# MovieLens Report

*T. Boziuk*

## MovieLens: Summary

This is a report explaining the methods used to create movie predictions as part of the MovieLens Project as part of the EdX Data Science Capstone, HarvardX: PH125.9x.

The data set used is the MovieLens 10M, with the goal of minimizing the root mean squared error (RMSE) between a predicted rating (given a particular user and movie) and the actual rating given by the particular user to said movie.

Given the large size of the data set, many techniques proved difficult to use on a home computer. The chosen solution used a relatively simple model including a bias based on the movie, a bias based on the user, and a composite bias function based on the genres of the movie and the user's past ratings for movies with similar genres (an affinity for movies of a particular type).

$$Y_{u,i} = \mu + b_i + b_u + f(genre_i, affinity_u) + \epsilon_{u,i}$$

This approach yielded an RMSE on the test set of **0.849**.

NOTE: This report is currently being built. If this line has not yet been removed, it is likely incomplete.

## Analysis

### Preparation

The first step is to partition the data in a meaningful manner, allowing us to have a train set, a validation set, and a test set.

```
#Set a seed and randomly create a train/test split (90% train)from the edx data set.
#This will allow us to have a test set to find the RMSE after our script has made
#predictions.
set.seed(1)
test_index<- createDataPartition(y=edx$rating, times=1, p=0.1, list=FALSE)
test_set=edx[test_index,]
train_set=edx[-test_index,]

#Then, randomly create a validation set out of the train set. We can use this to
#determine any parameters without using the test data (which would wrongly allow
#information to leak from the test data to our algorithm).
valid_index=createDataPartition(y=train_set$rating,times=1,p=0.1,list=FALSE)
valid_set=train_set[valid_index,]
train_set=train_set[-valid_index,]
```

Next, we prepare the data set by finding all the unique genres, splitting genres for an individual movie and adding columns such that each movie has true or false if the genre is applicable. Because they are so many combinations of individual genres, this is a better approach than treating "Thriller" and "Thriller|Horror" as two completely unique genres.

```
#Generates the unique genre combination list, as well as the base list of genres.
#Note that there are too many unique combinations to make the use of the
#combinations practical; instead, we will have to depend on the individual genres.
genre_combo_list=unique(train_set$genres)
genre_list=unique(flatten(str_split(genre_combo_list,"\\|")))

# Adds columns for each unique genre, with true entries if that appears with the
#combined genre string. This is done for both the train and test set
for (ii in 1:20) {
  train_set[as.character(genre_list[ii])]=grepl(genre_list[ii] ,train_set$genres)
  test_set[as.character(genre_list[ii])]=grepl(genre_list[ii] ,test_set$genres)
}
```

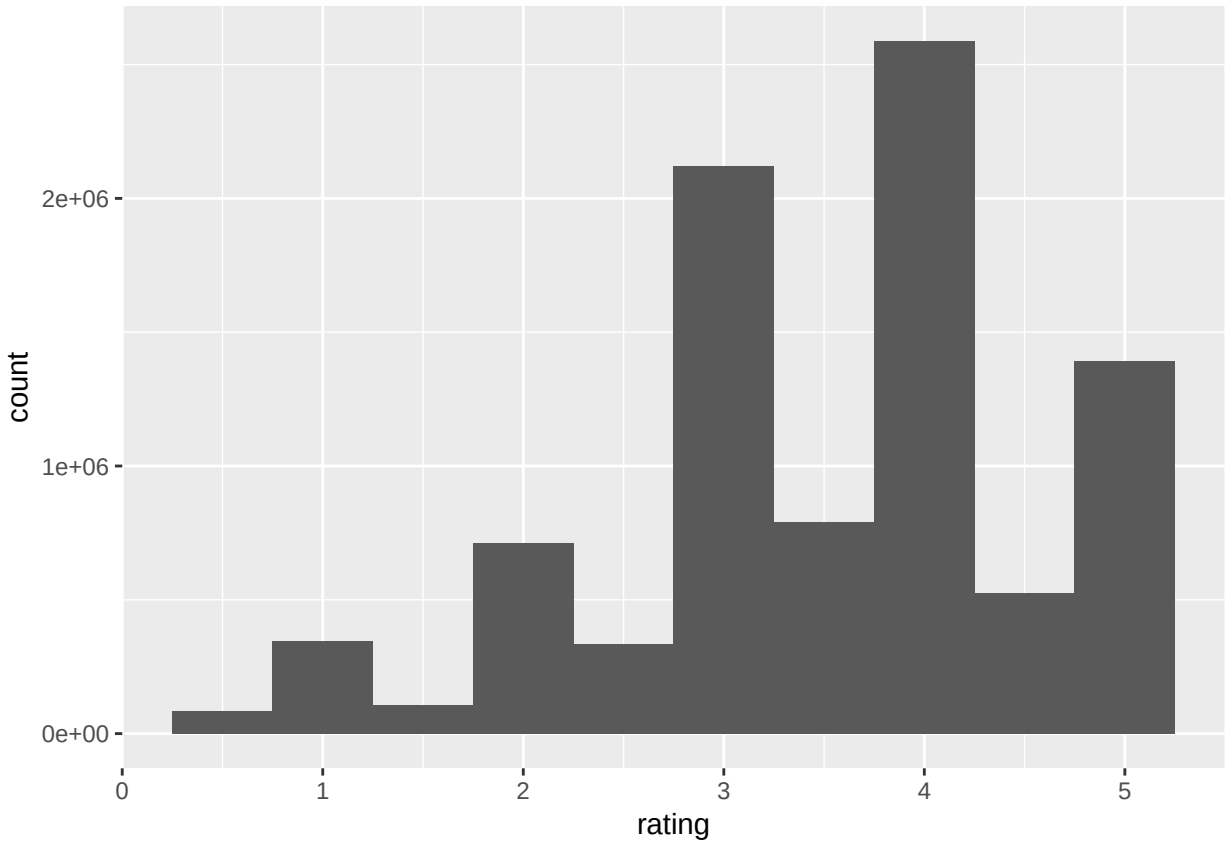
We then define two useful functions, which exist as possible goals: either minimizing RMSE, or maximizing accuracy. In this project, we ended with a goal of minimizing RMSE.

```
#Define two loss functions: RMSE and Accuracy (the original metric; no longer used)
RMSE <- function(predicted_ratings, true_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2,na.rm=TRUE))
}

compute_accuracy <- function(predicted_ratings, true_ratings){
  predicted_ratings=round(predicted_ratings*2)/2
  mean(predicted_ratings==true_ratings,na.rm=TRUE)
}
```

### Exploration

When the assignment was first posted, the goal was maximize the accuracy (i.e., percentage of predictions which were exactly correct). Because users were more likely to grant full star ratings (as opposed to half stars), significant increases in accuracy could be obtained by predicting full stars only, as opposed to the nearest half-star rating. This can be seen in the histogram below.




Unfortunately, this observation does not seem to aid in minimizing RMSE – the final goal of this assignment. Additionally, methods such as Naive Bayes classifiers (which could predict only full or half star ratings) that worked well for accuracy goals are not ideal for RMSE goals.

Given the goal of minimizing RMSE, the natural starting point is a simple bias model.

## Insights

While preparing our final model, there are insights at each step in the refinement process.

### Using the mean

The first step is to remove the mean rating; i.e, use the following model:

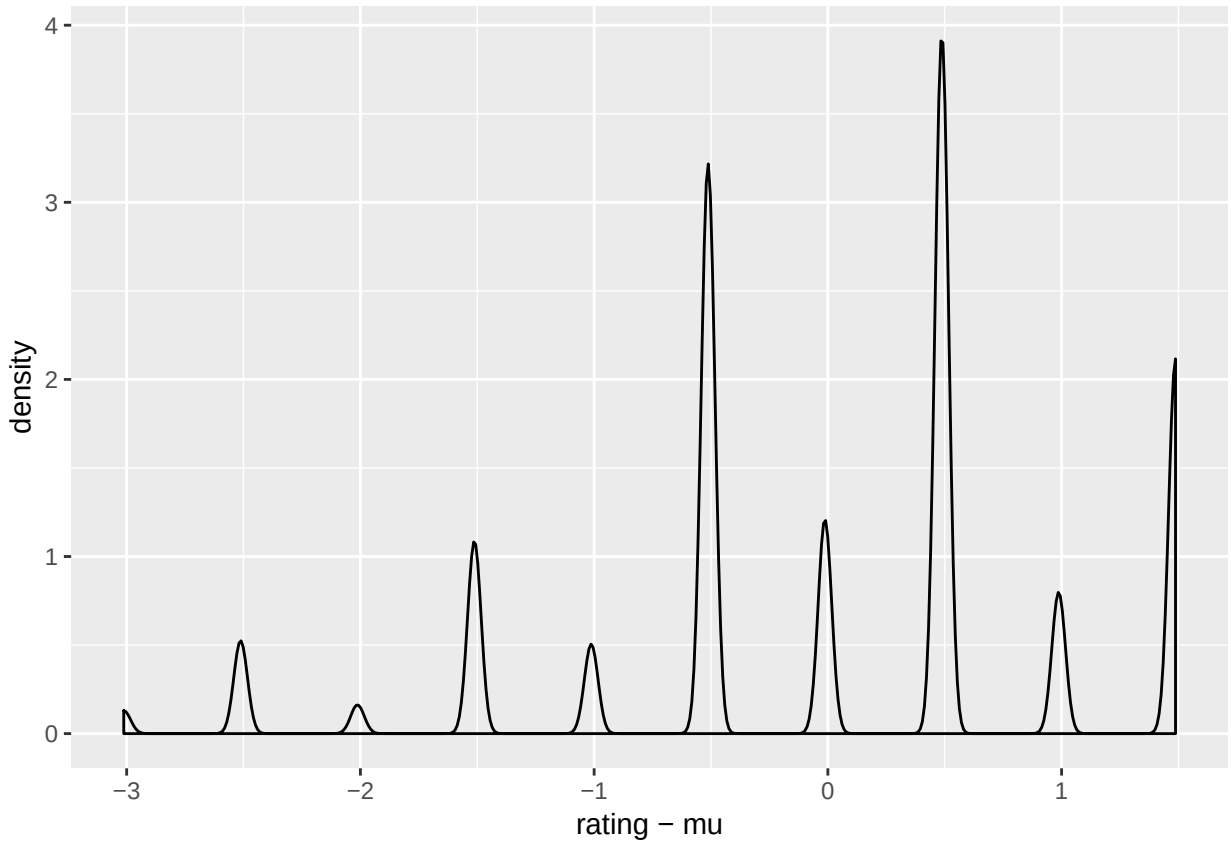
$$Y_{u,i} = \mu + \epsilon_{u,i}$$

It is instructive to also look at the residual ($\epsilon_{u,i}$) by plotting the density function of $Y_{u,i} - \mu$

```
#Compute the average for all movies (mu)
mu <- mean(train_set$rating)
print(mu)
```

```
## [1] 3.512447
```

```
train_set %>% ggplot(aes(x=rating-mu)) +geom_density()
```




As expected, the residual still shows peaks at all the half-star increments, although the mean value is now at zero. There is also significant skewness – a small number of bad ratings lead to a long left tail. Since we would like $\epsilon_{u,i}$ to represent only random noise, and there is clearly a pattern left in the residual, this is not sufficient.

### Adding a movie bias term

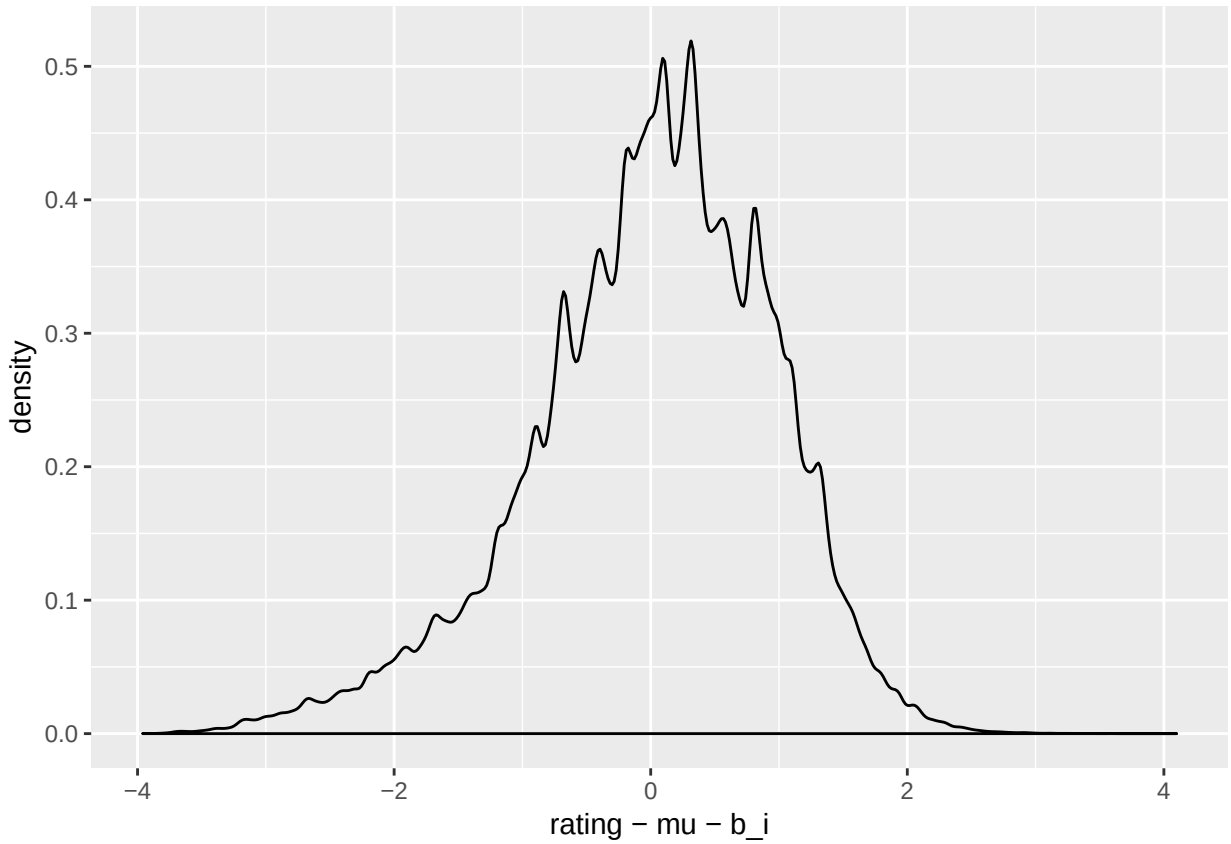
We continue to refine this model by including a bias term for the movie itself - i.e., "how good is this movie, generally?"

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

Again, we will plot the density function of the residual $Y_{u,i} - \mu - b_i$ .

```
#Compute the average residual for each individual movie (b_i)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

train_set_resid %>% ggplot(aes(x=rating-mu-b_i)) + geom_density()
```




Although the residual is now closer to a normal distribution, there are still clearly patterns in the data which have not been dealt by adding an average rating for an individual movie: there are still local peaks in the density, which could indicate groups of users who tend to rate movies higher or lower on average, as well as a skewness to the distribution (note that the left tail is longer than the right tail – there are a smaller number of more critical reviewers).

### Adding a user bias term

We continue to refine the model again, by including a bias term for an individual user - i.e., "how much of a curmudgeon is this user, generally?"

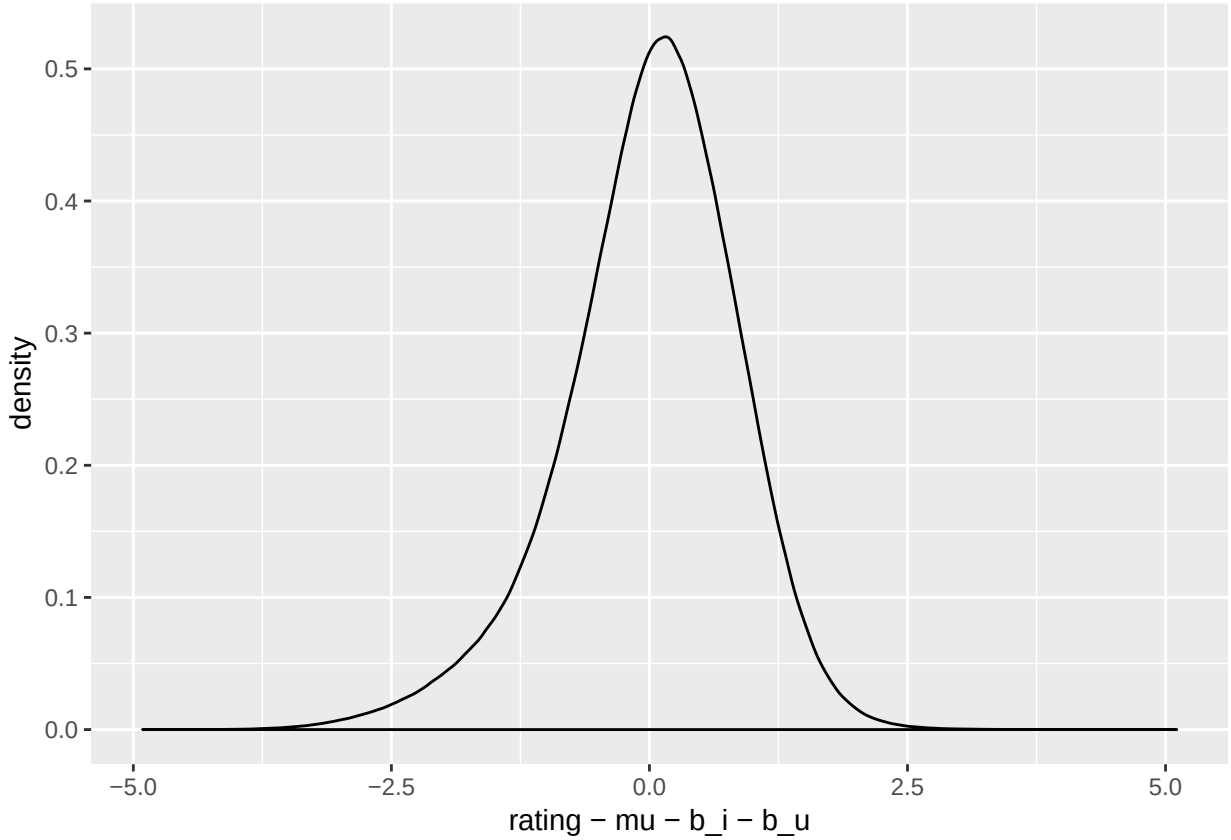
$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

Again, we will plot the density function of the residual $Y_{u,i} - \mu - b_i - b_u$ .

```
#Compute the average residual for each user (b_u) after removing the average score
#and individual movie score.
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#Create a train set that includes the residual left after removing the average,
#movie effect, and user effect.
train_set_resid <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u,resid=rating-pred)
```

```
train_set_resid %>% ggplot(aes(x=rating-mu-b_i-b_u)) + geom_density()
```




The residual now appears to be close to a normal distribution, despite some residual skewness, indicating that the remaining error could very well be close to random. Computing the RMSE on the test set at this stage shows that this model achieves **RMSE = 0.865**, which is below the goal threshold for the project. This does not mean, however, that we must end our model here.

## Final Model

It was decided to use genres effects to improve the predicted ratings. The broad theory is that certain users will have an affinity towards certain genres: for example, they may tend to rate horror movies highly and romance movies poorly; another user may rate drama movies highly but science fiction movies poorly. By using an individual user's past ratings, we can develop a model for their affinity towards certain genres. Then, when asked what they might rate a new movie, we use the genre of the new movie and the user's known affinities to modify the predicted rating.

It is worth noting that all of this computation - creating individual users' affinities, and modifying predicted ratings - happens *after* using the mean movie rating, the movie bias, and the user bias. Therefore, their affinities should hopefully model how much they like science fiction, *independent* of how good the movie is (the movie bias) and how much of a curmudgeon they generally are (the user bias).

$$Y_{u,i} = \mu + b_i + b_u + f(genre_i, affinity_u) + \epsilon_{u,i}$$

First we use the training data to create users' affinities (the residual rating for movies of a certain genre). We also record how many movies of that type they have rated. Both will be used in our function.

```
#Compute the residual and number of movies rated for each genre for each user.
#Only the first few genres are shown for reference.
genre_resids = train_set_resid %>% filter(.[,7] == TRUE) %>% group_by(userId) %>%
    summarize('Comedy resid'=mean(resid),'Comedy n' = n())
genre_resids = train_set_resid %>% filter(.[,8] == TRUE) %>% group_by(userId) %>%
    summarize('Romance resid'=mean(resid),'Romance n' = n()) %>%
      full_join(genre_resids,.,by='userId')
genre_resids = train_set_resid %>% filter(.[,9] == TRUE) %>% group_by(userId) %>%
    summarize('Action resid'=mean(resid),'Action n' = n()) %>%
      full_join(genre_resids,.,by='userId')
```

We then will join these affinities to the train set and test set, based on userId.

```
#Join that residual back to the training data
full_data=full_join(train_set_resid,genre_resids,by='userId')
# Join the genre affinities to the test data. Also add the first prediction
#using movie effect and second prediction using movie and user effect.
test_set_resid <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred1=mu+b_i,pred2 = mu + b_i + b_u) %>%
  full_join(.,genre_resids,by='userId')
```

We can then use these affinities to create a genre-based correction for predicted ratings in the test set. Because our genre affinity may be non-representative if, for example, a user has rated only 1 romance movie, we regularize their affinity by weighing it by a factor of $(1 - \frac{1}{n+1})$ , where n is the number of movies in a particular genre the user has rated. This weighs their affinity between 0 (if n is small) and 1 (if n is large). It could be possible to refine this method, for example by using *(n+m)* rather than *(n+1)*, where m is found in a parametric search using the validation set.

Because movies may have more than one genre, it did not seem prudent to weigh the genre effect more heavily when more genres are listed. Therefore, the weighted affinities for all given genres are subsequently averaged, rather than summed. This is then used to modify the predicted rating.

These genre corrections must be done in a loop for a given pair of user and movie, as shown below.

```
#Generates a correction for a combinate of user + movie, based on the genres of the movie,
#for the test set.
#It will first check which genres are needed to be used for the prediction.
#Then, it computes a genre correction based on the residual for that user
#based on the genre residuals from the train set.
#The desired correction is the average of the weighted residuals for the
#genres of the movie.
  desired_corrections=0
  for (ii in 1:dim(test_set)[1]){
    needed_indices=seq(1:20)[test_set_resid[ii,c(seq(7,26))]==TRUE]
    genre_correct=0
    for (jj in 1:length(needed_indices)){
           genre_correct[jj]=(test_set_resid[ii,((needed_indices[jj])-1)*2+31]*
                               (1-1/(test_set_resid[ii,((needed_indices[jj]-1)*2)+32]+1)))
    }
    genre_correct_mean=mean(genre_correct,na.rm=TRUE)
    desired_corrections[ii]=genre_correct_mean
  }
```

**The results of this final model are discussed in the following section.**

***Unused Approaches***

It was previously stated that a Naive Bayes classifier was used initially when the goal of the project was to maximize accuracy. This approach was inappropriate for minimizing RMSE, however.

The R package "recommenderlab" was explored, including methods such as singular value decomposition (SVD) and user- or item-based collaborative filtering (UBCF and IBCF, respectively). The drawback of this approach was the inability of a home computer to realistically run these algorithms due to memory limitations. A review of literature showed that there exist methods of locally-computed collaborative filtering, but it was beyond the capabilities of the author to attempt to develop such a routine.

Another approach involved installing the Keras front-end into R Studio, which allowed a TensorFlow back-end to run a simple neural network approach to fit predictions to the training data. While the author was able to get their home computer to use the GPU to accelerate the process, fitting a simple neural network still took a prohibitively long time. A simple network showed minimal improvement over the procedure described previously, and more advanced networks were likely to be prohibitively expensive in time to train. Therefore, this approach was abandoned, despite its promising ability to fit based on latent variables (for example: in theory, a neural network could train to learn that a user has an affinity to movies with a twist ending, even without explicitly knowing which movies in particular have twist endings). Refinement of the network architecture could create a model which trains sufficiently quickly on a home computer while still providing improvement to the predictions, and should be considered for the future.

# Results

The final model achieved a RMSE between the predicted and actual ratings of **0.849** on the test set. A table below summarizes the RMSE for each step of the model. A column also includes the percentage of reviews for which the error was reduced when moving from one model to the next (i.e., moving from the average + movie bias + user bias to the final model improved the error for 53.6% of reviews).

```
## Warning in mu + test_set_resid$b_i + test_set_resid$b_u +
## desired_corrections: longer object length is not a multiple of shorter
## object length
```

```
## Warning in test_set_resid$pred2 + desired_corrections: longer object length
## is not a multiple of shorter object length
```

| Model | RMSE | Percent.Improved |
|---|---|---|
| Mu | 1.0600561 | NA |
| Mu + b_i | 0.9430158 | 62.01078 |
| Mu + b_i + b_u | 0.8652710 | 57.14471 |
| Final Model | 0.8494086 | 53.60433 |

As expected, with each refinement, the RMSE decreases, while more than half of the predictions see a reduction in their error. However, both the RMSE decreases and percentages of predictions showing improvement are decreasing with each refinement, as further improvements are more difficult to obtain.

It should be noted that calculating users' affinities is relatively time efficient, even on a home computer. Predicting ratings, however, takes longer: 30 minutes for ~1M combinations. While not prohibitively long, it is likely that there exist more efficient coding techniques to compute a rating given a movie, user, and the user's genre affinities.

The final model $Y_{u,i} = \mu + b_i + b_u + f(genre_i, affinity_u) + \epsilon_{u,i}$ results in a RMSE significantly below the goal threshold, and is considered adequate. The future improvements would likely focus on methods that can use latent variables (such as neural networks) and would require significant increases in computation time.

# Conclusions

The MovieLens 10M data set was used to build a model for predicting ratings for a given movie and user pair. By using a model which incorporated a movie bias effect, a user bias effect, and a genre affinity effect, a final RMSE of the predicted ratings of **0.849** was obtained. This was considered a success based on the project's goal of RMSE $< 0.8775$. Future improvements should focus on using methods which use latent variables to further decrease the RMSE.

# Acknowledgements

I would like to thank *you*, the peer-grader, for using your time to provide feedback on this project. I hope that you also enjoyed this assignment.

I would also like to thank Dr. Irizarry for putting together a wonderful series of courses on EdX, as well as freely publishing the accompanying textbook on GitHub.