MovieLens Final Report

Christina Shao

06/07/2021

Introduction

The movielens Dataset

The provided movie lens data set consists of 9000055 individual reviews created by users for a large variety of movies.

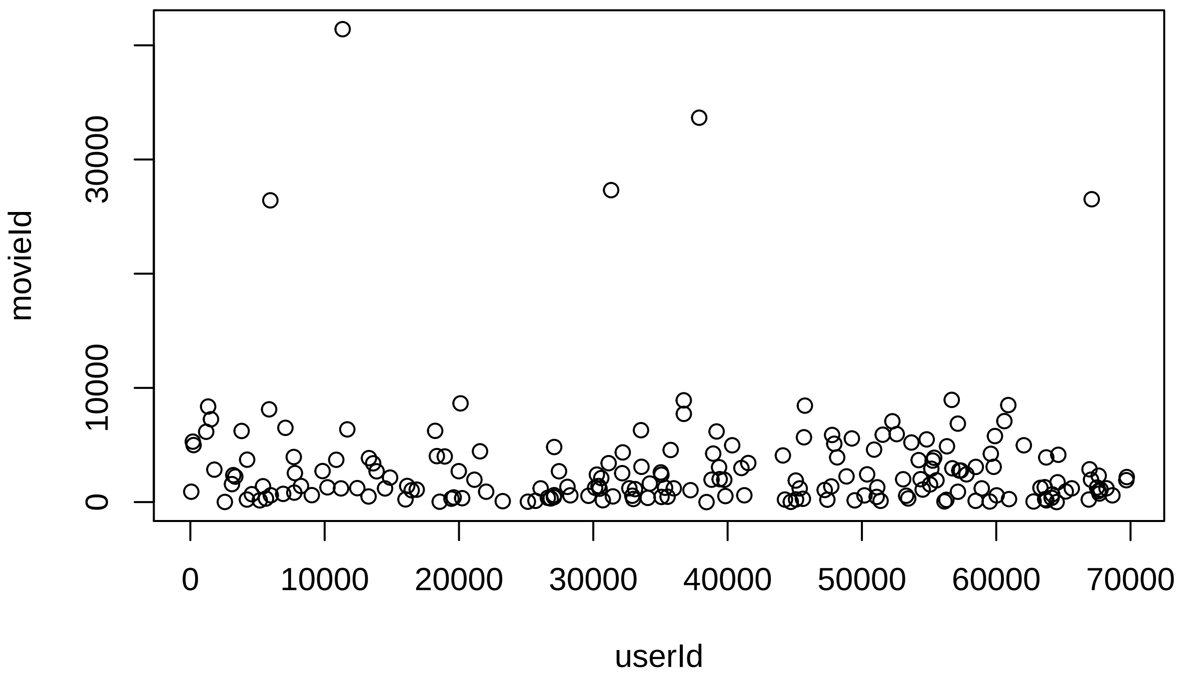
Each one of these reviews contains the following 6 different pieces of information:

- The numerical userId of the reviewer
- The numerical movieId of the movie they wrote this review about
- The title of that movie with the year it was released in brackets
- A timestamp of when the review was written
- The genres the movie is catagorized under
- The rating out of 5 that the reviewer gave the movie

Here is an example of 5 randomly selected reviews from the data set:

userId	movieId	rating	timestamp	title	genres
73	1262	4	974297754	Great Escape, The (1963)	Action|Adventure|Drama|War
59	292	3	838984731	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
65	2197	3	987892787	Firelight (1997)	Drama
34	724	2	981826221	Craft, The (1996)	Comedy|Drama|Fantasy|Horror|Thriller
65	2369	4	950887090	Desperately Seeking Susan (1985)	Comedy|Drama|Romance

As discussed during the Recommendation Systems section of this course, the movielens data set is **very sparse**, as visualized in this plot of 100 randomly selected reviews mapping movieIds and userIds.



```
## NULL
```

And this property of the dataset is what defines the goal of this project:

> This project aims to use the data provided in the movielens data set to create a machine learning algorithm that can predict what rating a user would give a movie given its genres, title, movieId, and the timing of the rating to a degree of accuracy where the root mean square error will fall below 0.86490.

Such an algorithm would essentially fill in the empty spaces left in the matrix of movie reviews with predicted values. Given the uneven nature of this distribution, it can be inferred that there will be strong—or at least directional—effects acting on the ratings that users give.

Key Steps

Building this algorithm will require the following tasks:

1. Exploring and formatting the movielens data set to identify key predictors
2. Determining the individual and cumulative effects those key predictors have on the rating via linear regression

- This will involve building multiple models for each predictor to determine the best performing model

3. Performing regularization on the combined model to eliminate large effects created by a small number of datasets
4. Testing the model on the movielens data set

Methods and Analysis

Exploring and formatting the edx data set to identify key predictors

The following predictors were determined to be useful pieces of information to train the movielens algorithm on: genre, release year, user, and movie.

Movies and Users As was also discussed in the Recommendation Systems section of this course, some movies tend to be rated higher than others as seen in the following table.

movieId	mean	zScore
2281	3.000000	-0.4833066
3882	3.085342	-0.4028206
33132	2.857143	-0.6180354
609	3.049659	-0.4364733
5887	2.326923	-1.1180864

A similar effect can be seen with the users, with some being more generous than others when leaving reviews.

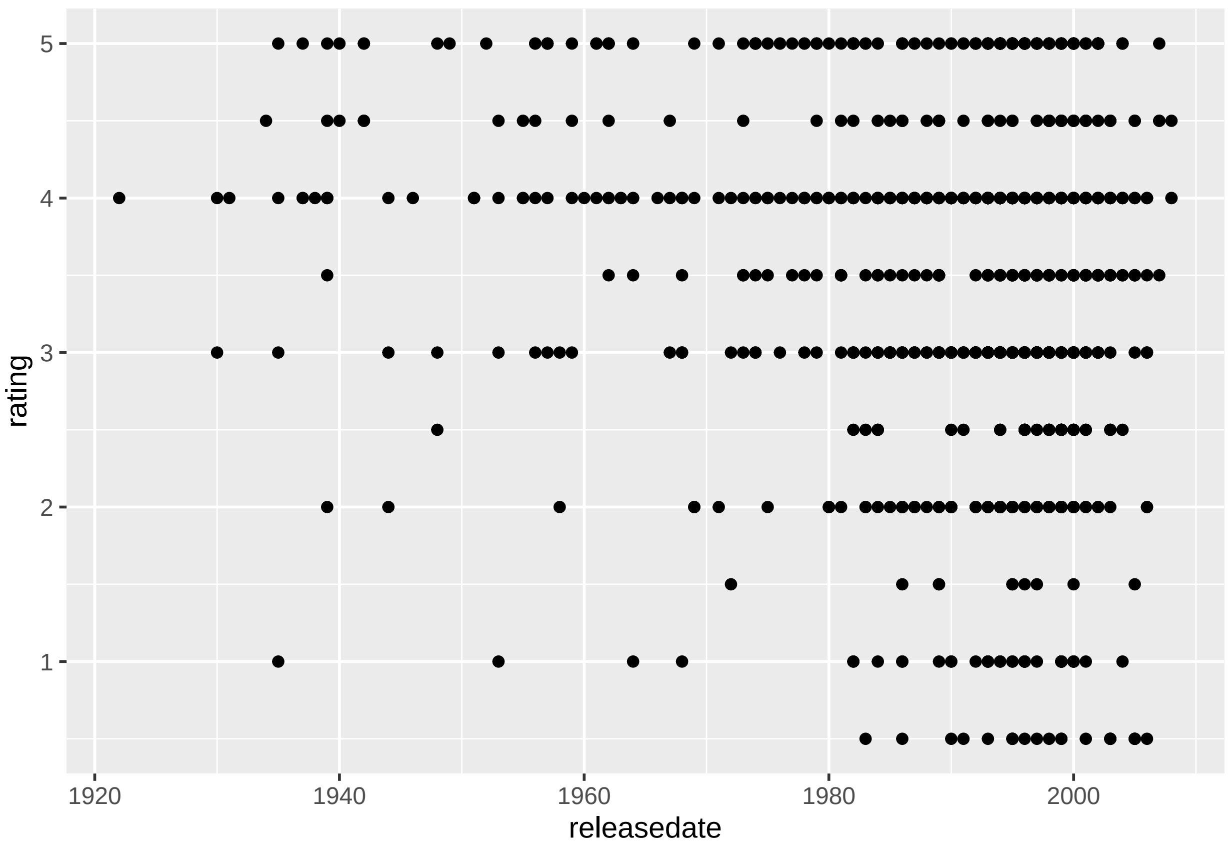
userId	mean	zScore
55592	2.676471	-0.7884277
30801	3.794872	0.2663380
7711	3.465471	-0.0443204
2663	3.333333	-0.1689395
16159	4.050000	0.5069498

Release Year Moving away from the Recommendation Systems course material, the focus now shifts to the effects that release year and genre have on the rating. The release dates of each movie are written in brackets within the title column of the data frame and therefore must be extracted with the following code:

```r
edx <- edx %>% mutate(releasedate=as.numeric(sub('.*(\\d{4}).*', '\\1', edx$title)))
kable(edx[sample(nrow(edx), 5, replace=F),])
```

userId	movieId	rating	timestamp	title	genres	releasedate
9531	1377	3.0	875636733	Batman Returns (1992)	Action|Crime|Fantasy|Mystery|Romance|Thriller	1992
68804	3527	4.0	970536823	Predator (1987)	Action|Sci-Fi|Thriller	1987
61404	5829	1.0	1129618456	Men with Brooms (2002)	Comedy|Drama|Romance	2002
55246	6934	3.5	1074613446	Matrix Revolutions, The (2003)	Action|Sci-Fi|Thriller	2003
50005	4963	4.0	1206382041	Ocean's Eleven (2001)	Comedy|Crime|Thriller	2001

However, unlike with movie and user effects, there does not seem to be any clear effects that result from release years. There is no clear singular effect for any given release year or genre on the rating.



It is not obvious that any particular release year tends to be rated much higher than others. A possible explanation for this phenomenon is personal preference; some users will be partial to movies from a particular era. **This presents two methods with which release year can be used as a predictor.** It can either:

1. Be evaluated as a whole regardless of personal preference. This risks producing an effect/prediction that is near negligible as it did not show any strong correlation with rating.
2. Be evaluated with regards to each individual user, thus taking personal preference into account. This risks over training the model.

Genre The genres column of the data frame lists combinations of individual genres. These can be separated with the following code:

```
edx_g <- cSplit(edx,"genres","|")
kable(edx_g[sample(nrow(edx_g), 5, replace=F),])
```

userId	movieId	rating	timestamp	title	releasedate	genres_1	genres_2	genres_3	genres_4	genres_5	genres_6	genres_7	genres_8
55228	1136	5.0	1127341925	Monty Python and the Holy Grail (1975)	1975	Comedy	NA	NA	NA	NA	NA	NA	NA
30104	590	3.5	1225315038	Dances with Wolves (1990)	1990	Adventure	Drama	Western	NA	NA	NA	NA	NA
16458	7700	4.0	1220767231	Wages of Fear, The (Le Salaire de la peur) (1953)	1953	Action	Adventure	Drama	Thriller	NA	NA	NA	NA
9104	1580	4.0	1088620445	Men in Black (1997)	1997	Action	Comedy	Sci-Fi	NA	NA	NA	NA	NA
62413	414	3.0	1018146116	Air Up There, The (1994)	1994	Comedy	NA	NA	NA	NA	NA	NA	NA

This creates the opportunity to, again, either:

1. Evaluate genre as a whole, looking at unique genre combinations as genres in and of themselves.
2. Evaluate the effects of each individual genre as partial effects and then adding applicable genre effects together to determine the overall effect.

Determining the individual and cumulative effects predictors have on rating via linear regression

As using the validation set to determine which model is the most effective must be avoided, a testing and training set was created by subsetting the edx data set. With these two data sets, the individual predictors can then be used to create models with linear regression.

Movie Bias The effects of each individual movie can be determined with linear regression. As stated in the Recommendation Systems lesson, it would be easy to simply do so with

```
lm(rating ~ as.factor(movieId), data = test_set)
```

but that would likely crash R given how massive the movielens data set and the subsetted test data sets are. As such, an estimation can be made by looking at the equation that is being built for the predictions.

Taking movie bias into account, the equation looks as follows:

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

Where Y = rating, μ = average rating, b = bias/effect of the predictor of interest, and ε = error. Rearranging this equation yields an approximation for b_i, or the movie bias:

$$b_i = Y_{\mu,i} - \mu$$

Disregarding error($\varepsilon_{\mu,i}$), the following code will produce the movie bias (b_i):

```
b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + l))
  # NAs to 0
  b_i[is.na(b_i)] <- 0
  # add movie avgs to train set for future use
  train_set <- train_set %>% left_join(b_i, by='movieId')
```

The l found in the denominator of the mean calculation (μ) will be determined as the model is regularized later in the process.

User Bias This process can be repeated for the user bias. The equation for predicted rating now becomes:

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

This makes it so that the user effect (b_u) becomes:

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

This can be applied on the dataset with the following code:

```
b_u <- train_set %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + l))
  # NAs to 0
  b_u[is.na(b_u)] <- 0
  # add user biases to train_set for future use
  train_set <- train_set %>% left_join(b_u, by='userId')
```

Release Date Bias As mentioned in the predictors section, there are two ways with which release date can be taken into account.

Release Date as a Whole Here, the equation becomes:

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

And the year effect can be found via:

$$b_y = Y_{\mu,i,u,y} - \mu - b_i - b_u$$

Which is implemented with the following code:

```
b_y <- train_set %>%
    group_by(releasedate) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u) / (n() + 1))
  # NAs to 0
  b_y[is.na(b_y)] <- 0
  train_set <- train_set %>% left_join(b_y, by='releasedate')
```

Release Date with Personal Preference To take personal preference into account, the effect of release date must be determined for each year-userId combination using the same equation displayed above. After creating vectors with all unique userIDs and release years, this can be determined with the following code:

```
byYear <- function(x){
  yearEffects <- train_set %>%
    filter(releasedate==x) %>%
    group_by(userId) %>%
    summarize(b_y2 = sum(rating - mu - b_i - b_u)/(n() + 1)) %>%
    distinct()
  # ensuring that every iteration of this function will yield the same # of rows by filling in missing
  full_join(users, yearEffects)$b_y2
}
# matrix of all b_y2 effects organizes by userid against releasedate
b_y2 <- sapply(releaseYears, byYear) %>% replace(is.na(.), 0)
```

Genre Bias This predictor will also require two different approaches.

By Genre Combinations Here, the equation becomes:

$$Y_{\mu,i,u,y,g} = \mu + b_i + b_u + b_y + b_g + \varepsilon_{\mu,i,u,y,g}$$

And the genre effect can be found via:

$$b_g = Y_{\mu,i,u,y,g} - \mu - b_i - b_u - b_y$$

Which is implemented with the following code:

```
b_g <- train_set %>%
    group_by(genres) %>%
    summarize (b_g = sum(rating - mu - b_i - b_y)/(n() + 1))
  # NAs to 0
  b_g[is.na(b_g)] <- 0
  # add genre biases to train_set for future use
  train_set <- train_set %>% left_join(b_g, by='genres')
```

Individual Partial Effects Here the individual partial effects of each genre must be found and then all applicable effects must be added together to determine the overall genre bias acting on the ratings. The equation then becomes:

$$Y_{\mu,i,u,y,g} = \mu + b_i + b_u + b_y + \sum(b_g) + \varepsilon_{\mu,i,u,y,g}$$

The partial effects of each genre can be found with the following code:

```r
genreEffects <- function (target){
  effect <-train_set %>% filter_at(vars(contains("genres")), any_vars(.==target))%>%
    summarize(b_g = mean(rating - mu - b_i - b_u - b_y)) %>% as.numeric()
}# create dataframe of effects by genre
genre_avgs <- sapply(genres, genreEffects,simplify = "array") %>% as.data.frame()
```

The following function will then take these partial effects and detect which genres are listed in every test rating to determine which partial effects to add together and apply to a given review:

```r
byGenre<- function (y, set, list){
  # empty vector to list which genres are present for a certain rating
  testGenres <- vector(mode='character')
  # cycle through all eight genre columns with for loop
  for(x in 1:8){
    # add genre entries into the empty testGenres vector
    entry<- set[y, 6+x] %>% as.character()
    testGenres <- append(testGenres, entry)
  }
  # remove NAs
  testGenres <- testGenres[!is.na(testGenres)]
  # empty variable for sum of all unweighted b_g
  b_g <-0
  for (i in 1: length(testGenres)){
    # matching the assigned genre with the appropriate unweighted b_g
    b_g <- b_g + genre_avgs$b_g[which(genre_avgs$genre== testGenres[i])]
  }
  # adding total b_g to list of b_gs for prediction
  list <- list %>% append(b_g)
}
```

Regularization In order to eliminate any large effects created by a very small number of reviews (e.g. only one person reviewed an indie film and loved it, causing the model to incorrectly predict that everyone else will also have the same reaction), the model must be regularized. To do so, we must minimize an equation that includes a penalty for small sample sizes. This equation looks like this:

$$\sum(Y + \mu + b_{i,u,y,g})^2 + \lambda \sum(b_{i,u,y,g})$$

Where

$$b_{i,u,y,g}$$

represents all calculated effects added together. Rearranging this equation yields that the

$$b_{i,u,y,g}$$

values needed to minimize the above equation can be calculated via the following:

$$\hat{b}_{i,u,y,g}(\lambda) = \frac{1}{\lambda + n} \sum_{u=1}^{n} (Y_{i,u,y,g} - \hat{b_i}^2)$$

The above equation had already been implemented in each predictor applied to the model while determining their respective biases. Therefore, all that must be done is to determine the lambda (λ) value that will minimize the root mean square error of the model. This can be done by encompassing the code that has been built so far with the following code:

```r
lambdas <- seq(0, 10, 0.25)

regularization <- function(l, train_set, test_set){
  # movie bias code
  # user bias code
  # release date bias code
  # genre bias code
}
rmses <- sapply(lambdas, regularization, train_set=train_set, test_set=test_set)
lambda <- lambdas[which.min(rmses)]
```

This code applies the model built so far on a vector of possible lambda values before determining which one results in the lowest root mean square error.

Results

Linear Regression Results

The naive root mean square error of this model was **1.059844**

Applying movie bias with the equation $Y_{\mu,i} = \mu + b_i + \varepsilon_{\mu,i}$ without regularization resulted in a root mean square error of **0.94398**.

Applying user bias with the equation $Y_{\mu,i,u} = \mu + b_i + b_u + \varepsilon_{\mu,i,u}$ without regularization resulted in a root mean square error of **0.87061**.

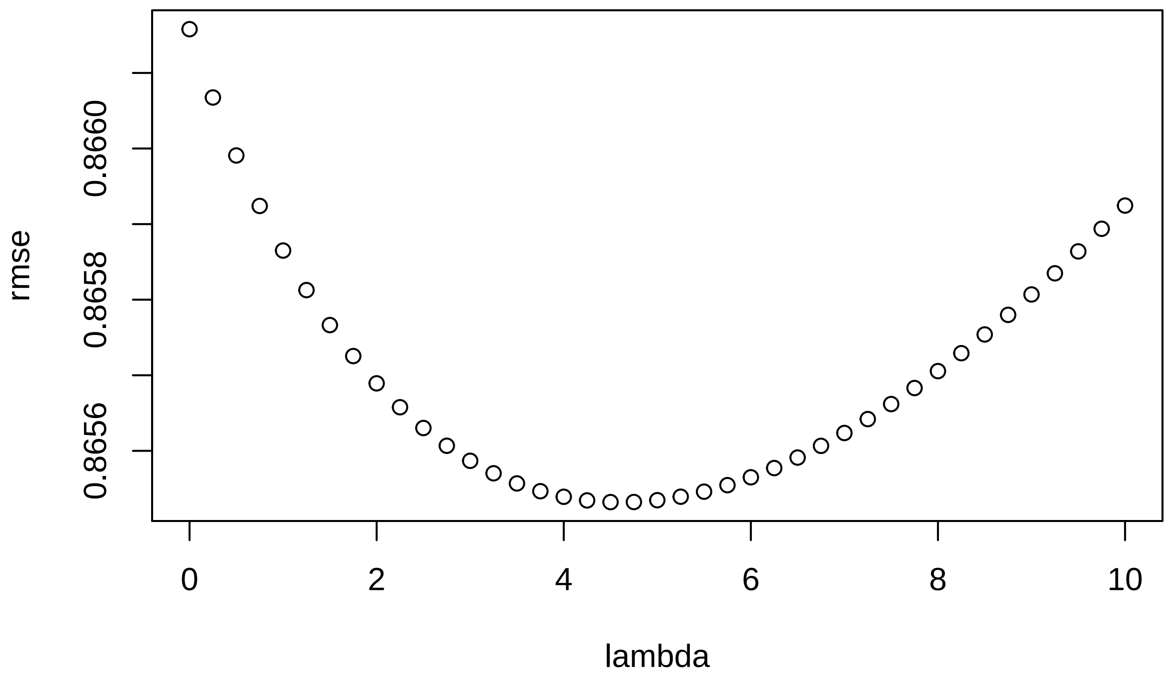
Applying year bias as a whole (disregarding personal preference) with the equation $Y_{\mu,i,u,y} = \mu + b_i + b_u + b_y + \varepsilon_{\mu,i,u,y}$ without regularization resulted in a root mean square error of **0.8703063**.
Applying year bias by user (accounting for personal preference) with the same equation without regularization resulted in a root mean square error of **0.8741204**, just slightly higher.
Accounting for personal preference likely produced a less accurate model due to overtraining. When performing linear regression on data that is sorted by both year and user, there are simply not enough data points per unique combination to train an accurate model.
As such, determining year bias as a whole was chosen as the model to proceed with.

Applying genre bias by combination with the equation $Y_{\mu,i,u,y,g} = \mu + b_i + b_u + b_y + b_g + \varepsilon_{\mu,i,u,y,g}$ without regularization resulted in a root mean square error of **0.8702153**.
Applying genre bias by individual genre (adding together partial effects) with the equation $Y_{\mu,i,u,y,g} = \mu + b_i + b_u + b_y + \sum(b_g) + \varepsilon_{\mu,i,u,y,g}$ without regularization resulted in a root mean square error of **0.8708216**.
Calculating genre bias by individual genre likely produced a less accurate model as not all combinations of genres are simply a the sum of its parts. Some combinations of genres may work much better or worse than expected (e.g. a comedy and horror movie would likely do worse than a comedy movie and a horror movie would individually).
As such, determining genre bias by genre combinations was chosen as the model to proceed with.

Given these results, the final model that would be subjected to regularization would involve movie bias, user bias, overall year bias, and genre combination bias.

Regularization Results

Substituting the appropriate code segments as determined above into the code template outlined in the Regularization section of Methods results in an ideal lambda value of **4.75** via cross validation as seen in the figure



below:

Applying this lambda value to train the finalized model on the complete edx dataset and running it on the validation dataset resulted in an RMSE of **0.8646305**.

Conclusion

Through the methods outlined in the above sections, a model was built via linear regression using the predictors of movie reviewed, user, release year of the movie, and the combination of genres the movie was classified under. After regularizing the model with a lambda value of 4.75, it was trained on the edx data set and tested on the validation data set. This produced a root mean square error of 0.8646305.

Limitations As this model was built primarily to meet a certain RMSE benchmark and not to be as accurate as possible, it could be vastly improved by simply building upon it more given a more ambitious goal.

One of the primary limitations of this model is concerning efficiency. As the model was first written as stand-alone code before being converted to a function, there are certainly parts of the code that could be optimized. For example, the createSet function could very easily be defined out of the overall function, thus reducing the need to redefine createSet every time the function is called. However, given that this assignment was to create a single function, such an idea could not be executed.

The final models that were chosen for year and genre bias could also certainly be improved. The improvements those two variables made on the RMSE were incredibly small at 0.003 and 0.001, respectively. The alternative models developed that took user preference and partial effects into account could likely become more effective than the ones chosen for this report had they been given a larger data set to train on, thus preventing overtraining.

Future Work Given limited time and a rather weak computer to run this code on, there could be a lot of work done to improve this model in the future. As mentioned above, if there was access to a larger data set that would allow for the user preference and partial genre effect models to train without being overtrained, it could massively improve the model—as well as making it resemble Netflix's actual algorithm more closely. Methods that allow for more robust cross validation like random forests would also result in a more accurate model. Random forests, however, is not feasible for this computer with what is already a very large data set.