MovieLens Project - HarvardX - PH125.9x Capstone Project

Gaurav Pitroda

13/06/2019

Contents

1 Introduction

This project is part of assignment for course HarvardX - PH125.8x Data Science - Capstone module. This projects analyzes MovieLens database to calculate RMSE (Root Mean Square Error) to compare it against actual data.

2 Get the Data Set

2.1 Install required packages

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

```
## Loading required package: tidyverse
```

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

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.1     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

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

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

```
##
## Attaching package: 'caret'
```

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

2.2 Download the data

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

ratings <- read.table(text = gsub("::", "\t",
                                  readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                      col.names = c("userId", "movieId", "rating", "timestamp"))
```

2.3 Load and tidy thd data

```
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data

#set.seed(1) # if using R 3.6.0: set.seed(1, sample.kind = "Rounding")
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

# Add rows removed from validation set back into edx set

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

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

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

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

2.4 Find the dimension

```
print("Dimension: ")
```

```
## [1] "Dimension: "
```

```
dim(edx)
```

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

```
print("Rows:")
```

```
## [1] "Rows:"
```

```
nrow(edx)
```

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

```
print("Columns:")
```

```
## [1] "Columns:"
```

```
ncol(edx)
```

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

2.5 Inspect the data

```
head(edx)
```

```
print("The number of unique movies: ")
```

```
## [1] "The number of unique movies: "
```

```
n_distinct(edx$movieId)
```

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

```
print("The number of unique users")
```

```
## [1] "The number of unique users"
```

```
n_distinct(edx$userId)
```

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

```
print("Movie ratings by genre")
```

```
## [1] "Movie ratings by genre"
```

```
edx %>% separate_rows(genres, sep = "\\|") %>%
    group_by(genres) %>%
    summarize(count = n()) %>%
    arrange(desc(count))
```

```
print ("Movie with highest ratings in descending order")
```

```
## [1] "Movie with highest ratings in descending order"
```

```
edx %>% group_by(movieId, title) %>%
    summarize(count = n()) %>%
    arrange(desc(count))
```

```
print ("Top 5 ratings")
```

```
## [1] "Top 5 ratings"
```

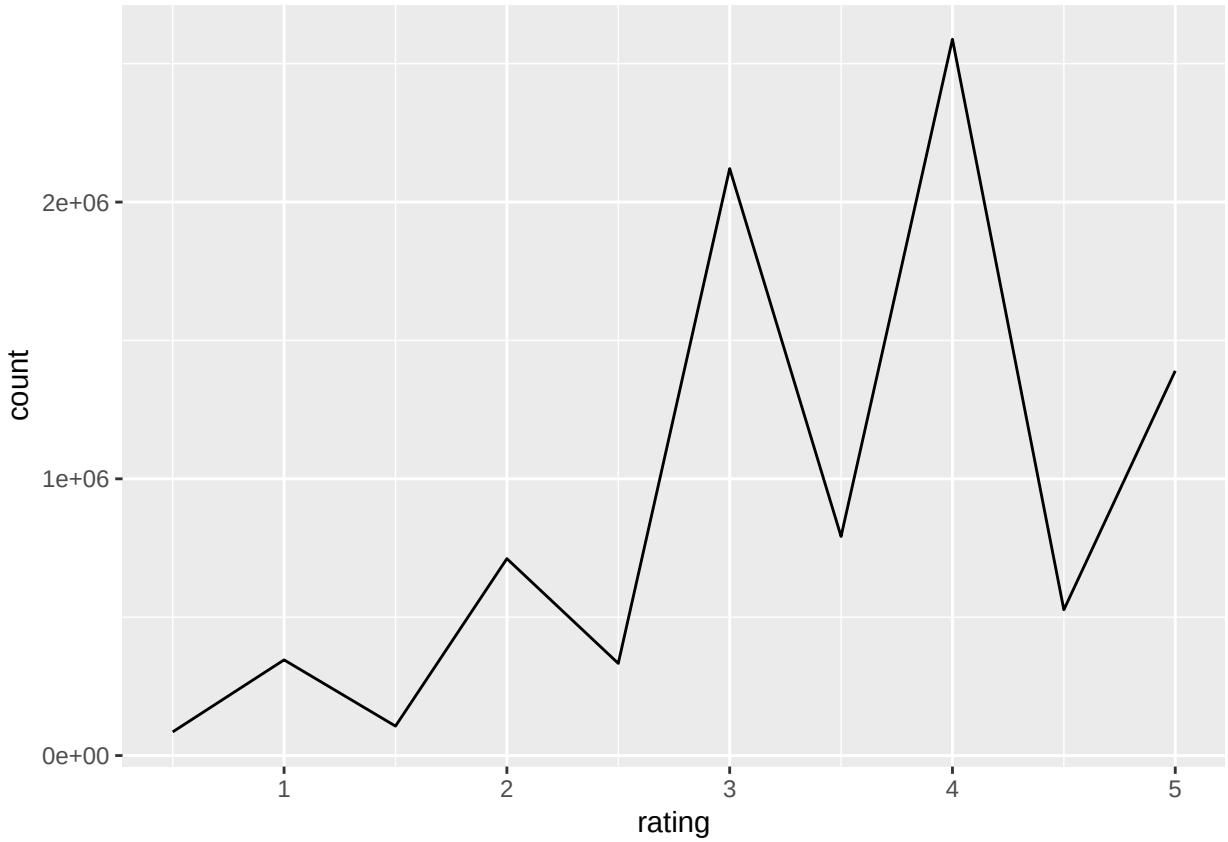
```
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5) %>%
    arrange(desc(count))
```

```
## Selecting by count
```

```
print("Graph of ratings the movies")
```

```
## [1] "Graph of ratings the movies"
```

```
edx %>%
    group_by(rating) %>%
    summarize(count = n()) %>%
    ggplot(aes(x = rating, y = count)) +
    geom_line()
```



The root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSD represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences. These deviations are called residuals when the calculations are performed over the data sample that was used for estimation and are called errors (or prediction errors) when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various times into a single measure of predictive power. It tells you how concentrated the data is around the line of best fit

Loss-function that compute the RMSE is defined as follows:

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

Where, N - is the number of user-movie combinations and the sum over all these combinations. RMSE - is our measure of model accuracy.

RMSE is very similar to a standard deviation. it is the typical error we make while predicting a movie rating by user. RMSE > 1 means that actual rating vs predicted rating is distanced by 1, which is not a good prediction. Lower the value of RMSE implies better prediction.

The function to compute the RMSE for vectors of ratings and their corresponding predictions in R is.

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

The approach to minimize RMSE is like this.

- Build Initial model and calculate baseline RMSE
- Check Movie effects on the model and optimize
- Check Movie and User effects on the model and optimize RMSE
- Improve model based on above results

3 Modelling

##Initial Model

Initial model predicts the same rating for all movies, so we shall compute the dataset's mean rating. The expected rating of the underlying data set is between 3 and 4.

We start by building the simplest possible recommendation system by predicting the same rating for all movies irrespective of user who rates it. A model based approach assumes the same rating for all movie with all differences explained by random variation :

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

Where $\epsilon_{u,i}$ is the independent error sample from the same distribution centered at 0 μ the "true" rating for all movies.

This initial model assumes that all differences in movie ratings are a result of a random variable alone.

We know that the estimate that minimize the RMSE is the least square estimate of $Y_{u,i}$, in this case, is the average of all ratings.

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

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

The expected rating of the underlying data set is 3.512464, which is between between 3 and 4.

If we predict all unknown ratings with μ or mu, we obtain the first naive RMSE:

```
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.061202
```

Here, we represent results table with the first RMSE:

```
rmse_results <- tibble(method = "Average movie rating model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.061202

We can use this initial baseline RMSE to compare against models we are going to consider next.

3.1 Movie effect on model

Some movies are just generally rated higher than others depending on various factors. These higher ratings are generally linked to popular movies among users and the opposite is true for unpopular movies.

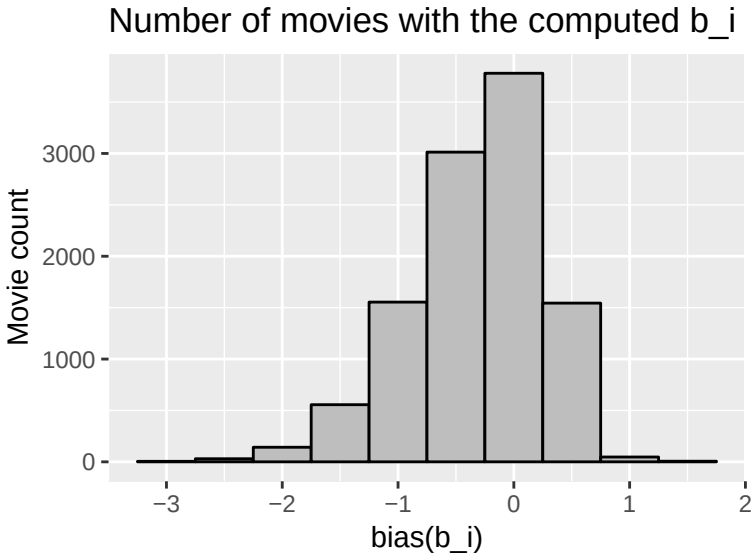
We compute the estimated deviation of each movie's mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

The histogram is left skewed which implies that more movies have negative effects

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
movie_avgs %>% ggplot(mapping = aes(x = b_i)) +
                  geom_histogram(bins = 10, fill="grey", colour = "black") +
                  labs(x = "bias(b_i)", y = "Movie count", title = "Number of movies with the computed b
```



This we call the penalty term movie effect. We can improve our prediction by taking this into the account.

```
predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

movie_effect_model_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results, tibble(method="Movie effect model", RMSE = movie_effect_model_rm
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087

Now we have movie rating prediction based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse that the average rating of all movies μ , we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average.

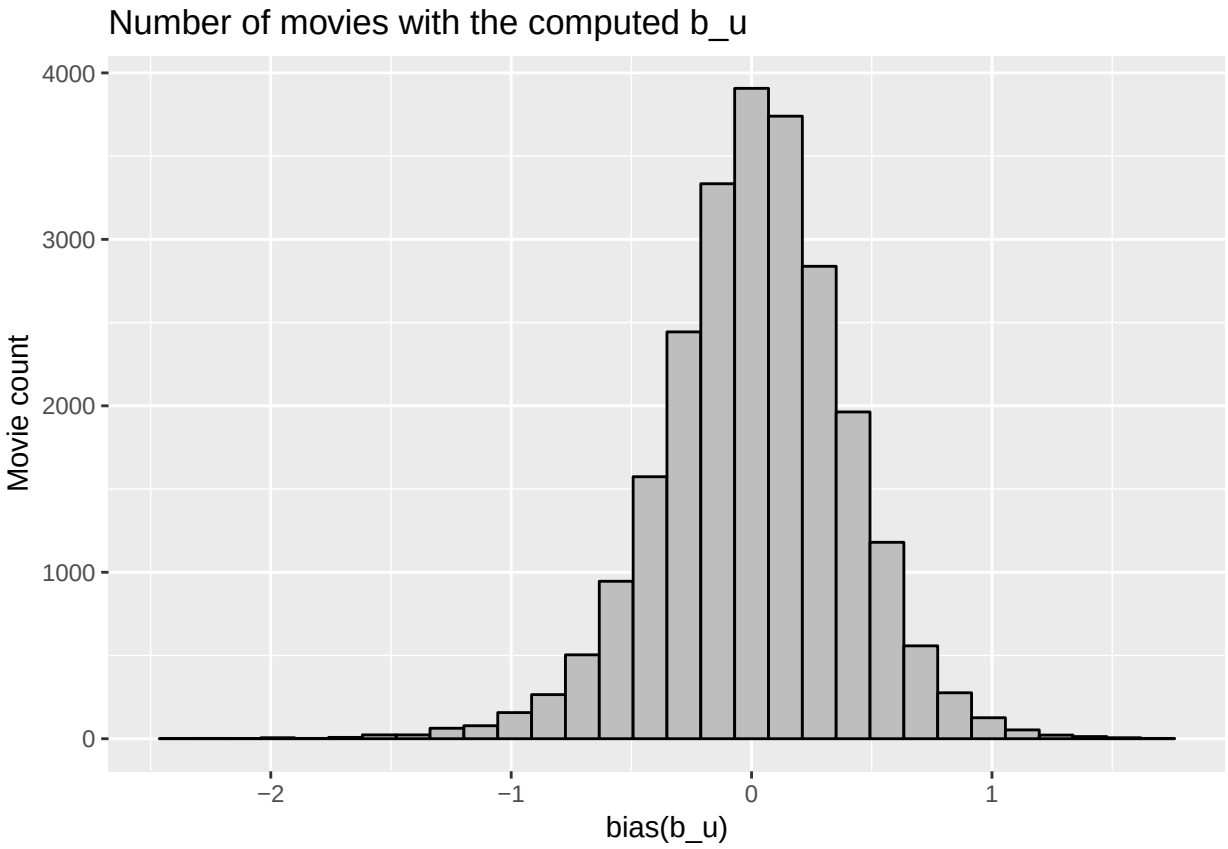
There is an improvement over previous model. Let's include user rating into consideration as well.

3.2 Movie and user effect on model

To improve the model, let's take users into account who have rated at least 100 movies We compute the average rating for user μas below.

```
user_avgs<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
user_avgs %>% ggplot(mapping = aes(x = b_u)) +
  geom_histogram(bins = 30, fill="grey", colour = "black") +
  labs(x = "bias(b_u)", y = "Movie count", title = "Number of movies with the computed b_u")
```



There is a substantial variability across users as well. We can see that some users give good ratings to all movies, while others are very selective. We can further improve our model by taking this into account

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

where b_u is a user-specific effect.

If a selective user (negative b_u rates a great movie (positive b_i), the effects counter each other and we may be able to better predict that this user gave this good movie a 3 rather than a 5.

We compute an approximation by computing μ and b_i, and estimating b_u, as the average of

$$Y_{u,i} - \mu - b_i$$

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

We can now construct predictors and see RMSE improves:

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
```

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

models_user_movies_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results, tibble(method="Movie and user effect model", RMSE = models_user
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

Our rating predictions further reduced the RMSE. The supposes "best" and "worst" movies were rated by few users, in most cases just one user. These movies were mostly obscure ones. This is because with a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely. These large errors can increase our RMSE.

Until now, we computed standard error and constructed confidence intervals to account for different levels of uncertainty. However while making predictions single prediction is better than a range. To acualize this idea, we introduce the concept of regularization, which permits to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize. Regularization is a method used to reduce the effect of overfitting.

3.3 Improve the model

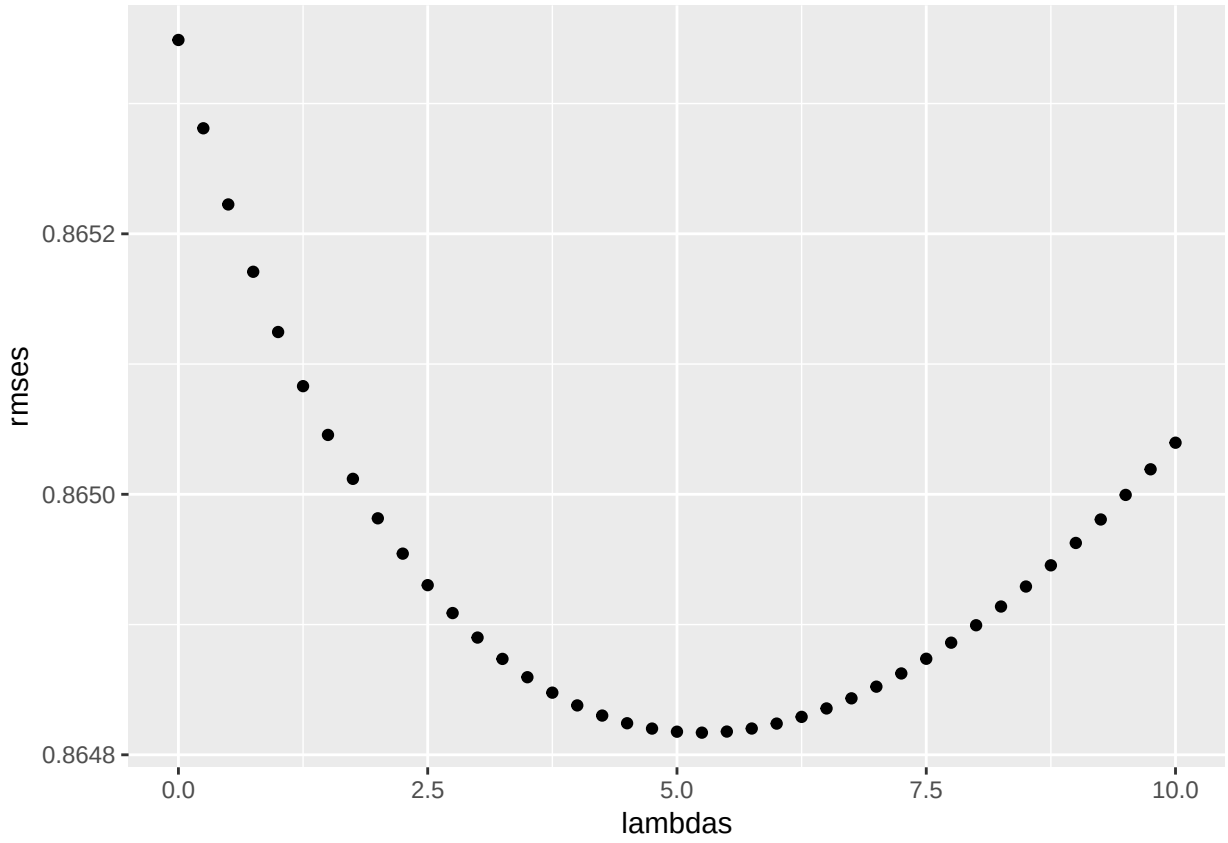
Estimates of b_i and b_u are caused by movies with very few ratings and in some cases by users who rated a very small number of movies. This can have a significant influence the our prediction. The use of the regularization permits to penalize these aspects. We should find the value of lambda (that is a tuning parameter) that will minimize the RMSE. It will also shrink the b_i and b_u in case of small number of ratings.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l) {
  mu <- mean(edx$rating)

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

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

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

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

```
})
```

Let's plot RMSE vs lambdas to select the optimal lambda

```
qplot(lambdas, rmses)
```



For the entire model, the optimal lambda is:

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

```
## [1] 5.25
```

For the full model, the optimal lambda is 5.25

The new results will be:

```
rmse_results <- bind_rows(rmse_results, tibble(method="Improvized movie and user effect model", RMSE = r
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Improvized movie and user effect model	0.8648170

method	RMSE

4 Appendix - Enviroment

This report file was created on following environment

```
print("R software version details:")
```

```
## [1] "R software version details:"
```

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.0
## year           2019
## month          04
## day            26
## svn rev        76424
## language       R
## version.string R version 3.6.0 (2019-04-26)
## nickname       Planting of a Tree
```