A Simple Recommendation System

Leo PeBenito

4/12/2020

Introduction

Netflix is a streaming service for films and similar content. A defining feature of the Netflix platform is a user's ability to rate movies she or he has viewed. These ratings provide a valuable source of raw data that can be used to train a machine learning algorithm to make recommendations to Netflix users. Such algorithms are called recommendation systems.

Recommendation systems are a formidable machine learning challenge. There are multiple predictors to consider that may be continuous or categorical. In addition, not every user has rated every film so outcomes need not depend on the same set of predictors. Rather than attempt to make movie recommendations to users, the following demonstrates the ability to predict the rating a user has given to a film she or he has already rated. This is because in practice there are no users to make actual recommendations to, making it impossible to validate the recommendation system. The Lens Lab data set used in this exercise consists of 10 million movie ratings. Films are rated on a scale from 0.5 to 5 stars in increments of 0.5. Zero star ratings are equivalent to no rating (NULL).

The residual mean squared error (RMSE, eq.(1)) is defined as the loss function for model evaluation

$$\texttt{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2} \tag{1}$$

where $\hat{Y}$ is the predicted rating, Y is the actual rating, and N is the total number of ratings. The average rating serves as an initial model, which can then be refined based on the statistics of additional features of the data. The difference between the bulk average and the average rating associated with a specific component of the data is often called an effect or a bias. Here, biases are computed for four features of the data. The model is extended in a recursive fashion such that subsequent biases are computed relative to the preceding model. First biases are computed at the level of the individual such as the specific film and user. Next, biases are computed based on collections of the data. Biases are computed for continuous and categorical predictors, change over time and genre, respectively. Regularization is applied as the last step of model refinement. The resulting model has an RMSE below 0.8490.

Methods

The complete data set is partitioned into `edx` and `validation` sets, to start. The `edx` set is used exclusively for the first stage of model development. The `edx` set is further split up into `training` and `testing` sets. The model is developed using the `edx` subsets, `training` and `testing`, momentarily feigning ignorance of the `validation` set. Only after demonstrating consistency in the model is it then re-trained on the intact `edx` set. In this fashion the penultimate step in model development mimics the final stage of training the model on the `edx` set before lastly applying it to the `validation` set. **Exploratory data analysis and accompanying insights are included in the results section**, following a detailed description of the procedure, to facilitate discussion and interpretation.

R 3.6 is used for data analysis. The tidyverse is used for wrangling. The lubridate package is used to extract date information from time-stamp data. The caret package is used to generate partitions. The MovieLens 10M data set is obtained from the Lens Group web page. The code for loading the requisite packages, and downloading the data set and putting it into tidy format is prepended to the code for obtaining the final model provided in the accompanying script.

Procedure

Define the residual mean squared error as the loss function.

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

Partition the `edx` set into `training` and `testing` sets.

```
# Partition edx into testing and training sets based on ratings (outcomes).
test_indices <- createDataPartition(y=edx$rating, times=1, p=0.1, list=FALSE)
training <- edx[-test_indices,]
temp  <- edx[test_indices,]

# Omit movies and users in the testing set that do not appear in the training set
testing  <- temp %>%
  semi_join(training, by="movieId") %>%
  semi_join(training, by="userId")

# Specify the training and testing sets for the generic algorithm inputs
train_set <- training
test_set  <- testing
```

Begin with the simplest model, which amounts to simply guessing the average rating for all ratings.

```
# Compute average
mu <- mean(train_set$rating)

# Predict the average for every rating
prediction1 <- rep(mu, nrow(test_set))

# Update results table
rmse_results  <- tibble(method="Simplest Model", RMSE=RMSE(prediction1, test_set$rating))
```

Refine the model one step at a time based on explicit features of the data set. Account for bias toward each film relative to the overall average.

```
# Compute movie effect
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu))

# Update model
prediction2 <- mu + test_set %>%
  left_join(movie_effect, by="movieId") %>%
```

```r
  .$b_movie

# Update results table
rmse_results  <- bind_rows(rmse_results,
                          tibble(method="Movie Effect",
                                 RMSE=RMSE(prediction2, test_set$rating)))
```

Append the user bias associated with each individual user to the previous model in a linear fashion.

```r
# Compute user effect
user_effect <- train_set %>%
  left_join(movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_user = mean(rating - mu - b_movie))

# Update model
prediction3 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect,  by="userId")  %>%
  mutate(pred = mu + b_movie + b_user)  %>%
  .$pred

# Update results table
rmse_results  <- bind_rows(rmse_results,
                           tibble(method="Movie + User Effects",
                                  RMSE=RMSE(prediction3, test_set$rating)))
```

Treat the time-stamp as a continuous independent variable. Use the lubridate package to extract date information.

```r
# Compute date from timestamp rounding by week
train_set <- training %>%
  mutate(date_week = round_date(as_datetime(timestamp), unit="week"))

test_set  <- testing  %>%
  mutate(date_week = round_date(as_datetime(timestamp), unit="week"))
```

R provides myriad ways to find the best-fit curve. However, given the relatively large size of the data set, group ratings on a weekly basis. Compute the temporal bias associated with the moving average rating after each week.

```r
# Compute date effect
date_effect <- train_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect,  by="userId")  %>%
  group_by(date_week) %>%
  summarize(b_date = mean(rating - mu - b_movie - b_user))

# Update model
prediction4 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect,  by="userId")  %>%
  left_join(date_effect,  by="date_week") %>%
  mutate(pred = mu + b_movie + b_user + b_date) %>%
  .$pred

# Update results table
rmse_results  <- bind_rows(rmse_results,
                           tibble(method="Movie + User + Date Effects",
                                  RMSE=RMSE(prediction4, test_set$rating)))
```

A film typically falls into more than one genre. For each entry, the pipe character is used to separate each genre within the genres field, ie:

title	genres
Boomerang (1992)	Comedy|Romance

For simplicity, do not decompose values in the genres column into constituent components. Compute the bias based on genres treating it as an aggregate predictor.

```r
# Compute genres effect
genres_effect <- train_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect,  by="userId")  %>%
  left_join(date_effect,  by="date_week") %>%
  group_by(genres) %>%
  summarize(b_genres = mean(rating - mu - b_movie - b_user - b_date))

# Update model
prediction5 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect,  by="userId")  %>%
  left_join(date_effect,  by="date_week") %>%
  left_join(genres_effect, by="genres") %>%
  mutate(pred = mu + b_movie + b_user + b_date + b_genres) %>%
  .$pred

# Update results table
rmse_results  <- bind_rows(rmse_results,
                           tibble(method="Movie + User + Date + Genres Effects",
                                  RMSE=RMSE(prediction5, test_set$rating)))
```

Provided there is a progressive decrease in RMSE at each model refinement step it is appropriate to perform regularization. Consider the simple case during the early stage of model refinement when the first bias is computed. Minimize the following equation (eq.(2)) rather than the RMSE

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_{i}b_i^2. \tag{2}$$

The first summation is the least squares term. The second term is a penalty term. Estimate b_i as (eq.(3))

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu}). \tag{3}$$

To also include the user effect minimize the following (eq.(4))

$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda\Big(\sum_i b_i^2 + \sum_u b_u^2\Big). \tag{4}$$

Each subsequent bias (b) is computed in a similar fashion. Cross-validation is used to find λ, which is a tuning parameter. Cross-validation is first performed using only the `training` and `testing` sets.

```r
# Regularization of movie rating by movieId, userId, date rounded by week, genres
lambdas <- seq(4.5, 6, 0.1)

# Cross-validation
training_rmses <- sapply(lambdas, function(lambda){

  b_movie <- train_set %>%
    group_by(movieId)  %>%
    summarize(b_movie = sum(rating - mu)/(n()+lambda))

  b_user <- train_set %>%
    left_join(b_movie, by="movieId") %>%
    group_by(userId)  %>%
    summarize(b_user = sum(rating - mu - b_movie)/(n()+lambda))

  b_date <- train_set %>%
    left_join(b_movie, by="movieId") %>%
    left_join(b_user,  by="userId")  %>%
    group_by(date_week) %>%
    summarize(b_date = sum(rating - mu - b_movie - b_user)/(n()+lambda))

  b_genres <- train_set %>%
    left_join(b_movie, by="movieId") %>%
    left_join(b_user,  by="userId")  %>%
    left_join(b_date,  by="date_week") %>%
    group_by(genres) %>%
    summarize(b_genres = sum(rating - mu - b_movie - b_user - b_date)/(n()+lambda))

  predicted_ratings <- test_set %>%
    left_join(b_movie, by="movieId") %>%
    left_join(b_user,  by="userId")  %>%
    left_join(b_date,  by="date_week") %>%
    left_join(b_genres, by="genres") %>%
    mutate(pred = mu + b_movie + b_user + b_date + b_genres) %>%
    .$pred

  return(RMSE(predicted_ratings, test_set$rating))
})

# Regularization lambda from cross-validation on training and testing sets
training_lambda <- lambdas[which.min(training_rmses)]

# Update results table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="All Effects + Regularization",
                                 RMSE=min(training_rmses)))
```

Training the resulting model on the `edx` set and applying it on the `validation` set using the parameters derived by cross-validation on the `edx` partitions, `training` and `testing`, is a key step in model validation. If the two RMSEs are comparable this suggests consistency among model outcomes. The last step is to re-train the model on the `edx` set, rounding unphysical predictions at the extrema to the nearest possible rating, and apply it to the `validation` set.

Discretize the model as an additional check. Create bins of width 0.5 stars symmetrically bordering internal positions on the rating scale. Shift predicted ratings within each bin to the central rating within each bin. All remaining predictions are near the scale boundaries, shift these to the nearest possible rating of 0.5 or 5 stars.

```
# Discretize model
disc_predicted_ratings <- data.frame(predicted_ratings) %>%
  mutate(rating = ifelse(predicted_ratings >= 4.75, 5,
                  ifelse(between(predicted_ratings, 4.25, 4.75), 4.5,
                  ifelse(between(predicted_ratings, 3.75, 4.25), 4,
                  ifelse(between(predicted_ratings, 3.25, 3.75), 3.5,
                  ifelse(between(predicted_ratings, 2.75, 3.25), 3,
                  ifelse(between(predicted_ratings, 2.25, 2.75), 2.5,
                  ifelse(between(predicted_ratings, 1.75, 2.25), 2,
                  ifelse(between(predicted_ratings, 1.25, 1.75), 1.5,
                  ifelse(between(predicted_ratings, 0.75, 1.25), 1, 0.5))))))))))
```

Results

Movie ratings tend to be high (fig. 1). The mean is greater than the midpoint of the rating scale, and the median equal to 4 is above the mean. Whole star ratings are more common than half star ratings, and ratings are concentrated around the mean of 3.5124652.

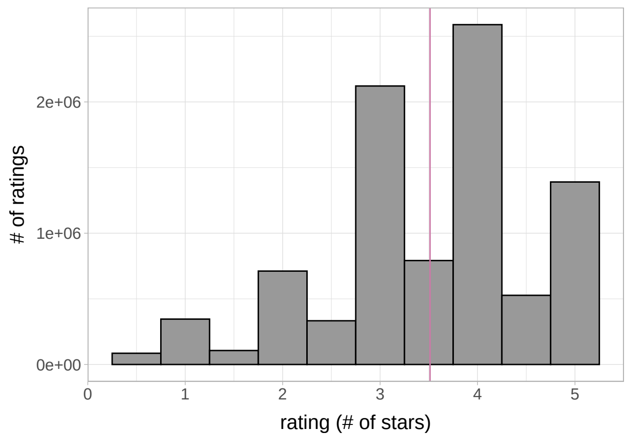

Figure 1: Histogram (10 bins) for movie ratings (number of stars) from the edx set. The vertical red line corresponds to the average rating, which serves as the starting point for model development.

Accounting for the movie effect generates a distribution of ratings (fig. 2).

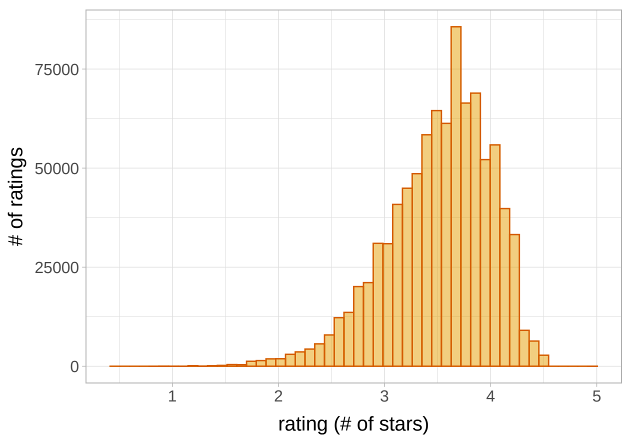

Figure 2: Histogram of predicted ratings based on the model including the movie effect (50 bins).

Accommodating the user effect smooths the distribution (fig. 3).

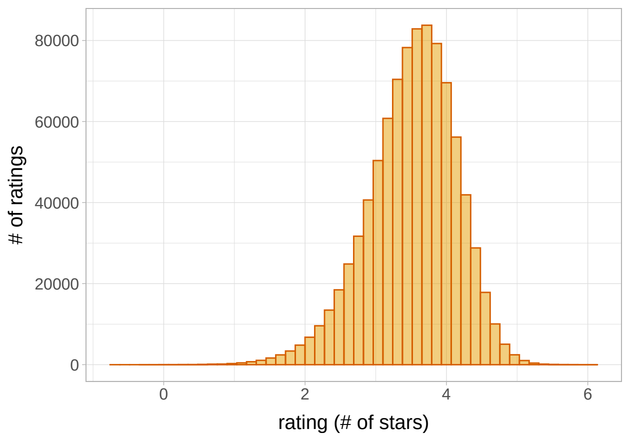

Figure 3: Histogram of predicted ratings based on the model including the movie and user effects (50 bins).

There are unphysical predictions, those less than zero or above five stars, at the model stage where movie and user effects are included. The minimum and maximum predicted ratings are -0.6913045 and 6.0640963 stars, respectively. To adjust for this, predictions less than 0.5 and above 5, are shifted to 0.5 and 5, respectively. This procedure tends to decrease the RMSE but is postponed until the last step.

The average rating on a given week (fig. 4) fluctuates around the mean. Starting in the year 2000 there is some overall trend that changes gradually over the course of years. In the early stages of data collection the fluctuations tend to be a bit larger, however these data points typically rely on smaller samples.

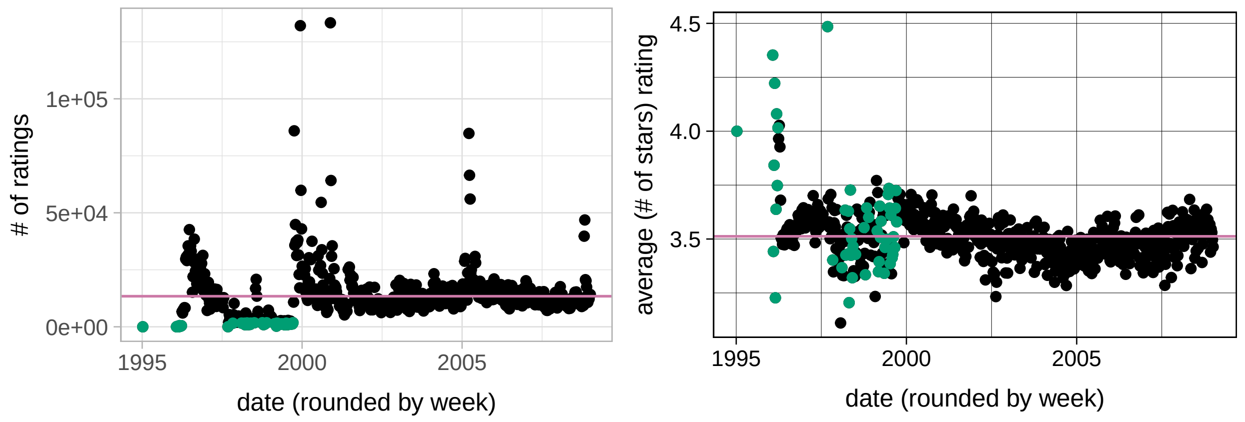

Figure 4: (Left) Scatter plot of the number of ratings made on a weekly basis v. the date rounded to the nearest week. The average number of ratings made on a given week is shown by a horizontal red line. Points corresponding to a number of ratings that is less than or equal to one standard deviation below the mean are colored green. (Right) Scatter plot of the average rating computed on a weekly basis. The overall average is shown by the red horizontal line. The overall weekly average rating (number of stars) is shown by a red horizontal line. Again, points corresponding to a number of ratings that is less than or equal to one standard deviation below the mean number of weekly ratings are colored green.

The bias related to changes in the average rating over time contributes only slightly to the model's distribution of predictions (fig. 5).

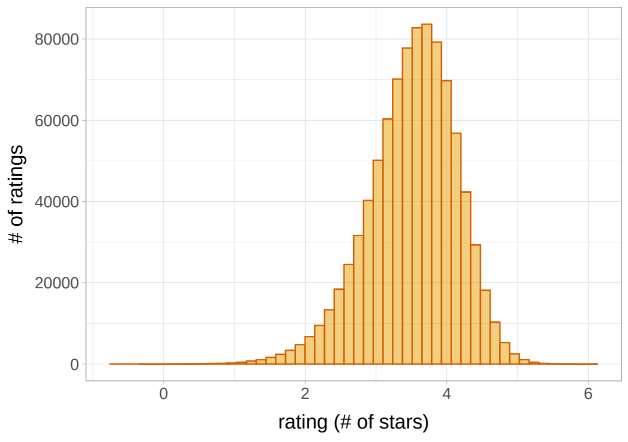

Figure 5: Histogram of predicted ratings based on the model including the movie, user, and date effects (50 bins).

There are 797 distinct genre combinations (categories), in the genres field. The average rating for genres ranges from 1.4491115 to 4.7142857 stars. The top rated genres category is Animation|IMAX|Sci-Fi. The lowest rated genres category is Documentary|Horror. Genres at the tails of the average rating spectrum tend to have relatively few data points. In contrast, genres with more ratings tend to have average ratings closer to the mean, as shown below.

genres	Ave_Rating	N_Ratings
Drama	3.712364	733296
Comedy	3.237858	700889
Comedy|Romance	3.414486	365468
Comedy|Drama	3.598961	323637
Comedy|Drama|Romance	3.645824	261425
Drama|Romance	3.605471	259355

The genres predictor has a subtle but nonetheless beneficial impact on the RMSE. The resulting distribution of rating predictions (fig. 6) is almost indistinguishable from the previous model.

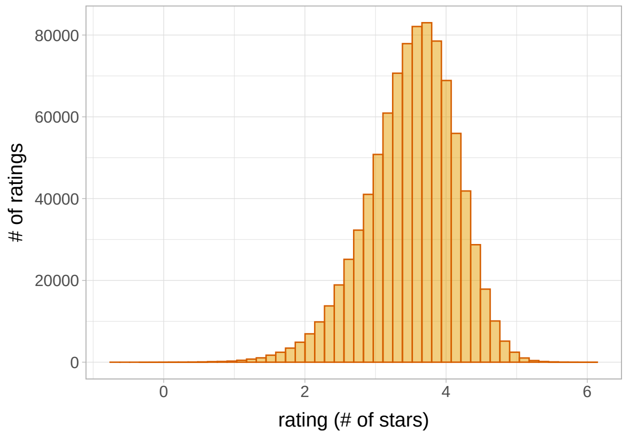

Figure 6: Histogram of predicted ratings based on the model including the movie, user, and date effects (50 bins).

Regularization is performed at the model stage consisting of movie, user, date, and genres effects. Cross-validation on the `training` and `testing` sets produces a λ value of 5.2. When the algorithm derived using the `training` and `testing` sets is applied to the `edx` and `validation` sets the RMSE is further improved. The optimal value of λ is recalculated using cross-validation on the `edx` and `validation` sets .

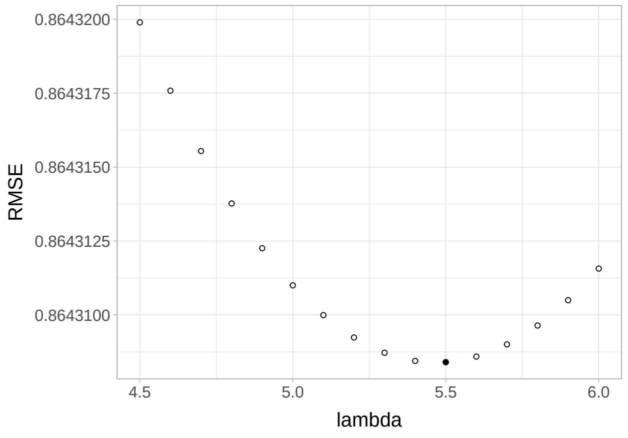

Figure 7: Scatter plot of rmse v. lambda with the minimum RMSE value indicated by a filled point illustrates the cross-validation process.

Now, the λ value determined by cross-validation equals 5.5. Unphysical ratings beyond the minimum or maximum rating are shifted to the nearest possible rating when computing RMSE (fig. 7). The distribution of the final model is show in figure 8.

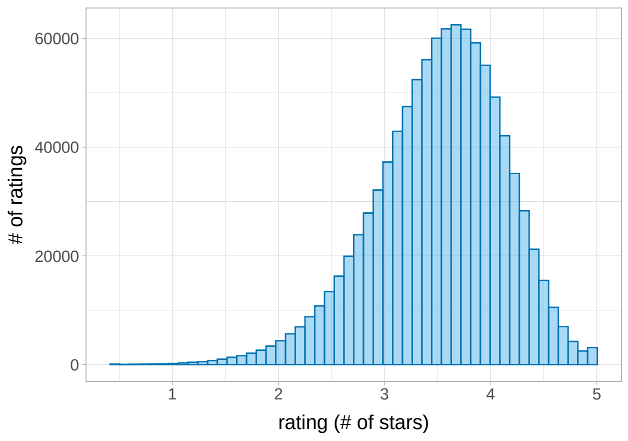

Figure 8: Histogram of predicted ratings based on the final model (50 bins).

RMSE values at each stage of model development are shown in table 1. There is a continuous decrease in the RMSE at each stage of model development. The largest improvements to the model are made in the early stages of development, when movie and user effects are introduced. Effects coming from the date and genres have less of an impact. Regularization also has only a modest influence on the RMSE. The RMSE for the final model trained on the `edx` set and applied to the `validation` set is less than 0.86490.

Table 1: RMSE Results at Each Stage of Model Refinement

method	RMSE
Simplest Model	1.0612047
Movie Effect	0.9447238
Movie + User Effects	0.8668933
Movie + User + Date Effects	0.8668261
Movie + User + Date + Genres Effects	0.8664401
All Effects + Regularization	0.8658292
Apply Model using Training-lambda to edx and validation sets	0.8643092
Final Model	0.8641922

Discretizing the final model has a detrimental impact on the RMSE. The RMSE goes up to 0.8763109. The discretized model (red) is show behind the final model (blue) and in front of the actual ratings in the `validation` set (grey) in figure 9.

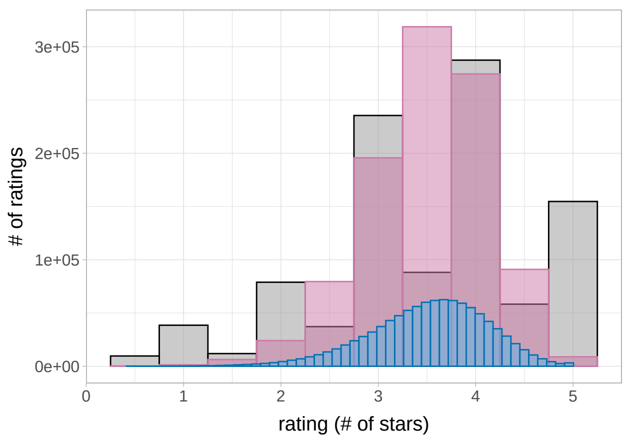

Figure 9: Histogram for the number of stars. The final model is shown in blue, the result of discretizing the final model is shown in red, and the actual ratings from the validation set are shown in grey with black outlines. The number of bins is 10 for the actual ratings and the discretized model, and 50 for the final model predictions.

Conclusions

In the case of recommendation systems accurate predictions generally evade simple machine learning approaches like linear and logistic regression. K-nearest neighbors (Knn) and decision trees are attractive alternatives for their flexibility. However, computational intensity limits application of such approaches on standard laptop computers for even a modestly large data set. Principle component analysis (PCA) is a natural addendum to the model described above as it can be included linearly. While less so than Knn and decision trees, PCA is sufficiently computationally demanding that it is left for future work.

The methodology shown here consists of computing the average and computing biases (also known as effects) associated with explicit features of the data. The more specific these predictors are the better. As seen for predictors based on the specific film and on a given user, accounting for bias at the level of the individual results in the greatest decrease in RMSE. Predictors based on collections of the data such as the moving average over time, or categories such as genre, may not deviate much from the average but the associated bias may nonetheless provide improvements to the RMSE. It is straight forward to compute the temporal and genre effects on an individual basis for movies an users. Instead regularization is used to make the most of available predictors by penalizing large estimates that come from a relatively small number of observations, thereby sacrificing granularity for speed.

Two common factors contribute to the difficulty of creating the recommendation system. The first is the inherent uncertainty of movie ratings, these outcomes are not deterministic but random. That is to say that obtaining more data with higher granularity will not necessarily improve predictions. The second is that the rating system is discrete. The model generates a fairly smooth distribution that concentrates predictions around the mean rating (number of stars) of ~3.5, but the most common ratings are four and three. Due to tapering at the tail of the distribution, five star predictions almost never occur although it is the third most common rating, accounting for ~15% of the data. In conclusion, a next step in exploratory data analysis might be directed at discerning whole start from half star ratings.