MovieLens Project

Greg Baguhin

27/07/2021

INTRODUCTION:

Project Overview: Recommendation systems are one of the most important and useful applications of Machine Learning. Major companies like Amazon, Netflix and Apple collect massive amounts of data from their customers through rating surveys which are then used to predict customer preferences and tailor their sales strategies based on those predicted preferences. The more accurate the predictions are, the more successful the companies become in selling their products and services.

In 2006 Netflix issued a challenge to the data science community to improve its recommendation algorithm by 10% and win one million dollars. As the Netflix dataset is not publicly available for analysis, this project will attempt to develop its own recommendation system using the 10M version Movielens movie ratings dataset generated by the Grouplens research lab.

Objectives: We will show how the algorithm evolves from analyzing the effects of a single variable to measuring the effects of a combination of variables including the addition of a penalty term in a technique called regularization used by one of the winners of the Netflix challenge.

Challenges: The recommendation system challenge is more complicated than previous machine learning datasets discussed in the course because each outcome, Y has a different sets of predictors since different users rate different movies and different number of movies, essentially using the entire matrix as predictors of each cell.

Our approach to this problem is how to design a model that would account for the different biases inherent in rating systems may it be movie bias, user bias, a combination thereof and applying regularization techniques to penalize noisy estimates and outliers caused by movies with fewer ratings and users who rate fewer movies.

METHODS AND ANALYSIS:

Data Preparation: The code below was provided as starting point for the Capstone Movielens Project to create Test and Validation sets. It also cleans up the working environment.

Create edx set, validation set (final hold-out test set) Note: this process could take a couple of minutes.

```r
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)
```

```r
# 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 <- 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")

#If using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

#Join movies and ratings sharing common movieIds
movielens <- left_join(ratings, movies, by = "movieId")


#Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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)

#Tidy up work environment
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

```r
#Looking at the first 6 rows of data
head(edx)
```

Inspecting the dataset and its basic summary statistics

```
##    userId movieId rating timestamp                          title
## 1:      1     122      5 838985046               Boomerang (1992)
## 2:      1     185      5 838983525                Net, The (1995)
## 3:      1     292      5 838983421                Outbreak (1995)
## 4:      1     316      5 838983392                Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                 Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:        Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:        Children|Comedy|Fantasy
```

```
#Basic summary statistics
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title               genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Let's find out the total number of unique users and unique movie titles in our training dataset

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

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

If all users rated every movie title, the dataset should contain 746 Million rows but there are only have 9 Million records in our training set, just over 1%. This shows the sparseness of data points we can work with which highlights the daunting challenges of developing a recommendation system algorithm which would yield accurate rating predictions for a given movie and a given user.

Loss function: The Netflix challenge used the typical error loss similar to a standard deviation, called the residual mean square error (RMSE) on a test set to decide the winner.

Let's define an RMSE function for vectors of ratings and their predictors:

RMSE <- function(true_ratings, predicted_ratings){

sqrt(mean((true_ratings - predicted_ratings)^2))

}

First (simplest) model: Building the simplest model. Predicting the same rating for all movies independent of users. The estimate that minimizes RMSE is the least squares estimate of the true rating for all movies, mu, which in this case is the mean of all training set ratings, mu_hat

```
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.512465
```

```
#Simplest RMSE:

lazy_rmse <- RMSE(validation$rating, mu_hat)
lazy_rmse
```

```
## [1] 1.061202
```

```
rmse_summary <- data_frame(method = "Just the Mean", RMSE = lazy_rmse)
rmse_summary
```

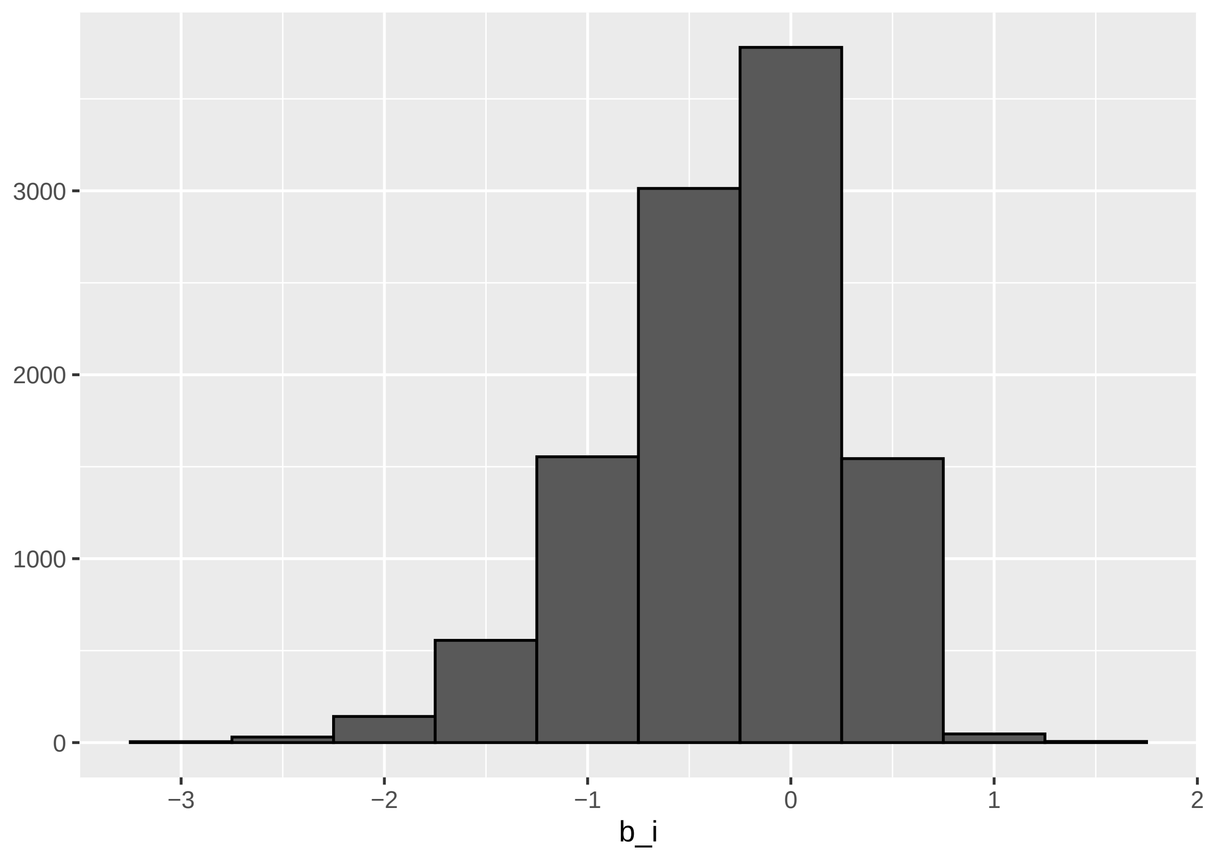
```
## # A tibble: 1 x 2
##   method         RMSE
##   <chr>         <dbl>
## 1 Just the Mean  1.06
```

Modeling movie effects: Accounting for movie bias, b_i, as the least squares estimates, mu, is a good estimate for b_i and should improve RMSE.

```
#Let's look at the histogram of movie ratings bias, b_i.

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

qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```



Notice that the highest number of movies had a ratings bias of almost or equal to 0, which translates to a 3.5 ratings, our mu value, with fewer movies having higher or lower b_i. Let's see if using b_i as least square estimate will improve our RMSE.

```
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_1_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_summary <- bind_rows(rmse_summary,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_summary %>% knitr::kable()
```

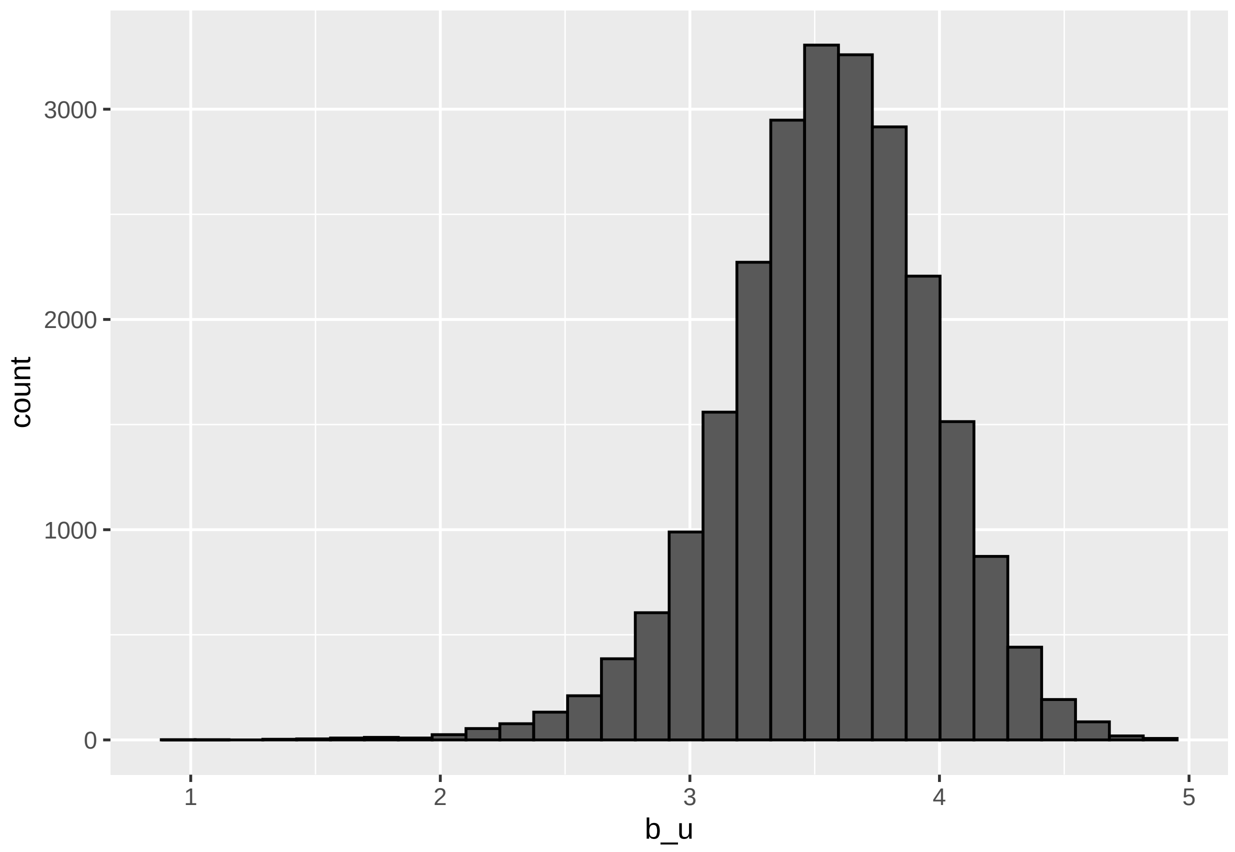
method	RMSE
Just the Mean	1.0612018
Movie Effect Model	0.9439087

```
#We have improved our RMSE from 1.06 to 0.94
```

Modeling User effects: We still need to adjust for user bias, b_u. To estimate b_u we take the mean of the observed rating minus mean test set rating minus the movie bias factor, b_i.

Let's see the average rating given by users with a minimum 100 movies rated:

```r
edx %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



Note that the highest frequency of ratings fall between 3 & 4. Let's see if adding b_u to the least squares estimate would improve our RMSE

```r
#Approximate user rating averages

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

#Predicting user-effect rmse:

predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

```r
model_2_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_summary <- bind_rows(rmse_summary,
                          data_frame(method="Movie + user Effect Model",
                                     RMSE = model_2_rmse ))
rmse_summary %>% knitr::kable()
```

method	RMSE
Just the Mean	1.0612018
Movie Effect Model	0.9439087
Movie + user Effect Model	0.8653488

Regularized movie effects: Let's calculate how many movie titles had 10 or less number of ratings

```r
edx %>%
  group_by(movieId) %>%
  filter(n() <= 10) %>% tally()
```

```
## # A tibble: 1,139 x 2
##    movieId     n
##      <dbl> <int>
##  1     109     6
##  2     395     5
##  3     399     6
##  4     403     9
##  5     604     2
##  6     607    10
##  7     642     8
##  8     644     9
##  9     654     9
## 10     739     7
## # ... with 1,129 more rows
```

We see that just over a thousand movies have 10 or less ratings. This will impact our attempt to use regularization on the Movie Effect model.

A penalty (or tuning) parameter lambda is used to reduce the effects of movies with fewer ratings to the overall b_i value.

```r
lambda <- 3 #optimum lambda value

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

#Predicted ratings:

predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

model_3_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_summary <- bind_rows(rmse_summary,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse ))
rmse_summary %>% knitr::kable()
```

method	RMSE
Just the Mean	1.0612018
Movie Effect Model	0.9439087
Movie + user Effect Model	0.8653488
Regularized Movie Effect Model	0.9438538

This model produced quite an insignificant difference from the Movie Effect Model. A deeper look into it is therefore recommended as time permits.

Regularized movie + user effects: Let's calculate the number of users who rated 10 or less movies:

```
edx %>%
  group_by(userId) %>%
  filter(n() <= 10) %>% tally()
```

```
## # A tibble: 1 x 2
##   userId     n
##    <int> <int>
## 1  62516    10
```

Just one user has rated 10 movies or less. A very small number of noisy estimates. This could affect the result of the regularization of the user effect.

```
#lambda is a tuning parameter. We use cross-validation to choose its optimum value.

lambdas <- seq(0, 10, 0.25)

#For each lambda,find b_i & b_u, followed by rating prediction & testing

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

  mu <- mean(edx$rating)

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

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
```

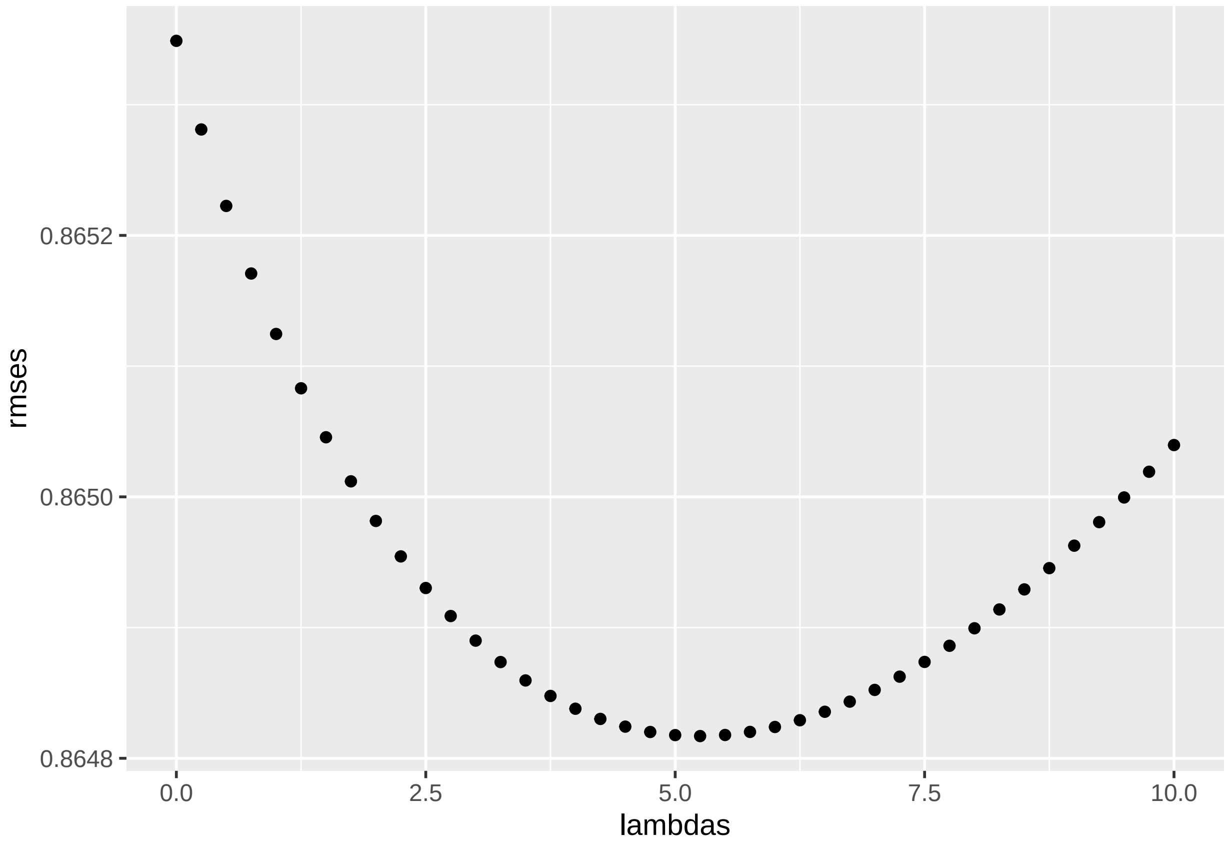
```r
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

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

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

#Plot rmses vs lambdas to select the optimal lambda

qplot(lambdas, rmses)
```



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

lambda
```

Get optimum value for lambda:

```
## [1] 5.25
```

Compute regularized estimates of b_i using lambda:

```r
movie_avgs_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

#Compute regularized estimates of b_u using lambda
user_avgs_reg <- edx %>%
  left_join(movie_avgs_reg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+lambda), n_u = n())

 #Predict ratings

rmse_summary <- bind_rows(rmse_summary,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses) ))
rmse_summary %>% knitr::kable()
```

method	RMSE
Just the Mean	1.0612018
Movie Effect Model	0.9439087
Movie + user Effect Model	0.8653488
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170

```r


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

DISCUSSION OF RESULTS:

method	RMSE
Just the Mean	1.0612018
Movie Effect Model	0.9439087
Movie + user Effect Model	0.8653488
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170

Despite limitations due to the sparseness of ratings empirical data to work with which could result in noisy estimates that need to be attenuated, we were able to model in different stages recommendation system algorithms using our training data. First we set a baseline using the mean of the training data ratings, we named mu. Next we added movie effect, b_i using least squares estimate and improved our RMSE from 1.061 to 0.944 or an improvement of 11%. Then we added the user effect into the movie effect model which further improved our RMSE to 0.865 or an improvement of 18.5% from the baseline model.

As we earlier mentioned, our training data included users rating just 10 or less movies and movies that had just 10 ratings. We introduced regularization where a penalty term lambda is added to constrain the effects of these data on the RMSE value. Using the penalty term on the movie effect model did not significantly improve the RMSE did improve the RMSE value to 0.865 when used with least squares estimate of Movie + User effect model. This is an improvement of 18.5%.

CONCLUSION:

What we learned: From the results we can gather a few useful observations: There were significant differences in RMSE improvement between "Just the Mean" model and the Movie Effect and Movie + User Effect Model.

There was not the same significant difference in RMSE improvement between the Regularized and least squares estimates. This suggests that there may be less noisy estimates in our training dataset than we first thought.Another possibility could be that we need to model other factors that could be used to improve our recommendation algorithm more such as genre effect, or year of release effect.

Looking ahead: Machine learning has a such myriad applications to business, science, and medicine that in cases where there is not enough available data for training algorithms, we should also explore the possibility of manufacturing data ("synthetic data") to mimic real data and then using them to train machine learning applications for efficiently, more effectively and more accurately. Companies like American Express are exploring this approach to increase its ability to detect and prevent fraud.