MovieLens Reccomender System Project

HarvardX PH125.9x Data Science: Capstone

Shunsuke Kobayashi

final: 2021-12-01

Contents

1 Introduction

The goal of this project is to use the MovieLens dataset to create a recommender system.

The version of the Movielens dataset that we will use in this final project contains about 10 milion movie evaluations, divided into 9 milion for training (edx) and 1 milion for validation (validation). The training dataset contains 70,000 users and 10,500 different movies, divided into 20 genres such as Drama, Comedy, Action, and Romance.

We first perform data exploration to understand an overview of the model building process. The training data set is further divided into two parts and the resulting test set is used to evaluate the recommendation system. We select a model with a small Root Mean Squared Error (RMSE) and evaluate it using the validation test set, which is below the target of **0.8649**.

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

As a FINAL model, **the Matrix Factorization - recosystem** reached a RMSE of **0.7825**.

```
# Define Root Mean Squared Error (RMSE)
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

2 Data Preprocessing and Explanatory Data Analysis

2.1 Initial Data Preprocessing

The version of the movielens dataset we use contains about 10 milion movie evaluations, divided into 9 milion for training (edx) and 1 milion for validation (validation).

edx dataset contains 9,000,000 rows with 70,000 Users, 10,500 Movies and 797 Genres conbination. There is no missing values.

```
# Data Exploaration
head(edx) %>% kable()
```

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

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
```

```
## $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
## $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
## $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
## $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
## - attr(*, ".internal.selfref")=<externalptr>
```

```r
dim(edx)
```

```
## [1] 9000055       6
```

```r
sum(is.na(edx))
```

```
## [1] 0
```

```r
summary(edx) %>% kable()
```

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

```r
edx%>% summarize(n_users = n_distinct(userId),
                 n_movies = n_distinct(movieId),
                 n_genres = n_distinct(genres)) %>% kable()
```

n_users	n_movies	n_genres
69878	10677	797

The data is difficult to process because of the timestamp, which is difficult for humans to understand, and the concatenation of the title and the year of airing. We perform data formatting. The training set is further divided into 9:1 and a test set is prepared for use in the modeling process.

```r
# Convert timestamp to a human readable date
edx$date <- as.POSIXct(edx$timestamp, origin="1970-01-01")
validation$date <- as.POSIXct(validation$timestamp, origin="1970-01-01")

# Extract the year of Rate in both data sets

edx$year_Rate <- as.integer(format(edx$date,"%Y"))
validation$year_Rate <- as.integer(format(validation$date,"%Y"))
```

```r
# Extract the year of release for each movie in both data set
# edx dataset
edx <- edx %>%
  mutate(title = str_trim(title)) %>%
  extract(title,
          c("titleTemp", "release"),
          regex = "^(.*) \\(([0-9 \\-]*)\\)$",
          remove = F) %>%
  mutate(release = if_else(str_length(release) > 4,
                           as.integer(str_split(release, "-",
                                                 simplify = T)[1]),
                           as.integer(release))
  ) %>%
  mutate(title = if_else(is.na(titleTemp),
                         title,
                         titleTemp)
  ) %>%
  select(-titleTemp)

# validation data set
validation <- validation %>%
  mutate(title = str_trim(title)) %>%
  extract(title,
          c("titleTemp", "release"),
          regex = "^(.*) \\(([0-9 \\-]*)\\)$",
          remove = F) %>%
  mutate(release = if_else(str_length(release) > 4,
                           as.integer(str_split(release, "-",
                                                 simplify = T)[1]),
                           as.integer(release))
  ) %>%
  mutate(title = if_else(is.na(titleTemp),
                         title,
                         titleTemp)
  ) %>%
  select(-titleTemp)

# Preparation test and train data set for model selection
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]


# Make sure userId and movieId in test set are also in train set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
```

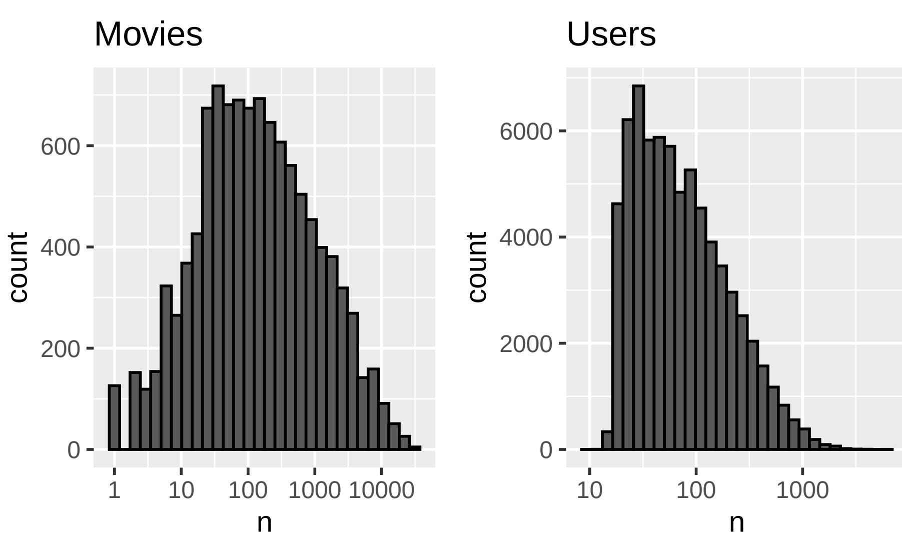
```
rm(test_index, temp, removed)
```

2.2 Explanatory Data Analysis (EDA)

2.2.1 Whole count distribution

According to the histograms, we can see that the distributions are not uniform respectively.

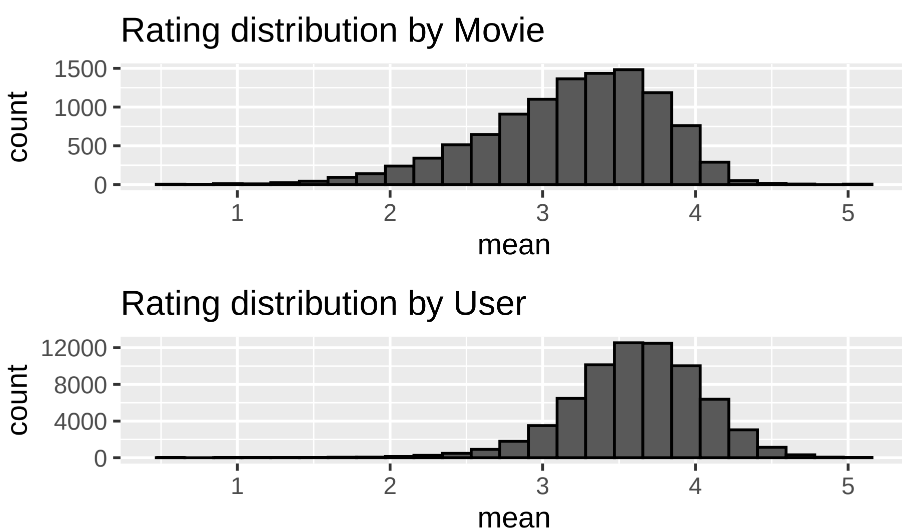
```
# Whole count distribution
p_c1 <- edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")

p_c2 <- edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users")

# compare count distribution by movie and user
p_c1 + p_c2
```



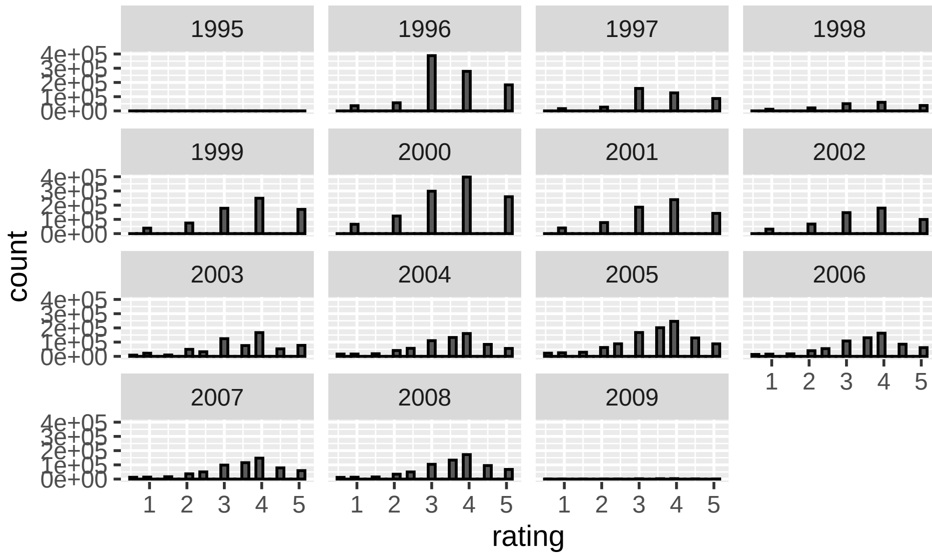
2.2.2 Whole rating distribution

The graph below shows the distribution of the average Rating by Movie and by User. The rating distribution by year shows that before 2002, the rating was based on integer values only, while after 2003, the rating method of half stars was added. The year of Rate is considered in the model construction.

```
# Whole rating distribution
p_d1 <- edx %>% group_by(movieId) %>%
  summarise(mean = mean(rating)) %>%
  ggplot(aes(mean)) + geom_histogram(bins = 25,col="black") +
  ggtitle("Rating distribution by Movie")

p_d2 <- edx %>% group_by(userId) %>%
  summarise(mean = mean(rating)) %>%
  ggplot(aes(mean)) + geom_histogram(bins = 25,col="black") +
  ggtitle("Rating distribution by User")

# compare rating distribution by movie and user
p_d1 / p_d2
```




```
# rating distribution by year of Rating
edx %>%  ggplot(aes(rating)) +
  geom_histogram(bins = 25,col="black") +
  facet_wrap(~year_Rate)
```



2.2.3 Genres

There are 797 genres, but they are composed of multiple combinations. Each of them is sorted into rows to check the overall number and relationship.

We used a ranking system to check whether the year of evaluation would affect the confirmed genres. The 20 genres were found to show generally the same ranking trends throughout the period of the data set.

```
# Genres
edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  head()
```

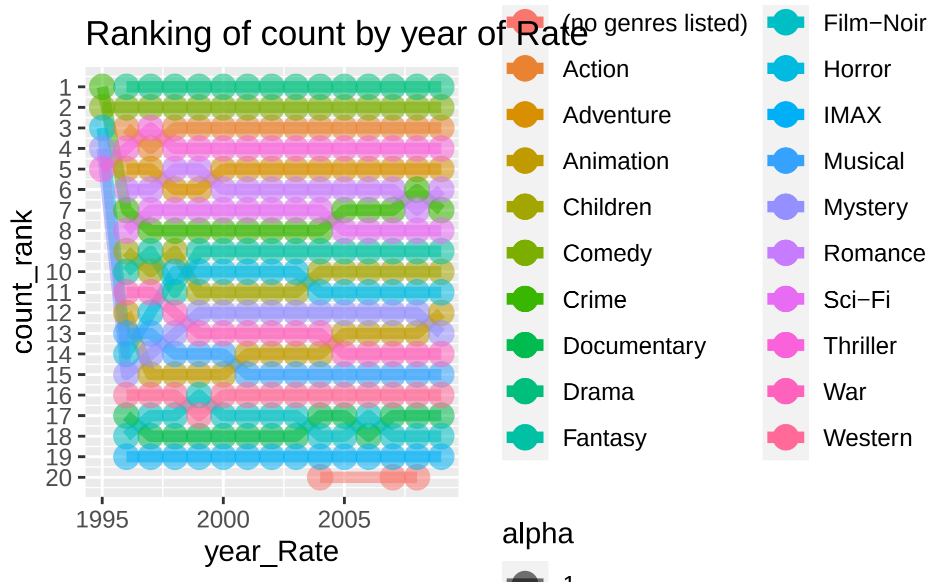
```
## # A tibble: 6 x 2
##   genres                                              n
##   <chr>                                           <int>
## 1 (no genres listed)                                  7
## 2 Action                                          24482
## 3 Action|Adventure                                68688
## 4 Action|Adventure|Animation|Children|Comedy       7467
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy   187
## 6 Action|Adventure|Animation|Children|Comedy|IMAX      66
```

```
# Separate genres and rating distribution by genres
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  select(genres, rating) %>%
  group_by(genres) %>%
  summarize(count = n(), mean = mean(rating)) %>% kable()
```

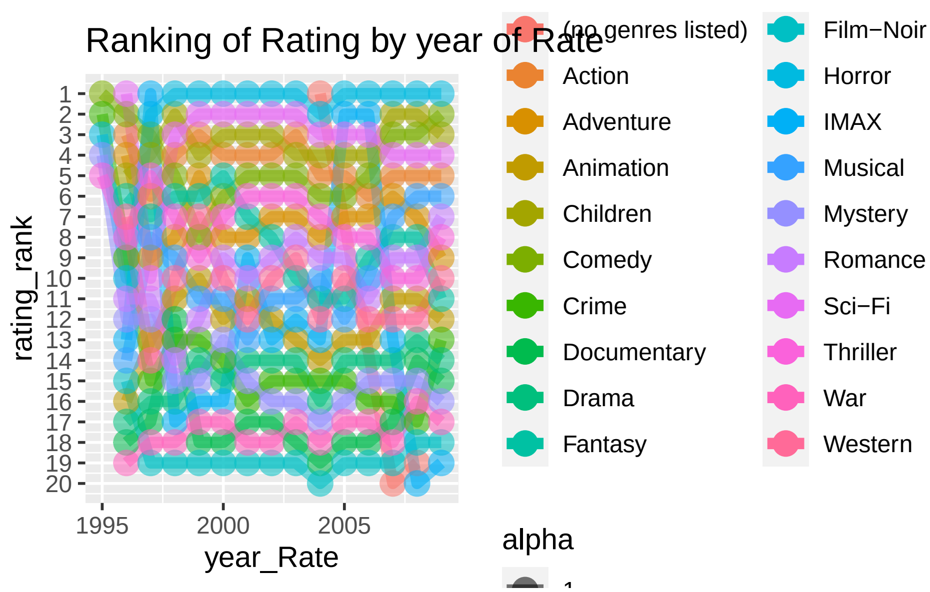
genres	count	mean
(no genres listed)	7	3.642857

genres	count	mean
Action	2560545	3.421405
Adventure	1908892	3.493544
Animation	467168	3.600644
Children	737994	3.418715
Comedy	3540930	3.436908
Crime	1327715	3.665925
Documentary	93066	3.783487
Drama	3910127	3.673131
Fantasy	925637	3.501946
Film-Noir	118541	4.011625
Horror	691485	3.269815
IMAX	8181	3.767693
Musical	433080	3.563305
Mystery	568332	3.677001
Romance	1712100	3.553813
Sci-Fi	1341183	3.395743
Thriller	2325899	3.507676
War	511147	3.780813
Western	189394	3.555918

```r
# Identify trends in the time period being evaluated focusing ranking.
genresByYear <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  select(movieId, year_Rate, genres, rating) %>%
  group_by(year_Rate, genres) %>%
  filter(!is.na(rating)) %>%
  summarise(count = n(),
            rating_avg = mean(rating)) %>%
  mutate(count_rank = row_number(desc(count)),
         rating_rank = row_number(rating_avg))

# Create the graph count ranking
p_genre1 <- genresByYear %>% #filter(year_Rate>=1993) %>%
  ggplot(aes(x = year_Rate, y= count_rank, group=genres))+
  geom_line(aes(color = genres, alpha = 1), size = 2) +
  geom_point(aes(color = genres, alpha = 1), size = 4) +
  scale_y_reverse(breaks = 1:nrow(genresByYear)) +
  guides(color=guide_legend(ncol=2)) +
  ggtitle("Ranking of count by year of Rate")
p_genre1
```




```r
# Create the graph rating ranking
p_genre2 <- genresByYear %>% #filter(year_Rate>=1993) %>%
  ggplot(aes(x = year_Rate, y= rating_rank, group=genres))+
  geom_line(aes(color = genres, alpha = 1), size = 2) +
  geom_point(aes(color = genres, alpha = 1), size = 4) +
  scale_y_reverse(breaks = 1:nrow(genresByYear)) +
  guides(color=guide_legend(ncol=2)) +
  ggtitle("Ranking of Rating by year of Rate")
p_genre2
```




```r
# Extract the genres name
genre_name <- unique(genresByYear$genres)
```

3 Methods and Analysis

3.1 Regression Models and Regularized Models

In order to build a linear model, we start from a simple mean model and add each variable identified in the previous chapter to build the models.

3.1.1 Mean model

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

We start by building the simplest possible recommendation system, mean model. A model that assumes the same rating ($\hat{\mu}$) for all movies and users with all the differences explained by random variation($\epsilon_{u,i}$). $\hat{\mu}$ can be calculated as the average whole rating value.

3.1.2 Movie effects

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

We can extend the previous model by adding terms, b_i. It represents the movie effect of movie_i. As an approximation ($\hat{b}_i$) can be calculated as the average value by **movieId** after subtracting the average value from each Rating.

3.1.3 User effects

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

As we saw in the previous chapter, there are differences in Rating for each user. We can extend the model by incorporating differences between users. Since it is difficult to calculate with `lm()` function due to the environment of computing resources, we compute an approximation by computing $\hat{\mu}$ and $\hat{b}_i$ and estimating $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$.

3.1.4 year of Rate effects

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

We extend the model with the yearly impact of Rating. We use $f(year_{u,i})$ to more accurately represent the year-to-year impact. However, there is a major change in the evaluation methodology as identified in the previous chapter. To simplify the model, we express the impact in years as b_y, and as in previous models, we compute an approximation by computing $\hat{\mu}$, $\hat{b}_i$ and $\hat{b}_u$ and estimating $\hat{b}_y$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u$.

This implies that a further improvement to our model may be:

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

3.1.5 Genre Effect

$$Y_{u,i} = \mu + b_i + b_u + b_y + \sum_{k=1}^{K} x_{u,i}^k \beta_k + \epsilon_{u,i}, \quad with\ x_{u,i}^k = 1 \quad if\ g_{u,i}\ is\ genre\ k$$

We extend the model with the Genre effect. We use $\sum_{k=1}^{K} x_{u,i}^k \beta_k$ to more accurately represent the each genre impact. As we saw in the previous chapters, there are 20 genres. On the other hand, the number of genre combinations is approximately 800, which is sufficiently small compared to the number of 10500 movies and

9 million total data. As in the previous model, in order to simplify the model, we represent the effect of genre by b_g and perform the calculation as the combination of genres. We compute an approximation by computing $\hat{\mu}$, $\hat{b}_i$, $\hat{b}_u$ and $\hat{b}_y$ and estimating $\hat{b}_g$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_y$.

This implies that a further improvement to our model may be:

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

Note: $\sum_{k=1}^{K} x_{u,i}^k \beta_k$ incorporated into the modeling in the Gradient Boosting section below.

3.1.6 Regularized Models

In order to avoid over training, we consider minimizing the model equation by adding a penalty (λ) to the models we have created. We consider minimization for Movie+User effect model and 4 effect model.

This implies that a further improvement to our model may be:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda\left(\sum_{i} b_i^2 + \sum_{u} b_u^2\right)$$

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_y - b_g)^2 + \lambda\left(\sum_{i} b_i^2 + \sum_{u} b_u^2 + \sum_{u,i} b_y^2 + \sum_{u,i} b_g^2\right)$$

3.2 Matrix-factorization

The main function of the recommender system is to predict the unknown values of the evaluation matrix based on the observed values. We consider m users and n items. The users will be represented by an n-dimensional vector, which we aim to transform by dimensionality reduction to k dimensions where $m > k > 0$. This can be approximated as follows by considering a kÖm matrix P and a kÖn matrix Q representing the user elements for an mÖn matrix R representing the evaluation values.

$$R \approx P^T Q$$

As a method to perform matrix factorization using R, we use A Matrix-factorization Library for Recommender Systems, LIBMF, developed and published by W.-S. Chin et al. The evaluation value of item v evaluated by user u can be expressed as p_u,q_v. Matrix Factorization is to learn p_u and q_v for each user and each item from the known evaluation values. P and Q that satisfy the following equation are derived from the training data by using the library.

$$min_{P,Q} \sum_{(u,v)\in R} ((r_{u,v} - \boldsymbol{p}_u^T \boldsymbol{q}_i)^2 + \lambda_P ||\boldsymbol{p}_u||^2 + \lambda_Q ||\boldsymbol{p}_v||^2$$

where $||.||$ is the Euclidean norm, $(u, v) \in R$ indicates that rating $r_{u,v}$ is available, λ_P and λ_Q are regularization coefficients for avoiding over-fitting.

3.3 Gradient Boosting

We consider the simplification of $\sum_{k=1}^{K} x_{u,i}^k \beta_k$ when building the regression models. In order to examine the impact of each genre, the matrix is extended to include whether or not the corresponding movie has 20 genres. Since it has a huge dimension with 20 additional columns, we model it using XGBoost, one of the Gradient Boosting commonly used in machine learning of table data. In addition to building a recommendation model for this Project, we also build a model excluding user and/or movie data for new users and/or movies.

Gradient boosting is a machine learning method for tasks such as regression and classification that generates a predictive model in the form of an ensemble of weak prediction models (usually decision trees).

4 Results

4.1 Model Selection

We builded each model using the Train and Test datasets created by splitting the edx dataset. We selected the model that achieves the target RMSE, and finally validated it using the Validation set.

```
# Creating the Target
result <- data.frame(Method = "Target", RMSE = 0.8649)
```

4.1.1 Regression Models and Regularized Models

4.1.1.1 Mean model

The mean value alone showed a large RMSE.

```
# Initial Prediction-----
# Mean of observed values
mu <- mean(train_set$rating)

# Update the result table
result <- bind_rows(result, data.frame(Method = "Mean",
                                       RMSE = RMSE(test_set$rating, mu)))

# Show the RMSE
result %>% kable()
```

Method	RMSE
Target	0.864900
Mean	1.060054

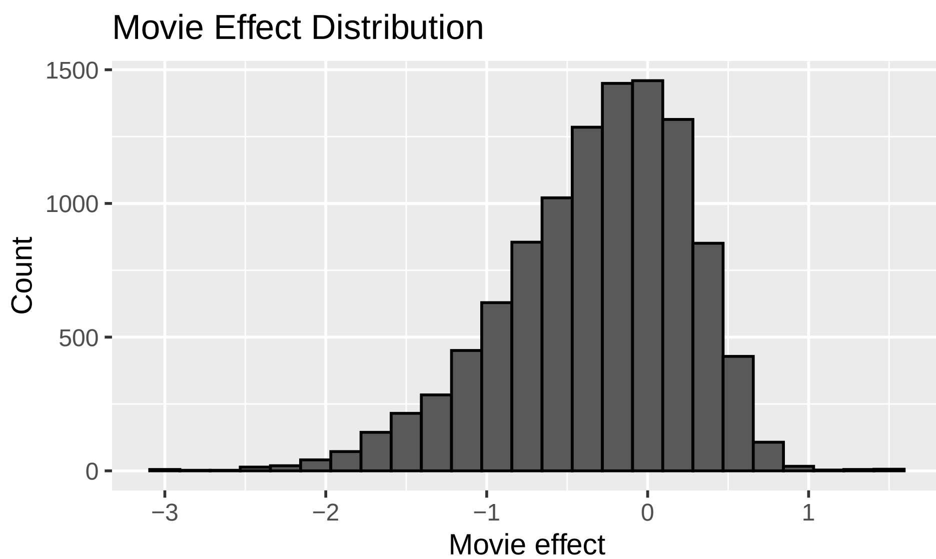
4.1.1.2 Movie effects

Improvements were seen by adding movie effects.

```
# Add Movie Effect(bi) -----
# Movie effects (bi)
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
head(bi) %>% kable()
```

movieId	b_i
1	0.4150040
2	-0.3064057
3	-0.3613952
4	-0.6372808
5	-0.4416058
6	0.3018943

```r
# Confirm the Movie effects distribution
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=25, col = I("black")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
  ylab("Count")
```



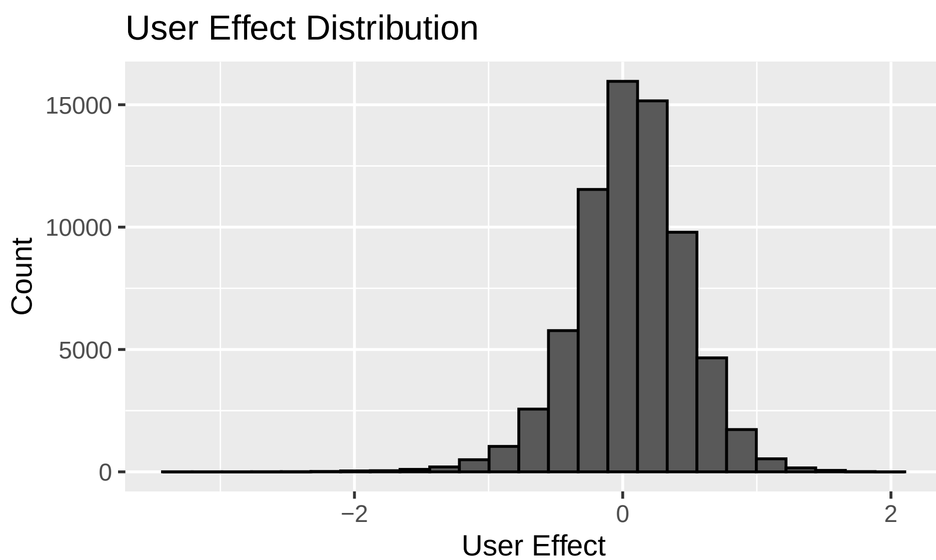

```r
# Predict the rating with mean + bi
y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>% pull(b_i)


# Calculate the RMSE and update the result
result <- bind_rows(result,
                    data.frame(Method = "Mean + bi",
                               RMSE = RMSE(test_set$rating, y_hat_bi)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615

4.1.1.3 User effects Further improvement was seen by adding user-specific effects.

```r
# Add User Effect(bu) ------
# User effect (bu)
bu <- train_set %>%
  left_join(bi, by = "movieId") %>%
```

```r
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Confirm the User effects distribution
bu %>% # filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins=25,color = "black") +
  ggtitle("User Effect Distribution") +
  xlab("User Effect") +
  ylab("Count")
```

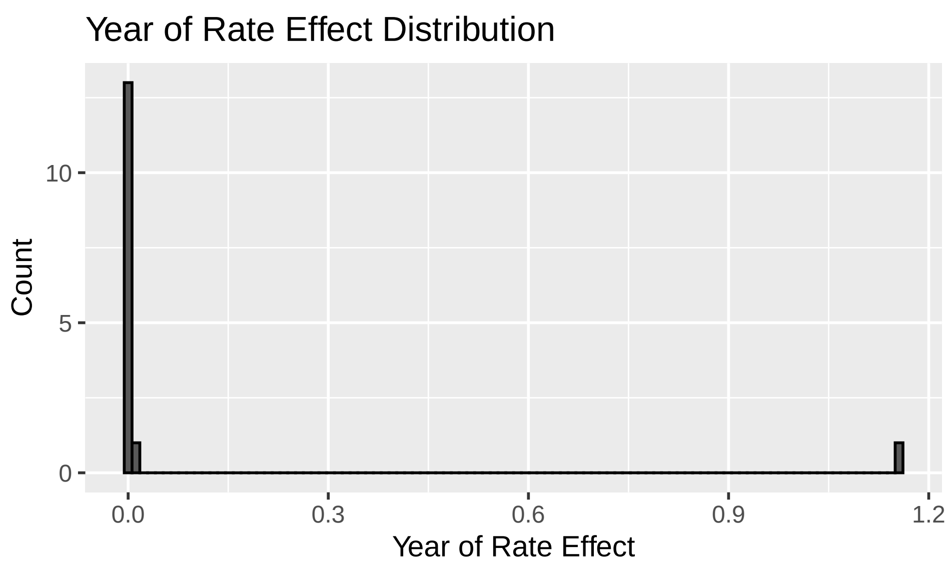



```r
# Prediction
y_hat_bi_bu <- test_set %>%
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  mutate(pred = mu + b_i + b_u) %>% pull(pred)

# Calculate the RMSE and update result
result <- bind_rows(result,
                    data.frame(Method = "Mean + bi + bu",
                               RMSE = RMSE(test_set$rating, y_hat_bi_bu)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843

4.1.1.4 year of Rate effects The addition of the RATE year showed little improvement.

```
# Add year of Rate Effect(by) ------
# year of Rate effect (by)
by <- train_set %>%
  left_join(bi, by = "movieId")%>%
  left_join(bu, by="userId") %>%
  group_by(year_Rate) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))

# Confirm the year of Rate effects distribution
by %>%
  ggplot(aes(b_y)) +
  geom_histogram(bins=100,color = "black") +
  ggtitle("Year of Rate Effect Distribution") +
  xlab("Year of Rate Effect") +
  ylab("Count")
```

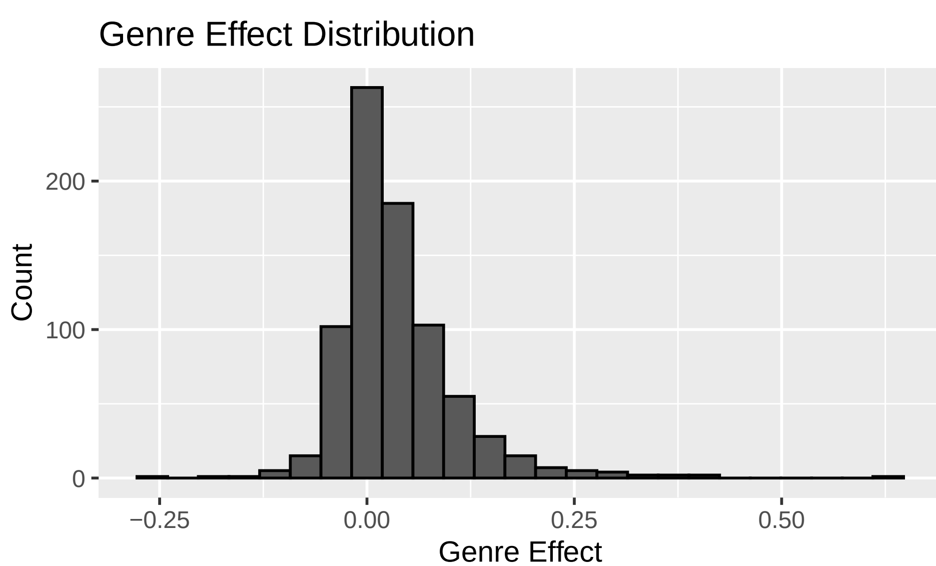



```
# Prediction
y_hat_bi_bu_by <- test_set %>%
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  left_join(by, by="year_Rate") %>%
  mutate(pred = mu + b_i + b_u + b_y) %>% pull(pred)

# Calculate the RMSE and update result
result <- bind_rows(result,
                    data.frame(Method = "Mean + bi + bu + by",
                               RMSE = RMSE(test_set$rating, y_hat_bi_bu_by)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811

4.1.1.5 Genre Effect Even a simple addition of a genre showed some improvement.

```
# Add Genre Effect(bg) ------
# Genre Rate effect (bg)
bg <- train_set %>%
  left_join(bi, by = "movieId")%>%
  left_join(bu, by="userId") %>%
  left_join(by, by="year_Rate") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i- b_u - b_y))

# Confirm the Genre effects distribution
bg %>%
  ggplot(aes(b_g)) +
  geom_histogram(bins=25,color = "black") +
  ggtitle("Genre Effect Distribution") +
  xlab("Genre Effect") +
  ylab("Count")
```



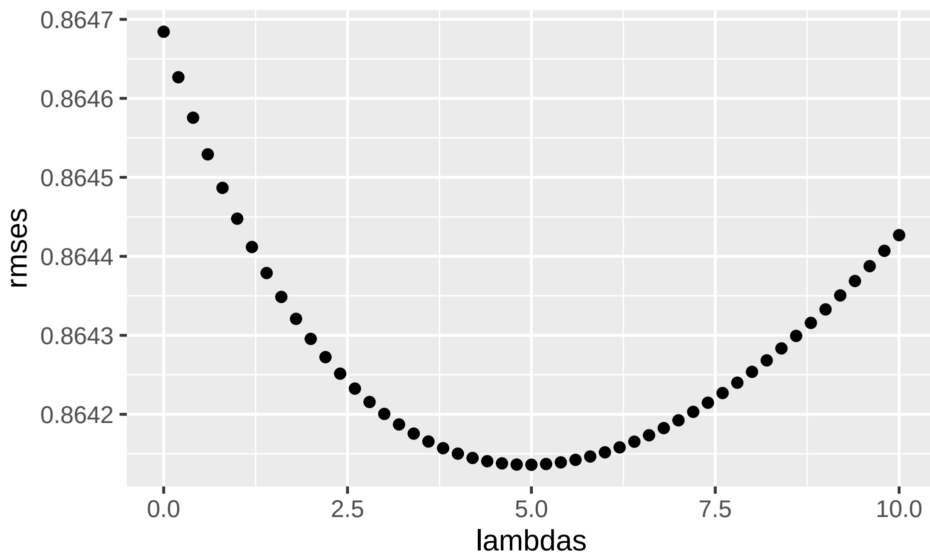

```
# Prediction
y_hat_bi_bu_by_bg <- test_set %>%
  left_join(bi, by="movieId") %>%
  left_join(bu, by="userId") %>%
  left_join(by, by="year_Rate") %>%
  left_join(bg, by="genres") %>%
```

```
  mutate(pred = mu + b_i + b_u + b_y + b_g) %>% pull(pred)

# Calculate the RMSE and update result
result <- bind_rows(result,
                    data.frame(Method = "Mean + bi + bu + by + bg",
                               RMSE = RMSE(test_set$rating, y_hat_bi_bu_by_bg)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811
Mean + bi + bu + by + bg	0.8643201

4.1.1.6 Regularized Models By regularizing each model, further improvement over the original model was observed. The obtained lambda value ($\lambda = 5$) was used for validation.

```
# Regularization----
# Movie + User (Same methods the course)
lambdas <- seq(0, 10, 0.2)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  bi <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  bu <- train_set %>%
    left_join(bi, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

# Check the lambdas
qplot(lambdas,rmses)
```




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

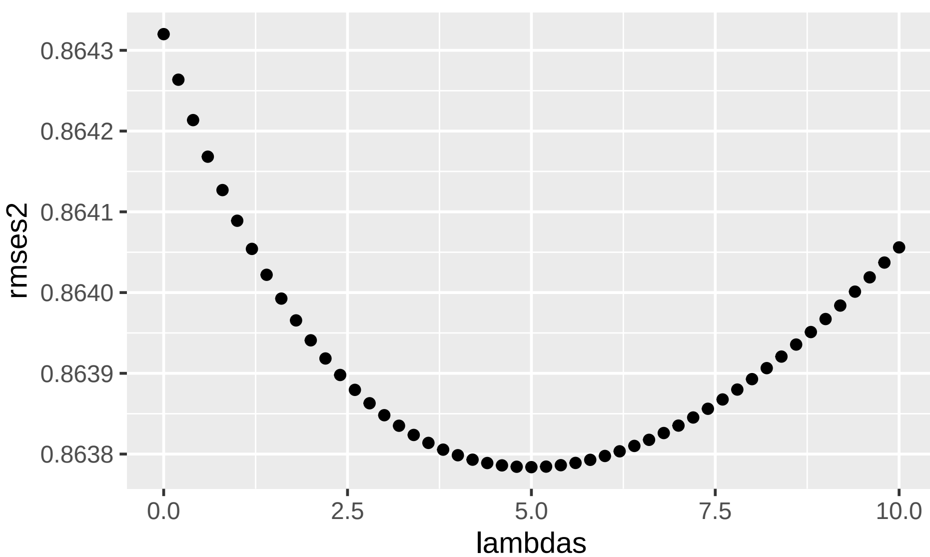
```
## [1] 5
```

```
# Calculate the RMSE and update result
result <- bind_rows(result,
                    data.frame(Method = "Regularized Movie + User Effect",
                               RMSE = min(rmses)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811
Mean + bi + bu + by + bg	0.8643201
Regularized Movie + User Effect	0.8641362

```
# Movie + User + Year of Rate + Genre (new one, regularization with 4 biases)
lambdas <- seq(0, 10, 0.2)
rmses2 <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  bi <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  bu <- train_set %>%
    left_join(bi, by = "movieId") %>%
    group_by(userId) %>%
```

```
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
  by <- train_set %>%
    left_join(bi, by = "movieId")%>%
    left_join(bu, by="userId") %>%
    group_by(year_Rate) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))
  bg <- train_set %>%
    left_join(bi, by = "movieId")%>%
    left_join(bu, by="userId") %>%
    left_join(by, by="year_Rate") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    left_join(by, by="year_Rate") %>%
    left_join(bg, by="genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

# Check the lambdas
qplot(lambdas,rmses2)
```




```
lambda <- lambdas[which.min(rmses2)]
lambda
```

```
## [1] 5
```

```
# Calculate the RMSE and update result
result <- bind_rows(result,
```

```
                    data.frame(Method = "Regularized Movie + User + Year of Rate + Genre Effect",
                               RMSE = min(rmses2)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811
Mean + bi + bu + by + bg	0.8643201
Regularized Movie + User Effect	0.8641362
Regularized Movie + User + Year of Rate + Genre Effect	0.8637838

4.1.2 Matrix-factorization

Significant improvement was observed by using the recosystem library.

```
# Matrix Factorization-----
set.seed(1, sample.kind = "Rounding")

# Convert "train" and "test" sets to recosystem input format
train_reco <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating = rating))
test_reco  <-  with(test_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))

# Create the model object
reco <-  recosystem::Reco()

# Tune the parameters
opts <-  reco$tune(train_reco, opts = list(dim = c(10, 20, 30),
                                           lrate = c(0.1, 0.2),
                                           costp_l2 = c(0.01, 0.1),
                                           costq_l2 = c(0.01, 0.1),
                                           nthread  = nc, niter = 10))

# Train the model
reco$train(train_reco, opts = c(opts$min, nthread = nc, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9828   1.1036e+07
##    1       0.8761   8.9945e+06
##    2       0.8432   8.3549e+06
##    3       0.8203   7.9576e+06
##    4       0.8035   7.6873e+06
##    5       0.7910   7.4922e+06
```

```
##     6        0.7806   7.3497e+06
##     7        0.7720   7.2297e+06
##     8        0.7648   7.1367e+06
##     9        0.7585   7.0564e+06
##    10        0.7532   6.9938e+06
##    11        0.7482   6.9367e+06
##    12        0.7440   6.8875e+06
##    13        0.7402   6.8453e+06
##    14        0.7366   6.8087e+06
##    15        0.7331   6.7741e+06
##    16        0.7302   6.7444e+06
##    17        0.7274   6.7189e+06
##    18        0.7247   6.6934e+06
##    19        0.7223   6.6715e+06
```

```
# Calculate the prediction
y_hat_reco <-  reco$predict(test_reco, out_memory())

# Update the result table
result <- bind_rows(result,
                    data.frame(Method = "Matrix Factorization - recosystem",
                               RMSE = RMSE(test_set$rating, y_hat_reco)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811
Mean + bi + bu + by + bg	0.8643201
Regularized Movie + User Effect	0.8641362
Regularized Movie + User + Year of Rate + Genre Effect	0.8637838
Matrix Factorization - recosystem	0.7860689

4.2 Validation

Through the analysis using the training set, the Matrix factorization model that showed the best results was selected as the final model and evaluated using the Validation set. As a FINAL model, **the Matrix Factorization - recosystem** reached a RMSE of **0.7825**.

```
# Final Validation-------
set.seed(1, sample.kind = "Rounding")

# Convert "edx" and "validation" sets to recosystem input format
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))
valid_reco  <-  with(validation, data_memory(user_index = userId,
```

```r
                                                 item_index = movieId,
                                                 rating = rating))

# Create the model object
reco_final <-  recosystem::Reco()

# Tune the parameters
opts_final <-  reco_final$tune(edx_reco, opts = list(dim = c(10, 20, 30),
                                                      lrate = c(0.1, 0.2),
                                                      costp_l2 = c(0.01, 0.1),
                                                      costq_l2 = c(0.01, 0.1),
                                                      nthread  = nc, niter = 10))

# Train the model
reco_final$train(edx_reco, opts = c(opts_final$min, nthread = nc, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9730   1.2016e+07
##    1       0.8718   9.8723e+06
##    2       0.8382   9.1711e+06
##    3       0.8164   8.7462e+06
##    4       0.8011   8.4686e+06
##    5       0.7896   8.2776e+06
##    6       0.7797   8.1203e+06
##    7       0.7713   8.0002e+06
##    8       0.7642   7.9010e+06
##    9       0.7581   7.8212e+06
##   10       0.7529   7.7515e+06
##   11       0.7482   7.6922e+06
##   12       0.7441   7.6425e+06
##   13       0.7405   7.6007e+06
##   14       0.7371   7.5618e+06
##   15       0.7340   7.5279e+06
##   16       0.7312   7.4978e+06
##   17       0.7285   7.4682e+06
##   18       0.7261   7.4447e+06
##   19       0.7239   7.4230e+06
```

```r
# Calculate the prediction
y_hat_final_reco <-  reco_final$predict(valid_reco, out_memory())

# Update the result table
result <- bind_rows(result,
                    data.frame(Method = "[FINAL model] Matrix Factorization - recosystem",
                               RMSE = RMSE(validation$rating, y_hat_final_reco)))

# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000

Method	RMSE
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811
Mean + bi + bu + by + bg	0.8643201
Regularized Movie + User Effect	0.8641362
Regularized Movie + User + Year of Rate + Genre Effect	0.8637838
Matrix Factorization - recosystem	0.7860689
[FINAL model] Matrix Factorization - recosystem	0.7824372

As a reference, the obtained lambda was used to validate the Regularized 4 Effects model. We also achieved our target of 0.8649.

```
# Reference: What is the RMSE of regularized 4 effects
# With edx and validation data sets
# Movie + User + Year of Rate + Genre (new one, regularization with 4 biases)
lambda <- 5

# Attention: Validation data set cannot be used for select the model
rmse_v <- sapply(lambda, function(l){
  mu <- mean(edx$rating)
  bi <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  bu <- edx %>%
    left_join(bi, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
  by <- edx %>%
    left_join(bi, by = "movieId")%>%
    left_join(bu, by="userId") %>%
    group_by(year_Rate) %>%
    summarize(b_y = sum(rating - mu - b_i - b_u)/(n()+l))
  bg <- edx %>%
    left_join(bi, by = "movieId")%>%
    left_join(bu, by="userId") %>%
    left_join(by, by="year_Rate") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u - b_y)/(n()+l))
  predicted_ratings <-
    validation %>%
    left_join(bi, by = "movieId") %>%
    left_join(bu, by = "userId") %>%
    left_join(by, by="year_Rate") %>%
    left_join(bg, by="genres") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_g) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})


# Update result
```

```
result <- bind_rows(result,
                    data.frame(Method = "[Reference] Regularized 4 Effects with Lambda=5", RMSE = rmse_
# Show the RMSE improvement
result %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0600537
Mean + bi	0.9429615
Mean + bi + bu	0.8646843
Mean + bi + bu + by	0.8646811
Mean + bi + bu + by + bg	0.8643201
Regularized Movie + User Effect	0.8641362
Regularized Movie + User + Year of Rate + Genre Effect	0.8637838
Matrix Factorization - recosystem	0.7860689
[FINAL model] Matrix Factorization - recosystem	0.7824372
Reference Regularized 4 Effects with Lambda=5	0.8644116

4.3 Additional: Gradient Boosting

We performed the analysis using XGBoost, but due to the performance of the PC (amount of memory), we could not perform the desired analysis with the edx data set. Therefore, a smaller dataset, Movielens100K, was used for the analysis.

The same model construction as for the edx set was used for each analysis. Due to the small amount of data, not much improvement was observed in the recosystem, but good results were obtained in XGBoost. Also, although we did not get good scores for new users, we did get some scores for new movies.

```
# For Discussion
data("movielens")

# Preparation test and train data set
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
train_small <- movielens[-test_index,]
temp <- movielens[test_index,]


# Make sure userId and movieId in test set are also in train set
test_small <- temp %>%
  semi_join(train_small, by = "movieId") %>%
  semi_join(train_small, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test_small)
train_small <- rbind(train_small, removed)

rm(test_index, temp, removed)

# To compare, targeted same value: 0.8649
```

```r
result_small <- data.frame(Method = "Target", RMSE = 0.8649)

# Initial Prediction-----
# Mean of observed values
mu <- mean(train_small$rating)

# Update the result_small
result_small <- bind_rows(result_small, data.frame(Method = "Mean",
                                                   RMSE = RMSE(test_small$rating, mu)))

# Show the RMSE
result_small %>% kable()
```

Method	RMSE
Target	0.864900
Mean	1.054766

```r
# Add Movie Effect(bi) -----
# Movie effects (bi)
bi <- train_small %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict the rating with mean + bi
y_hat_bi <- mu + test_small %>%
  left_join(bi, by = "movieId") %>% pull(b_i)


# Calculate the RMSE and update result_small
result_small <- bind_rows(result_small,
                          data.frame(Method = "Mean + bi",
                                     RMSE = RMSE(test_small$rating, y_hat_bi)))

# Show the RMSE improvement
result_small %>% kable()
```

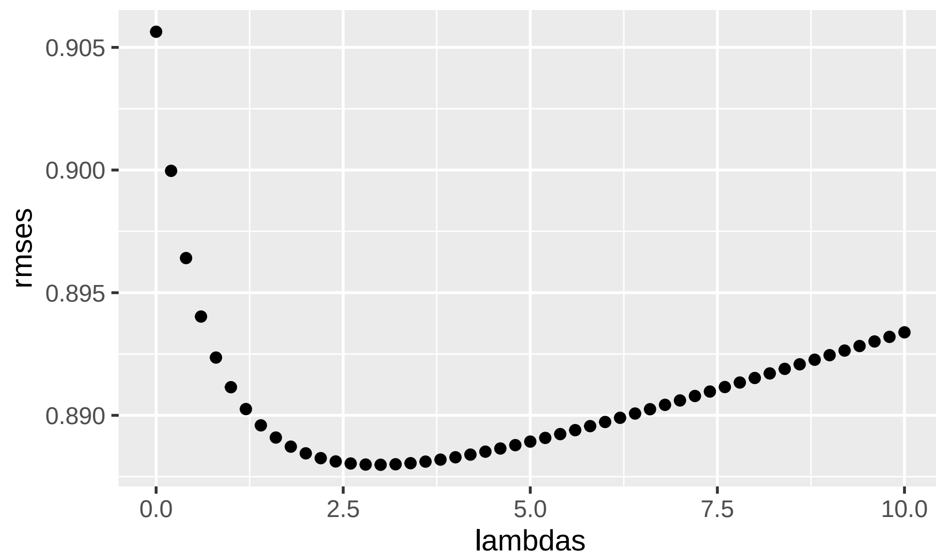
Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457

```r
# Add User Effect(bu) ------
# User effect (bu)
bu <- train_small %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Prediction
```

```r
y_hat_bi_bu <- test_small %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>% pull(pred)

# Calculate the RMSE and update result_small
result_small <- bind_rows(result_small,
                          data.frame(Method = "Mean + bi + bu",
                                     RMSE = RMSE(test_small$rating, y_hat_bi_bu)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371

```r
# Regularization----
lambdas <- seq(0, 10, 0.2)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_small$rating)
  b_i <- train_small %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_small %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test_small %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, test_small$rating))
})

qplot(lambdas,rmses)
```




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

```
## [1] 3
```

```
# Calculate the RMSE and update result_small
result_small <- bind_rows(result_small,
                          data.frame(Method = "Regularized Movie + User Effect",
                                     RMSE = min(rmses)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371
Regularized Movie + User Effect	0.8879858

```
# Matrix Factorization-----
set.seed(1, sample.kind = "Rounding")

# Convert 'train_small' and 'test_small' sets to recosystem input format
train_reco <-  with(train_small, data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating = rating))
test_reco  <-  with(test_small, data_memory(user_index = userId,
                                            item_index = movieId,
                                            rating = rating))

# Create the model object
```

```r
reco <-  recosystem::Reco()

# Tune the parameters
opts <-  reco$tune(train_reco, opts = list(dim = c(10, 20, 30),
                                           lrate = c(0.1, 0.2),
                                           costp_l2 = c(0.01, 0.1),
                                           costq_l2 = c(0.01, 0.1),
                                           nthread  = nc, niter = 10)) # nc is depended on PC

# Train the model
reco$train(train_reco, opts = c(opts$min, nthread = nc, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       1.3033   2.2591e+05
##    1       0.9283   1.4616e+05
##    2       0.8957   1.3873e+05
##    3       0.8724   1.3391e+05
##    4       0.8477   1.2953e+05
##    5       0.8198   1.2513e+05
##    6       0.7900   1.2104e+05
##    7       0.7632   1.1743e+05
##    8       0.7388   1.1459e+05
##    9       0.7146   1.1172e+05
##   10       0.6952   1.0966e+05
##   11       0.6770   1.0765e+05
##   12       0.6604   1.0595e+05
##   13       0.6468   1.0475e+05
##   14       0.6333   1.0344e+05
##   15       0.6217   1.0230e+05
##   16       0.6123   1.0155e+05
##   17       0.6041   1.0084e+05
##   18       0.5957   1.0010e+05
##   19       0.5873   9.9320e+04
```

```r
# Calculate the prediction
y_hat_final_reco <-  reco$predict(test_reco, out_memory())

# Update the result_small table
result_small <- bind_rows(result_small,
                          data.frame(Method = "Matrix Factorization - recosystem",
                                     RMSE = RMSE(test_small$rating, y_hat_final_reco)))

# Show the RMSE improvement
result_small
```

```
##                               Method      RMSE
## 1                             Target 0.8649000
## 2                               Mean 1.0547663
## 3                          Mean + bi 0.9860457
## 4                     Mean + bi + bu 0.9056371
## 5   Regularized Movie + User Effect 0.8879858
## 6 Matrix Factorization - recosystem 0.9307754
```

```r
# Xgboost------
# genres is same as edx data set
genre_name
```

```
##  [1] "Comedy"             "Crime"              "Horror"
##  [4] "Mystery"            "Thriller"           "Action"
##  [7] "Adventure"          "Animation"          "Children"
## [10] "Documentary"        "Drama"              "Fantasy"
## [13] "Film-Noir"          "IMAX"               "Musical"
## [16] "Romance"            "Sci-Fi"             "War"
## [19] "Western"            "(no genres listed)"
```

```r
# Copy the modify data sets
train_small2 <- train_small
test_small2 <- test_small

# To use Xgboost, modify genres such like one-hot encoding(but not same)
for (n in genre_name){
  train_small2 <- train_small2 %>%
    mutate(!!n := if_else(str_detect(genres,n),1,0))
}
for (n in genre_name){
  test_small2 <- test_small2 %>%
    mutate(!!n := if_else(str_detect(genres,n),1,0))
}


train_small2 %>% select(-c(title, genres)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> train_small2

test_small2 %>% select(-c(title, genres)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> test_small2

# Note: it took 45 minutes!!
# Xgboost Linear model
set.seed(1, sample.kind = "Rounding")
system.time(
  modelXgboostLinear <- train(
    rating ~ .,
    data = train_small2,
    method = "xgbLinear",
    preProcess = c('center', 'scale'),
    trControl = trainControl(method = "cv"),
    tuneLength = 4)
)
```

```
##    user  system elapsed
##   41.87    1.17 2667.94
```

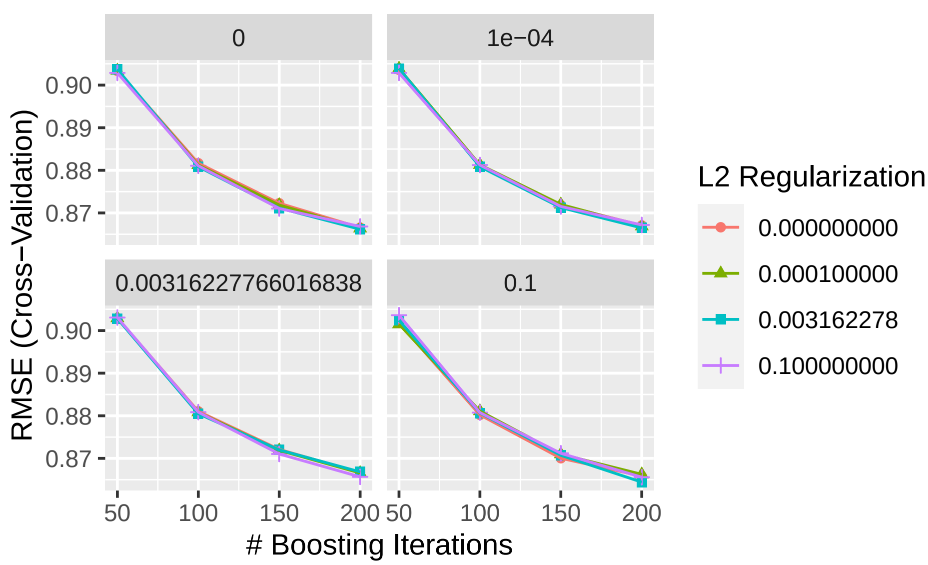
```r
print(modelXgboostLinear)
```

```
## eXtreme Gradient Boosting
```

```
##
## 90324 samples
##    24 predictor
##
## Pre-processing: centered (24), scaled (24)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 81292, 81291, 81291, 81291, 81292, 81292, ...
## Resampling results across tuning parameters:
##
##   lambda       alpha        nrounds  RMSE       Rsquared   MAE
##   0.000000000  0.000000000   50      0.9033259  0.2773919  0.7030091
##   0.000000000  0.000000000  100      0.8816748  0.3083933  0.6821762
##   0.000000000  0.000000000  150      0.8722778  0.3216070  0.6725054
##   0.000000000  0.000000000  200      0.8665554  0.3299025  0.6668118
##   0.000000000  0.000100000   50      0.9035868  0.2769350  0.7032041
##   0.000000000  0.000100000  100      0.8811419  0.3093070  0.6817493
##   0.000000000  0.000100000  150      0.8718377  0.3223293  0.6721066
##   0.000000000  0.000100000  200      0.8670754  0.3291055  0.6670052
##   0.000000000  0.003162278   50      0.9029278  0.2780996  0.7025638
##   0.000000000  0.003162278  100      0.8810435  0.3094610  0.6816991
##   0.000000000  0.003162278  150      0.8721130  0.3220075  0.6726284
##   0.000000000  0.003162278  200      0.8666518  0.3297934  0.6667293
##   0.000000000  0.100000000   50      0.9017776  0.2801151  0.7014335
##   0.000000000  0.100000000  100      0.8802022  0.3109638  0.6810510
##   0.000000000  0.100000000  150      0.8700321  0.3253341  0.6708925
##   0.000000000  0.100000000  200      0.8658326  0.3310694  0.6662852
##   0.000100000  0.000000000   50      0.9034021  0.2772511  0.7030066
##   0.000100000  0.000000000  100      0.8812461  0.3091676  0.6818602
##   0.000100000  0.000000000  150      0.8718582  0.3223190  0.6723818
##   0.000100000  0.000000000  200      0.8663824  0.3301983  0.6666269
##   0.000100000  0.000100000   50      0.9038936  0.2764973  0.7033419
##   0.000100000  0.000100000  100      0.8812972  0.3091649  0.6818719
##   0.000100000  0.000100000  150      0.8719905  0.3221804  0.6724858
##   0.000100000  0.000100000  200      0.8668708  0.3294634  0.6668299
##   0.000100000  0.003162278   50      0.9029283  0.2780985  0.7025651
##   0.000100000  0.003162278  100      0.8808295  0.3098372  0.6815315
##   0.000100000  0.003162278  150      0.8718726  0.3224277  0.6726806
##   0.000100000  0.003162278  200      0.8665699  0.3299198  0.6667882
##   0.000100000  0.100000000   50      0.9015018  0.2805396  0.7011845
##   0.000100000  0.100000000  100      0.8810965  0.3094385  0.6815910
##   0.000100000  0.100000000  150      0.8709849  0.3237860  0.6716801
##   0.000100000  0.100000000  200      0.8662541  0.3303684  0.6665060
##   0.003162278  0.000000000   50      0.9037230  0.2766631  0.7032327
##   0.003162278  0.000000000  100      0.8808332  0.3098966  0.6817866
##   0.003162278  0.000000000  150      0.8711504  0.3235381  0.6719729
##   0.003162278  0.000000000  200      0.8661562  0.3306033  0.6665590
##   0.003162278  0.000100000   50      0.9038861  0.2763494  0.7032469
##   0.003162278  0.000100000  100      0.8808483  0.3098413  0.6818832
##   0.003162278  0.000100000  150      0.8712561  0.3233605  0.6721923
##   0.003162278  0.000100000  200      0.8664764  0.3301022  0.6668395
##   0.003162278  0.003162278   50      0.9028207  0.2781904  0.7028330
##   0.003162278  0.003162278  100      0.8804793  0.3103928  0.6813900
##   0.003162278  0.003162278  150      0.8719943  0.3220154  0.6723424
##   0.003162278  0.003162278  200      0.8668322  0.3294580  0.6672339
```

```
##    0.003162278  0.100000000   50      0.9024158  0.2789113  0.7017852
##    0.003162278  0.100000000  100      0.8806475  0.3103430  0.6813528
##    0.003162278  0.100000000  150      0.8707192  0.3242720  0.6714503
##    0.003162278  0.100000000  200      0.8644279  0.3333085  0.6654507
##    0.100000000  0.000000000   50      0.9028645  0.2781094  0.7023696
##    0.100000000  0.000000000  100      0.8810723  0.3095378  0.6813761
##    0.100000000  0.000000000  150      0.8710416  0.3237279  0.6718570
##    0.100000000  0.000000000  200      0.8668150  0.3295844  0.6673573
##    0.100000000  0.000100000   50      0.9028619  0.2781144  0.7023675
##    0.100000000  0.000100000  100      0.8812441  0.3092454  0.6816132
##    0.100000000  0.000100000  150      0.8715205  0.3229700  0.6722319
##    0.100000000  0.000100000  200      0.8671735  0.3290032  0.6678042
##    0.100000000  0.003162278   50      0.9030585  0.2778794  0.7024823
##    0.100000000  0.003162278  100      0.8808579  0.3099648  0.6815566
##    0.100000000  0.003162278  150      0.8710348  0.3238341  0.6714864
##    0.100000000  0.003162278  200      0.8656673  0.3313997  0.6660079
##    0.100000000  0.100000000   50      0.9036062  0.2768013  0.7033907
##    0.100000000  0.100000000  100      0.8807416  0.3099968  0.6815187
##    0.100000000  0.100000000  150      0.8711901  0.3234925  0.6716861
##    0.100000000  0.100000000  200      0.8655786  0.3315022  0.6657733
##
## Tuning parameter 'eta' was held constant at a value of 0.3
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 200, lambda =
##  0.003162278, alpha = 0.1 and eta = 0.3.
```

```
ggplot(modelXgboostLinear)
```




```
# Note: it took 30 minutes!!
# Xgboost Tree model
set.seed(1, sample.kind = "Rounding")
system.time(
  modelXgboostTree <- train(
    rating ~ .,
```

```r
    data = train_small2,
    method = "xgbTree",
    preProcess = c('center', 'scale'),
    trControl = trainControl(method = "cv"),
    tuneLength = 4)
)
```

```
##    user  system elapsed
##   34.71    0.69 1717.75
```

```r
print(modelXgboostTree)
```

```
## eXtreme Gradient Boosting
##
## 90324 samples
##    24 predictor
##
## Pre-processing: centered (24), scaled (24)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 81292, 81291, 81291, 81291, 81292, 81292, ...
## Resampling results across tuning parameters:
##
##   eta  max_depth  colsample_bytree  subsample  nrounds  RMSE       Rsquared
##   0.3  1          0.6               0.5000000   50      1.0226586  0.07301263
##   0.3  1          0.6               0.5000000  100      1.0139590  0.08842350
##   0.3  1          0.6               0.5000000  150      1.0079665  0.09889056
##   0.3  1          0.6               0.5000000  200      1.0035463  0.10595198
##   0.3  1          0.6               0.6666667   50      1.0231373  0.07251672
##   0.3  1          0.6               0.6666667  100      1.0144425  0.08815203
##   0.3  1          0.6               0.6666667  150      1.0086808  0.09847579
##   0.3  1          0.6               0.6666667  200      1.0043692  0.10529285
##   0.3  1          0.6               0.8333333   50      1.0233018  0.07239859
##   0.3  1          0.6               0.8333333  100      1.0148741  0.08736482
##   0.3  1          0.6               0.8333333  150      1.0091881  0.09797547
##   0.3  1          0.6               0.8333333  200      1.0051008  0.10434956
##   0.3  1          0.6               1.0000000   50      1.0235645  0.07195906
##   0.3  1          0.6               1.0000000  100      1.0154284  0.08684681
##   0.3  1          0.6               1.0000000  150      1.0100389  0.09670124
##   0.3  1          0.6               1.0000000  200      1.0060408  0.10350381
##   0.3  1          0.8               0.5000000   50      1.0227364  0.07248407
##   0.3  1          0.8               0.5000000  100      1.0136712  0.08902322
##   0.3  1          0.8               0.5000000  150      1.0075308  0.09973745
##   0.3  1          0.8               0.5000000  200      1.0031699  0.10684651
##   0.3  1          0.8               0.6666667   50      1.0229547  0.07279690
##   0.3  1          0.8               0.6666667  100      1.0142443  0.08844710
##   0.3  1          0.8               0.6666667  150      1.0083209  0.09909485
##   0.3  1          0.8               0.6666667  200      1.0041183  0.10585721
##   0.3  1          0.8               0.8333333   50      1.0231819  0.07242248
##   0.3  1          0.8               0.8333333  100      1.0147366  0.08778523
##   0.3  1          0.8               0.8333333  150      1.0091582  0.09779421
##   0.3  1          0.8               0.8333333  200      1.0049514  0.10478427
##   0.3  1          0.8               1.0000000   50      1.0235224  0.07214455
##   0.3  1          0.8               1.0000000  100      1.0154089  0.08721944
```

```
##    0.3  1         0.8         1.0000000  150      1.0099037  0.09701929
##    0.3  1         0.8         1.0000000  200      1.0060069  0.10356107
##    0.3  2         0.6         0.5000000   50      0.9913098  0.13049824
##    0.3  2         0.6         0.5000000  100      0.9721928  0.16199941
##    0.3  2         0.6         0.5000000  150      0.9593136  0.18362174
##    0.3  2         0.6         0.5000000  200      0.9489554  0.20105492
##    0.3  2         0.6         0.6666667   50      0.9918613  0.12947764
##    0.3  2         0.6         0.6666667  100      0.9728922  0.16097853
##    0.3  2         0.6         0.6666667  150      0.9595068  0.18380980
##    0.3  2         0.6         0.6666667  200      0.9497022  0.19994126
##    0.3  2         0.6         0.8333333   50      0.9916226  0.13059517
##    0.3  2         0.6         0.8333333  100      0.9718795  0.16372098
##    0.3  2         0.6         0.8333333  150      0.9582546  0.18647535
##    0.3  2         0.6         0.8333333  200      0.9485624  0.20239685
##    0.3  2         0.6         1.0000000   50      0.9912467  0.13211399
##    0.3  2         0.6         1.0000000  100      0.9720915  0.16396284
##    0.3  2         0.6         1.0000000  150      0.9585410  0.18677985
##    0.3  2         0.6         1.0000000  200      0.9482124  0.20364174
##    0.3  2         0.8         0.5000000   50      0.9908643  0.13140139
##    0.3  2         0.8         0.5000000  100      0.9715190  0.16340815
##    0.3  2         0.8         0.5000000  150      0.9583088  0.18537171
##    0.3  2         0.8         0.5000000  200      0.9484260  0.20168339
##    0.3  2         0.8         0.6666667   50      0.9899833  0.13311195
##    0.3  2         0.8         0.6666667  100      0.9709379  0.16468827
##    0.3  2         0.8         0.6666667  150      0.9581689  0.18612636
##    0.3  2         0.8         0.6666667  200      0.9479141  0.20317884
##    0.3  2         0.8         0.8333333   50      0.9910132  0.13121180
##    0.3  2         0.8         0.8333333  100      0.9713712  0.16444170
##    0.3  2         0.8         0.8333333  150      0.9582220  0.18643236
##    0.3  2         0.8         0.8333333  200      0.9480158  0.20318484
##    0.3  2         0.8         1.0000000   50      0.9903050  0.13344275
##    0.3  2         0.8         1.0000000  100      0.9711208  0.16554438
##    0.3  2         0.8         1.0000000  150      0.9573179  0.18885716
##    0.3  2         0.8         1.0000000  200      0.9472731  0.20545718
##    0.3  3         0.6         0.5000000   50      0.9663615  0.17416709
##    0.3  3         0.6         0.5000000  100      0.9432569  0.21194254
##    0.3  3         0.6         0.5000000  150      0.9284151  0.23566845
##    0.3  3         0.6         0.5000000  200      0.9188407  0.25049630
##    0.3  3         0.6         0.6666667   50      0.9670342  0.17412210
##    0.3  3         0.6         0.6666667  100      0.9428536  0.21325247
##    0.3  3         0.6         0.6666667  150      0.9276743  0.23759264
##    0.3  3         0.6         0.6666667  200      0.9176376  0.25310712
##    0.3  3         0.6         0.8333333   50      0.9676080  0.17304467
##    0.3  3         0.6         0.8333333  100      0.9425071  0.21393836
##    0.3  3         0.6         0.8333333  150      0.9274053  0.23810143
##    0.3  3         0.6         0.8333333  200      0.9176075  0.25332029
##    0.3  3         0.6         1.0000000   50      0.9664128  0.17531185
##    0.3  3         0.6         1.0000000  100      0.9415729  0.21645826
##    0.3  3         0.6         1.0000000  150      0.9260892  0.24100800
##    0.3  3         0.6         1.0000000  200      0.9159702  0.25674400
##    0.3  3         0.8         0.5000000   50      0.9651945  0.17598704
##    0.3  3         0.8         0.5000000  100      0.9406283  0.21643763
##    0.3  3         0.8         0.5000000  150      0.9257588  0.24012330
##    0.3  3         0.8         0.5000000  200      0.9157725  0.25561365
```

```
##   0.3  3        0.8        0.6666667   50     0.9648512  0.17787305
##   0.3  3        0.8        0.6666667  100     0.9400060  0.21779421
##   0.3  3        0.8        0.6666667  150     0.9255038  0.24080938
##   0.3  3        0.8        0.6666667  200     0.9156093  0.25625647
##   0.3  3        0.8        0.8333333   50     0.9650391  0.17745935
##   0.3  3        0.8        0.8333333  100     0.9396638  0.21892017
##   0.3  3        0.8        0.8333333  150     0.9240639  0.24394312
##   0.3  3        0.8        0.8333333  200     0.9141749  0.25924324
##   0.3  3        0.8        1.0000000   50     0.9651812  0.17760262
##   0.3  3        0.8        1.0000000  100     0.9400907  0.21881042
##   0.3  3        0.8        1.0000000  150     0.9246857  0.24310654
##   0.3  3        0.8        1.0000000  200     0.9139225  0.26007741
##   0.3  4        0.6        0.5000000   50     0.9460461  0.20855817
##   0.3  4        0.6        0.5000000  100     0.9209968  0.24763946
##   0.3  4        0.6        0.5000000  150     0.9075960  0.26786933
##   0.3  4        0.6        0.5000000  200     0.8975508  0.28324098
##   0.3  4        0.6        0.6666667   50     0.9442277  0.21244097
##   0.3  4        0.6        0.6666667  100     0.9192758  0.25107722
##   0.3  4        0.6        0.6666667  150     0.9044217  0.27378541
##   0.3  4        0.6        0.6666667  200     0.8947473  0.28835905
##   0.3  4        0.6        0.8333333   50     0.9445529  0.21216773
##   0.3  4        0.6        0.8333333  100     0.9182173  0.25353547
##   0.3  4        0.6        0.8333333  150     0.9030972  0.27668103
##   0.3  4        0.6        0.8333333  200     0.8934993  0.29097217
##   0.3  4        0.6        1.0000000   50     0.9451715  0.21135954
##   0.3  4        0.6        1.0000000  100     0.9173691  0.25558565
##   0.3  4        0.6        1.0000000  150     0.9028785  0.27755107
##   0.3  4        0.6        1.0000000  200     0.8927868  0.29265808
##   0.3  4        0.8        0.5000000   50     0.9439029  0.21157761
##   0.3  4        0.8        0.5000000  100     0.9186932  0.25105801
##   0.3  4        0.8        0.5000000  150     0.9051516  0.27155098
##   0.3  4        0.8        0.5000000  200     0.8956893  0.28590485
##   0.3  4        0.8        0.6666667   50     0.9414719  0.21677989
##   0.3  4        0.8        0.6666667  100     0.9155339  0.25726304
##   0.3  4        0.8        0.6666667  150     0.9011375  0.27921352
##   0.3  4        0.8        0.6666667  200     0.8916175  0.29323088
##   0.3  4        0.8        0.8333333   50     0.9420563  0.21589567
##   0.3  4        0.8        0.8333333  100     0.9155500  0.25745161
##   0.3  4        0.8        0.8333333  150     0.9008521  0.27984193
##   0.3  4        0.8        0.8333333  200     0.8913117  0.29414604
##   0.3  4        0.8        1.0000000   50     0.9426137  0.21575900
##   0.3  4        0.8        1.0000000  100     0.9148319  0.25945077
##   0.3  4        0.8        1.0000000  150     0.9007632  0.28050207
##   0.3  4        0.8        1.0000000  200     0.8910541  0.29503040
##   0.4  1        0.6        0.5000000   50     1.0189689  0.07882919
##   0.4  1        0.6        0.5000000  100     1.0089763  0.09700781
##   0.4  1        0.6        0.5000000  150     1.0029902  0.10683366
##   0.4  1        0.6        0.5000000  200     0.9989367  0.11306656
##   0.4  1        0.6        0.6666667   50     1.0193276  0.07827531
##   0.4  1        0.6        0.6666667  100     1.0098156  0.09601805
##   0.4  1        0.6        0.6666667  150     1.0037565  0.10597329
##   0.4  1        0.6        0.6666667  200     0.9995994  0.11239676
##   0.4  1        0.6        0.8333333   50     1.0195218  0.07818816
##   0.4  1        0.6        0.8333333  100     1.0104173  0.09518175
```

```
##   0.4  1   0.6   0.8333333  150   1.0045551  0.10488608
##   0.4  1   0.6   0.8333333  200   1.0005353  0.11135912
##   0.4  1   0.6   1.0000000   50   1.0197306  0.07796894
##   0.4  1   0.6   1.0000000  100   1.0109129  0.09489080
##   0.4  1   0.6   1.0000000  150   1.0053328  0.10426218
##   0.4  1   0.6   1.0000000  200   1.0013430  0.11057779
##   0.4  1   0.8   0.5000000   50   1.0185344  0.07953627
##   0.4  1   0.8   0.5000000  100   1.0086960  0.09782798
##   0.4  1   0.8   0.5000000  150   1.0024671  0.10816057
##   0.4  1   0.8   0.5000000  200   0.9984382  0.11395498
##   0.4  1   0.8   0.6666667   50   1.0189571  0.07891660
##   0.4  1   0.8   0.6666667  100   1.0095001  0.09659283
##   0.4  1   0.8   0.6666667  150   1.0035771  0.10640017
##   0.4  1   0.8   0.6666667  200   0.9996006  0.11254525
##   0.4  1   0.8   0.8333333   50   1.0194859  0.07840170
##   0.4  1   0.8   0.8333333  100   1.0100813  0.09601060
##   0.4  1   0.8   0.8333333  150   1.0041828  0.10613759
##   0.4  1   0.8   0.8333333  200   1.0003141  0.11172359
##   0.4  1   0.8   1.0000000   50   1.0196963  0.07834468
##   0.4  1   0.8   1.0000000  100   1.0108806  0.09481781
##   0.4  1   0.8   1.0000000  150   1.0051775  0.10488646
##   0.4  1   0.8   1.0000000  200   1.0013996  0.11058599
##   0.4  2   0.6   0.5000000   50   0.9843692  0.13999545
##   0.4  2   0.6   0.5000000  100   0.9634702  0.17537362
##   0.4  2   0.6   0.5000000  150   0.9506807  0.19676895
##   0.4  2   0.6   0.5000000  200   0.9415710  0.21155595
##   0.4  2   0.6   0.6666667   50   0.9844019  0.14073691
##   0.4  2   0.6   0.6666667  100   0.9640274  0.17461690
##   0.4  2   0.6   0.6666667  150   0.9499603  0.19849180
##   0.4  2   0.6   0.6666667  200   0.9405238  0.21372931
##   0.4  2   0.6   0.8333333   50   0.9845405  0.14083097
##   0.4  2   0.6   0.8333333  100   0.9625604  0.17803080
##   0.4  2   0.6   0.8333333  150   0.9487538  0.20067059
##   0.4  2   0.6   0.8333333  200   0.9389626  0.21688310
##   0.4  2   0.6   1.0000000   50   0.9854163  0.13937484
##   0.4  2   0.6   1.0000000  100   0.9629799  0.17766168
##   0.4  2   0.6   1.0000000  150   0.9485134  0.20177288
##   0.4  2   0.6   1.0000000  200   0.9381386  0.21886944
##   0.4  2   0.8   0.5000000   50   0.9825286  0.14383005
##   0.4  2   0.8   0.5000000  100   0.9632099  0.17579214
##   0.4  2   0.8   0.5000000  150   0.9510462  0.19577604
##   0.4  2   0.8   0.5000000  200   0.9407088  0.21301451
##   0.4  2   0.8   0.6666667   50   0.9826008  0.14418568
##   0.4  2   0.8   0.6666667  100   0.9617236  0.17910095
##   0.4  2   0.8   0.6666667  150   0.9485898  0.20062933
##   0.4  2   0.8   0.6666667  200   0.9385085  0.21735985
##   0.4  2   0.8   0.8333333   50   0.9830164  0.14344730
##   0.4  2   0.8   0.8333333  100   0.9621000  0.17890601
##   0.4  2   0.8   0.8333333  150   0.9478686  0.20260125
##   0.4  2   0.8   0.8333333  200   0.9376887  0.21927548
##   0.4  2   0.8   1.0000000   50   0.9825976  0.14517317
##   0.4  2   0.8   1.0000000  100   0.9605538  0.18227242
##   0.4  2   0.8   1.0000000  150   0.9465222  0.20547818
##   0.4  2   0.8   1.0000000  200   0.9367238  0.22152204
```

```
##   0.4  3            0.6                 0.5000000   50        0.9585270  0.18484961
##   0.4  3            0.6                 0.5000000  100        0.9341938  0.22444539
##   0.4  3            0.6                 0.5000000  150        0.9203762  0.24648533
##   0.4  3            0.6                 0.5000000  200        0.9116180  0.26013009
##   0.4  3            0.6                 0.6666667   50        0.9570624  0.18827898
##   0.4  3            0.6                 0.6666667  100        0.9322731  0.22842050
##   0.4  3            0.6                 0.6666667  150        0.9181098  0.25081205
##   0.4  3            0.6                 0.6666667  200        0.9088855  0.26523788
##   0.4  3            0.6                 0.8333333   50        0.9580447  0.18695520
##   0.4  3            0.6                 0.8333333  100        0.9320863  0.22935372
##   0.4  3            0.6                 0.8333333  150        0.9178930  0.25179863
##   0.4  3            0.6                 0.8333333  200        0.9080176  0.26716413
##   0.4  3            0.6                 1.0000000   50        0.9569041  0.18921401
##   0.4  3            0.6                 1.0000000  100        0.9314507  0.23083158
##   0.4  3            0.6                 1.0000000  150        0.9171739  0.25326638
##   0.4  3            0.6                 1.0000000  200        0.9071175  0.26894475
##   0.4  3            0.8                 0.5000000   50        0.9561116  0.18884886
##   0.4  3            0.8                 0.5000000  100        0.9330781  0.22623525
##   0.4  3            0.8                 0.5000000  150        0.9202267  0.24659220
##   0.4  3            0.8                 0.5000000  200        0.9111325  0.26065681
##   0.4  3            0.8                 0.6666667   50        0.9565786  0.18848980
##   0.4  3            0.8                 0.6666667  100        0.9309716  0.23038507
##   0.4  3            0.8                 0.6666667  150        0.9172275  0.25201827
##   0.4  3            0.8                 0.6666667  200        0.9073751  0.26740778
##   0.4  3            0.8                 0.8333333   50        0.9548307  0.19229987
##   0.4  3            0.8                 0.8333333  100        0.9296925  0.23299231
##   0.4  3            0.8                 0.8333333  150        0.9148094  0.25660047
##   0.4  3            0.8                 0.8333333  200        0.9052594  0.27136696
##   0.4  3            0.8                 1.0000000   50        0.9557951  0.19102336
##   0.4  3            0.8                 1.0000000  100        0.9296872  0.23351761
##   0.4  3            0.8                 1.0000000  150        0.9148125  0.25700235
##   0.4  3            0.8                 1.0000000  200        0.9051817  0.27188397
##   0.4  4            0.6                 0.5000000   50        0.9362683  0.22215174
##   0.4  4            0.6                 0.5000000  100        0.9131063  0.25797071
##   0.4  4            0.6                 0.5000000  150        0.9003427  0.27765861
##   0.4  4            0.6                 0.5000000  200        0.8926253  0.28928974
##   0.4  4            0.6                 0.6666667   50        0.9361923  0.22246840
##   0.4  4            0.6                 0.6666667  100        0.9117656  0.26078599
##   0.4  4            0.6                 0.6666667  150        0.8986247  0.28085332
##   0.4  4            0.6                 0.6666667  200        0.8895069  0.29468213
##   0.4  4            0.6                 0.8333333   50        0.9342998  0.22612092
##   0.4  4            0.6                 0.8333333  100        0.9086924  0.26602283
##   0.4  4            0.6                 0.8333333  150        0.8954275  0.28624076
##   0.4  4            0.6                 0.8333333  200        0.8867860  0.29933403
##   0.4  4            0.6                 1.0000000   50        0.9327059  0.22957478
##   0.4  4            0.6                 1.0000000  100        0.9065250  0.27055617
##   0.4  4            0.6                 1.0000000  150        0.8923694  0.29211660
##   0.4  4            0.6                 1.0000000  200        0.8837402  0.30491113
##   0.4  4            0.8                 0.5000000   50        0.9327488  0.22726976
##   0.4  4            0.8                 0.5000000  100        0.9114542  0.26009068
##   0.4  4            0.8                 0.5000000  150        0.9003549  0.27704090
##   0.4  4            0.8                 0.5000000  200        0.8918439  0.29023061
##   0.4  4            0.8                 0.6666667   50        0.9333663  0.22669310
##   0.4  4            0.8                 0.6666667  100        0.9084342  0.26581873
```

```
##   0.4  4          0.8                  0.6666667  150       0.8960290  0.28473599
##   0.4  4          0.8                  0.6666667  200       0.8878913  0.29704618
##   0.4  4          0.8                  0.8333333   50       0.9326909  0.22884030
##   0.4  4          0.8                  0.8333333  100       0.9067255  0.26925376
##   0.4  4          0.8                  0.8333333  150       0.8932442  0.28975303
##   0.4  4          0.8                  0.8333333  200       0.8851745  0.30177643
##   0.4  4          0.8                  1.0000000   50       0.9309782  0.23195682
##   0.4  4          0.8                  1.0000000  100       0.9046510  0.27339980
##   0.4  4          0.8                  1.0000000  150       0.8914355  0.29334555
##   0.4  4          0.8                  1.0000000  200       0.8836294  0.30482324
##   MAE
##   0.8187048
##   0.8096154
##   0.8031344
##   0.7980205
##   0.8192679
##   0.8102477
##   0.8039038
##   0.7991864
##   0.8194361
##   0.8106379
##   0.8046295
##   0.7999392
##   0.8197890
##   0.8113216
##   0.8056354
##   0.8011423
##   0.8189655
##   0.8093410
##   0.8025122
##   0.7976579
##   0.8190924
##   0.8100287
##   0.8035628
##   0.7987578
##   0.8192757
##   0.8105688
##   0.8045182
##   0.7998662
##   0.8197399
##   0.8114065
##   0.8055367
##   0.8011328
##   0.7858054
##   0.7660084
##   0.7535924
##   0.7437937
##   0.7864832
##   0.7670141
##   0.7538235
##   0.7446152
##   0.7862627
##   0.7664174
##   0.7531327
```

```
## 0.7436605
## 0.7857111
## 0.7665214
## 0.7535599
## 0.7437154
## 0.7854258
## 0.7660667
## 0.7528638
## 0.7436481
## 0.7843756
## 0.7651121
## 0.7528587
## 0.7430303
## 0.7855704
## 0.7659992
## 0.7531711
## 0.7432056
## 0.7848004
## 0.7656250
## 0.7525684
## 0.7427621
## 0.7612074
## 0.7391796
## 0.7254844
## 0.7163780
## 0.7622497
## 0.7387341
## 0.7244059
## 0.7147572
## 0.7623226
## 0.7381641
## 0.7241507
## 0.7148549
## 0.7616182
## 0.7377001
## 0.7230659
## 0.7136649
## 0.7604038
## 0.7367736
## 0.7228744
## 0.7135501
## 0.7599792
## 0.7359167
## 0.7222255
## 0.7128236
## 0.7600218
## 0.7361497
## 0.7214788
## 0.7121588
## 0.7605940
## 0.7365432
## 0.7221821
## 0.7119955
## 0.7417163
```

```
## 0.7181615
## 0.7052370
## 0.6958611
## 0.7401781
## 0.7163628
## 0.7025855
## 0.6935395
## 0.7405391
## 0.7158025
## 0.7015380
## 0.6926252
## 0.7410062
## 0.7152498
## 0.7017142
## 0.6922600
## 0.7400522
## 0.7164426
## 0.7037909
## 0.6947512
## 0.7380946
## 0.7136365
## 0.7005137
## 0.6914281
## 0.7388941
## 0.7140466
## 0.7004829
## 0.6915602
## 0.7391161
## 0.7134443
## 0.7000713
## 0.6909557
## 0.8148917
## 0.8041480
## 0.7975060
## 0.7926960
## 0.8152024
## 0.8053482
## 0.7984288
## 0.7936127
## 0.8155115
## 0.8060191
## 0.7992693
## 0.7946719
## 0.8157912
## 0.8065836
## 0.8002670
## 0.7957354
## 0.8143362
## 0.8038822
## 0.7969262
## 0.7922085
## 0.8148210
## 0.8049768
## 0.7982792
```

```
##   0.7936089
##   0.8155048
##   0.8056044
##   0.7989754
##   0.7944587
##   0.8157233
##   0.8065123
##   0.8001753
##   0.7957834
##   0.7775425
##   0.7570102
##   0.7445209
##   0.7359705
##   0.7781646
##   0.7579205
##   0.7444542
##   0.7357474
##   0.7784995
##   0.7567893
##   0.7433983
##   0.7340795
##   0.7792842
##   0.7572692
##   0.7432023
##   0.7332621
##   0.7769017
##   0.7569755
##   0.7453241
##   0.7356823
##   0.7768657
##   0.7555923
##   0.7432081
##   0.7337915
##   0.7768324
##   0.7560525
##   0.7425484
##   0.7328058
##   0.7765734
##   0.7551907
##   0.7414111
##   0.7321878
##   0.7528467
##   0.7294874
##   0.7167354
##   0.7084404
##   0.7515054
##   0.7283356
##   0.7152461
##   0.7064329
##   0.7533013
##   0.7281895
##   0.7149175
##   0.7058665
##   0.7519173
```

```
##   0.7276993
##   0.7142684
##   0.7046890
##   0.7502392
##   0.7283329
##   0.7162500
##   0.7077888
##   0.7511541
##   0.7267153
##   0.7135526
##   0.7047620
##   0.7496017
##   0.7258436
##   0.7121405
##   0.7031407
##   0.7506579
##   0.7262294
##   0.7121964
##   0.7031102
##   0.7321511
##   0.7099759
##   0.6980793
##   0.6910041
##   0.7325958
##   0.7093285
##   0.6966933
##   0.6880165
##   0.7305312
##   0.7062984
##   0.6941173
##   0.6859043
##   0.7288383
##   0.7044627
##   0.6913016
##   0.6831407
##   0.7286127
##   0.7082546
##   0.6981311
##   0.6903308
##   0.7295579
##   0.7061566
##   0.6940746
##   0.6866793
##   0.7292102
##   0.7051451
##   0.6921615
##   0.6847352
##   0.7279803
##   0.7034129
##   0.6911057
##   0.6837324
##
## Tuning parameter 'gamma' was held constant at a value of 0
## Tuning
```

```
##  parameter 'min_child_weight' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 200, max_depth = 4, eta
##  = 0.4, gamma = 0, colsample_bytree = 0.8, min_child_weight = 1 and subsample
##  = 1.
```

```
# Calculate the prediction
y_hat_final_xgbL <-  predict(modelXgboostLinear, test_small2)

# Update the result_small table
result_small <- bind_rows(result_small,
                          data.frame(Method = "XGboostLinear",
                                     RMSE = RMSE(test_small$rating, y_hat_final_xgbL)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371
Regularized Movie + User Effect	0.8879858
Matrix Factorization - recosystem	0.9307754
XGboostLinear	0.8676153

```
# Calculate the prediction
y_hat_final_xgbT <-  predict(modelXgboostTree, test_small2)

# Update the result_small table
result_small <- bind_rows(result_small,
                          data.frame(Method = "XGboostTree",
                                     RMSE = RMSE(test_small$rating, y_hat_final_xgbT)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371
Regularized Movie + User Effect	0.8879858
Matrix Factorization - recosystem	0.9307754
XGboostLinear	0.8676153
XGboostTree	0.8868028

```
# To make a model for new user / new movie (ONLY genre selection)
train_small2 %>% select(-c(movieId, year, userId, timestamp)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> train_small3
```

```r
test_small2 %>% select(-c(movieId, year, userId, timestamp)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> test_small3
```

```r
# Note: it took 40 minutes!!
# Xgboost Linear model for new user/movie
set.seed(1, sample.kind = "Rounding")
system.time(
  modelXgboostLinear_new <- train(
    rating ~ .,
    data = train_small3,
    method = "xgbLinear",
    preProcess = c('center', 'scale'),
    trControl = trainControl(method = "cv"),
    tuneLength = 4)
)
```

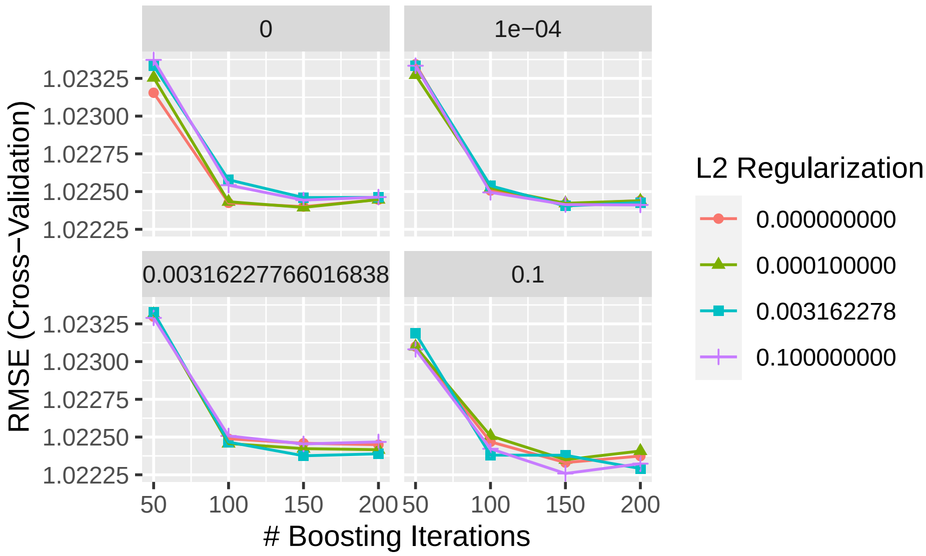
```
##    user  system elapsed
##   29.58    1.25 2290.09
```

```r
print(modelXgboostLinear_new)
```

```
## eXtreme Gradient Boosting
##
## 90324 samples
##    20 predictor
##
## Pre-processing: centered (20), scaled (20)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 81292, 81291, 81291, 81291, 81292, 81292, ...
## Resampling results across tuning parameters:
##
##   lambda       alpha        nrounds  RMSE      Rsquared    MAE
##   0.000000000  0.000000000   50      1.023155  0.06564875  0.8124933
##   0.000000000  0.000000000  100      1.022425  0.06718737  0.8111448
##   0.000000000  0.000000000  150      1.022400  0.06741502  0.8108422
##   0.000000000  0.000000000  200      1.022447  0.06746456  0.8107015
##   0.000000000  0.000100000   50      1.023343  0.06532308  0.8125872
##   0.000000000  0.000100000  100      1.022507  0.06705514  0.8112632
##   0.000000000  0.000100000  150      1.022417  0.06738914  0.8108207
##   0.000000000  0.000100000  200      1.022440  0.06747727  0.8106731
##   0.000000000  0.003162278   50      1.023298  0.06541659  0.8125052
##   0.000000000  0.003162278  100      1.022488  0.06707666  0.8111424
##   0.000000000  0.003162278  150      1.022459  0.06731463  0.8108316
##   0.000000000  0.003162278  200      1.022449  0.06745872  0.8106411
##   0.000000000  0.100000000   50      1.023099  0.06577150  0.8123362
##   0.000000000  0.100000000  100      1.022468  0.06713222  0.8112222
##   0.000000000  0.100000000  150      1.022331  0.06754676  0.8108081
##   0.000000000  0.100000000  200      1.022374  0.06758450  0.8106259
##   0.000100000  0.000000000   50      1.023255  0.06546429  0.8125892
##   0.000100000  0.000000000  100      1.022433  0.06716289  0.8111902
```

```
##   0.000100000  0.000000000  150      1.022395  0.06741537  0.8107977
##   0.000100000  0.000000000  200      1.022447  0.06744828  0.8106707
##   0.000100000  0.000100000   50      1.023273  0.06545203  0.8125506
##   0.000100000  0.000100000  100      1.022524  0.06702677  0.8112781
##   0.000100000  0.000100000  150      1.022423  0.06738114  0.8108159
##   0.000100000  0.000100000  200      1.022439  0.06747570  0.8106739
##   0.000100000  0.003162278   50      1.023314  0.06539103  0.8125366
##   0.000100000  0.003162278  100      1.022457  0.06713394  0.8111387
##   0.000100000  0.003162278  150      1.022423  0.06737839  0.8108287
##   0.000100000  0.003162278  200      1.022416  0.06751742  0.8106028
##   0.000100000  0.100000000   50      1.023099  0.06577153  0.8123361
##   0.000100000  0.100000000  100      1.022508  0.06705907  0.8112685
##   0.000100000  0.100000000  150      1.022350  0.06751041  0.8108405
##   0.000100000  0.100000000  200      1.022407  0.06752274  0.8106861
##   0.003162278  0.000000000   50      1.023333  0.06534837  0.8125681
##   0.003162278  0.000000000  100      1.022578  0.06692002  0.8113101
##   0.003162278  0.000000000  150      1.022460  0.06731398  0.8108859
##   0.003162278  0.000000000  200      1.022461  0.06742994  0.8106930
##   0.003162278  0.000100000   50      1.023333  0.06534844  0.8125681
##   0.003162278  0.000100000  100      1.022539  0.06698221  0.8112863
##   0.003162278  0.000100000  150      1.022405  0.06740218  0.8108115
##   0.003162278  0.000100000  200      1.022427  0.06748961  0.8106304
##   0.003162278  0.003162278   50      1.023326  0.06536710  0.8125555
##   0.003162278  0.003162278  100      1.022466  0.06712004  0.8111428
##   0.003162278  0.003162278  150      1.022376  0.06746516  0.8108000
##   0.003162278  0.003162278  200      1.022389  0.06755095  0.8105908
##   0.003162278  0.100000000   50      1.023186  0.06563675  0.8124563
##   0.003162278  0.100000000  100      1.022380  0.06726178  0.8111882
##   0.003162278  0.100000000  150      1.022379  0.06743415  0.8108675
##   0.003162278  0.100000000  200      1.022291  0.06770342  0.8106115
##   0.100000000  0.000000000   50      1.023372  0.06529482  0.8126703
##   0.100000000  0.000000000  100      1.022543  0.06700449  0.8112986
##   0.100000000  0.000000000  150      1.022443  0.06734986  0.8109165
##   0.100000000  0.000000000  200      1.022464  0.06743071  0.8107428
##   0.100000000  0.000100000   50      1.023334  0.06536177  0.8126644
##   0.100000000  0.000100000  100      1.022495  0.06709115  0.8112536
##   0.100000000  0.000100000  150      1.022414  0.06740032  0.8108715
##   0.100000000  0.000100000  200      1.022412  0.06751516  0.8107056
##   0.100000000  0.003162278   50      1.023289  0.06543406  0.8125539
##   0.100000000  0.003162278  100      1.022507  0.06706731  0.8112550
##   0.100000000  0.003162278  150      1.022454  0.06732874  0.8109118
##   0.100000000  0.003162278  200      1.022468  0.06743602  0.8107397
##   0.100000000  0.100000000   50      1.023081  0.06580730  0.8123527
##   0.100000000  0.100000000  100      1.022422  0.06720006  0.8111909
##   0.100000000  0.100000000  150      1.022259  0.06765055  0.8107862
##   0.100000000  0.100000000  200      1.022324  0.06764578  0.8106414
##
## Tuning parameter 'eta' was held constant at a value of 0.3
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 150, lambda = 0.1, alpha
##  = 0.1 and eta = 0.3.
```

```
ggplot(modelXgboostLinear_new)
```




```r
# Calculate the prediction
y_hat_final_xgbL_new <-  predict(modelXgboostLinear_new, test_small3)

# Update the result_small table
result_small <- bind_rows(result_small,
                          data.frame(Method = "XGboostLinear for New User/Movie",
                                     RMSE = RMSE(test_small$rating, y_hat_final_xgbL_new)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371
Regularized Movie + User Effect	0.8879858
Matrix Factorization - recosystem	0.9307754
XGboostLinear	0.8676153
XGboostTree	0.8868028
XGboostLinear for New User/Movie	1.0147697

```r
# Again, to make a model for new user (KEEP the movieId)
train_small2 %>% select(-c(year, userId, timestamp)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> train_small4

test_small2 %>% select(-c(year, userId, timestamp)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> test_small4



# Note: it took 35 minutes!!
# Xgboost Linear model for new user/movie
```

```r
set.seed(1, sample.kind = "Rounding")
system.time(
  modelXgboostLinear_newUser <- train(
    rating ~ .,
    data = train_small4,
    method = "xgbLinear",
    preProcess = c('center', 'scale'),
    trControl = trainControl(method = "cv"),
    tuneLength = 4)
)
```

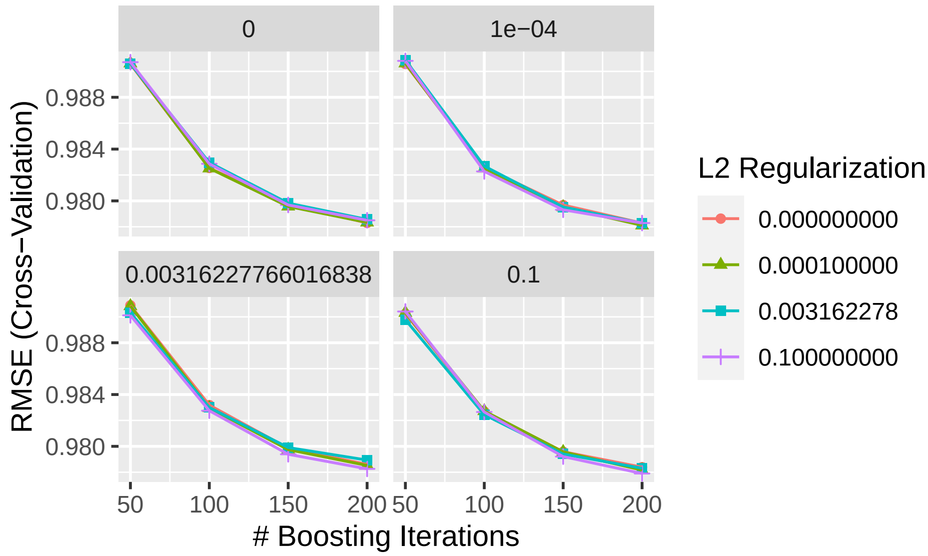
```
##    user  system elapsed
##   44.10    0.89 1986.98
```

```r
print(modelXgboostLinear_newUser)
```

```
## eXtreme Gradient Boosting
##
## 90324 samples
##    21 predictor
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 81292, 81291, 81291, 81291, 81292, 81292, ...
## Resampling results across tuning parameters:
##
##   lambda       alpha        nrounds  RMSE       Rsquared   MAE
##   0.000000000  0.000000000   50      0.9905718  0.1252092  0.7818742
##   0.000000000  0.000000000  100      0.9825392  0.1384463  0.7729092
##   0.000000000  0.000000000  150      0.9797368  0.1432080  0.7691322
##   0.000000000  0.000000000  200      0.9782981  0.1460119  0.7669334
##   0.000000000  0.000100000   50      0.9905718  0.1252092  0.7818742
##   0.000000000  0.000100000  100      0.9825851  0.1383726  0.7729963
##   0.000000000  0.000100000  150      0.9796785  0.1433102  0.7691285
##   0.000000000  0.000100000  200      0.9782857  0.1460369  0.7669162
##   0.000000000  0.003162278   50      0.9908884  0.1245655  0.7823068
##   0.000000000  0.003162278  100      0.9831837  0.1373276  0.7737418
##   0.000000000  0.003162278  150      0.9798573  0.1430094  0.7694102
##   0.000000000  0.003162278  200      0.9786080  0.1454886  0.7672619
##   0.000000000  0.100000000   50      0.9902436  0.1258803  0.7815445
##   0.000000000  0.100000000  100      0.9826899  0.1382200  0.7730423
##   0.000000000  0.100000000  150      0.9795823  0.1435174  0.7689600
##   0.000000000  0.100000000  200      0.9783889  0.1459059  0.7669422
##   0.000100000  0.000000000   50      0.9906263  0.1251272  0.7819922
##   0.000100000  0.000000000  100      0.9825156  0.1385047  0.7729141
##   0.000100000  0.000000000  150      0.9795706  0.1434990  0.7689973
##   0.000100000  0.000000000  200      0.9783347  0.1459635  0.7669137
##   0.000100000  0.000100000   50      0.9906264  0.1251271  0.7819922
##   0.000100000  0.000100000  100      0.9825157  0.1385045  0.7729143
##   0.000100000  0.000100000  150      0.9795185  0.1435908  0.7690359
##   0.000100000  0.000100000  200      0.9781091  0.1463480  0.7667518
##   0.000100000  0.003162278   50      0.9908348  0.1246694  0.7822129
##   0.000100000  0.003162278  100      0.9829611  0.1377242  0.7735969
```

```
##   0.000100000  0.003162278  150      0.9797469  0.1431856  0.7693683
##   0.000100000  0.003162278  200      0.9785213  0.1456040  0.7673675
##   0.000100000  0.100000000   50      0.9903146  0.1257455  0.7814686
##   0.000100000  0.100000000  100      0.9827267  0.1381826  0.7730890
##   0.000100000  0.100000000  150      0.9795718  0.1435487  0.7689951
##   0.000100000  0.100000000  200      0.9781626  0.1463143  0.7668421
##   0.003162278  0.000000000   50      0.9906079  0.1250642  0.7820176
##   0.003162278  0.000000000  100      0.9829593  0.1377122  0.7733981
##   0.003162278  0.000000000  150      0.9798241  0.1430850  0.7693851
##   0.003162278  0.000000000  200      0.9785649  0.1456147  0.7673183
##   0.003162278  0.000100000   50      0.9908694  0.1246199  0.7822843
##   0.003162278  0.000100000  100      0.9826822  0.1382051  0.7732123
##   0.003162278  0.000100000  150      0.9795268  0.1435710  0.7692177
##   0.003162278  0.000100000  200      0.9782727  0.1460837  0.7671844
##   0.003162278  0.003162278   50      0.9903269  0.1256178  0.7815567
##   0.003162278  0.003162278  100      0.9830016  0.1376637  0.7734435
##   0.003162278  0.003162278  150      0.9799047  0.1429209  0.7693579
##   0.003162278  0.003162278  200      0.9789373  0.1449654  0.7673970
##   0.003162278  0.100000000   50      0.9897787  0.1267296  0.7813190
##   0.003162278  0.100000000  100      0.9824438  0.1386684  0.7729609
##   0.003162278  0.100000000  150      0.9794072  0.1438220  0.7689269
##   0.003162278  0.100000000  200      0.9783025  0.1460663  0.7668465
##   0.100000000  0.000000000   50      0.9907131  0.1250247  0.7820704
##   0.100000000  0.000000000  100      0.9828633  0.1379342  0.7733804
##   0.100000000  0.000000000  150      0.9796847  0.1433170  0.7693557
##   0.100000000  0.000000000  200      0.9785007  0.1456319  0.7674193
##   0.100000000  0.000100000   50      0.9908137  0.1248435  0.7820966
##   0.100000000  0.000100000  100      0.9822745  0.1389785  0.7729065
##   0.100000000  0.000100000  150      0.9793120  0.1439612  0.7689832
##   0.100000000  0.000100000  200      0.9782890  0.1460394  0.7671353
##   0.100000000  0.003162278   50      0.9901168  0.1261836  0.7815049
##   0.100000000  0.003162278  100      0.9827509  0.1381070  0.7731920
##   0.100000000  0.003162278  150      0.9793874  0.1437917  0.7689444
##   0.100000000  0.003162278  200      0.9782592  0.1460406  0.7670333
##   0.100000000  0.100000000   50      0.9904147  0.1256314  0.7818342
##   0.100000000  0.100000000  100      0.9826503  0.1383127  0.7729789
##   0.100000000  0.100000000  150      0.9792085  0.1441337  0.7687148
##   0.100000000  0.100000000  200      0.9779021  0.1466735  0.7666347
##
## Tuning parameter 'eta' was held constant at a value of 0.3
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 200, lambda = 0.1, alpha
##  = 0.1 and eta = 0.3.
```

```
ggplot(modelXgboostLinear_newUser)
```




```
# Calculate the prediction
y_hat_final_xgbL_newUser <-  predict(modelXgboostLinear_newUser, test_small4)

# Update the result_small table
result_small <- bind_rows(result_small,
                          data.frame(Method = "XGboostLinear for New User",
                                     RMSE = RMSE(test_small$rating, y_hat_final_xgbL_newUser)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371
Regularized Movie + User Effect	0.8879858
Matrix Factorization - recosystem	0.9307754
XGboostLinear	0.8676153
XGboostTree	0.8868028
XGboostLinear for New User/Movie	1.0147697
XGboostLinear for New User	0.9728732

```
# Once more, to make a model for new movie (KEEP the userId)
train_small2 %>% select(-c(year, movieId, timestamp)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> train_small5

test_small2 %>% select(-c(year, movieId, timestamp)) %>%
  mutate_all(~ifelse(is.na(.), median(., na.rm = TRUE), .)) -> test_small5



# Note: it took 35 minutes!!
```

```r
# Xgboost Linear model for new user/movie
set.seed(1, sample.kind = "Rounding")
system.time(
  modelXgboostLinear_newMovie <- train(
    rating ~ .,
    data = train_small5,
    method = "xgbLinear",
    preProcess = c('center', 'scale'),
    trControl = trainControl(method = "cv"),
    tuneLength = 4)
)
```

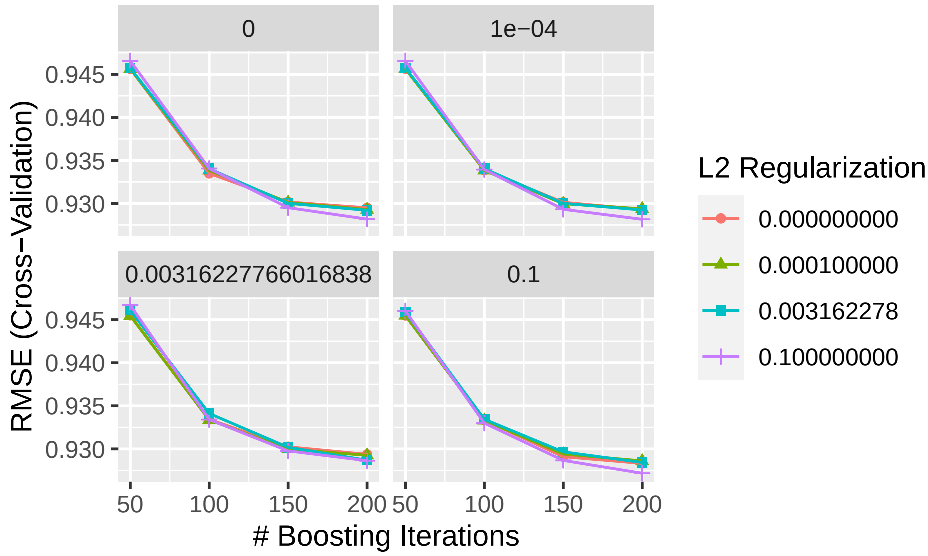
```
##    user  system elapsed
##   39.50    0.91 2084.00
```

```r
print(modelXgboostLinear_newMovie)
```

```
## eXtreme Gradient Boosting
##
## 90324 samples
##    21 predictor
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 81292, 81291, 81291, 81291, 81292, 81292, ...
## Resampling results across tuning parameters:
##
##   lambda       alpha        nrounds  RMSE       Rsquared   MAE
##   0.000000000  0.000000000   50      0.9456368  0.2050042  0.7394535
##   0.000000000  0.000000000  100      0.9334930  0.2226497  0.7265766
##   0.000000000  0.000000000  150      0.9301666  0.2276977  0.7223797
##   0.000000000  0.000000000  200      0.9294869  0.2292551  0.7207675
##   0.000000000  0.000100000   50      0.9456505  0.2049876  0.7395026
##   0.000000000  0.000100000  100      0.9338877  0.2219811  0.7268404
##   0.000000000  0.000100000  150      0.9301310  0.2277544  0.7222199
##   0.000000000  0.000100000  200      0.9292987  0.2295448  0.7205373
##   0.000000000  0.003162278   50      0.9455125  0.2052888  0.7395488
##   0.000000000  0.003162278  100      0.9334305  0.2227722  0.7266116
##   0.000000000  0.003162278  150      0.9302448  0.2275659  0.7225952
##   0.000000000  0.003162278  200      0.9293659  0.2294233  0.7208566
##   0.000000000  0.100000000   50      0.9454969  0.2054991  0.7392016
##   0.000000000  0.100000000  100      0.9333471  0.2229396  0.7265349
##   0.000000000  0.100000000  150      0.9290986  0.2294539  0.7216903
##   0.000000000  0.100000000  200      0.9283141  0.2310321  0.7199741
##   0.000100000  0.000000000   50      0.9456335  0.2050569  0.7396036
##   0.000100000  0.000000000  100      0.9338389  0.2220710  0.7268325
##   0.000100000  0.000000000  150      0.9301123  0.2277949  0.7222712
##   0.000100000  0.000000000  200      0.9292860  0.2295662  0.7205707
##   0.000100000  0.000100000   50      0.9456334  0.2050572  0.7396036
##   0.000100000  0.000100000  100      0.9338380  0.2220726  0.7268306
##   0.000100000  0.000100000  150      0.9299700  0.2280324  0.7220791
##   0.000100000  0.000100000  200      0.9293743  0.2294265  0.7205305
##   0.000100000  0.003162278   50      0.9454596  0.2053874  0.7394779
```

```
##   0.000100000  0.003162278  100      0.9333765  0.2228972  0.7266376
##   0.000100000  0.003162278  150      0.9299682  0.2280354  0.7224897
##   0.000100000  0.003162278  200      0.9292565  0.2296006  0.7207442
##   0.000100000  0.100000000   50      0.9454880  0.2054719  0.7392963
##   0.000100000  0.100000000  100      0.9333020  0.2230091  0.7265240
##   0.000100000  0.100000000  150      0.9294058  0.2289393  0.7218420
##   0.000100000  0.100000000  200      0.9286062  0.2305687  0.7201968
##   0.003162278  0.000000000   50      0.9457237  0.2048777  0.7395789
##   0.003162278  0.000000000  100      0.9340330  0.2217273  0.7269491
##   0.003162278  0.000000000  150      0.9300138  0.2279451  0.7223617
##   0.003162278  0.000000000  200      0.9292036  0.2296660  0.7210091
##   0.003162278  0.000100000   50      0.9457236  0.2048779  0.7395788
##   0.003162278  0.000100000  100      0.9340328  0.2217278  0.7269490
##   0.003162278  0.000100000  150      0.9300134  0.2279458  0.7223616
##   0.003162278  0.000100000  200      0.9292378  0.2296133  0.7209882
##   0.003162278  0.003162278   50      0.9461039  0.2043235  0.7399164
##   0.003162278  0.003162278  100      0.9341140  0.2216154  0.7269243
##   0.003162278  0.003162278  150      0.9301847  0.2276601  0.7223270
##   0.003162278  0.003162278  200      0.9286841  0.2304862  0.7202667
##   0.003162278  0.100000000   50      0.9458931  0.2047508  0.7395238
##   0.003162278  0.100000000  100      0.9334431  0.2228081  0.7267514
##   0.003162278  0.100000000  150      0.9296563  0.2285458  0.7224768
##   0.003162278  0.100000000  200      0.9284234  0.2308997  0.7204987
##   0.100000000  0.000000000   50      0.9465615  0.2035480  0.7402912
##   0.100000000  0.000000000  100      0.9340568  0.2217650  0.7274988
##   0.100000000  0.000000000  150      0.9295094  0.2287799  0.7223541
##   0.100000000  0.000000000  200      0.9281779  0.2312318  0.7203746
##   0.100000000  0.000100000   50      0.9465607  0.2035489  0.7402896
##   0.100000000  0.000100000  100      0.9339316  0.2219768  0.7274960
##   0.100000000  0.000100000  150      0.9293187  0.2290974  0.7222441
##   0.100000000  0.000100000  200      0.9281561  0.2312669  0.7203711
##   0.100000000  0.003162278   50      0.9466990  0.2033827  0.7401534
##   0.100000000  0.003162278  100      0.9334074  0.2229166  0.7264732
##   0.100000000  0.003162278  150      0.9297704  0.2283653  0.7224056
##   0.100000000  0.003162278  200      0.9286298  0.2305179  0.7201823
##   0.100000000  0.100000000   50      0.9460278  0.2043751  0.7390793
##   0.100000000  0.100000000  100      0.9329931  0.2235792  0.7260371
##   0.100000000  0.100000000  150      0.9286647  0.2301933  0.7212111
##   0.100000000  0.100000000  200      0.9271721  0.2328562  0.7191945
##
## Tuning parameter 'eta' was held constant at a value of 0.3
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nrounds = 200, lambda = 0.1, alpha
##  = 0.1 and eta = 0.3.
```

```
ggplot(modelXgboostLinear_newMovie)
```




```r
# Calculate the prediction
y_hat_final_xgbL_newMovie <-  predict(modelXgboostLinear_newMovie, test_small5)

# Update the result_small table
result_small <- bind_rows(result_small,
                          data.frame(Method = "XGboostLinear for New Movie",
                                     RMSE = RMSE(test_small$rating, y_hat_final_xgbL_newMovie)))

# Show the RMSE improvement
result_small %>% kable()
```

Method	RMSE
Target	0.8649000
Mean	1.0547663
Mean + bi	0.9860457
Mean + bi + bu	0.9056371
Regularized Movie + User Effect	0.8879858
Matrix Factorization - recosystem	0.9307754
XGboostLinear	0.8676153
XGboostTree	0.8868028
XGboostLinear for New User/Movie	1.0147697
XGboostLinear for New User	0.9728732
XGboostLinear for New Movie	0.9282118

5 Conclusion

As a result of training various models, it is clear that movieId and userId contribute more to the prediction than genre and year of Rate (timestamp). The results of the analysis using recosystem showed that with a large data size, only two pieces of information, movieId and userId, are highly accurate.

Although we were not able to run it on the target data set, we were able to build a model with movieId and userId missing and find a possibility for improvement. We did multiple machine learning runs, but not enough studies with tuning. Since there are many places where default values are used, Future work is to tune the parameters in an environment where faster calculations can be done.

Reference

- rafalab.github https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems
- Pandoc https://pandoc.org/index.html
- The caret Package https://topepo.github.io/caret/index.html
- XGBoost.ai https://xgboost.ai/
- kaggle: Simple R - xgboost - caret kernel https://www.kaggle.com/nagsdata/simple-r-xgboost-caret-kernel
- LIBMF: A Matrix-factorization Library for Recommender Systems https://www.csie.ntu.edu.tw/~cjlin/libmf/
- MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf
- Wikipedia: Curse of dimensionally https://en.wikipedia.org/wiki/Curse_of_dimensionality
- Wikipedia: Netrlix Prize https://en.wikipedia.org/wiki/Netflix_Prize
- Wikipedia: Matrix factorization (recommender systems) https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)
- Wikipedia: Gradient boosting https://en.wikipedia.org/wiki/Gradient_boosting
- W.-S. Chin, et at. "*A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems*", (ACM TIST, 2015) https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_journal.pdf