HarvardX : PH125.9x Data Science Capstone - Movie Recommendation System

Meghana RV

23/12/2021

Contents

1 Abstract

This project is part of the HarvardX : PH125.9x Data Science : Capstone course. In this project, we have two data sets - `edx` , which is a train set and `validation` which is the test set. The aim of this project is to make use of the `edx` data-set to train a machine learning algorithm to predict user ratings of various movies. This algorithm is then used to predict movie ratings in the `validation` set.

2 Introduction

Recommendation Systems use ratings that users have given items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive data sets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

For example, Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. We will follow a similar approach. We use the MovieLens-Dataset (10M version) to build a Movie Recommendation System. The R code used to construct these data sets, models and plots is available in **this** GitHub repository.

3 Evaluating Factor

The **Root-Mean-Square-Error** (in short, RMSE) is a measure of the difference between values predicted by a model and the observed values. It is a frequently used measure of accuracy. These deviations are called **residuals** when when the calculations are performed over the data sample that was used for estimation and are called **errors** when computed out-of-sample.

RMSE is always non-negative. Lower the RMSE, greater is the accuracy of the prediction. Hence we try to achieve the lowest possible value, albeit a value of 0 is almost never achieved in practice. The RMSE for a model m is given by :

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1..N} (y_i - x_i)^2}$$

where N is the sample size, y_i are the predicted ratings and x_i are the observe values.

We will test multiple models and decide which one is the best, based on the RMSE value. Finally, the best model will be used on the `validation` set.

4 Initial Process

4.1 Attaching the required packages

```
library(tidyverse)
library(caret)
library(data.table)
library(dslabs)
library(dplyr)
library(stringr)
library(tidyr)
library(kableExtra)
library(knitr)
library(knitLatex)
library(randomForest)
library(Rborist)
library(recommenderlab)
library(recosystem)
```

4.2 Downloading the required Dataset

We first download the MovieLens Dataset :

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

4.3 Creating edx and validation datasets

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

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

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

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

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

4.4 Splitting edx data-set into test and train sets

We will split the `edx` data set into a `test` and `train` set. We will use these for the initial models. Also, we will remove entries in the `test` set that don't appear in the `train` set to prevent `NAs` .

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.5, list = FALSE)
edx_train <- movielens[-test_index,]
edx_test <- movielens[test_index,]

edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

The function for computing RMSE is defined as follows :

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

5 Analyzing the given data

Let us first obtain a general idea/picture (quite literally) of the data we have via a summary table.

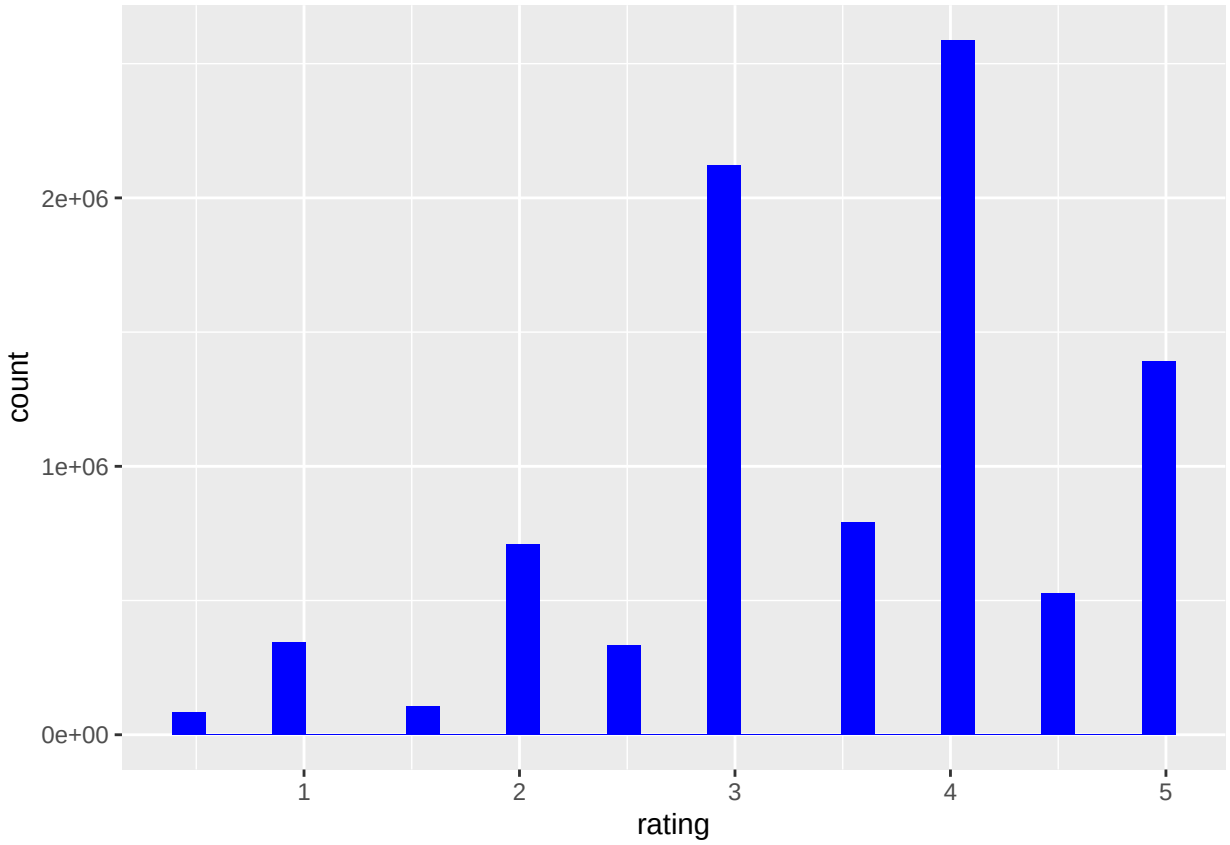
```
summary <- data.frame(Total_Rows = nrow(edx),
                      Total_Columns = ncol(edx),
                      Total_Movies = n_distinct(edx$movieId),
                      Total_Users = n_distinct(edx$userId),
                      Total_Genres = n_distinct(edx$genres),
                      Average_rating = round(mean(edx$rating),2),
                      Standard_Deviation = round((sd(edx$rating)), digits = 3))

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

Total_Rows	Total_Columns	Total_Movies	Total_Users	Total_Genres	Average_rating	Standard_Deviation
9000055	6	10677	69878	797	3.51	1.06

There are *9000055* rows, *6* columns, *10677* movies, *69878* users who rated the movies and *797* genres. The average rating for a movie is *3.51*, with a standard deviation of about *1.06*.

A visual representation of the rating distribution

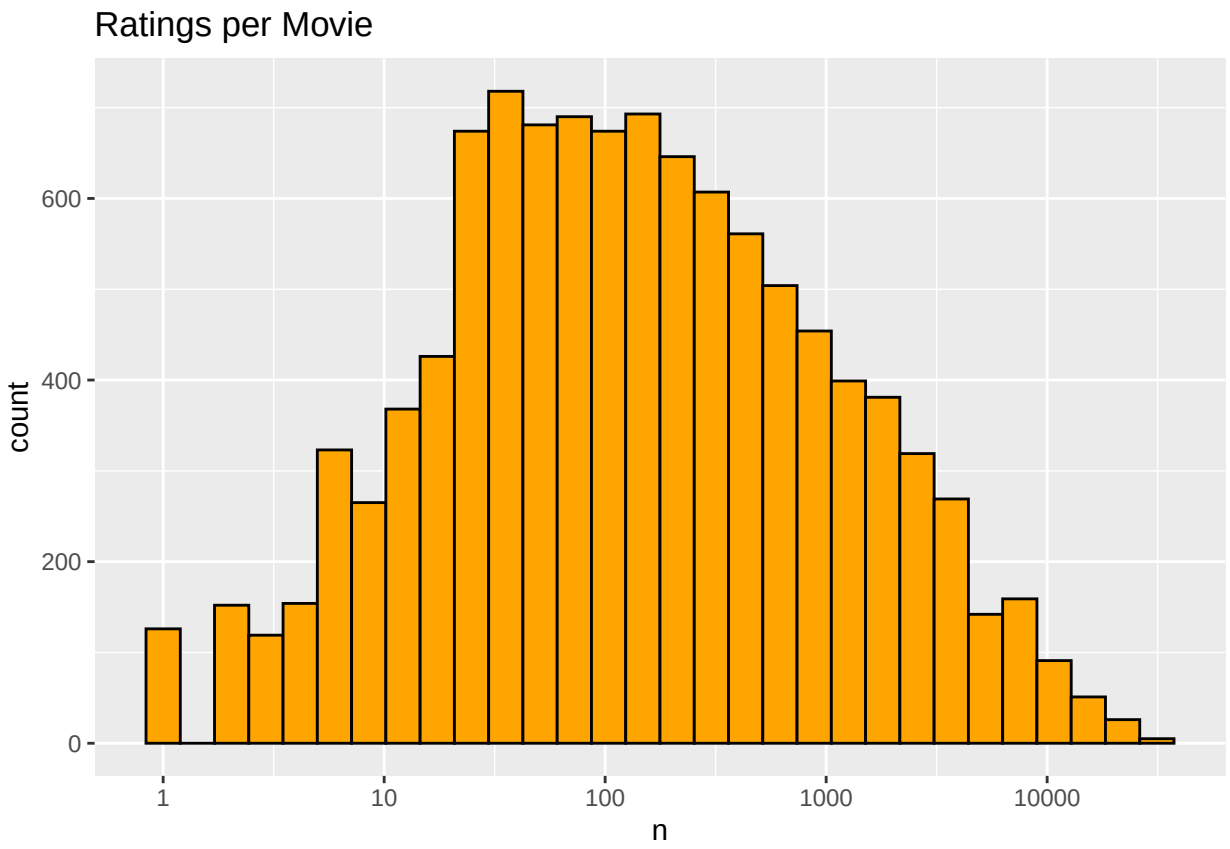
```
ggplot(edx, aes(rating)) +
  geom_histogram(fill= "blue")
```



We can see that, the most common rating is 4 stars. Also, some movies have been rated very often, whereas some movies have very few ratings.

Ratings per movie

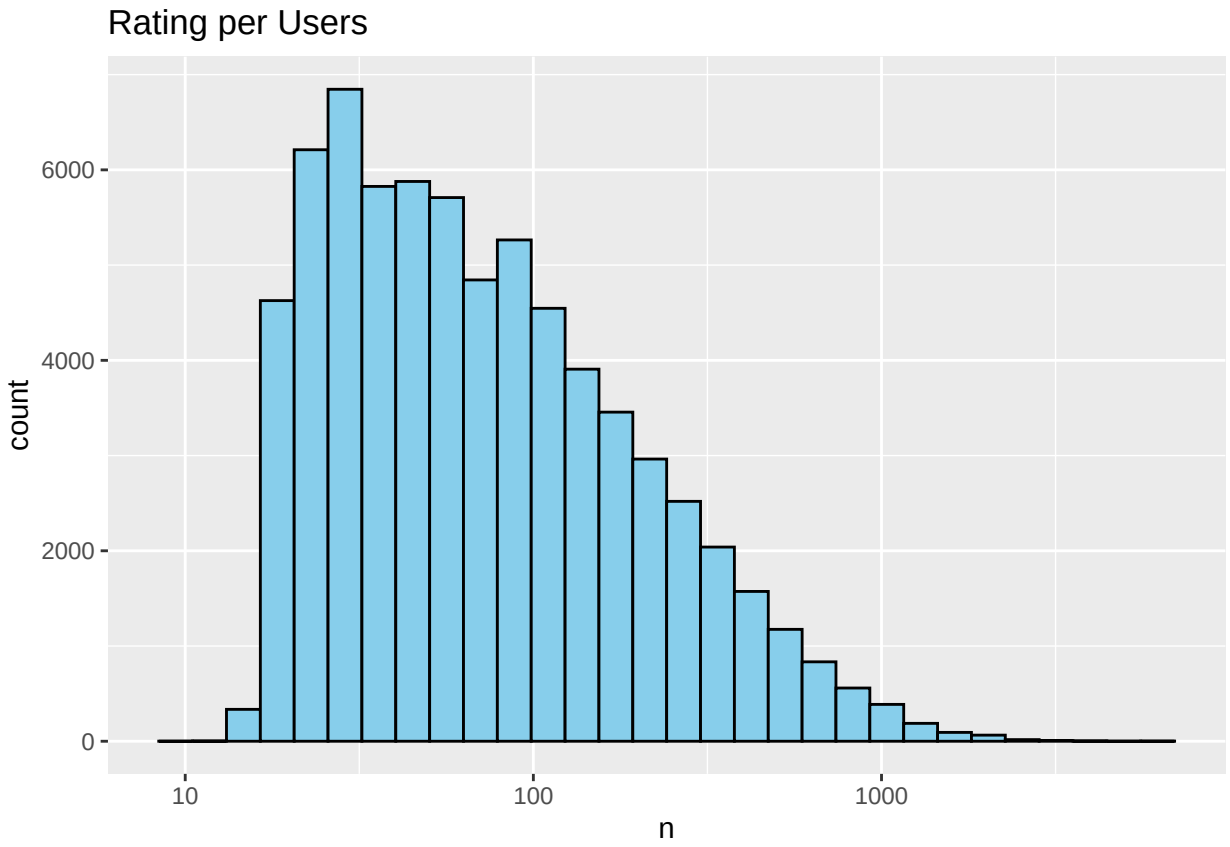
```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, fill = "orange", color = "black") +
  scale_x_log10() +
  ggtitle("Ratings per Movie")
```



Clearly, there are movies that get rated very frequently (these could be blockbuster movies) and movies are obscure and don't get rated so often.

Ratings per user

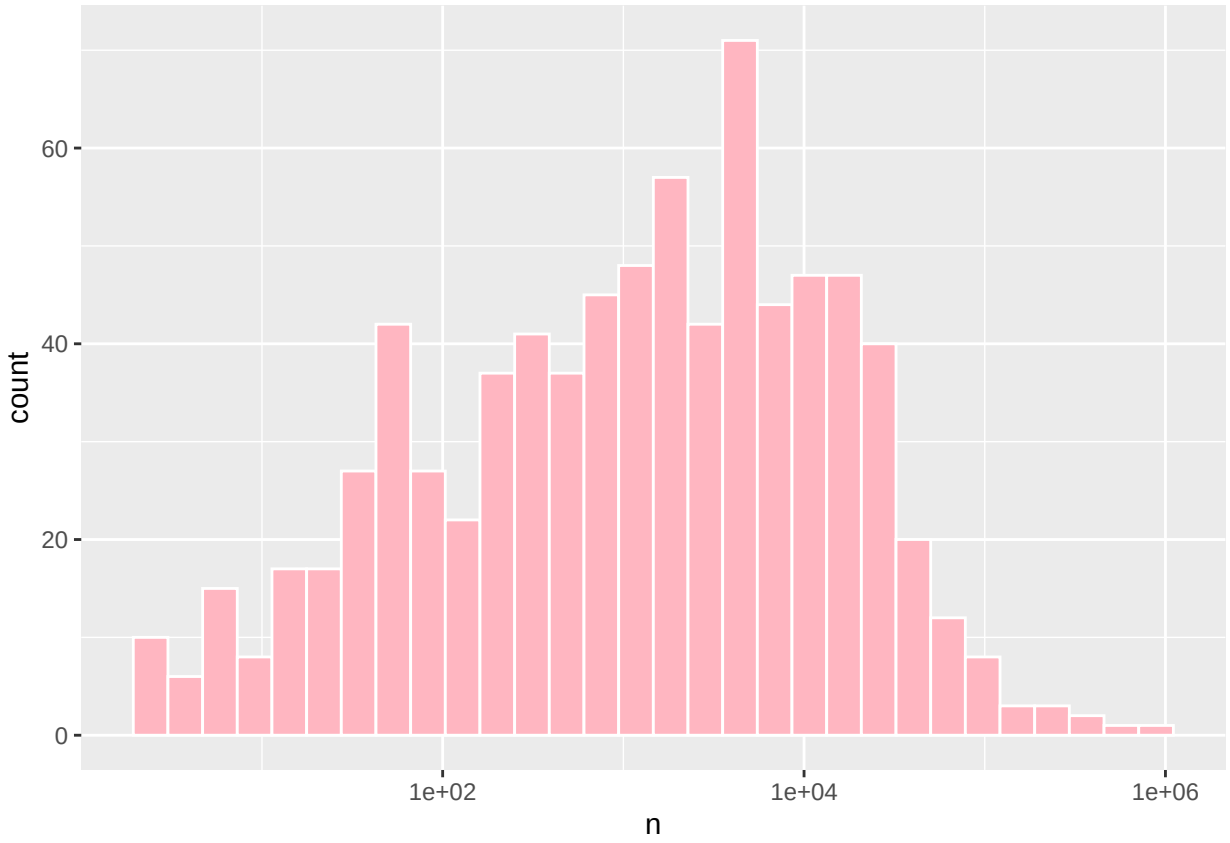
```
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  scale_x_log10() +
  ggtitle("Rating per Users")
```



Many movies have been rated by just a single user, showing that some users are more active than others.

Ratings per Genre

```
edx %>%
  dplyr::count(genres) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "lightpink", color = "white") +
  scale_x_log10()
```




```
  ggtitle("Ratings per Genre")
```

```
## $title
## [1] "Ratings per Genre"
##
## attr(,"class")
## [1] "labels"
```

There are some genres that have many ratings and those with very few ratings.

6 Fitting Models to the data

Model 1

Average Rating Model

This is the most simple model that predicts the same rating for all movies without considering the users or any other effects. All the differences are explained by random variation.

```
mean <- round(mean(edx_train$rating), digits = 3)
print(paste("The mean value of the ratings is", mean))
```

```
## [1] "The mean value of the ratings is 3.541"
```

```
# compute rmse for average model
naive_rmse <- RMSE(edx_train$rating,mean)

# tabulate the results
rmse_results <- tibble(method = "Average Rating Model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

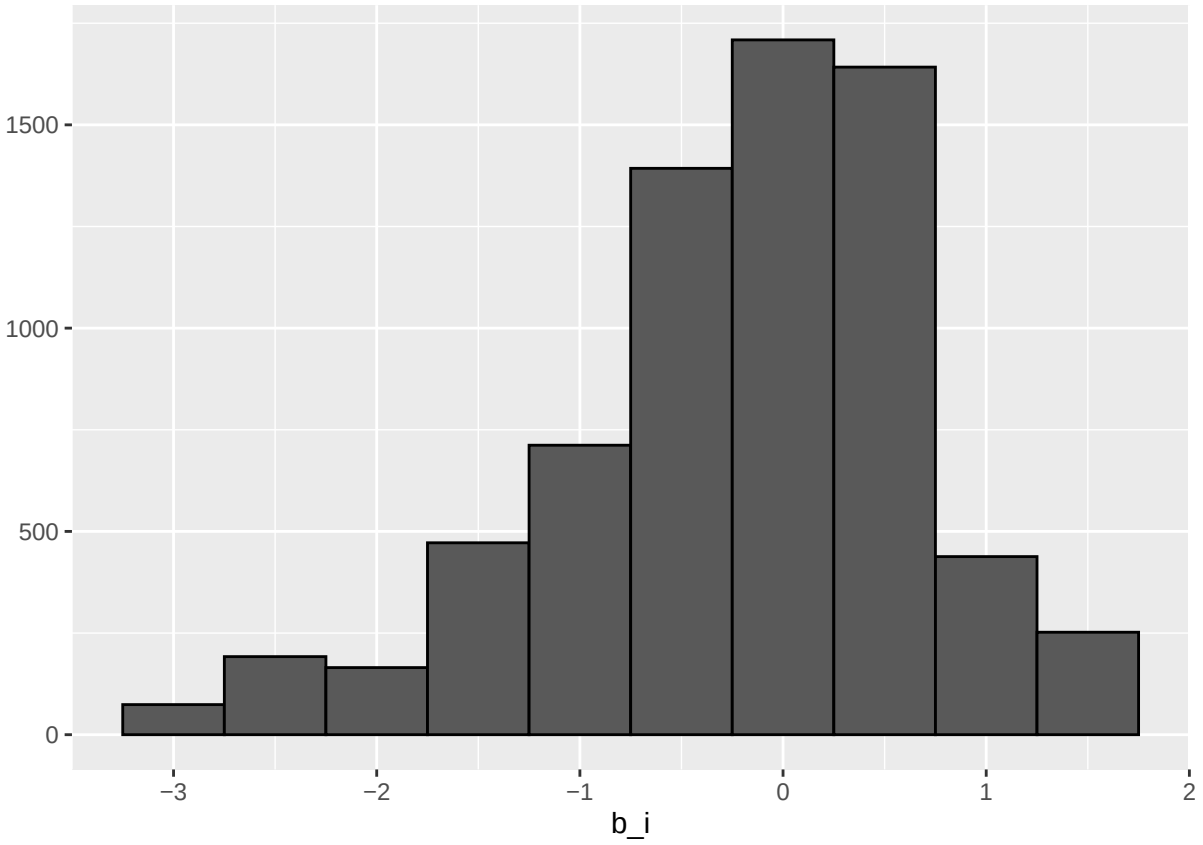
method	RMSE
Average Rating Model	1.056516

Model 2

Movie Effects

We have already seen how some movies get rated more often than others. We will call this 'Movie Effects'. This variation is accounted for by including it in our model.

```
# computing b for different movies
mean <- mean(edx_train$rating)
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mean))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```




```
# new predicted ratings after taking movie effects into consideration
predicted_ratings2 <- mean + edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

# find new rmse
new_rmse <- RMSE(predicted_ratings2, edx_test$rating)

# tabulate new results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effects",
                                     RMSE = new_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Average Rating Model	1.056516
Movie Effects	1.008392

We find that there is a slight improvement in the accuracy of our model (decrease in RMSE value).

Model 3

Movie & User Effects

Next we will take into consideration the variation introduced due to the differences in rating patterns among various users - the User Effects. We will include this when calculating RMSE and hence look at the combined effect of movies and users on RMSE. We can represent the model like this :

$$r_{u,m} = \mu + \beta_m + \beta_u + \epsilon_{u,m}$$

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

We will apply this model to the `test` set and compute the new RMSE. The model can predict values that lie outside the possible range of rating values (0.5 to 5). So predictions above(>5) or below(<0) the interval are given values of 5 or 0.5 respectively.

```
# construct predictors to see improvement in rmse
predicted_ratings3 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mean + b_i + b_u) %>%
  .$pred

# ensure the predictions don't exceed the rating limits
predicted_ratings3[predicted_ratings3<0.5] <- 0.5
predicted_ratings3[predicted_ratings3>5] <- 5

# calculating rmse
newer_rmse <- RMSE(predicted_ratings3, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie & User-Effects Model",
                                 RMSE = newer_rmse))
```

The results are tabulated :

method	RMSE
Average Rating Model	1.0565162
Movie Effects	1.0083917
Movie & User-Effects Model	0.9308244

Clearly, there is an improvement when considering both user and movie bias.

Model 4

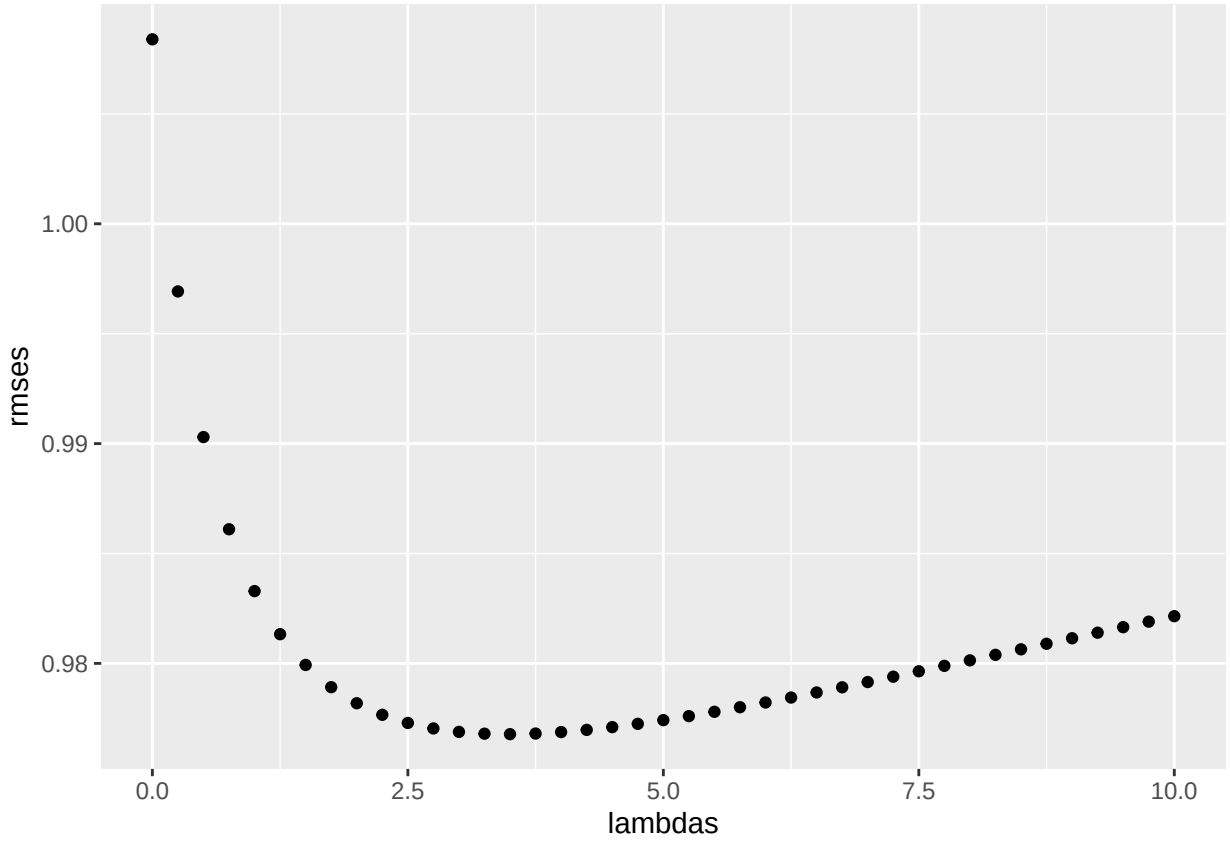
Regularized Movie Effects Model

In the fourth model, the aspect of *regularization* is used. Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. This penalty term controls the excessively fluctuating function in such a way that the coefficients don't take extreme values. That is, it shrinks them towards zero.

Here a tuning parameter - *lambda* - is used.

We will apply this concept to Movie Effects.

```
# plot rmse vs lambda
# using cross-validation for finding lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

  mean <- mean(edx_train$rating)

  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mean)/(n()+l))
   predicted_ratings <-
    edx_test %>%
    left_join(b_i, by = "movieId") %>%
    mutate(pred = mean + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, edx_test$rating))
})
qplot(lambdas, rmses)
```



We will now pick the right value of lambda and then calculate the RMSE :

```
# picking the right lambda value
lambda <- lambdas[which.min(rmses)]
print(paste("The value of lambda is ",lambda))
```

```
## [1] "The value of lambda is  3.5"
```

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized movie effect model",
                                 RMSE = min(rmses)))
```

We get the this :

method	RMSE
Average Rating Model	1.0565162
Movie Effects	1.0083917
Movie & User-Effects Model	0.9308244
Regularized movie effect model	0.9767802

There is a slight decrease in accuracy.

Model 5

The Regularized Movie & User Effects Model

This time we will include even user effects in the regularized model, to try and obtain a lower value of RMSE.

```
lambdas <- seq(0, 10, 0.25)
rmses1 <- sapply(lambdas, function(l){

  mean <- mean(edx_train$rating)

  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mean)/(n()+l))
  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mean)/(n()+l))
  predicted_ratings <-
    edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mean + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, edx_test$rating))
})
```

Cross-validation is used to pick the correct value of lambda.

```
lambda <- lambdas[which.min(rmses1)]
print(paste("The value of lambda now is ",lambda))
```

```
## [1] "The value of lambda now is  4"
```

The RMSE now is,

method	RMSE
Average Rating Model	1.0565162
Movie Effects	1.0083917
Movie & User-Effects Model	0.9308244
Regularized movie effect model	0.9767802
Regularized Movie & User effect model	0.8941680

As with Movie & User Effects, there is a decrease in the value of RMSE.

Model 6

User Based Collaborative Filtering

Collaborative filtering (CF) is a technique used by recommender systems. The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue, than that of a randomly chosen person. It takes account of the information about different users. The word "collaborative" refers to the fact that users collaborate with each other to recommend items. The starting point is a rating matrix in which rows correspond to users and columns correspond to items. Each user rates some movies. It is highly probable that many movies remain 'un-rated'. These take the form of missing values in the matrix. The goal of CF is to fill up these missing entries with predicted values.

There are two main methods of implementing Collaborative Filtering :

1. Item-Based CF :

Given a new user, the algorithm considers the user's purchases and recommends similar items. The core algorithm is based on these steps:

1. For each two items, measure how similar they are in terms of having received similar ratings by similar users
2. For each item, identify the k-most similar items
3. For each user, identify the items that are most similar to the user's purchases

2. User-Based CF :

Similarly,

1. Look for users who share the same rating patterns with the active user (the user whom the prediction is for).
2. Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user.

In this report we will explore and implement UBCF.

User Based Collaborative Filtering

User-Based Collaborative Filtering is a technique used to predict the items that a user might like on the basis of ratings given to that item by the other users who have similar taste with that of the target user.

Steps involved :

Step 1: Finding the similarity of users to the target user U :

For a user-movie rating matrix M , the **Cosine Similarity** for any two users A and B can be calculated from the given formula,

$$sim(A,B) = \frac{\sum_p (r_{ap} - \overline{r_a})(r_{ab} - \overline{r_b})}{\sqrt{\sum (r_{ap} - \overline{r_a})^2}\sqrt{\sum (r_{ap} - \overline{r_a})^2}}$$

where r_{up} is the rating of user U against item p.

Step 2: Prediction of missing rating of an item :

Now, the target user might be very similar to some users and may not be much similar to the others. Hence, the ratings given to a particular item by the more similar users should be given more weightage than those given by less similar users and so on. This problem can be solved by using a weighted average approach. In this approach, you multiply the rating of each user with a similarity factor calculated using the above mention formula.

The missing rating can be calculated as,

$$r_{up} = \overline{r}_u + \frac{\sum_{i \in users} sim(u,i) * r_{ip}}{\sum_{i \in users} |sim(u,i)|}$$

The `edx` data set is huge. It is not possible to implement UBCF just like that. Therefore we make use of the `Recommenderlab` package. We first create a copy of the `edx` data set to train the model. A sparse-matrix containing the `userId`, `movieId` and `ratings` is created. This is then converted to a `realRatingMatrix` as required by the `Recommenderlab` package.

It is observed that there are users who have rated only a few movies and movies that have been viewed a few times. The ratings associated with these users and movies might most likely be biased. Hence it is better to include only those users and movies which have a fair number of ratings. (We choose the top 10%)

After all these steps are completed, we apply UBCF.

```
# Training the model
# create a copy of edx
edx_new <- edx

# converting userID and movieID to factors
edx_new$userId <- as.factor(edx_new$userId)
edx_new$movieId <- as.factor(edx_new$movieId)

# converting userid and movieid to numeric vectors to make use of sparse-matrix function
edx_new$userId <- as.numeric(edx_new$userId)
edx_new$movieId <- as.numeric(edx_new$movieId)

sparse_ratings <- sparseMatrix(i = edx_new$userId,j = edx_new$movieId ,x = edx_new$rating,
                               dims = c(length(unique(edx_new$userId)),
                                        length(unique(edx_new$movieId))),
                               dimnames = list(paste("u", 1:length(unique(edx_new$userId)), sep = ""),
                                               paste("m", 1:length(unique(edx_new$movieId)), sep = "")))
#we dont need the copy
rm(edx_new)

#Convert rating matrix into a recommenderlab sparse matrix
```

```
ratingMatrix <- new("realRatingMatrix", data = sparse_ratings)
ratingMatrix
```

```
## 69878 x 10677 rating matrix of class 'realRatingMatrix' with 9000055 ratings.
```

```
# selecting appropriate users and movies
min_movies <- quantile(rowCounts(ratingMatrix), 0.9)
min_users <- quantile(colCounts(ratingMatrix), 0.9)


ratings_movies <- ratingMatrix[rowCounts(ratingMatrix) > min_movies,
                               colCounts(ratingMatrix) > min_users]
ratings_movies
```

```
## 6978 x 1068 rating matrix of class 'realRatingMatrix' with 2313148 ratings.
```

To train and test the model, we will split it. This is done using the `evaluationScheme function.` Prediction is then done on the 30% of data reserved for testing.

```
# 5 ratings of 30% of users are excluded for testing
set.seed(1)
e <- evaluationScheme(ratings_movies, method="split", train=0.7, given=-5)

# using UBCF
model <- Recommender(getData(e, "train"), method = "UBCF",
                     param=list(normalize = "center", method="Cosine", nn=50))
prediction <- predict(model, getData(e, "known"), type="ratings")
rmse_ubcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]

# adding this rmse value to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User Based Collaborative Filtering",
                                 RMSE = rmse_ubcf))
```

The results are :

method	RMSE
Average Rating Model	1.0565162
Movie Effects	1.0083917
Movie & User-Effects Model	0.9308244
Regularized movie effect model	0.9767802
Regularized Movie & User effect model	0.8941680
User Based Collaborative Filtering	0.8275340

There is a drastic increase in accuracy. The value of RMSE falls from 0.977 to 0.827534 .

As is seen, User Based Collaborative Filtering is a very good approach with high accuracy.

7 Conclusion

We have seen seven models for calculating the RMSE of the predicted values. The first was a the most basic form - just the Average Rating Model. This gave a very high value of RMSE (1.0565162). With the second model, Movie Effects,there was a slight decrease in RMSE (1.0083917). But it was not satisfactory. The third model, Movie & User Effects showed a noticeable improvement (RMSE : 0.9308244). Regularization seemed to have little effect when applied on movie-effects alone : the fourth model Regularized Movie Effects Model gave an RMSE of 0.977. When including even user effects, the regularized model performed better - The Regularized Movie & User Effects Model - RMSE : 0.894.

A drastic improvement in performance was brought about by User Based Collaborative Filtering. It showed the lowest RMSE (0.827534).

UBCF has a couple of disadvantages :

1. *Cannot handle fresh items :*

   The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. This issue is often called the **cold-start problem**.

   However, the following techniques can address the cold-start problem to some extent :

   1) Projection in WALS. Given a new item i_0 not seen in training, if the system has a few interactions with users, then the system can easily compute an embedding v_{i0} for this item without having to retrain the whole model. The system simply has to solve the following equation or the weighted version:

   $$min_{v_{i_0} \in R^d} ||A_{i_0} - U_{v_{i_0}}||$$

   2) Heuristics to generate embeddings of fresh items. If the system does not have interactions, the system can approximate its embedding by averaging the embeddings of items from the same category, from the same uploader (in YouTube), and so on.

2. *Side features* are any features beyond the query or item ID. For movie recommendations, the side features might include country or age. Including available side features improves the quality of the model.

However, it offers some advantages as well :

1. The model can help users discover new interests. In isolation, the machine learning system may not know the user is interested in a given item, but the model might still recommend it because similar users are interested in that item.

2. To some extent, the system needs only the feedback matrix to train a matrix factorization model. In particular, the system doesn't need contextual features. In practice, this can be used as one of multiple candidate generators.

8 References

[1] Rafael A. Irizarry, 2019, *Introduction to Data Science Data Analysis and Prediction Algorithms with R*

[2] Suresh, K. G, Michele, U., 2015, *Building a Recommendation System with R*

[3] https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b

[4] https://www.rdocumentation.org/packages/recosystem/versions/0.3