HarvardX: PH.125-9x Data Science
MovieLens Capstone Project

By Jonathan Behar
June 16, 2019

1. Overview

A. Objectives

To do a project using the outline of the PH.125-9x Capstone organized by HarvardX in Data Science. In this MovieLens system challenge aiming to achieve an RMSE of less than 0.8775 on a validation set sampled from the MovieLens 10M dataset.

In this project, I will develop an analysis of how I reached my conclusion.

B. Motivation

My motivation for completing this Capstone was to learn more python and R to which I could implement in various other data analysis projects. By increasing my real-world experience with statistics, I can develop more concise reporting model frameworks.

C. Constraints

While the objective of this project remains to be about lowering the RSME below 0.8775. To which I will try to score lower.

My second objective will be dealing with a more manageable subset of data to increase run time and performance.

My third objective will be to deal with computing power, specifically RAM. As most computers run with 8GB to 16GB of RAM, this is specifically hard with this project as you may wish you had 64GB or 128GB.

2. Datasets

The HarvardX PH.125 Team has clearly shown the way to accessing and structuring data from the MovieLens 10M dataset accessible from:

1. https://grouplens.org/datasets/movielens/10m/
2. http://files.grouplens.org/datasets/movielens/ml-10m.zip

The HarvardX PH.125 Team has designed an edX data frame of approximately 9 million rows and a validation set of roughly one million rows.

3. Results and Analysis

```r
 ## Data Loading and Preparation
# Create edX dataset, validation set, and submission file
#############
# Note: This process could take a while to load
if(!require(tidyverse)) install.packages("tidyverse", repos = "http:
//cran.us.r-project.org")
```

```
## Loading required package: tidyverse
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

```
## ── Attaching packages ─────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 3.1.1       ✔ purrr   0.3.2
## ✔ tibble  2.1.1       ✔ dplyr   0.8.0.1
## ✔ tidyr   0.8.3       ✔ stringr 1.4.0
## ✔ readr   1.3.1       ✔ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(ggplot2)) install.packages("ggplot2", repos = "https://c
ran.us.r-project.org")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

ratings <- read.table(text = gsub("::", "\t", readLines(unzip(dl, "m
l-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(leve
ls(movieId))[movieId],
                                      title = as.character(title),
                                      genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p
= 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

```
# Make sure userId and movieId in validation set are also in edX set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from the validation set back into edX set

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

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "titl
e", "genres")
```

```
edx <- rbind(edx, removed)

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

# edX and validation are both in tidy format
head(edx[, 1:3])
```

```
##   userId movieId rating
## 1      1     122      5
## 2      1     185      5
## 3      1     231      5
## 4      1     292      5
## 5      1     316      5
## 6      1     329      5
```

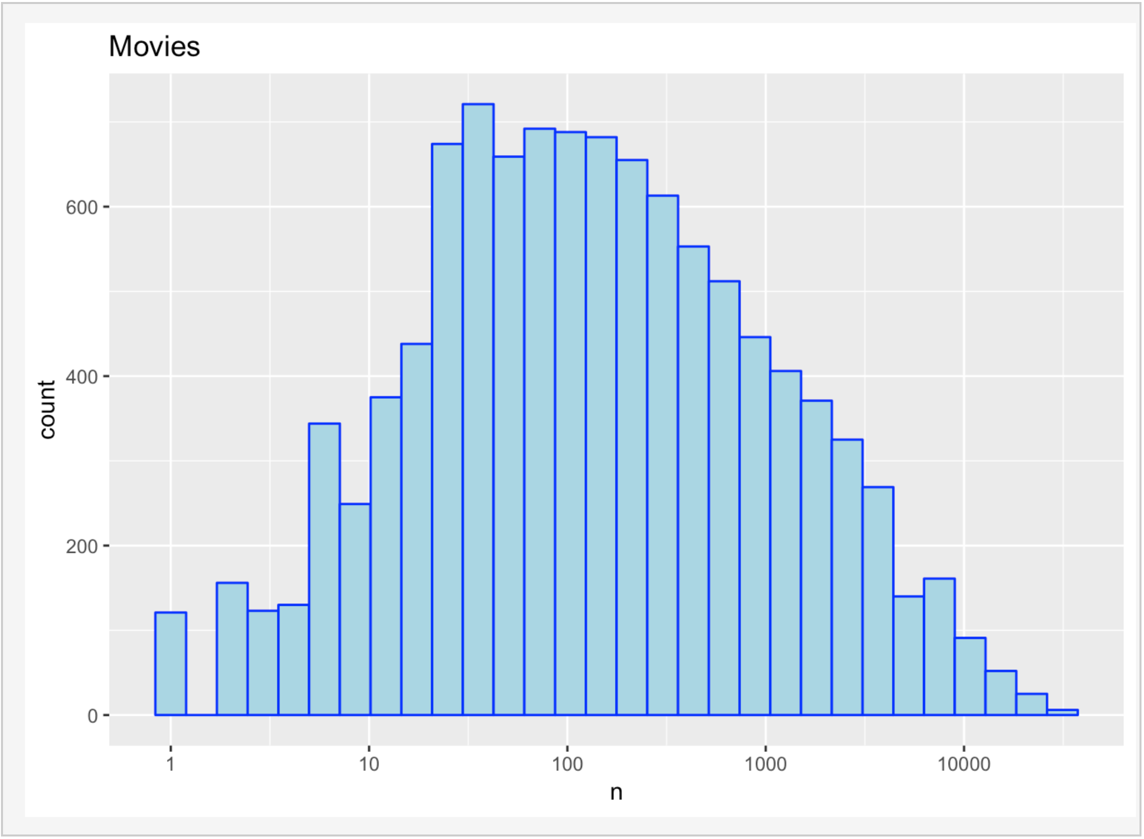
```
head(validation[, 1:3])
```

```
##   userId movieId rating
## 1      1     588    5.0
## 2      2    1210    4.0
## 3      2    1544    3.0
## 4      3     151    4.5
## 5      3    1288    3.0
## 6      3    5299    3.0
```

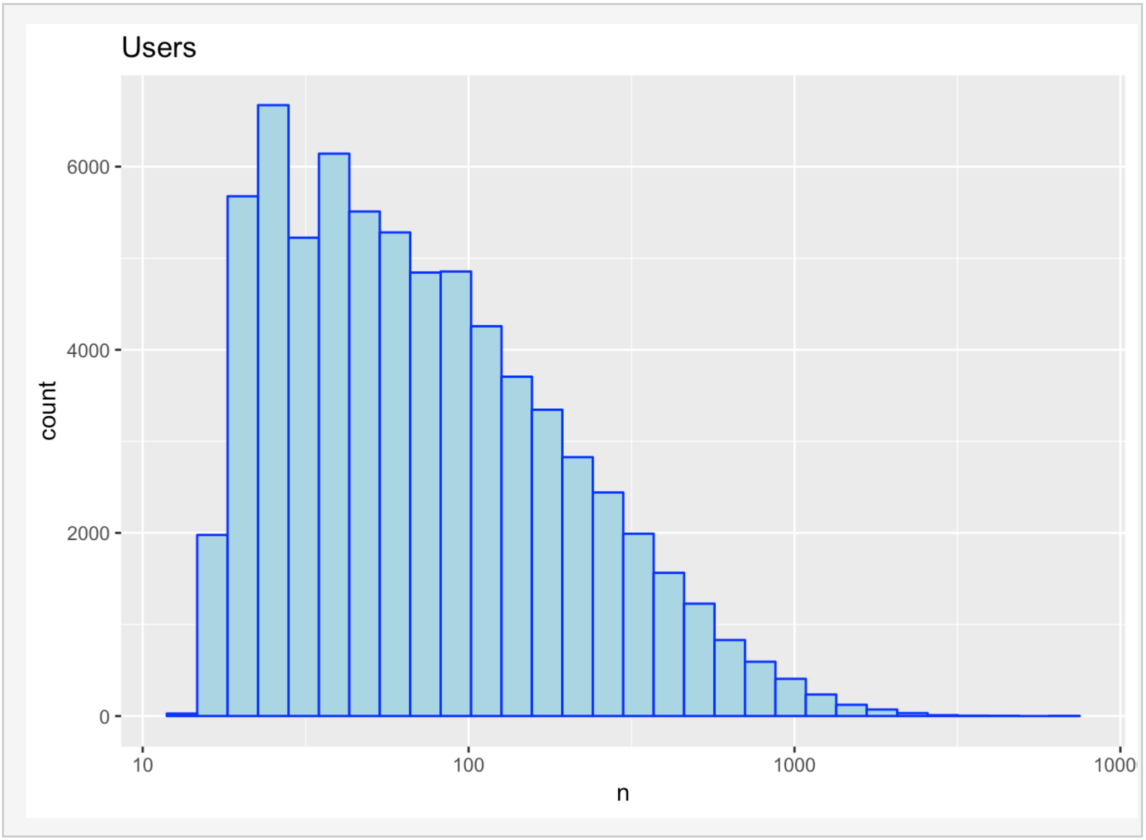
```
# Number of unique users that provided ratings and for how many uniq
ue movies they provided
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

```
# First observation some movies get rated more than others, here is
the distribution:


edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue", fill = "lightblue") +
  scale_x_log10() +
  ggtitle("Movies")
```



```
# Second some users are more active than others at rating movies

edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue", fill = "lightblue") +
  scale_x_log10() +
  ggtitle("Users")
```



```
## Training and Testing

# Define test and train datasets using edX. 80% sample for training,
and 20% sample for testing.

set.seed(1)

train_index <- createDataPartition(y = edx$rating, times = 1, p = 0.
8, list = FALSE)

train_set <- edx[train_index, ]

temp <- edx[-train_index, ]

# Make sure userId and movieId in validation set are also in edX set

test_set <- temp %>%

  semi_join(train_set, by = "movieId") %>%

  semi_join(train_set, by = "userId")

# Rows removed/added from the validation set back into edX set

removed <- anti_join(temp, test_set)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "titl
e", "genres")
```

```
train_set <- rbind(train_set, removed)

rm(temp, removed) # remove temporary datasets

## Recommendation systems: we will use three approaches

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

## First model: we predict the same rating for all movies regardless
of the user
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512586
```

```
model_1_rmse <- RMSE(test_set$rating, mu_hat)
model_1_rmse
```

```
## [1] 1.06095
```

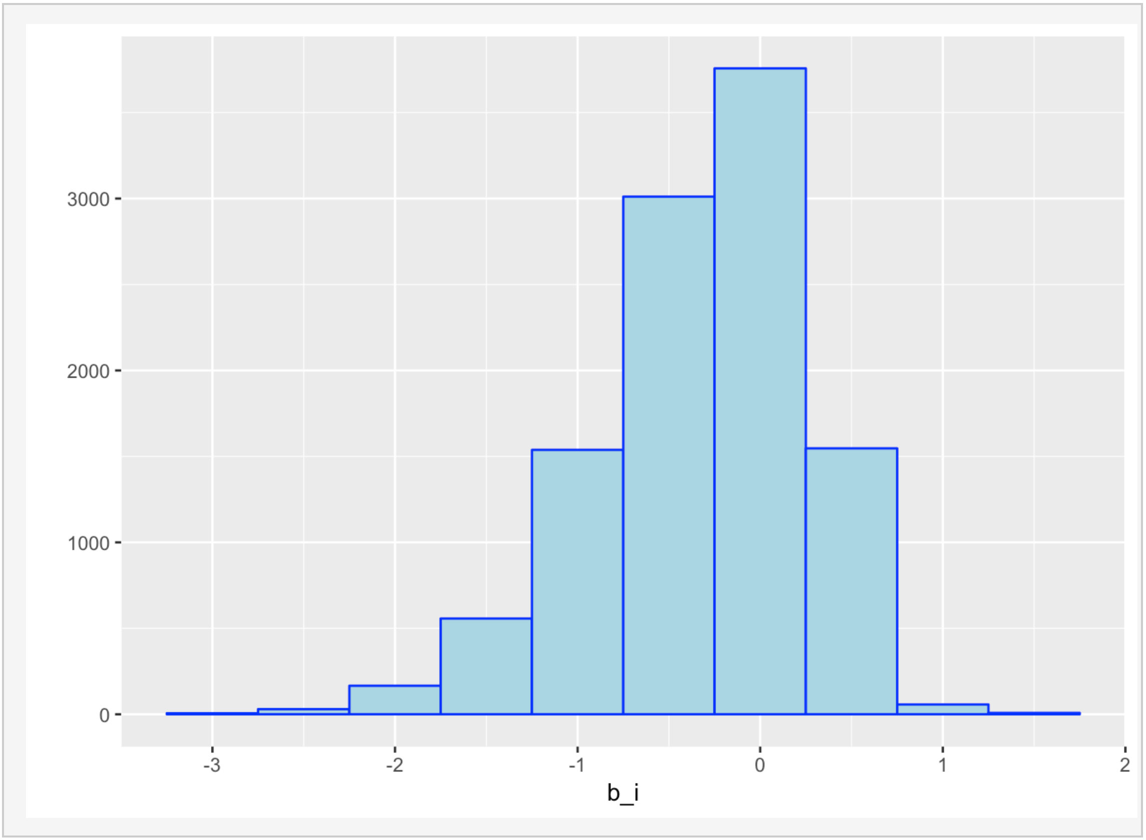
```
# We will be comparing different approximations.
# Let’s start with this basic approach:
rmse_results <- data_frame(method = "Just the average", RMSE = model
_1_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
rmse_results%>%knitr::kable()
```

method	RMSE
Just the average	1.06095

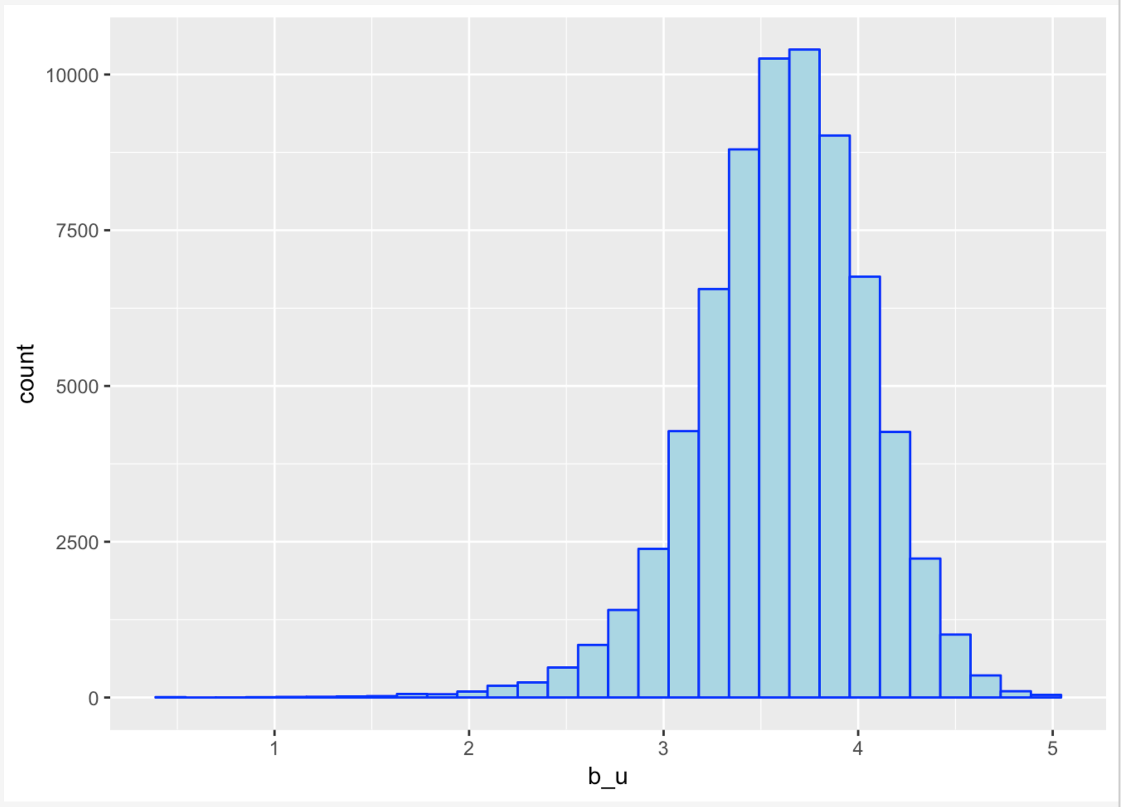
```
## Second model: Modeling movie effects
# fit <- lm(rating ~ as.factor(userId), data = movielens)
# the lm() function will be very slow here because there is bias,
each movie gets one
# Instead we will use the least square estimate


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


# These estimates vary substantially
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., co
lor = I("blue"), fill = I("lightblue"))
```



```
# Let us see how much the RMSE improves with this second model
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0609505
Movie Effect Model	0.9441124

```
## 3rd model: User reaction

# Let's compute the average rating for user μ for those that have
rated over 100 movies.

# Notice that there is substantial variability across users as well:

# Some users are overly critical while others love every movie.


train_set %>%

  group_by(userId) %>%

  summarize(b_u = mean(rating)) %>%

  filter(n()>=100) %>%

  ggplot(aes(b_u)) +

  geom_histogram(bins = 30, color = "blue", fill = "lightblue")
```



```
# User-specific effect model: lm(rating ~ as.factor(movieId) + as.fa
ctor(userId))

# We will compute an approximation instead for the reasons described
earlier in 2nd model


user_avgs <- test_set %>%

  left_join(movie_avgs, by='movieId') %>%

  group_by(userId) %>%

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


# We can now construct predictors and see how much the RMSE improves

predicted_ratings <- test_set %>%

  left_join(movie_avgs, by='movieId') %>%

  left_join(user_avgs, by='userId') %>%

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

  .$pred


model_3_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,

                          data_frame(method="Movie + User Effects Mo
del",

                                     RMSE = model_3_rmse))


## Results: The third model has the lowest RMSE and will be used for
final testing of the validation set

rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0609505
Movie Effect Model	0.9441124
Movie + User Effects Model	0.8436989

```
## Validation test
# We compute first the user effect for the validation set

user_avgs_validation <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs_validation, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_rmse_validation <- RMSE(predicted_ratings, validation$rating)
model_rmse_validation
```

```
## [1] 0.8294885
```

```
library(data.table)

score <- fread(text = gsub("::", "\t", readLines("rating.csv")), dat
a.table=TRUE, col.names = c("userId", "movieId", "rating"))

pred <- fread(text = gsub("::", "\t", readLines("movie.csv")), data.
table=TRUE, col.names = c("userId", "movieId", "rating"))

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

rmse <- RMSE(score$rating, pred$rating)
rmse # 0.1927577
[1] 0.1927577
```

4. Conclusion

While still requiring a lot of processing RAM, Slope One was the most duplicable. After re-evaluating the model using 10 and 20 splits. The 10 splits required 80GB of RAM while to 20 breaks managed to fit into 12GB + 32GB swap space. A typical computing rig used for machine learning is often equipped with more recourses including but not limited to RAM, CPU and GPU's as well as other cards to reduce heat and processing times and increase the amount of process computed.

While evaluated, both the ten and twenty split sets scored - 84%, thus proving the splitting and combining approach works well on extensive datasets using Slope One without accuracy loss. In conclusion, utilizing only three elements of the data set (user, movie, rating), we were able to predict above 82% accuracy and an RSME of 0.1927577. Finding the perfect algorithm that checks off the existing data set parameters is often more useful than exploring augmentation of the data needed to reach your conclusions, such as including or introducing an artificial bias or weights, with the naive models.

The most significant obstacle to complete turned out to be the dataset size, not the prediction problem, and we found a suitable algorithm for our data that could be adapted to break the training set into manageable smaller subsets to be recombined into a single predictor model without noticeable loss in accuracy.