Capstone - MovieLens (2021)

Martin Knell

28/04/2021

1. Overview

This project is part of the Capstone Assignment of the HarvardX Professional Certificate in Data Science Program. The objective of the project is to develop a movie recommendation algorithm based on a subset of the MovieLens data set, with a target RMSE (Root Mean Square Error) below 0.86490.

1.1 Introduction

Movie recommendation systems predict the rating that a user u will give to a movie (item) i. The system will then recommend movies to a user, for which the algorithm predicts a high rating from this user. Recommendations are typically based on

a) Previous user viewing history and ratings, finding similar movies to the ones the user liked
b) Movie ratings from similar users

The aim of this project is to train a machine learning algorithm to predict user ratings based on a 10M row subset of the MovieLens data set.

1.2 Success Measure

The quality of the algorithm will be evaluated on a validation data set. The success measure is the RMSE (Root Mean Square Error) 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 combination, y_ui is the rating of of movie i by user u, and y_hat is the prediction of this rating.

This formula is executed with the following R-function

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

This project aims to find an algorithm with an RMSE below 0.86490

1.3 Data Set

The subset of 10M rows from the movie lens data set is produced using the provided code. The code creates a training data set (edx) and a validation set (validation) containing 10% of the data. The code can be found in the R-script and R-markdown document. It will not be displayed in the pdf report.

2. Data Exploration

2.1 Structure of Data Set

The edx dataset has the following structure

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The information contains the user ID, movie ID, title, genre, rating and time of rating. The total training data set contains approx. 9M ratings for over 10,000 movies from nearly 70,000 users.

```
nrow(edx)
```

```
## [1] 9000055
```

```
movies <- unique(edx$movieId)
length(movies)
```

```
## [1] 10677
```

```
users <- unique(edx$userId)
length(users)
```

```
## [1] 69878
```

The summary information for the dataset confirms there are no missing values.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

2.2 Rating Distribution

The most common movie rating is 4, followed by 3. Full star ratings are more common than half star ratings. Figure 1 shows the general distribution of ratings

```
rating_count <- edx %>% group_by(rating) %>% summarize(n = n())
rating_count[order(rating_count$n),]
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1.5  106426
##  3    2.5  333010
##  4    1    345679
##  5    4.5  526736
##  6    2    711422
##  7    3.5  791624
##  8    5   1390114
##  9    3   2121240
## 10    4   2588430
```

```
half_star <- rating_count %>% filter(rating %in% c(0.5, 1.5, 2.5, 3.5, 4.5))
full_star <- rating_count %>% filter(rating %in% c(1,2,3,4,5))

sum(half_star$n)
```

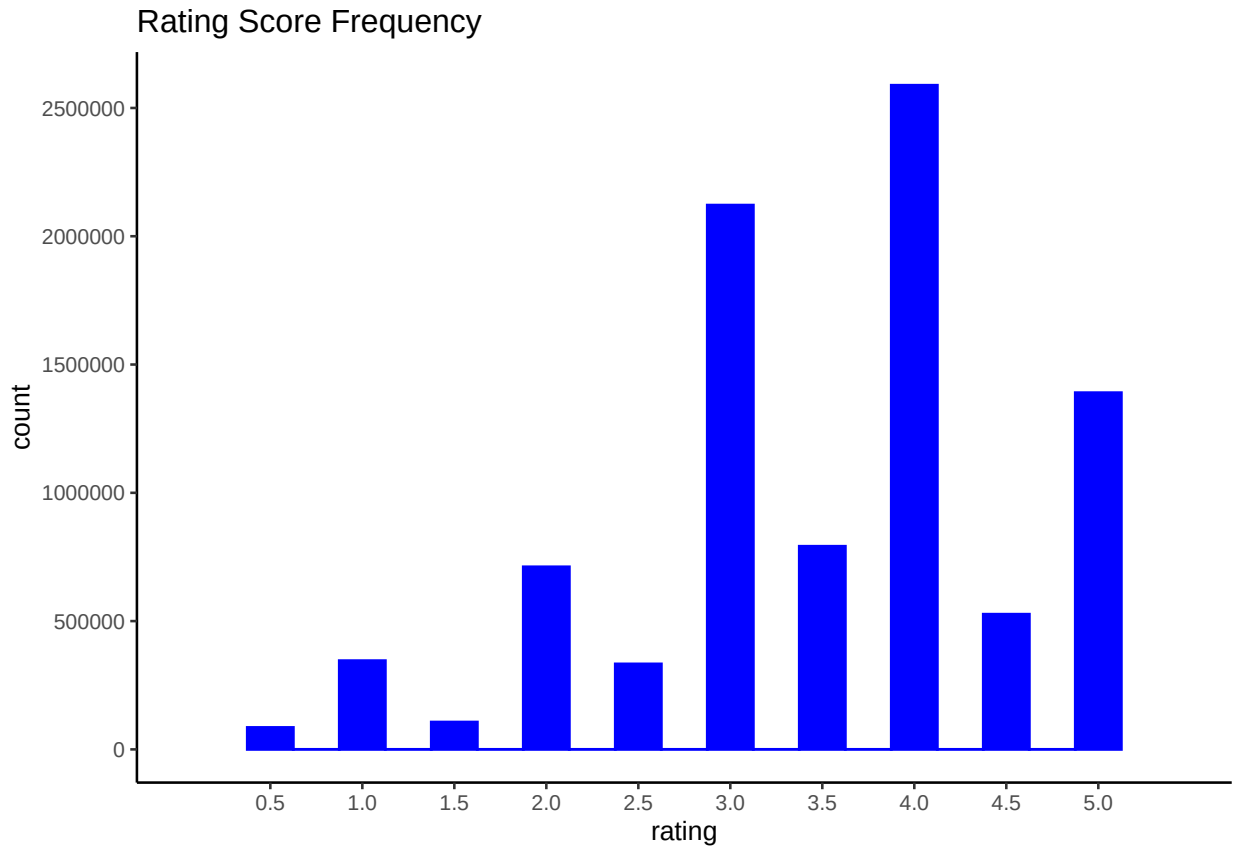
```
## [1] 1843170
```

```
sum(full_star$n)
```

```
## [1] 7156885
```

Figure 1: Rating Score Distribution

```
rating_plot <- edx %>% ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "blue", fill = "blue") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  theme_classic(base_size = 10) + ggtitle("Rating Score Frequency")
rating_plot
```



1% of movies in the dataset have over 10,000 ratings. About half of movies have over 100 ratings. A significant amount of 12% of movies have only one rating.

```
movie_count <- edx %>% group_by(movieId) %>% summarize(n = n())
movie_count[order(-movie_count$n), ]
```

```
## # A tibble: 10,677 x 2
##    movieId     n
##      <dbl> <int>
##  1     296 31362
##  2     356 31079
##  3     593 30382
##  4     480 29360
##  5     318 28015
##  6     110 26212
##  7     457 25998
##  8     589 25984
##  9     260 25672
```

```
## 10     150 24284
## # ... with 10,667 more rows
```

```
mean(movie_count$n == 1)
```

```
## [1] 0.01180107
```

```
mean(movie_count$n >= 100)
```

```
## [1] 0.5348881
```

```
mean(movie_count$n >= 1000)
```

```
## [1] 0.1781399
```

```
mean(movie_count$n >= 10000)
```

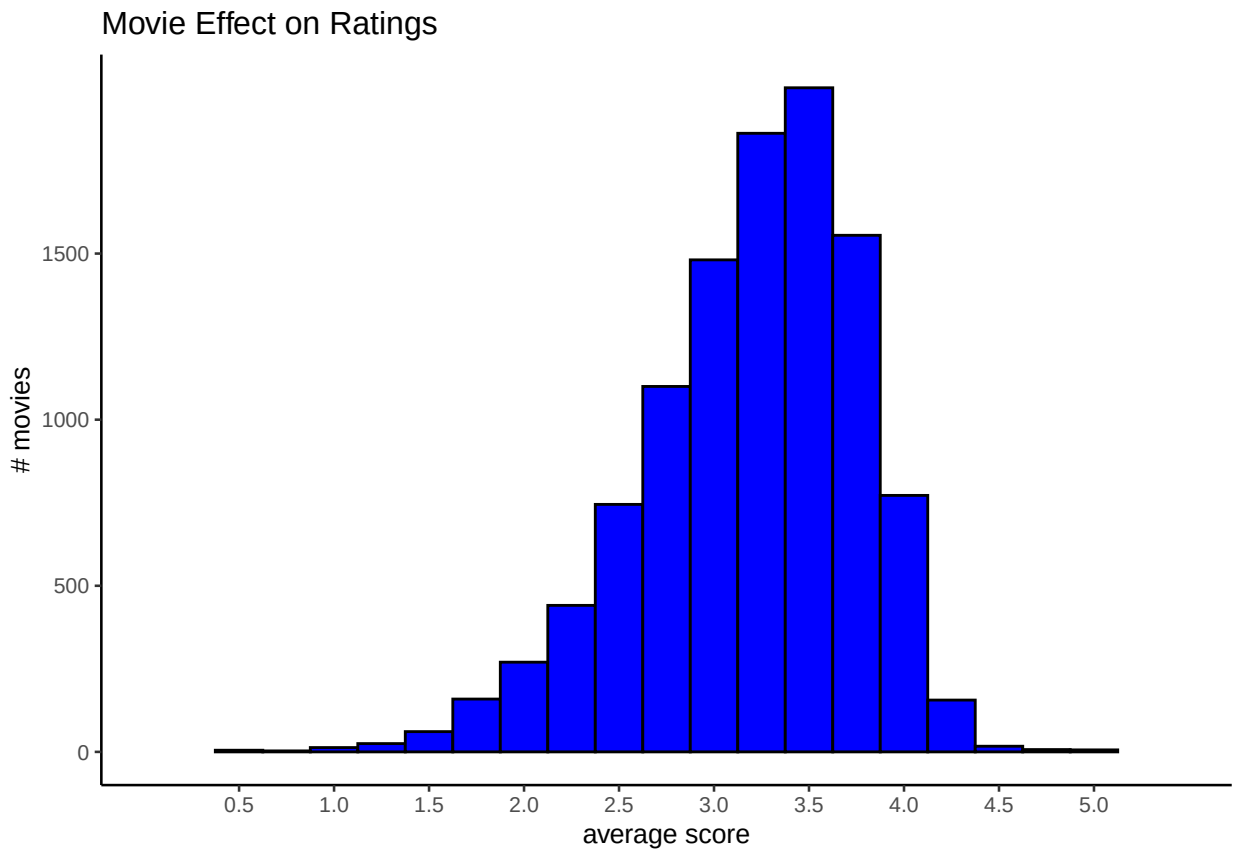
```
## [1] 0.01339328
```

2.3 Movie Effect on Ratings

The average rating on a movie is very dependent on the movie, which suggests a strong movie effect on ratings. Figure 2 shows the distribution of movie rating averages. The total average of movie ratings is 3.5.

Figure 2: Movie Effect on Ratings

```
movie_effect <- edx %>% group_by(movieId) %>% summarize(avg_score = mean(rating)) %>%
  ggplot(aes(avg_score)) +
  geom_histogram(binwidth = 0.25, color = "black", fill = "blue") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 1500, 500))) +
  theme_classic(base_size = 10) + ggtitle("Movie Effect on Ratings") +
  xlab("average score") + ylab("# movies")
movie_effect
```




```
mean(edx$rating)
```

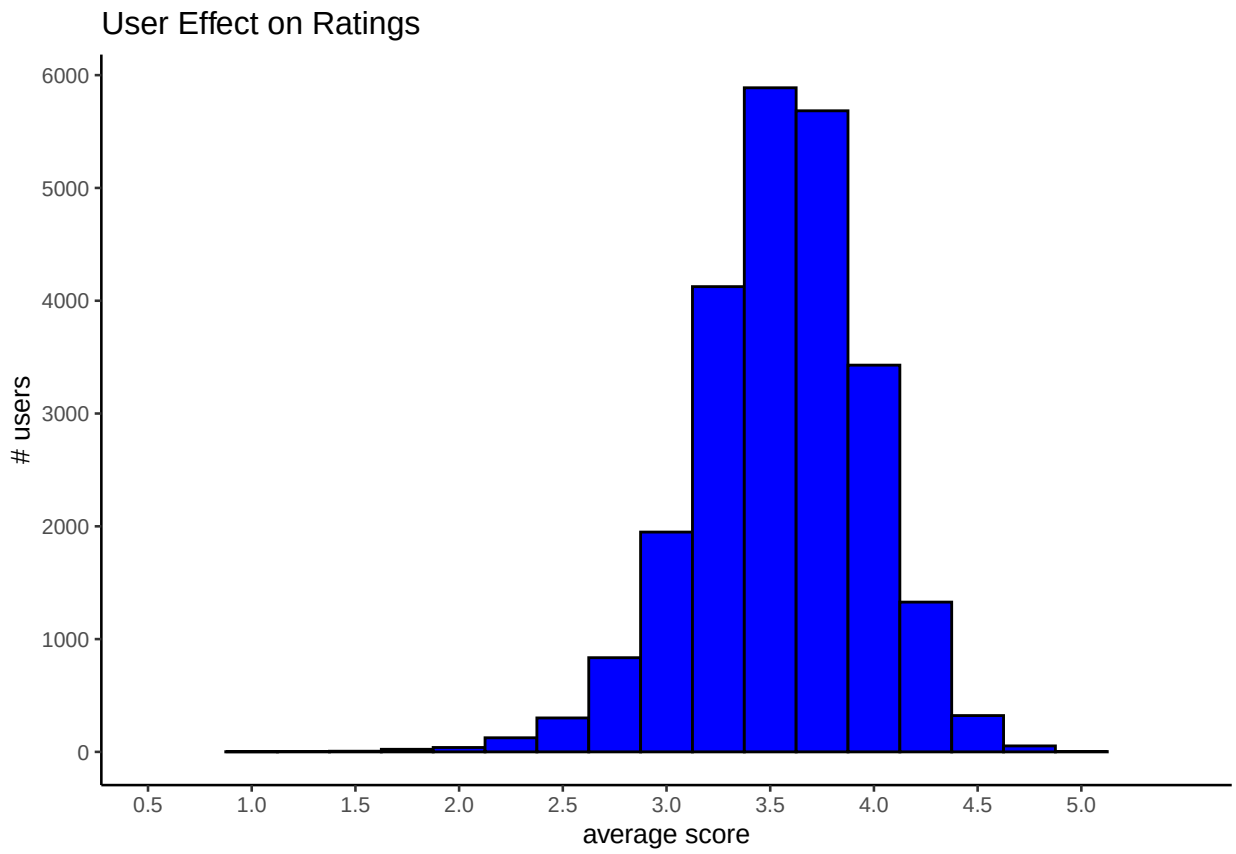
```
## [1] 3.512465
```

2.4 User Effect on Ratings

There is also a user effect of ratings. That means different users tend to give different rating averages. User averages can range from generous to harsh. Figure 3 shows the distribution of user rating averages.

Figure 3: User Effect on Ratings

```
user_effect <- edx %>% group_by(userId) %>% summarize(avg_score = mean(rating), n = n()) %>%
  filter(n >= 100) %>% ggplot(aes(avg_score)) +
  geom_histogram(binwidth = 0.25, color = "black", fill = "blue") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 10000, 1000))) +
  theme_classic(base_size = 10) + ggtitle("User Effect on Ratings") +
  xlab("average score") + ylab("# users")
user_effect
```

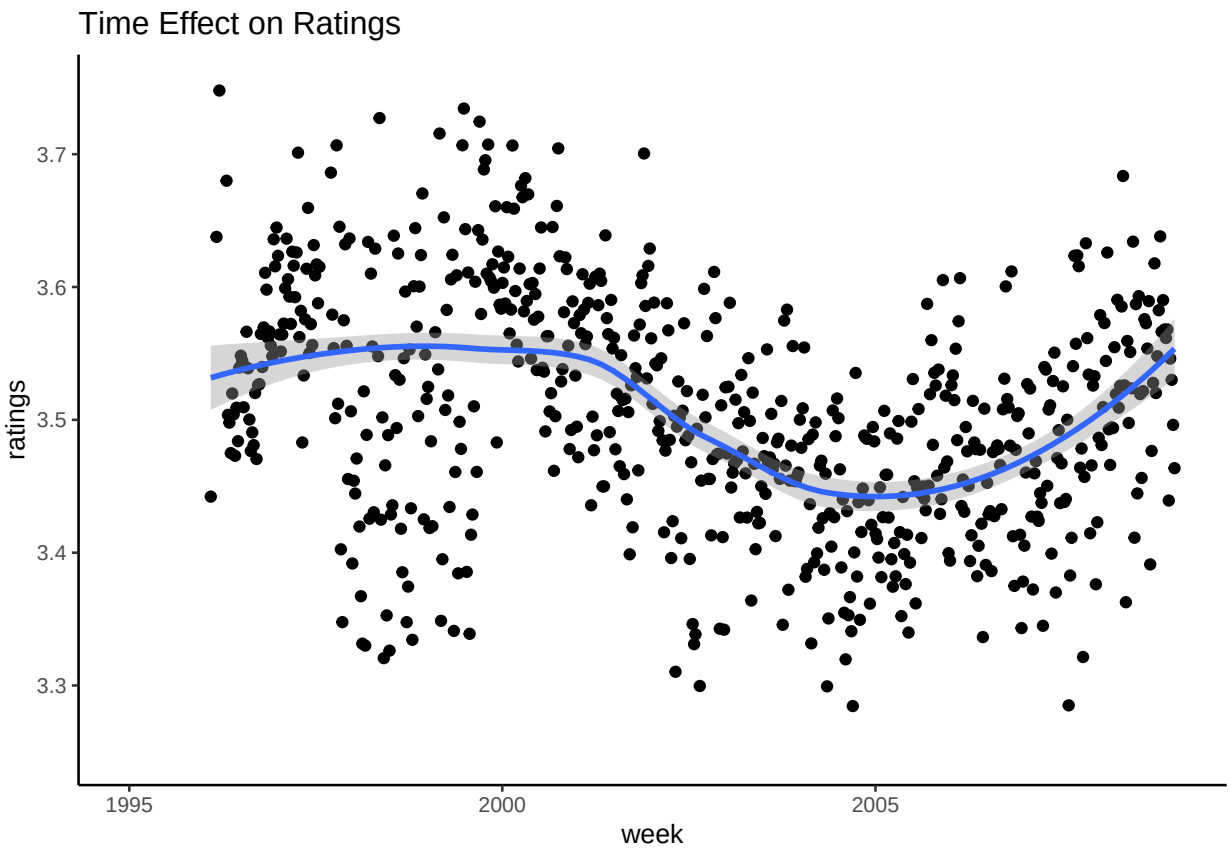


2.5 Time Effect on Ratings

Apart from movie effect and user effect on ratings, overall rating averages also change over time as shown in Figure 4. The lowest average ratings were observed around 2005.

Figure 4: Time Effect on Ratings

```
time_effect <- edx %>% mutate(date = as_datetime(timestamp)) %>%
  mutate(week = round_date(date, unit = "week")) %>%
  group_by(week) %>% summarize(avg = mean(rating)) %>% ggplot(aes(week, avg)) +
  geom_point() + geom_smooth() + ylim(3.25,3.75) +
  ylab("ratings") + ggtitle("Time Effect on Ratings") +
  theme_classic(base_size = 10)
time_effect
```



3.0 Methods and Analysis

We will create a baseline for RMSE comparison by predicting the overall average for movie ratings. We will then add the movie effect and user effect on ratings as shown in chapter 2. Finally, we will compare Regularization and Gradient Boosting Machines to achieve the target RMSE for this assignment. Note a knn approach with k = 25 was also attempted but abandoned due to required run times on a standard laptop. The code for the knn model can be found in the R-script.

3.1 Average Rating

As a baseline, we simply predict the average of all ratings and derive the RMSE

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

Predicting just the average rating delivers an RMSE of 1.061202

```
avg <- mean(edx$rating)

rmse_1 <- RMSE(validation$rating, avg)

rmses <- data_frame(method = "Predict Average", RMSE = rmse_1)
rmses %>% knitr::kable()
```

method	RMSE
Predict Average	1.061202

3.2 Adding the Movie Effect Term

As shown in chapter 2.3 there is a movie effect on ratings. That means different movies attract different average ratings.We account for this effect by adding the movie effect bi to the prediction formula

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

The least square estimate bi_hat is equal to the deviation of the individual movie averages from the overall average. Adding the movie effect to the algorithm delivers an RMSE of 0.9439087.

```
movie_effect <- edx %>% group_by(movieId) %>% summarize(bi = mean(rating - avg))
predicted_2 <- avg + validation %>% left_join(movie_effect, by = 'movieId') %>% .$bi

rmse_2 <- RMSE(predicted_2, validation$rating)

rmses <- bind_rows(rmses, data_frame(method = "Add Movie Effect", RMSE = rmse_2))
rmses %>% knitr::kable()
```

method	RMSE
Predict Average	1.0612018
Add Movie Effect	0.9439087

3.3 Adding the User Effect Term

As shown in chapter 2.4 there is also a user effect on ratings. That means different users tend to give different average ratings. Some users will be generous with their ratings, others will be harsh, others will be somewhere in the middle. We account for the user effect by adding an additional user term to the prediction formula

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

Again, the least square estimate can be obtained by adding the individual user average deviation from the overall average and movie average. Adding the user effect to the approach delivers an RMSE of 0.8653488.

```
user_effect <- edx %>% left_join(movie_effect, by = 'movieId') %>% group_by(userId) %>%
  summarize(bu = mean(rating - avg - bi))

predicted_3 <- validation %>% left_join(movie_effect, by = 'movieId') %>%
  left_join(user_effect, by = 'userId') %>% mutate(pred_3 = avg + bi + bu) %>% .$pred_3

rmse_3 <- RMSE(predicted_3, validation$rating)

rmses <- bind_rows(rmses, data_frame(method = "Add User Effect", RMSE = rmse_3))
rmses %>% knitr::kable()
```

method	RMSE
Predict Average	1.0612018
Add Movie Effect	0.9439087
Add User Effect	0.8653488

3.4 Regularization

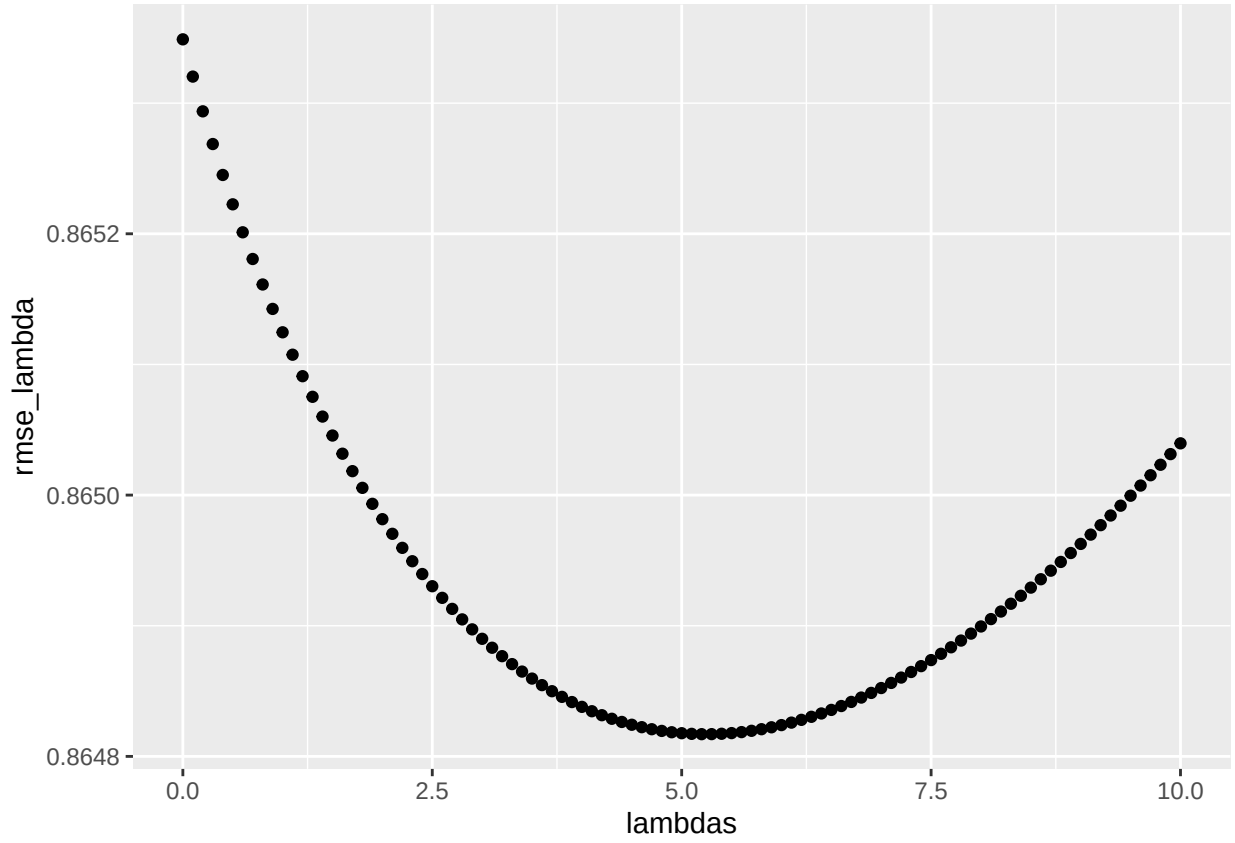
Movie ratings that are based on very low numbers of ratings, especially the 12% based on only one rating, lead to noisy estimates that impact the overall performance of the model. We therefore expect that the RMSE can be further improved by introducing a penalty term that shrinks the impact of those ratings.

We achieve this with the code below. It can be seen that for large sample sizes the penalty term gets effectively ignored. For small sample sizes it is shrunken towards zero. Figure 5 shows the impact of different lambdas on model performance. The best performance is achieved for lambda = 5.2, which delivers an RMSE of 0.8648170. This result is below the target RMSE for this assignment.

```
lambdas <- seq(0,10,0.1)

rmse_lambda <- sapply(lambdas, function(l){
  mu <- mean(edx$rating)
  bi <- edx %>% group_by(movieId) %>% summarize(bi = sum(rating - mu)/(n()+l))
  bu <- edx %>% left_join(bi, by="movieId") %>% group_by(userId) %>% summarize(bu = sum(rating - bi - mu

  predicted_ratings <- validation %>% left_join(bi, by = "movieId") %>% left_join(bu, by = "userId") %>%
    mutate(pred = mu + bi + bu) %>% .$pred

  return(RMSE(predicted_ratings, validation$rating))
})
```

Figure 5: Impact of Lambda on Model Performance

```
# Find optimal lambda

qplot(lambdas, rmse_lambda)
```




```
lambda <- lambdas[which.min(rmse_lambda)]
lambda
```

```
## [1] 5.2
```

```
rmse_4 <- min(rmse_lambda)

rmses <- bind_rows(rmses, data_frame(method = "Regularization", RMSE = rmse_4))
rmses %>% knitr::kable()
```

method	RMSE
Predict Average	1.0612018
Add Movie Effect	0.9439087
Add User Effect	0.8653488
Regularization	0.8648170

3.5 Gradient Boosting: Basic Grid Search

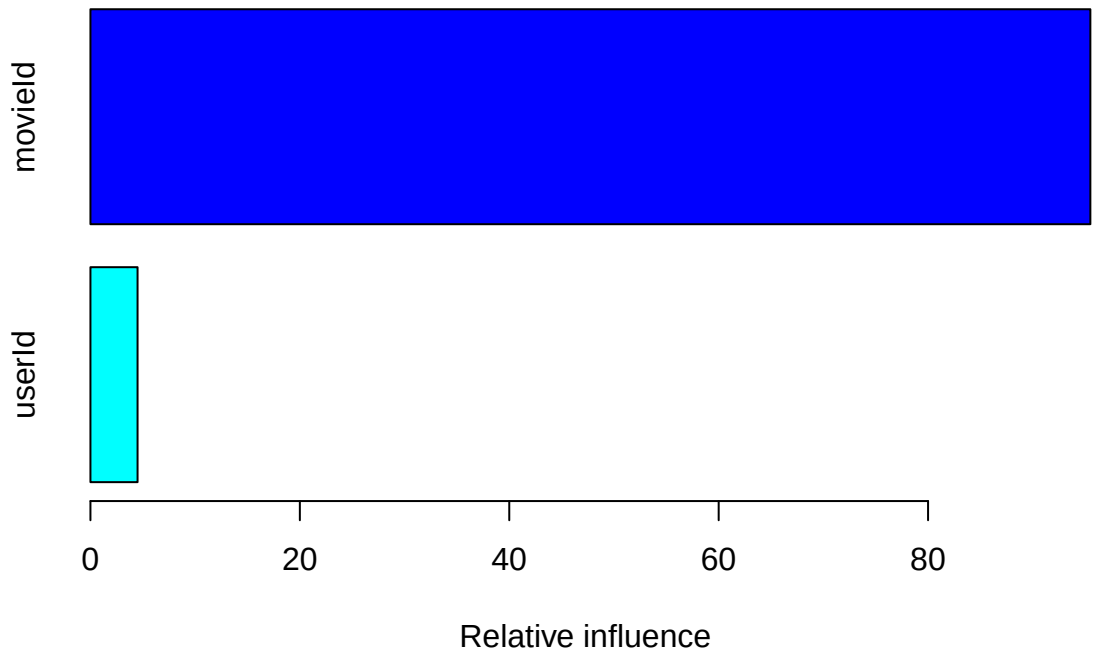
As an alternative approach we run GBM (Gradient Boosting Machines) over the two input variables movie ID and user ID. A basic grid search suggests that a learning rate of 0.1 delivers better results than 0.01. Furthermore a larger number of trees and higher interaction depth delivers better results. Figure 6 shows the impact of the selected tune grid parameters on model performance.

```r
# Grid Search (w. 3-fold cross validation)

fitControl <- trainControl(method = "repeatedcv", number = 3, repeats = 1)
gbmGrid <-  expand.grid(interaction.depth = c(3,5,7), n.trees = c(100,300,500),
                        shrinkage = c(0.1, 0.01), n.minobsinnode = 10)

rating_pred <- train(rating ~ movieId + userId, data = edx, method = "gbm",
                     trControl = fitControl, tuneGrid = gbmGrid)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1233             nan     0.0100    0.0005
##      2        1.1228             nan     0.0100    0.0005
##      3        1.1222             nan     0.0100    0.0005
##      4        1.1217             nan     0.0100    0.0005
##      5        1.1212             nan     0.0100    0.0005
##      6        1.1207             nan     0.0100    0.0005
##      7        1.1202             nan     0.0100    0.0005
##      8        1.1198             nan     0.0100    0.0005
##      9        1.1193             nan     0.0100    0.0005
##     10        1.1188             nan     0.0100    0.0005
##     20        1.1147             nan     0.0100    0.0004
##     40        1.1086             nan     0.0100    0.0003
##     60        1.1045             nan     0.0100    0.0002
##     80        1.1016             nan     0.0100    0.0001
##    100        1.0995             nan     0.0100    0.0001
##    120        1.0979             nan     0.0100    0.0001
##    140        1.0967             nan     0.0100    0.0000
##    160        1.0958             nan     0.0100    0.0000
##    180        1.0950             nan     0.0100    0.0000
##    200        1.0941             nan     0.0100    0.0001
##    220        1.0934             nan     0.0100    0.0000
##    240        1.0928             nan     0.0100    0.0000
##    260        1.0920             nan     0.0100    0.0000
##    280        1.0913             nan     0.0100    0.0001
##    300        1.0905             nan     0.0100    0.0000
##    320        1.0896             nan     0.0100    0.0000
##    340        1.0888             nan     0.0100    0.0000
##    360        1.0881             nan     0.0100    0.0000
##    380        1.0874             nan     0.0100    0.0000
##    400        1.0867             nan     0.0100    0.0000
##    420        1.0860             nan     0.0100    0.0000
##    440        1.0854             nan     0.0100    0.0000
##    460        1.0848             nan     0.0100    0.0000
##    480        1.0842             nan     0.0100    0.0000
##    500        1.0835             nan     0.0100    0.0000
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1232             nan     0.0100    0.0006
##      2        1.1226             nan     0.0100    0.0006
##      3        1.1220             nan     0.0100    0.0006
##      4        1.1215             nan     0.0100    0.0006
##      5        1.1209             nan     0.0100    0.0006
##      6        1.1203             nan     0.0100    0.0006
```

```
##     320        1.0146             nan     0.1000    0.0001
##     340        1.0124             nan     0.1000    0.0001
##     360        1.0103             nan     0.1000    0.0000
##     380        1.0082             nan     0.1000    0.0003
##     400        1.0063             nan     0.1000    0.0001
##     420        1.0045             nan     0.1000    0.0002
##     440        1.0024             nan     0.1000    0.0000
##     460        1.0003             nan     0.1000    0.0001
##     480        0.9983             nan     0.1000    0.0001
##     500        0.9965             nan     0.1000    0.0001
```

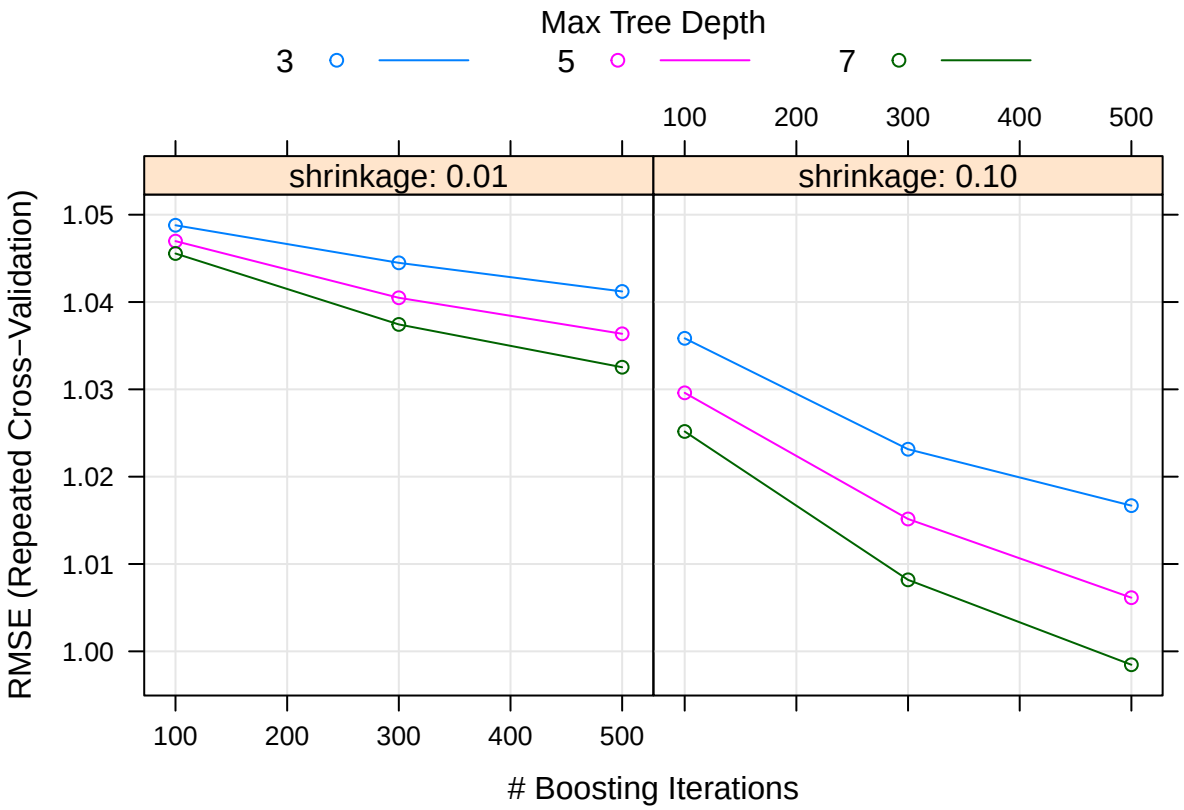
```
forecast <- predict(rating_pred, validation)

summary(rating_pred)
```




```
##             var   rel.inf
## movieId movieId 95.499058
## userId   userId  4.500942
```

Figure 6: Results of GBM Grid Search

```
plot(rating_pred)  # Optimal: Shrinkage 0.1, interaction.depth 7, 500 trees
```




```
rmse_5 <- RMSE(forecast, validation$rating)

rmses <- bind_rows(rmses, data_frame(method = "GBM Initial Grid", RMSE = rmse_5))
rmses %>% knitr::kable()
```

method	RMSE
Predict Average	1.0612018
Add Movie Effect	0.9439087
Add User Effect	0.8653488
Regularization	0.8648170
GBM Initial Grid	0.9990090

3.6 Gradient Boosting: Tuned Parameters

Based on the findings in 3.5 we run a tuned model with a learning rate of 0.1, interaction depth of 9, 3-fold cross validation and 3000 trees. This algorithm takes 10 hours to run on a standard laptop. The approach delivers an improved RMSE compared to the initial grid, but still not as good as the Regularization approach.

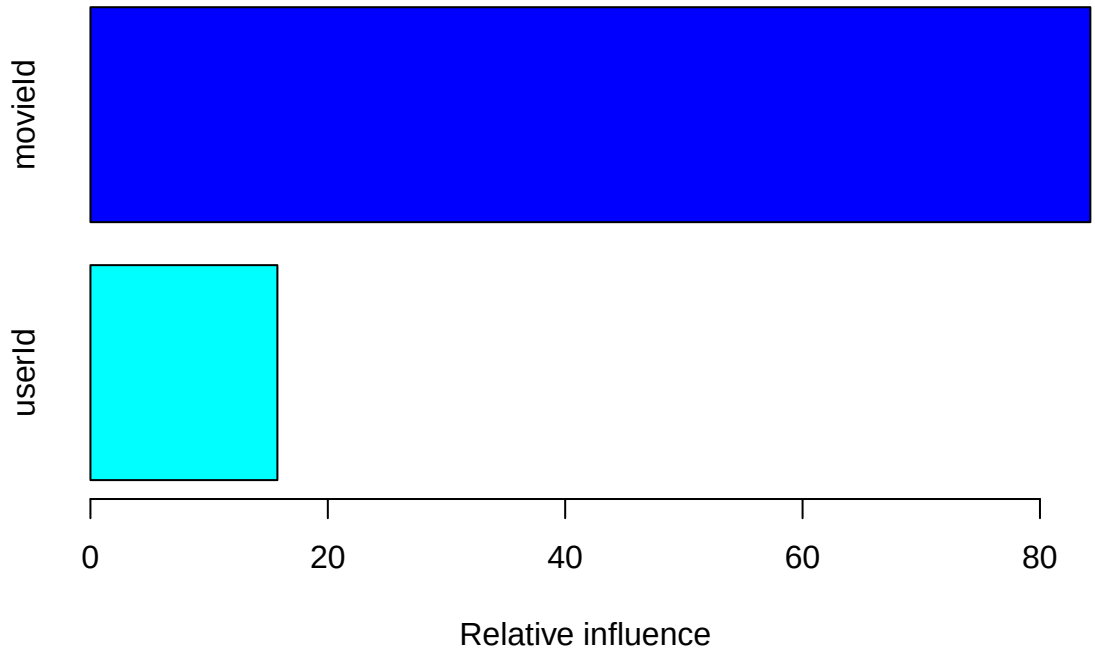
```
fitControl <- trainControl(method = "repeatedcv", number = 3, repeats = 1)
gbmGrid <-  expand.grid(interaction.depth = 9, n.trees = 3000, shrinkage = 0.1,
                        n.minobsinnode = 10)

rating_pred <- train(rating ~ movieId + userId, data = edx, method = "gbm",
```

```
                    trControl = fitControl, tuneGrid = gbmGrid)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.1173             nan     0.1000    0.0068
##      2        1.1117             nan     0.1000    0.0057
##      3        1.1072             nan     0.1000    0.0044
##      4        1.1035             nan     0.1000    0.0037
##      5        1.1003             nan     0.1000    0.0032
##      6        1.0975             nan     0.1000    0.0028
##      7        1.0951             nan     0.1000    0.0023
##      8        1.0928             nan     0.1000    0.0023
##      9        1.0913             nan     0.1000    0.0016
##     10        1.0895             nan     0.1000    0.0018
##     20        1.0783             nan     0.1000    0.0011
##     40        1.0636             nan     0.1000    0.0006
##     60        1.0550             nan     0.1000    0.0002
##     80        1.0485             nan     0.1000    0.0002
##    100        1.0426             nan     0.1000    0.0001
##    120        1.0389             nan     0.1000    0.0002
##    140        1.0340             nan     0.1000    0.0001
##    160        1.0297             nan     0.1000    0.0006
##    180        1.0255             nan     0.1000    0.0005
##    200        1.0218             nan     0.1000    0.0001
##    220        1.0185             nan     0.1000    0.0003
##    240        1.0150             nan     0.1000    0.0003
##    260        1.0122             nan     0.1000    0.0000
##    280        1.0091             nan     0.1000    0.0005
##    300        1.0067             nan     0.1000    0.0001
##    320        1.0043             nan     0.1000    0.0002
##    340        1.0026             nan     0.1000    0.0000
##    360        1.0004             nan     0.1000    0.0001
##    380        0.9977             nan     0.1000    0.0000
##    400        0.9950             nan     0.1000    0.0002
##    420        0.9931             nan     0.1000    0.0001
##    440        0.9912             nan     0.1000    0.0000
##    460        0.9894             nan     0.1000    0.0000
##    480        0.9876             nan     0.1000    0.0000
##    500        0.9860             nan     0.1000    0.0000
##    520        0.9839             nan     0.1000    0.0001
##    540        0.9822             nan     0.1000    0.0001
##    560        0.9806             nan     0.1000    0.0001
##    580        0.9786             nan     0.1000    0.0001
##    600        0.9773             nan     0.1000    0.0001
##    620        0.9759             nan     0.1000    0.0001
##    640        0.9745             nan     0.1000    0.0000
##    660        0.9732             nan     0.1000    0.0000
##    680        0.9720             nan     0.1000    0.0000
##    700        0.9707             nan     0.1000    0.0000
##    720        0.9695             nan     0.1000    0.0001
##    740        0.9684             nan     0.1000    0.0001
##    760        0.9673             nan     0.1000    0.0001
##    780        0.9660             nan     0.1000    0.0000
##    800        0.9652             nan     0.1000    0.0000
```

```
##   2980        0.9003             nan     0.1000    0.0000
##   3000        0.9001             nan     0.1000    0.0000
```

```
forecast <- predict(rating_pred, validation)

summary(rating_pred)
```




```
##             var  rel.inf
## movieId movieId 84.24731
## userId   userId 15.75269
```

```
rmse_6 <- RMSE(forecast, validation$rating)

rmses <- bind_rows(rmses, data_frame(method = "GBM 3000 Trees", RMSE = rmse_6))
rmses %>% knitr::kable()
```

method	RMSE
Predict Average	1.0612018
Add Movie Effect	0.9439087
Add User Effect	0.8653488
Regularization	0.8648170
GBM Initial Grid	0.9990090
GBM 3000 Trees	0.9499079

4. Results

Below is the summary of RMSE results obtained from the models discussed in chapter 3.

method	RMSE
Predict Average	1.0612018
Add Movie Effect	0.9439087
Add User Effect	0.8653488
Regularization	0.8648170
GBM Initial Grid	0.9990090
GBM 3000 Trees	0.9499079

Modeling movie ratings considering both movie ID and user ID as input variables appears to be a good approach. Based on this approach we have compared Regularization and Gradient Boosting Machines with regards to their predictive power. Regularization delivers the best RMSE of 0.8648170. This delivers on the target of this assignment, which was to produce an RMSE below 0.86490.

5. Conclusion

Movie ratings have a strong movie effect and user effect. That means different movies attract different average ratings and different users give different ratings averages for the movies they rate. Therefore a modeling approach that considers both movie effect and user effect on ratings delivers good results.

Regularization delivers the best RMSE by applying a penalty term on movie rating averages that are the result of a small number of ratings. Different regularization coefficients have been evaluated with lambda = 5.2 delivering the best results. The RMSE on the Regularization model outperforms Gradient Boosting with 3000 Trees, 3-fold cross validation, an interaction depth of 9 and learning rate of 0.1.

For the comparison of alternative machine learning algorithms and grid search run times on a standard laptop have been found to be a major limiter. The Gradient Boosting model in chapter 3.6 requires 10 hours of run time on a standard laptop. Therefore a more comprehensive grid search on hyper parameters was not practical. A knn model was also attempted but had to be abandoned due to run times exceeding 24 hours. Therefore running the presented models on a server and doing a more comprehensive grid search is recommended and will likely further improve RMSEs.

Furthermore, we have seen in chapter 2 that there is also a time effect to movie ratings. Therefore it is possible that adding the year, month or week of ratings as a third input variable may further improve RMSEs.

The Regularization approach presented in this report produces an RMSE of 0.8648170, which meets the objective for this assignment.