MovieLens Rating Recommendation

Rajesh Haridas

November 15, 2021

Preface

This document and the associated code is based on the Capstone HarvardX Professional Certificate in Data Science (PH125.9x) course work and additional reading material provided in the course.

My project work on the ratings and RMSE computation for the MovieLens system is included in this and dependent files. This file references **MovieLens.R** and the given code is included in ProjGivenCode.R.

The KnitR package has generated html and pdf reports that are included with this rmd file

The following files (3 types) are included in the upload:

- **MovieLens.rmd** - Markdown for the main summary report file
- **MovieLens.R** - Main R code for the project
- **ProjGivenCode.R** - The given R code for the project
- **MovieLensSummaryReport.pdf** - Main summary report containing analysis
- **MovieLensExecutionReport.pdf** - Main execution report containing output of the runs

Contents

List of Figures

Introduction to MovieLens rating recommendation

MovieLens dataset was created by the GroupLens project. The GroupLens Research Project is a research group in the Department of Computer Science and Engineering at the University of Minnesota. According to the summary of the dataset, it contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommendation service MovieLens. More information can be found at https://files.grouplens.org/datasets/movielens/ml-10m-README.html.

In this project for the Capstone course of the HarvardX Professional Certificate in Data Science (PH125.9x), we will explore the MovieLens data set. The objective will be to develop a machine-learning model by creating training and test sets to predict movie ratings on a validation set that achieves a Root Mean Square Error (RMSE) of less than 0.8649

Here is a glimpse of edx dataset. The columns that are of interest are userId, movieId, and rating

```
# glimpse(edx)

# Rows: 9,000,055

# Columns: 6

# $ userId <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2~
# $ movieId <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420,
# 4~ $ rating <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
# 5~ $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392,
# 838984~ $ title <chr> 'Boomerang (1992)', 'Net, The (1995)', 'Outbreak
# (1995)', 'St~ $ genres <chr> 'Comedy|Romance', 'Action|Crime|Thriller',
# 'Action|Drama|Sci-~
```

The **goal** of this capstone exercise is to compute the predicted ratings and RMSE associated with the ratings to build a recommendation system. We will build this recommendation by applying different models with different predictors and progressively reduce the RMSE (error) in the recommendation. The goal is to keep errors less than 0.86490 RMSE.

The **outcome** is rating and the **predictors** are movie and user identified by the movieId and userId respectively. RMSE and predictions are calculated for the following scenarios

- predicted rating and RMSE with just the movie as the predictor

- predicted rating and RMSE with movie and user
- regularized rating prediction and RMSE for user and movie

RMSE is used to evaluate how close the predictions are to the true values in the validation set. All RMSEs are tabulated in the results section of this report. The details on how the results were arrived at are also in a subsection.

Due to the size of the dataset, most of the caret package functions for train and validation do not work due to the inability to load large vectors into memory. I have used manual least squared estimates to compute bias instead of using caret package. All analysis for computing the predictions and RMSE has been done with the edx dataset. 90% of the edx dataset is used for training set and the 10% of the dataset is used for test set.

The validation data set is not used for training, and developing. It is only used for evaluating the RMSE of the final algorithm.

Analysis

Data Exploration

The movielens dataset from grouplens is a massive movie ratings database. As you can see below, there are 69878 users and 10677 movies. There are around 750 million possibilities.

```
# train_set %>% summarize(

# n_users=n_distinct(userId),# unique users from train set

# n_movies=n_distinct(movieId),# unique movies from train set

# n_genres=n_distinct(genres), # unique genres from train set

# min_rating=min(rating), # the lowest rating

# max_rating=max(rating) # the highest rating

# )

# n_users n_movies n_genres min_rating max_rating

# 1 69878 10663 797 0.5 5
```

However, we see only 8 million or so ratings.

```
# > train_set %>% as_tibble()

# A tibble: 8,100,048 x 6

# userId movieId rating timestamp title genres

# <int> <dbl> <dbl> <int> <chr> <chr>

# 1 1 122 5 838985046 Boomerang (1992) Comedy|Romance
```

```
# 2 1 185 5 838983525 Net, The (1995) Action|Crime|Thriller

# 3 1 316 5 838983392 Stargate (1994) Action|Adventure|Sci-Fi

# 4 1 329 5 838983392 Star Trek: Generation~ Action|Adventure|Drama~

# 5 1 355 5 838984474 Flintstones, The (199~ Children|Comedy|Fantasy

# 6 1 356 5 838983653 Forrest Gump (1994) Comedy|Drama|Romance|W~

# 7 1 362 5 838984885 Jungle Book, The (199~ Adventure|Children|Rom~

# 8 1 364 5 838983707 Lion King, The (1994) Adventure|Animation|Ch~

# 9 1 370 5 838984596 Naked Gun 33 1/3: The~ Action|Comedy

# 10 1 377 5 838983834 Speed (1994) Action|Romance|Thriller

# ... with 8,100,038 more rows
```

This indicates that the users-movie rating matrix is very sparse as shown below. Here is a random sample of 200 movies and 200 users with colored cell indicating a user/movie combination for which we have a rating.

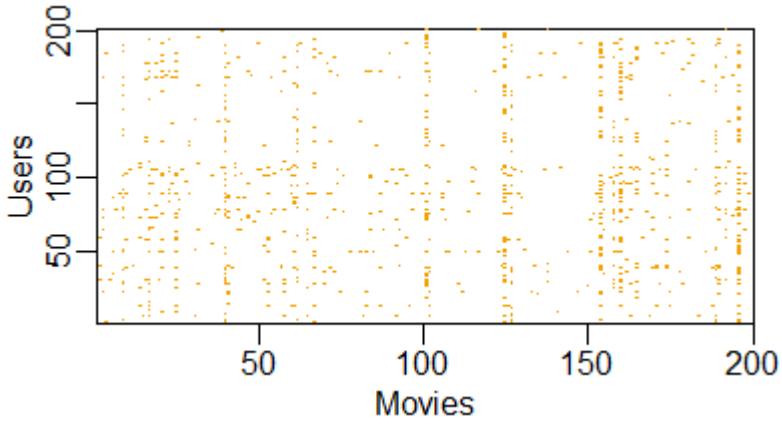


Figure 1: Movie rating distribution

This machine learning challenge is complicated because each outcome Y has a different set of predictors. Users rate movies. Each movie is rated by multiple users. Not every movie is rated by every user. Some movies are rated more frequently. Some movies were rated only once. Some users rate more movies than others. Some users have different temperaments than others. Some users are more active than others at rating movies. Some genres are rated more frequently than others. The lowest rating is 0.5. 4 is the most common rating, followed 3 and 5. Half rating are less common than whole star ratings. All movies were rated. There are no NAs. Here is the movie and users distribution where n is the number of ratings.

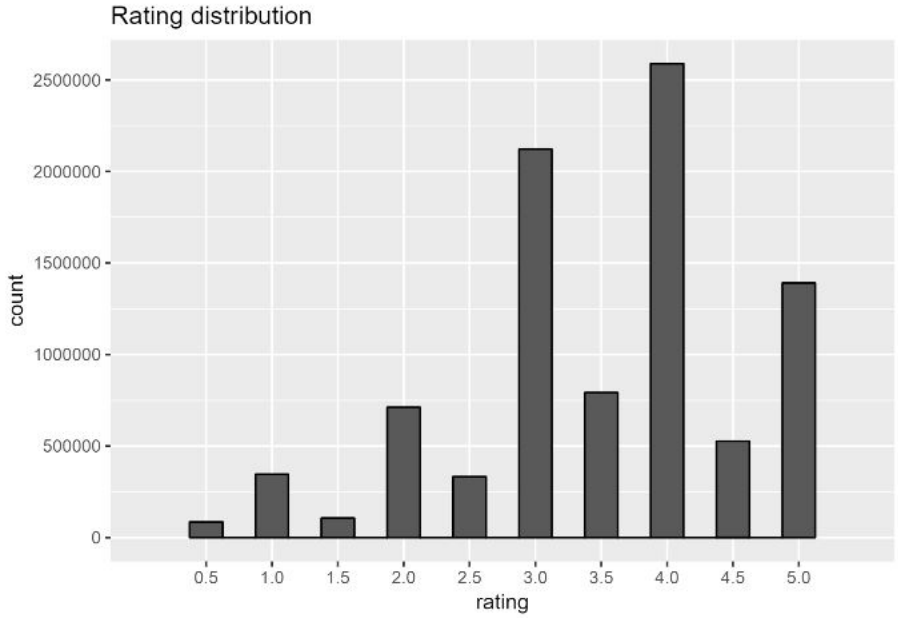


Figure 2: Ratings distribution

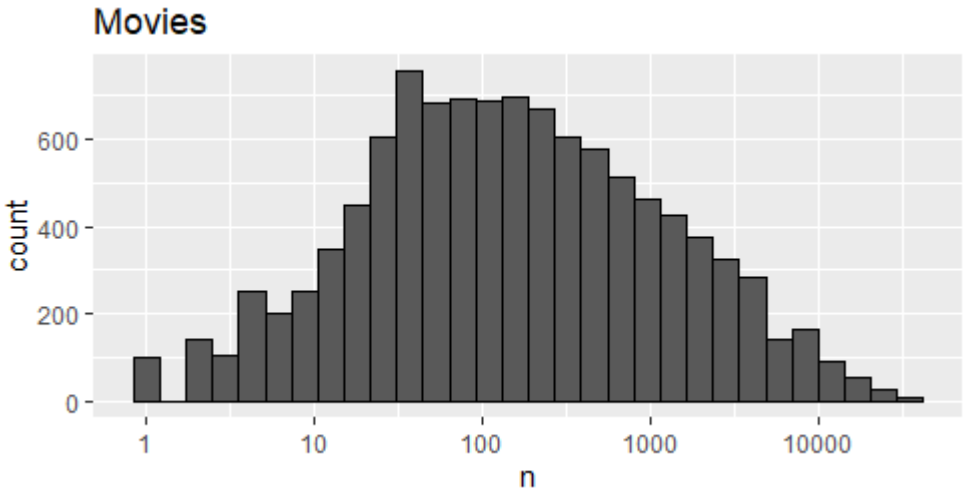


Figure 3: Movies distribution

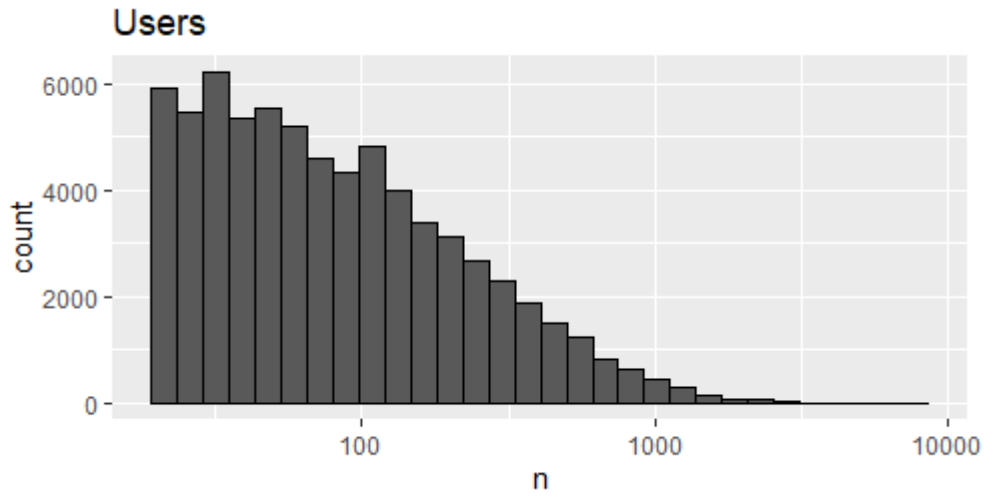


Figure 4: Users distribution

The ratings vary by the movie and by the user. There is great degree variability in ratings in each genre, movie and for each user. Furthermore, each user rate movie differently based on the current temperament. Here are some charts showing the variability by genres, movie and user.

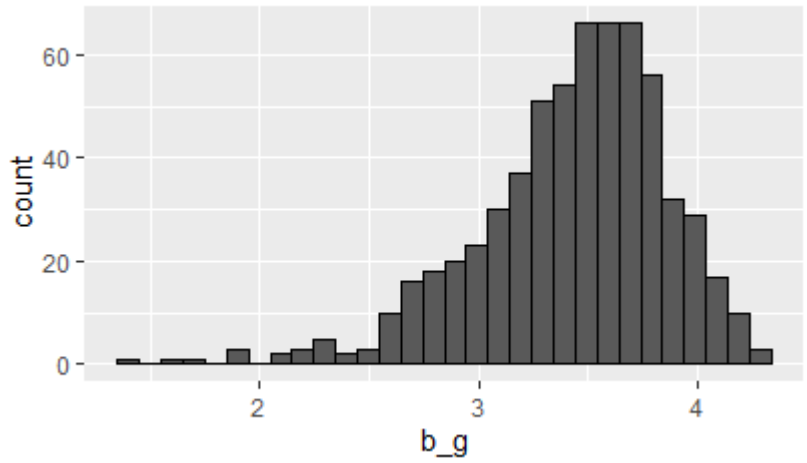


Figure 5: Variability by genres

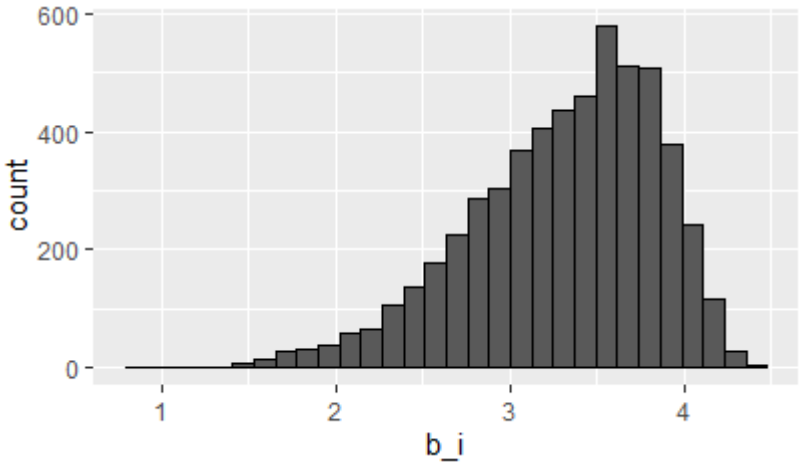


Figure 6: Variability by movie

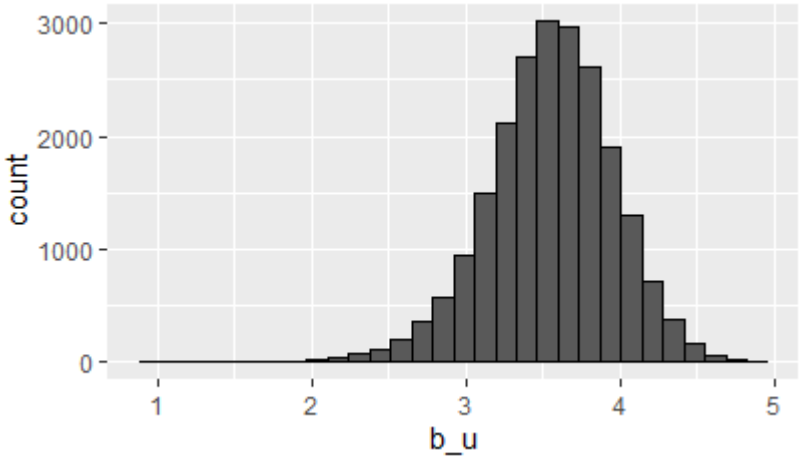


Figure 7: Variability by user

This indicates that we will need to take into account the bias and penalty due to variability and work towards a solution with lower RMSE. Lets see how we can achieve this.

Process

The code (ProjGivenCode.R) that is provided in the project downloads the latest ratings and movie dataset (The data are contained in files, movies.dat, and ratings.dat) from grouplens and keeps only those titles which have at least 1 rating. It then splits the original dataset into development and validation datasets. All analysis is done on this development dataset

In order to compute predictions and RMSE, this development dataset is further split 90/10 into training and test sets respectively. Cleaning of the dataset removes NAs and ensure that the data is in tidy form. To make sure we don't include users and movies in the test set that do not appear in the training set, we remove these entries using the semi_join function. After we create train and test sets, we remove the removed, and temp intermediate data to save space.

Model Creation

The plain model that uses same rating for all movies regardless of the user would be best described with this formula

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

```
# mu = mean(train_set$rating)

# mu

# [1] 3.512421
```

where μ is the observed mean rating. The RMSE for this plain model would be 1.059358

```
# naive_rmse = RMSE(test_set$rating,mu)

# naive_rmse

# [1] 1.059358
```

Our goal is to reduce this RMSE to less than 0.86490. Hence we further analyze the impact of other predictors on the rating.

Using movie as a predictor

We now add the predictor movie to the formula as there is great deal of variability in ratings for each movie. We will investigate how the movie itself impacts the rating. Some movies are better that others. Some are box-office hits while others are not.

We can again use least squares to estimate the bi in the following way,

```
# fit <- lm(rating ~ as.factor(movieId), data = train_set)

# Error: cannot allocate vector of size 643.5 Gb
```

but because of memory requirements for the caret package, the lm() function will not be able to load the data or will be very slow here if not impossible to run.

We know the least square estimate $\hat{b}_i$ is just the average of $y_{u,i} - \hat{\mu}$ for each i

The prediction rating for movie can be defined as:

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

```
# b_i_hat <- train_set %>%

# group_by(movieId) %>%

# summarize(b_i = mean(rating - mu_hat))
```

where μ is the least square estimate and the term b_i represents average ranking for movie i.

We can see this improves our prediction rating to 0.943281

```
# predicted_ratings <- mu_hat + test_set %>%

# left_join(b_i_hat, by='movieId') %>%

# pull(b_i)

# rmse_movie <- RMSE(predicted_ratings, test_set$rating)

# > rmse_movie

# [1] 0.943281
```

We can now improve on this rating by adding additional predictor user.

Using movie and user as predictors

We could again use lm like this:

```
# lm(rating ~ as.factor(movieId) + as.factor(userId))
```

but, for the reasons described earlier, we won't.

The prediction rating for movie and user can be defined as:

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

where the term b_i represents average ranking for movie i and b_u represents the average ranking by users. The bias $\hat{b}_{..}$ is also calculated using the least squared estimates.

The bias $\hat{b}_i$ is calculated using the average of $y_{u,i} - \hat{\mu}$ for each i, where i is the movie.

The bias $\hat{b}_u$ is calculated using the average of $y_{u,i} - \hat{\mu} \ - \hat{b}_i$ for each i, where i is the movie and u is the user.

$\epsilon_{u,i}$ is the independent errors sampled from the same distribution centered at 0 and μ the observed rating for movies.

The RMSE (Residual Mean Square Error) is the error made in prediction and is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

where N is the number of user/movie combinations and the sum occurring over all these combinations and, $y_{u,i}$ is the rating for movie i by user u and the prediction is $\hat{y}_{u,i}$.

The effect of each error on RMSE is proportional to the size of the squared error;thus larger errors have disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers.

We now run the predictions for movie and user.

```
# b_u_hat <- train_set %>%

# left_join(b_i_hat, by='movieId') %>%

# group_by(userId) %>%

# summarize(b_u = mean(rating - mu_hat - b_i))

# Calculate the predicted ratings and RMSE for the movieid and user id together

# predicted_ratings_w_user <- test_set %>%

# left_join(b_i_hat, by='movieId') %>%

# left_join(b_u_hat, by='userId') %>%

# mutate(pred = mu_hat + b_i + b_u) %>%

# pull(pred)

# rmse_movie_user <- RMSE(predicted_ratings_w_user, test_set$rating)

# rmse_movie_user

# [1] 0.8647667
```

We can see that the rating 0.8647667 is already meeting our 0.86490 RMSE goal. However, we do see there is considerable noise when we look at the rating of best and worst movies. In many cases only 1 user rated some movies. Uncertainty in ratings increase with fewer users rating the movie.

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

Figure 8: Obscure ratings

Regularization

Regularization allows us to reduce the noise by penalizing large estimates that are formed using small sample sizes.

The first step in regularization is to choose a penalty term. We do this by calculating user and movie averages with a lambda constant from a sequence of lambda and then calculating RMSEs for each of the chosen lambda. The lowest lambda is then used as the penalty term.

```
# use cross-validation to pick a lambda:

# lambda <- seq(0, 10, 0.25)


# rmses <- sapply(lambda, function(l){

# mu <- mean(train$rating)

# b_i <- train %>%

# group_by(movieId) %>%

# summarize(b_i = sum(rating - mu)/(n()+l))

# b_u <- train %>%

# left_join(b_i, by='movieId') %>%

# group_by(userId) %>%

# summarize(b_u = sum(rating - b_i - mu)/(n()+l))

# predicted_ratings <-

# train %>%

# left_join(b_i, by = 'movieId') %>%

# left_join(b_u, by = 'userId') %>%

# mutate(pred = mu + b_i + b_u) %>%

# .$pred

# return(RMSE(train$rating, predicted_ratings))

# })

# qplot(lambda, rmses)
```

This generates a lambda of 4.75

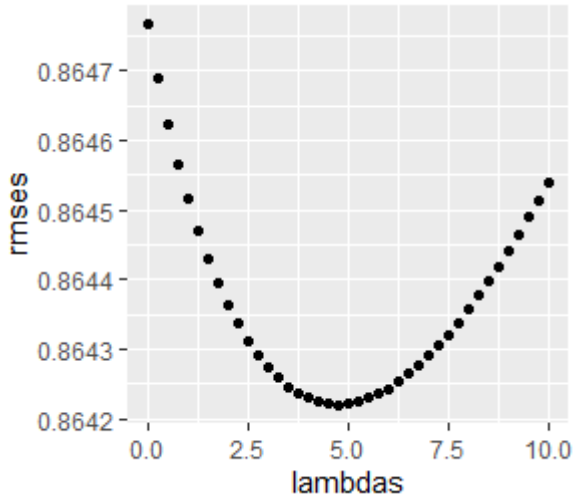


Figure 9: Penalty term

We then use this to compute the regularized ratings and RMSE

```
# compute these regularized estimates for movie and user effect

# mu_hat <- mean(train_set$rating)

# b_i_hat <- train_set %>%

# group_by(movieId) %>%

# summarize(b_i = sum(rating - mu_hat)/(n()+lambda), n_i = n())


# b_u_hat <- train_set %>%

# left_join(b_i_hat, by='movieId') %>%

# group_by(userId) %>%

# summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda), n_i = n())

# Compute the predictions and RMSE using regularized estimates

# predicted_ratings_regularized <- test_set %>%

# left_join(b_i_hat, by = 'movieId') %>%

# left_join(b_u_hat, by = 'userId') %>%

# mutate(pred = mu_hat + b_i + b_u) %>%

# pull(pred)

# rmse_movie_user_regularized <- RMSE(predicted_ratings_regularized,
```

	method	RMSE
a.	Just the movie	0.943035
b.	Movie and User	0.8652
c.	Movie and User regularized	0.864649
d.	Regularized RMSE final test	0.86466

```
# #test_set$rating)
```

This prevents obscure movies and improves the RMSE to 0.86464. The last step is to validate the results with the validation set. This gives us a final rating of 0.86466 that matches with our results. Goal achieved!

Results

To summarize the approach is to work on a smaller training set, compute the least squared estimates ratings for the just the movie and then add the variability of user bias. The resulting RMSE is tuned further by regularizing the model. Regularization improved the RMSE estimates by reducing noisy estimates that cannot be trusted. Regularization permitted us to penalize large estimates that are formed using small sample sizes

Above is the tabulated summary of all the RMSEs for the different methods.

Conclusion

The MovieLens dataset is very long, so we train models using average approximation of least square and processed within the available RAM in personal machine. To improve precision, we would need to use knn, glm, lm or such training algorithms from caret package.

The ratings in the MovieLens dataset have a varying degree of variability by genre, movie and the user. There is some degree of noise in the dataset.

However, I was able to use least square estimates algorithm to predict the ratings based on movie and user variability within the requested RMSE parameters. The RMSE improved further with regularization. The **final RMSE 0.86466** is below our target RMSE 0.86499

There are additional predictors like movie age that can been explored to further improve the accuracy of the model. Additionally, methods such as matrix factorization using the recosystem package could be utilized to provide significant improvements in the RMSE. More advanced models featuring principal component analysis (PCA) and Singular Value Decomposition (SVD) generate higher levels of accuracy, though they are also far more computationally intensive

Appendix A - Complete code

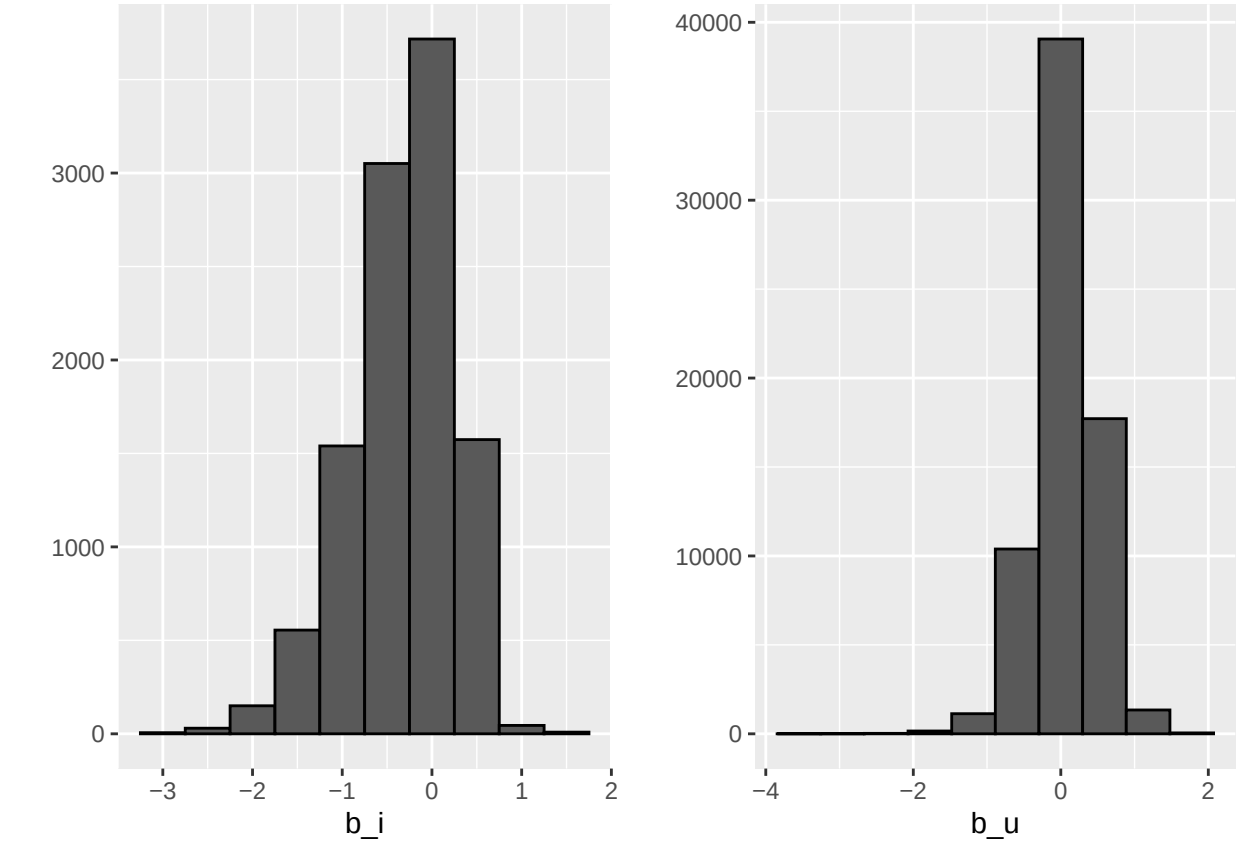
```
## @knitr MovieLensR

# Execute the given source code for the project
source("ProjGivenCode.R")

library(caret)
library(gridExtra)
```

```
library(kableExtra)


set.seed(1996, sample.kind = "Rounding")

# Split the development dataset 90% - training set and 10% test set
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index, ]

# create a test set to assess the accuracy of the models implemented during
# development.
temp <- edx[test_index, ]

# Exclude users and movies in the test set that do not appear in the training
# set using the semi_join
test_set <- temp %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set)

train_set <- rbind(train_set, removed)  #add back removed items

# remove temporary data to save space
rm(removed, temp)

# create a function that computes the RMSE for vectors of ratings and their
# corresponding predictors:
RMSE <- function(true_ratings, predicted_ratings) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}

# A model with just rating and movie explained by random variation would look
# like this:

# mean rating
mu_hat <- mean(train_set$rating)

naive_rmse <- RMSE(test_set$rating, mu_hat)

# RMSE for plain model with same rating for all movies
naive_rmse
```

```
## [1] 1.059691
```

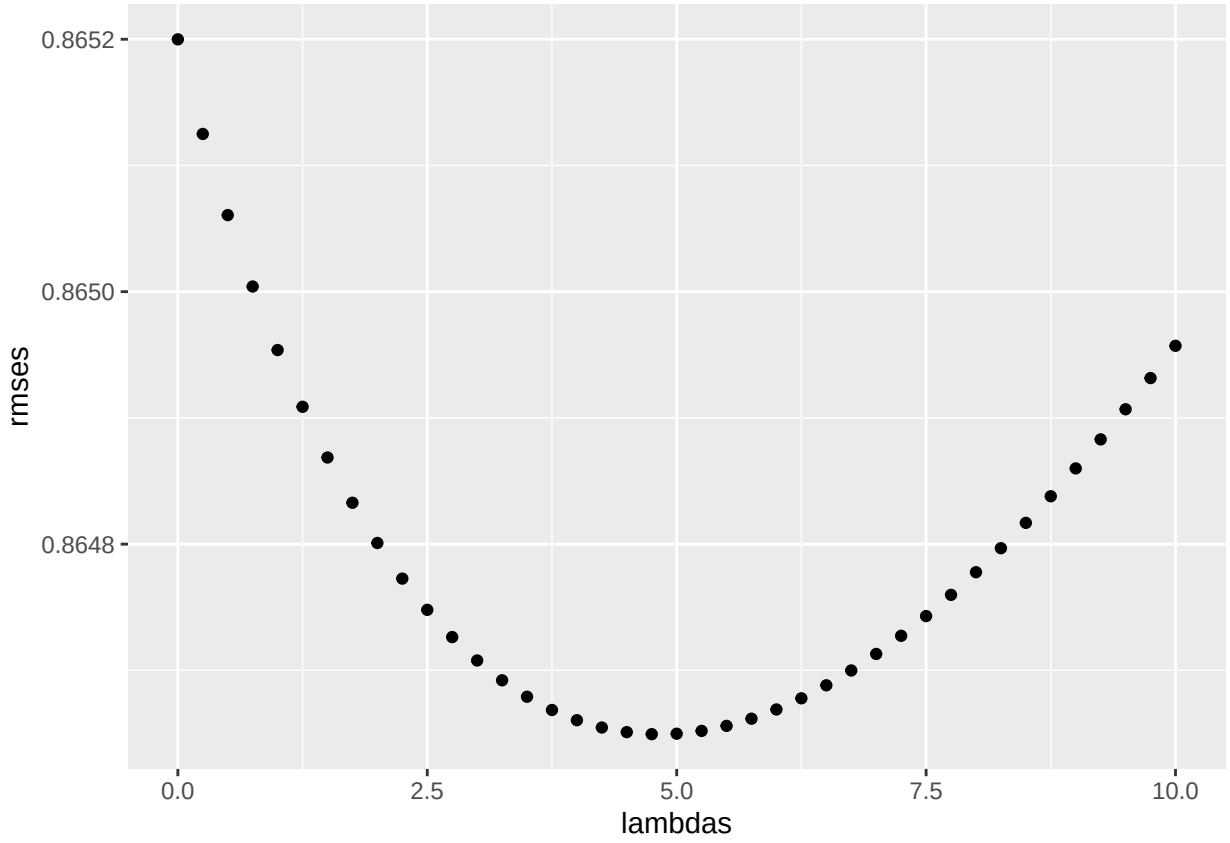
```
# the least squares estimate b_i_hat is just the average of mean of rating -
# avg. rating
b_i_hat <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))

# quick plot to review least square estimates for movie bias
movie_plot <- b_i_hat %>%
```

```
    qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"))

# Calculate the predicted ratings and RMSE for just the movieid
predicted_ratings <- mu_hat + test_set %>%
    left_join(b_i_hat, by = "movieId") %>%
    pull(b_i)

rmse_movie <- RMSE(predicted_ratings, test_set$rating)

# now include user id in the calculation of b_u_hat
b_u_hat <- train_set %>%
    left_join(b_i_hat, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))

# quick plot to review least square estimates for user bias
user_plot <- b_u_hat %>%
    qplot(b_u, geom = "histogram", bins = 10, data = ., color = I("black"))

# review the two bias effect in a side-by-side plot
grid.arrange(movie_plot, user_plot, ncol = 2)
```




```
# Calculate the predicted ratings and RMSE for the movieid and user id together
predicted_ratings_w_user <- test_set %>%
    left_join(b_i_hat, by = "movieId") %>%
```

```
    left_join(b_u_hat, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)

rmse_movie_user <- RMSE(predicted_ratings_w_user, test_set$rating)

# Regularization Choosing the penalty term (lambda)
lambdas <- seq(0, 10, 0.25)

# iterate through the sequence of lambdas and compute the ratings and RMSEs
rmses <- sapply(lambdas, function(l) {

    mu_hat <- mean(train_set$rating)

    b_i <- train_set %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu_hat)/(n() + l))

    b_u <- train_set %>%
        left_join(b_i, by = "movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu_hat)/(n() + l))

    predicted_ratings <- test_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(pred = mu_hat + b_i + b_u) %>%
        pull(pred)

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

# verify the lambda penalty term
qplot(lambdas, rmses)
```




```
# find the lambda with the lowest RMSE
lambda <- lambdas[which.min(rmses)]

# optimal lambda
lambda
```

```
## [1] 4.75
```

```
# compute these regularized estimates for movie and user effect
mu_hat <- mean(train_set$rating)
b_i_hat <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n() + lambda), n_i = n())

b_u_hat <- train_set %>%
    left_join(b_i_hat, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_hat - b_i)/(n() + lambda), n_i = n())

# Compute the predictions and RMSE using regularized estimates
predicted_ratings_regularized <- test_set %>%
    left_join(b_i_hat, by = "movieId") %>%
    left_join(b_u_hat, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
```

	method	RMSE
a.	Just the movie	0.943035
b.	Movie and User	0.8652
c.	Movie and User regularized	0.864649
d.	Regularized RMSE final test	0.86466

```
rmse_movie_user_regularized <- RMSE(predicted_ratings_regularized, test_set$rating)

# Get the validation set
validation_test_index <- createDataPartition(y = validation$rating, times = 1, p = 0.1,
    list = FALSE)
validation_test_set <- validation[validation_test_index, ]

# test the regularized estimates on the validation set
validation_ratings_regularized <- validation_test_set %>%
    left_join(b_i_hat, by = "movieId") %>%
    left_join(b_u_hat, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)

# compute the RMSE for the validation set
rmse_validation_test <- RMSE(validation_ratings_regularized, validation_test_set$rating)

# tabulate all the RMSE results
rmse_results <- matrix(c("Just the movie", round(rmse_movie, 6), "Movie and User",
    round(rmse_movie_user, 6), "Movie and User regularized", round(rmse_movie_user_regularized,
        6), "Regularized RMSE final test", round(rmse_validation_test, 6)), nrow = 4,
    ncol = 2, byrow = TRUE, dimnames = list(c("a.", "b.", "c.", "d."), c("method",
        "RMSE")))




rmse_results %>%
    knitr::kable() %>%
    kable_styling(bootstrap_options = c("striped", "hover", "condensed"))
```

Appendix B - Links

https://www.edx.org/professional-certificate/harvardx-data-science–

https://www.crcpress.com/Introduction-to-Data-Science-Data-Analysis-and-Prediction-Algorithms-with/Irizarry/p/book/9780367357986–

https://leanpub.com/datasciencebook–

Citations

Irizarry, Rafael A., "Introduction to Data Science: Data Analysis and Prediction Algorithms in R" https://rafalab.github.io/dsbook/

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