MovieLens Capstone

Christopher C. Smith

1/15/2022

Designing a Movie Recommendation Engine

Introduction

The objective of this project is to build a Netflix-style recommendation system that can predict a user's star rating for a movie (from 0.5 to 5.0) with a residual mean squared error (RMSE) of less than 0.86490. Since Netflix doesn't make its data available, we use the MovieLens 10M data set, which is freely available online at https://grouplens.org/datasets/movielens/10m/. According to the data set summary, "This data set contains 10,000,054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users of the online movie recommender service MovieLens. Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided." For the purposes of this study, I will use the movie and ratings data while ignoring the user-applied descriptive tags.

After downloading the data, I perform some rudimentary cleaning steps and merge the ratings.dat and movies.dat files into a single data frame with the columns userId, movieId, rating, timestamp, title, and genres. (The "title" variable takes the format "Boomerang (1992)"; it's possible to extract the release date to create an additional variable, should I wish to do so.) The "genres" variable is a single character string containing all genres into which a given movie fits, separated by "|", e.g.: "Comedy|Romance". We randomly partition the data into a training set, which we will use for building our model, and a validation set, which we won't look at or touch until it comes time to test our completed model at the end. (To prevent overfitting, we will follow the cardinal rule of machine learning: do not use validation data to make any modeling decisions!) Our training set includes 90% of the data, while our validation set includes 10%. For reproducibility, we set the seed to 1. We then run a check to ensure that all movies and users in the validation set also appear in the training set. If any do not, we transfer them into the training set.

To allow for testing and choosing between models, we repeat this process to further partition our training data into train and test data sets. We again use a randomized 90-10 partition and set the seed to 1.

To measure the accuracy of our predictions, we will use residual mean squared error, defined as the square root of the mean of the squared differences between our true ratings and our predicted ratings. To calculate residual mean squared error, we define the following function:

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

The most accurate model will be the one the minimizes RMSE. As mentioned above, we are targeting an RMSE of less than 0.86490 when testing the final model on the validation data set.

Achieving this target RMSE proves to be fairly trivial; it isn't even necessary to take into account any pairwise correlations between movies or individual user preferences (apart from average user rating). We can get there with a simple model that uses just five averages and a statistical technique called regularization. The five regularized averages used in the final model are: overall average rating, average rating for each movie, average rating for each user, average rating for each combination of genres, and average rating by binned date (which variable I will tune to find the optimal bin size).

Methods

I build two models. The second model builds on the first, so that my final model is highly accurate.

In my base model, I examine in sequence the overall average rating, the average rating for each movie, the average rating for each user, the average rating for each combination of genres, and the average rating by binned date (which variable I “tune” with 5-fold cross-validation to find the optimal bin size). In each case, I calculate the average, use it to predict ratings on the test set, and then subtract it from the data, leaving “residuals” from which to calculate the next average. By this process, I build a cumulative model that beats my target RMSE when predicting ratings for my test set, but with an insufficient margin of safety.

(When making predictions with this and subsequent models, I disallow predictions above the maximum possible rating and below the minimum possible rating. The range of ratings in my data is 0.5 to 5, but a little data exploration reveals that MovieLens did not begin to allow half-star ratings until 2003. The first half-star rating in my training set is time stamped 2003-02-12 17:31:34. Therefore, whenever making predictions with one of my models, I disallow predictions below 0.5 or above 5 for any rating made after 2003-02-12 17:31:34. For ratings prior to this time, I disallow any prediction below 1 or above 5.)

In my regularized model, I refine my base model using a statistical technique called regularization. Recognizing that that some of my averages are calculated for a small sample size (i.e. for bins that contain a small number of data points), I “punish” these small-sample averages by regressing them toward the mean. A regularized average is calculated as the sum of observations divided by the sum of sample size plus a correction factor lambda. In each case, I “tune” lambda by finding the value of lambda that minimizes RMSE in 5-fold cross-validation. The resulting model outperforms the base model for predicting ratings on my test set and again beats my target RMSE, this time with a good margin of safety. In an attempt to refine this model, I also try a smoothed loess function rather than a binned average for my date variable. I tune the function’s smoothing span with 10-fold cross validation. Unfortunately, this approach slightly underperforms the regularized average, so I reject this refinement and proceed to validation.

Base Model

For my first attempt to build a model, I first find the average rating for each user, mu = 3.5124556. Using this value to predict all ratings in the test set gives me an RMSE of 1.0600537. This is my baseline. In calculating all subsequent averages, I will first subtract or “sweep out” mu from all ratings in the training set. This converts my ratings to a set of positive and negative “residual” values with a mean of zero. Positive values represent a rating above mu, and negative values represent a rating below mu. A residual value of exactly zero represents a rating of exactly mu.

Some movies are better than others. Consequently, different movies have different average ratings. To account for this, I group the data set by movie ID and find the mean “residual” rating for each individual movie, b_i. For instance, my residual average for movie ID #1 is 0.415004, meaning that this movie’s average rating is 0.415004 above mu. I use this formula to predict ratings in my test set.

After thus using my model to predict ratings for the test set (and converting all out-of-bounds ratings to 0.5, 1, or 5), I calculate the RMSE for this cumulative “Naive average + movie average model”: 0.9429615. This is a very significant improvement over using just our naive average, mu. To prepare to calculate our next set of averages, we “sweep out” each movie’s residual average by subtracting it from all ratings for that movie. This leaves us with another set of “residual” ratings, with a mean of zero for each movie.

Some users have a naturally positive temperament and give the movies they watch a high average rating. Other users are quite critical and give the movies they watch a low average rating. Also, some genres are better liked than others overall. To incorporate these insights into my model, I repeat the process of calculating and then sweeping out residual averages for each user and for each combination of genres. Predicting ratings in the test set with my naive average mu, my residual movie average b_i, and my residual user average b_u (mu + b_i + b_u) produces an improved RMSE of 0.8644466. Adding genre effects (b_g) to the model brings RMSE down a little further, to 0.864078. In theory, I am already beating my target RMSE of 0.86490, but I'm not beating it with any margin of safety, and there's no guarantee that the model will perform as well as on the validation data as it does on the test data, so I'm going to continue refining the model.

What about time effects? Seasonal patterns or world events might affect users' mood, causing them to rate movies higher or lower on some days than on other days. (For instance, one study found that stock prices perform worse in gloomy, rainy weather!) To incorporate these effects into my model using an average, however, proves complicated, because my rating timestamps are a continuous rather than categorical variable. I must convert them to a categorical variable by "stratifying" or "rounding" the timestamps into "bins". But how do I choose the optimal "bin" size? Should I round to the nearest week? Month? Year? To answer this question, I use 5-fold cross-validation to "tune" the bin size. (In effect, this means that I partition my training data five times, each time with 4/5 of my data used for training and and 1/5 used for testing, and then I test the three different bin sizes on each of the five partitions and select the bin size that minimizes RMSE on average over the five tests. This operation is somewhat computationally intensive and takes a few minutes to run.)

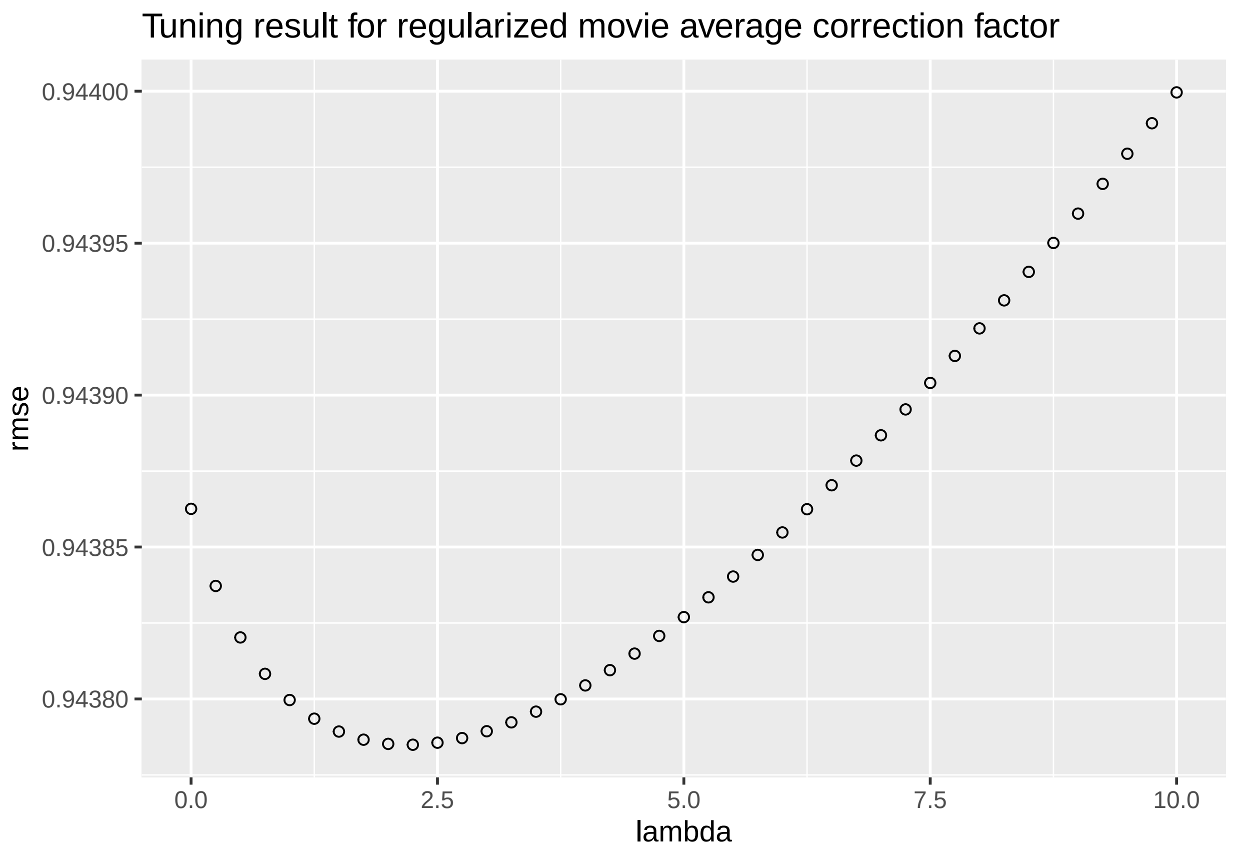
Plotting my tuning results from the 5-fold cross-validation, I see that RMSE is minimized when I stratify by week. Therefore, I calculate and sweep out weekly residual averages from my training data and then predict ratings in my test set using the formula mu + b_i + b_u + b_g + b_d, with b_d representing my residual weekly averages. This brings RMSE for the cumulative model down a little further, to 0.8639782. In theory, this should be sufficient. But I'd like to further refine the model and to increase my margin of safety before testing on the validation set. Table 1 summarizes the results of the cumulative model.

Table 1: Table 1

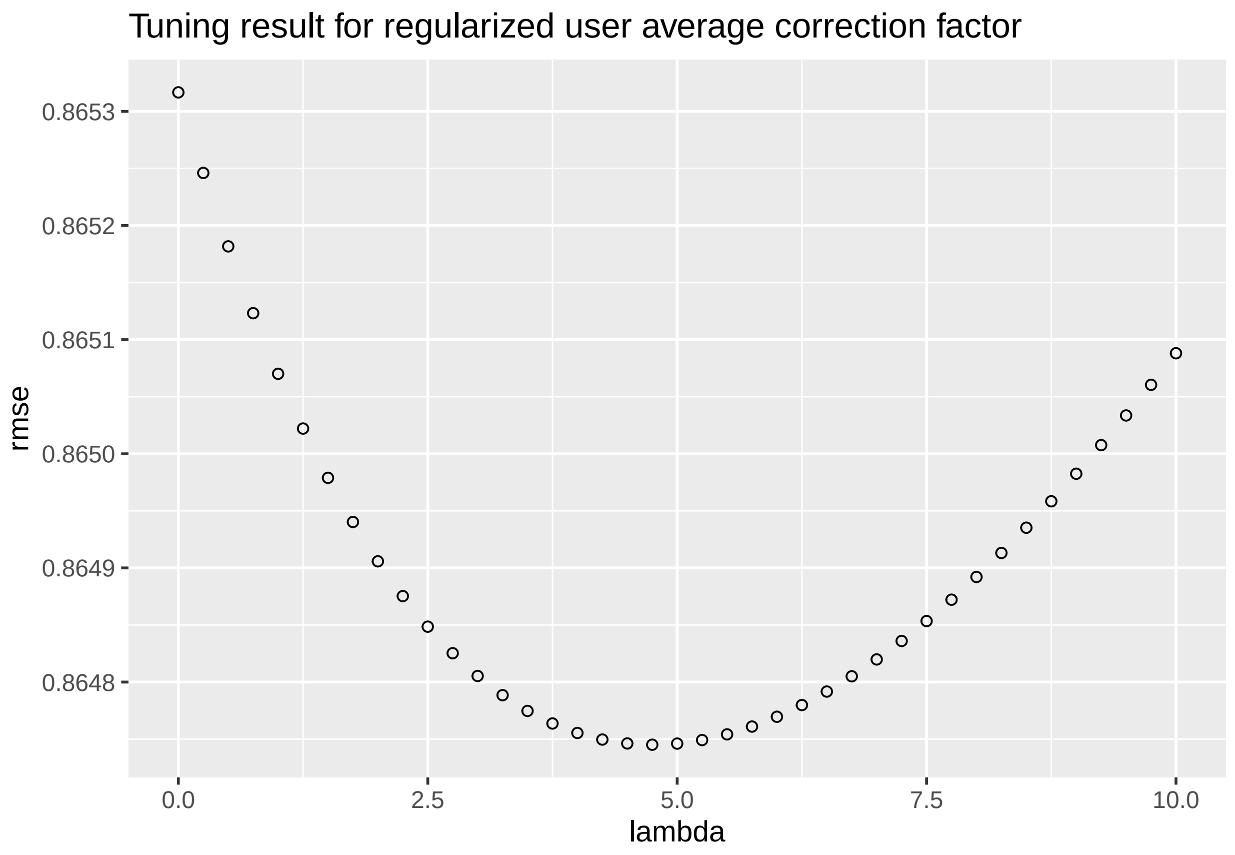
model	RMSE	tuning_param
Naive average	1.0600537	NA
Naive + movie average	0.9429615	NA
Naive + movie + user average	0.8644818	NA
Naive + movie + user + genre average	0.8641138	NA
Naive + movie + user + genre + date average	0.8640137	week

Regularized Model

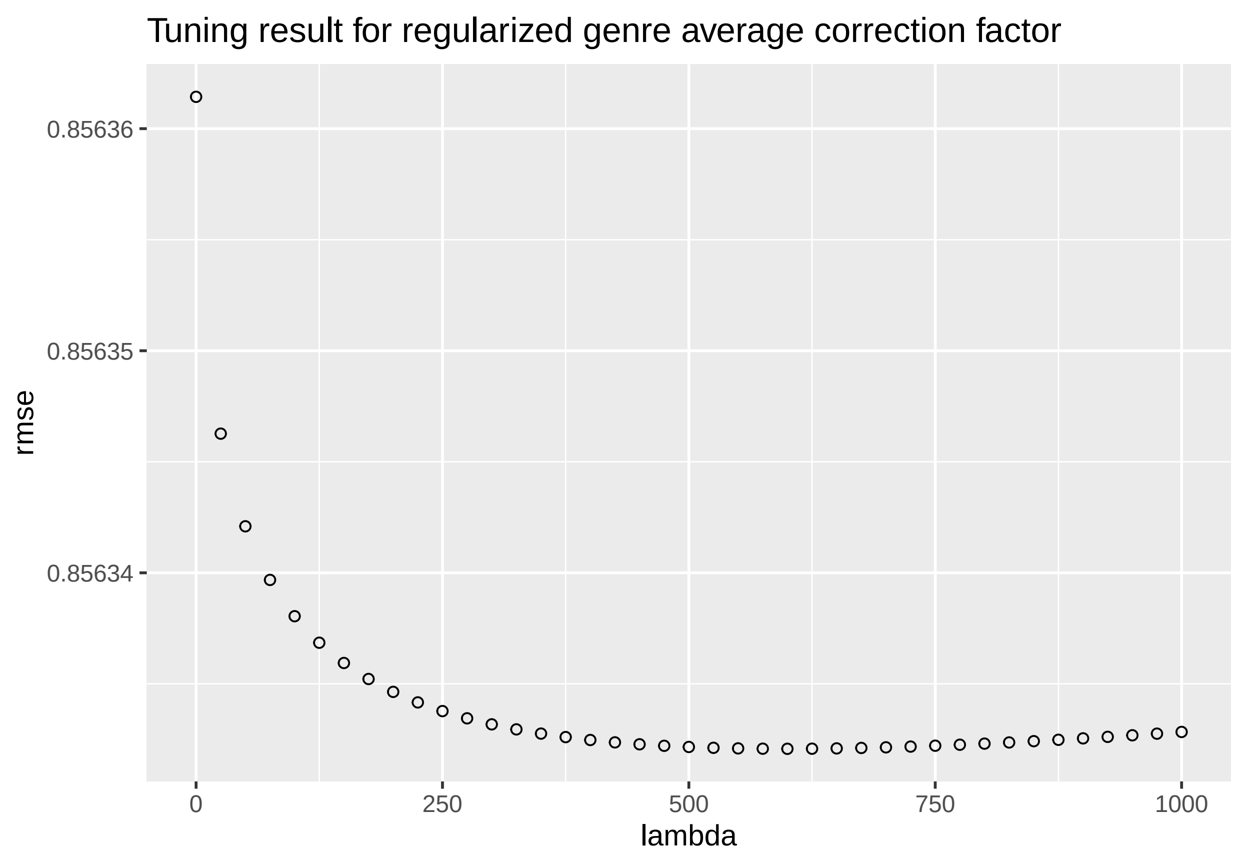
To refine my base model, I will use a statistical technique called "regularization". Regularization punishes large estimates based on a small sample size by shrinking (dividing) the estimates by some factor lambda. To calculate a regularized average, I sum the ratings and then divide the sum by the number of ratings plus some correction factor lambda. To select a value for lambda, I "tune" lambda with 5-fold cross-validation.



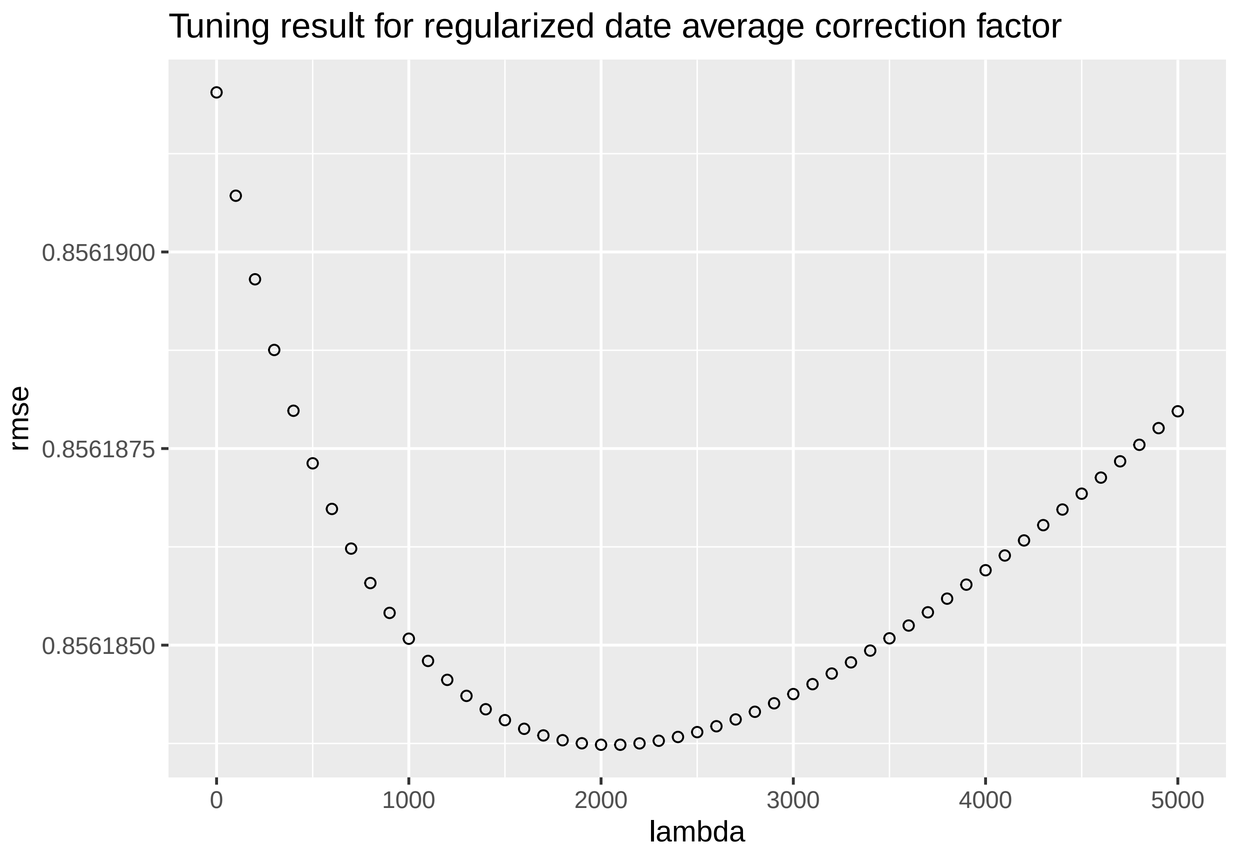
I begin by regularizing my movie averages. To ensure that I've selected a good range of values to try out for lambda, I plot my tested lambda values against the average RMSE from cross-validation. I see that there is a local minimum at lambda = 2.25 and that I'm unlikely to get a different minimum by widening the range of tested lambda values. I choose the value of lambda that minimizes RMSE in the plot, and I use this value of lambda to predict ratings on my test set. The resulting RMSE is 0.9429389, slightly better than the 0.9429615 RMSE result obtained earlier when predicting with a non-regularized movie average in my base model.



Repeating the regularization and cross-validation process for my user averages, I plot my tested lambda values and find a local minimum at 4.75. Using this value of lambda to predict ratings on my test set, I get an RMSE of 0.8640154, much better than the 0.8644466 RMSE result previously obtained when predicting ratings with a non-regularized user average in my base model. In fact, with just three regularized averages, I've achieved nearly the same RMSE as the full base model achieved with five.



Extending the regularization method to my genre averages, I find that for this variable, it's necessary to try a much larger range of values for lambda. (That likely means that at this stage of the model I'm seeing less signal and more noise, so I need to be a little more aggressive about tuning out the noise with a higher value of lambda.) I find a local minimum at 600. Using this value of lambda to predict ratings on my test set, I get an RMSE of 0.8637026, substantially better than the 0.864078 RMSE result previously obtained when predicting ratings with a non-regularized genre average in my base model. I'm now officially outperforming my full base model.



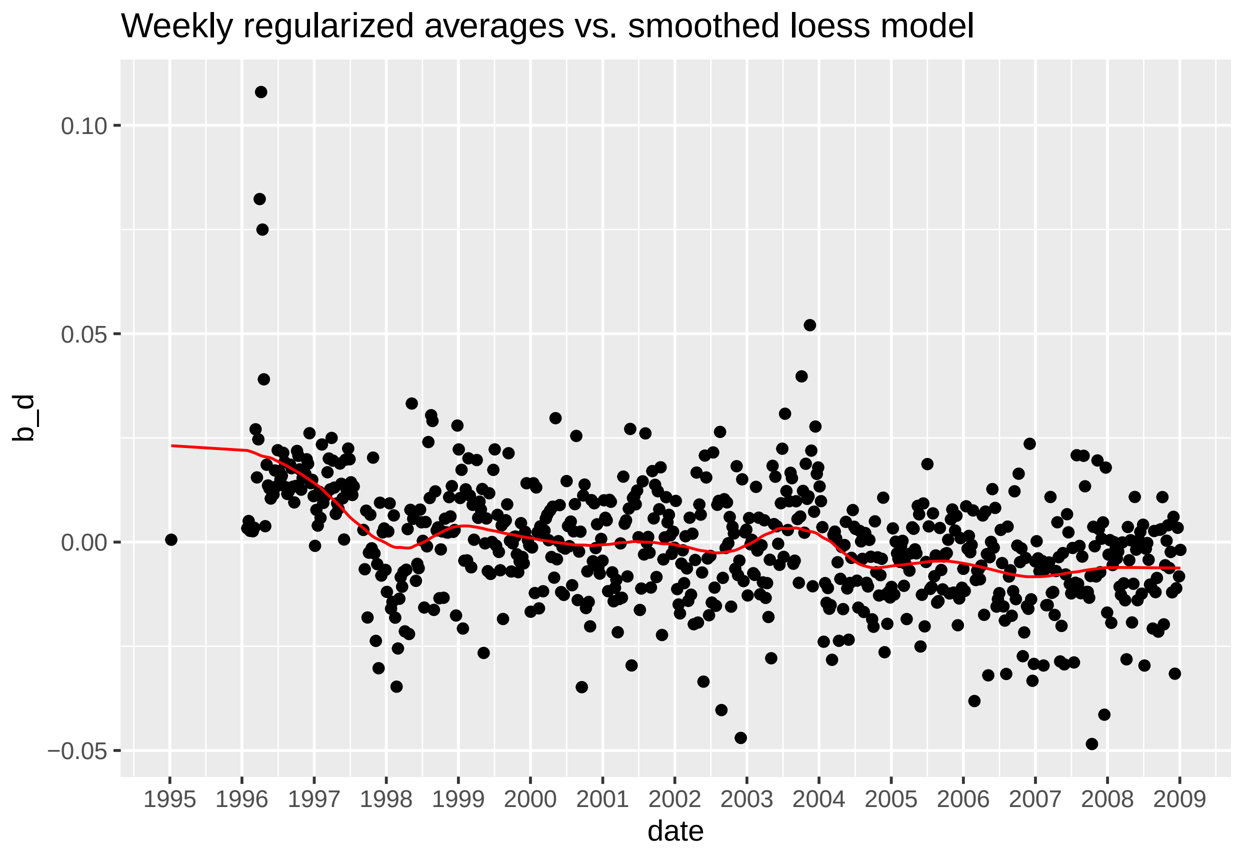
Finally, I also regularize my averages by date (again using "week" as my rounding unit as in the base model). Tuning lambda, I find a local minimum at 2000. Using this value of lambda to predict ratings on the test set, I get an RMSE of 0.8635759. This beats both my base model and my target RMSE of 0.86490 by a comfortable margin. Table 2 summarizes the results from my cumulative base model and my cumulative regularized model.

Table 2: Table 2

model	RMSE	tuning_param
Naive average	1.0600537	NA
Naive + movie average	0.9429615	NA
Naive + movie + user average	0.8644466	NA
Naive + movie + user + genre average	0.8640780	NA
Naive + movie + user + genre + date average	0.8639782	week
Reg. naive + movie average	0.9429389	2.25
Reg. naive + movie + user average	0.8640154	4.75
Reg. naive + movie + user + genre average	0.8637026	600
Reg. naive + movie + user + genre + date average	0.8635759	2000

Before calling this model complete, however, I'd like to attempt one last refinement. Can I use loess smoothing on my "date" average to improve my predictions?

To refine my regularized model, I will try smoothing my regularized weekly average ratings with a loess function. To that end, I train a loess model and tune its smoothing span using a 10-fold cross-validation method built into R's caret package. I plot the smoothed model against my regularized weekly averages to visualize the difference between the two prediction methods. The loess model is obviously much less noisy, but it may be *too* smooth. In particular, it appears to underestimate the magnitude of a dip in the data around 1998 and a spike around 2004.



To check this intuition, I use the loess model to predict ratings on the test set in lieu of weekly regularized averages. Sure enough, the model underperforms my weekly regularized averages. The full regularized averages model predicted ratings with an RMSE of 0.8635759, whereas the modified model with loess smoothing predicts with a slightly inferior RMSE of 0.8636699. The results with loess smoothing are summarized in Table 3.

Table 3: Table 3

model	RMSE	tuning_param
Naive average	1.0600537	NA
Naive + movie average	0.9429615	NA
Naive + movie + user average	0.8644466	NA
Naive + movie + user + genre average	0.8640780	NA
Naive + movie + user + genre + date average	0.8639782	week
Reg. naive + movie average	0.9429389	2.25
Reg. naive + movie + user average	0.8640154	4.75
Reg. naive + movie + user + genre average	0.8637026	600
Reg. naive + movie + user + genre + date average	0.8635759	2000
Reg. naive + movie + user + genre average + date loess model	0.8636699	0.155555555555556

Thus, for this model I reject loess smoothing and use regularized averages instead. Now I'm ready to proceed to validation with my regularized averages model as my final model.

Results

With final model in hand, I proceed to validation. Using my final, regularized averages model to predict ratings on the validation data set, I get an RMSE of 0.8646289. This is a little higher than expected based on my testing, but it's below my target threshold of 0.86490. My regularized model is officially a success.

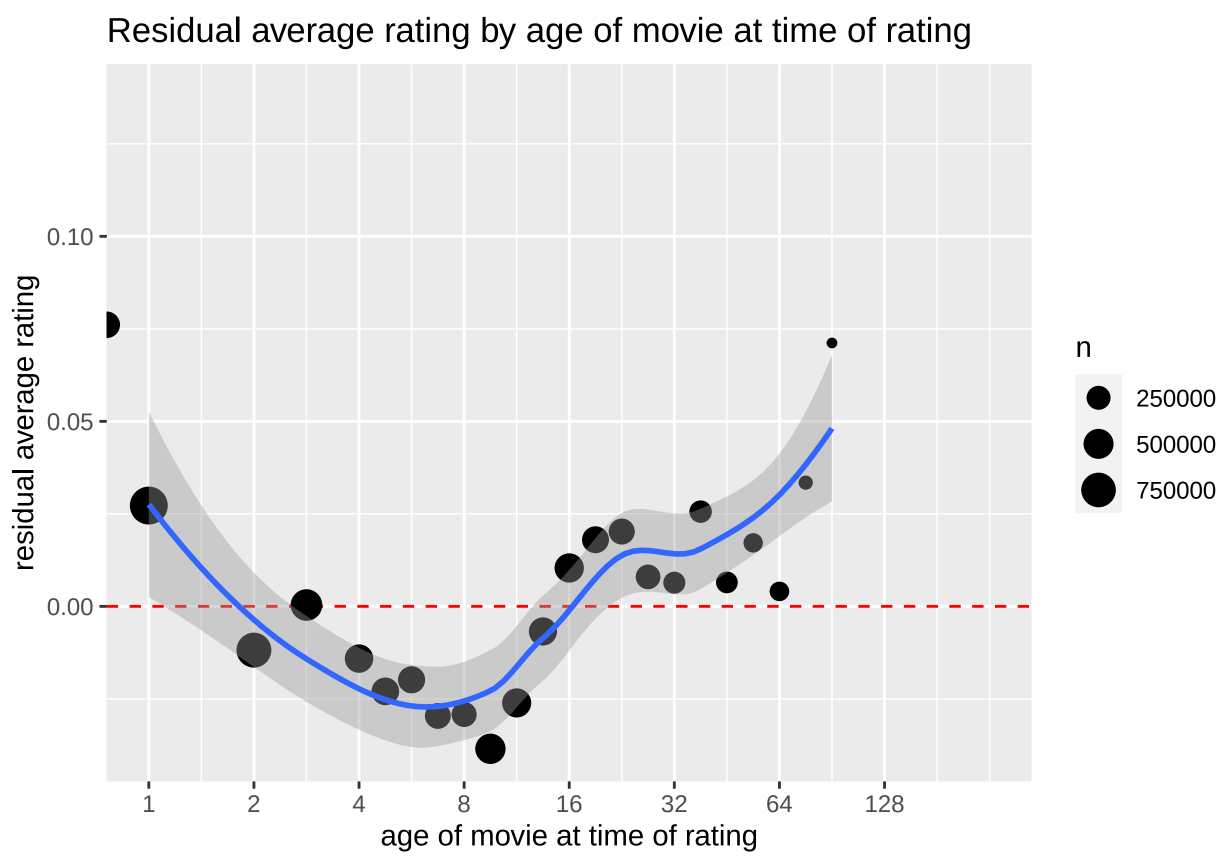
Surprisingly, achieving an RMSE below 0.86490 proves to be possible with just five regularized averages. It turns out to require no matrix factorization and no attention to the preferences of specific users beyond each user's average rating. I believe that what made this possible is the combination of effective variable tuning (using a process of five-fold cross-validation that I manually coded myself) and forbidding predictions above or below the lowest and highest possible ratings. Indeed, had I failed to notice that the minimum possible rating changed from 1.0 to 0.5 in 2003, I probably could not have hit my target RMSE with just five regularized averages.

Conclusion

This report summarizes a model that predicts movie ratings with surprising accuracy with just a few summary statistics. Just five regularized averages are used in the final model: overall average rating, average rating for each movie, average rating for each user, average rating for each combination of genres, and average rating by week. Each of these averages has been carefully tuned using five-fold cross-validation.

The strength of this model is its simplicity. Its strength is also its main limitation: that it is not especially tailored to individual users. The model predicts movie ratings entirely from population-level information. Future work should look at individual user preferences and pairwise correlations between movies using matrix factorization.

In fact, there's more that could be done even at the population level. A little data exploration leads me to believe the model could be further improved by taking into account the age of a movie at the time of rating. When I extract each movie's release year from the "title" variable and subtract it from the year of rating, I get a measure of how "fresh" or "classic" a movie is. Quickly plotting this "age" variable against the remaining residuals after subtracting my five regularized averages reveals a clear user bias in favor of movies between 0 and 2 years old, and against movies 2 to 13 years old. Users also seem to enjoy "classics" of more than 13 years of age.



However, development of a model that uses this information will have to wait for a future study.