MovieLens Project: Creating a Film Recommendation Algorithm

Andrew Infantino

11 August 2020

Introduction

MovieLens is an online community and movie recommendation system operated at the University of Minnesota's Department of Computer Science and Engineering. [1] Through collaborative filtering, MovieLens demotes and promotes new movies to its individual users with respect to how they rate its movies on a five-star scale. The GroupLens Research laboratory processes the data to serve MovieLens users and to study personalization and filtering technologies. [2] The MovieLens database contained approximately 27 million ratings by 280,000 users for 58,000 movies as of 2018. The data, containing no user personal information, is freely available for public and commercial access. [3][4]

The purpose of this project is to independently create a movie recommendation algorithm from the MovieLens data. An inspiration for the project was the Netflix Prize, an open competition to redesign Netflix's movie recommendation algorithm. The contest was initiated in 2006 and concluded in 2009, awarding Yehuda Koren $1 million for his "Pragmatic Chaos" algorith, which yielded an RMSE of 0.8567, improving Netflix's algorithmic accuracy by 10.06% [5][6]

This project uses the 2009 edition of the MovieLens dataset, which consists of 10 million ratings by 72,000 users for 10,000 movies. [7] The dataset was downloaded and partitioned into a training set `edx`, containing 90% of the data, and a validation set `validation`, containing 10% of the data. (Appendix A) Programmed in R 3.6.2, the algorithm is trained to analyze rating patterns in `edx` and predict film ratings in `validation`. The root mean square error (RMSE) of the predicted ratings $\hat{y}_{u,i}$ from the true ratings $y_{u,i}$ by users u for films i will be the metric for evaluating the algorithm. Let N be the number of total user/film combinations. The formula and code for computing RMSE follow:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})}$$

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

An RMSE of fewer than 0.86490 stars is considered the benchmark for a successful algorithm in this project. The complete code used to develop and execute the algorithm is available in the attached script MovieLens_FinalProject.R. `edx` is partitioned into a test set `test` and a training set `train` to develop, train, and test the algorithm. `train` and `test` respectively contain 80% and 20% of the data in `edx`:

```
set.seed(1, sample.kind="Rounding")
index <- createDataPartition(y = edx$rating, times = 1,
                             p = 0.2, list = FALSE)
train <- edx[-index,]
test <- edx[index, ] %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")
```

The data in `train` is used to predict ratings in `test`. As the algorithm for doing so is developed, `RMSE` is intermittently used to compare predicted and real ratings. `RMSE` is finally used to evaluate the completed algorithm's `edx`-based predictions of `validation` ratings.

Analysis

The simplest model for predicting film ratings is one that assumes all ratings $y_{u,i}$ to be equal:

$$y_{u,i} = \mu + \epsilon_{u,i}$$

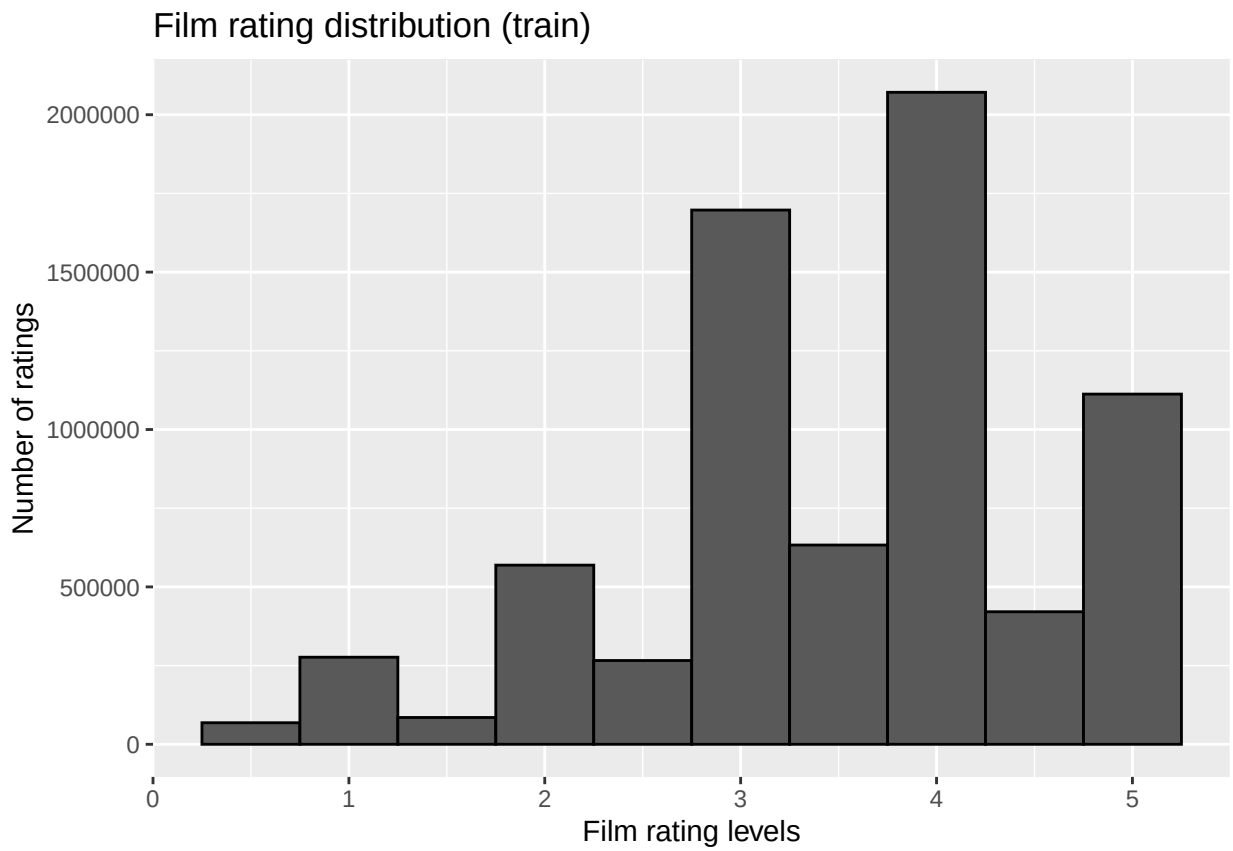
Let μ be the common value for all ratings and $\epsilon_{u,i}$ be the observed random error for each individual rating $y_{u,i}$. All variation from the common rating is thus assumed to be random. The predicted common value $\hat{\mu}$ is calculated as the average of all `train` ratings. The standard deviation of the `train` ratings is calculated with it:

```
## Average rating =  3.512482  stars.

## Standard deviation =  1.060437  stars.

## 70.67062 % of 'train' ratings range from  2.452045 to  4.572919  stars.
```

The above parameters indicate that more than 70% of the `train` ratings fall within a standard deviation from the mean rating. Note, the given percentage is somewhat higher than one would expect with a normal distribution [8] — all MovieLens ratings were collected in increments of 0.5 stars and relatively few users rated in terms of half-stars rather than only whole stars, yielding a data distribution not perfectly normal.



Nonetheless, 70% of the films recommended by the model above would vary in user favorability from roughly as low as *Star Wars: Episode I — The Phantom Menace* to roughly as high as *The Lord of the Rings: The Two Towers*. (Appendix B) The imprecision with which the model would predict user preferences renders it unlikely to yield high user satisfaction rates. Expectedly, using the model to predict `test` ratings, as seen below, yields an RMSE roughly equal to the standard deviation reported above:

```
##           Method     RMSE
## 1 Simple average 1.059904
```

The result above is not acceptable for the purpose of this project. Rafael A. Irizarry developed a more complex model that incorporates how individual films tend to be rated and how individual users tend to rate films. [9] It hypothesizes certain systemic patterns, rather than only randomness, to perpetuate rating variance - some films are generally rated more highly than others are; some users generally rate films more selectively or "harshly" than others do. The model follows:

$$y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

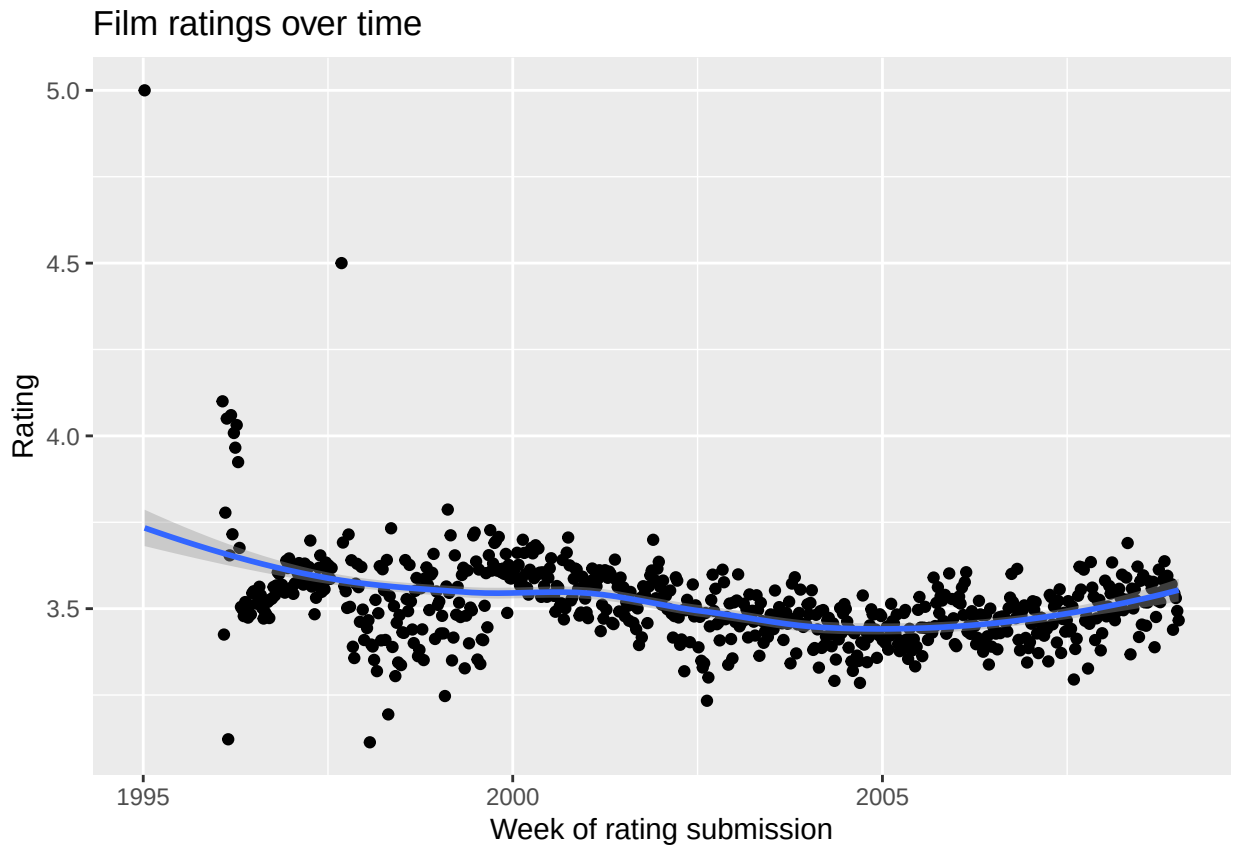
The predicted effect $\hat{b}_i$ of each film i is calculated as the average difference between all of film i's ratings $y_{u,i}$ and $\hat{\mu}$. The predicted user effect $\hat{b}_u$ for each user u is then calculated as the average difference between between all of user u's ratings $y_{u,i}$, $\hat{\mu}$, and the predicted film effects $\hat{b}_i$ of all user u's rated films. Irizarry's model is thus coded below and yields the following RMSE value:

```
b_i_hat <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
b_u_hat <- train %>%
  left_join(b_i_hat, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
rai_predictions <- test %>%
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred
```

```
##                   Method     RMSE
## 1        Simple average 1.059904
## 2 Film and user effects 0.865932
```

The results are significantly better but still need improvement. Film ratings tend to generally vary somewhat over time:

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Irizarry's model can incorporate time effects:

$$y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \epsilon_{u,i}$$

Let f be a smooth function of the date $d_{u,i}$ user u rates film i. Let n_d be the total number of ratings submitted on a date d. The predicted value of $f(d_{u,i})$ for each d can be approximated as $\hat{b}_d$:

$$\hat{b}_d = \frac{1}{n_d} \sum_{u,i=1}^{n_d} (y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

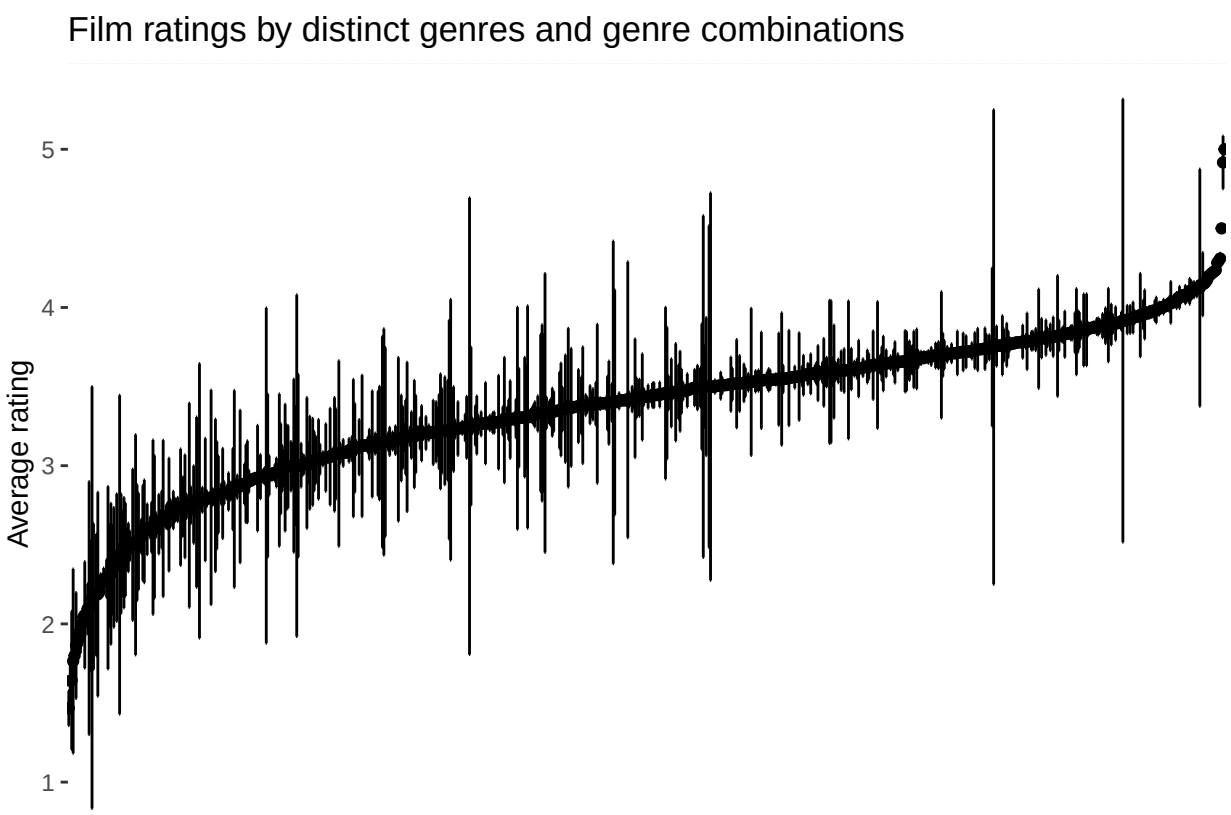
Below, the time effects are coded into Irizarry's model and tested. Note, the date d is rounded to the nearest week.

```
b_d_hat <- train %>%
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  group_by(week) %>%
  summarize(b_d = mean(rating - mu_hat - b_i - b_u))
date_predictions <- test %>%
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  left_join(b_d_hat, by = 'week') %>%
  mutate(pred = mu_hat + b_i + b_u + b_d) %>%
  .$pred
```

```
##                          Method      RMSE
```

```
## 1               Simple average 1.0599043
## 2        Film and user effects 0.8659320
## 3 Film, user, and time effects 0.8658367
```

There is more significant variation in film ratings by distinct genre and genre combination:



Note, the x-axis is not labelled above. `train` contains hundreds of genres and genre combinations (Appendix B), which are too numerous to mark discretely on the x-axis. Irizarry's model can incorporate genre effects:

$$y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + \epsilon_{u,i}$$

Define $g_{u,i}$ as the genre for a film i that user u rates. Let $x_{u,i} = 1$ if $g_{u,i}$ is genre k. One can represent $\sum_{k=1}^{K} x_{u,i}\beta_k$ more simply as b_g, treating genre combinations as distinct genres g themselves. The expected value of a genre's effect is formulated, coded into Irizarry's model, and tested below:

$$\hat{b}_g = \frac{1}{n_g} \sum_{u,i=1}^{n_g} (y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

```
b_g_hat <- train %>%
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))
genre_predictions <- test %>%
```

```
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  left_join(b_g_hat, by = 'genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred
```

```
##                           Method      RMSE
## 1                  Simple average 1.0599043
## 2           Film and user effects 0.8659320
## 3    Film, user, and time effects 0.8658367
## 4 Film, user, and genre effects 0.8655941
```

After reformulating the predicted genre effects to account for predicted time effects, a predictive model using film, user, time, and genre effects altogether is formulated, coded, and tested below:

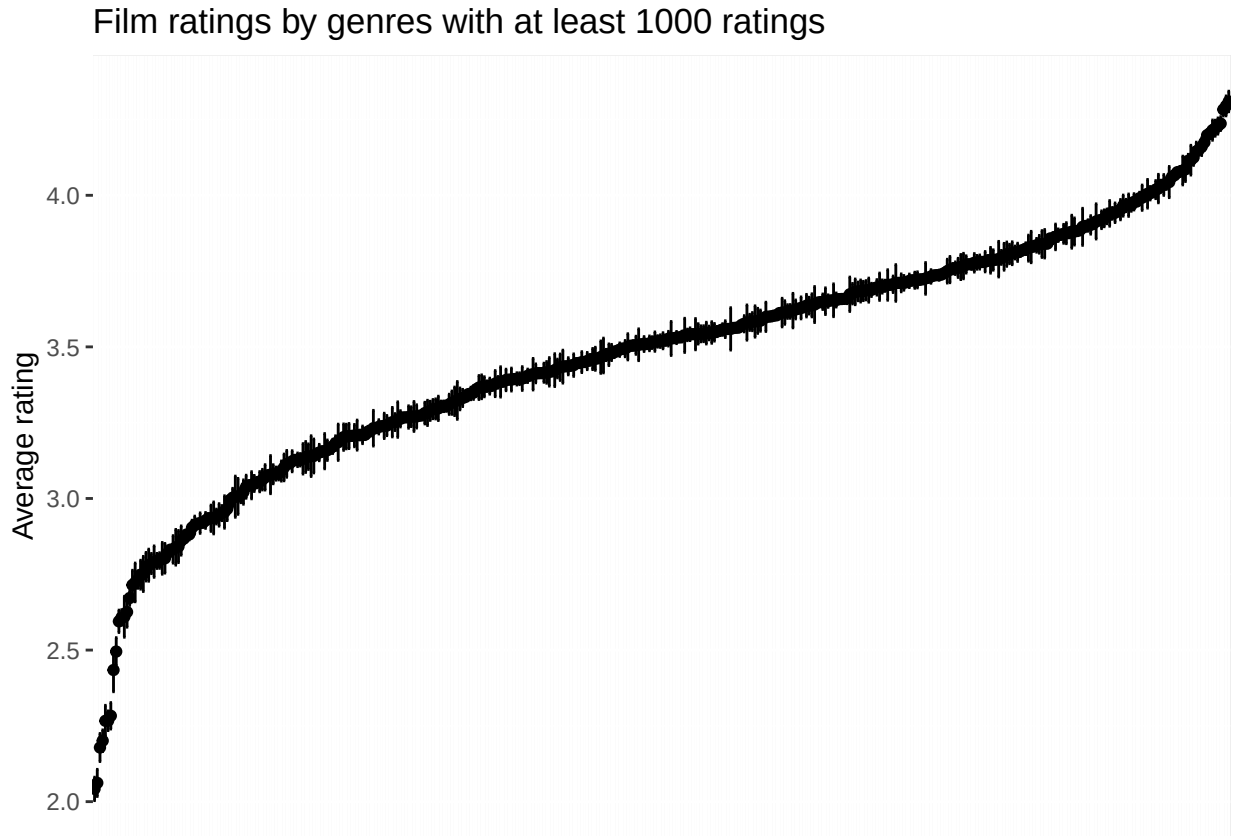
$$\hat{b}_g = \frac{1}{n_g} \sum_{u,i=1}^{n_g} (y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_d)$$

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{b}_d + \hat{b}_g$$

```
b_g_hat <- train %>%
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  left_join(b_d_hat, by = 'week') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u - b_d))
futg_predictions <- test %>%
  left_join(b_i_hat, by = 'movieId') %>%
  left_join(b_u_hat, by = 'userId') %>%
  left_join(b_d_hat, by = 'week') %>%
  left_join(b_g_hat, by = 'genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_d + b_g) %>%
  .$pred
```

```
##                               Method      RMSE
## 1                      Simple average 1.0599043
## 2               Film and user effects 0.8659320
## 3        Film, user, and time effects 0.8658367
## 4       Film, user, and genre effects 0.8655941
## 5 Film, user, time, and genre effects 0.8654976
```

As seen in the graph above, some genres yield very high margins of error. They tend to have relatively low numbers of ratings. Including only genres that have at least one thousand ratings, the relationship between genre and rating is visibly more stable:



The implications of the relationship between genre obscurity and rating predictability will soon be further analyzed. Irizarry noted that the model's most highly recommended films tend to be relatively obscure, containing low numbers of ratings: [10]

```
## # A tibble: 10 x 3
##    title                                                      rating count
##    <chr>                                                       <dbl> <int>
##  1 Hellhounds on My Trail (1999)                                 5        1
##  2 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to ~   5        3
##  3 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                           5        2
##  4 Shadows of Forgotten Ancestors (1964)                         5        1
##  5 Money (Argent, L') (1983)                                     5        1
##  6 Fighting Elegy (Kenka erejii) (1966)                          5        1
##  7 Sun Alley (Sonnenallee) (1999)                                5        1
##  8 Aerial, The (La Antena) (2007)                                5        1
##  9 Blue Light, The (Das Blaue Licht) (1932)                      5        1
## 10 More (1998)                                                   4.92     6
```

Small numbers of ratings yield high margins of error. Likewise, it's difficult to predict preferences for users who submit few ratings. To minimize the weight of obscure films and inactive users, Irizarry uses regularization, which entails penalizing extreme estimates that derive from small sample sizes. The goal is to constrain the total variability of the effect sizes. [10] Irizarry uses penalized least squares, which, in terms of regularizing film effects, entails minimizing the expression below:

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

The first term above is simply the sum of least squares. The second term is a penalty becomes large when many b_i values are large. It is minimized by recalculating expected film effects $\hat{b}_i$ with respect to a tuning parameter λ and the total number n_i of ratings for a film i:

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

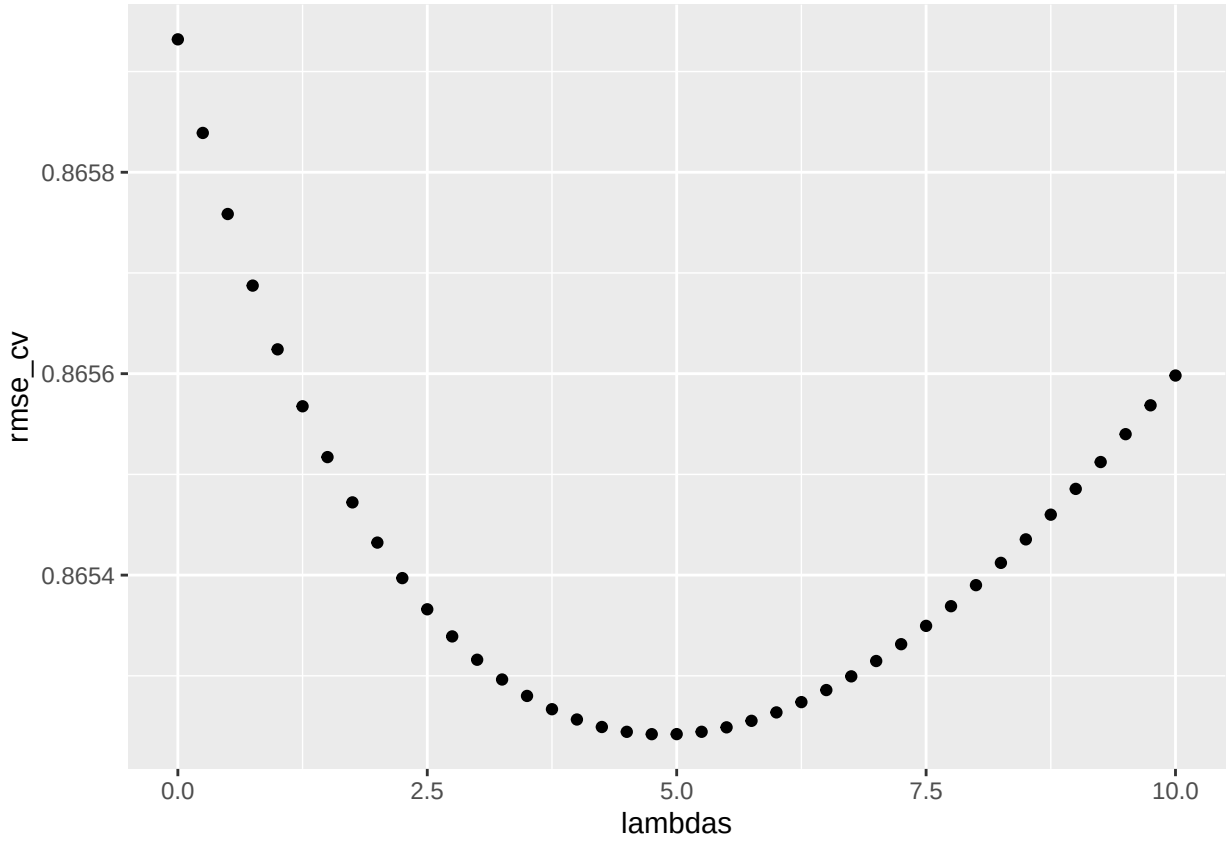
When n_i is significantly larger than λ, the expected effect for the corresponding film i doesn't change. Otherwise, the effect is reduced toward zero, giving obscure films equal or nearly equal weights to those of films with truly average ratings. To regularize user effects in addition to film effects, the expression below is minimized by recalculating the predicted user effects $\hat{b}_u$ with respect to λ:

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

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

All regularized variables within the model use a single tuning parameter λ. The following code is run to determine its best value, that which yields the lowest RMSE:

```
lambdas <- seq(0, 10, 0.25)
rmse_cv <- sapply(lambdas, function(l){
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l))
  b_u <- train %>%
    left_join(b_i, by = 'movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_i)/(n()+l))
  predictions <- test %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predictions, test$rating))
})
qplot(lambdas, rmse_cv)
```




```
## Tuning parameter for regularized film and user effects =  4.75
```

Irizarry's regularized model is then coded and tested below:

```
lambda <- lambdas[which.min(rmse_cv)]
b_i_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
b_u_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda))
rai_reg_pred <- test %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred
```

```
##                               Method      RMSE
## 1                      Simple average 1.0599043
## 2               Film and user effects 0.8659320
## 3        Film, user, and time effects 0.8658367
## 4       Film, user, and genre effects 0.8655941
## 5 Film, user, time, and genre effects 0.8654976
## 6   Regularized film and user effects 0.8652421
```

Regularization improves the results. To yield more precision, one can regularize genre effects by minimizing the following expression and recalculating $\hat{b}_g$ below:

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

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

A model featuring regularized film, user, and genre effects is thus formulated, coded with a new tuning parameter, and tested:

$$\hat{y}_{u,i}=\hat{\mu}+\hat{b}_u(\lambda)+\hat{b}_i(\lambda)+\hat{b}_g(\lambda)$$

```
## Tuning parameter for regularized film, user, and genre effects =  4.75
```

```
lambda <- lambdas[which.min(rmse_cv)]
b_i_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
b_u_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda))
b_g_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_i - b_u)/(n()+lambda))
genre_reg_pred <- test %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  left_join(b_g_reg, by = 'genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred
```

```
##                                     Method      RMSE
## 1                            Simple average 1.0599043
## 2                      Film and user effects 0.8659320
## 3               Film, user, and time effects 0.8658367
## 4              Film, user, and genre effects 0.8655941
## 5       Film, user, time, and genre effects 0.8654976
## 6         Regularized film and user effects 0.8652421
## 7 Regularized film, user, and genre effects 0.8649406
```

While regularizing time effects may yield the smallest change, it may still be useful to do so as general user activity may fluctuate over time. Below, the penalized least squares expression and equation for reguarlized time effects $\hat{b}_d(\lambda)$ are formulated, then the model is formulated, coded, and tested to regularized include film, user, and time effects:

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

$$\hat{b}_d(\lambda) = \frac{1}{\lambda + n_d} \sum_{u,i=1}^{n_d} (y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u)$$

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_u(\lambda) + \hat{b}_i(\lambda) + \hat{b}_d(\lambda)$$

```
## Tuning parameter for regularized film, user, and time effects =  5.25
```

```
lambda <- lambdas[which.min(rmse_cv)]
b_i_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
b_u_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda))
b_d_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  group_by(week) %>%
  summarize(b_d = sum(rating - mu_hat - b_i - b_u)/(n()+lambda))
date_reg_pred <- test %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  left_join(b_d_reg, by = 'week') %>%
  mutate(pred = mu_hat + b_i + b_u + b_d) %>%
  .$pred
```

```
##                                     Method      RMSE
## 1                            Simple average 1.0599043
## 2                      Film and user effects 0.8659320
## 3               Film, user, and time effects 0.8658367
## 4              Film, user, and genre effects 0.8655941
## 5       Film, user, time, and genre effects 0.8654976
## 6         Regularized film and user effects 0.8652421
## 7 Regularized film, user, and genre effects 0.8649406
## 8  Regularized film, user, and time effects 0.8651154
```

Re-define the predicted regularized genre effect to account for time effects:

$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_d)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2 + \sum_d b_d^2 + \sum_g b_g^2)$$

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

Finally, a model that includes regularized film, user, time, and genre effects is formulated, coded, and tested:

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_u(\lambda) + \hat{b}_i(\lambda) + \hat{b}_d(\lambda) + \hat{b}_g(\lambda)$$

```
## Tuning parameter for regularized film, user, time, and genre effects =  5
```

```
lambda <- lambdas[which.min(rmse_cv)]
b_i_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
b_u_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda))
b_d_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  group_by(week) %>%
  summarize(b_d = sum(rating - mu_hat - b_i - b_u)/(n()+lambda))
b_g_reg <- train %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  left_join(b_d_reg, by = 'week') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_hat - b_i - b_u - b_d)/(n()+lambda))
full_reg_pred <- test %>%
  left_join(b_i_reg, by = 'movieId') %>%
  left_join(b_u_reg, by = 'userId') %>%
  left_join(b_d_reg, by = 'week') %>%
  left_join(b_g_reg, by = 'genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_d + b_g) %>%
  .$pred
```

```
##                                        Method      RMSE
## 1                                Simple average 1.0599043
## 2                           Film and user effects 0.8659320
## 3                     Film, user, and time effects 0.8658367
## 4                    Film, user, and genre effects 0.8655941
## 5              Film, user, time, and genre effects 0.8654976
## 6                Regularized film and user effects 0.8652421
## 7       Regularized film, user, and genre effects 0.8649406
## 8        Regularized film, user, and time effects 0.8651154
## 9 Regularized film, user, time, and genre effects 0.8647936
```

The results above suggest that a model accounting for regularized film, user, time, and genre effects would suffice best as a for a film recommendation algorithm. The final tuning parameter calculated above will be used to test the model on the full MovieLens dataset.

Results

The model below is trained with respect to `edx` data — which contains 90% of the data used for this project — to predict ratings in `validation`. It accounts for regularized film, user, date, and genre effects:

$$y_{u,i} = \mu + b_i(\lambda) + b_u(\lambda) + b_d(\lambda) + b_g(\lambda) + \epsilon_{u,i}$$

Recoded and tested with respect to `edx` and `validation` data while using a tuning parameter of 5, it yields the following RMSE:

```
##                                             Method      RMSE
## 1                                    Simple average 1.0599043
## 2                              Film and user effects 0.8659320
## 3                       Film, user, and time effects 0.8658367
## 4                      Film, user, and genre effects 0.8655941
## 5               Film, user, time, and genre effects 0.8654976
## 6                 Regularized film and user effects 0.8652421
## 7         Regularized film, user, and genre effects 0.8649406
## 8          Regularized film, user, and time effects 0.8651154
## 9  Regularized film, user, time, and genre effects 0.8647936
## 10                                      Final model 0.8643110
```

The final model predicts `validation` ratings even more accurately than it predicts `test` ratings, yielding a lower RMSE. The lower RMSE results indicate less observed error between real and predicted ratings that can be quantitatively attributed to random or unexplained variance. The final RMSE falls below the benchmark of 0.86490, thus the project is successful.

Conclusion

The purpose of this project was to develop a film recommendation algorithm from a 2009 MovieLens dataset consisting of 10 million ratings by 72,000 users for 10,000 films. The root mean square error (RMSE) between the predicted and real ratings of the dataset served as the metric by which the model was evaluated. Rafael A. Irizarry's models served as the basis for developing the final model used in this project. Irizarry predicted all ratings to deviate from an average rating with respect to how individual users rated certain films. Additionally, some random variance was expected. Applying regularization, he scaled the weight of film and user rating averages on his predictions with respect to film notability and user activity, reducing the erroneous effects of obscure films and inactive users. The final model of this project predicted film ratings to deviate from an average rating with respect to the regularized average ratings of individual films, users, times, and genres. Some randomness was also expected in the final model.

The final model yielded an RMSE lower than 0.86490 stars, meriting a success for the purposes of this project. Such a result, however, may not be considered satisfactory within industrial or academic research contexts; the context of this project is simply educational. Furthermore, the 2009 data used to train this project's algorithm may not suffice in predicting ratings of films released over the past decade. Consumer reception of existing films may have also changed significantly during that time period. Fans of the *Star Wars* film franchise, for example, may have re-evaluated George Lucas's 1999-2005 prequel series more favorably after viewing Disney's 2015-2019 sequel series. [11]

This paper did not evaluate the significance of each effect — film, user, date, and genre — independently of each other in predicting film ratings. Future research may benefit from doing so; effects with more weight should receive more experimental attention. Diminishing returns in RMSE losses were observed as more effects were modelled, which may obscure the true hierarchy of significance. Several additional methods can be implemented to yield even sharper RMSE losses. Matrix factorization and principal component analysis, for example, would allow researchers to model the effects of film and user groupings that conventional categories such as genre do not capture. [12] Yehuda Koren's prize-winning model incorporates a combination of matrix factorization and regularization along with several more sophisticated approaches. [13] [14] Such approaches involve mathematical operations that were too complicated to perform with the available hardware on datasets as expansive as the one used for this project. Future research should explore the use of supercomputing to perform more accurate and efficient analyses.

References

[1] "What is GroupLens?" (2020). Retrieved July 31, 2020, from https://grouplens.org/about/what-is-grouplens/

[2] About MovieLens (2020). Retrieved July 31, 2020, from https://movielens.org/info/about

[3] "MovieLens Latest Datasets" (2020). Retrieved July 31, 2020, from https://grouplens.org/datasets/movielens/latest/

[4] Morrow, A. (2020). MovieLens Database. Retrieved July 31, 2020, from http://license.umn.edu/technologies/z05173_movielens-database

[5] "Netflix Prize," Wikipedia (April 17, 2020). Retrieved July 31, 2020, from https://en.wikipedia.org/wiki/Netflix_Prize

[6] "Grand Prize awarded to Team BellKor's Pragmatic Chaos" (September 18, 2009). Retrieved July 31, 2020, from https://web.archive.org/web/20120507045732/http://www.netflixprize.com/community/viewtopic.php?id=1537

[7] "MovieLens 10M Dataset" (2020). Retrieved July 31, 2020, from https://grouplens.org/datasets/movielens/10m/

[8] "68–95–99.7 rule," Wikipedia (July 23, 2020). Retrieved August 2, 2020, from https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule

[9] Irizarry, R.A. (2019). Large datasets: Recommendation systems. *Introduction to Data Science* (online). Retrieved August 3, 2020 from https://rafalab.github.io/dsbook/index.html

[10] Irizarry, R.A. (2019). Large datasets: Regularization. *Introduction to Data Science* (online). Retrieved August 3, 2020 from https://rafalab.github.io/dsbook/index.html

[11] Tylor, A. (February 22, 2020) "Why Star Wars Fans Have Forgiven George Lucas," Screen Rant. Retrieved August 9, 2020, from https://screenrant.com/star-wars-fans-george-lucas-forgive-prequels-reason/

[12] Irizarry, R.A. (2019). Large datasets: Matrix factorization. *Introduction to Data Science* (online). Retrieved August 9, 2020 from https://rafalab.github.io/dsbook/index.html

[13] Chen, E. (October 24, 2011) "Winning the Netflix Prize: A Summary," Edwin chen's Blog. Retrieved August 9, 2020, from http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/

[14] Koren, Y. (August, 2009) "The BellKor Solution to the Netflix Grand Prize," Netflix Prize. Retrieved August 9, 2020, from https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

Appendix A

Before downloading and analyzing the MovieLens data, the R packages `tidyverse`, `caret`, and `data.table` are loaded or installed:

```
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
```

The MovieLens dataset is then downloaded as a temp file and ultimately processed into the dataframe `movielens` for analysis in R 3.6.2:

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
```

```
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

The data in `movielens` are then randomly partitioned into the training and validation sets `edx` and `validation` respectively. The validation set contains 10% of the original data and only data for users and films contained in the training set:

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")
```

Rows removed from `validation` are reincluded in `edx`. All data (including `movielens`) external to the training and validation sets are then deleted:

```
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Appendix B

The five most commonly rated films in `train` with a score of 2.5 follow below. Among them is *Star Wars: Episode I — The Phantom Menace*:

```
## # A tibble: 5 x 3
##   movieId     N `title[1]`
##     <dbl> <int> <chr>
## 1     780   843 Independence Day (a.k.a. ID4) (1996)
## 2    2628   751 Star Wars: Episode I - The Phantom Menace (1999)
## 3    1917   669 Armageddon (1998)
## 4    1721   663 Titanic (1997)
## 5    2054   663 Honey, I Shrunk the Kids (1989)
```

The five most commonly rated films in `train` with a score of 4.5 follow below. Among them is *The Lord of the Rings: The Two Towers*:

```
## # A tibble: 5 x 3
##   movieId     N `title[1]`
##     <dbl> <int> <chr>
```

```
## 1     318  2247 Shawshank Redemption, The (1994)
## 2    2571  2218 Matrix, The (1999)
## 3     296  2204 Pulp Fiction (1994)
## 4    5952  2058 Lord of the Rings: The Two Towers, The (2002)
## 5    4993  2047 Lord of the Rings: The Fellowship of the Ring, The (2001)
```

The number of distinct genres and genre combinations in `train` follows:

```
## [1] 797
```