MovieLens Project PH125.9x

Martin Schiff

January 6, 2019

Introduction

In this project, we are asked to create a movie recommendation system. Specifically, we are to predict the rating a user will give a movie in a validation set, based on a given set of users and movie ratings. The prediction will be judged on the raw accuracy—i.e. the percentage of predicted ratings exactly equal to the true user rating. This is a somewhat different goal than the usual task of minimizing predicted error or selecting items a user is most likely to rate highly. Only exactly correct predictions are considered accurate.

The provided data is from the MovieLens 10M set (i.e. 10 million ratings), a much larger version of the data set contained in the `dslabs` library used during the Recommendation Systems portion of the course. The given data set `edx` is approximately 9 million records long and contains the following features:

```
names(edx)
```

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

while the test data set `validation` is roughly 1 million records long with the true `rating` column omitted, since it is what is to be predicted by the model.

Allowable ratings (and all of the ratings in the two provided sets) are from 0.5 to 5 in steps of 0.5, which could alternatively be considered a 10-step ordered classification.

The prediction algorithm used in this project generally follows the simple model used in the course, judging the "bias" or difference from the mean for each user, item, and genre and implementing a regularization to discount extreme, occasional values. While other algorithms may have proved more accurate, they were either beyond the scope of the course content or computationally prohibitive on this large set of data (particularly since the analysis needs to run on an unknown computer for peer grading). The course approach has the advantages of being fast, easily scalable, and simple to modify.

The task of determining discrete, half-to-5-star ratings from the real number prediction allowed for significant improvements over simple rounding, and is the focus of the novel work discussed in this report.

Methods & Analysis

As with the `dslabs` MovieLens set, the provided data is already well organized and in a clean, usable format. Before we begin exploring the data and assembling a model, we set up some helper functions to produce and evaluate our discrete prediction ratings.

Helper Functions & Data Preparation

While this analysis will make use of the root mean-square error (RMSE) and associated `caret` functions, the final judgement of our prediction will be based on true accuracy. It also requires discrete rating predictions to the nearest half star if we are to match the rubric of the true ratings. We set up the following helper functions to enable this.

```
# A function to calculate accuracy in terms of % exactly correct
accuracy <- function(true_ratings, predicted_ratings) {
  correct <- sum(true_ratings == predicted_ratings)
  return(correct / length(true_ratings))
}
```

```r
# A general function to discretize ratings vector with optional Whole flag vector for integers only
# The extra 0.01 additions are due to IEEE rounding, so we can be sure 0.5 always rounds up
flixStar <- function(ratings, whole = FALSE) {
    map2_dbl(ratings, whole, function (a, b) {
      if (a <= 0.5) a <- 0.51 else if (a > 5) a <- 5
      if (b) round(a + 0.01) else round(a*2)/2
    })
}
```

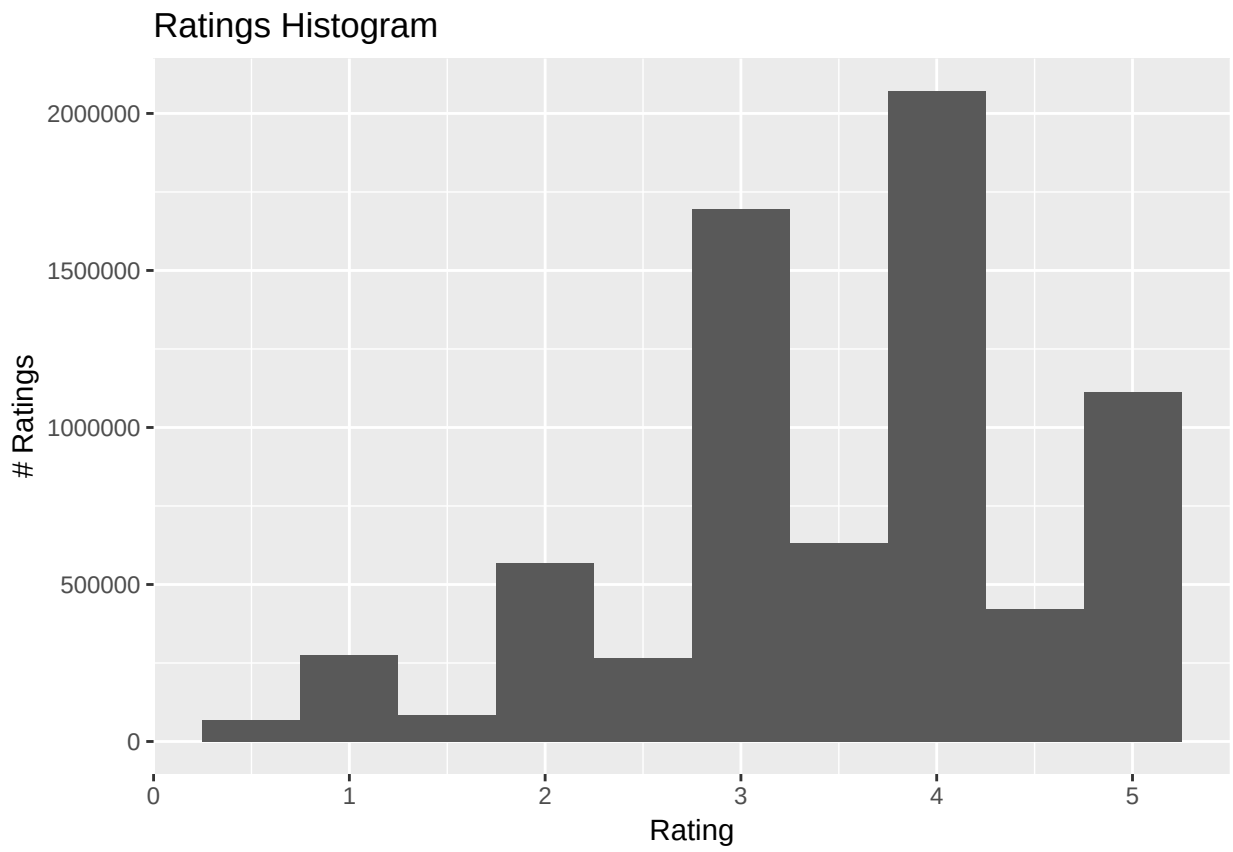
The benefit of the `flixStar()` function having the option of returning ratings discretized to either half or integer numbers will be motivated below. The additional 0.01 factors are to account for IEEE rounding behavior, which would otherwise round 0.5 to the nearest even number.

Finally, to enable learning and evaluation without the use of the `validation` set, we split the `edx` set into its own `trainSet` and `testSet` components, using the same method as in the provided `createDataSets.R` script.

Data Exploration

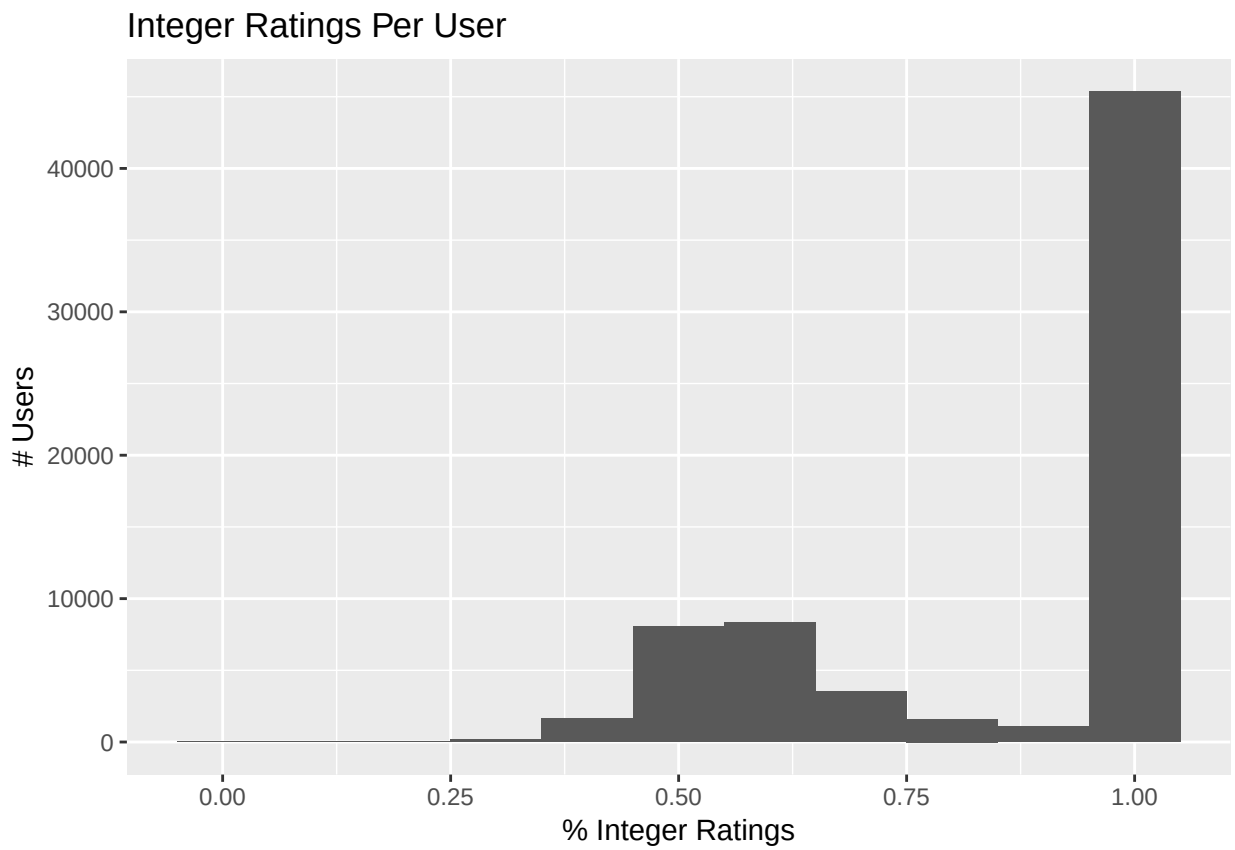
The breadth and distribution of the movies, users, ratings, and similar features were explored in the `dslabs` data set in the coursework, and are similar here. Some additional insight into the ratings distribution was gained and enabled a significant boost in prediction accuracy.

The overall distribution of ratings in `trainSet` is similar to that from the coursework:



with a median rating of 4 and a significantly higher proportion of integer versus half ratings (0.795).

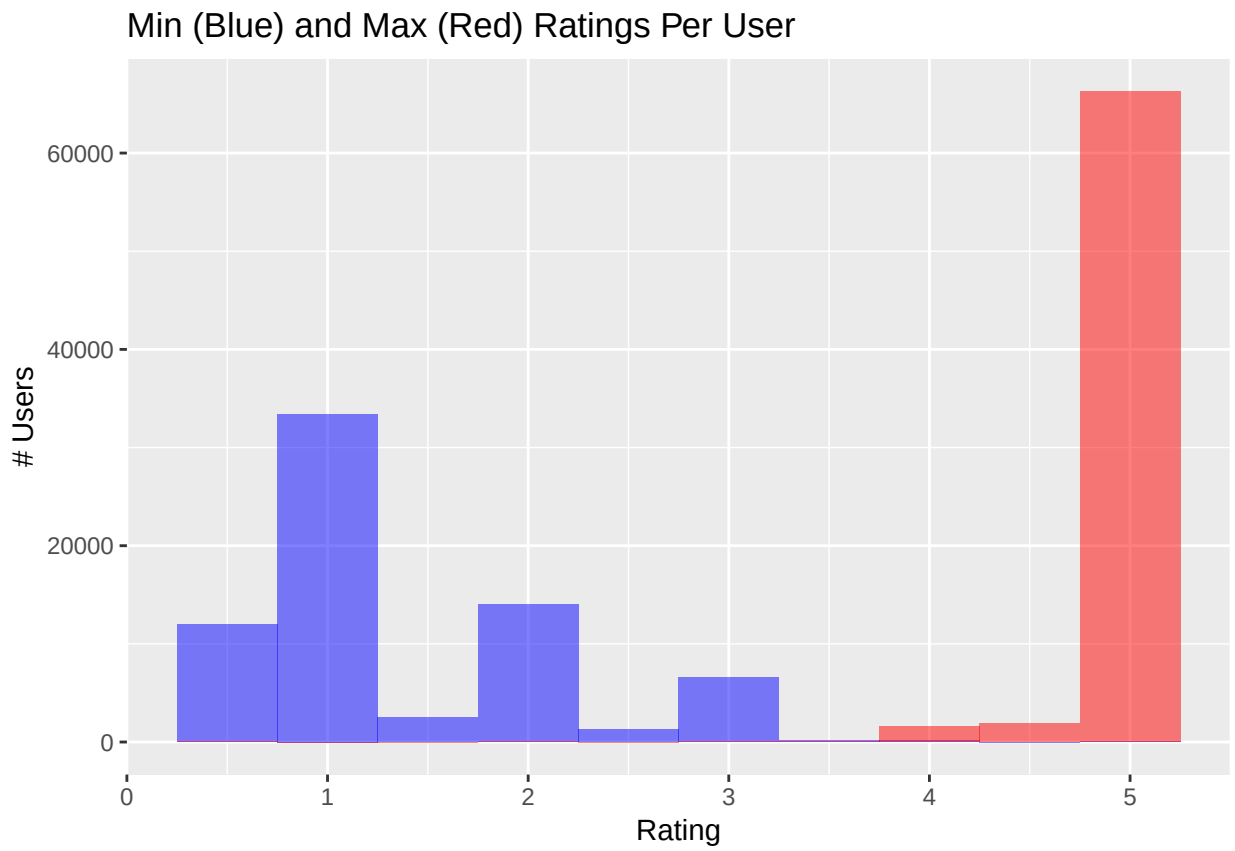
Grouping on a user-by-user basis, we also see that very many users only assign integer-valued ratings:



With this insight, we can tag certain users as preferring whole number ratings, and ensure that predictions for these users are rounded to an integer. While many users assign integers 100% of the time, setting this to a lower percentage captures many more ratings. An initial value of 75% is used here, and this percentage is tuned below to higher accuracy at an even lower proportion.

```
# We want to tag certain users as always or nearly always assigning whole number ratings
# We will tune the "nearly always" cutoff later.  75% for now
wholeCutoff <- 0.75
usersWhoWhole <- trainSet %>% group_by(userId) %>%
  summarize(total = length(rating),
            wholes = sum(rating %% 1 == 0),
            wholepct = wholes/total) %>%
  filter(wholepct >= wholeCutoff) %>%
  .$userId
```

Similarly, some users do not assign the entire range of ratings. While nearly all users assign ratings up to and including 5, some do not ever assign the lowest ratings:



Model Development

The model used for developing the prediction algorithm follows that from the course: the mean rating μ is modified by one or more "bias" terms b with a residual error ε expected.

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

To begin this development, we apply the `accuracy()` helper to the mean and median of our test set. Since the mean is not generally an integer, it is fed through our rounding helper function `flixStar()`.

```
# overall mean for the whole set
mu <- mean(trainSet$rating)
mean_acc <- accuracy(testSet$rating, flixStar(mu))
mean_RMSE <- RMSE(testSet$rating, mu)
mean_results <- data_frame(Method="Mean Alone", Accuracy = mean_acc, RMSE = mean_RMSE)

# overall median for the whole set
med <- median(trainSet$rating)
median_acc <- accuracy(testSet$rating, flixStar(med))
median_RMSE <- RMSE(testSet$rating, med)
```

Method	Accuracy	RMSE
Mean Alone	0.0881544	1.059904
Median Alone	0.2874054	1.166678

Interestingly, while the mean has lower RMSE, the median has much higher accuracy. This stands to reason since the median by definition captures the maximum number of true ratings.

Next, we center ratings on the movie average to determine movie bias b_i, on each user's average to determine user bias b_u, and on each genre "combo" average to determine genre bias b_g. This is because the `genres` feature includes concatenated tags for all genres that apply to the movie (e.g. "Comedy|Drama|Romance"). For current purposes, no attempt was made to split these into individual effects, and each combo was treated as its own genre.

During development, adding this genre effect provided only a minor improvement over movie + user factors alone. These intermediate steps are not detailed here. Splitting the genre into individual tags may have yielded more significant effects.

In addition to calculating the accuracy of the prediction rounded to the nearest half-star, we use the list of integer-preferring users `usersWhoWhole` to restrict that subset of users to whole number predictions. This yields a significant boost in accuracy.

```
# Center on user/movie/genre-combo
movie_avgs <- trainSet %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

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

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

predicted_ratings <- testSet %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  mutate(roundPred = flixStar(pred, userId %in% usersWhoWhole))

mean_umg_acc <- accuracy(testSet$rating, flixStar(predicted_ratings$pred))
mean_umg_accWholes <- accuracy(testSet$rating, predicted_ratings$roundPred)
mean_umg_RMSE <- RMSE(testSet$rating, predicted_ratings$pred)
```

Method	Accuracy	RMSE	Accuracy_Whole
Mean Alone	0.0881544	1.0599043	NA
Median Alone	0.2874054	1.1666776	NA
Movie + User + Genre	0.2492480	0.8655941	0.3590857

Regularization

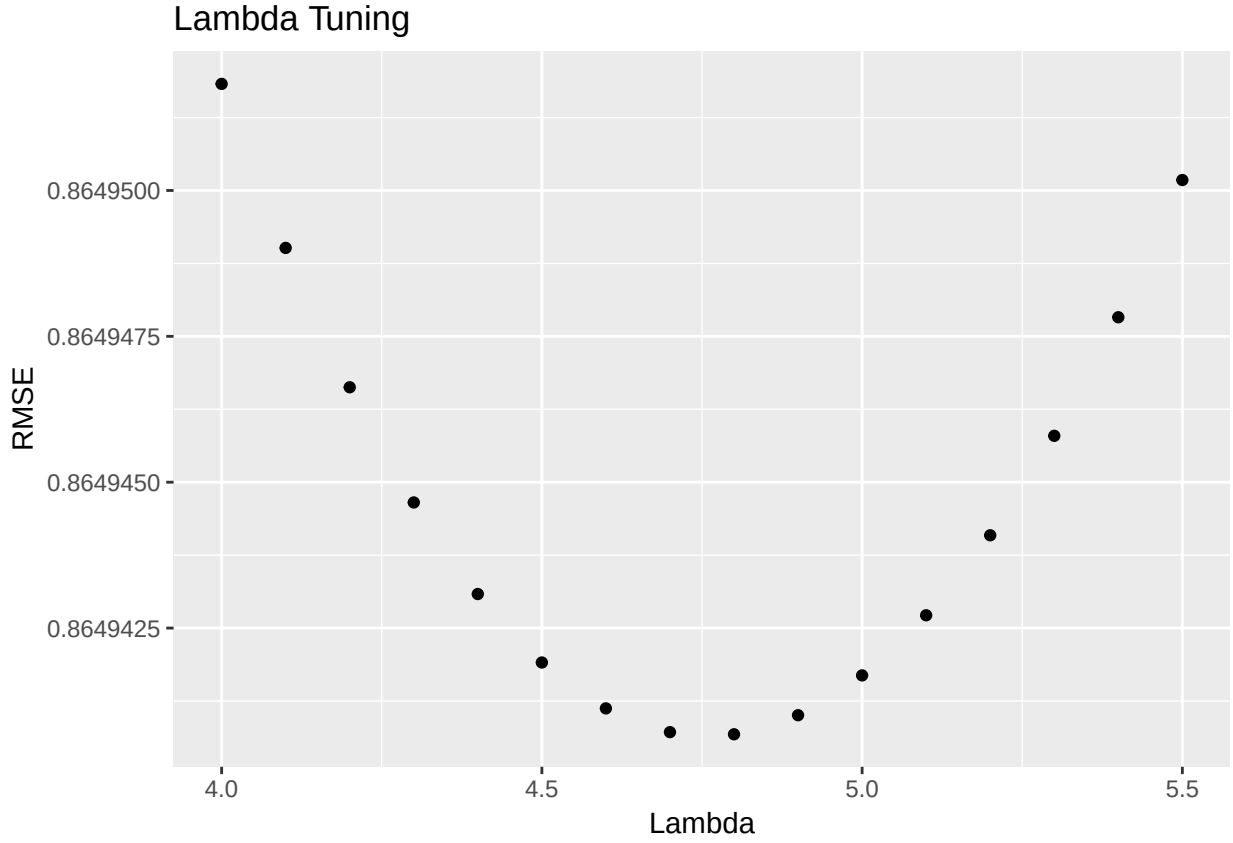
Regularization penalizes records which stray far from the mean and have few associated ratings, such as an obscure film with only a few very low ratings. Following the derivation in the course, we can select the bias values using a regularization factor λ as follows:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{i=1}^{n_i} (Y_{i,u,g} - \hat{\mu})$$

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + n_u} \sum_{u=1}^{n_u} \left(Y_{i,u,g} - \hat{b}_i - \hat{\mu}\right)$$

$$\hat{b}_g(\lambda) = \frac{1}{\lambda + n_g} \sum_{g=1}^{n_g} \left(Y_{i,u,g} - \hat{b}_i - \hat{b}_u - \hat{\mu}\right)$$

This regularization is tuned by running over a sequence of λ values and selecting the best RMSE result:



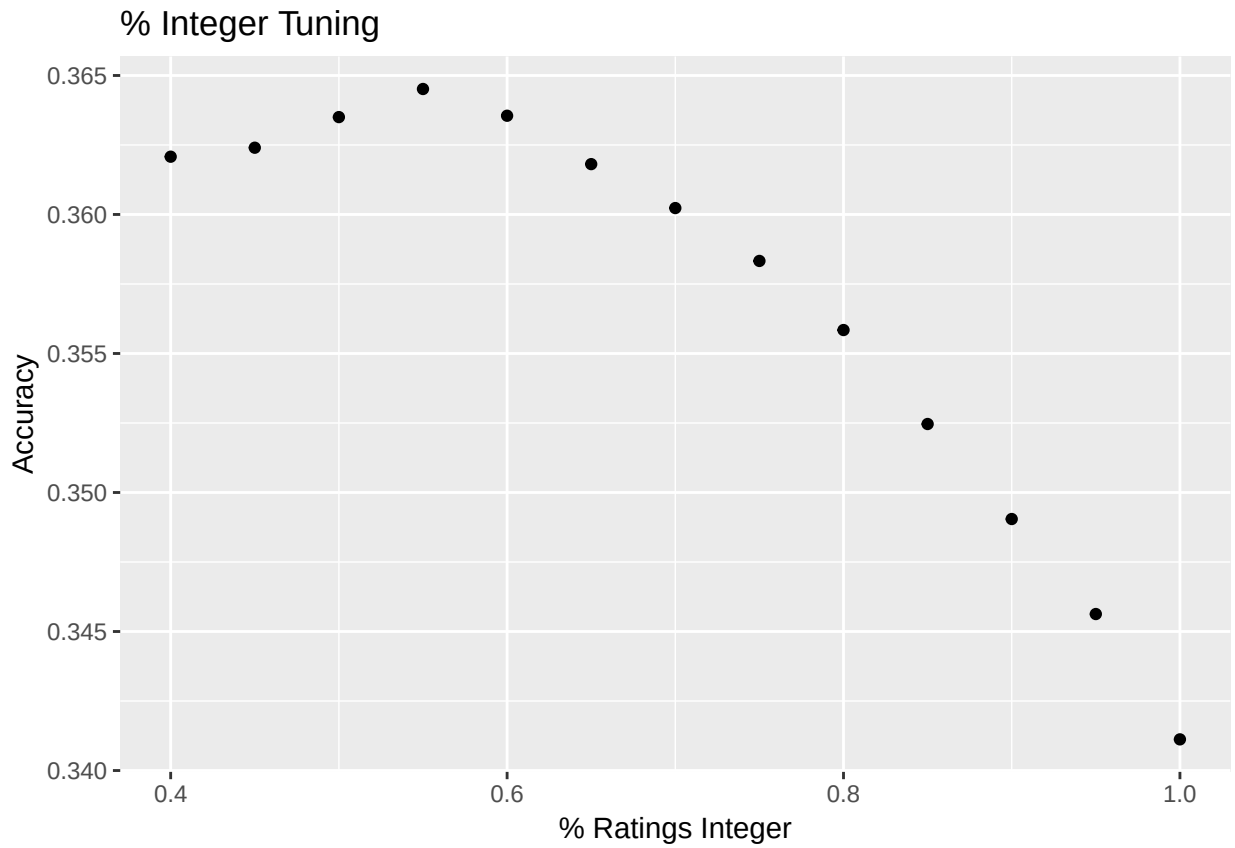
The tuned value of $\lambda = 4.8$ improves the un-rounded RMSE substantially, but with little effect on accuracy. The regularization effect may be getting lost in the rounding to mostly integer ratings.

Method	Accuracy	RMSE	Accuracy_Whole
Mean Alone	0.0881544	1.0599043	NA
Median Alone	0.2874054	1.1666776	NA
Movie + User + Genre	0.2492480	0.8655941	0.3590857
Regularized Movie + User + Genre	0.2485786	0.8649407	0.3583290

Final Model Tuning

With this prediction model in place, we turn back to the strategy we use to determine the discretized ratings. When we noticed that most users assign only integer ratings, we considered also including users that "mostly"

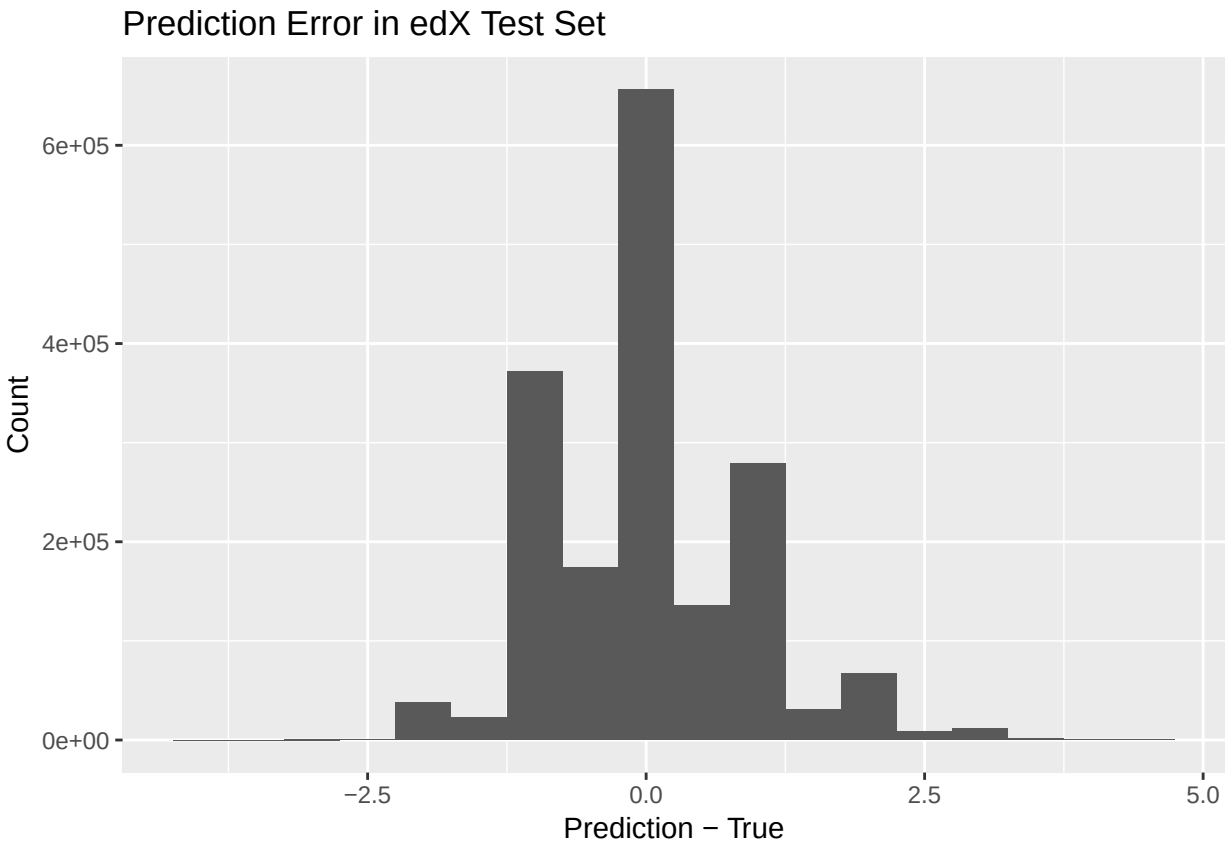
prefer integer ratings. We tune the proportion of "mostly" here.



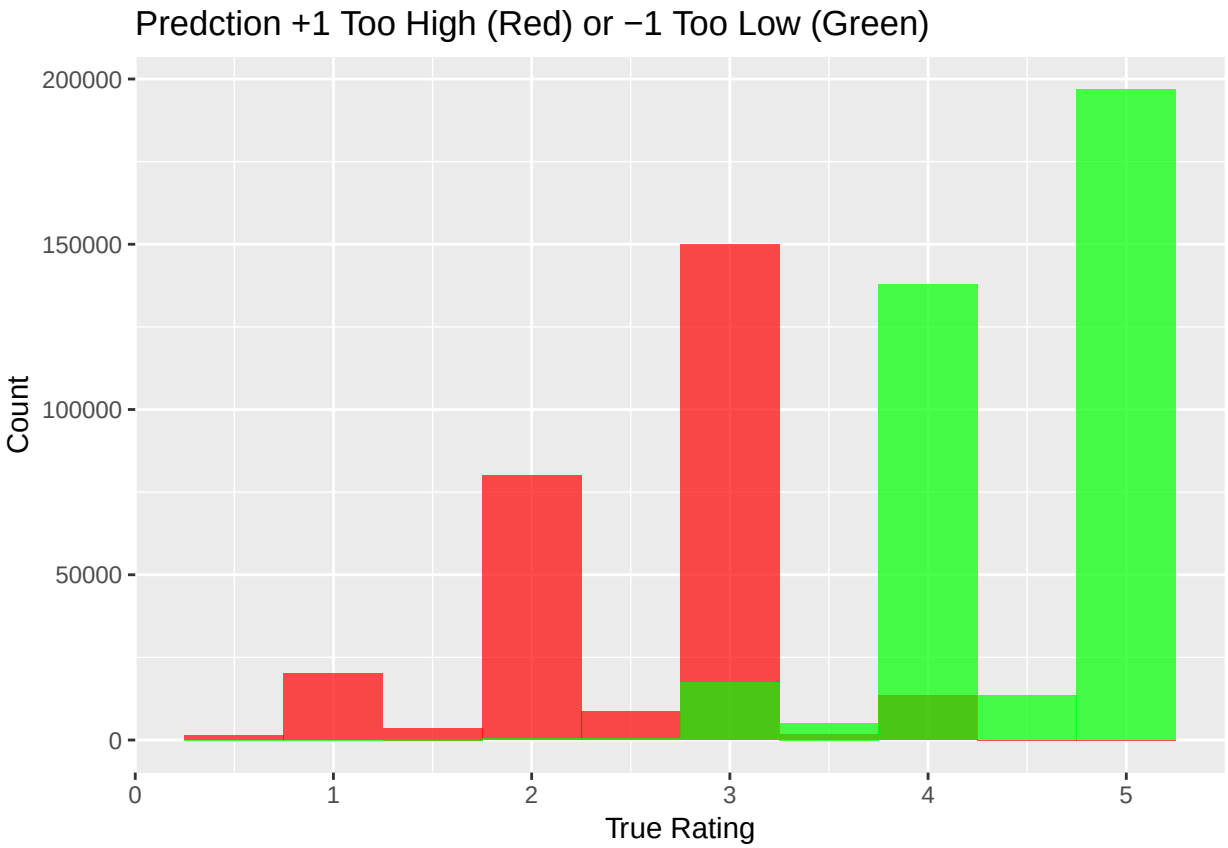
As shown above, assigning only integer ratings to users who use them at least 55 percent of the time in the training set provides the best result.

Method	Accuracy	RMSE	Accuracy_Whole
Mean Alone	0.0881544	1.0599043	NA
Median Alone	0.2874054	1.1666776	NA
Movie + User + Genre	0.2492480	0.8655941	0.3590857
Regularized Movie + User + Genre	0.2485786	0.8649407	0.3583290
+ Integer Tune	NA	NA	0.3645136

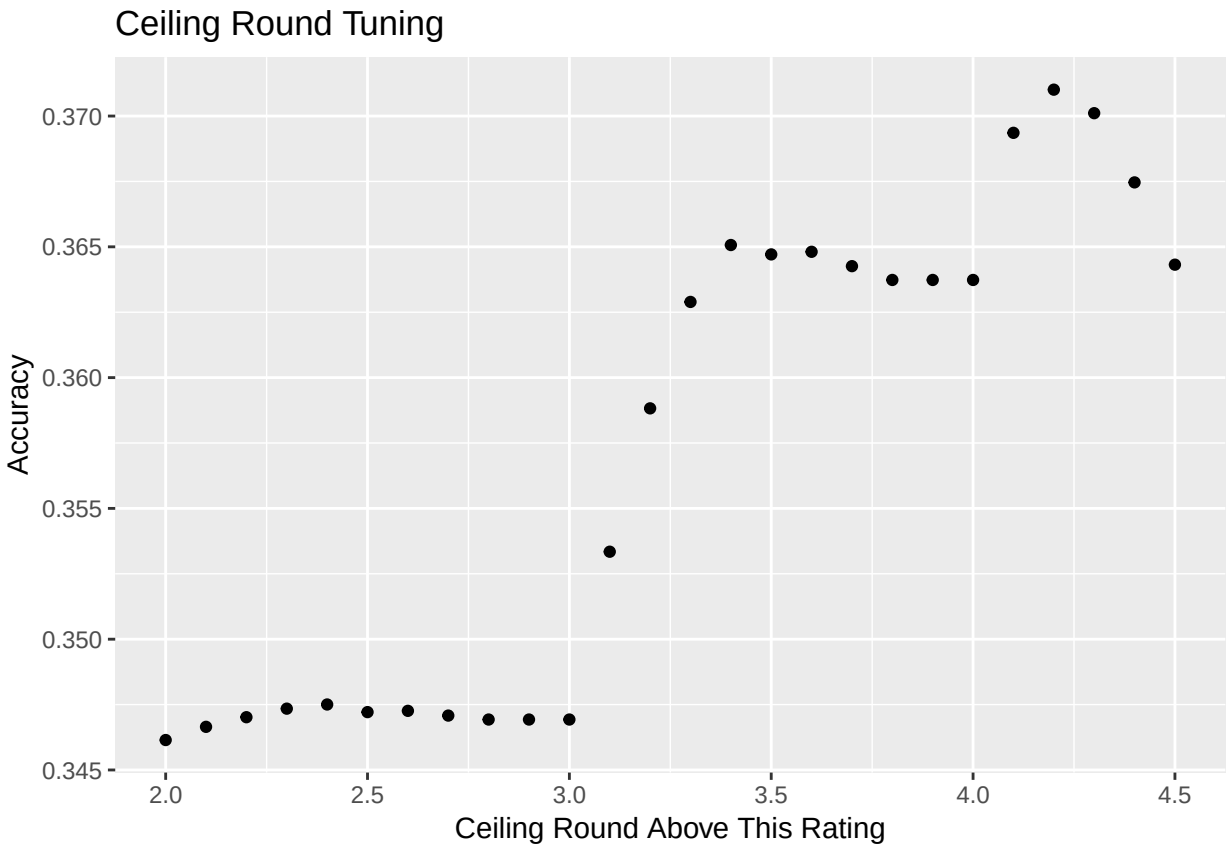
Looking at the raw prediction error (predicted rating minus the true rating) in the test set, we find the largest group of incorrect predictions at +1 and -1 from the true:



Isolating this group of +1/-1 errors, we find that a the largest group is predicted 4s that were actually 5s:



This motivates another adjustment to our rounding algorithm to make it more optimistic: perhaps instead of rounding predicted 4.5+ ratings up to 5, a lower cutoff should be used for this rounding. Perhaps a `ceiling()` rounding should be used above a certain "optimist" cutoff: users may be inclined to assign a 5 rating to anything that seems better than a 4, because it is a nice thing to do. This cutoff between traditional and ceiling rounding in our `flixStar()` function is tuned below.



Predicted ratings above the cutoff of 4.2 are now ceiling rounded. This adjustment to the rounding algorithm improves accuracy further.

Method	Accuracy	RMSE	Accuracy_Whole
Mean Alone	0.0881544	1.0599043	NA
Median Alone	0.2874054	1.1666776	NA
Movie + User + Genre	0.2492480	0.8655941	0.3590857
Regularized Movie + User + Genre	0.2485786	0.8649407	0.3583290
+ Integer Tune	NA	NA	0.3645136
+ Ceiling Tune	NA	NA	0.3710087

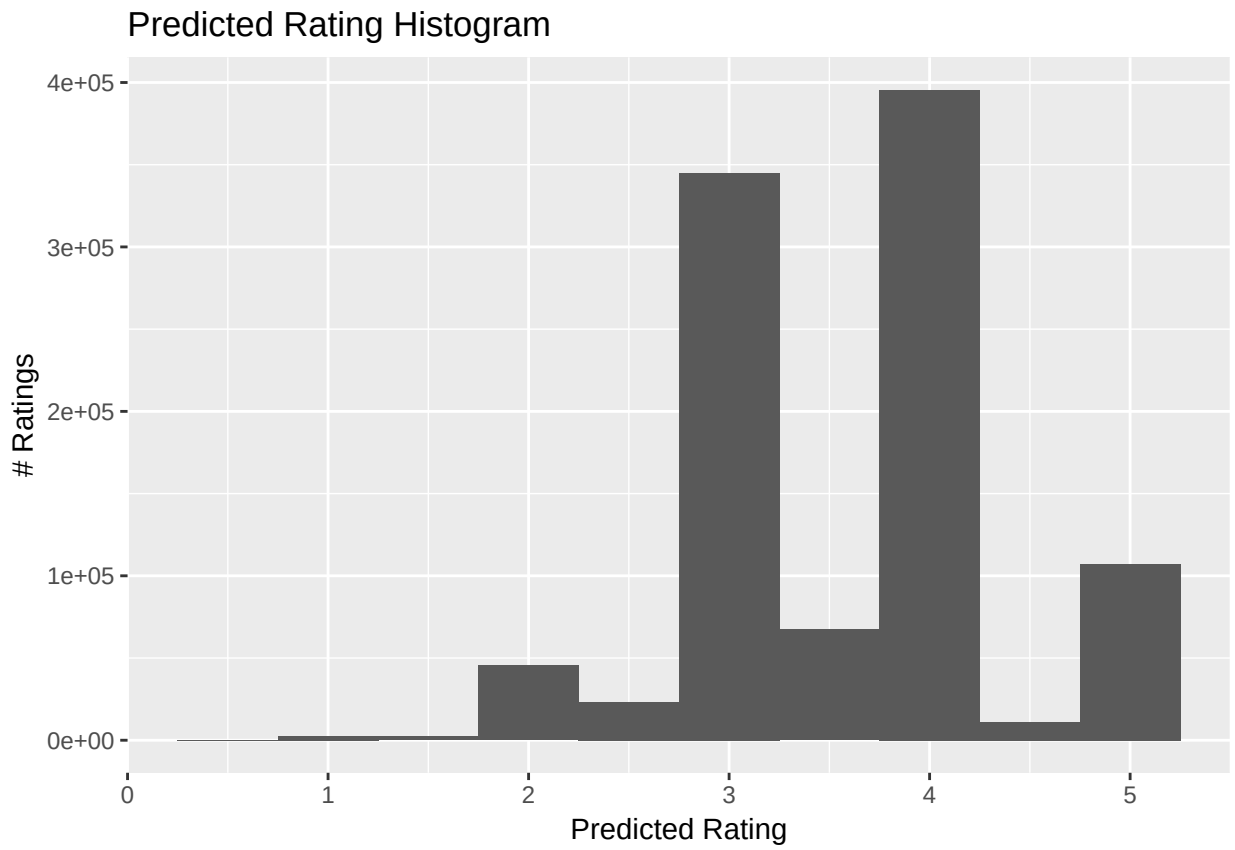
Other Methods

More complex methods were explored for this model using the `recommenderLab` package, which implements a matrix-based data type for the user/movie/rating data sets. A user-based collaborative filtering (UBCF) approach was tested on the much smaller `dslabs` MovieLens set. Even on this much smaller set, the computation time was prohibitive, and did not yield a substantial improvement in the rounded accuracy. While some parameter tuning may have improved this, the use of this package and method on the full 10M data set would not be feasible, nor a realistic task for a peer grader to re-run.

Results

As shown above, manipulation of the rounding method and parameters yielded a final un-rounded RMSE of 0.865 and rounded accuracy of 0.371 on the test set. While this accuracy does not meet the minimum threshold of 0.50 established in the evaluation rubric for the project, it is a significant improvement over the initial model as used in the course.

While we do not have access to the true ratings in the `validation` set until grading, running the final algorithm on this set yields the following rating distribution:



The distribution appears similar to that in the training set, particularly near the median rating of 4. Extreme values do not match as closely.

Conclusion

In this project, we implemented the recommendation model concept profiled in the course, with the modified goal of predicting exact, discrete half-star ratings. In the process of converting predicted ratings to these discrete ratings, significant insight was gained into user rating assignment, allowing improvements to the rounding algorithm.

Avenues for refinement of the current work include further exploration of the genre tags applied to each movie, perhaps implementing scores/bias for each individual genre. Similarly, implementing a method of genre preference per-user (rather than for all users rating a genre) would provide a more realistic effect of how each user tends to rate specific genres of movie.

Based on the current results and the brief exploration of the UBCF recommender algorithm, it is likely that much more sophisticated (and computationally intensive) methods would be needed to achieve high levels of accuracy exceeding 0.5. For the purposes of this project and given the very large size of the data set, this is left to future work.