A Prediction Model for the MovieLens Data Set

Xabriel J Collazo Mojica

1/13/2022

Introduction

Collaborative filtering, a type of recommendation system, allows users to leverage work made by other users. One common application is the recommendation of movies. Users rate movies they have seen, typically using a 5-star scale, and the system then predicts how the users would rate other movies based on ratings previously made.

The GroupLens Research Group at the University of Minnesota publishes *MovieLens*, a data set of movie ratings, enabling other interested parties to analyze and learn from the data. In this work, we utilize a version of their data set that includes 10 million ratings selected randomly from their complete data set [MovieLens].

The data set is composed of two files. The first file, `movies.dat`, contains information about the movies themselves. Its format is as follows:

```
MovieID::Title::Genres

Examples:
1::Toy Story (1995)::Adventure|Animation|Children|Comedy|Fantasy
2::Jumanji (1995)::Adventure|Children|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
```

Notice that this file is delimited by a double colon (`::`), and it includes a movie identifier, the movie name, and a set of genres, themselves separated by pipes (`|`). The second file, `ratings.dat`, contains information about user ratings. Its format is as follows:

```
UserID::MovieID::Rating::Timestamp

Examples:
1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
```

This file is delimited in the same format, and includes a user identifier, a movie identifier that we can match with the previous file, an integer rating from 1 to 5, and timestamps that represent the time of the rating in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

We use common data import and wrangling techniques to convert these two files into a single object that is more amenable to our purposes. We also divide the data randomly into three sets: 10% is a validation set, which we only utilize for final evaluation of the solution. 72% is a train set, utilized to build our model, and the remaining 18% of the data is used as a test set to continually asses our progress.

We model the problem as a sum of multiple effects given by different features of the data set. Later in this report we explain in detail our model, but for now we present its final form:

$$Y_{u,i} = \mu + b_i(t) + b_u + \varepsilon_{u,i}$$

where:

$Y_{u,i}$: the predicted ratings for all movie and user pairs.

μ: the average rating of all movies in the train set.

$b_i(t)$: term that accounts for the "effect" (or "bias") allocated to the rated item. We notice that our data also has a temporal component, thus we make this term a function of time.

b_u: term that accounts for the effect allocated to the user.

$\varepsilon_{u,i}$ the error for each movie and user pair when comparing to the actual ratings.

We try to learn each of the effects above by applying the least squares method [LeastSquares], that is, as we add more terms to the model, we try to minimize $\varepsilon_{u,i}$. As a final modeling step, we also apply regularization, to control the effect of movies that skew the results too much with too few ratings.

The goal of the project is to find a model and parameters that minimize the root mean square error $RMSE = sqrt(mean(Y_{u.i} - \hat{Y}_{u,i})^2)$. We intend to build a model that surpasses `RMSE < 0.86490`. We find that we reach this threshold by implementing the model above, without the need of more elaborate techniques.

Methodology and Implementation

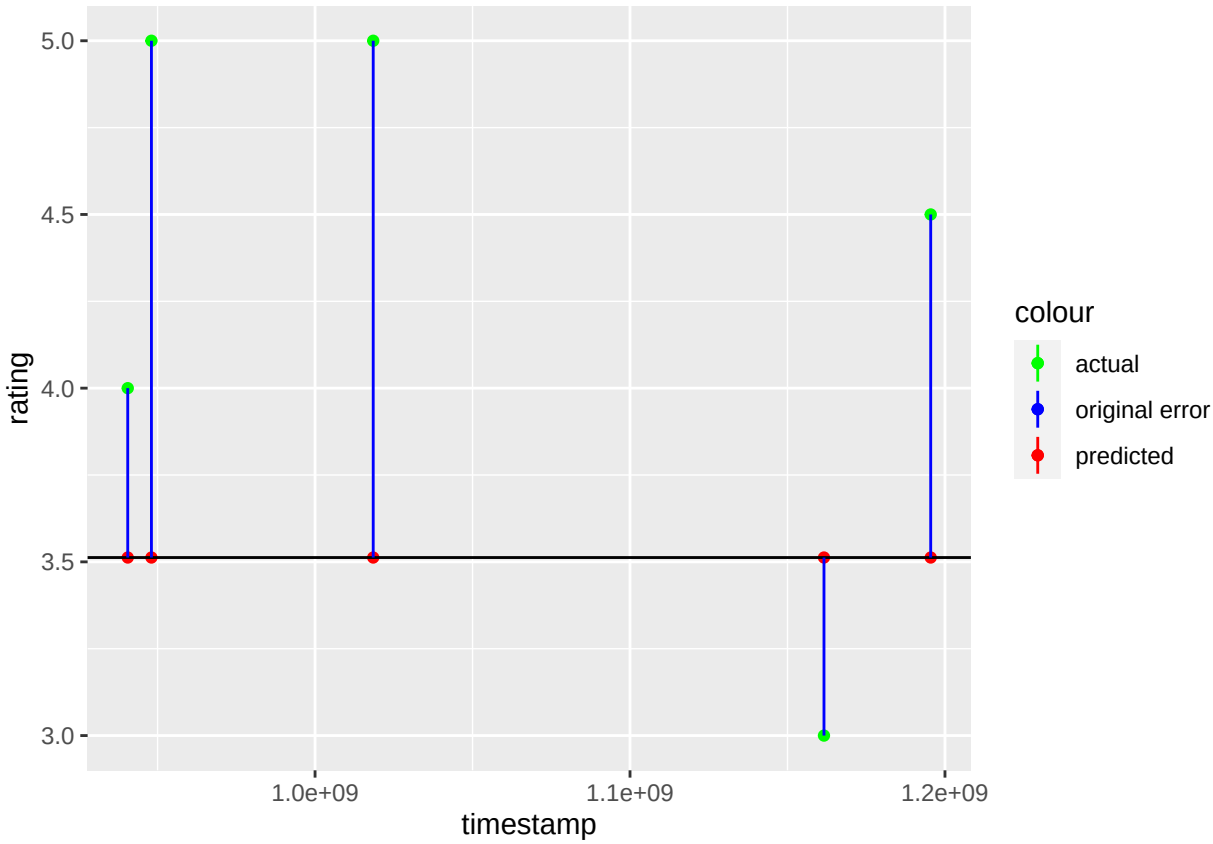
Understanding the model

There has been considerable study of collaborative filtering in existing literature [Koren, Irizarry], and we apply these insights below. We want to find a model for predicting a user u rating of an item i given other users ratings, as well as current users ratings of other items. This is typically modeled as the sum of multiple effects given by different features of the data set. That is, we explore the data set to try and find the main properties that seem to explain why a rating goes up or down. This is easier to understand with a visualization.

Let's take 5 random data points from movies with a significant amount of ratings and use them as a running example. Our first observation is that by taking the mean μ of all the ratings, we can start modeling our problem as $Y_{u,i} = \mu + \varepsilon_{u,i}$ where our prediction is just μ. Below we represent the mean with a black horizontal line, actual ratings in green, predicted ratings in red. $\varepsilon_{u,i}$ is the blue vertical line between μ and the data points. Notice that all the predictions lay in the mean; this model is not too useful, but a good starting point. We now want to continue making $\varepsilon_{u,i}$ smaller, by adding more effects in the equation that try to explain why a particular rating deviates from the mean.

```
mu <- mean(train_set$rating)

set.seed(1, sample.kind = "Rounding")
sample <- train_set %>%
  group_by(movieId) %>%
  filter(n() > 500) %>%
  ungroup() %>%
  sample_n(5)

sample %>%
  mutate(predicted = mu) %>% # we predict the mean for all points
  ggplot() +
  geom_point(aes(x = timestamp, y = rating, color = "actual")) +
  geom_point(aes(x = timestamp, y = predicted, color = "predicted")) +
  geom_hline(yintercept = mu) +
  geom_linerange(aes(
    x = timestamp,
    ymin = pmin(rating, mu),
    ymax = pmax(rating, mu),
```

```
    color = "original error"
  )) +
  scale_colour_manual(values = c("green", "blue", "red"))
```



With this base model, we can utilize our test set to calculate the current RMSE:

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

rmse_baseline <- RMSE(test_set$rating, mu)
print(paste("RMSE of average: ", rmse_baseline))
```

```
## [1] "RMSE of average:  1.05990428951531"
```

The item effect

As we observed in the previous visualization, most items will invariably be rated different than μ. In the case of movies, we know this by intuition: we like some movies much better than others. To account for this, we add a term that models the effect of the rated item:

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

We can use linear regression to calculate b_i, but with 7.2 million points in the train set, the calculation quickly becomes intractable. Thus we opt for grouping all the ratings for a particular item, and approximating its least squares solution:

```
regular_movie_effect <- train_set %>%
  group_by(movieId) %>%
```

```
  summarize(b_i = mean(rating - mu))
```

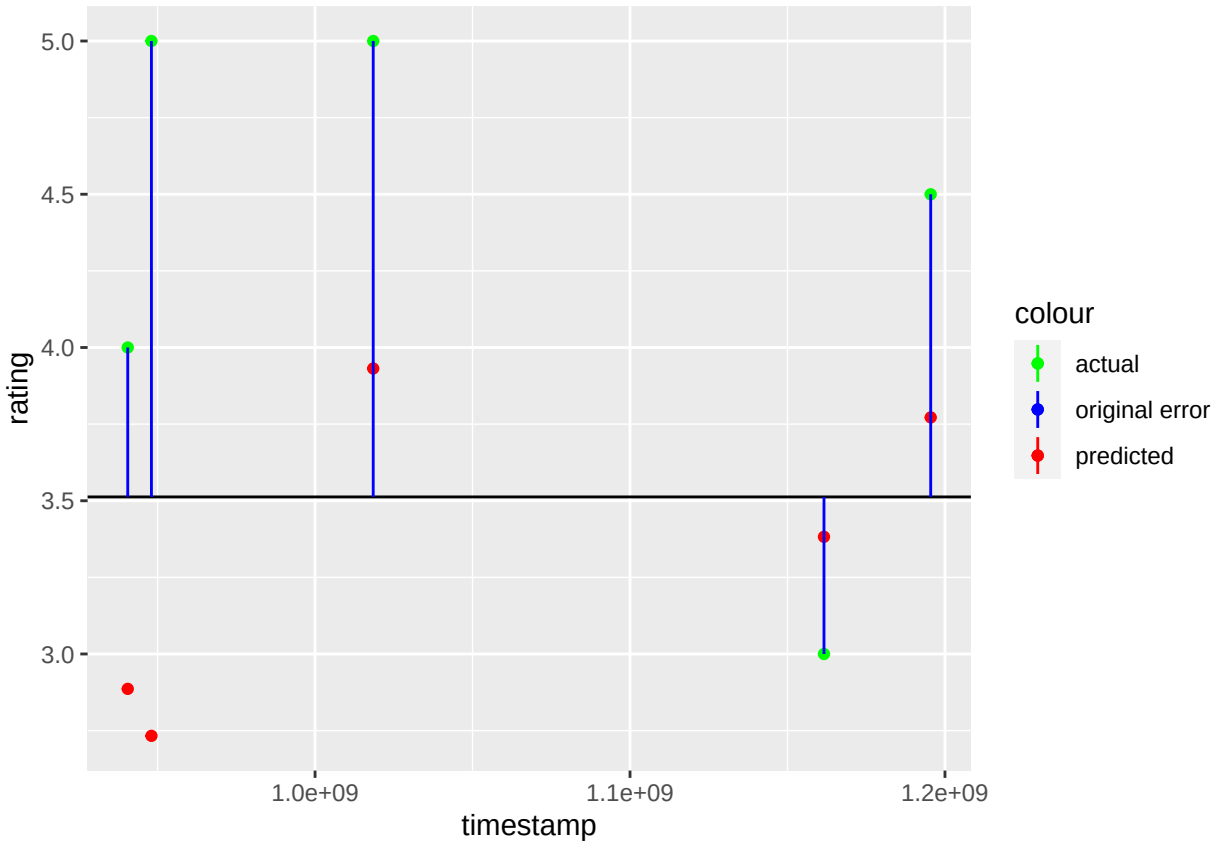
We now apply the modified model to the test set and compute the RMSE:

```
movie_effect_on_test_set <-
  test_set %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  mutate(model = mu + b_i)

rmse_movie_effect <- RMSE(test_set$rating, movie_effect_on_test_set$model)
print(paste("RMSE of average + movie effect: ", rmse_movie_effect))
```

```
## [1] "RMSE of average + movie effect:  0.9437429136878"
```

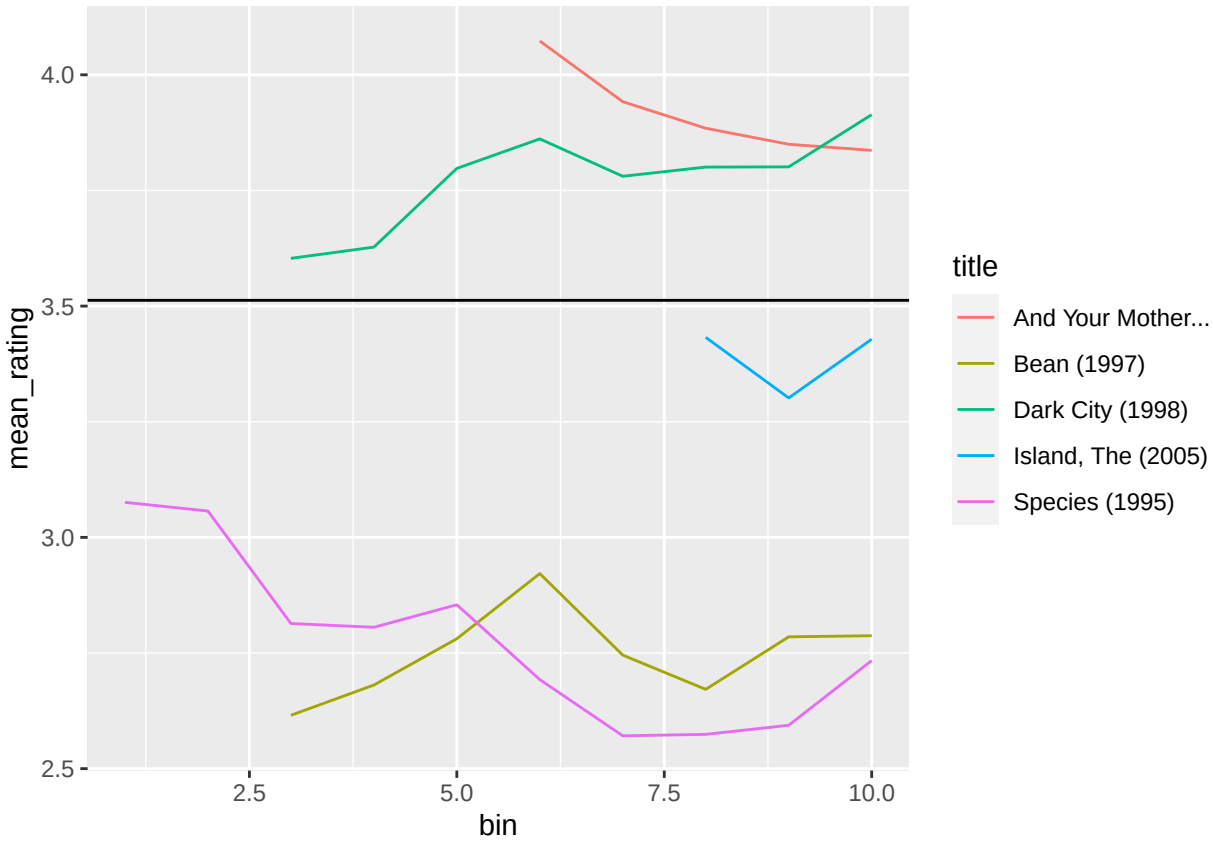
Going back to our running example, we visually confirm that indeed the movie effect materially changes the outcome. Notice that some predictions got better (red point closer to green point), while some others got worse! On average, however, we observed in the RMSE calculation that we are better off.

```
mu <- mean(train_set$rating)

sample %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  mutate(predicted = mu + b_i) %>%
  ggplot() +
  geom_point(aes(x = timestamp, y = rating, color = "actual")) +
  geom_point(aes(x = timestamp, y = predicted, color = "predicted")) +
  geom_hline(yintercept = mu) +
  geom_linerange(aes(
    x = timestamp,
    ymin = pmin(rating, mu),
    ymax = pmax(rating, mu),
    color = "original error"
  )) +
  scale_colour_manual(values = c("green", "blue", "red"))
```



Applying insight from the work of [Koren], we notice that there also seems to be a *temporal* item effect. That is, the mean rating of an item changes over time. Let's visualize this effect with the movies associated to our running example data points. In the code below we create temporal bins; we divide time into n bins and we put every rating into one of the bins. Later we calculate the mean rating of each movie for each bin.

```
movieIds <- sample %>% pull(movieId)

num_bins <- 10 # For now we set n = 10 arbitrarily.
max_timestamp <- max(train_set$timestamp)
min_timestamp <- min(train_set$timestamp)
bin_size <- (max_timestamp - min_timestamp) / num_bins

train_set %>%
  filter(movieId %in% movieIds) %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  group_by(movieId, bin) %>%
  summarise(mean_rating = mean(rating), title = str_trunc(first(title), 18)) %>%
  ggplot(aes(x = bin, y = mean_rating, color = title)) +
  geom_line() +
  geom_hline(yintercept = mu)
```



Notice how the movies' means do indeed change over time. This effect could be explained by seasonality of the movies, or changing taste of users over time. Regardless, it is there, so we modify our model to account for it:

$$Y_{u,i} = \mu + b_i(t) + \varepsilon_{u,i}$$

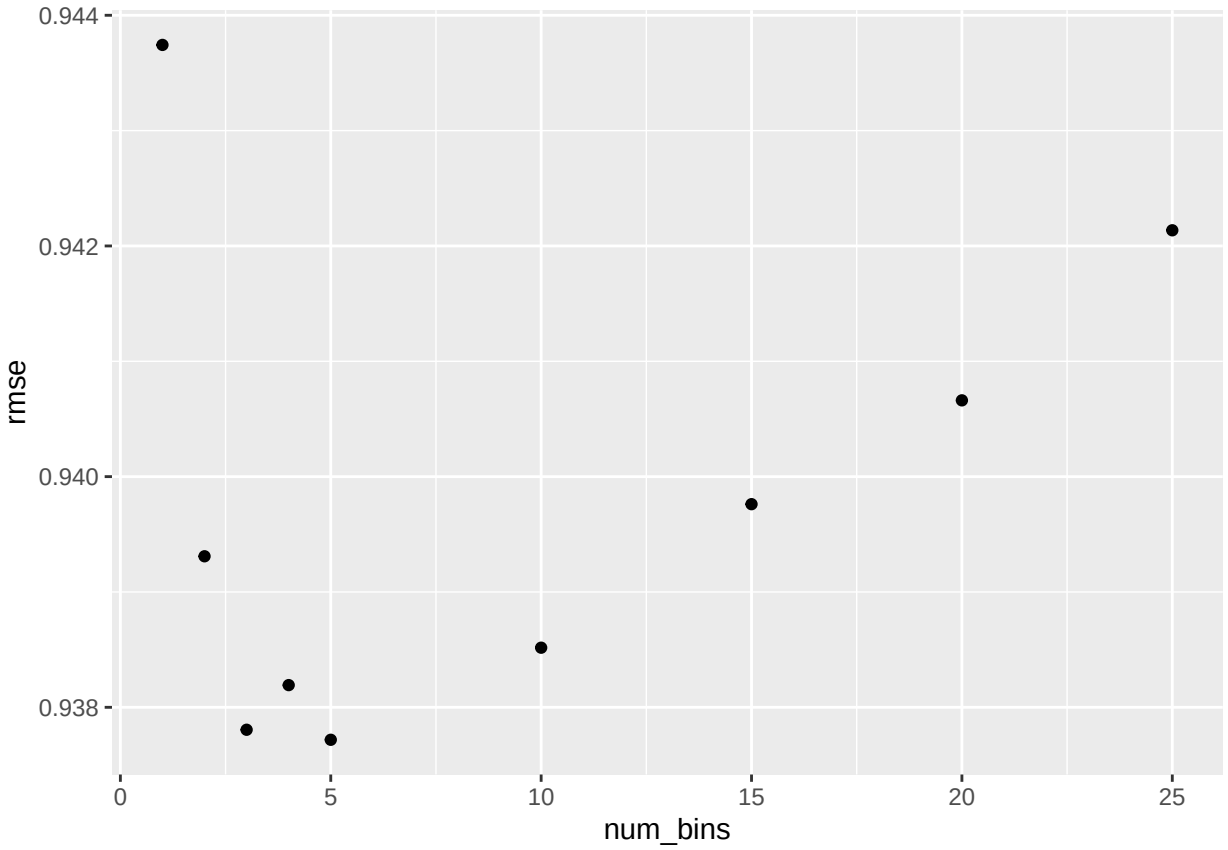
where:

$$b_i(t) = b_i + b_{i,Bin(t)}$$

Note the $b_{i,Bin(t)}$ term depends on both the item and the temporal bin. Before we had arbitrarily chosen $n = 10$ for illustration purposes, but to minimize the RMSE, the number of bins becomes a tuning parameter, and we use our test data and least squares to find it:

```
bin_range <- append(c(1, 2, 3, 4), seq(5, 25, 5))

binned_movie_effect_rmses <- sapply(bin_range, function(num_bins) {
  max_timestamp <- max(train_set$timestamp) + 86400 # one day margin to be safe
  min_timestamp <- min(train_set$timestamp) - 86400
  bin_size <- (max_timestamp - min_timestamp) / num_bins

  regular_movie_effect <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))

  binned_movie_effect <- train_set %>%
    left_join(regular_movie_effect, by = "movieId") %>%
    mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
    group_by(movieId, bin) %>%
```

```
    summarize(b_i_t = mean(rating - mu - b_i))

  movie_effect_on_test_set <-
    test_set %>%
    mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
    left_join(regular_movie_effect, by = "movieId") %>%
    left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
    # we coalesce b_i_t, as test set may have movie ratings in bins that we did not see
    # in the train set. Thus, in those cases, we default to 0.
    mutate(model = mu + b_i + coalesce(b_i_t, 0))

  RMSE(test_set$rating, movie_effect_on_test_set$model)
})

best_bin_num <- bin_range[which.min(binned_movie_effect_rmses)]

data.frame(num_bins = bin_range,
           rmse = binned_movie_effect_rmses) %>%
  ggplot(aes(x = num_bins, y = rmse)) +
  geom_point()
```



We find that 5 bins provide the best RMSE of:

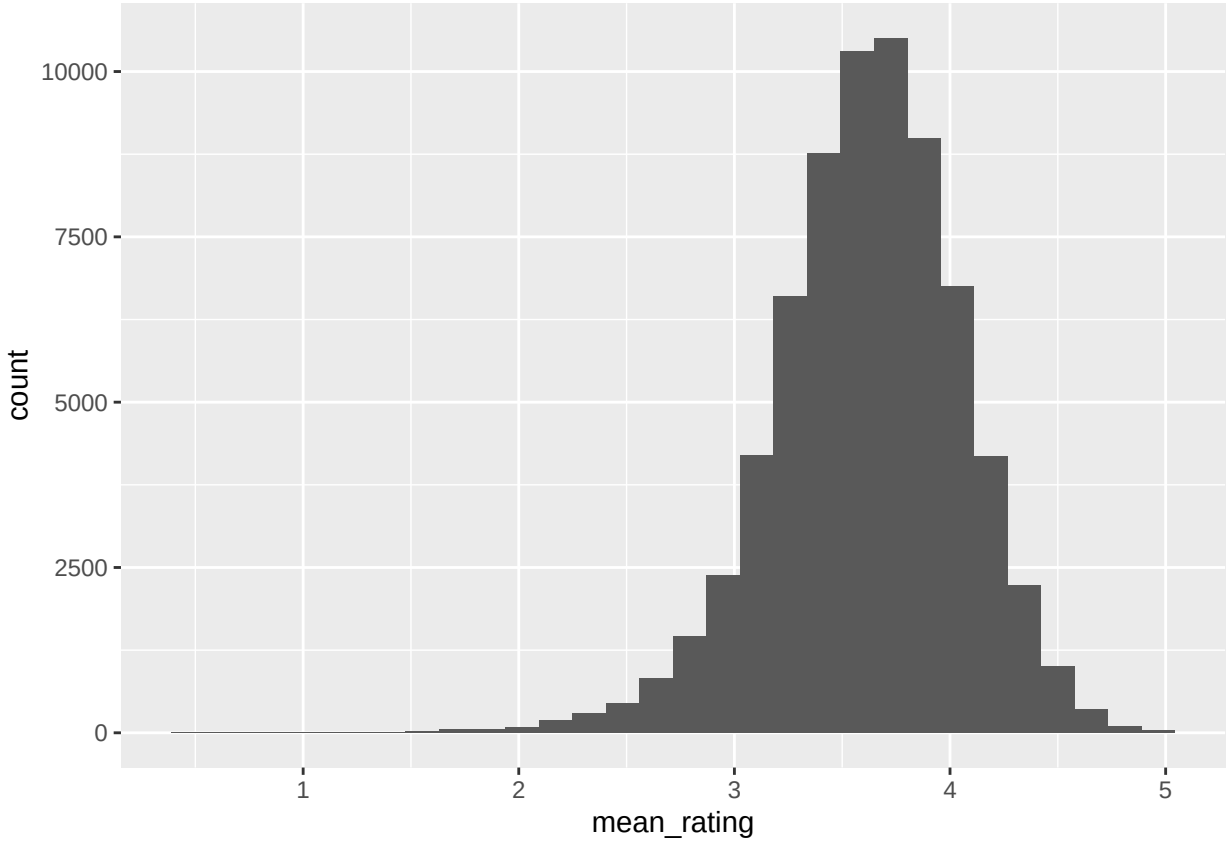
```
print(paste(
  "RMSE of average + temporal movie effect: ",
  min(binned_movie_effect_rmses)
))
```

```
## [1] "RMSE of average + temporal movie effect:  0.937718607530606"
```

The user effect

Some of us are harsh critics, some of us like most movies. This effect can be observed by looking at a histogram of average rating by user:

```
train_set %>%
  group_by(userId) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(x = mean_rating)) +
  geom_histogram(bins = 30)
```



We can account for this effect by adding one more term to our model:

$$Y_{u,i} = \mu + b_i(t) + b_u + \varepsilon_{u,i}$$

Just like with other effects, to compute b_u we group data by user, and approximate a least squares fit:

```
bin_size <- (max_timestamp - min_timestamp) / best_bin_num

regular_movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

binned_movie_effect <- train_set %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  group_by(movieId, bin) %>%
```

```
  summarize(b_i_t = mean(rating - mu - b_i))

user_effect <- train_set %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i - coalesce(b_i_t, 0)))
```

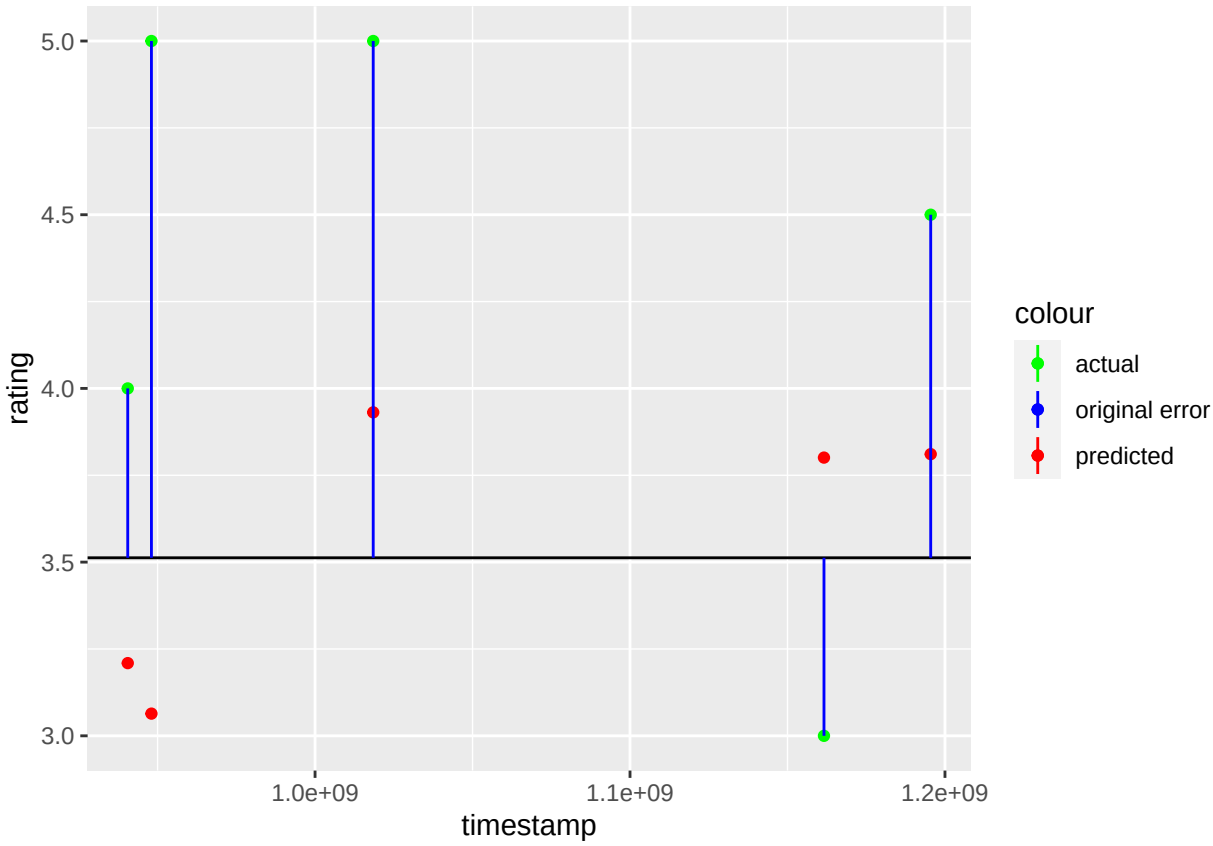
We now compute the RMSE on the test set:

```
user_effect_on_test_set <-
  test_set %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
  left_join(user_effect, by = "userId") %>%
  mutate(model = mu + b_i + coalesce(b_i_t, 0) + b_u)

rmse_movie_and_user_effect <-
  RMSE(test_set$rating, user_effect_on_test_set$model)
print(
  paste(
    "RMSE of baseline + temporal movie effect and user effect: ",
    rmse_movie_and_user_effect
  )
)
```

```
## [1] "RMSE of baseline + temporal movie effect and user effect:  0.862555756003579"
```

Finally, we confirm visually with the running example that there was material change:

```
mu <- mean(train_set$rating)

sample %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
  left_join(user_effect, by = "userId") %>%
  mutate(predicted = mu + b_i + coalesce(b_i_t, 0) + b_u) %>%
  ggplot() +
  geom_point(aes(x = timestamp, y = rating, color = "actual")) +
  geom_point(aes(x = timestamp, y = predicted, color = "predicted")) +
  geom_hline(yintercept = mu) +
  geom_linerange(aes(
    x = timestamp,
    ymin = pmin(rating, mu),
    ymax = pmax(rating, mu),
    color = "original error"
  )) +
  scale_colour_manual(values = c("green", "blue", "red"))
```



Regularization

The final improvement we do in this work is the application of regularization. We want to control the effect of ratings where we have low confidence. Movies that were rated by few users and users that made few ratings should weigh less. And we do have a significant amount of movies and users in that category:

```
# movies with few ratings
train_set %>%
  group_by(movieId) %>%
  summarise(n = n()) %>%
  filter(n <= 10) %>%
  select(n) %>%
  table()
```

```
## .
##   1   2   3   4   5   6   7   8   9  10
## 166 161 159 164 131 115 115 126 112 119
```

```
# users that made few ratings
train_set %>%
  group_by(userId) %>%
  summarise(n = n()) %>%
  filter(n <= 10) %>%
  select(n) %>%
  table()
```

```
## .
##  7  8  9 10
##  1  5 20 64
```

As per [Irizarry]:

> "Regularization permits us to penalize large estimates that are formed using small sample sizes... Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty."

Applying their insight to our model yields the following penalized least squares formula:

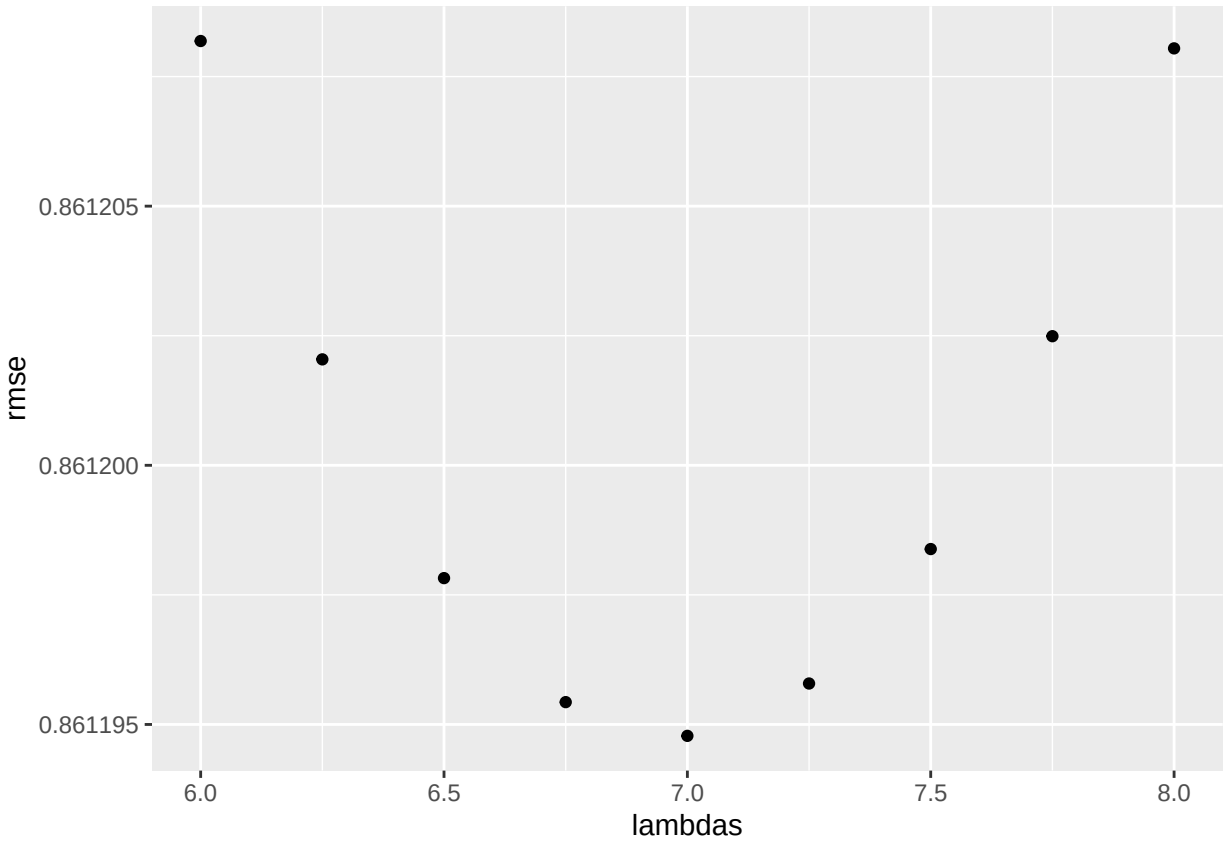
$$\sum_{u,i} \left(y_{u,i} - \mu - b_i - b_{i,Bin(t)} - b_u\right)^2 + \lambda \left(\sum_i b_i^2 + \sum_i b_{i,Bin(t)}^2 + \sum_u b_u^2\right)$$

It can be shown that the values for b_i, $b_{i,Bin(t)}$, and b_u that minimize the above equation are:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu}\right) \quad \hat{b}_{i,Bin(t)}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu} - \hat{b_i}\right) \quad \hat{b}_u(\lambda) = \frac{1}{\lambda + n_u} \sum_{i=1}^{n_u} \left(Y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_{i,Bin(t)}}\right)$$

Below we translate the equations into code. Note that the penalty λ becomes a tuning parameter:

```
lambdas <- seq(6, 8, 0.25)

regularized_rmses <- sapply(lambdas, function(lambda) {
  bin_size <- (max_timestamp - min_timestamp) / best_bin_num

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

  binned_movie_effect <- train_set %>%
    left_join(regular_movie_effect, by = "movieId") %>%
    mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
    group_by(movieId, bin) %>%
    summarize(b_i_t = sum(rating - mu - b_i) / (n() + lambda))

  user_effect <- train_set %>%
    mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
    left_join(regular_movie_effect, by = "movieId") %>%
    left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu - b_i - coalesce(b_i_t, 0)) / (n() + lambda))

  regularization_on_test_set <-
    test_set %>%
    mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
    left_join(regular_movie_effect, by = "movieId") %>%
    left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
    left_join(user_effect, by = "userId") %>%
    mutate(model = mu + b_i + coalesce(b_i_t, 0) + b_u)

  RMSE(test_set$rating, regularization_on_test_set$model)
})

data.frame(lambdas = lambdas,
           rmse = regularized_rmses) %>%
  ggplot(aes(x = lambdas, y = rmse)) +
  geom_point()
```




```
best_lambda <- lambdas[which.min(regularized_rmses)]
print(paste(
  "RMSE of baseline + temporal movie effect and user effect + regularization: ",
  min(regularized_rmses)
))
```

```
## [1] "RMSE of baseline + temporal movie effect and user effect + regularization:  0.861194780171966"
```

Results

We have iteratively improved our model as seen on our test set RMSE results:

```
##                   description      RMSE  difference
## 1                     Average 1.0599043          NA
## 2          Plus Movie effect 0.9437429 0.116161376
## 3 Plus Temporal movie effect 0.9377186 0.006024306
## 4           Plus User effect 0.8625558 0.075162852
## 5        Plus Regularization 0.8611948 0.001360976
```

We notice that we gain the most from the movie effect, as well as the user effect. The temporal movie effect and regularization gives us marginal benefits, but still help us reach our target RMSE.

We can now do a final evaluation with the validation set:

```
bin_size <- (max_timestamp - min_timestamp) / best_bin_num

regular_movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + best_lambda))
```

```
binned_movie_effect <- train_set %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  group_by(movieId, bin) %>%
  summarize(b_i_t = sum(rating - mu - b_i) / (n() + best_lambda))

user_effect <- train_set %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i - coalesce(b_i_t, 0)) / (n() + best_lambda))


results_on_validation <-
  validation %>%
  mutate(bin = ceiling((timestamp - min_timestamp) / bin_size)) %>%
  left_join(regular_movie_effect, by = "movieId") %>%
  left_join(binned_movie_effect, by = c("movieId", "bin")) %>%
  left_join(user_effect, by = "userId") %>%
  # coalesce b_i since validation set includes some unknown movies.
  mutate(model = mu + coalesce(b_i, 0) + coalesce(b_i_t, 0) + b_u)

rmse_validation = RMSE(validation$rating, results_on_validation$model)
print(paste("RMSE of final model on validation set: ", rmse_validation))
```

```
## [1] "RMSE of final model on validation set:  0.861446643841034"
```

The calculation of the tuning parameters are the most time-consuming parts of our solution. Of interest of practicality, the model, including the tuning and the fitting, can be built offline, meaning that a user of the system does not need to wait on it.

Conclusion

In this work, we have built a collaborative filtering system for the prediction of user ratings. Although we validate our system using a movie ratings data set, we note that the model could be applied to other rated items, such as books. We showed the derivation of all the terms in our model, and we used a visual running example to get a sense of the effect of newly added terms. Our model was fit with a technique that approximates the least squares method. We reached a RMSE of 0.8614466, which surpasses our target of `RMSE < 0.86490`.

The RMSE achieved is typical of models that use similar techniques as ours [Irizarry, Koren]. However, a divergence of ~0.86 stars may be limiting in practice. As per [Koren], we could explore other effects that could yield extra RMSE gains, such as evaluating whether there is a temporal effect on the user behavior, and also explore whether there are item to item and user to user effects. Additionally, we could add a factorization term that uses a technique such as singular value decomposition (SVD) to surface effects that are harder to explain. These techniques have been shown to significantly increase accuracy [Koren], but they also add to the runtime costs of the model, especially in the case of SVD. These conflicting goals would need to be taken into consideration once a production level solution is needed.

Citations

[Irizarry] : Rafael A. Irizarry. Introduction to Data Science: *Data Analysis and Prediction Algorithms with R.* Retrieved January 24, 2022 from https://rafalab.github.io/dsbook/index.html.

[Koren] : Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (*KDD '09*). Association for Computing Machinery, New York, NY, USA, 447–456. DOI:https://doi.org/10.1145/1557019.1557072. Available for free download at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.379.1951&rep=rep1&type=pdf.

[LeastSquares] : Wikipedia contributors. Least squares. Wikipedia, The Free Encyclopedia. January 10, 2022, 14:16 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Least_squares&oldid=1064847980. Accessed January 25, 2022.

[MovieLens] : F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872. Data set availabe for free download at: http://files.grouplens.org/datasets/movielens/ml-10m-README.html.