Movie Lens Project

Allison Patch

2019-05-01

Introduction

In October 2006, Netflix ran a contest to create a better movie recommendation system for its users. This project uses a portion of the data from Netflix's contest to attempt to create a recommendation algorithm that has a root mean squared error (RMSE) of 0.87750 or below. RMSE is important because it is a measurement of the average error our algorithm produces, which is helpful because it gives us an idea of how reliable our algorithm is.

The process of algorithm selection I used in my analysis was Matrix factorization. I chose this method because it allowed me to evaluate the success of iterative models with new parameter vectors included in each model iteration. This allows for easy comparison between models and shows how each new parameter influences the RMSE.

The following sections of this report are divided as follows: Methods, Results, and Conclusion. The Methods section will go over the data cleaning and exploration process and then set up the algorithm-building portion of the project. The results section will detail the results of the algortihm exploration. Finally, the conclusion will bring together the bigger picture of the results, and discuss other factors that could help in future research on this topic.

Methods

The first step to the process was downloading the data and splitting it into a training (data name: edx) and test set (data name: validation). For my project I used the code provided through the EdX course. Following this, I conducted a bit of data exploration guided by the quiz given in the EdX course. Finally, I developed several iterative models using the training set and measuring the quality of the model by calculating the RMSE using the test set for each model. The first two sections of code can be found together below. The remainder of the report will discuss the results of my recommendation system algorithm exploration.

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

# 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"))
```

```r
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(levels(movieId))[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)
edx <- rbind(edx, removed)

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


##########################################################################################################
## Exploring Data ########################################################################################
##########################################################################################################

#getting row and column length
nrow(edx)
```

```
## [1] 9000061
```

```r
ncol(edx)
```

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

```r
#frequency table to ratings to get number of 0 and 3 in ratings
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5
##   85420  345935  106379  710998  332783 2121638  792037 2588021  526309
##       5
## 1390541
```

```r
#number of unique movies
length(unique(edx$movieId))
```

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

```r
#number of unique users
length(unique(edx$userId))
```

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

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

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

```r
#Create data which give each genre category for a movie a separate entry
#Same movie_rating will occur in multiple rows if movie is categorized into multiple genres
by_genre<-separate_rows(edx, genres, convert = TRUE)

#number of movies by genre
by_genre %>% group_by(genres) %>% summarize(count=length(rating))
```

```
## # A tibble: 25 x 2
##    genres        count
##    <chr>         <int>
##  1 ""               12
##  2 Action      2560649
##  3 Adventure   1908692
##  4 Animation    467220
##  5 Children     737851
##  6 Comedy      3541284
##  7 Crime       1326917
##  8 Documentary   93252
##  9 Drama       3909401
## 10 Fantasy      925624
## # ... with 15 more rows
```

```r
##Do this again for validation dataset for future testing
by_genre_v<-separate_rows(validation, genres, convert = TRUE)


#Movie with highest rating
edx%>% group_by(title)%>% summarize(count=length(rating)) %>% arrange(desc(count))
```

```
## # A tibble: 10,676 x 2
##    title                                                       count
##    <chr>                                                       <int>
##  1 Pulp Fiction (1994)                                         31336
##  2 Forrest Gump (1994)                                          31076
##  3 Silence of the Lambs, The (1991)                             30280
##  4 Jurassic Park (1993)                                         29291
##  5 Shawshank Redemption, The (1994)                             27988
##  6 Braveheart (1995)                                            26258
##  7 Terminator 2: Judgment Day (1991)                            26115
##  8 Fugitive, The (1993)                                         26050
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25809
## 10 Batman (1989)                                                24343
## # ... with 10,666 more rows
```

Methods–Algorithm Exploration

In thinking of how to create the most accurate movie recommendation systems, we first have to reflect on the factors that go into movie recommendations. First, there is a general average rating that movies have. Second, we can think of how movies compare to the average—are they better, worse, the same? Third, we need to account for the individual movie watchers because every person has personal preferences and those preferences are likely to be seen through the individual's rating patterns. Finally, we need to think about the types of movies that are being shown. The different genres movies fall into are also likely to influence the rating we can expect a movie to receive.

Results

The following sections of this report exmaine each of these factors in turn, building to the final algortihm for recommendation that I produce.

To begin, I created a fuction which would calculate the RMSE for each of the models.

```
###Since the outcome of interest is RMSE, create a function to calculate RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

The overall average movie rating is the staring point to the analysis for my algorithms. The overall average was:

Using this overall average, the first model I ran used a naive bayes approach simply looking at the overall average ratings given and how well that could predict what rating movies in the test set woul be rated.

```
###calculate RMSE for the overall average
overall_average_rmse <- RMSE(validation$rating, mu_hat)
overall_average_rmse
```

```
## [1] 1.060651
```

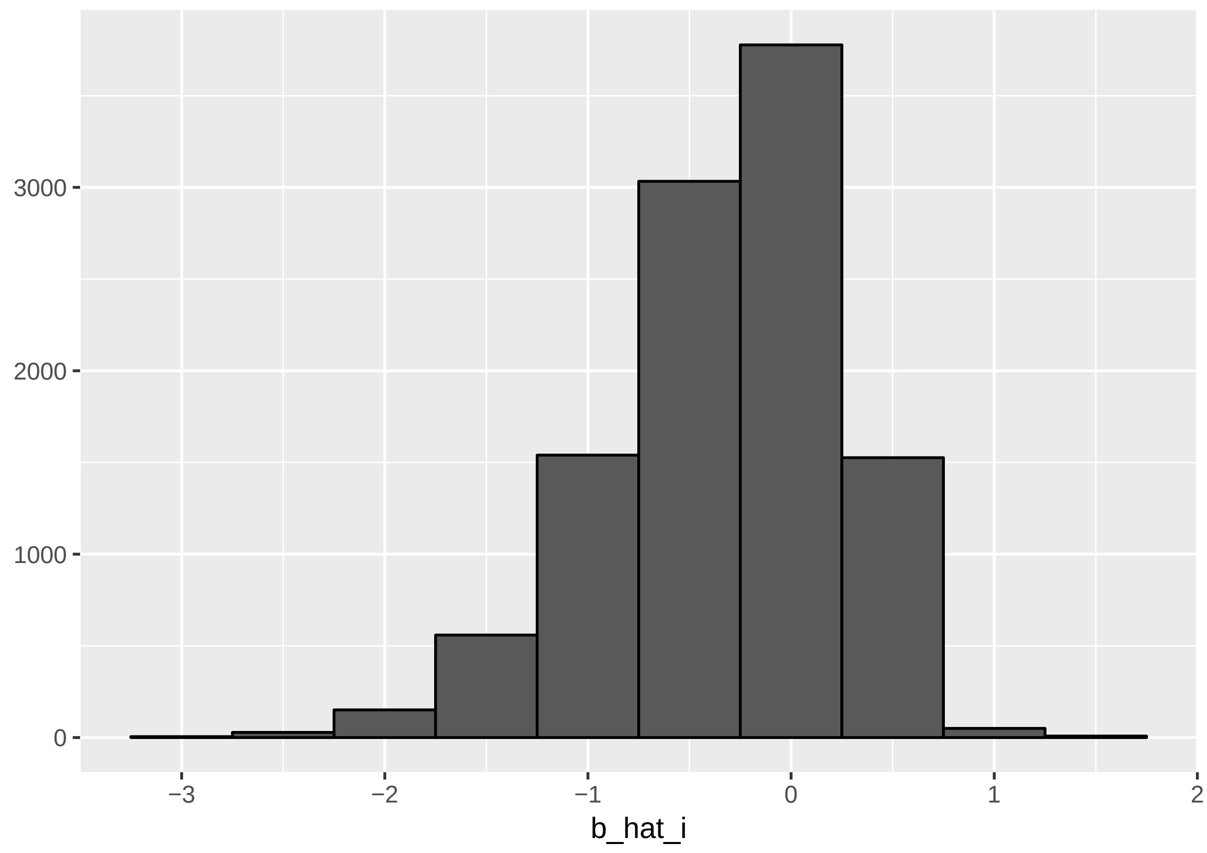
```
###Here I am creating a tibble to hold the results of the models to compare RMSE
rmse_results <- tibble(method = "Overall average", RMSE = overall_average_rmse)
```

method	RMSE
Overall average	1.060651

The RMSE result for the Overall Average Model is over 1, which is not very good. To improve this we will add another parameter to the model: Movie Effects. This will account for the difference in each movie's average rating from the overall average.

```
## b_hat_i is the average of the diffence between the rating given to the movie minus the overall avera
mu_hat <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_hat_i = mean(rating - mu_hat))
```

A histogram of this difference shows a mostly normal distribution around 0.



I then ran a second model including Movie Effects

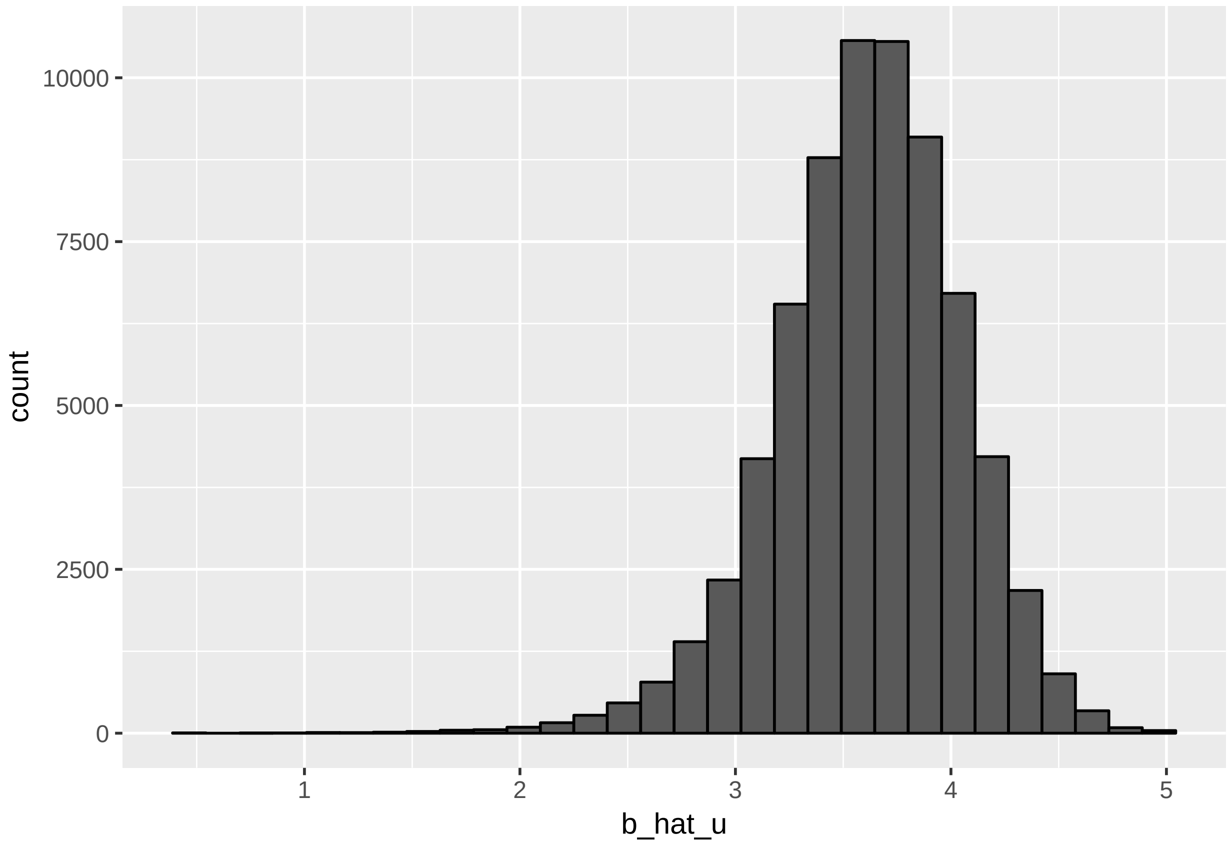
```
##Now we use the validation data to check the RSME for the mode

#first we have to calculate b_hat_i for the validation set and calculate the predicted ratings for the 
predicted_ratings <- mu_hat + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_hat_i

#next we test to see the RMSE
model_movieId_rmse <- RMSE(predicted_ratings, validation$rating)

#finally we add this model to our RSME table
rmse_results <- bind_rows(rmse_results,
```

```r
                      data_frame(method="Movie Effect Model",
                                 RMSE = model_movieId_rmse ))
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

method	RMSE
Overall average	1.0606506
Movie Effect Model	0.9437046

The RMSE result for the Movie Effects is still not very low so I added another parameter to the model: User Effects.A histogram of this difference shows a mostly normal distribution around 3.75.



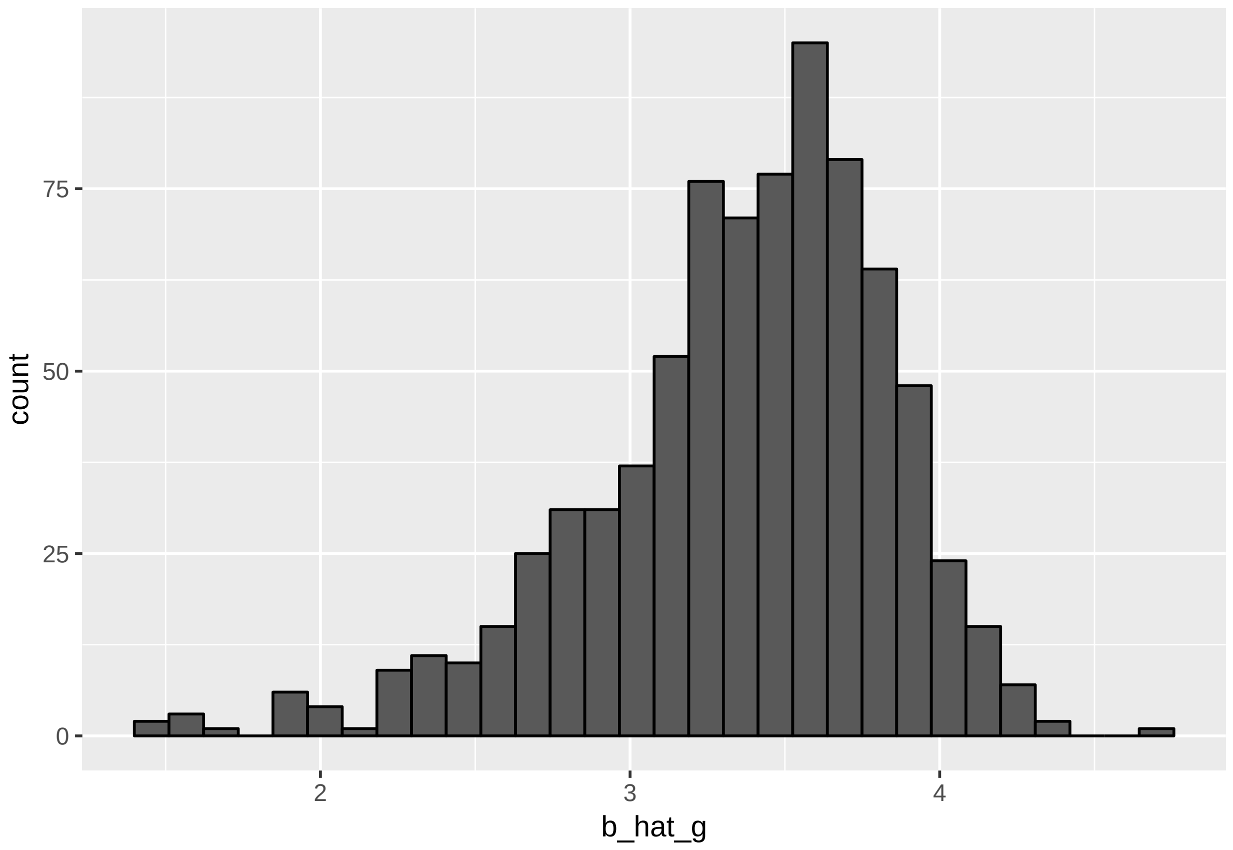
I then ran a third model including Movie Effects and User Effects

```r
##Now we use the validation data to check the RSME for the mode
#first we have to create user averages for the validation set so that we can make our predictions
user_avgs <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_hat_u = mean(rating - mu_hat - b_hat_i))

## Then we have to calculate the predicted ratings for the validation mode
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_hat_i + b_hat_u) %>%
  .$pred

#next we test to see the RMSE
model_movieUser_rmse <- RMSE(predicted_ratings, validation$rating)

#finally we add this model to our RSME table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_movieUser_rmse ))
```

method	RMSE
Overall average	1.0606506
Movie Effect Model	0.9437046
Movie + User Effects Model	0.8293252

We see that this RMSE result is sufficiently low for our recommendation system to qualify as a success

The RMSE result is fine, but we want to see if there is any additional help from: Genre Effects.A histogram of this difference shows a mostly normal distribution around 3.5.



I then ran a fourth model including Movie Effects, User Effects, and Genre Effects.

```r
##Now we use the validation data to check the RSME for the mode
#first we have to create user averages for the validation set so that we can make our predictions
genre_avgs <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_hat_g = mean(rating - mu_hat - b_hat_i - b_hat_u))

## Then we have to calculate the predicted ratings for the validation mode
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_hat_i + b_hat_u +b_hat_g) %>%
  .$pred

#next we test to see the RMSE
model_movieUserGenre_rmse <- RMSE(predicted_ratings, validation$rating)

#finally we add this model to our RSME table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + Genre Effects Model",
                                     RMSE = model_movieUserGenre_rmse ))
```

method	RMSE
Overall average	1.0606506
Movie Effect Model	0.9437046
Movie + User Effects Model	0.8293252
Movie + User + Genre Effects Model	0.8286165

However, in exploring the data, I discovered that individual movies were grouped into several genre categories at once, which made it difficult to really examine the genre effect. To account for this, I re-created the datasets with each genre for a movie receiving its own row. I then re-ran all of the algorithms I had originally created on these new datasets.

method	RMSE
Overall average	1.0606506
Movie Effect Model	0.9437046
Movie + User Effects Model	0.8293252
Movie + User + Genre Effects Model	0.8286165
Split Genres: Overall average	1.0548116
Split Genres: Movie Effect Model	0.9419567
Split Genres: Movie + User Effects Model	0.8215903
Split Genres: Movie + User + Genre Effects Model	0.8214970

With these new models, we see that using the data split by genre helps to lower RMSE ever slightly more.

##Conclusion

In attempting to create a high quality recommendation system for Netflix users with the dataset provided, it appears that two parameters combine to provide a relatively good system: Movie Effects and User Effects.

Including genre effects also appears to improve the model slightly, these differences may be a bit too small to justify the potential for over-training the model.

Knowing what movies a particular user likes and what the average ratings for particular movies are appears to be enough information to provide a decent recommendation for other movies that user would like. Several other demographic parameters (such as age, gender, education, etc.) that would help define the popluation individual users belong to could help to introduce other factors that could help direct the recommendation system to an even greater accuracy.