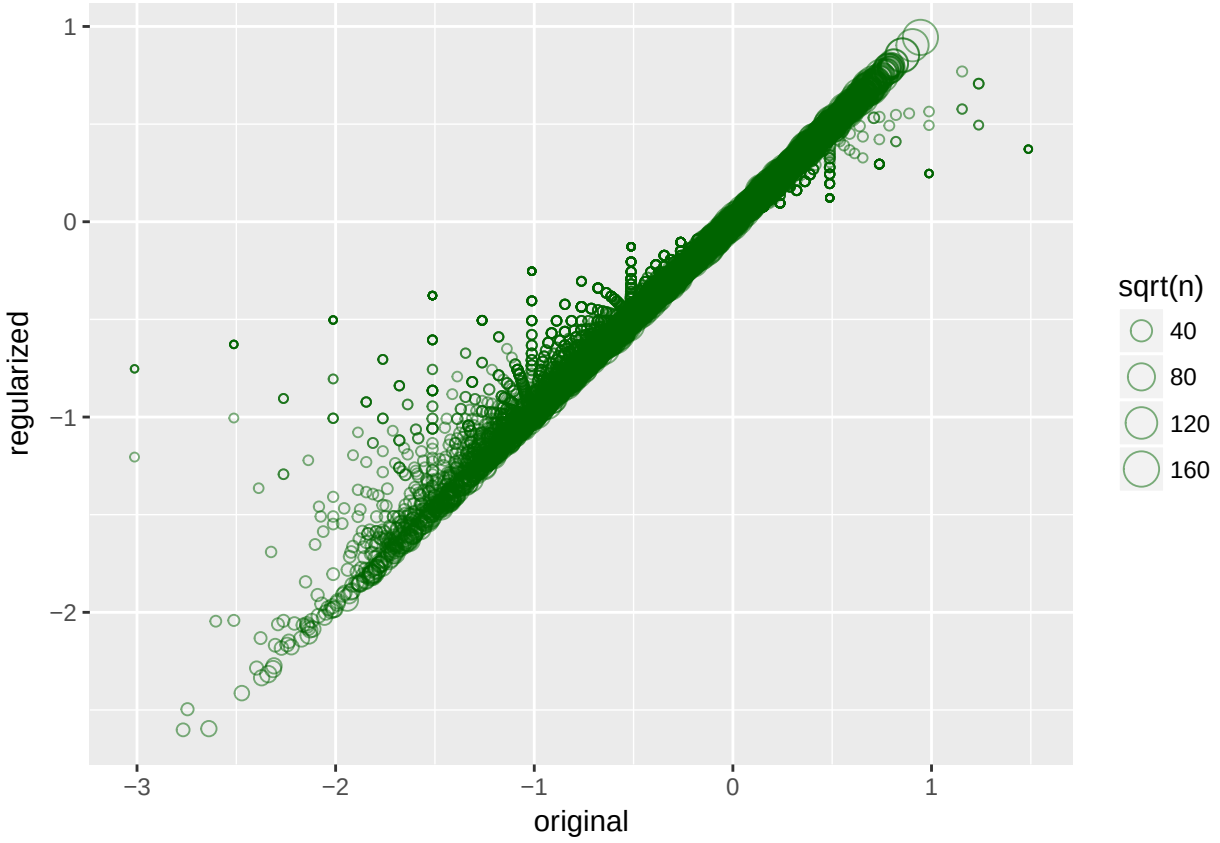
```
predictions <- movieTest %>%
  left_join(movie_avgs, by = 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

results <- rbind(results, data.frame(
  Model = "Movie Effect",
  RMSE = RMSE(movieTest$rating, predictions)
))

results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537

Model	RMSE
Movie Effect	0.9429615

Though this shows an improvement from our baseline, we still want to penalize movies based on the number of ratings they have. In order to do this, we will borrow the Netflix winner's method of regularization in movie and user bias. As professor Rafael Irizarry outlines on his (GitHub)[https://rafalab.github.io/dsbook/regularization.html], we can illustrate how this has an effect by making a plot of the regularized estimates vs. the least squares estimates:

```
lambda <- 3
mu <- mean(movieTrain$rating)
movie_reg_avgs <- movieTrain %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

data_frame(original = movie_avgs$b_i,
           regularized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
    ggplot(aes(original, regularized, size=sqrt(n))) +
        geom_point(shape=1, alpha=0.5, color = "darkgreen")
```




```
# Saving the predictions
predictions <- movieTest %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred
```

```
results <- rbind(results, data.frame(
  Model = "Regularized Movie Effect",
  RMSE = RMSE(movieTest$rating, predictions)
))

results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537
Movie Effect	0.9429615
Regularized Movie Effect	0.9429453

Based on this, there doesn't seem to be much of an improvement. Also, the choice for lambda was arbitrary, and completely based on Professor Irizarry's demonstration. It is a tuning parameter, and one that we will optimize in another step.

For our demonstration, we can see that though there did not seem to be an improvement in RMSE, the penalization had an effect when we view the top 5 best movies according to the new estimate:

```
movieTrain %>%
  count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:5) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

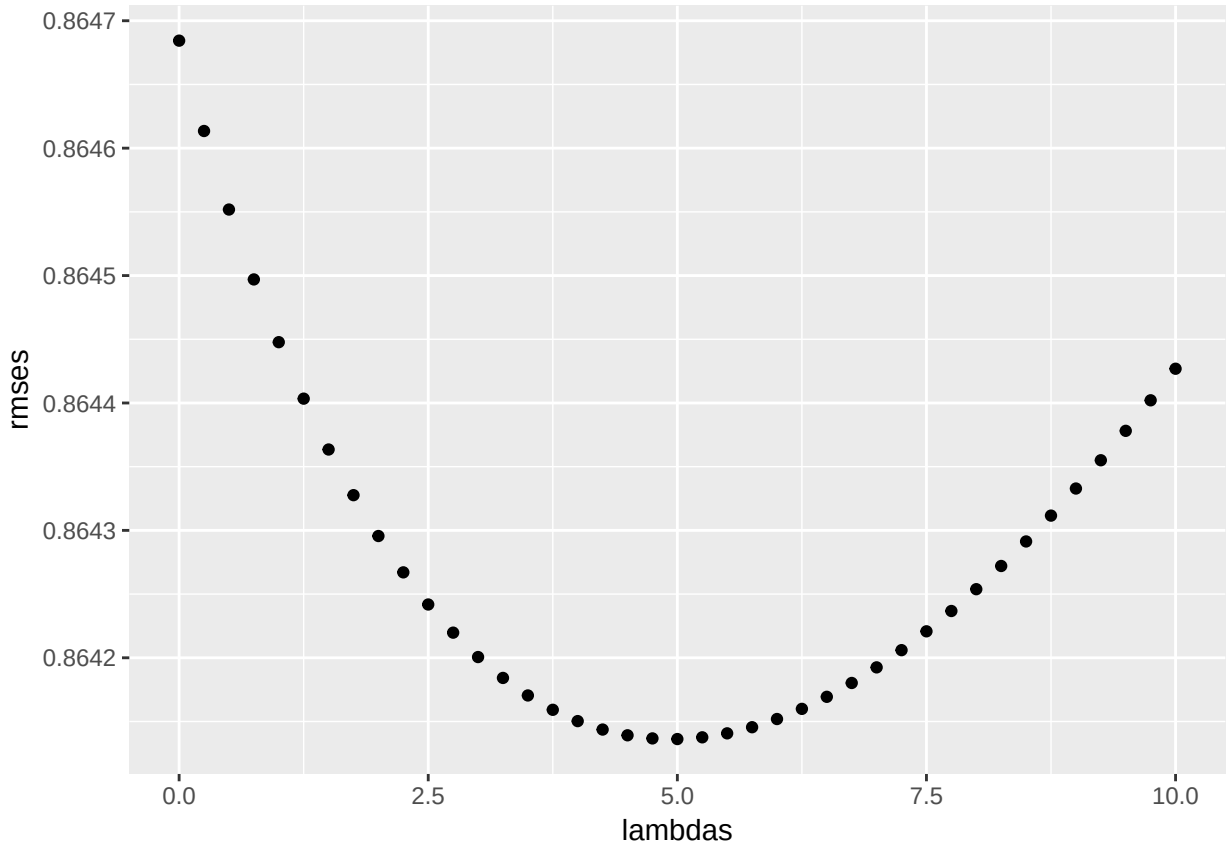
title	b_i	n
Shawshank Redemption, The (1994)	0.9439974	25188
Godfather, The (1972)	0.9040245	15975
Usual Suspects, The (1995)	0.8539634	19457
Schindler's List (1993)	0.8515057	20877
Rear Window (1954)	0.8121481	7115

These are movies that are well-known and now this makes much more sense.

In order to continue following the Netflix method, we can introduce the user bias as well. To do this, we can simply create the movie bias as before and repeat this step for users. The only difference here is that we need to account for movie bias when calculating user bias, thus joining movie bias when performing this step (illustrated below). Also, we can select the optimal lambda based on our metric we are trying to minimize, RMSE:

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(movieTrain$rating)

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

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

  predicted_ratings <-
    movieTest %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, movieTest$rating))
})

qplot(lambdas, rmses)
```




```
# Grabbing the optimal lambda
l<-lambdas[which.min(rmses)]
l
```

```
## [1] 5
```

Now that we have found the optimal lambda, we can incorporate the movie and user effects in our predictions:

```
# movie effect
b_i<-movieTrain %>%
  group_by(movieId) %>%
```

```r
  summarize(b_i = sum(rating-mu)/(n()+l))

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

# Joining back
movieTrain <- movieTrain %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu+b_i+b_u)

movieTest <- movieTest %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu+b_i+b_u)

# RMSE on test set
results<-rbind(results, data.frame(Model = "Regularized Movie and User Effects",
                                   RMSE = RMSE(movieTest$rating, movieTest$pred)))
results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537
Movie Effect	0.9429615
Regularized Movie Effect	0.9429453
Regularized Movie and User Effects	0.8641362

As expected, we see an increase to our RMSE.

Second Improvement: Bringing in Genres

As we hypothesized at the beginning of this report, we wanted to see how the different genres played an influence. Because we expect that some users prefer some genres over others, the goal here is to calculate a weighted score by user for each genre.

In order to accomplish our user-genre matrix, we can leverage some of the code from our pre-processing step and make a few slight modifications:

```
# Assigning the genres dataframe
genres <- movieTrain %>% select(userId, movieId, rating, genres) %>% unique()

# Splitting the genres out based on pipe deliminator
s <- strsplit(genres$genres, split = "\\|")

# Restructuring data frame to include each genre
genres<-data.frame(genre = unlist(s),
                   userId = rep(genres$userId, sapply(s, length)),
                   movieId = rep(genres$movieId, sapply(s, length)),
                   rating = rep(genres$rating, sapply(s, length)))
```

At this point, we will want to optimize our lambda as we did before. Because our genres aren't formatted as nicely as `userId` and `movieId`, this will involve a few additional steps:

1. Create new columns for effects by genre
2. Multiply to our existing binary (though stored as numeric) genre columns
3. Sum all of the genre effects and then divide by the `genreCount` field, i.e. taking the mean effect

The new formula will be require taking the genre effect into account while deriving user bias. This will involve dropping the pre-existing predictions, inclusive of user and movie bias fields:

```
movieTrain$b_i<-NULL
movieTrain$b_u<-NULL
movieTrain$pred<-NULL
movieTest$b_i<-NULL
movieTest$b_u<-NULL
movieTest$pred<-NULL
```

We can find our optimal lambda by running the following code. Due to computing constraints, the output will be supressed, however the optimal metric has been previously derived.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(movieTrain$rating)

  userPref <- genres %>%
    group_by(userId, genre) %>%
    summarize(b_g = sum(rating-mu)/(n()+l))

  # Fixing names to be matchable
  userPref <- userPref %>% mutate(genre = case_when(genre == '(no genres listed)' ~ 'No_Genre_Listed',
                                                    genre == 'Film-Noir' ~ 'Film_Noir',
                                                    genre == 'Sci-Fi' ~ 'Sci_Fi',
                                                    TRUE ~ as.character(genre)))

  # Spreading the user preferences across by genre
  userPref <- spread(userPref, genre, value = b_g)
  # If the user never rated any then there is no effect
```

```
userPref[is.na(userPref)]<-0
# Adding a suffix to the column names
colnames(userPref) <- paste0(colnames(userPref), "_Effect")
# Ignoring the (no genres listed) instance
userPref$No_Genre_Listed_Effect<-NULL

b_g_train <-  movieTrain %>%
  inner_join(userPref, by = c('userId' = 'userId_Effect')) %>%
  mutate(
    Action_Effect = Action_Effect*Action,
    Adventure_Effect = Adventure_Effect*Adventure,
    Animation_Effect = Animation_Effect*Animation,
    Children_Effect = Children_Effect*Children,
    Comedy_Effect = Comedy_Effect*Comedy,
    Crime_Effect = Crime_Effect*Crime,
    Documentary_Effect = Documentary_Effect*Documentary,
    Drama_Effect = Drama_Effect*Drama,
    Fantasy_Effect = Fantasy_Effect*Fantasy,
    Film_Noir_Effect = Film_Noir_Effect*Film_Noir,
    Horror_Effect = Horror_Effect*Horror,
    IMAX_Effect = IMAX_Effect*IMAX,
    Musical_Effect = Musical_Effect*Musical,
    Mystery_Effect = Mystery_Effect*Mystery,
    Romance_Effect = Romance_Effect*Romance,
    Sci_Fi_Effect = Sci_Fi_Effect*Sci_Fi,
    Thriller_Effect = Thriller_Effect*Thriller,
    War_Effect = War_Effect*War,
    Western_Effect = Western_Effect*Western) %>%
  mutate(b_g = Action_Effect + Adventure_Effect + Animation_Effect + Children_Effect +
           Comedy_Effect + Crime_Effect + Documentary_Effect + Drama_Effect + Fantasy_Effect +
           Film_Noir_Effect + Horror_Effect + IMAX_Effect + Musical_Effect + Mystery_Effect +
           Romance_Effect + Sci_Fi_Effect + Thriller_Effect + War_Effect + Western_Effect) %>%
  select(-c(Action_Effect,Adventure_Effect,Animation_Effect,Children_Effect,
            Comedy_Effect,Crime_Effect,Documentary_Effect,Drama_Effect,Fantasy_Effect,
            Film_Noir_Effect,Horror_Effect,IMAX_Effect,Musical_Effect,Mystery_Effect,
            Romance_Effect,Sci_Fi_Effect,Thriller_Effect,War_Effect,Western_Effect)) %>%
  mutate(b_g = b_g/genreCount) %>% select(userId, movieId, b_g)

b_g_test <- movieTest %>%
  inner_join(userPref, by = c('userId' = 'userId_Effect')) %>%
  mutate(
    Action_Effect = Action_Effect*Action,
    Adventure_Effect = Adventure_Effect*Adventure,
    Animation_Effect = Animation_Effect*Animation,
    Children_Effect = Children_Effect*Children,
    Comedy_Effect = Comedy_Effect*Comedy,
    Crime_Effect = Crime_Effect*Crime,
    Documentary_Effect = Documentary_Effect*Documentary,
    Drama_Effect = Drama_Effect*Drama,
    Fantasy_Effect = Fantasy_Effect*Fantasy,
    Film_Noir_Effect = Film_Noir_Effect*Film_Noir,
    Horror_Effect = Horror_Effect*Horror,
    IMAX_Effect = IMAX_Effect*IMAX,
```

```
      Musical_Effect = Musical_Effect*Musical,
      Mystery_Effect = Mystery_Effect*Mystery,
      Romance_Effect = Romance_Effect*Romance,
      Sci_Fi_Effect = Sci_Fi_Effect*Sci_Fi,
      Thriller_Effect = Thriller_Effect*Thriller,
      War_Effect = War_Effect*War,
      Western_Effect = Western_Effect*Western) %>%
    mutate(b_g = Action_Effect + Adventure_Effect + Animation_Effect + Children_Effect +
             Comedy_Effect + Crime_Effect + Documentary_Effect + Drama_Effect + Fantasy_Effect +
             Film_Noir_Effect + Horror_Effect + IMAX_Effect + Musical_Effect + Mystery_Effect +
             Romance_Effect + Sci_Fi_Effect + Thriller_Effect + War_Effect + Western_Effect) %>%
    select(-c(Action_Effect,Adventure_Effect,Animation_Effect,Children_Effect,
              Comedy_Effect,Crime_Effect,Documentary_Effect,Drama_Effect,Fantasy_Effect,
              Film_Noir_Effect,Horror_Effect,IMAX_Effect,Musical_Effect,Mystery_Effect,
              Romance_Effect,Sci_Fi_Effect,Thriller_Effect,War_Effect,Western_Effect)) %>%
    mutate(b_g = b_g/genreCount) %>% select(userId, movieId, b_g)


  # movie effect
  b_i<-movieTrain %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating-mu)/(n()+l))

  # user effect
  b_u <-movieTrain %>%
    left_join(b_g_train, by = c('userId', 'movieId')) %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating-b_g-b_i-mu)/(n()+l))

  # predictions
  predicted_ratings <- movieTest %>%
    left_join(b_g_test, by = c('userId' = 'userId', 'movieId' = 'movieId')) %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu+b_i+b_u+b_g) %>%
    .$pred

  return(RMSE(predicted_ratings, movieTest$rating))
})

qplot(lambdas, rmses)
```

This yields an optimal lambda of 4.25. To implement, we can follow the same method as before:

```
mu <- mean(movieTrain$rating)
l<-4.25

# Creating the userPref dataframe
userPref <- genres %>%
  group_by(userId, genre) %>%
  summarize(b_g = sum(rating-mu)/(n()+l))

# Clearing space
```

```r
rm(genres)
rm(s)

# Renaming genres to match with previously built columns
userPref <- userPref %>% mutate(genre = case_when(genre == '(no genres listed)' ~ 'No_Genre_Listed',
                                                  genre == 'Film-Noir' ~ 'Film_Noir',
                                                  genre == 'Sci-Fi' ~ 'Sci_Fi',
                                                  TRUE ~ as.character(genre)))
# Spreading the user preferences across by genre
userPref <- spread(userPref, genre, value = b_g)
# If the user never rated any then there is no effect
userPref[is.na(userPref)]<-0
# Adding a suffix to the column names
colnames(userPref) <- paste0(colnames(userPref), "_Effect")
# Ignoring the (no genres listed) instance
userPref$No_Genre_Listed_Effect<-NULL


movieTrain <- movieTrain %>%
  inner_join(userPref, by = c('userId' = 'userId_Effect')) %>%
  mutate(
    Action_Effect = Action_Effect*Action,
    Adventure_Effect = Adventure_Effect*Adventure,
    Animation_Effect = Animation_Effect*Animation,
    Children_Effect = Children_Effect*Children,
    Comedy_Effect = Comedy_Effect*Comedy,
    Crime_Effect = Crime_Effect*Crime,
    Documentary_Effect = Documentary_Effect*Documentary,
    Drama_Effect = Drama_Effect*Drama,
    Fantasy_Effect = Fantasy_Effect*Fantasy,
    Film_Noir_Effect = Film_Noir_Effect*Film_Noir,
    Horror_Effect = Horror_Effect*Horror,
    IMAX_Effect = IMAX_Effect*IMAX,
    Musical_Effect = Musical_Effect*Musical,
    Mystery_Effect = Mystery_Effect*Mystery,
    Romance_Effect = Romance_Effect*Romance,
    Sci_Fi_Effect = Sci_Fi_Effect*Sci_Fi,
    Thriller_Effect = Thriller_Effect*Thriller,
    War_Effect = War_Effect*War,
    Western_Effect = Western_Effect*Western) %>%
  mutate(b_g = Action_Effect + Adventure_Effect + Animation_Effect + Children_Effect +
           Comedy_Effect + Crime_Effect + Documentary_Effect + Drama_Effect + Fantasy_Effect +
           Film_Noir_Effect + Horror_Effect + IMAX_Effect + Musical_Effect + Mystery_Effect +
           Romance_Effect + Sci_Fi_Effect + Thriller_Effect + War_Effect + Western_Effect) %>%
  select(-c(Action_Effect,Adventure_Effect,Animation_Effect,Children_Effect,
            Comedy_Effect,Crime_Effect,Documentary_Effect,Drama_Effect,Fantasy_Effect,
            Film_Noir_Effect,Horror_Effect,IMAX_Effect,Musical_Effect,Mystery_Effect,
            Romance_Effect,Sci_Fi_Effect,Thriller_Effect,War_Effect,Western_Effect))

movieTest <- movieTest %>%
  inner_join(userPref, by = c('userId' = 'userId_Effect')) %>%
  mutate(
    Action_Effect = Action_Effect*Action,
```

```
    Adventure_Effect = Adventure_Effect*Adventure,
    Animation_Effect = Animation_Effect*Animation,
    Children_Effect = Children_Effect*Children,
    Comedy_Effect = Comedy_Effect*Comedy,
    Crime_Effect = Crime_Effect*Crime,
    Documentary_Effect = Documentary_Effect*Documentary,
    Drama_Effect = Drama_Effect*Drama,
    Fantasy_Effect = Fantasy_Effect*Fantasy,
    Film_Noir_Effect = Film_Noir_Effect*Film_Noir,
    Horror_Effect = Horror_Effect*Horror,
    IMAX_Effect = IMAX_Effect*IMAX,
    Musical_Effect = Musical_Effect*Musical,
    Mystery_Effect = Mystery_Effect*Mystery,
    Romance_Effect = Romance_Effect*Romance,
    Sci_Fi_Effect = Sci_Fi_Effect*Sci_Fi,
    Thriller_Effect = Thriller_Effect*Thriller,
    War_Effect = War_Effect*War,
    Western_Effect = Western_Effect*Western) %>%
  mutate(b_g = Action_Effect + Adventure_Effect + Animation_Effect + Children_Effect +
           Comedy_Effect + Crime_Effect + Documentary_Effect + Drama_Effect + Fantasy_Effect +
           Film_Noir_Effect + Horror_Effect + IMAX_Effect + Musical_Effect + Mystery_Effect +
           Romance_Effect + Sci_Fi_Effect + Thriller_Effect + War_Effect + Western_Effect) %>%
  select(-c(Action_Effect,Adventure_Effect,Animation_Effect,Children_Effect,
            Comedy_Effect,Crime_Effect,Documentary_Effect,Drama_Effect,Fantasy_Effect,
            Film_Noir_Effect,Horror_Effect,IMAX_Effect,Musical_Effect,Mystery_Effect,
            Romance_Effect,Sci_Fi_Effect,Thriller_Effect,War_Effect,Western_Effect))

movieTrain$b_g<-movieTrain$b_g/movieTrain$genreCount
movieTest$b_g<-movieTest$b_g/movieTest$genreCount

# movie effect
b_i<-movieTrain %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating-mu)/(n()+1))

# user effect
b_u <- movieTrain %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating-b_g-b_i-mu)/(n()+1))

# Joining back
movieTrain <- movieTrain %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu+b_i+b_u+b_g)

movieTest <- movieTest %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu+b_i+b_u+b_g)
```

After having applied this new prediction, we can see whether we have found an improvement:

```
results<-rbind(results, data.frame(Model = "Regularized Movie, User and Genre Effects",
                                   RMSE = RMSE(movieTest$rating, movieTest$pred)))
results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537
Movie Effect	0.9429615
Regularized Movie Effect	0.9429453
Regularized Movie and User Effects	0.8641362
Regularized Movie, User and Genre Effects	0.8517658

Third Improvement: Matrix Factorization

Doing matrix factorization has proven to be tricky for local stations that do not have sufficient RAM. After some digging, the `recosystem` package in R tends to provide quick and reasonable results. It does, however, have the same shortcoming that manually performing matrix factorization has shown to have when integrating it with our existing model: insufficient RAM. Thus, introducing the whole rating matrix to our current formula yields an error for computing power. Luckily, `recosystem` does have the capability to predict based on the user-movie matrix. Doing this then has the sacrifice of abandoning previously used techniques.

```
library(recosystem)

train_data <- data_memory(user_index = movieTrain$userId, item_index = movieTrain$movieId,
                          rating = movieTrain$rating, index1 = T)
test_data <- data_memory(user_index = movieTest$userId, item_index = movieTest$movieId,
                         index1 = T)

recommender <- Reco()
recommender$train(train_data, opts = c(dim = 20, costp_l2 = 0.1, costq_l2 = 0.1,
                                       lrate = 0.1, niter = 500, nthread = 6, verbose = F))

# Making predictions
prediction <- recommender$predict(test_data, out_memory())

results<-rbind(results, data.frame(Model = 'Matrix Factorization via Recosystem (random parameters)',
                                   RMSE = RMSE(prediction,movieTest$rating)))
results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537
Movie Effect	0.9429615
Regularized Movie Effect	0.9429453
Regularized Movie and User Effects	0.8641362
Regularized Movie, User and Genre Effects	0.8517658
Matrix Factorization via Recosystem (random parameters)	0.8095985

As can be seen, there has been a significant improvement to the RMSE. The ultimate desire, however, is to extract the `p` and `q` matrices from this method and apply them to our existing model that includes user, movie, and genre bias. Unfortunately, this step requires excessive RAM and is unable to be tested:

```
P_file = out_file(tempfile())
Q_file = out_file(tempfile())
P<-recommender$output(out_memory(P_file), out_nothing())
Q<-recommender$output(out_nothing(), out_memory(Q_file))
P<-P$P
Q<-Q$Q
r <- jitter(P %*% t(Q), factor=1)
```

Additionally, the `recosystem` package has the ability to tune it's parameters. As the following takes over 24 hours to train, it will not be used in this demonstration:

```
# Tuning to get the optimal parameters
opts = recommender$tune(train_data, opts = list(dim = c(5, 10, 20, 30), lrate = c(0.1, 0.2),
                                                costp_l1 = seq(0,.5,.1), costq_l1 = seq(0,.5,.1),
                                                nthread = 1, niter = 10))
```

```
recommender$train(train_data, opts = c(opts$min, nthread = 1, niter = 100))
```

After about 24 hours of tuning, we have our optimal parameters:

```
## > opts$min
## $dim
## [1] 30

## $costp_l1
## [1] 0

## $costp_l2
## [1] 0.01

## $costq_l1
## [1] 0

## $costq_l2
## [1] 0.1

## $lrate
## [1] 0.1

## $loss_fun
## [1] 0.7969552
```

These optimal parameters suggest that we may be getting a RMSE of 0.7969. To test this, we can simply plug these in and re-run our matrix factorization:

```
train_data <- data_memory(user_index = movieTrain$userId, item_index = movieTrain$movieId,
                          rating = movieTrain$rating, index1 = T)
test_data <- data_memory(user_index = movieTest$userId, item_index = movieTest$movieId,
                         index1 = T)


recommender <- Reco()
recommender$train(train_data, opts = c(dim = 30,
                                       costp_l1 = 0.0, costp_l2 = 0.01,
                                       costq_l1 = 0.0, ostq_l2 = 0.1,
                                       lrate = 0.1, niter = 500, nthread = 6, verbose = F))

# Making predictions
prediction <- recommender$predict(test_data, out_memory())


results<-rbind(results, data.frame(Model = 'Matrix Factorization via Recosystem (optimal parameters)',
                                   RMSE = RMSE(prediction,movieTest$rating)))
results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537
Movie Effect	0.9429615
Regularized Movie Effect	0.9429453
Regularized Movie and User Effects	0.8641362
Regularized Movie, User and Genre Effects	0.8517658

Model	RMSE
Matrix Factorization via Recosystem (random parameters)	0.8095985
Matrix Factorization via Recosystem (optimal parameters)	0.7916443

Clearly, optimizing the parameters for `recosystem` has yielded the best RMSE.

Discussion on Other Improvement Attempts

As this is a prediction problem, other methods of machine learning were experimented with. Aside from what was used above, none of these were used in the project as they either took too long to train or did not show significant improvements. Other methods used included:

- Naive Bayes from the `e1071` package
- Gradient Boosting from the `gbm` package
- Random Forest from the `randomForest` package
- Least Squares from Base R's `lm()`

When testing these methods, all variables were considered including the genre flags, timestamp, year the film was released, and the predictions from both methods (user, movie, & genre effects and matrix factorization).

Methods such as Naive Bayes and Least Squares showed to have the least improvement (if any) to the model, while Random Forest and Gradient Boosting had the largest improvements. In particular, the Random Forest method had the most significant improvement when training multiple batches and combining into a single majority-vote ensemble of forests. Using this method, observed RMSEs were as low as 0.784. Because the training time exceeds the suggested amount for peer review, they are not used in the final model's prediction.

Results

As we can see, the results have varied from method to method. Ultimately, the more complex the method, the better the result. Revisiting, we can see this when observing our RMSEs on the `movieTest` set:

```
results %>% knitr::kable()
```

Model	RMSE
Baseline	1.0600537
Movie Effect	0.9429615
Regularized Movie Effect	0.9429453
Regularized Movie and User Effects	0.8641362
Regularized Movie, User and Genre Effects	0.8517658
Matrix Factorization via Recosystem (random parameters)	0.8095985
Matrix Factorization via Recosystem (optimal parameters)	0.7916443

Variable Impacts

The introduction of the following effects have yielded good results:

- Movie Effects
- Genre Effects
- User Effects

Though results from the random forest models were not used in the final model, it was found that time stamp of rating and the year of the film's release were important predictors. It would not be surprising to see that including a time effect would have a positive impact on the model, especially when determining the amount of time that has elapsed since the film's release and the time it was rated (i.e. the age of the movie).

Conclusion

In conclusion, predicting movie ratings is a difficult yet interesting problem. Combining methods have repeatedly shown to give the best performance. In particular, including regularization in user, movie, and genre effects have dramatically decreased our RMSE from the baseline model of using the mean.

Using matrix factorization has also showed to have a significant impact. Combining the movie, user and genre effects with matrix factorization on the user-movie matrix would undoubtedly yield powerful results. Unfortunately, due to excessive computing constraints, this method was not able to be tested in this project. Having said that, the best methods in predicting movie ratings have combined these approaches.

With the right computing power, a mixed method should be applied. Though these basic methods were used, we were still able to find a RMSE as low as 0.8098 on our `validation` set using the full `edx` set for training. The method that achieved this performance was matrix factorization using the `recosystem` package.