MovieLens

Dagvadorj Galbadrakh

7/13/2021

1 Executive summary

MovieLens database is examined and models were reviewed and fit for the Hardvard edX Data Science: Capstone project.

The initial section of the code is based on the boilerplate code provided at the "Create Train and Final Hold-out Test Sets" section of the course [1]. Following columns were added to the movielens dataframe in order to be used as potential predictors: releaseYear, age, year, month, week, weekday, hour, avgRating, numRating, and numRatingStrata. releaseYear was extracted from the movie title; age was calculated by subtracting the rating date from the release year; year, month, week, weekday, and hour are extracted from the date of rating using the lubridate package. Average rating and the number of ratings were calculated based on overall ratings of movies. Furthermore, the number of ratings were stratified by thousands factor of the number of ratings and stored in numRatingStrata since the actual number will be too specific to be used as a predictor.

Different models were used by utilizing train method from the caret library. However, every try took unfeasible amount of time in my computer with 16 gig memory. Therefore linear model provided at the "Regularization" section of the Data Science: Machine Learning course [2] that examines accumulative biases of predictors were used.

Exploratory data analyses were performed in order to understand how the predictors correlate with the ratings.

Upon fitting models and assessing the RMSEs of the predicted values on a subset of the edx dataset, it was understood that the age of the ratings and the existing average rating of the movie weighted by the number of rating stratifications are relatively well performing as predictors.

Regularization method of assigning a tuning parameter as in [2] yielded 0.86460 and 0.86434 RMSEs for the models accounting biases of movies and users plus the age of rating and the number of rating weighted average rating respectively.

2 Methods

2.1 Data preparation

```r
# tinytex::install_tinytex()
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")
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
if(!require(ggridges)) install.packages("ggridges", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(GGally)
library(ggridges)
library(ggthemes)

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

# Don't need to run this everytime when it is downloaded already
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)

# In order to save time, this needs to be run once the dataset zip is downloaded already.
# ratings <- fread(text = gsub("::", "\t", readLines("ml-10M100K/ratings.dat")),
#                  col.names = c("userId", "movieId", "rating", "timestamp"))
# movies <- str_split_fixed(readLines("ml-10M100K/movies.dat"), "\\::", 3)
```

```r
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                             title = as.character(title),
#                                             genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

Extracting release date from titles

```r
movielens <- movielens %>% mutate(releaseYear = str_extract(movielens$title, "\\((\\d{4})\\)$")) %>%
  mutate(releaseYear = as.integer(substring(releaseYear, 2, nchar(releaseYear)-1)))
```

Extracting components of the date of the ratings

```r
movielens <- movielens %>% mutate(datetime = lubridate::as_datetime(timestamp))
movielens <- movielens %>% mutate(year = year(datetime), month = month(datetime),
                                  week = week(datetime), weekday = wday(datetime),
                                  hour = hour(datetime))
```

The difference between the movie's release year and the year of rating is selected as a potential predictor

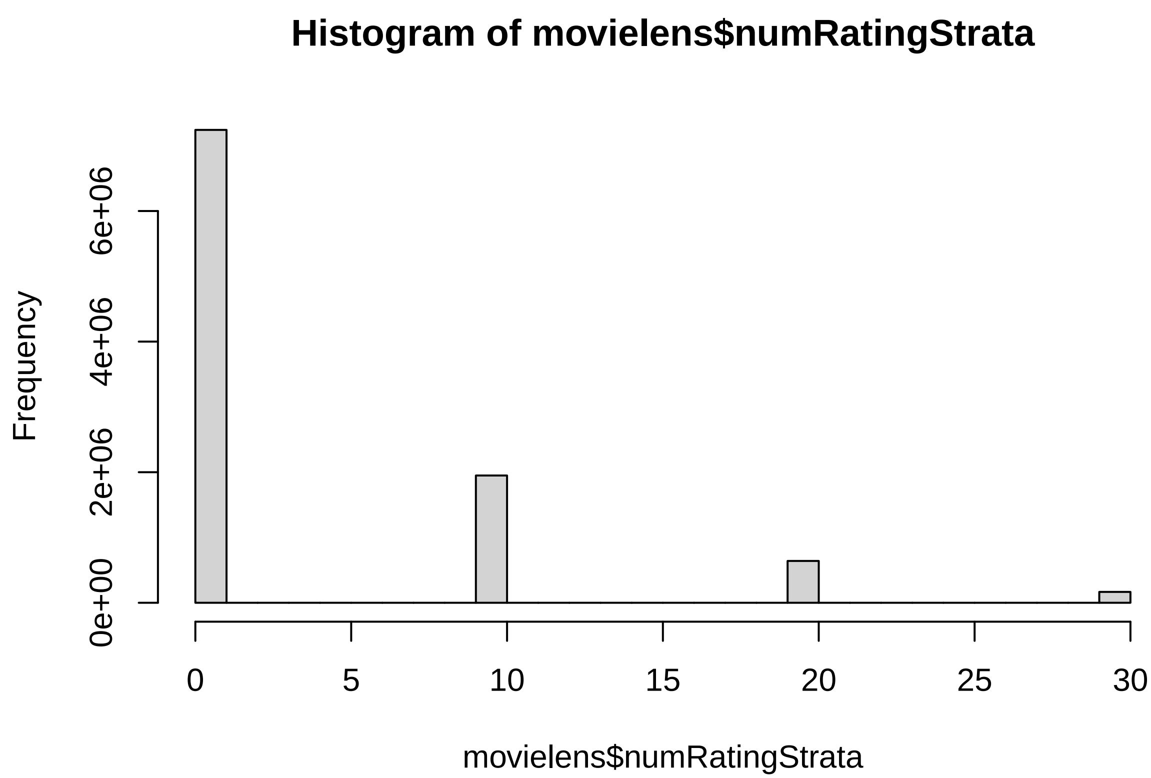
```r
movielens <- movielens %>% mutate(age = year - releaseYear)
```

Existing average rating will be used as a predictor

```r
movielens <- movielens %>% group_by(movieId) %>% mutate(avgRating = as.integer(sum(rating) / n()),
                                                        numRating = n()) %>% ungroup()
```

Number of existing ratings may be a good predictor when used with existing average ratings of movies. I am stratifying the number of existing ratings since the numbers are too unique per movie and this may result in overfitting.

```r
movielens <- movielens %>% mutate(numRatingStrata = floor(numRating/10000)*10)

hist(movielens$numRatingStrata)
```



We will not be using datetime, genres, and timestamp as predictors

```r
movielens <- movielens %>% select(-datetime, -genres, -timestamp)
```

2.2 Training and testing data set preparation

```r
summary(movielens)
```

```
##      userId         movieId          rating          title          
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:10000054   
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   Class :character  
##  Median :35741   Median : 1834   Median :4.000   Mode  :character  
##  Mean   :35870   Mean   : 4120   Mean   :3.512                     
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000                     
##  Max.   :71567   Max.   :65133   Max.   :5.000                     
##   releaseYear        year          month             week          weekday     
##  Min.   :1915   Min.   :1995   Min.   : 1.000   Min.   : 1.00   Min.   :1.000  
##  1st Qu.:1987   1st Qu.:2000   1st Qu.: 4.000   1st Qu.:14.00   1st Qu.:2.000  
##  Median :1994   Median :2002   Median : 7.000   Median :28.00   Median :4.000  
##  Mean   :1990   Mean   :2002   Mean   : 6.786   Mean   :27.74   Mean   :3.906  
##  3rd Qu.:1998   3rd Qu.:2005   3rd Qu.:10.000   3rd Qu.:42.00   3rd Qu.:6.000  
##  Max.   :2008   Max.   :2009   Max.   :12.000   Max.   :53.00   Max.   :7.000  
##       hour            age           avgRating       numRating    
##  Min.   : 0.00   Min.   :-2.00   Min.   :0.000   Min.   :    1  
##  1st Qu.: 6.00   1st Qu.: 2.00   1st Qu.:3.000   1st Qu.: 1814  
##  Median :14.00   Median : 7.00   Median :3.000   Median : 4699  
##  Mean   :12.48   Mean   :11.98   Mean   :3.001   Mean   : 7542  
##  3rd Qu.:19.00   3rd Qu.:16.00   3rd Qu.:3.000   3rd Qu.:10928  
##  Max.   :23.00   Max.   :93.00   Max.   :5.000   Max.   :34864  
##  numRatingStrata 
##  Min.   : 0.000  
##  1st Qu.: 0.000  
##  Median : 0.000  
##  Mean   : 3.731  
##  3rd Qu.:10.000  
##  Max.   :30.000  
```

Validation set will be 10% of MovieLens data

```r
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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 validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

# Freeing up memory and workspace
rm(dl, ratings, movies, test_index, temp, movielens, removed)

# Divide into train and test sets

set.seed(1, sample.kind = "Rounding")

test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)

train_set <- edx[-test_index,]
test_set <- edx[test_index,]

test_set <- test_set %>%semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

I will use RMSE function to examine the accuracy of our models since we are dealing with continuous outcomes.

```r
RMSE <- function(predicted, actual) {
  sqrt(mean((predicted - actual)^2, na.rm = TRUE))
}
```

2.3 Selection of methodology

Using caret training models are not feasible for this exercise as simple linear regression model for two predictors in train_set data set are taking approximately 19 minutes. Using other models like recursive partitioning and random forest models give insufficient memory errors or run hours when memory usage limit is increased by memory.limit(9999999999).

```
# model.lm <- train_set %>% train(rating ~ movieId + userId, data = ., method = "glm")

# Training linear regression on a fraction of the train_set also
# yielded unfeasibly long runtimes for this practice.

# train_set_1 <- sample_n(train_set, 100000)
# test_set_1 <- sample_n(test_set, 100000)

# test_set_1 <- test_set_1 %>%semi_join(train_set_1, by = "movieId") %>%
#     semi_join(train_set_1, by = "userId")
# model.lm <- train_set_1 %>% train(rating ~ movieId + userId, data = ., method = "glm")
```

Hence, I will be using the simpler model that accounts for the movie and user biases by grouping the data by movieId and userId and analyzes their effects on the ratings. This model is mentioned in Irizarry, 2019 [2].

2.4 Exploratory analysis

I liked reading Economist or Wall Street Journal on airplanes just to feel fancy.

```
theme_set(theme_economist())
```

Summary of our dataset using the "summary" method will give us idea about statistical information about each predictor.

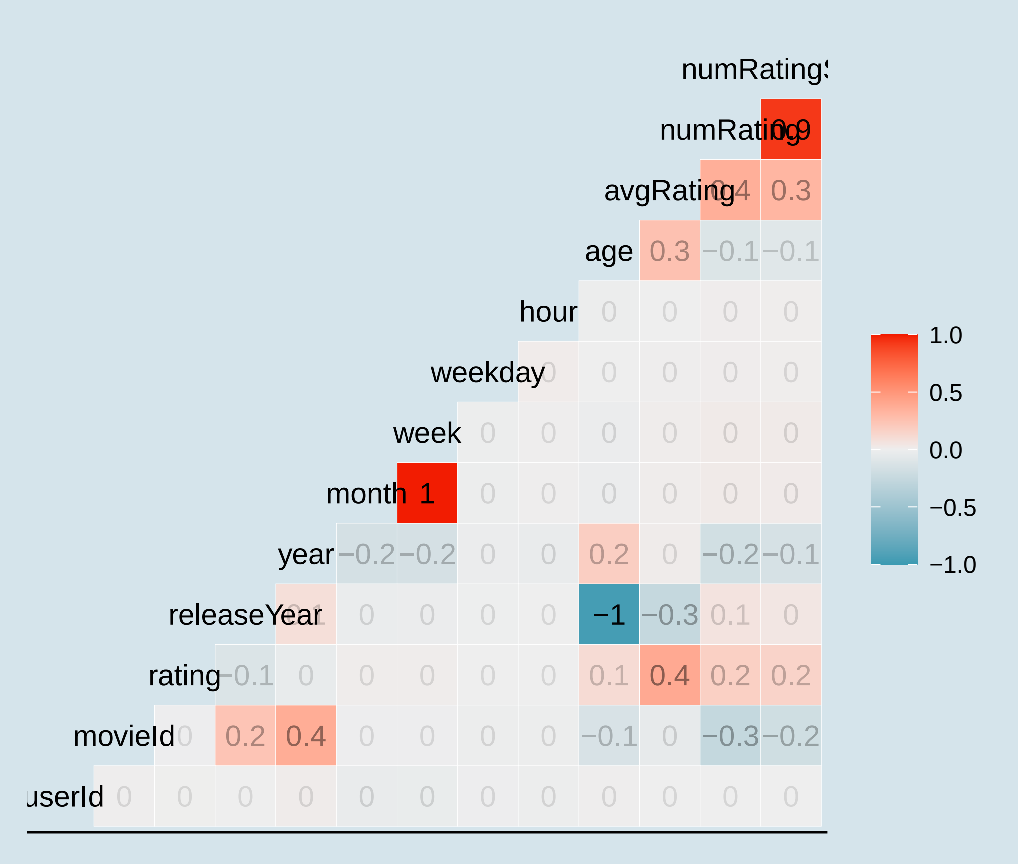
```
summary(edx)
```

```
##      userId         movieId          rating         title
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:9000055
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   Class :character
##  Median :35738   Median : 1834   Median :4.000   Mode  :character
##  Mean   :35870   Mean   : 4122   Mean   :3.512
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000
```

```
## Max.   :71567   Max.   :65133   Max.   :5.000
##  releaseYear        year          month             week          weekday
## Min.   :1915   Min.   :1995   Min.   : 1.000   Min.   : 1.00   Min.   :1.000
## 1st Qu.:1987   1st Qu.:2000   1st Qu.: 4.000   1st Qu.:14.00   1st Qu.:2.000
## Median :1994   Median :2002   Median : 7.000   Median :28.00   Median :4.000
## Mean   :1990   Mean   :2002   Mean   : 6.786   Mean   :27.74   Mean   :3.906
## 3rd Qu.:1998   3rd Qu.:2005   3rd Qu.:10.000   3rd Qu.:42.00   3rd Qu.:6.000
## Max.   :2008   Max.   :2009   Max.   :12.000   Max.   :53.00   Max.   :7.000
##      hour            age          avgRating       numRating
## Min.   : 0.00   Min.   :-2.00   Min.   :0.000   Min.   :    1
## 1st Qu.: 6.00   1st Qu.: 2.00   1st Qu.:3.000   1st Qu.: 1814
## Median :14.00   Median : 7.00   Median :3.000   Median : 4699
## Mean   :12.48   Mean   :11.98   Mean   :3.001   Mean   : 7541
## 3rd Qu.:19.00   3rd Qu.:16.00   3rd Qu.:3.000   3rd Qu.:10928
## Max.   :23.00   Max.   :93.00   Max.   :5.000   Max.   :34864
## numRatingStrata
## Min.   : 0.000
## 1st Qu.: 0.000
## Median : 0.000
## Mean   : 3.731
## 3rd Qu.:10.000
## Max.   :30.000
```

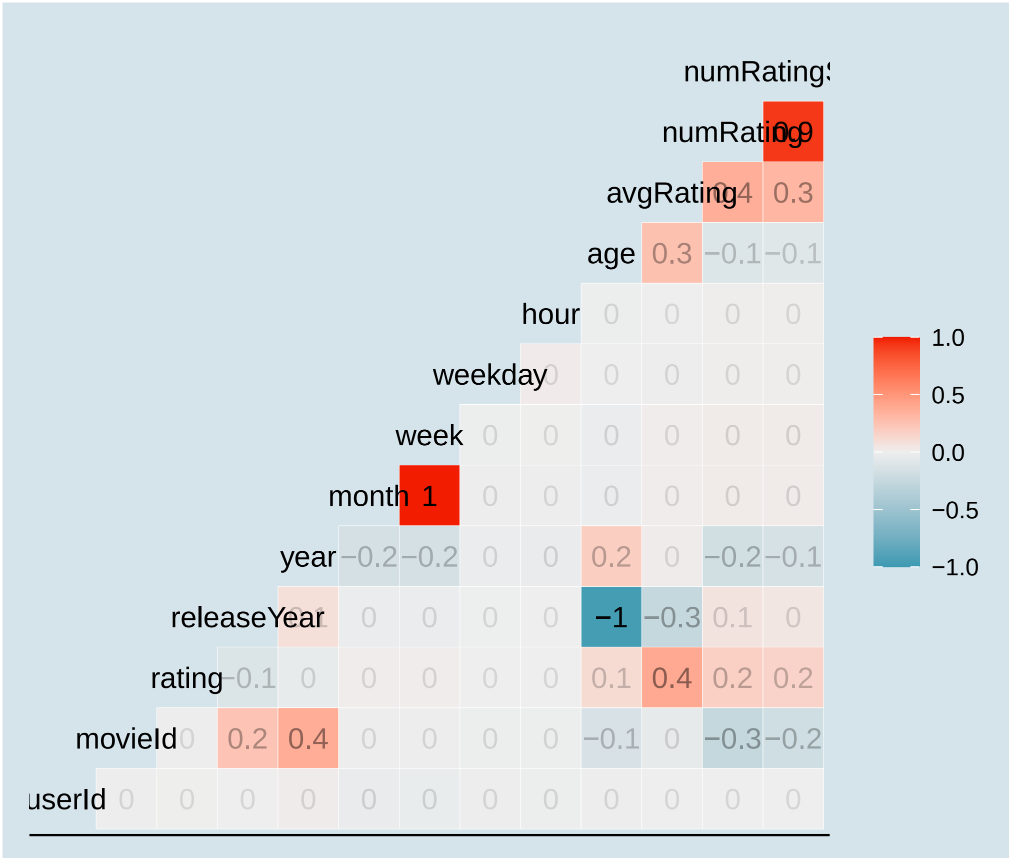
MovieLens is a well studied data set. In this section, here I will examine the predictors and how they may correlate with the outcome rather than performing comprehensive and overwhelming exploratory data analysis.

```r
ggcorr(edx, label = TRUE, label_alpha = TRUE)
```



As edx is a relatively large dataset, it will consume a lot of time doing exploratory analysis. As such, I will be performing the analysis and the model fitting based on train_set.

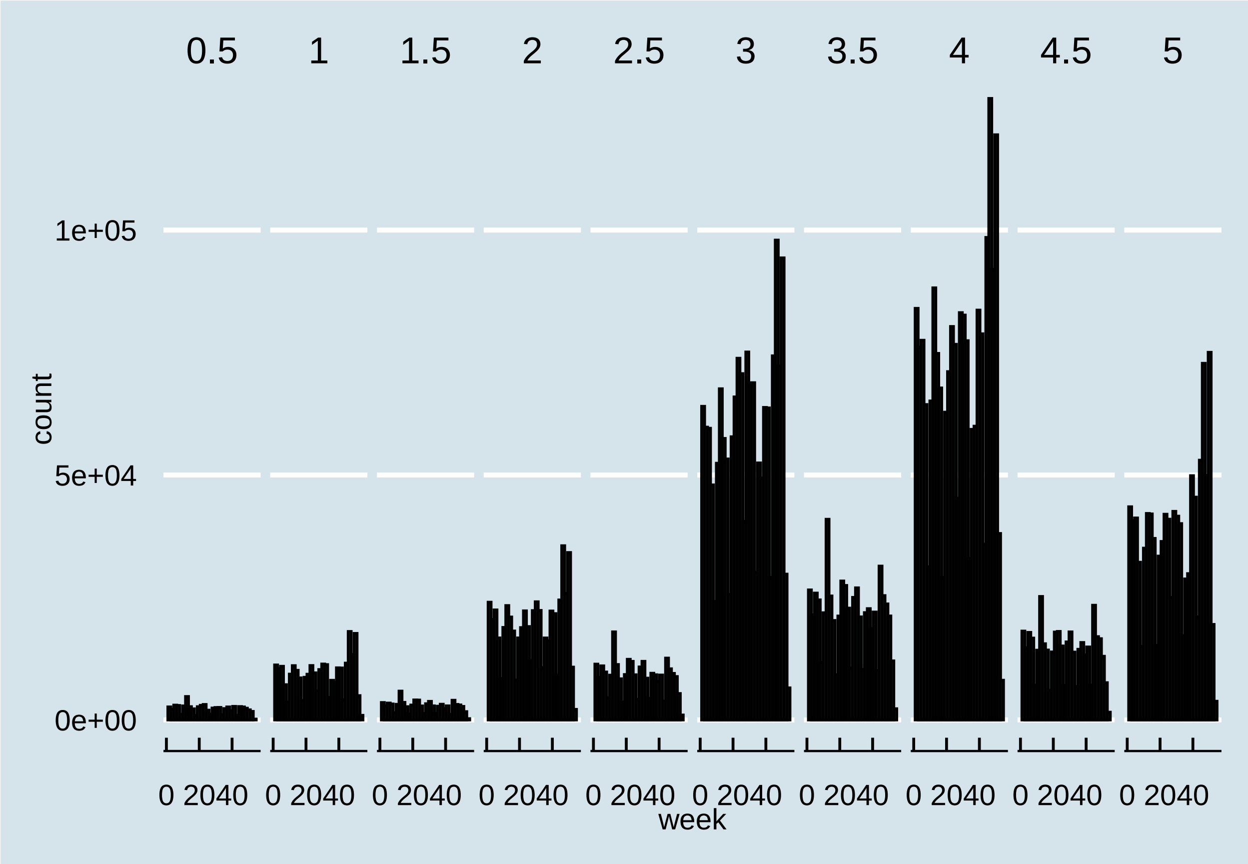
```
ggcorr(train_set, label = TRUE, label_alpha = TRUE)
```



Indeed the correlation analysis on train_set indicates that

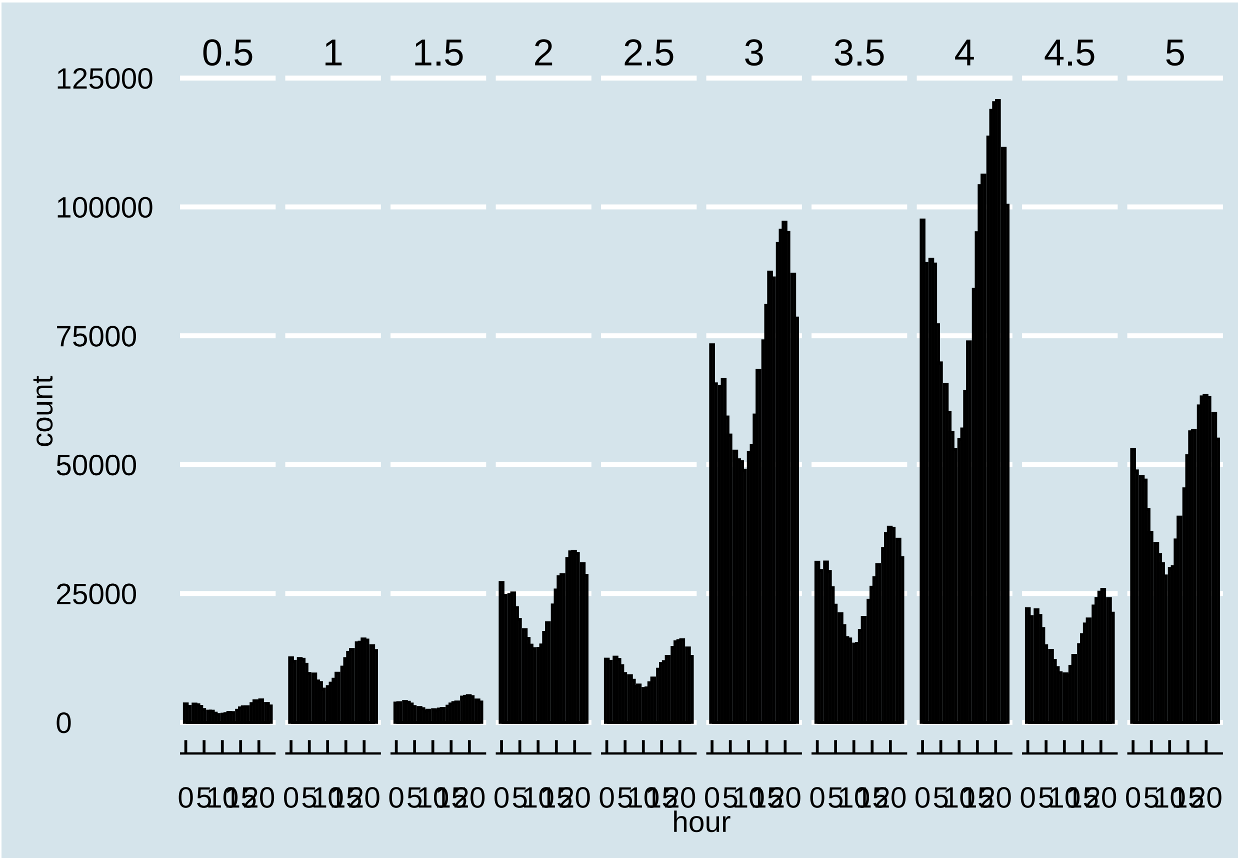
Of course there are significant correlations between the number of ratings and number of ratings strata, age and release year, month and week, movie and year, since these data are based on one another. Correlations that needs to be noted are average rating and rating, age and rating, number of ratings strata and rating.

```
train_set %>% ggplot(aes(week)) + geom_histogram(color = "black") + facet_grid(~rating)
```



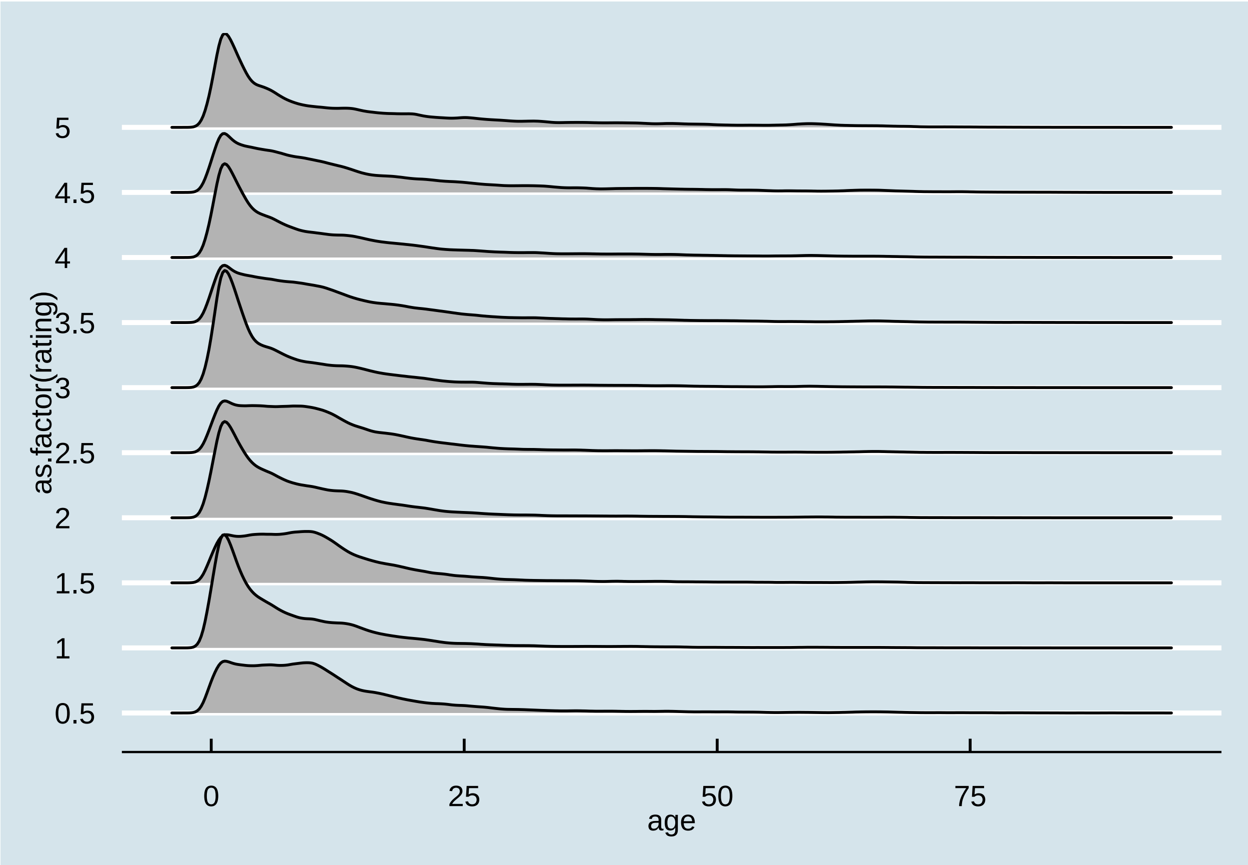
This analysis indicates that people tend to rate movies more with full rates rather than half rates as well as that people rated more during the holiday season.

```
train_set %>% ggplot(aes(hour)) + geom_histogram(color = "black") + facet_grid(~rating)
```



This analysis indicates that people tend to start rating movies more afternoon peaking during the primetime and gradually decreasing towards morning. However, the pattern is same for all ratings and does not effect on the specific rating they choose.

```
train_set %>% ggplot(aes(x = age, y = as.factor(rating))) + geom_density_ridges()
```



This analysis indicates that regardless of the rating, fewer people rated older movies. People tend to rate half points to older movies.

2.5 Models

Course benchmark data consists of 5 digits after period. So it will make more sense to output our data this way.

```
options(digits = 5)
```

I will define a function which I will use first use to examine train_set and test_set and later edx and validation

```r
fit.models <- function(train, test) {

  table.results = data.frame()

  mu <- mean(train$rating)

  bias.movies <- train %>%
    group_by(movieId) %>%
    summarize(b_movie = mean(rating - mu))

  pred.movies <- test %>%
    left_join(bias.movies, by = "movieId") %>%
    mutate(pred = mu + b_movie)

  table.results <- rbind(table.results, data.frame(name = "Movie bias",
                                                    rmse = RMSE(pred.movies$pred, test$rating)))

  bias.users <- train %>%
    left_join(bias.movies, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_user = mean(rating - mu - b_movie))

  pred.users <- test %>%
    left_join(bias.movies, by='movieId') %>%
    left_join(bias.users, by='userId') %>%
    mutate(pred = mu + b_movie + b_user)

  table.results <- rbind(table.results, data.frame(name = "Movie + user bias (MU)",
                                                    rmse = RMSE(pred.users$pred, test$rating)))

  bias.week <- train %>%
    left_join(bias.movies, by='movieId') %>%
    left_join(bias.users, by='userId') %>%
    group_by(week) %>%
    summarize(b_week = mean(rating - mu - b_movie - b_user))

  pred.week <- test %>%
    left_join(bias.movies, by='movieId') %>%
```

```
  left_join(bias.users, by='userId') %>%
  left_join(bias.week, by='week') %>%
  mutate(pred = mu + b_movie + b_user + b_week)

table.results <- rbind(table.results, data.frame(name = "MU + effect of week of the rating",
                                                 rmse = RMSE(pred.week$pred, test$rating)))

bias.hour <- train %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  group_by(hour) %>%
  summarize(b_hour = mean(rating - mu - b_movie - b_user))

pred.hour <- test %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  left_join(bias.hour, by='hour') %>%
  mutate(pred = mu + b_movie + b_user + b_hour)

table.results <- rbind(table.results, data.frame(name = "MU + effect of hour of the rating",
                                                 rmse = RMSE(pred.hour$pred, test$rating)))

bias.avgRating <- train %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  group_by(avgRating) %>%
  summarize(b_avgRating = mean(rating - mu - b_movie - b_user))

pred.avgRating <- test %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  left_join(bias.avgRating, by='avgRating') %>%
  mutate(pred = mu + b_movie + b_user + b_avgRating)

table.results <- rbind(table.results, data.frame(name = "MU + effect of existing average ratings",
                                                 rmse = RMSE(pred.avgRating$pred, test$rating)))

bias.avgRatingNumRatingStrata <- train %>%
```

```r
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  left_join(bias.avgRating, by='avgRating') %>%
  group_by(numRatingStrata) %>%
  summarize(b_avgRatingNumRatingStrata = mean(rating - mu - b_movie - b_user - b_avgRating))

pred.avgRatingNumRatingStrata <- test %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  left_join(bias.avgRating, by='avgRating') %>%
  left_join(bias.avgRatingNumRatingStrata, by='numRatingStrata') %>%
  mutate(pred = mu + b_movie + b_user + b_avgRating + b_avgRatingNumRatingStrata)

table.results <- rbind(table.results,
                       data.frame(name = "MU + effect of both existing average ratings and the
                                  strata of number of ratings",
                                  rmse = RMSE(pred.avgRatingNumRatingStrata$pred, test$rating)))

bias.releaseYear <- train %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  group_by(releaseYear) %>%
  summarize(b_releaseYear = mean(rating - mu - b_movie - b_user))

pred.releaseYear <- test %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  left_join(bias.releaseYear, by='releaseYear') %>%
  mutate(pred = mu + b_movie + b_user + b_releaseYear)

table.results <- rbind(table.results, data.frame(name = "MU + effect of release year",
                                                 rmse = RMSE(pred.releaseYear$pred, test$rating)))

bias.age <- train %>%
  left_join(bias.movies, by='movieId') %>%
  left_join(bias.users, by='userId') %>%
  group_by(age) %>%
  summarize(b_age = mean(rating - mu - b_movie - b_user))
```

```r
  pred.age <- test %>%
    left_join(bias.movies, by='movieId') %>%
    left_join(bias.users, by='userId') %>%
    left_join(bias.age, by='age') %>%
    mutate(pred = mu + b_movie + b_user + b_age)

  table.results <- rbind(table.results, data.frame(name = "MU + effect of difference between release year and rating year",
                                                   rmse = RMSE(pred.age$pred, test$rating)))

  table.results
}
```

```r
fit.models(train_set, test_set)
```

```
##                                                                                               name
## 1                                                                                       Movie bias
## 2                                                                              Movie + user bias (MU)
## 3                                                                     MU + effect of week of the rating
## 4                                                                     MU + effect of hour of the rating
## 5                                                                MU + effect of existing average ratings
## 6 MU + effect of both existing average ratings and the \n                    strata of number of ratings
## 7                                                                            MU + effect of release year
## 8                                         MU + effect of difference between release year and rating year
##      rmse
## 1 0.94374
## 2 0.86593
## 3 0.86593
## 4 0.86593
## 5 0.86592
## 6 0.86573
## 7 0.86561
## 8 0.86550
```

I will select rating age based and average rating as well as the number of rating stratification based models because they are the best performers and I will also use regularization in order to improve against overfitting.

```r
table.results = fit.models(edx, validation)
```

3 Results

regularize.avgRating function calculates RMSEs of a training set and a test set using different values of tuning parameters using the model that is based on the biases of movies and users as well as the existing average ratings of the movies.

```r
regularize.avgRating <- function(train, test, lambda) {

  mu <- mean(train$rating)

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

  bias_users <- train %>%
    left_join(bias_movies, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - mu - b_movie)/(lambda + n()))

  bias.avgRating <- train %>%
    left_join(bias_movies, by='movieId') %>%
    left_join(bias_users, by='userId') %>%
    group_by(avgRating) %>%
    summarize(b_avgRating = mean(rating - mu - b_movie - b_user))

  bias.avgRatingNumRatingStrata <- train %>%
    left_join(bias_movies, by='movieId') %>%
    left_join(bias_users, by='userId') %>%
    left_join(bias.avgRating, by='avgRating') %>%
    group_by(numRatingStrata) %>%
    summarize(b_avgRatingNumRatingStrata = mean(rating - mu - b_movie - b_user - b_avgRating))

  pred.avgRatingNumRatingStrata <- test %>%
    left_join(bias_movies, by='movieId') %>%
    left_join(bias_users, by='userId') %>%
    left_join(bias.avgRating, by='avgRating') %>%
```

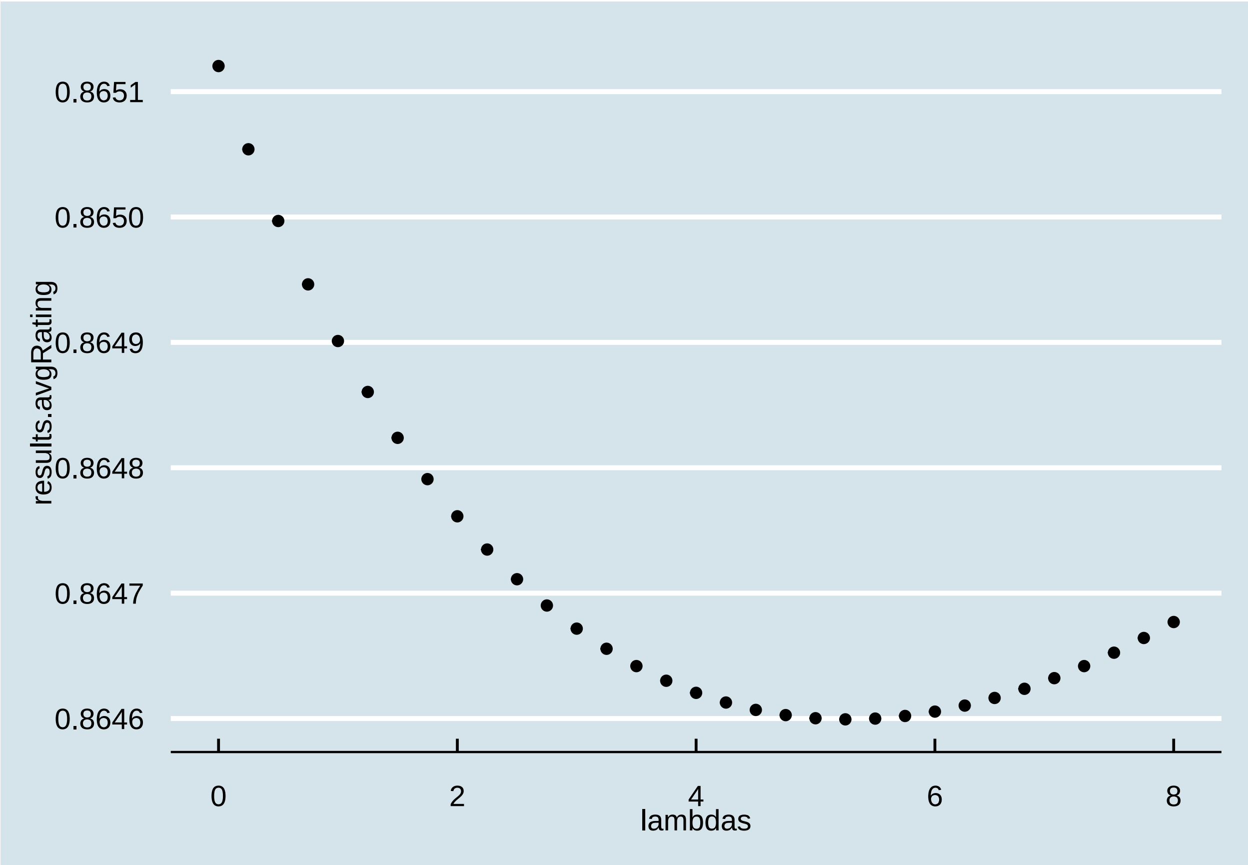
```r
    left_join(bias.avgRatingNumRatingStrata, by='numRatingStrata') %>%
    mutate(pred = mu + b_movie + b_user + b_avgRating + b_avgRatingNumRatingStrata)

  RMSE(pred.avgRatingNumRatingStrata$pred, test$rating)

}
```

regularize.age function calculates RMSEs of a training set and a test set using different values of tuning parameters using the model that is based on the biases of movies and users as well as the age of the rating weighted by the stratification of the number of ratings.

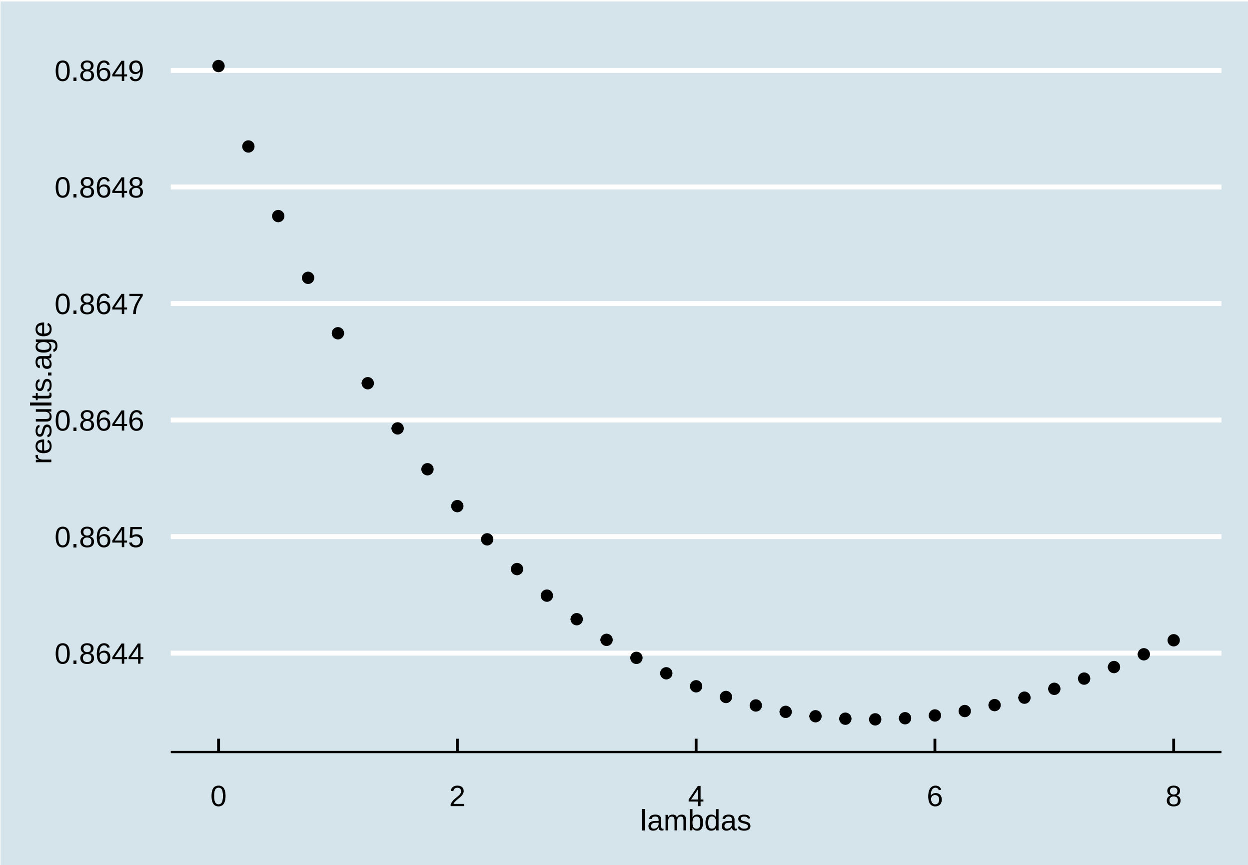
```r
regularize.age <- function(train, test, lambda) {

  mu <- mean(train$rating)

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

  bias_users <- train %>%
    left_join(bias_movies, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_user = sum(rating - mu - b_movie)/(lambda + n()))

  bias.age <- train %>%
    left_join(bias_movies, by='movieId') %>%
    left_join(bias_users, by='userId') %>%
    group_by(age) %>%
    summarize(b_age = mean(rating - mu - b_movie - b_user))

  pred.age <- test %>%
    left_join(bias_movies, by='movieId') %>%
    left_join(bias_users, by='userId') %>%
    left_join(bias.age, by='age') %>%
    mutate(pred = mu + b_movie + b_user + b_age)

  RMSE(pred.age$pred, test$rating)

}
```

Cross-validating lambdas for the selected models

```r
lambdas = seq(0, 8, .25)

results.avgRating <- sapply(lambdas, function(lambda) {
  regularize.avgRating(edx, validation, lambda)
})

results.age <- sapply(lambdas, function(lambda) {
  regularize.age(edx, validation, lambda)
})

qplot(lambdas, results.avgRating)
```




```r
qplot(lambdas, results.age)
```



The best performing regularized model yields RMSE of 0.8646

```
min(results.avgRating)
```

```
## [1] 0.8646
```

```
table.results <- rbind(table.results,
                        data.frame(name = "MU + effect of existing average ratings + regularization",
                                   rmse = min(results.avgRating)))
```

The best performing regularized model yields RMSE of 0.86434

```r
min(results.age)
```

```
## [1] 0.86434
```

```r
table.results <- rbind(table.results,
                       data.frame(name = "MU + effect of difference between release year and rating year + regularization",
                                  rmse = min(results.age)))
```

4 Conclusion

```r
table.results
```

```
##                                                                                                        name
## 1                                                                                                Movie bias
## 2                                                                                      Movie + user bias (MU)
## 3                                                                            MU + effect of week of the rating
## 4                                                                            MU + effect of hour of the rating
## 5                                                                      MU + effect of existing average ratings
## 6  MU + effect of both existing average ratings and the \n                          strata of number of ratings
## 7                                                                                   MU + effect of release year
## 8                                                     MU + effect of difference between release year and rating year
## 9                                                           MU + effect of existing average ratings + regularization
## 10                                     MU + effect of difference between release year and rating year + regularization
##       rmse
## 1  0.94391
## 2  0.86535
## 3  0.86534
## 4  0.86534
## 5  0.86532
## 6  0.86512
## 7  0.86500
## 8  0.86490
## 9  0.86460
## 10 0.86434
```

References

[1] https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+1T2021/block-v1:HarvardX+PH125.9x+1T2021+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+1T2021+type@vertical+block@e9abcdd945b1416098a15fc95807b5db retrieved in June 2021

[2] Irizarry, Introduction to Data Science, 2019, found at https://leanpub.com/datasciencebook