MovieLens: Studying the Efficiency of Simple and Complex Modeling in Large, Biased Datasets

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following object is masked from 'package:base':
##
##     date
```

Introduction

MovieLens is a collection of datasets that is pooled from the online movie recommendation service Movielens. The particular dataset being used for this project- 10M MovieLens- is a combination of the subsets of 10,000,000 entries of the ratings and movies dataset. It contains 6 different columns:

userId- Integer; the identifier of an anonymous user that was randomly selected Movielens user

movieId- Numeric; the identifier of the movie being rated

rating- Numeric; the rating of the movie on a 5 star scale with increments of .5

timestamp- Numeric; the amount of seconds elapsed since January 1, 1970

title- Character; the title of the corresponding movieId entry

genres- Character; the genre of the movie

The goal of the project is to successfully create an algorithm- without overtraining- using the edx training set in order to successfully predict the validation testing set. In essence, it is a further performance review of the Movielens service: we can expect users who use the service a lot to have a lot of high ratings, otherwise the recommendation service itself would not be effective. After exhausting neighbor-based and linear regression models, it was decided that a matrix factorization model would be the most effective, least intensive method to predict the validation set.

Methods/Analysis

Code was provided to create a train set called edx- which is a dataframe containing 9,000,055 entries from the 10M MovieLens datasets:

```
dim(edx)
```

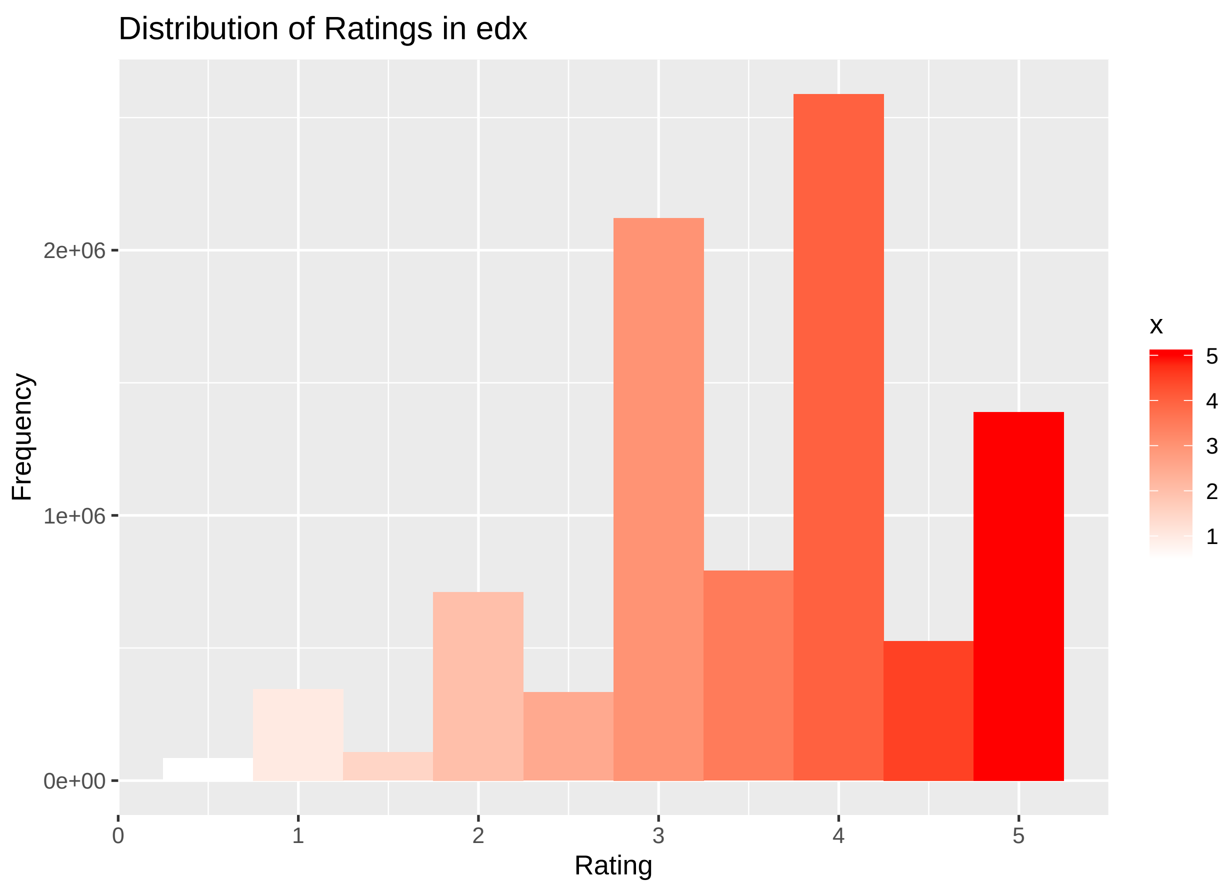
```
## [1] 9000055       6
```

The testing set of 10% of the length of edx was also prepared, which is a dataframe called validation. The code provided ensured that the userId/movieId combinations in the testing set were also included in the training set. This was done to ensure that there was not a significant amount of movies being tested that could potentially harm the evaluation of our algorithm due to erraneous rating/movie combinations (i.e, if a user had 10 contributions to the validation set and had 0 contributions in edx, it would be spliced from validation and added to edx). This was likely performed for cleaning, simplifying the evaluation process.

We then want to see the distribution of ratings in the dataset, just to get an idea of what we're working with:

```
mean(edx$rating)
```

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



The first observation we see is that a majority of the ratings are greater than or equal to 3.0. We can see that this is a heavily skew left distribution, so we can expect ratings to be high. The cause for a large distribution of high ratings is likely due to the fact that people who use a recommendation service like Movielens likely enjoy watching movies, and thus are more likely to consider a movie as mediocre rather than bad and are also more likely to enjoy movies than hate them. The ratings ultimately also speak to the performance of the recommendation algorithm: having a median rating of 3.51 means that consumers must have liked a lot of the recommendations, thus offering higher ratings. Thus, as a whole, we can expect the ratings to be high because this dataset is reflective of Movielens' performance, and should not be completely regarded solely as a list of individual observations, but as a list that demonstrates a company's ability to successfully recommend good movies to a user.

With this observation in mind, we should consider our algorithm method to act as a recommendation service as well: it asks the user how they rate a certain movie, that movie has a title and genres attached to it, and it recommends movies back to that user, fully expecting that over time, the ratings are high. We can attempt to forecast the basic responsibilities of each category in MovieLens:

userId-Will act as the main category; users have different tastes, so their heuristic will be based off their first entry, and the ratings will likely trend upward in theory

movieID-Will act as an identifier tied to the movie name and genre

rating-What we are predicting. We can expect that over time, this number will go up. We may be able to use timestamp as the predictor. We can also expect this to be tied to a combination of genres, certain words in the title, or both.

timestamp-Acts as a sense of chronology; as this increases, ratings should increase.

title-An asset tied to movieID. Can be used in a knn to predict a user's likelihood to like a movie given the key words, but the features would be incredibly difficult to consider in a large dataset compared to using this with individual observations.

genres-An asset tied to movieID. Can be used in a simple knn, as there would only be 18 features compared to using a knn with titles.

What immediately comes to mind is a massive knn algorithm. The algorithm's process will be laid out like this: consider the userID, look at the user's first rating/title/genres, use the title and genres as features in a knn, run a knn through the whole list of movies, set the 200 closest movies as an expected 5.0 rating, next 200 to a 4.5, etc, check to see if the next movie on the user's test set passes the algorithm, modify the features based on a good or bad rating, continue the process till the user has no more entries in edx, restart the process for the next user, and continue till edx is completely evaluated. This explicit method is also known as User-User Collaborative Filtering. Theoretically, this should work. But it fails when a user's first entry is poor, as it means we dont have a heuristic to continue. Perhaps what could work is we look at the user's highest recommendation and work chronologically from there, but that kind of retrospect is unavailable in a true recommendation system. It may be possible though to skip entries until a user has a rating that is higher than 3.5(the average rating for this dataset) and then start the knn algorithm process from there. This would likely work with rare complications, but it seems highly unoptimized, and the evaluations would likely require a lot of training.

We should then consider some of the features that modern recommendation services use: trending films per category and films liked by others with similar tastes as yours. It's possible that we can expect a film that is high rated by a majority to be high rated again by another, and it is possible to use one person's results with similar features (combinations of genres, title words, and ratings) as a heuristic as others: training rating and feature sets for each genre and then predicting ratings for other users using this heuristic. You could view these recently stated approaches as being 2 dimensional as, in a sense, it would be looking across individuals instead of observing each individual in isolation. In theory, this really works and would not be intensive as the first knn method discussed, but optimizing a global heuristic per genre would yield difficulties in predicting ratings for movies that are not popular or for individuals that have differing tastes from popular opinion. This Item-Item Collaborative Filtering approach will likely yield to overfitting and suggest the same shows to many people, which would result in people only watching popular films and would likely over time heavily skew its recommendation system towards recent and popular films.

These approaches seem like they could work to a degree, but let's continue to exhaust our avenues of analysis. Perhaps a strong option would be to consider matrix factorization. Due to the complexity and potential problems we have exhausted, it is likely that we must rely on reduction.

Using matrices with reduction, we can limit erraneous observations (noise), find more hidden trends/observations, and best of all, it won't be as intensive compared to individually training every user like a User-User collaborative fit would yield. We can expect a matrix model to yield good results without overfitting, so let's prepare it. First, to limit erraneous observations, let's filter edx to only consider users with more than 40 reviews:

```
edx_part <- edx%>%
  group_by(userId)%>%
  filter(n() >=40) %>% ungroup()
#Cleans up what we're using
y <- edx_part %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
#Constructs the matrix
rownames(y)<- y[,1]
y <- y[,-1]
title_labels <- edx %>%
  select(movieId, title) %>%
  distinct()
#Makes a vector of movie names
colnames(y) <- with(title_labels, title[match(colnames(y), movieId)])
#Labels each data point
y<-sweep(y, 2, colMeans(y, na.rm=TRUE))
y<-sweep(y,1,rowMeans(y, na.rm=TRUE))
```

```
#Converts points to residuals
```

The code above would be the basic steps to construct a matrix factorization model eventually following the format:

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

The scope of the class did not cover explicit methods to conduct complex parallel matrix factorization, but the *recosystem* package is able to handle these processes effectively and efficiently. Let's make a model using the edx partition and use that algorithm on the validation set:

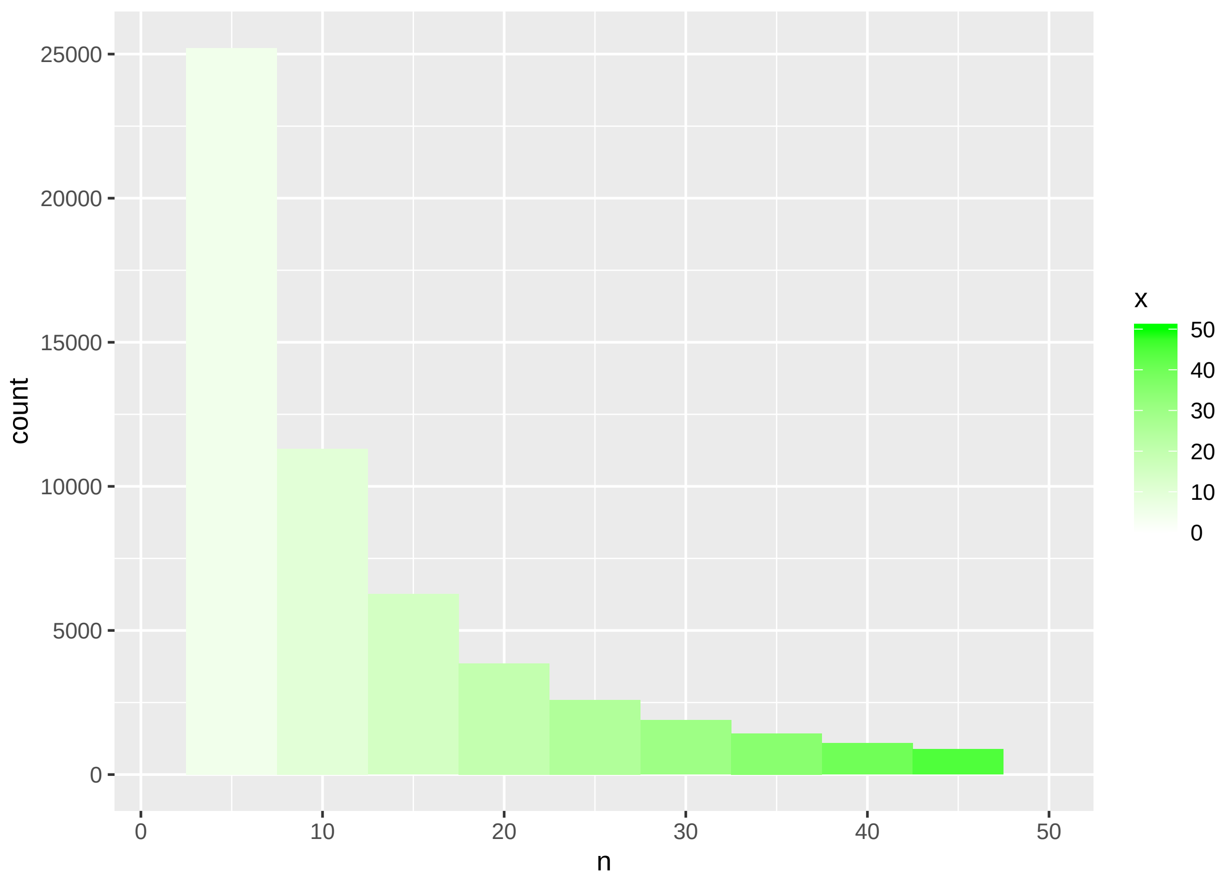
```
library(recosystem)
train_data <- data_memory(user_index = edx_part$userId,
                          item_index= edx_part$movieId,
                          rating=edx_part$rating, index1= TRUE)

test_data <- data_memory(user_index = validation$userId,
                         item_index= validation$movieId,
                         rating=validation$rating, index1= TRUE)
prediction_model <- Reco()
set.seed(3080)
prediction_model$train(train_data,
                       opts=c(dim=35, costp_l2=.1, costq_l2=.1,
                              lrate=.1, niter=300, nthread=8, verbose=F))

validation_prediction <- prediction_model$predict(test_data, out_memory())
```

Recosystem works to optimize the costp and the costq we know in the standard matrix factorization model. The reason why that is critical in this process is because we have a lot of sparse data; a simple linearized model would be unable to effectively handle it. Parallel matrix factorization is critical in order to limit the effect of these biases by scoring, weighting the data over and over, limiting the costs through parallelized training.

Before evaluating the effectivity of our matrix factorization model, let's consider the structure and distribution of the validation (testing) set:



We quickly see that most users have 1-5 entries, but there are users that have over 50 entries. In fact, we can see that the highest number of entries for a single user is 743:

```
max(entryCounts$n)
```

```
## [1] 743
```

With these at hand, let us look at our model's performance review.

Results

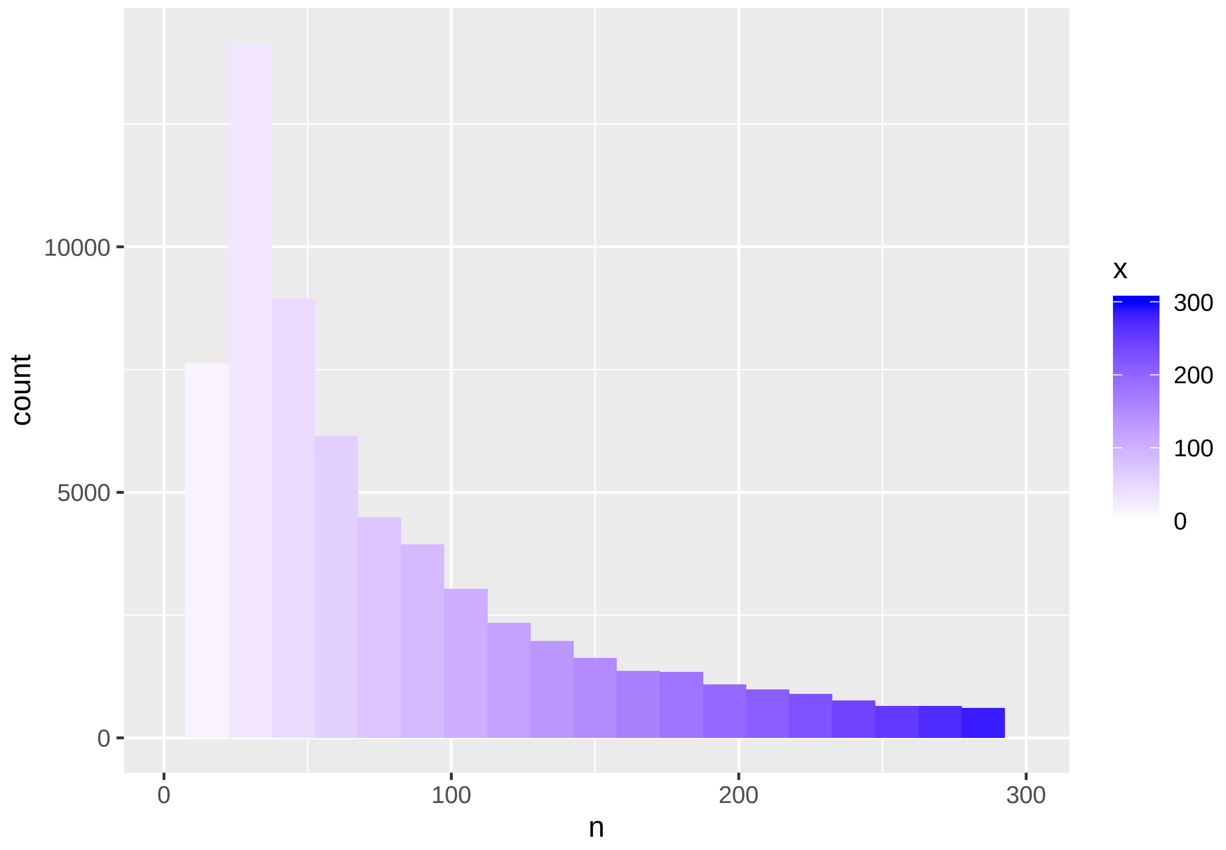
Let's calculate the RMSE with caret:

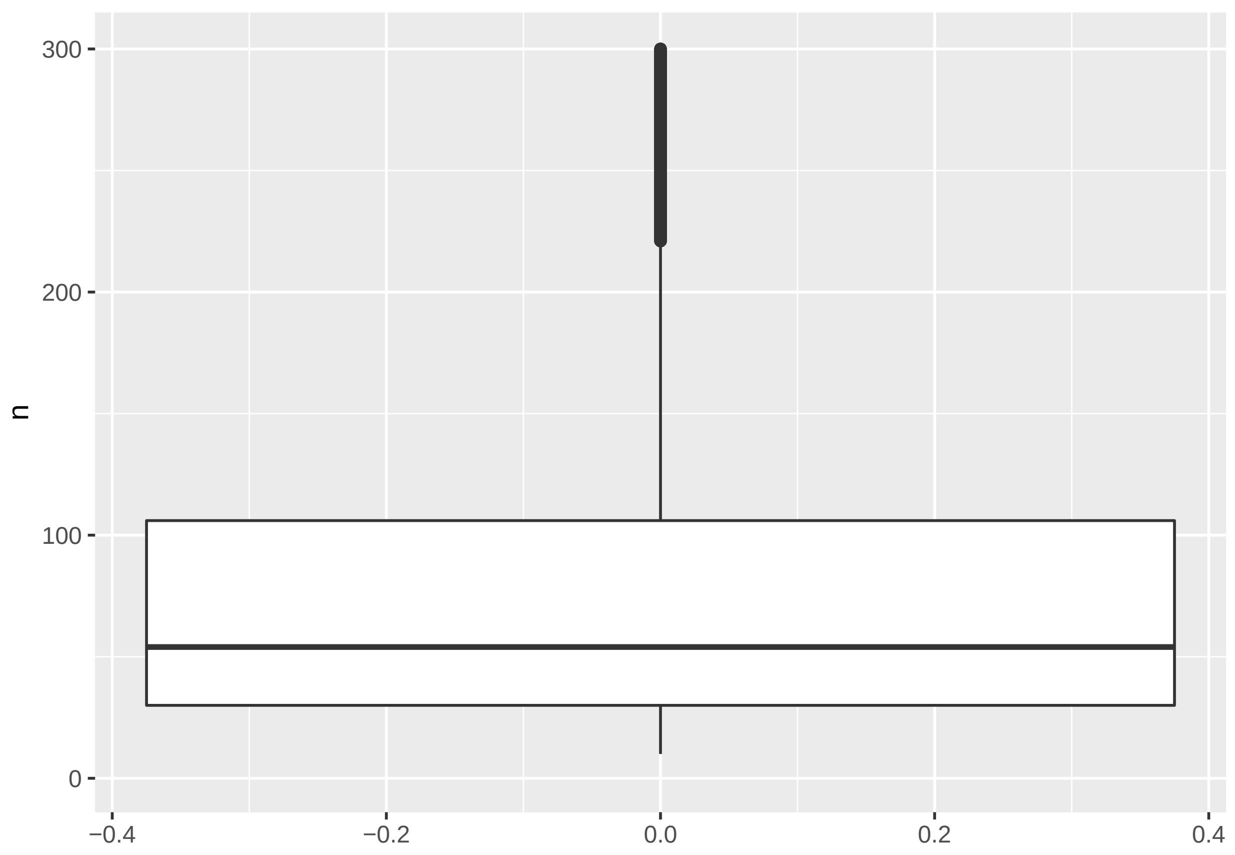
```
RMSE(validation$rating, validation_prediction)
```

```
## [1] 0.829645
```

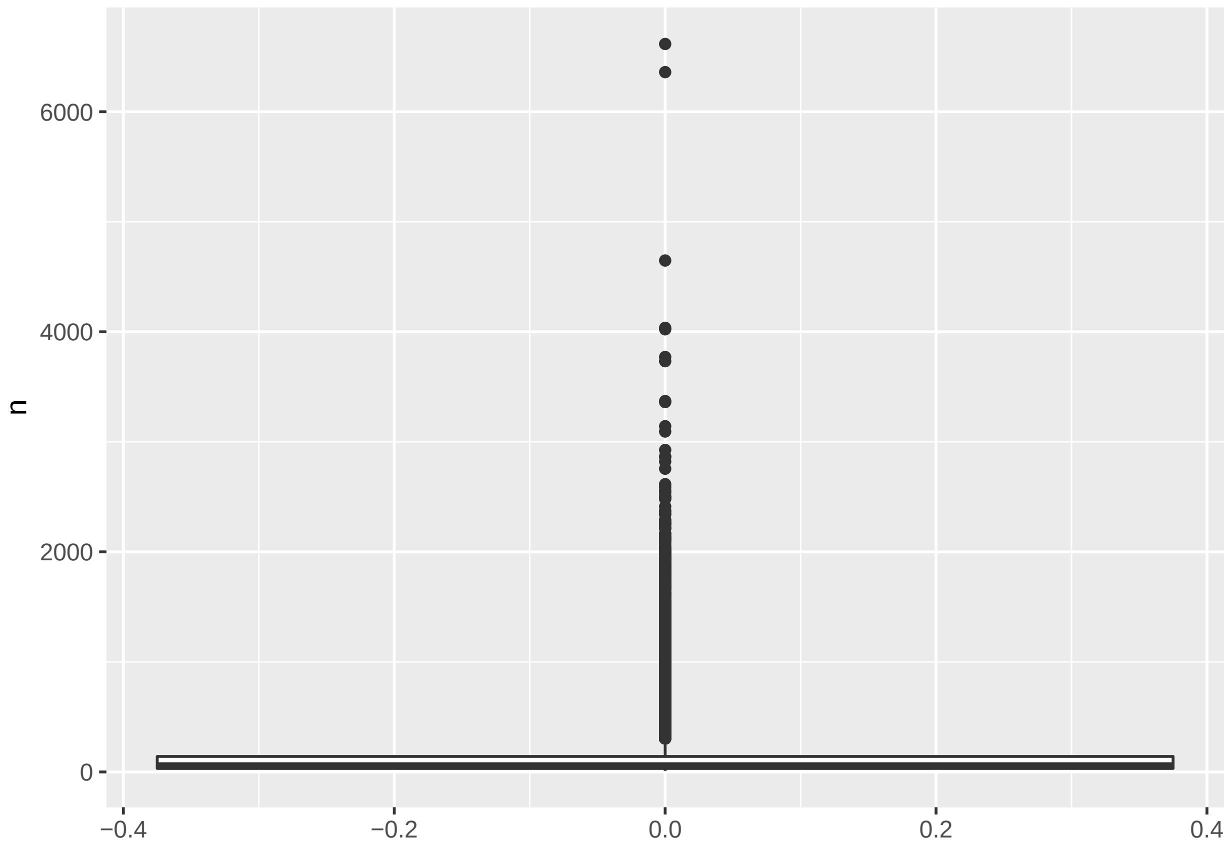
Remarkable! We achieved an RMSE of approximately 0.83. In the textbook, the linear regression model variations had RMSE's of approximately 0.9. This decrease in RMSE is expected because of the nature of matrix factorization; its complexity and efficiency is nearly uncontestable.

As regarded in the analysis section, item-item collaborative filtering and user-user collaborative filtering would have encountered many problems in this dataset, the main issue being too much data. At 10 million datapoints, the amount of processing required to establish decent heuristics and having to predict using globalized heuristics is clearly ineffective with so many opinions to consider. In the validation set, the majority of users only had 1-10 entries. This means that the algorithm would have been forced to predict a user's tendencies based off an incredibly small sample. Recall that earlier, a decision was made to only include users that made over 40 reviews. Let's retrospectively analyze that decision. Here is a histogram and a box plot of the number of entries per user in edx:





At first glance, the histogram makes it appear that we made a questionable decision, but looking at the bar graph, it is evident that we considered a bit over 50% of the majority of the data, and that's not inclusive of several outliers that we can see here:



Some users had thousands of entries. Clearly, their opinion dominated the algorithm. In essence, this is a fair evaluation despite the appearance of the histogram because the service's algorithm itself was likely affected by the users with thousands of entries. As regarded early in this report, this project is looking at data from a pre-existing movie recommendation service. This means Movielens' algorithm also had to be developed and trained over time, and clearly users with thousands of ratings have used the service plenty and thus impacted it the most. Therefore, the replication of the Movielens algorithm should actually mostly consider these high values of n>40 more than the approximately 30-40% of data that is n<40.

Conclusion

Conclusively, a matrix factorization model that considered users with more than 40 entries/reviews was very effective, as intuitively speaking, users with the most reviews formed the algorithm the most. Because we concentrated on those figures, we were able to closely replicate the algorithm that produced the actual data moreso than a theoretical algorithm that considered more of the right skew of data that was made available.

Further study of this dataset would be to actually replicate the true method that Movielens uses. It would be interesting to see if those numers would be truly replicable in a simulation. Clearly, by building our own models, we are establishing our own sense of effectivity and recommendation, and one could consider the expectation of having an excellent RMSE with a poorly developed/dissimilarly developed algorithm as ridiculous, or even impossible. This project's results in RMSE- in a larger lens- showed us how to build a good model that closely resembles/fits the Movielens algorithm itself. Likely, that is the important takeaway, and it goes back to the basics of statistics: an individual studying data is limited to report using the source of that data. This project was moreso of a practice of how well or how poorly certain machine learning techniques fare under conditions with high latent content, biases, and massive amounts of data to consider and manipulate.