MovieLens Recommendation System Capstone Project

Pamela Stameris Casey

4/30/2022

Introduction

This Movie Recommendation System is part of a Capstone Project for the edX Data Science Program. The goal of this project is to create a model for a movie recommendation organization (such as Netflix), to recommend movies to users that they would be likely to enjoy. Recommendation systems are widely used in marketing and social media to recommend books, clothing, restaurants, recipes, songs, potential mates, and other goods and services that people need (or don't need!). A model is developed using a curated dataset to analyze users' previous choices and, combined with with patterns found in the data, predict other items or services that are likely matches for the users.

The data source for this program is a dataset of movies, ratings, users, genres, and time stamps, which comes from the movie recommendation service MovieLens. The dataset provided contains about 10 million ratings, and will be analyzed, then used to predict ratings that individual users would likely give to individual movies. As will be shown, many of the users have never rated a large percentage of the movies, giving the opportunity for the system to provide many recommendations. The data has been formatted and provided as part of the edX Data Science program.

Dataset Specifics

Some selected information and statistics are provided below to provide familiarity with the format of the data, including the columns names and selected metrics of the MovieLens dataset.

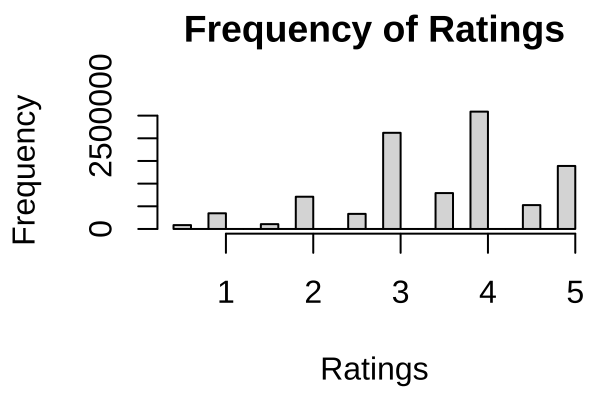
```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

Feature	Count or Value
Ratings	100,000,054
Movies	10,677
Users	69,878
Genre Groupings	797
*Possible Ratings	746,087,406
** % of all User/Movie Combinations with a Rating	1.2%
Average of All Ratings	3.512

* Based on the numbers shown of over 10,000 movies and nearly 70,000 users in **edx**, if each user had rated each movie, there would be a total of over 746 million ratings.

**In fact, MovieLens contains only about 9 million ratings - only 1.2% of the user/movie combinations. This offers the opportunity for hundreds of millions of ratings to be predicted in our recommendation system.

The following graph shows a breakdown of the ratings, which range from 0.5 to 5 stars. We can see that the most frequent rating given movies is 4 stars.



Partitioning of Dataset for Model Development

The MovieLens dataset has been split into two subsets - a training set, called "**edx**", which contains 90% of the data and will be used for analysis and the development of a prediction model; and a test set, called "**validation**", which will be used exclusively for testing and providing accuracy results on the final model.

To develop my model, I further partitioned the training dataset, **edx**, into two separate datasets: "**mod_train**", which accounts for about 90% of **edx**, that I use for model development, and "**mod_test**", the remaining 10% of edx, to test my model at various stages. By training and testing my model using these partitions, the final model can be applied using the entire **edx** dataset as the training set from which to extract necessary averages and parameters, and performing the test on the **validation** set, which will not have been used at all during the development of the model. This method of holding out the **validation** set prevents overtraining of the model.

```r
# Create a train and test set from the edx dataset to
# train and test models. Test set is 10% of edx data.
set.seed(1, sample.kind="Rounding")
model_test_index <-
  createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
mod_train <- edx[-model_test_index,]
tempm <- edx[model_test_index,]

# Ensure userId & movieId in test set are also in train set
mod_test <- tempm %>%
  semi_join(mod_train, by = "movieId") %>%
  semi_join(mod_train, by = "userId")
# Add rows removed from test set back into train set
removed <- anti_join(tempm, mod_test)
mod_train <- rbind(mod_train, removed)
```

Analysis and Model Development

Model development consists of creating and testing various methods of predicting users' ratings of the movies based on movie and user information gleaned from the training dataset. Data characteristics and patterns can be observed using numerical analysis techniques and visualization of the data. For each method module developed, the code is run on the training set to predict the ratings. The predicted ratings are then compared to the actual ratings on the test set to assess accuracy of the model.

My model development consisted of developing similar modules to the ones that we learned about in our course on Machine Learning. I trained the model modules on my training set (**mod_train**) and tested them on my test set (**mod_test**) to check their performance. I continued to test and add modules to the process until I was satisfied with the performance (which is also when I ran out of ideas!). My final model is a series combination of these modules.

In the Model Development section of this document, I will explain the basis for each module method and show the supporting code used to train and test the module using **mod_train** and **mod_test**. The performance results from each of these methods will be displayed and assessed. Each module will use the results of the previous module as input, with the intention of improving the accuracy of the rating predictions.

In the "Results" section, I will show the performance results from running the final model, which is the combination of the modules from the development phase. This final model will use the **edx** dataset as input to re-establish the averages and parameters (which had been established previously on mod_train). The code then computes the new predicted ratings and compares them to the actual ratings in the **validation** set to assess accuracy of the final model.

The statistic that will be used to compare the predicted and actual ratings and assess the accuracy of the model is the Root Mean Square Error (RMSE). The lower the RMSE, the closer the model is to matching the actual ratings in the test set and the better it is at predicting ratings that have not been made.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$

In this formula, N is the number of actual ratings that exist in the data that is being assessed. (In development, this is the number of observations in **mod_test**; in the final test, this is the number of rows in **validation**). The quantity $\left(\hat{y}_{u,i} - y_{u,i}\right)^2$ is the square of the difference between the predicted and actual rating for each of these user/movie observations. Ratings will be predicted for each row in the **mod_test** and **validation** sets following the parameters set forth in the model. These predicted ratings will be compared with their corresponding actual ratings, and the RMSE will be calculated. An RMSE of 0 would mean that our model is a veritable crystal ball - having accurately predicted every user's rating of each movie they rated. Of course, this would be a nearly impossible task. The goal of this recommendation system is to have an RMSE of less than 0.86490.

Naive Model

This initial analysis (known as the Naive Model) simply takes the average of all movie ratings from the training set (this will be called **mu**), and uses that one value as the prediction value for all ratings. The calculation of the Root Mean Squared Error (RMSE), will serve as a baseline with which to compare more complex models.

The code for the RMSE function, used throughout this analysis, is shown below, followed by the average of all ratings and value of the Naive RMSE using this method.

```r
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Values for the average of all ratings and the Naive RMSE are calculated using this code.

```
mu <- mean(mod_train$rating)
naive_rmse <- RMSE(mod_test$rating, mu)
```

Calculation	Value
Average of All Ratings	3.513
Naive RMSE	1.060

This RMSE value of about 1 indicates that using 3.513 as the prediction for all ratings resulted in an average error of about 1 star per rating. The expectation of this model, is to improve prediction accuracy which will in turn lower the RMSE.

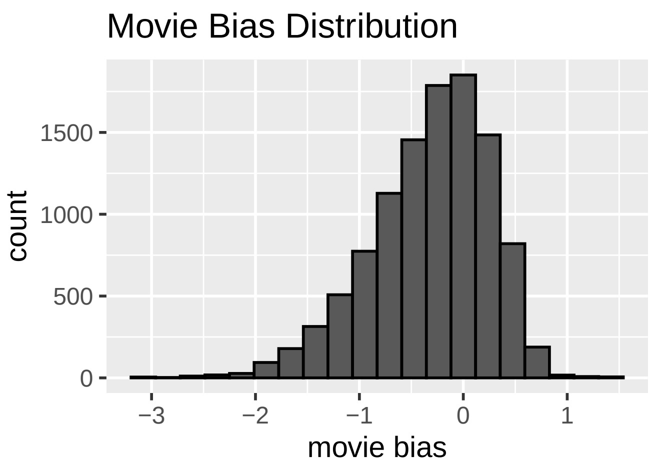
Model development will consist of adding various adjustments to the Naive Model based on patterns observed or considered in the training dataset. Adjustments will include Movie Bias, User Bias, Regularization of the Biases, and Readjustment of predicted ratings that are above or below the MovieLens rating range (0.5 to 5 stars).

Movie and User Bias Model

This first adjustment to the Naive model adds **movie bias**. To improve on the simple average model, we can use the observation that some movies are just more liked (or disliked) than others, as indicated by their higher or lower average ratings. This bias is calculated by grouping by movie and taking the mean rating of each movie. Once each individual movie mean has been found, the overall average (**mu**) will be subtracted, leaving the movie bias. For example, in my training set analysis, the average rating for the movie *Forest Gump* is 4.013 and the average rating for the movie Titanic is 3.300. This gives a movie bias to Forest Gump of +0.500 and a bias of -0.213 to Titanic compared to our mu value of 3.513.

```
#Movie Effect Adjustment
# bias_m = (sum(actual movie ratings - mu))/(# ratings)
movie_bias <- mod_train %>% group_by(movieId) %>%
  summarize(bias_m = mean(rating - mu))
```

The following graph shows the movie bias distribution. The plot is skewed left, indicating that more movies were rated less than the average. The highest bar at 0 indicates that the highest number of movies were rated around the average.



To incorporate this movie bias into our model, it is added to each prediction value (which now is mu from the Naive Model) for each movie and this sum is compared it to our mod_test ratings by calculating the new RMSE. This seems a long way around just using the individual movie averages as the prediction, but this movie bias value is used in all the upcoming modules and is logical to be generated here.

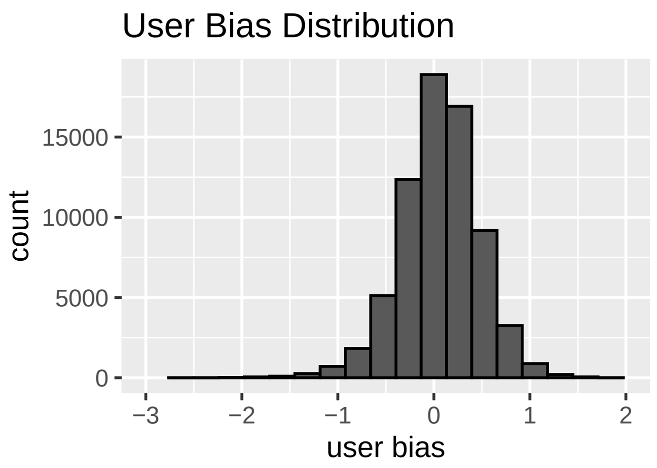
```
# Add movie-specific bias to mu to predict rating
pred_ratings_bm <-
  left_join(mod_test, movie_bias, by='movieId') %>%
  mutate(adj_rating_bm = mu + bias_m)
#RMSE based on predicted ratings adjusted for movie bias
rmse_with_bias_m <-
  RMSE(mod_test$rating, pred_ratings_bm$adj_rating_bm)
```

```
## [1] RMSE with movie bias: 0.943
```

Adjust for User Bias

Using similar logic, users also have different rating “personalities”, ranging from “easy pleasers” to “caustic critics”. The following code calculates individual **user bias** based on the ratings that were adjusted for movie bias. The calculation for user bias is **mean(rating - mu - movie bias)**. The graph shows the user bias distribution. The plot is fairly symmetrical about the mean, but there is indication of spread.

```
user_movie_adj <- mod_train %>%
  left_join(movie_bias, by='movieId') %>%
  group_by(userId) %>%
  mutate(bias_u = mean(rating - mu - bias_m)) %>%
  distinct(userId,bias_u)
```



Now we will adjust the model with the addition of the user bias to the rating prediction generated with movie bias, on an individual user basis, and calculate the new RMSE.

```
## [1] RMSE with user and movie bias: 0.8647
```

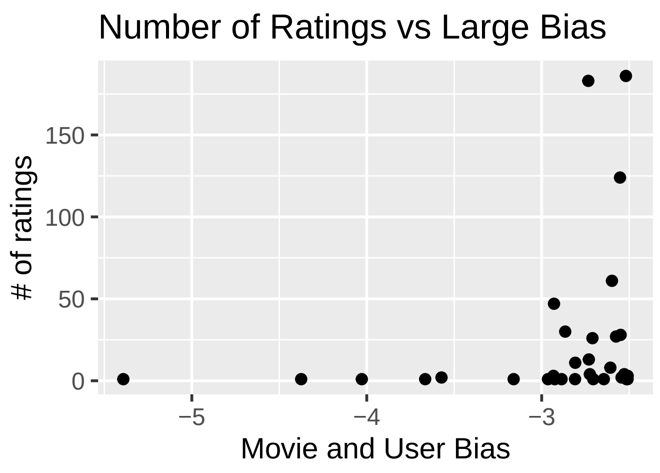
Here is a summary of our RMSE values on our test set so far. We see an improvement with the movie and user adjustments, and it is barely below the goal of 0.86490. However, these are being compared with the mod_test dataset, and not the validation set. I would like to have more of a confidence buffer between this RMSE and the target. So, I will try a couple more techniques to try and lower the RMSE further.

Prediction_Method	RMSE
Naive RMSE	1.0601
Adjusted for Movie Bias	0.9430
Adjusted for Movie and User Bias	0.8647

Regularization

Another attempt to improve the accuracy in predicting how our users would rate movies they haven't yet seen, is to regularize the biases. This method is used to lessen the effect of large values of movie bias and user bias that come from movies which have low numbers of ratings. The concept is to add a number (tuning parameter) to the denominator (representing total number of ratings for each movie) when calculating the bias, which will have negligible effect on the movies that have been rated many times, but will shrink the bias value of movies which have relatively few ratings. Support for trying this method can be shown with the following graph, which shows that, of the 10,677 movies in the mod_train dataset, the only ones that had a mean adjusted rating of greater than 2.5 or less than -2.5 were just 32 movies that had been rated fewer than 200 times, and most of then rated fewer tan 20 times. These movies actually all had adjusted ratings less than -2.5, with an average bias of -3.

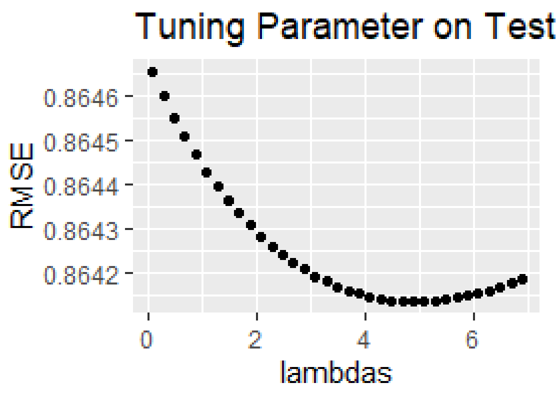
```r
large_bias <- mod_train %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_movie_adj, by = "userId") %>%
  mutate(bias = bias_m + bias_u) %>%
  group_by(movieId) %>%
  summarize(mean_bias = mean(bias), n = n()) %>%
  filter(abs(mean_bias) > 2.5)
```



Perhaps regularizing the combined user and movie biases will improve the accuracy of the predicted ratings. A function was created that used various values of the tuning parameter (represented as "lambda" in the plot below) to attempt to optimize the RMSE by regularizing the data, effectively adjusting the movies with large bias and low number of ratings, to give them less influence in the calculation of the bias which in turn is used to predict ratings. In the modeling process, a set of tuning parameters between 0 and 7 was applied to the training set to regularize the movie and user biases. The parameter was then optimized to find the minimum RMSE as determined on the test set. As can be seen on the plot below, the optimal value of lambda was 4.9.

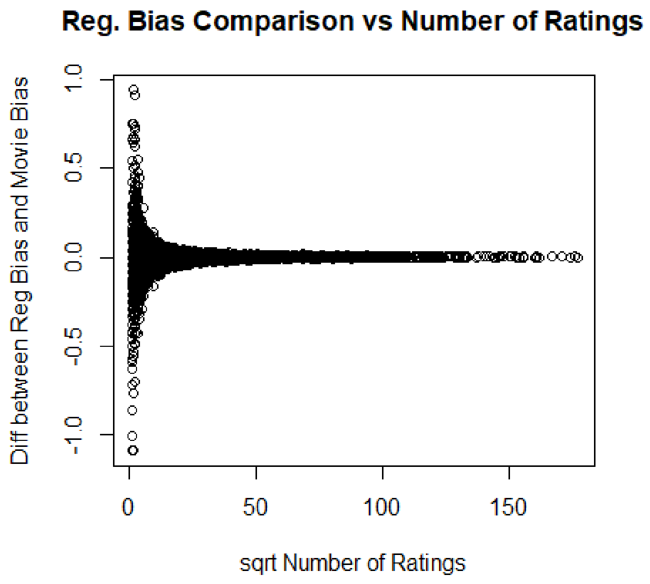
The code and "lambda" plot are shown here.

```
lambdas <- seq(0.1, 7, 0.2)
rmses_on_train <- sapply(lambdas, function(lambda){
movie_bias_reg <- mod_train %>%
  group_by(movieId) %>%
  summarize(bias_m_reg = sum(rating - mu)/(n()+lambda))

user_bias_reg <- mod_train %>%
  left_join(movie_bias_reg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(bias_u_reg = sum(rating - mu - bias_m_reg)/(n()+lambda))

ratings_reg <-
  mod_test %>%
  left_join(movie_bias_reg, by = "movieId") %>%
  left_join(user_bias_reg, by = "userId") %>%
  mutate(pred_ratings_reg = mu + bias_m_reg + bias_u_reg) %>%
  pull(pred_ratings_reg)

return(RMSE(mod_test$rating, ratings_reg))
})
```



```
## [1] Optimal Tuning Parameter (lambda): 4.9    Lowest RMSE: 0.8641
```

Once this tuning parameter was used on the test set, I compared the regularized bias (adjusted by the tuning parameter of 4.9) to the unregularized bias. The following graph shows the difference in bias (unregularized bias vs regularized bias), plotted against the number of ratings per movie. You can see that, as expected, the greatest adjustments were made to movies that had the fewest ratings.



Since we have gained another reduction in the RMSE (by 0.0006) using this method, we will add this layer of regularization to our final model, using he value of 4.9 as our tuning parameter.

Note: Since we needed to run this tuning parameter on data other than mod_train (the one that we used to generate the new regularized biases), we used the test set for this purpose. In running our final model, we are not allowed to use the validation set for anything other than testing, so we will use this lambda value of 4.9 as our tuning parameter.

Adjust Ratings over 5

Finally, after adjusting for movie bias, user bias and regularization, an analysis of the adjusted ratings showed that there were more than 2000 predicted ratings that were over 5 stars and under 0.5 stars. There are no ratings over 5 stars or under 0.5 stars in the MovieLens dataset, so an improvement in RMSE would be expected by eliminating this over-adjustment. The following code changes all predicted ratings over 5 stars to a 5-star rating and all predicted ratings under 0.5 stars to 0.5, then calculates a new RMSE.

```
high_low <- mod_test %>%
  left_join(movie_bias_regl, by = "movieId") %>%
  left_join(user_bias_regl, by = "userId") %>%
  mutate(pred_ratings_regl =mu+bias_m_regl+bias_u_regl) %>%
  mutate(no_over=ifelse(pred_ratings_regl>5, 5,
                        pred_ratings_regl)) %>%
  mutate(no_over_under=ifelse(pred_ratings_regl<0.5,0.5,
                              no_over))
```

```
## [1] Adjusted for out-of-range ratings: 0.86402
```

This technique also produced a very slight improvement in our RMSE, so will also be added to the final prediction model.

Results

We are now ready to run the final model on the **validation** set, using **edx** as the training set and the tuning parameter for regularization set to 4.9. The code is not shown, as it is simply the same code used in model development, but using edx in place of mod_train, and validation in place of mod_test. As a reminder, our target RMSE is to be below 0.86490. Fingers crossed…..

And, finally, here is the final summary of RMSE using the five layers of prediction modules, as tested on the validation dataset. The final value represents the RMSE of the prediction model as a whole.

Prediction Method	RMSE
Average of Ratings (Naive Model)	1.0612
Adjusted for Movie Bias	0.9440
Adjusted for Movie and User Bias	0.8653
Adjusted with Regularization	0.8648
Adjusted for Predicted Values out of Range	0.8647
RMSE OF PREDICTION MODEL	0.8647

The final value of the table above shows that the model, consisting of the combination of modules which each contribute an adjustment to the Naive Model, has met the goal of an RMSE less than 0.86490 when tested against the validation set.

Conclusion

The most influential adjustments in my model were the movie bias adjustment ("we can all agree it's a really good film") and the user bias adjustment ("easy pleasers vs caustic critics"). Overall, the RMSE improvement with all the tweaks compared to just using the Naive Model as a prediction is only about one-fifth of a star.

I did not use the genre and time stamp features in my modeling process. Although no discussion of these features was included in the report nor applied to the calculation of the RMSE, I did some analysis and visualization these features while working out the model development plan. For example, I tried to separate various genres from the others, such as Romantic and Adventure, but no patterns became apparent in a way that I felt I could use in my modeling strategies; therefore, in the end, were not taken into account in the formulation of the final model. Perhaps a future analysis and model development attempt would include these features as a way to reduce the RMSE, and thus improve the accuracy of the predictive model.

Other variables, although not included in the data provided, that would most likely lower the RMSE if taken into account, would be gender and age of the individuals providing the rating. I believe there would be definite patterns found when analyzing gender, age and genre features which could be used to further lower the value of the RMSE.

Recommendation systems are an effective way to present appealing offers to users/customers that are likely to result in a match that is enjoyable and matches their proven preferences. This ability to predict and offer goods and services which are likely to be ranked highly by the user is beneficial to the customer and business making recommendations as well. An understanding of Data Science and facility with a coding language is essential for effective analysis and manipulation of data for the development of these recommendation systems.

A Personal Note I must end this report by sharing that I gained so much knowledge completing this project. Before I started taking this Data Science Program almost two years ago (to learn something new as a way to pass the time during Covid isolation), I knew nothing about coding in R, or pretty much any language since FORTRAN from when I was in college 40-something years ago. I spent dozens upon dozens of hours working on this project, sometimes frustrated, but mostly determined to get it to work. I look at this code now, and though it may not look like very efficient or elegant to more experienced programmers, I got it to do what I wanted it to do, and I am delighted!