Building Recommendation System using Machine Learning

By: Sherif Allam

Contents

1 Introduction

This document aims to explain building Recommendation Systems using Machine Learning algorithms.

Recommendation systems use historical ratings that users have given items to make specific recommendations for a Particular user. Companies that sell many products to many customers and let these customers to rate their products, like Amazon, are able to collect massive historical ratings datasets 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.

To explain how Machine Learning algorithms could be used in building Recommendation Systems. First of all, it is required to have an applied case with historical rating data.

1.1 Goal of the Project

The Project goal is to indicate how machine learning could be very valuable and accurate solution for predicting the rate that customer will give to the product and to show how close is the predicted rate to the actual rate given by the customer.

For our project purpose, Movies recommendation system has been chosen as an applied case and MovieLens dataset that contains millions of records for users' ratings for different movies is selected as a case study dataset. This dataset will be used to train selected Machine Learning algorithms on how users rate movies and to discover the pattern of rating then the same discovered pattern will be used to predict rating on another dataset with known actual rating. Hence, accuracy of selected Machine Learning algorithms will be measured by comparing predicted rating to the actual rating.

1.2 Dataset Overview

Before start training the selected Machine Learning algorithms on the MovieLens dataset. It is helpful to describe our data and discover any patterns that could help in customizing Machine Learning algorithms to fit our data correctly.

For the purpose of that Project MovieLens dataset will be used. The entire latest MovieLens dataset could be found here. To make it easy to reuse the provided sample codes on any PC with medium specification, a 10M record version of the MovieLens dataset will be used instead of the full version.

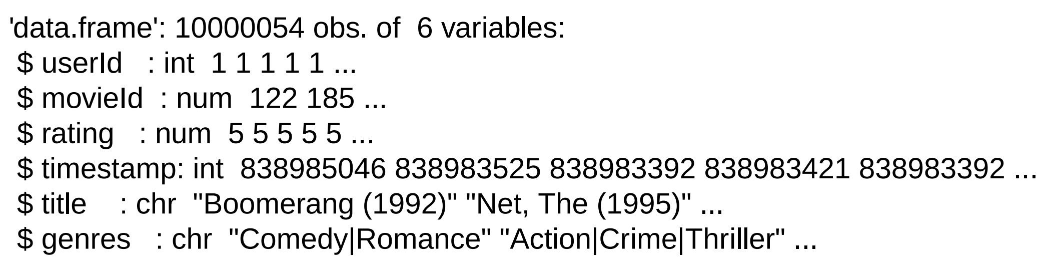
Let us now start by downloading and Loading MovieLens dataset to R environment.

```r
#################################
# Downalod movielens Dataset
#################################
#
# # Note: this process could take several minutes
# # Data saved to a "movielens.rda" file after some manpulations
# # Code checks if the file is "movielens_year.rda" is alrady exsit,
# # then no need to redownload data
# # MovieLens 10M dataset:
# # https://grouplens.org/datasets/movielens/10m/
# # http://files.grouplens.org/datasets/movielens/ml-10m.zip
#
# Check if there is a local copy of the data or shall we start downlaoding

if(!file.exists("movielens.rda")){

    dl <- tempfile()

    download.file(
    "http://files.grouplens.org/datasets/movielens/ml-10m.zip"
    , dl)
    # Read ratings
    ratings <- fread(text = gsub("::", "\t",
    readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
    col.names = c("userId", "movieId", "rating", "timestamp"))
    # Read Movies
    movies <- str_split_fixed(readLines(unzip(dl,
    "ml-10M100K/movies.dat")), "\\::", 3)
    # Name Columns
    colnames(movies) <- c("movieId", "title", "genres")
    # Create movies dataframe and Add movieId
    movies <- as.data.frame(movies) %>%
    mutate(movieId = as.numeric(levels(movieId))[movieId],
    title = as.character(title),
    genres = as.character(genres))
    # Create movielens dataframe which has the full data
    movielens <- left_join(ratings, movies, by = "movieId")
    # save a loacal copy
    save(movielens,file="./movielens.rda")
}else
{
  # Load data from local file
  load("movielens.rda")
}
```

```
# General Note: For better memory management, and to avoid re-run algorithms Iterations
# again and again. I have saved each procedure output in a data file.
# Before running the procedure, I check if the output already saved
# on local machine. Hence, no need to re-run code again and
# the required output shoud be loaded from saved file.
```

As we have now Movielens dataset loaded into our environment, we can explore the dataset columns:

```
# # Explore dataset structure
print_output(str(movielens, 2, 2), 2.5)
```



```
'data.frame': 10000054 obs. of  6 variables:
 $ userId   : int  1 1 1 1 1 ...
 $ movieId  : num  122 185 ...
 $ rating   : num  5 5 5 5 5 ...
 $ timestamp: int  838985046 838983525 838983392 838983421 838983392 ...
 $ title    : chr  "Boomerang (1992)" "Net, The (1995)" ...
 $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" ...
```

The Movielens dataset has six columns as the following:

- userId: A unique Id for the user who rated the movie

- movieId: A unique Id for each movie
- rating: Movie rating from 0 to 5
- timestamp: When the rating was given
- title: Movie Name
- genres: Combined Movie genre like Comedy, Romance, Action…etc separated by '|' character

We can see some sample Data as the following:

```r
# # Explore dataset rows
kable(head(movielens), "latex", booktabs = T) %>% kable_styling(latex_options = "scale_down")
```

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	231	5	838983392	Dumb & Dumber (1994)	Comedy
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

Each row represents a rating given by one user to one movie. We can see the number of unique users that provided ratings and how many unique movies were rated:

```r
# # Check number of unique users and movies
movielens %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

If we multiply those two numbers, we get a number larger than 10 million, yet our data table has about 10,000,000 rows. This implies that not every user rated every movie. So we can think of these data as a very large matrix, with users on the rows and movies on the columns, with many empty cells. The gather function permits us to convert it to this format, but if we try it for the entire matrix, it will crash R. Let's show the matrix for random seven users and five movies.

```r
# # Spread movies titles to columns for random selected mvoies and users
set.seed(1)
movielens_7 <- movielens %>% filter(movieId %in% sample(movielens$movieId, 7) &
    userId %in% sample(movielens$userId, 7)) %>% select(userId, rating, title) %>%
    spread(title, rating)
kable(movielens_7, "latex", booktabs = T) %>% kable_styling(latex_options = "scale_down")
```

userId	City Slickers II: The Legend of Curly's Gold (1994)	Faculty, The (1998)	Nightmare on Elm Street, A (1984)	Patriot, The (2000)	Saving Private Ryan (1998)	South Park: Bigger, Longer and Uncut (1999)
19732	NA	NA	NA	3.5	2.5	NA
32449	NA	NA	NA	3.5	5.0	NA
32651	NA	NA	NA	NA	4.0	5.0
38875	NA	NA	NA	NA	4.0	NA
41276	NA	NA	NA	NA	4.0	NA
51771	2	3	4	2.0	4.0	3.5
56734	NA	NA	NA	NA	4.5	4.5

```
rm(movielens_7)
```

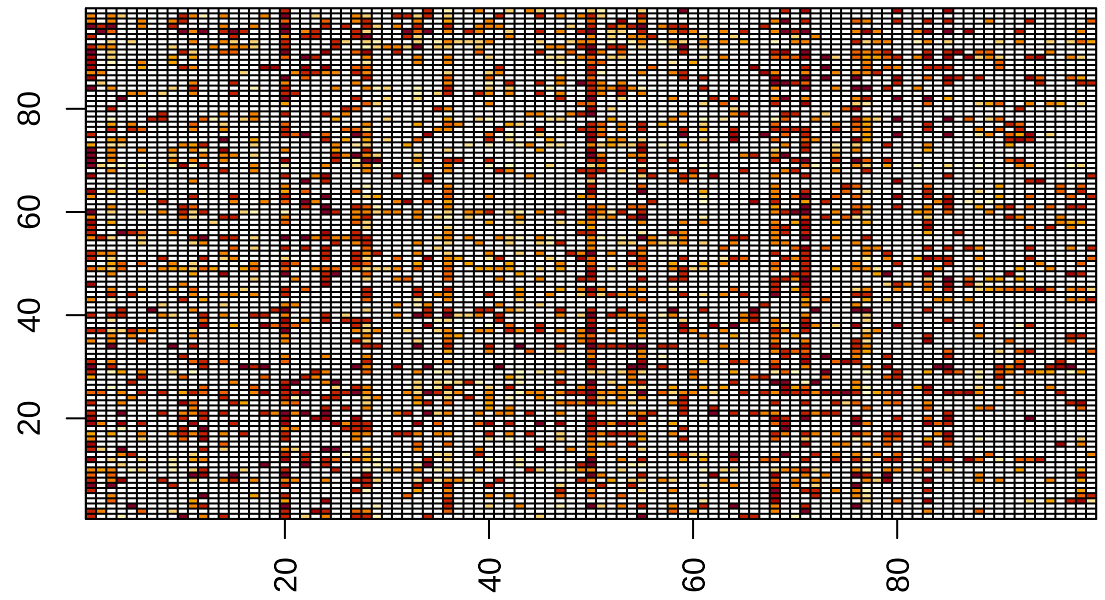
You can think of the task of a recommendation system as filling in the NAs in the table above.

To see how sparse the matrix is, here is an image for the matrix of a random sample of 100 movies and 100 users with red indicating a user/movie combination for which we have a rating.

```
# # Spread movies titles to columns for random selected 100 mvoies and 100
# users # Get sample 100 movies that were rated more than 500
set.seed(1)
movieId_list <- sample((movielens %>% group_by(movieId) %>% filter(n() > 500) %>%
    select(movieId) %>% distinct())[[1]], 100, replace = FALSE)
movieId_df <- as.data.frame(movieId_list)
colnames(movieId_df) <- c("movieId")
# # Get sample 100 user that give rating more than 500
userid_list <- sample((movielens %>% group_by(userId) %>% filter(n() > 500) %>%
    select(userId) %>% select(userId) %>% distinct())[[1]], 100, replace = FALSE)
userid_df <- as.data.frame(userid_list)
colnames(userid_df) <- c("userId")
# # Get common rows where random selected moviews were rated by randome
# selected users
intersect_df <- movielens %>% filter(movieId %in% movieId_list & userId %in%
    userid_list) %>% select(userId, rating, movieId)
# # Add un-common rows where selected movies were not rated by selected
# users
movielens_100 <- userid_df %>% left_join(intersect_df, by = "userId") %>% right_join(movieId_df,
    by = "movieId") %>% spread(movieId, rating)

rm(movieId_list, movieId_df, userid_list, userid_df, intersect_df)
```

After running the above code, we will have dataset, but we need to reshape it to a matrix form to display it as an image for illustrative purposes.

```
# # Convert dataset to a matrix
m_movielens_100 <- as.matrix(movielens_100)
# # Remove first column 'userId' and last row and column which are nothing
# rather than NA values
m_movielens_100 <- m_movielens_100[1:nrow(movielens_100) - 1, 2:(ncol(movielens_100) -
    1)]
# # Rename rows by userId
rownames(m_movielens_100) <- movielens_100[1:nrow(movielens_100) - 1, 1]
# # Convert rated values to TRUE and not rated to FALSE m_movielens_100 <-
# ifelse( !is.na (m_movielens_100) ,TRUE,FALSE )

# # Display matrix as an image
cols <- 1:ncol(m_movielens_100)
rows <- 1:nrow(m_movielens_100)

image(cols, rows, t(m_movielens_100[rev(rows), , drop = FALSE]), xaxt = "n",
    yaxt = "n", xlab = "", ylab = "")

abline(h = rows + 0.5, v = cols + 0.5)
axis(1, at = seq(0, 100, by = 20), las = 2)
axis(2, at = seq(0, 100, by = 20))
```



```r
rm(movielens_100, m_movielens_100)
```

Based on our intuition, we can expect also that not all movies have the same number of ratings and number of ratings is also affected by movie genres.

1.3 Summarize/ Key Steps

In brief, we will do a detailed analysis for the dataset to validate if our intuition that user, movie and genres are affecting the user rating to the movies or not. This validation will be done by predicting user ratings using the following Machine Learning algorithms :

- Regression
- Regression with Regularization
- Matrix Factorization
- Singular Value Decomposition

Next, we will measure each algorithm performance by using residual mean squared error (RMSE) on a test set.

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

- N: Number of user/movie combinations (Ratings)
- $\hat{y}_{u,i}$: Expected Ratings

- $y_{u,i}$: Actual Ratings

We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. The more smallest RMSE on the test dataset -less than one- means the our Prediction is more accurate.

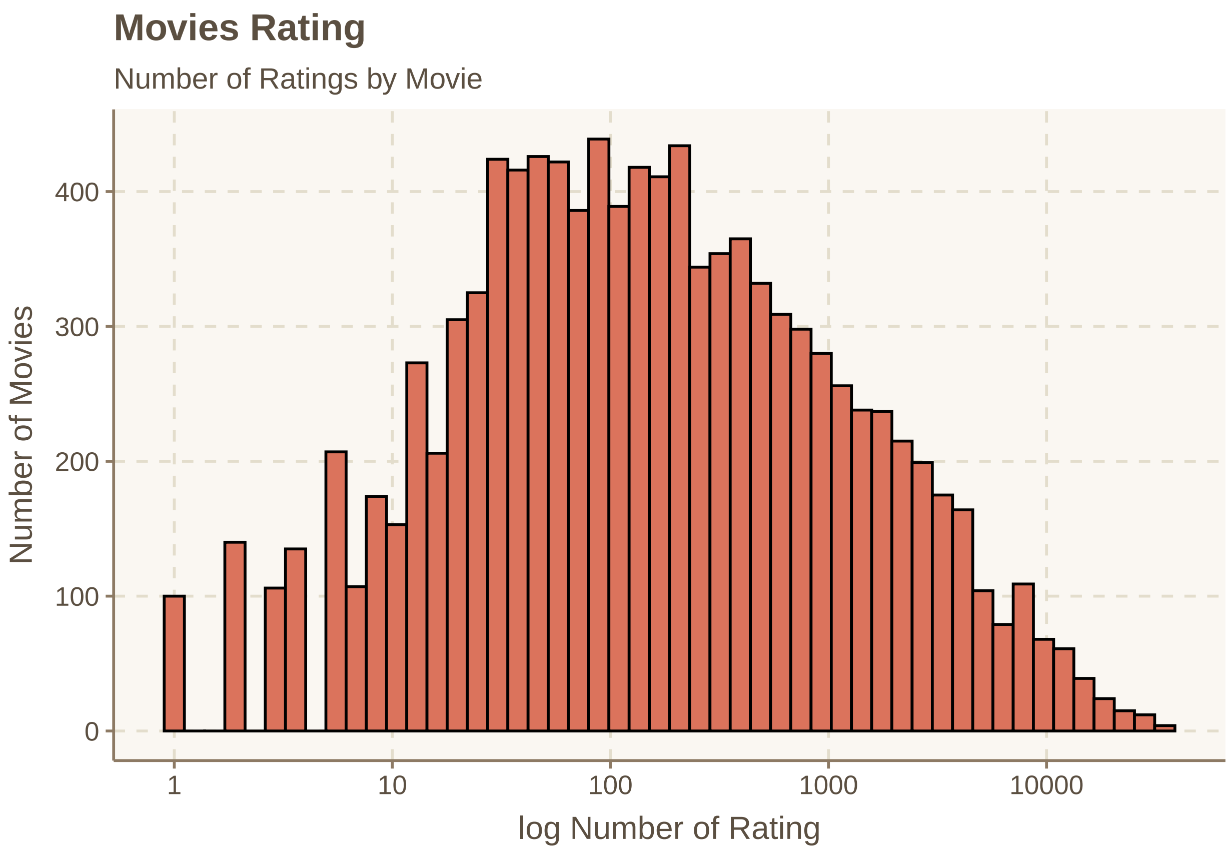
2 Methods/Analysis

As we discussed in the Introduction section, there are three main assumptions that we have assumed based on preliminary exploration of the dataset. Let us now take a deep dive to the data to validate our intuitions and assumptions.

2.1 Data Exploration/ Insights

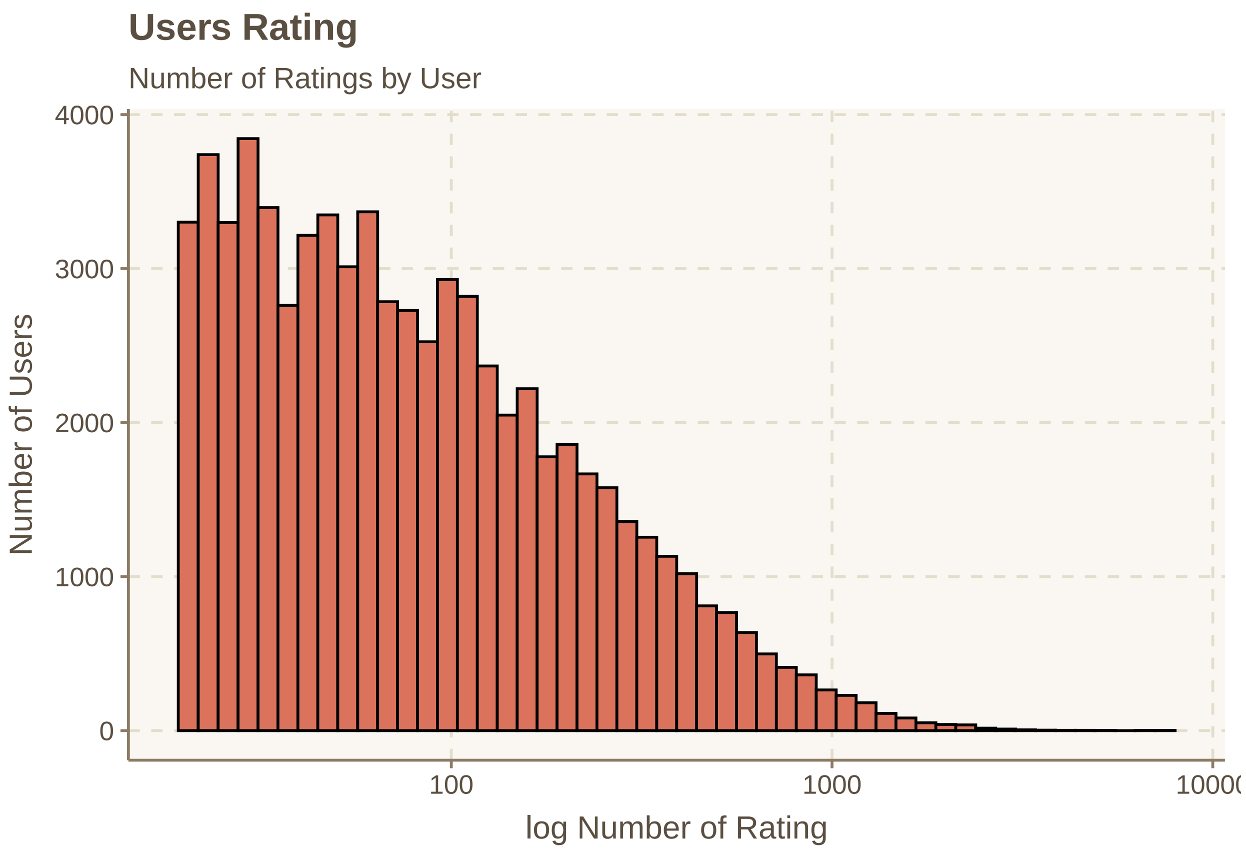
For more clarification about data characteristics let us see the distribution of movies. Distribution indicates how many time each movie was ranked.

```
# # Rating distribustion based on movies
movielens %>% group_by(movieId) %>% summarize(n_movie_rated = n()) %>% ggplot(aes(n_movie_rated)) +
    geom_histogram(bins = 50, color = "black") + scale_x_log10() + ggtitle("Movies Rating") +
    labs(subtitle = "Number of Ratings by Movie", x = "log Number of Rating",
        y = "Number of Movies")
```



The above graph is clearly saying that some movies get rated more than others. On the other hands let us check users behavior on rating movies.

```
# # Rating distribustion based on users
movielens %>% group_by(userId) %>% summarize(n_user_rated = n()) %>% ggplot(aes(n_user_rated)) +
    geom_histogram(bins = 50, color = "black") + scale_x_log10() + ggtitle("Users Rating") +
    labs(subtitle = "Number of Ratings by User", x = "log Number of Rating",
        y = "Number of Users")
```



The above graph clearly saying that some users are more active than others.

Let us also check if the movie genres has an impact on the number of rating movies got or not.

Each movie in MovieLens dataset has multiple genres. To simplify our task, let us assign two genres to each movie. The selected genres is genres with biggest number of rating.

First, we need to identify unique list of genres. That list could be obtained using the following code.

```
# # Get unique genres list
combiend_genres <- movielens %>% distinct(genres)

genres_list <- lapply(combiend_genres[[1]], function(x) {

    as.vector(strsplit(as.character(x), "\\|"))[[1]]

})
genres_list <- unlist(genres_list)
Uniqe_genres_list <- genres_list[-which(duplicated(genres_list))]

rm(combiend_genres)
```

Hence, we obtained the list of genres: Comedy, Romance, Action, Crime, Thriller, Drama, Sci-Fi, Adventure, Children, Fantasy, War, Animation, Musical, Western, Mystery, Horror, Film-Noir, Documentary, IMAX, (no genres listed)

let us now sort that list in descending order based on number of ratings. To sort the genres list from genres which got the biggest number of rating to smallest number, we need to know the number of rating for each genres as the following:

```r
# # Order unique genres list based on highest ratings count
if(!file.exists("movielens_year.rda")){
genres_count <- sapply(Uniqe_genres_list, function(x){
  movielens %>% filter(genres %like% x) %>% summarise(n=n())
})

genres_count_df <- data.frame(genres=names(genres_count),
                              genres_count=matrix(unlist(genres_count),
                                                  nrow=length(genres_count), byrow=T))

genres_count_df <- genres_count_df %>%arrange(desc(genres_count))

Uniqe_genres_list_Orderd <- substr(genres_count_df[,1], 1,
nchar(as.character(genres_count_df[,1]))-2)

save(Uniqe_genres_list_Orderd, file = "./Uniqe_genres_list_Orderd.rda")

}else {load("Uniqe_genres_list_Orderd.rda")}
```

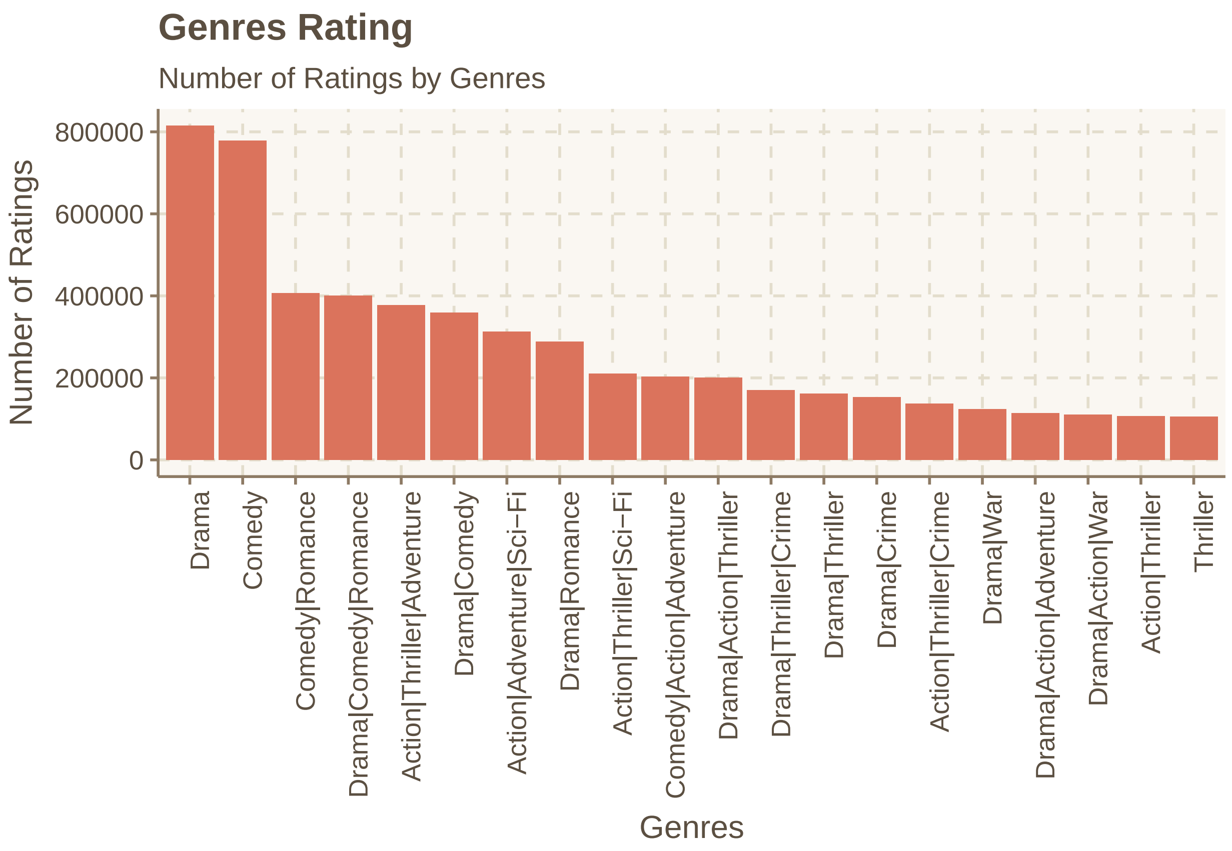
Now, we have the genres list sorted in descending order based on the number of ratings. Drama, Comedy, Action, Thriller, Adventure, Romance, Sci-Fi, Crime, Fantasy, Children, Horror, Mystery, War, Animation, Musical, Western, Film-Noir, Documentary, IMAX, (no genres listed)

Next, is to map each movie to the two top most rated genres, we will select two genres from each movie genres that have biggest number of rating using the following code.

```r
# # Add new column contains 3 common genres for each movie

if (!file.exists("movielens_year.rda")) {
    l <- length(Uniqe_genres_list_Orderd)
    movielens_genre <- movielens %>% mutate(three_major_genres = sapply(genres,
        function(x) {
            n <- 0
            three_major_genres <- ""
            for (i in 1:l) {
                if (grepl(Uniqe_genres_list_Orderd[[i]], x) == TRUE) {
                  n <- n + 1
                  three_major_genres <- paste(three_major_genres, as.character(Uniqe_genres_list_Orderd[[i]])
                    sep = "|")
                  if (n == 3) {

                    break
                  }

                } else {
                  next
                }
            }
            three_major_genres <- substr(three_major_genres, 2, nchar(three_major_genres))
            return(three_major_genres)
        }))
} else {
    assign("movielens_genre", get(load("movielens_year.rda")))
}
```

Now, let us see the distribution of genres based on number of rating

```
# # Group movies abased on 3 main genres for each movie

major_genres_df <- movielens_genre %>%
group_by(three_major_genres) %>%
summarize(n_genres_rated = n()) %>%
arrange(desc(n_genres_rated))


# # Rating distribustion based on 3 main genres
major_genres_df %>%
filter(n_genres_rated > 100000)%>%
ggplot(aes( x = reorder(three_major_genres,- n_genres_rated),y = n_genres_rated)) +
geom_bar(stat = "identity") +
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
ggtitle("Genres Rating") +
labs(subtitle ="Number of Ratings by Genres", x="Genres" , y="Number of Ratings")
```



Clear memory from unused objects.

```r
rm(major_genres_df, Uniqe_genres_list_Orderd, genres_list)
```

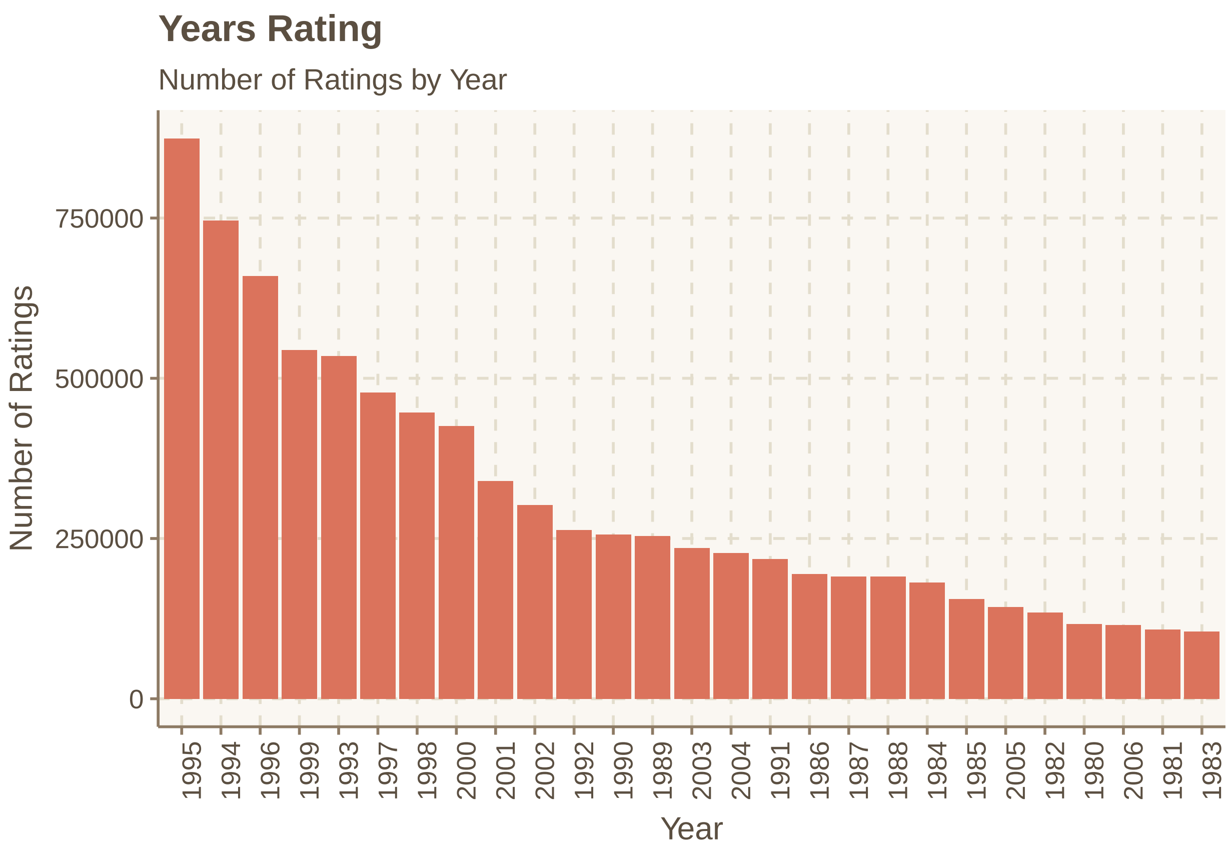
The above graph is clearly saying that some Genres have more ratings than others.

Let us also check if the movie year has an impact on the number of rating movies got or not.

Each movie in MovieLens dataset has a release year as a part of Movie Title. To simplify our task, let us separate the year in a dedicated column. .

```r
# # Add new column contains Release year for each movie
if (!file.exists("movielens_year.rda")) {
    movielens_year <- movielens_genre %>% mutate(year = as.numeric(substr(title,
        nchar(title) - 4, nchar(title) - 1)))
} else {
    movielens_year <- movielens_genre
}
```

```r
# # Group movies abased on Release year
major_year_df <- movielens_year %>% group_by(year) %>% summarize(n_year_rated = n()) %>%
    arrange(desc(n_year_rated))


# # Rating distribustion based on on Release year
major_year_df %>% filter(n_year_rated > 100000) %>% ggplot(aes(x = reorder(year,
    -n_year_rated), y = n_year_rated)) + geom_bar(stat = "identity") + theme(axis.text.x = element_text(angle
    hjust = 1)) + ggtitle("Years Rating") + labs(subtitle = "Number of Ratings by Year",
    x = "Year", y = "Number of Ratings")
```



```r
rm(major_year_df)
```

The above graph is clearly saying that some Years have more ratings than others.

Now, let us save our work.

```r
# # Save movielens dataset after adding genres and year
if (!file.exists("movielens_year.rda")) {
    save(movielens_year, file = "./movielens_year.rda")
}
```

3 Algorithms Training, Validation and Prediction

After we have a deep look to dataset insists, we can start train our selected algorithms on the training dataset and make a prediction on the test dataset

3.1 Building Training Dataset/ Test Dataset

Before starting training out Machine Learning Algorithms, let us first start by separating our Movieslens dataset to training and test data. edx will be the name for training dataset that contains 90% of the records and test dataset will be called validation and will have 10% of the records.

```r
# Validation set will be 10% of MovieLens data
if (!file.exists("edx.rda")) {
    set.seed(1)

    test_index <- createDataPartition(y = movielens_year$rating, times = 1,
        p = 0.1, list = FALSE)
    edx <- movielens_year[-test_index, ]
    temp <- movielens_year[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") %>% semi_join(edx, by = "three_major_genres") %>% semi_join(edx,
        by = "year")


    # Add rows removed from validation set back into edx set

    removed <- anti_join(temp, validation)
    edx <- rbind(edx, removed)

    save(edx, file = "./edx.rda")
    save(validation, file = "./validation.rda")

    rm(test_index, temp, movielens_year, movielens_genre, movielens, removed)

} else {
    load("edx.rda")
    load("validation.rda")
    rm(movielens_year, movielens_genre, movielens)
}
```

3.2 Building Cross Validation and Test datasets

The main pain for all Machine Learning algorithms is overfitting, it happened when algorithm works with good performance on training set. However, when algorithm is tested on data that were not seen before, the performance got decreased heavily.

To avoid that, we will not use validation set in building or measuring performance. It will be used only for final prediction.

Before proceeding with building training and validation dataset. Let us remember that:

- edx : main training dataset have around 9 million records
- validation: final prediction set have around 1 million records

While building or measuring performance,we will use cross validation method. In simple words:

- We will divide training set 'edx' to 2 random sets or folds. One set for training and another set for cross-validation.
- Will repeat the above process for 9 times.
- On each time, the training set will have around 8 million records and cross-validation set will have around 1 million records.
- As we will have 9 random training and 9 random cross-validation sets, each algorithm will be evaluated 9 times on the cross-validation sets.
- We will Pick the best performance from the 9 tries.
- We will use the training set that gives best performance, to build final algorithm.
- Predict and check the performance on test dataset called 'validation'

```r
# # The following code will do these steps:

# - We will divide training set 'edx' to 2 random sets or folds.  One set
# for training and another set for cross-validation

#- Will repeat the above process for 9 times.

#- On each time, the training set will have around 8 milion records and
# cross-validation set will have around 1 milion records.
if (!file.exists("Training_Index_List.rda")) {
    folds = createFolds(edx$rating, 9, list = TRUE, returnTrain = TRUE)
}
#- folds: returing parameter is a list of 9 lists. Each list contains
# a vector of 8 milion random indicies
```

As the above process is done randomly, we may face an issue that the created Training sets may not have all movies, users, genres or years on cross-validation sets. Or, they may not have also all movies, users, genres or years on final prediction set.

To handle that issue, we will do the following:

- If the Training set doesn't have any of movies, users, genres or years that are rated in final prediction set. That means, these missing are there in random created cross-validation set.
- We will pull these training set missing records' from cross-validation set, and add them to training set randomly by percentage 8:1. That percentage is same to the percentage of number of records in training set to the number of records in cross-validation set.
- At this point, we have solved the issues that randomly created training sets may not have movies, users, genres or years that are rated in final prediction set.
- Then, we will check, If the cross-validation set has any moves, users, genres or years that are not rated in training set.
- We will solve that issue by removing these records from cross-validation set and adding them to the training sets.

The following code is performing the above steps:

```r
# # If the Training set doesn't have any of moives, users, genres or years
# that are rated in final prediction set. That means, these missing are
# there in random created cross-validation set.

#- We will pull these training set missing records' from cross-validation set,
# and add them to training set randomly by percentage 8:1.  That percentage
# is same to the percentage of number of records in training set to the
# number of records in cross-validation set.

#- At this point, we have solved the issues that randomly
# created training sets may not have moives, users, genres or years that are
# rated in final prediction set.

#- Then, we will check, If the cross-validation set has any moives,
# users, genres or years that are not rated in training set.

#- We will solve that issue by removing these records from cross-validation
# set and adding them to the training sets.

if (!file.exists("Training_Index_List.rda")) {

    n_loop <- seq(1, 9, 1)

    Training_Index_List <- sapply(n_loop, function(x) {

        training_set <- edx[folds[[x]], ]
        movie_rows <- validation %>% anti_join(training_set, by = "movieId")
        user_rows <- validation %>% anti_join(training_set, by = "userId")
        genres_rows <- validation %>% anti_join(training_set, by = "three_major_genres")
        year_rows <- validation %>% anti_join(training_set, by = "year")

        index_list <- c()

        if (nrow(movie_rows) > 0) {
            row_ind <- which(edx$movieId %in% as.factor(movie_rows$movieId))
            added_index <- sample(c(row_ind), size = round(0.88 * length(row_ind)),
                replace = FALSE)
            index_list <- append(index_list, added_index)
        }
        if (nrow(user_rows) > 0) {
            row_ind <- which(edx$userId %in% user_rows$userId)
            added_index <- sample(c(row_ind), size = round(0.88 * length(row_ind)),
                replace = FALSE)
            index_list <- append(index_list, added_index)
        }

        if (nrow(genres_rows) > 0) {
            row_ind <- which(edx$three_major_genres %in% genres_rows$three_major_genres)
            added_index <- sample(c(row_ind), size = round(0.88 * length(row_ind)),
                replace = FALSE)
            index_list <- append(index_list, added_index)
        }
        if (nrow(year_rows) > 0) {
            row_ind <- which(edx$year %in% year_rows$year)
            added_index <- sample(c(row_ind), size = round(0.88 * length(row_ind)),
                replace = FALSE)
            index_list <- append(index_list, added_index)
        }

        train_index <- append(folds[[x]], index_list)

        training_set <- edx[train_index, ]

        cross_validation_set <- edx[-train_index, ]


        movie_rows <- cross_validation_set %>% anti_join(training_set, by = "movieId")
        user_rows <- cross_validation_set %>% anti_join(training_set, by = "userId")
        genres_rows <- cross_validation_set %>% anti_join(training_set, by = "three_major_genres")
        year_rows <- cross_validation_set %>% anti_join(training_set, by = "year")

        if (nrow(movie_rows) > 0) {
            row_ind <- which(edx$movieId %in% as.factor(movie_rows$movieId))
            index_list <- append(index_list, row_ind)
        }
        if (nrow(user_rows) > 0) {
            row_ind <- which(edx$userId %in% user_rows$userId)
            index_list <- append(index_list, row_ind)
        }

        if (nrow(genres_rows) > 0) {
            row_ind <- which(edx$three_major_genres %in% genres_rows$three_major_genres)
            index_list <- append(index_list, row_ind)
        }
        if (nrow(year_rows) > 0) {
            row_ind <- which(edx$year %in% year_rows$year)
            index_list <- append(index_list, row_ind)
        }

        return(train_index <- append(train_index, index_list))
    })
    save(Training_Index_List, file = "./Training_Index_List.rda")
} else {
    load("Training_Index_List.rda")
}
```

3.3 Regression

3.3.1 Naive Model

In the first Regression model, let us try by neglecting movie, user, genres and year effect and we assume that the predicted rating is just the average rating for all movies as the following:

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

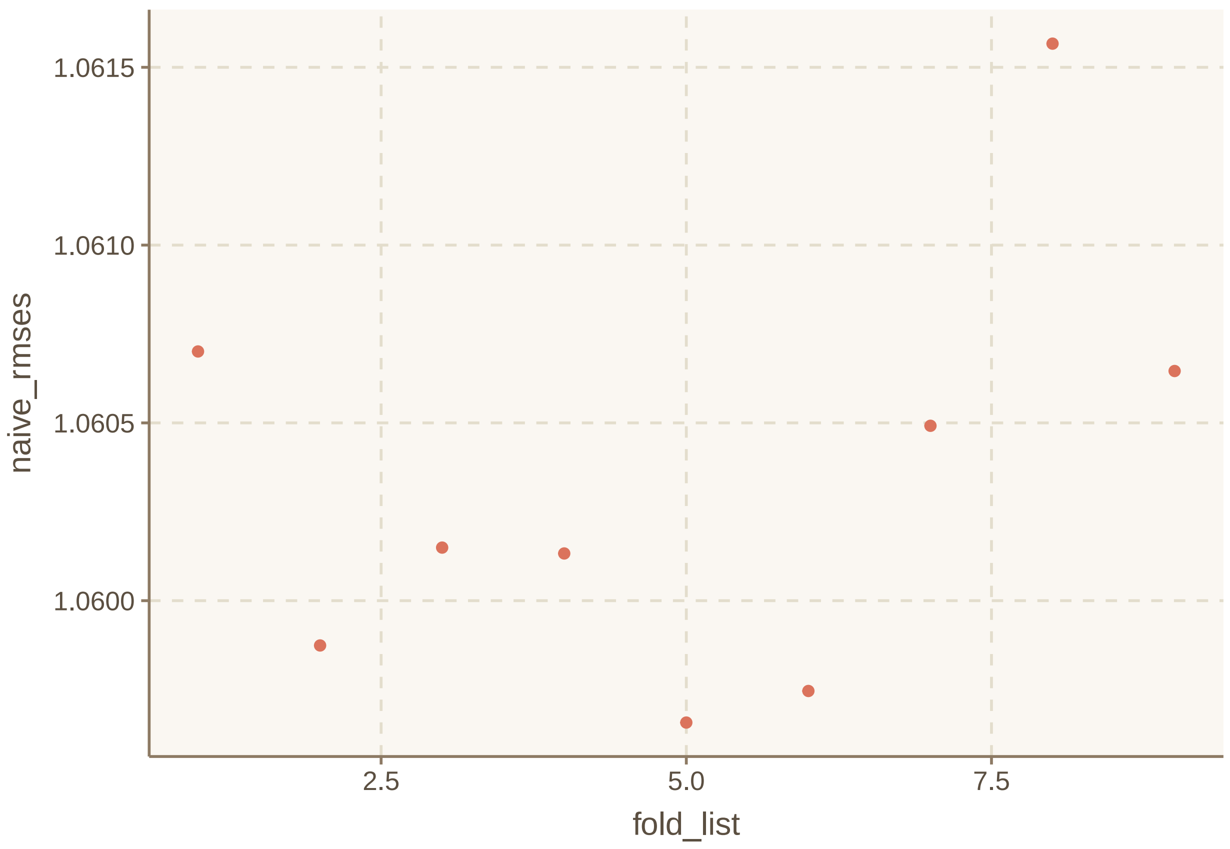
- μ: Is the average rating for all movies in training dataset.
- $\epsilon_{u,i}$: Is independent errors for each rating (difference between actual rating and predicted rating)

When it comes to measure the performance of our first try. By comparing predicted Rating to the actual rating on the test set. As explained, we will use RMSE to evaluate algorithm performance.

```r
# # Function the would be used to measure the performance
RMSE_Custom <- function(true_ratings, predicted_ratings) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Now, we can measure the first try performance on cross-validation sets as the following:

```r
fold_list <- seq(1, 9, 1)
if (!file.exists("naive_rmses.rda")) {
    # # Run the algorithm for 9 times on each training set and Pick best
    # performance on the cross-validation set
    naive_rmses <- sapply(fold_list, function(x) {

        # # Get Training set
        training_set <- edx[Training_Index_List[[x]], ]
        # # Get cross-validation
        cross_validation_set <- edx[-Training_Index_List[[x]], ]
        # # Get cross-validation
        mu <- mean(training_set$rating)
        # # Measure Performance against cross-validation set
        RMSE_Custom(cross_validation_set$rating, mu)

    })
    save(naive_rmses, file = "./naive_rmses.rda")
} else {
    load("naive_rmses.rda")
}
# # Plot training set number against performance
qplot(fold_list, naive_rmses)
```



```
# # Pick Number of the training set that achieve the best performance
fold_n <- fold_list[which.min(naive_rmses)]
```

let us see the performance on the test dataset

```
# # Get the Training set that achieve the best performance
training_set <- edx[Training_Index_List[[fold_n]], ]
# # Run algorithm against validation set
mu <- mean(training_set$rating)
# # Measure performance against validation set
result <- RMSE_Custom(validation$rating, mu)
# # Print results
result
## [1] 1.0612
```

Now, let us create a table to save our RMSE results for each algorithm.

```
rmse_pesults_tb <- NULL
rmse_pesults_tb <- tibble(method = "Just the average", RMSE = result)
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.0612

3.3.2 Movie Effect

To address movie effect, we will simply update our initial equation for estimate using average to take into consideration the movie effect. Movie effect is the difference between the average movie rate and standard average rate for all movies. Hence we can update our equation as the following:

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

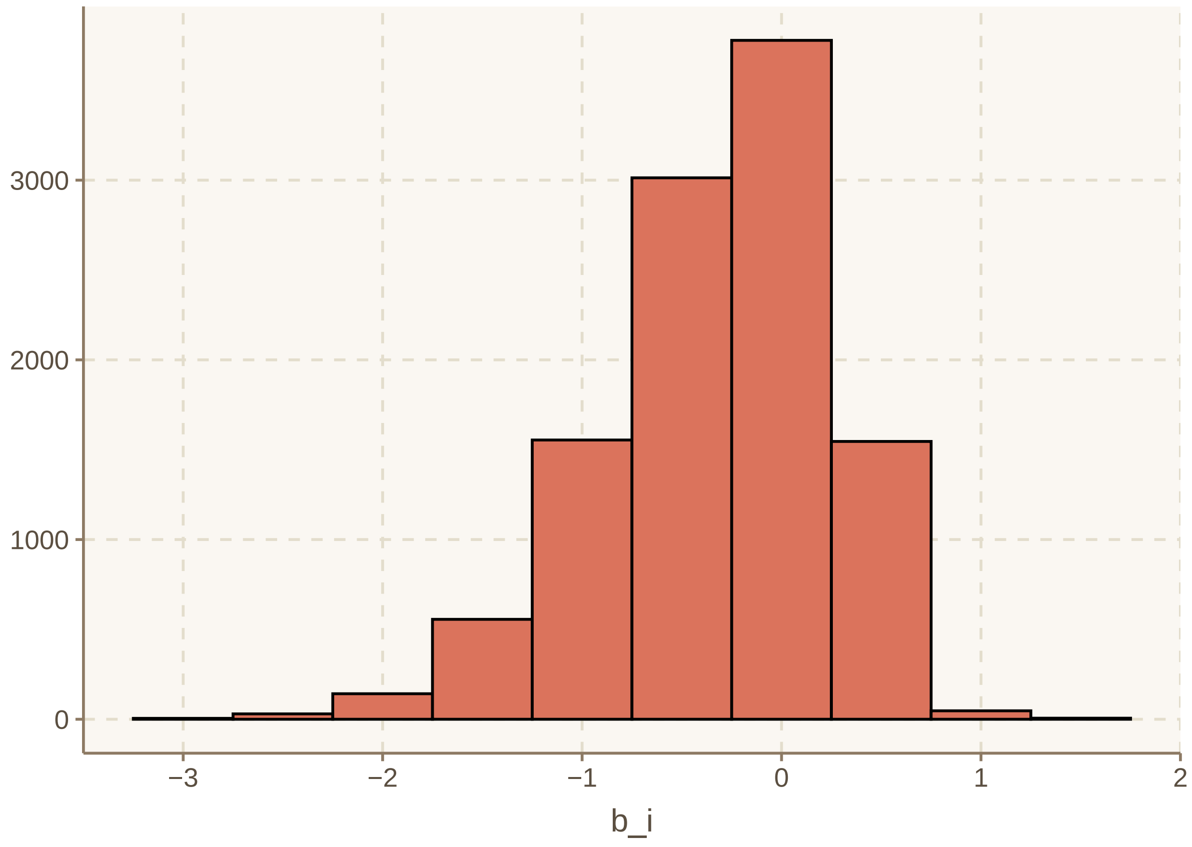
- b_i: Is the the difference between the average movie rate and standard movie rate.

Let us now do the calculation for each movie as the following:

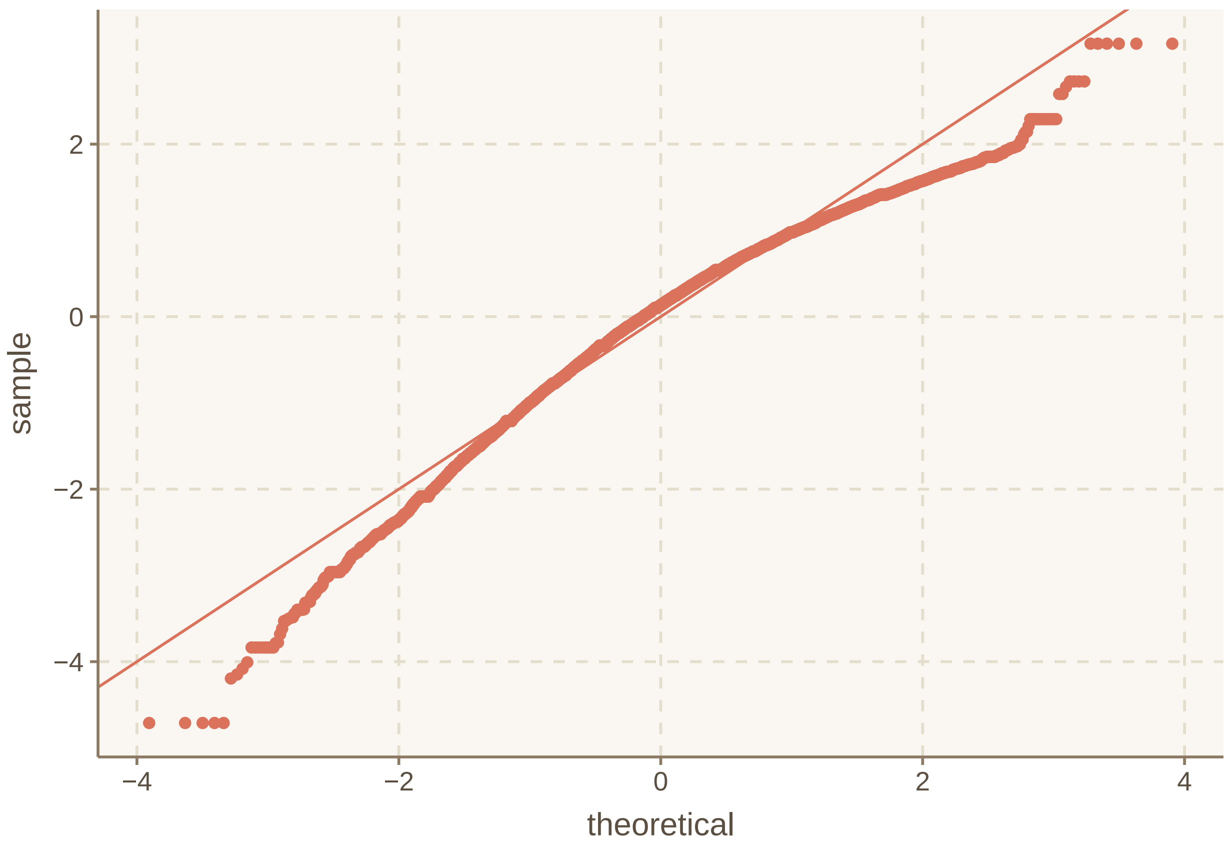
```
movie_avgs_df <- edx %>% group_by(movieId) %>% summarize(b_i = mean(rating) -
    mu) %>% mutate(movie_train_ratings = mu + b_i)
```

Before going forward let us examine b_i and check whether it follows normal distribution or not. To do so we will plot the b_i histogram and plot as the following:

```
movie_avgs_df %>% qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"))
```



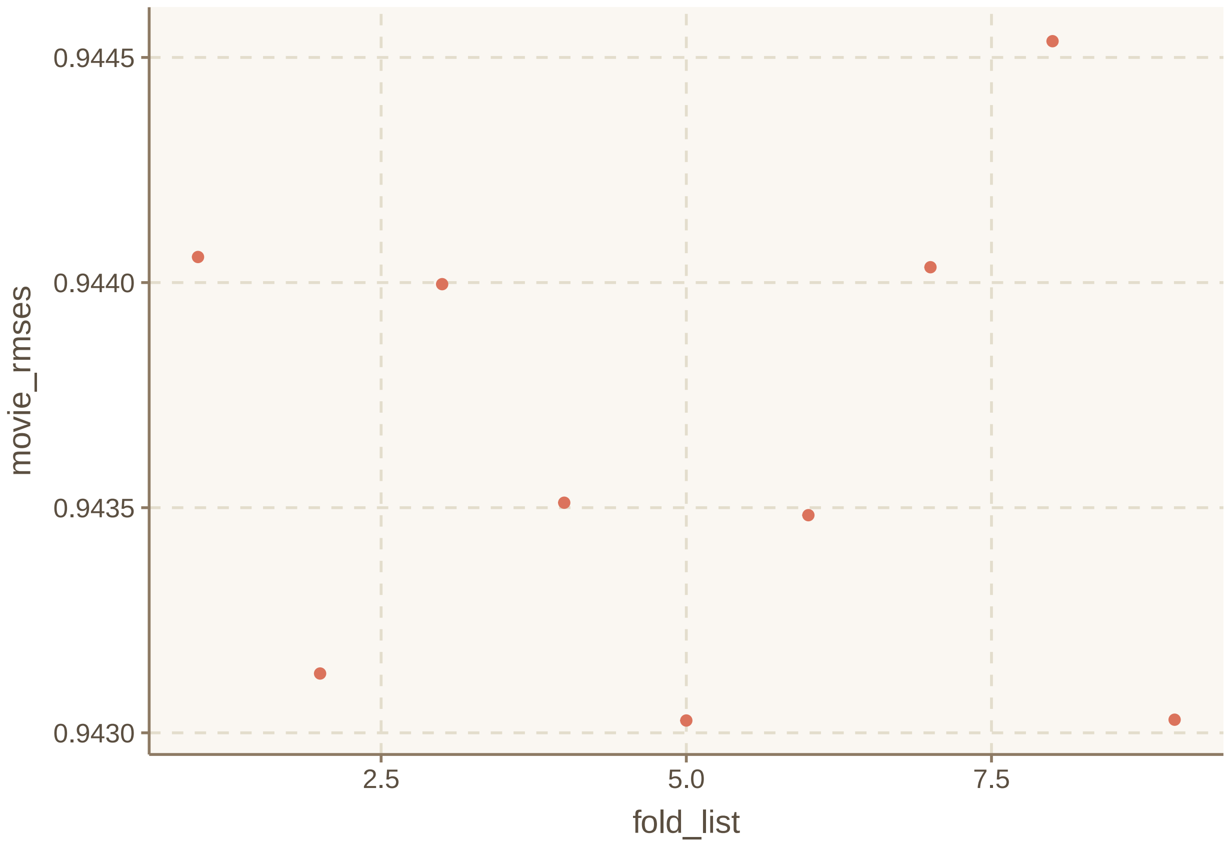
```
movie_avgs_df %>% select(b_i) %>% ggplot(aes(sample = scale(b_i))) + geom_qq() +
    geom_abline()
```



We can see that these estimates vary substantially. Hence, we are in the proper track.

Next, is to calculate the prediction on the cross-validation datasets as the following:

```
fold_list <- seq(1, 9, 1)
if (!file.exists("movie_rmses.rda")) {
    # # Run the algorithm for 9 times on each training set and Pick best
    # performance on the cross-validation set
    movie_rmses <- sapply(fold_list, function(x) {
        # # Get Training set
        training_set <- edx[Training_Index_List[[x]], ]
        # # Get Cross-Validation set
        cross_validation_set <- edx[-Training_Index_List[[x]], ]
        # # Build algorithm

        mu <- mean(training_set$rating)
        # # Calculate b_i
        b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating -
            mu))

        ############### # Run algorithm on the Cross-Validation set

        predicted_ratings_df <- cross_validation_set %>% left_join(b_i, by = "movieId") %>%
            mutate(pred = mu + b_i)

        ################
        # # Measure Performance against cross-validation set
        RMSE_Custom(cross_validation_set$rating, predicted_ratings_df$pred)

    })
    save(movie_rmses, file = "./movie_rmses.rda")
} else {
    load("movie_rmses.rda")
}
qplot(fold_list, movie_rmses)
```



```
# # Pick Number of the training set that achive the best performance
fold_n <- fold_list[which.min(movie_rmses)]
```

Now, we can measure the second try performance on the test dataset as the following:

```
# # Get the Training set that achive the best performance
training_set <- edx[Training_Index_List[[fold_n]], ]

mu <- mean(training_set$rating)

b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating -
    mu))

# # Run algorithm against validation set
predicted_ratings_df <- validation %>% left_join(b_i, by = "movieId") %>% mutate(pred = mu +
    b_i)

# # Measure performance against validation set
result <- RMSE_Custom(validation$rating, predicted_ratings_df$pred)
# # Print result
result
## [1] 0.94398
```

That is good improvement in RMSE as the number went down under 1.0.

let us see a sample 10 records with actual and predicted rating

```
kable(sample_n(predicted_ratings_df, 10) %>% select(title, rating, pred))
```

title	rating	pred
Beauty and the Beast (1991)	2.5	3.6780
Cannibal Holocaust (1980)	3.5	3.2126
While You Were Sleeping (1995)	3.0	3.5091
Gods and Monsters (1998)	5.0	3.8486
Red Violin, The (Violon rouge, Le) (1998)	4.0	3.9080
Office Space (1999)	5.0	3.9613
Reservoir Dogs (1992)	5.0	4.0869
Da Vinci Code, The (2006)	4.0	3.0852
Dead Man Walking (1995)	3.0	3.9604
Star Wars: Episode VI - Return of the Jedi (1983)	3.0	3.9995

Finally for the second try, we will add the result to RMSE table.

```
rmse_pesults_tb <- bind_rows(rmse_pesults_tb, tibble(method = "Movie Effect",
    RMSE = result))
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398

remove unwanted objects from memory

```
rm(b_i, predicted_ratings_df)
```

3.3.3 User Effect

To address movie effect, we will simply update our movie effect equation to take into consideration the user effect. User effect is the difference between the average user given rate and protected movie rate from second equation. Hence we can update our equation as the following:

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

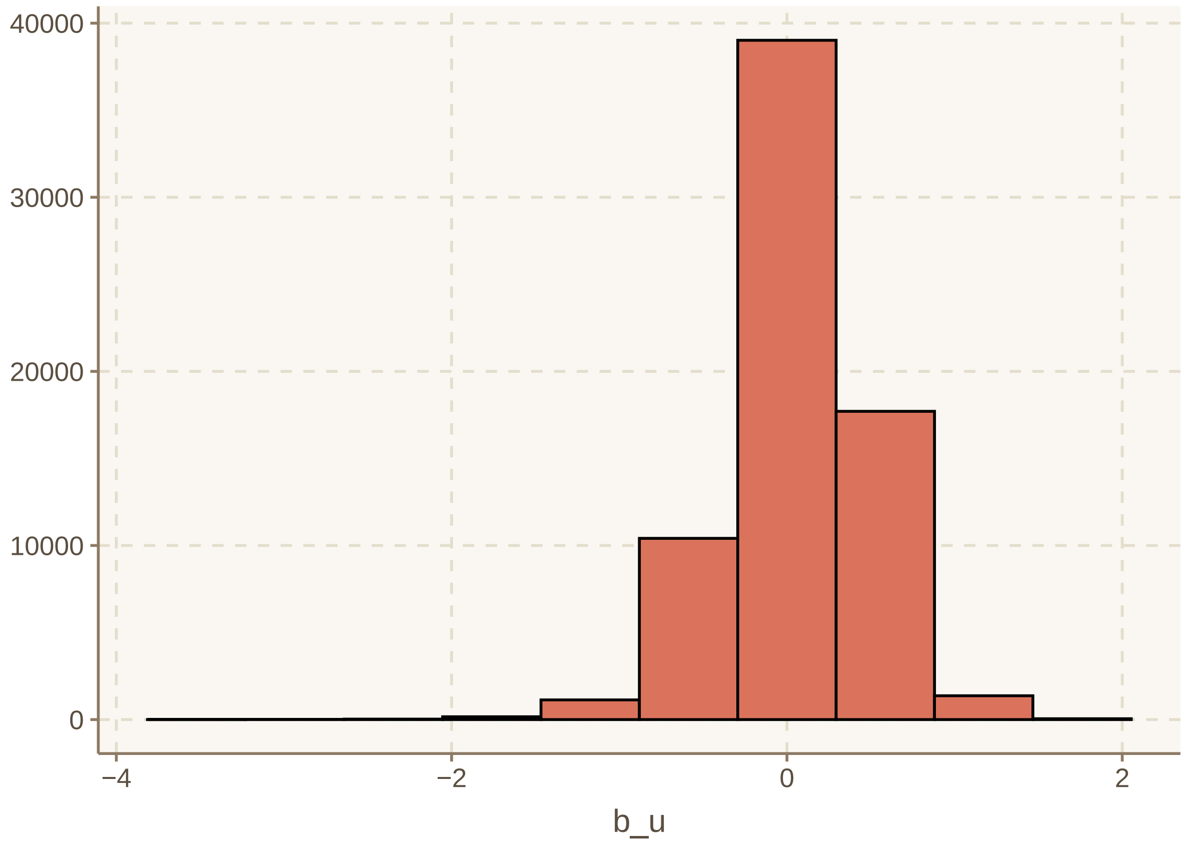
- b_u: Is the the difference between the average user given rate and predicted movie rate from second equation.

Let us now do the calculation for each user as the following:

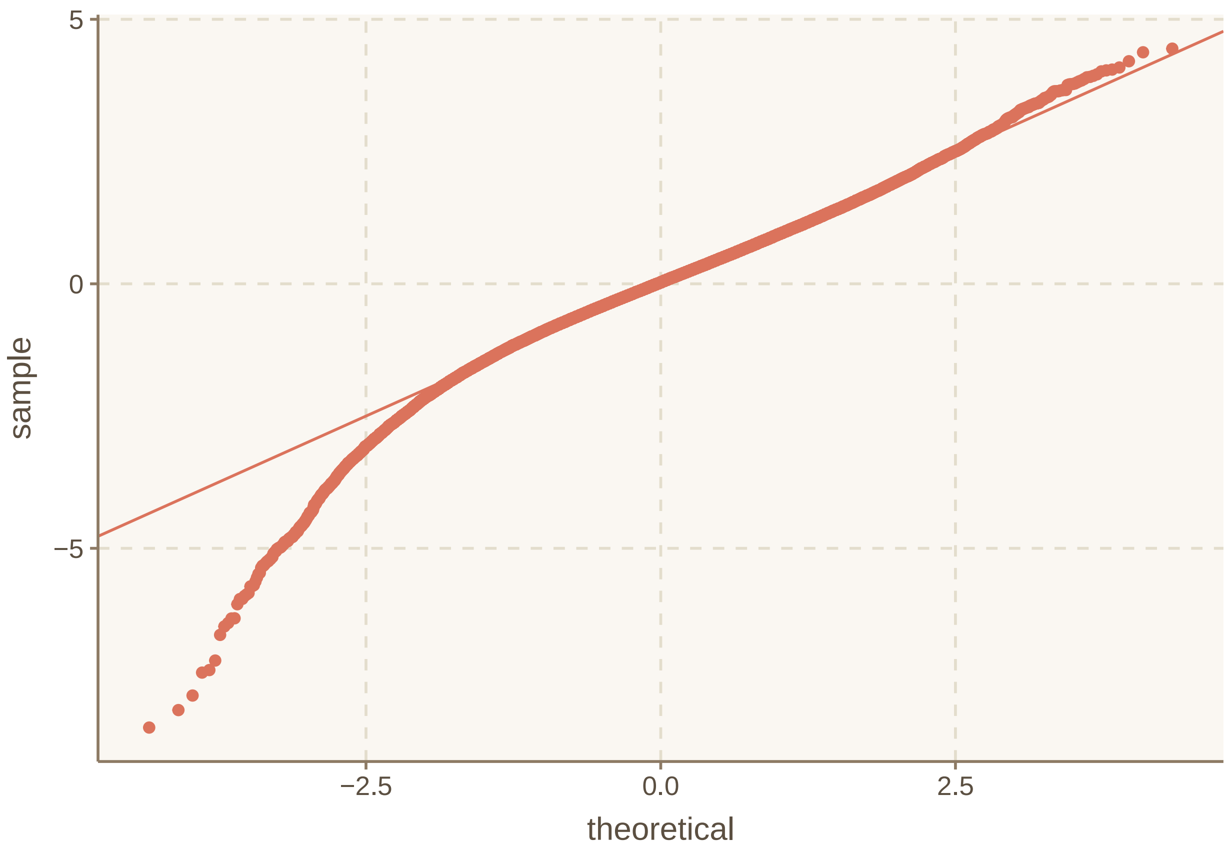
```
user_avgs_df <- edx %>% left_join(movie_avgs_df, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = mean(rating - b_i - mu))
```

Before going forward let us examine b_u and check whether it follow normal distribution or not. To do so we will plot the b_u histogram and plot as the following:

```
user_avgs_df %>% qplot(b_u, geom = "histogram", bins = 10, data = ., color = I("black"))
```



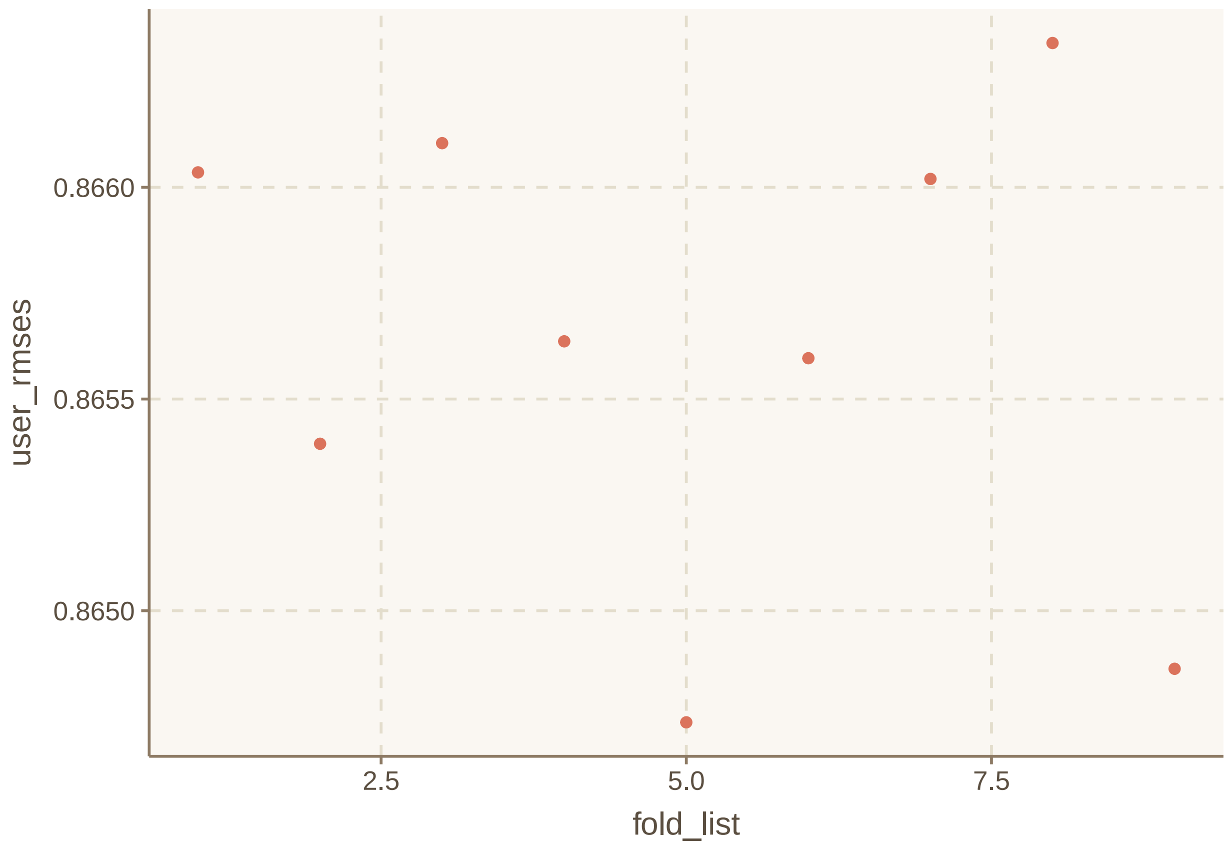
```
user_avgs_df %>% select(b_u) %>% ggplot(aes(sample = scale(b_u))) + geom_qq() +
    geom_abline()
```



We can see that these estimates vary substantially. Hence, we are in the right track.

Now, we can measure the third try performance on cross-validation sets as the following:

```
fold_list <- seq(1, 9, 1)

if (!file.exists("user_rmses.rda")) {
    # # Run the algorithm for 9 times on each training set and Pick best
    # performance on the cross-validation set
    user_rmses <- sapply(fold_list, function(x) {
        # # Get Training set
        training_set <- edx[Training_Index_List[[x]], ]
        # # Get Cross-Validation set
        cross_validation_set <- edx[-Training_Index_List[[x]], ]
        # # Build algorithm

        mu <- mean(training_set$rating)

        b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating) -
            mu)

        b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
            summarize(b_u = mean(rating - mu - b_i))
        ################ # Run algorithm on the Cross-Validation set
        predicted_ratings_df <- cross_validation_set %>% left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>% mutate(pred = mu + b_i + b_u)
        # # Measure Performance against cross-validation set
        RMSE_Custom(cross_validation_set$rating, predicted_ratings_df$pred)

    })
    save(user_rmses, file = "./user_rmses.rda")
} else {
    load("user_rmses.rda")
}
# # Plot training set number against performance
qplot(fold_list, user_rmses)
```



```
# # Pick Number of the training set that achieve the best performance
fold_n <- fold_list[which.min(user_rmses)]
```

Next, is to calculate the prediction on the test dataset as the following:

```
# # Get the Training set that achieve the best performance
training_set <- edx[Training_Index_List[[fold_n]], ]

mu <- mean(training_set$rating)

b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating) -
    mu)

b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))
```

```
# # Run algorithm against validation set
predicted_ratings_df <- validation %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
    by = "userId") %>% mutate(pred = mu + b_i + b_u)
# # Run algorithm against validation set
result <- RMSE_Custom(validation$rating, predicted_ratings_df$pred)
# # Print results
result
```

```
## [1] 0.8659
```

That is good improvement in RMSE as the number went down.

let us see a sample 10 records with actual and predicted ratings

```
kable(sample_n(predicted_ratings_df, 10) %>% select(title, rating, pred))
```

title	rating	pred
Heavy Metal (1981)	4.0	3.8675
Aquamarine (2006)	3.5	2.9760
Amadeus (1984)	4.0	4.0399
Intolerable Cruelty (2003)	3.5	3.3344
Hellraiser (1987)	3.5	3.0379
Kill Bill: Vol. 2 (2004)	4.5	4.4440
Assignment, The (1997)	3.5	2.6569
Royal Tenenbaums, The (2001)	5.0	4.3863
Good Will Hunting (1997)	4.5	3.5400
Real Women Have Curves (2002)	4.0	3.7586

Finally for the third try, we will add the result to RMSE table.

```
rmse_pesults_tb <- bind_rows(rmse_pesults_tb, tibble(method = "User & Movie Effect",
    RMSE = result))
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398
User & Movie Effect	0.86590

Clean Memory from unused objects.

```
rm(b_i, b_u, predicted_ratings_df)
```

3.3.4 Genres effect

To address genres effect, we will simply update our user effect equation to take into consideration the genres effect. Genres effect is the difference between the average rating of each genres and predicted movie rate from third equation. Hence we can update our equation as the following:

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

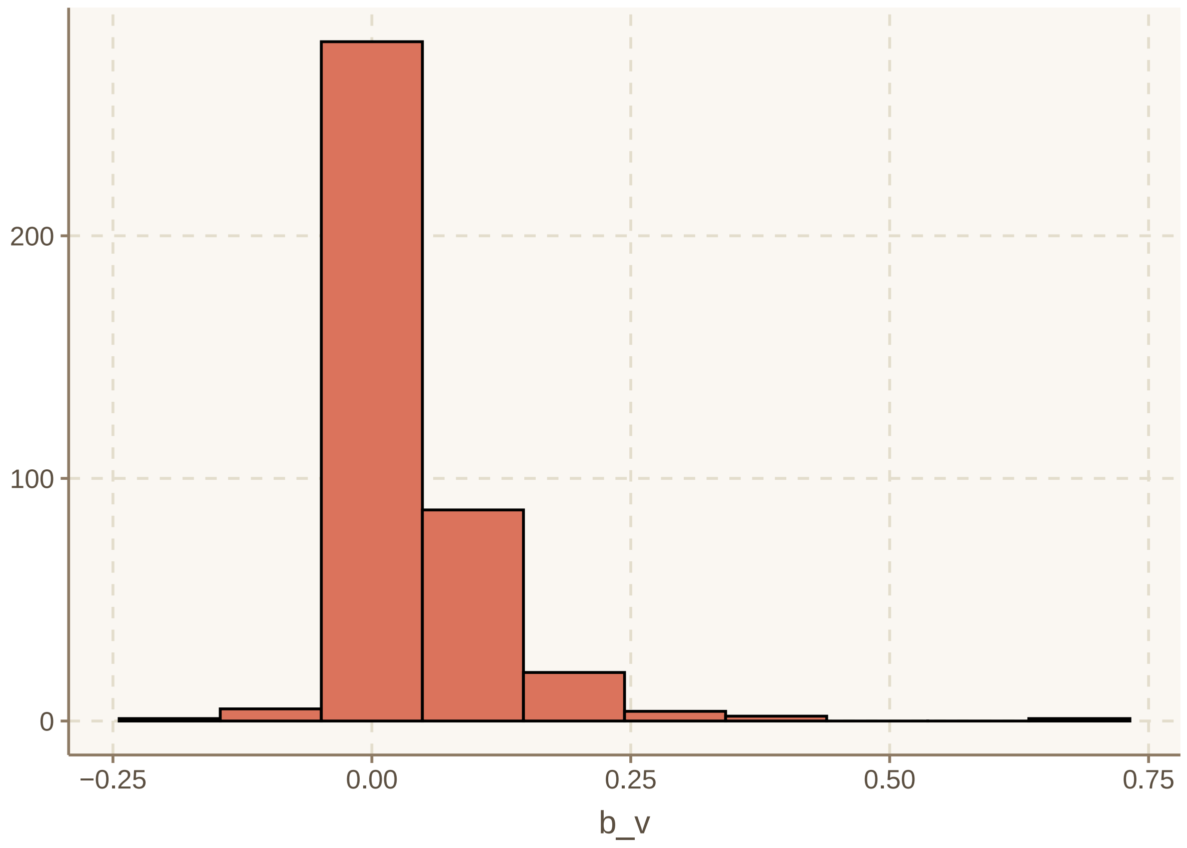
- b_v:Is the difference between the average rating of each genres and predicted movie rate from third equation.

Let us now do the calculation for each genres as the following:

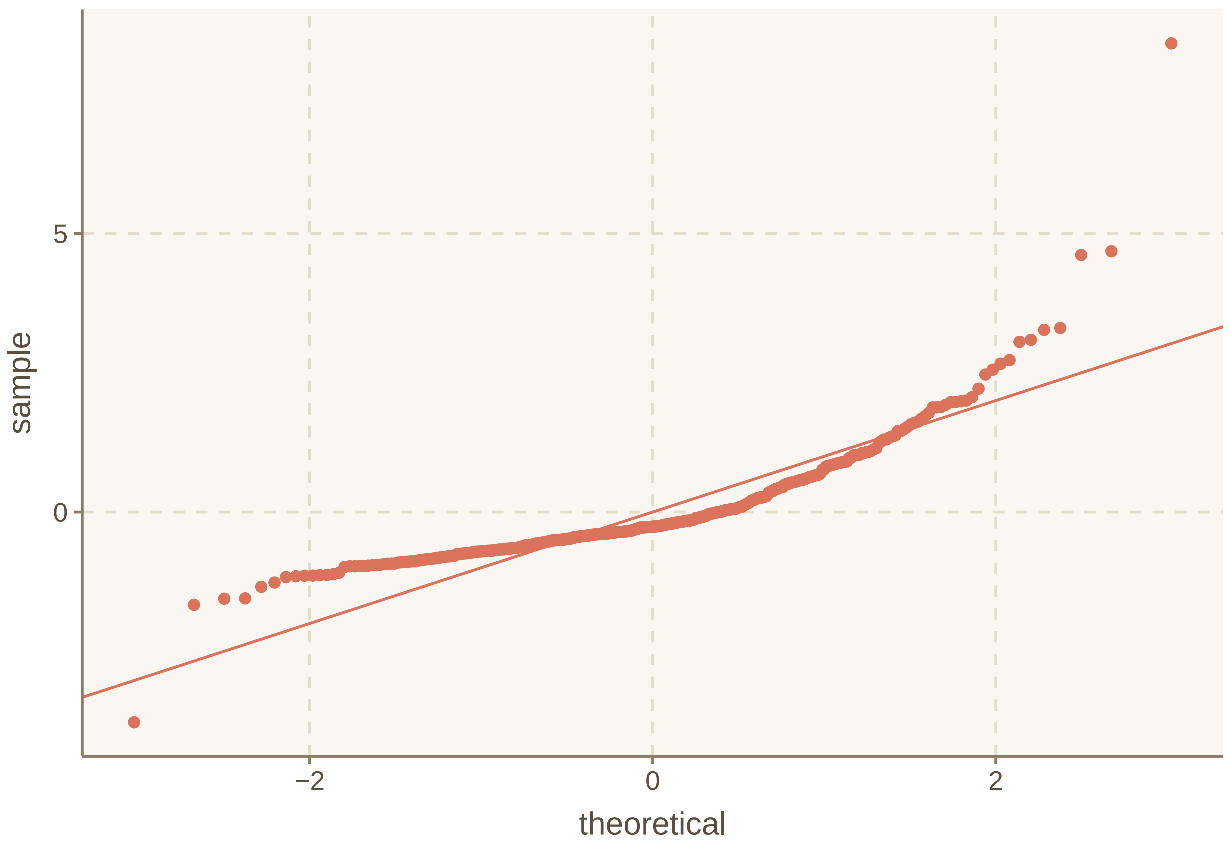
```
genres_avgs_df <- edx %>% left_join(movie_avgs_df, by = "movieId") %>% left_join(user_avgs_df,
    by = "userId") %>% group_by(three_major_genres) %>% summarize(b_v = mean(rating -
    mu - b_i - b_u))
```

Before going forward let us examine b_v and check whether it follow normal distribution or not. To do so we will plot the b_v histogram and plot as the following:

```
genres_avgs_df %>% qplot(b_v, geom = "histogram", bins = 10, data = ., color = I("black"))
```



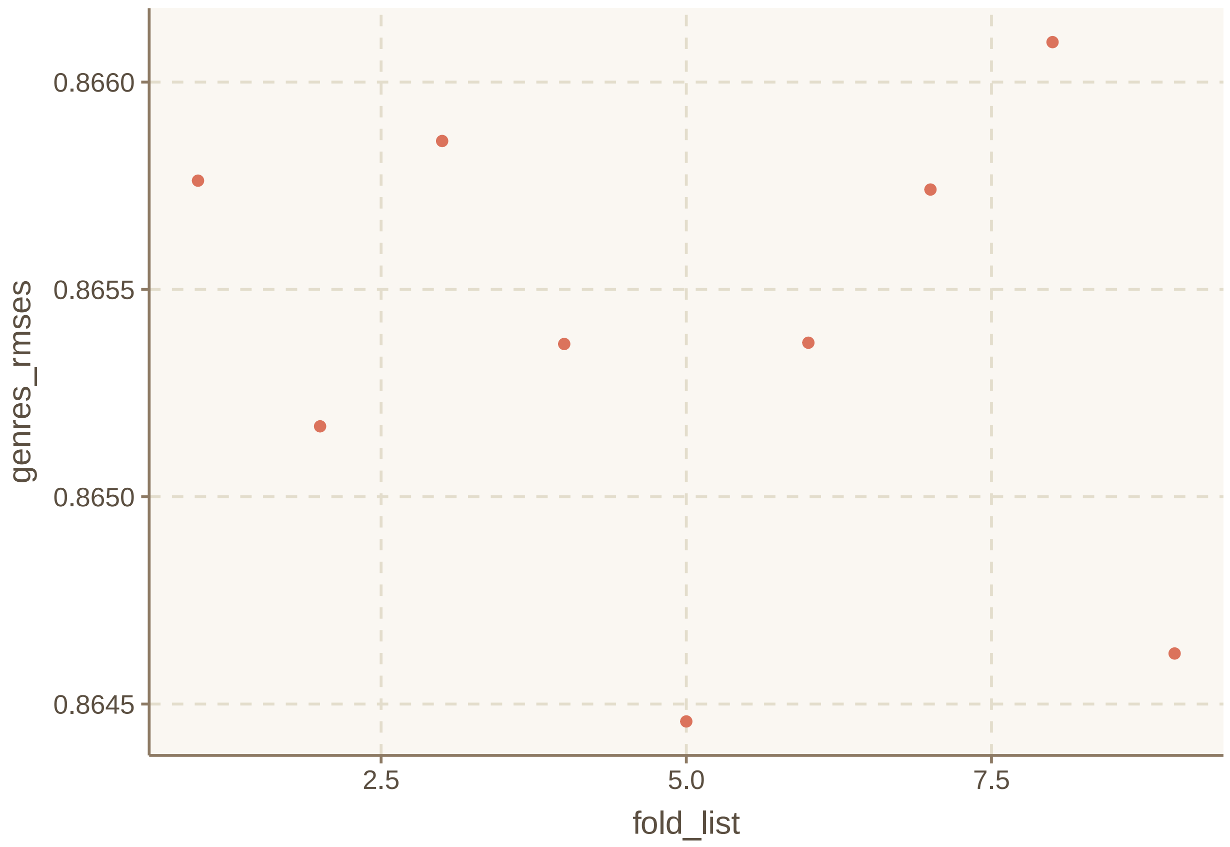
```
genres_avgs_df %>% select(b_v) %>% ggplot(aes(sample = scale(b_v))) + geom_qq() +
    geom_abline()
```



We can see that these estimates vary substantially. Hence, we are in the right track.

Now, we can measure the fourth try performance on cross-validation sets as the following:

```
fold_list <- seq(1, 9, 1)
if (!file.exists("genres_rmses.rda")) {
    # # Run the algorithm for 9 times on each training set and Pick best
    # performance on the cross-validation set
    genres_rmses <- sapply(fold_list, function(x) {
        # # Get Training set
        training_set <- edx[Training_Index_List[[x]], ]
        # # Get Cross-Validation set
        cross_validation_set <- edx[-Training_Index_List[[x]], ]
        # # Build algorithm
        mu <- mean(training_set$rating)
        # # Calculate b_i
        b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating) -
            mu)
        # # Calculate b_u
        b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
            summarize(b_u = mean(rating - mu - b_i))
        # # Calculate b_v
        b_v <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
            by = "userId") %>% group_by(three_major_genres) %>% summarize(b_v = mean(rating -
            mu - b_i - b_u))
        # # Run algorithm on the Cross-Validation set
        predicted_ratings_df <- cross_validation_set %>% left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>% left_join(b_v, by = "three_major_genres") %>%
            mutate(pred = mu + b_i + b_u + b_v)
        # # Measure Performance against cross-validation set
        RMSE_Custom(cross_validation_set$rating, predicted_ratings_df$pred)
    })
    save(genres_rmses, file = "./genres_rmses.rda")
} else {
    load("genres_rmses.rda")
}
# # Plot training set number against performance
qplot(fold_list, genres_rmses)
```



```
# # Pick Number of the training set that achieve the best performance
fold_n <- fold_list[which.min(genres_rmses)]
```

Next, is to calculate the prediction on the test dataset as the following:

```
# # Get the Training set that achieve the best performance
training_set <- edx[Training_Index_List[[fold_n]], ]

mu <- mean(training_set$rating)

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

```r
b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))

b_v <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
    group_by(three_major_genres) %>% summarize(b_v = mean(rating - mu - b_i -
    b_u))

# # Run algorithm against validation set
predicted_ratings_df <- validation %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
    by = "userId") %>% left_join(b_v, by = "three_major_genres") %>% mutate(pred = mu +
    b_i + b_u + b_v)
# # Measure performance against validation set
result <- RMSE_Custom(validation$rating, predicted_ratings_df$pred)
# # Print results
result
```

```
## [1] 0.8656
```

let us see a sample 10 records with actual and predicted ratings.

```r
kable(sample_n(predicted_ratings_df, 10) %>% select(title, rating, pred))
```

title	rating	pred
Usual Suspects, The (1995)	4.0	3.8534
Gremlins 2: The New Batch (1990)	3.0	2.8296
Blood Simple (1984)	5.0	3.8226
Richard Pryor Live on the Sunset Strip (1982)	5.0	3.5827
Van Helsing (2004)	2.0	2.7585
High Fidelity (2000)	4.0	3.3560
Stigmata (1999)	0.5	2.5278
Birdcage, The (1996)	5.0	3.7688
Fish Called Wanda, A (1988)	4.0	3.8638
Cable Guy, The (1996)	4.0	2.9884

Finally for the fourth try, we will add the result to RMSE table.

```r
rmse_pesults_tb <- bind_rows(rmse_pesults_tb, tibble(method = "User & Movie & Genres Effect",
    RMSE = result))
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398
User & Movie Effect	0.86590
User & Movie & Genres Effect	0.86560

```r
rm(b_i, b_u, b_v, predicted_ratings_df)
```

3.3.5 Year effect

To adores genres effect, we will simply update our user effect equation to take into consideration the genres effect. Genres effect is the difference between the average rating of each genres and predicted movie rate from third equation. Hence we can update our equation as the following:

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

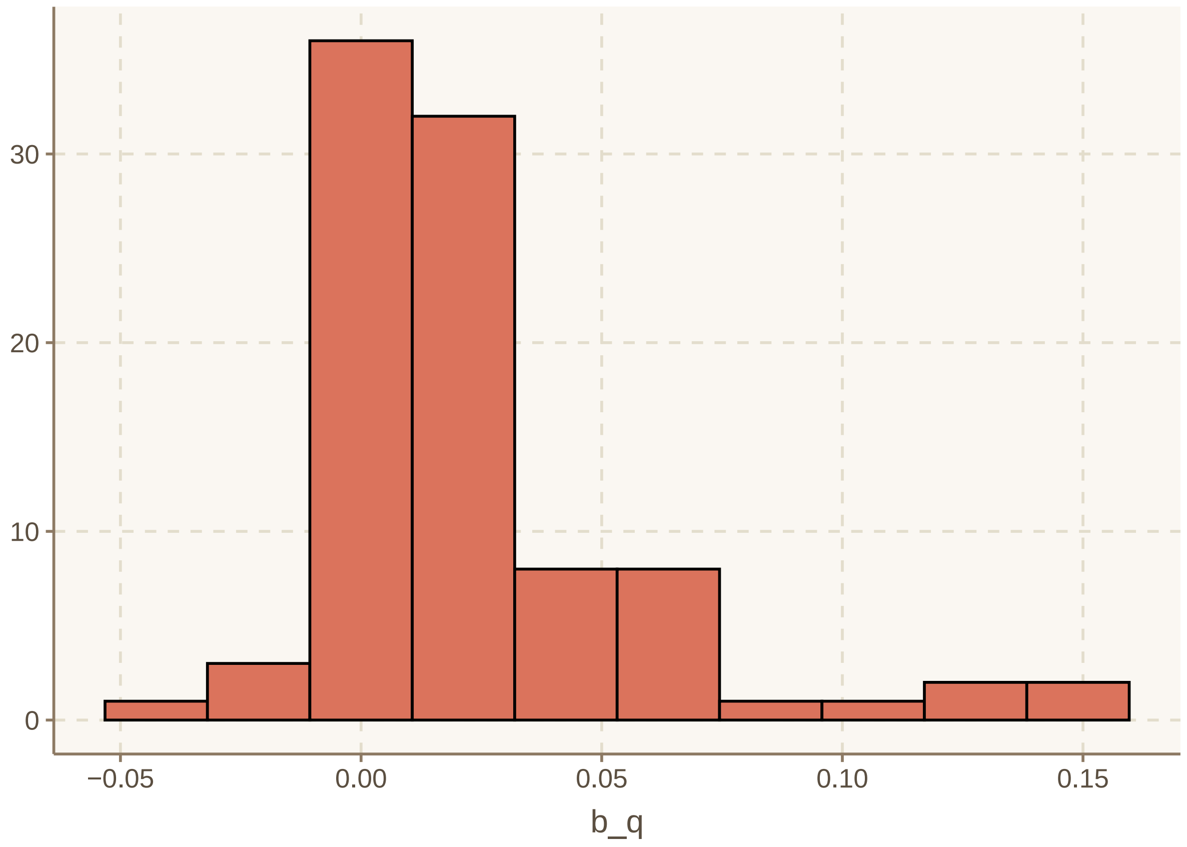
- b_q:Is the the difference between the average rating of each Year and predicted movie rate from fourth equation.

Let us now do the calculation for each Year as the following:

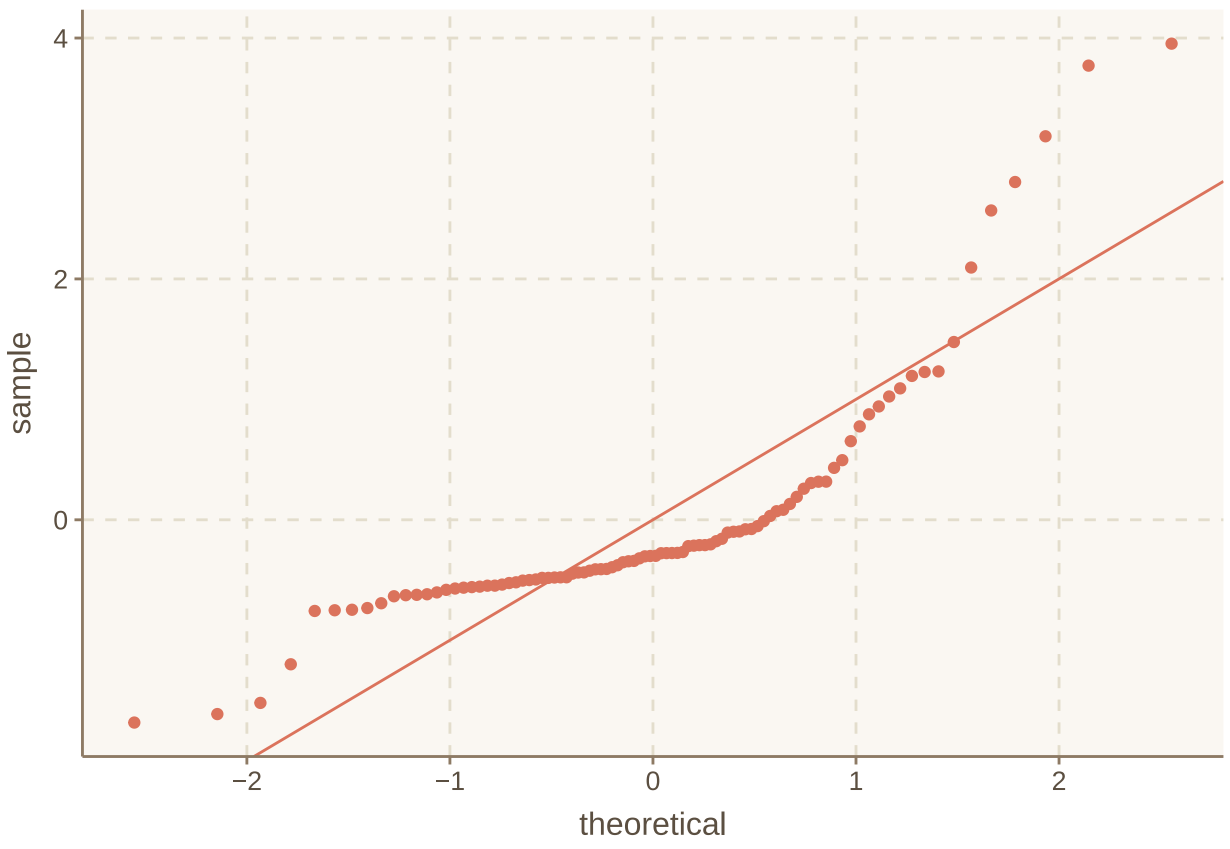
```
year_avgs_df <- edx %>% left_join(movie_avgs_df, by = "movieId") %>% left_join(user_avgs_df,
    by = "userId") %>% left_join(genres_avgs_df, by = "three_major_genres") %>%
    group_by(year) %>% summarize(b_q = mean(rating - mu - b_i - b_u - b_v))
```

Before going forward let us examine b_q and check whether it follow normal distribution or not. To do so we will plot the b_q histogram and plot as the following:

```
year_avgs_df %>% qplot(b_q, geom = "histogram", bins = 10, data = ., color = I("black"))
```



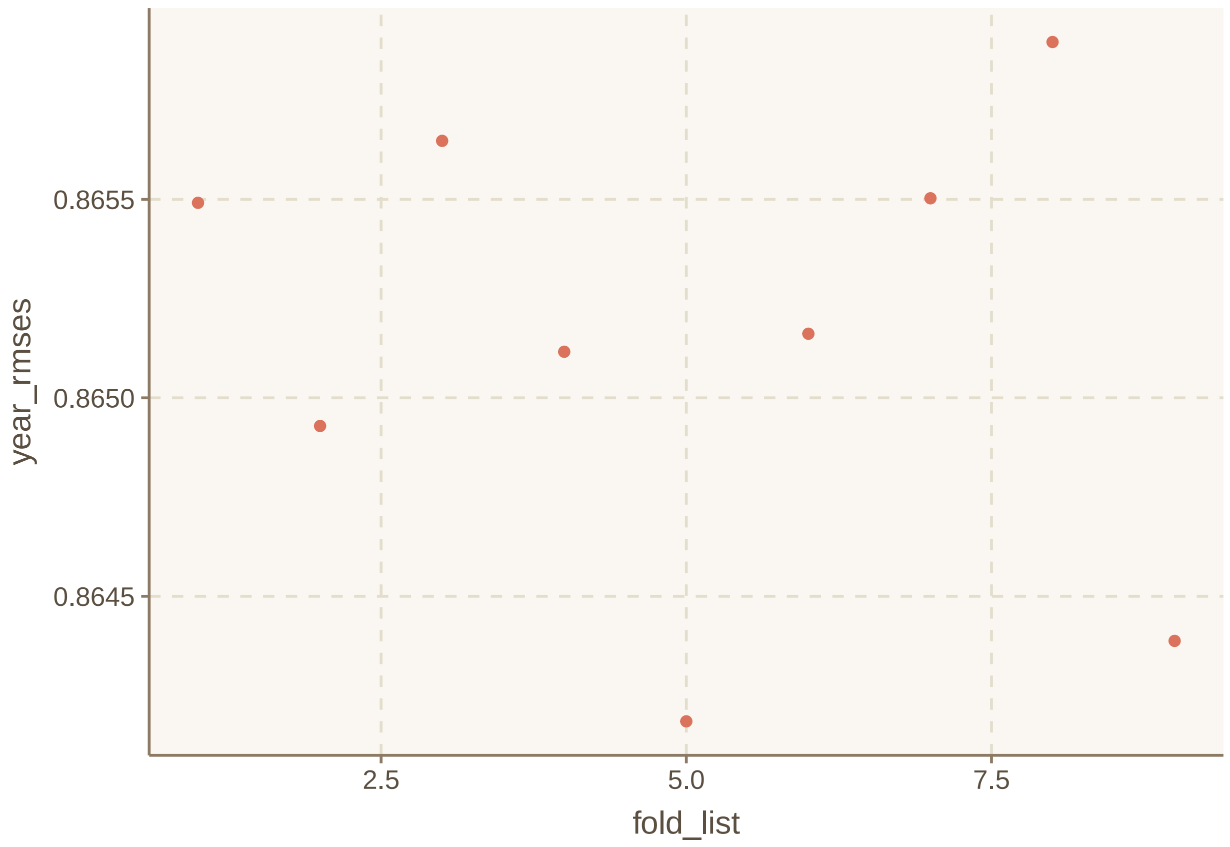
```
year_avgs_df %>% select(b_q) %>% ggplot(aes(sample = scale(b_q))) + geom_qq() +
    geom_abline()
```



We can see that these estimates vary substantially. Hence, we are in the right track.

Now, we can measure the fifth try performance on cross-validation sets as the following:

```
fold_list <- seq(1, 9, 1)
if (!file.exists("year_rmses.rda")) {
    # # Run the algorithm for 9 times on each training set and Pick best
    # performance on the cross-validation set
    year_rmses <- sapply(fold_list, function(x) {
        # # Get Training set
        training_set <- edx[Training_Index_List[[x]], ]
        # # Get Cross-Validation set
        cross_validation_set <- edx[-Training_Index_List[[x]], ]
        # # Build algorithm
        mu <- mean(training_set$rating)
        # # Calculate b_i
        b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating) -
            mu)
        # # Calculate b_u
        b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
            summarize(b_u = mean(rating - mu - b_i))
        # # Calculate b_v
        b_v <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
            by = "userId") %>% group_by(three_major_genres) %>% summarize(b_v = mean(rating -
            mu - b_i - b_u))
        # # Calculate b_q
        b_q <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
            by = "userId") %>% left_join(b_v, by = "three_major_genres") %>%
            group_by(year) %>% summarize(b_q = mean(rating - mu - b_i - b_u -
            b_v))
        # # Run algorithm on the Cross-Validation set
        predicted_ratings_df <- cross_validation_set %>% left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>% left_join(b_v, by = "three_major_genres") %>%
            left_join(b_q, by = "year") %>% mutate(pred = mu + b_i + b_u + b_v +
            b_q)
        # # Measure Performance against cross-validation set
        RMSE_Custom(cross_validation_set$rating, predicted_ratings_df$pred)
    })
    save(year_rmses, file = "./year_rmses.rda")
} else {
    load("year_rmses.rda")
}
# # Plot training set number against performance
qplot(fold_list, year_rmses)
```



```
# # Pick Number of the training set that achieve the best performance
fold_n <- fold_list[which.min(year_rmses)]
```

Next, is to calculate the prediction on the test dataset as the following:

```
# # Get the Training set that achieve the best performance
training_set <- edx[Training_Index_List[[fold_n]], ]

mu <- mean(training_set$rating)

b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = mean(rating) -
    mu)

b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))

b_v <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
    group_by(three_major_genres) %>% summarize(b_v = mean(rating - mu - b_i -
    b_u))

b_q <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
    left_join(b_v, by = "three_major_genres") %>% group_by(year) %>% summarize(b_q = mean(rating -
    mu - b_i - b_u - b_v))
# # Run algorithm against validation set
predicted_ratings_df <- validation %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
    by = "userId") %>% left_join(b_v, by = "three_major_genres") %>% left_join(b_q,
    by = "year") %>% mutate(pred = mu + b_i + b_u + b_v + b_q)

# # Measure performance against validation set
result <- RMSE_Custom(validation$rating, predicted_ratings_df$pred)
result
```

```
## [1] 0.86535
```

That is good improvement in RMSE as the number went down.

let us see a sample 10 records with actual and predicted ratings.

```
kable(sample_n(predicted_ratings_df, 10) %>% select(title, rating, pred))
```

title	rating	pred
Sesame Street Presents Follow That Bird (1985)	3.0	3.8498
Sound of Music, The (1965)	4.0	3.8559
Roman Holiday (1953)	4.0	3.9579
Rambo III (1988)	3.0	2.5602
Citizen Kane (1941)	3.0	3.6851
Wild Things (1998)	4.0	3.1983
Rumble in the Bronx (Hont faan kui) (1995)	3.0	3.2358
Liar Liar (1997)	4.0	3.5221
Say Anything... (1989)	3.5	3.0212
Sling Blade (1996)	5.0	3.8370

Finally for the fifth try, we will add the result to RMSE table.

```
rmse_pesults_tb <- bind_rows(rmse_pesults_tb, tibble(method = "User & Movie & Genres & Year Effect",
    RMSE = result))
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398
User & Movie Effect	0.86590
User & Movie & Genres Effect	0.86560
User & Movie & Genres & Year Effect	0.86535

```
rm(b_i, b_u, b_v, b_q, movie_avgs_df, user_avgs_df, genres_avgs_df, year_avgs_df,
    predicted_ratings_df)
```

3.4 Regression with Regularization

As we have explored the dataset, we have seen that some movies rated only for very few times other movies rated many times. This also apply for users and movie genres.

However, in Regression equation all movies have the same effect on the equation which is not so accurate. We need to penalize estimates that are formed using small sample sizes.

Regularization permits us to penalize large estimates that are formed using small sample sizes.

3.4.1 User, Movie, Genre and Year effect

We can update the final prediction equation as the following to take Regularization into consideration:

$$Y_{u,i} = \mu + b_i(\lambda) + b_i(\lambda) + b_u(\lambda) + b_v(\lambda) + b_q(\lambda) + \epsilon_{u,i}$$

- $b_i(\lambda)$: Regularized movie effect which could be calculated as the following:

$$b_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \mu)$$

- n_i: Number of ratings made for movie i.
- λ: Regularization Factor/ Penalty Terms

Respectively:

- $b_u(\lambda)$: Regularized user effect

$$b_u(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - b_i(\lambda) - \mu)$$

- $b_v(\lambda)$: Regularized genres effect

$$b_v(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - b_i(\lambda) - b_u(\lambda) - \mu)$$

- $b_q(\lambda)$: Regularized year effect

$$b_q(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - b_i(\lambda) - b_u(\lambda) - b_v(\lambda) - \mu)$$

Now let us choose a value for λ using full cross validation on the training dataset.

- We will calculate b__x for 9 Training dataset using one selected value for lambda
- Second, for each b__x, we will test performance on the corresponding cross-validation dataset
- Third, we will pick another value of lambda and repeat the previous two steps

Which means for each value of lambda, we will have 9 corresponding RMSE, hence if we will try for example n values for lambda, at the end we will have 9*n RMSE results.

Finally, we will pick the best Training dataset with corresponding lambda value that achieves lowest RMSE.

```
lambdas <- seq(0, 10, 0.25)

if (!file.exists("reg_rmses.rda")) {

    # # loop on training/cross-validation dataset
    reg_rmses <- sapply(fold_list, function(x) {

        # # loop on training/cross-validation dataset
        training_set <- edx[Training_Index_List[[x]], ]

        # # Get Cross-Validation set
        cross_validation_set <- edx[-Training_Index_List[[x]], ]

        mu <- mean(training_set$rating)

        # # loop on lambda values
        rmses <- sapply(lambdas, function(l) {

            # # Build algorithm

            # # Calculate b_i for each lambda
            b_i <- training_set %>% group_by(movieId) %>% summarize(b_i = sum(rating -
                mu)/(n() + l))

            # # Calculate b_u for each lambda
            b_u <- training_set %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
                summarize(b_u = sum(rating - b_i - mu)/(n() + l))

            # # Calculate b_v for each lambda
            b_v <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
                by = "userId") %>% group_by(three_major_genres) %>% summarize(b_v = sum(rating -
                b_i - b_u - mu)/(n() + l))

            # # Calculate b_q for each lambda
            b_q <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u,
```

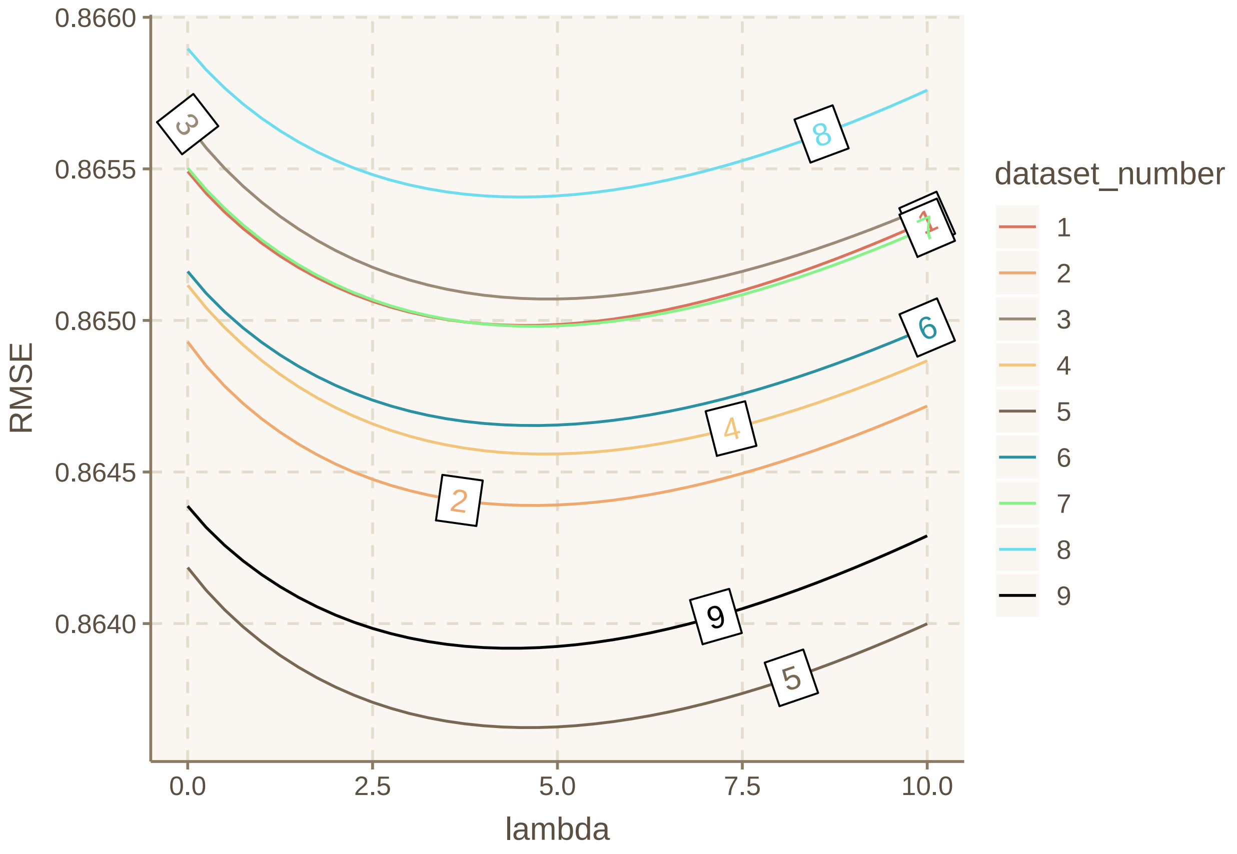
```r
                by = "userId") %>% left_join(b_v, by = "three_major_genres") %>%
                group_by(year) %>% summarize(b_q = sum(rating - b_i - b_u -
                b_v - mu)/(n() + l))

            # # Run algorithm against validation set
            predicted_ratings <- cross_validation_set %>% left_join(b_i, by = "movieId") %>%
                left_join(b_u, by = "userId") %>% left_join(b_v, by = "three_major_genres") %>%
                left_join(b_q, by = "year") %>% mutate(pred = mu + b_i + b_u +
                b_v + b_q) %>% pull(pred)

            # # Measure performance against validation set
            return(RMSE_Custom(cross_validation_set$rating, predicted_ratings))
        })
        ## try_no Repeated for 41 time = 41 value for lambda
        return(list(try_no = rep(x, 41), lambda = lambdas, RMSE = rmses))
    })
    save(reg_rmses, file = "./reg_rmses.rda")
} else {
    load("reg_rmses.rda")
}
```

For easy plotting, I have converted the result to a dataframe.

```r
# # Convert rmse results to a dataframe with 3 columns:
# 'Cross_Val_no','lambda','RMSE'
reg_rmses_list <- lapply(1:9, function(x) {
    l1 <- (x * 3) - 2
    l2 <- x * 3
    df <- matrix(unlist(reg_rmses[l1:l2]), nr = 41)
})
reg_rmses_df <- as.data.frame(as.matrix(unlist(do.call(rbind, reg_rmses_list))),
    ncol = 3)

colnames(reg_rmses_df) <- c("Cross_Val_no", "lambda", "RMSE")
```

Plot RMSE values against lambda values for each cross-validation try

```r
set_swatch(append(swatch(), "black"))
reg_rmses_df %>% mutate(dataset_number = factor(Cross_Val_no)) %>% ggplot(aes(lambda,
    RMSE, color = dataset_number, group = dataset_number, label = dataset_number)) +
    geom_line() + geom_dl(aes(label = dataset_number), method = list("angled.boxes"))
```



From the above graph, it is clear the using training dataset number 5 with lambda around 4 will five us lowest RMSE.

```
min_RMSE <- reg_rmses_df[which.min(reg_rmses_df$RMSE), 3]
reg_rmses_df %>% filter(RMSE == min_RMSE)
##   Cross_Val_no lambda    RMSE
## 1            5    4.5 0.86366
```

Now, let us apply on test dataset.

```
x <- reg_rmses_df[which.min(reg_rmses_df$RMSE), 1]
l <- reg_rmses_df[which.min(reg_rmses_df$RMSE), 2]


training_set <- edx[Training_Index_List[[x]], ]

mu <- mean(training_set$rating)

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

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

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

```r
b_q <- training_set %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "userId") %>%
    left_join(b_v, by = "three_major_genres") %>% group_by(year) %>% summarize(b_q = sum(rating -
    b_i - b_u - b_v - mu)/(n() + l))

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

result <- RMSE_Custom(validation$rating, predicted_ratings)
```

Finally for the fifth try, we will add the result to RMSE table.

```r
rmse_pesults_tb <-
bind_rows(rmse_pesults_tb,
          tibble(method="User & Movie & Genres & Year Regularized Effect",
                 RMSE = result))
kable(rmse_pesults_tb,"latex", booktabs = T)%>%
  kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398
User & Movie Effect	0.86590
User & Movie & Genres Effect	0.86560
User & Movie & Genres & Year Effect	0.86535
User & Movie & Genres & Year Regularized Effect	0.86477

3.5 Matrix Factorization

Third algorithm, that we will try is Matrix Factorization which is a popular technique for solving recommenders system issues. The main idea is to approximate the matrix R(m x n) by the product of two matrices of a lower dimension: P (m x k) and Q (k x n).

Matrix P represents latent factors of users. So, each k-elements column of matrix P represents each user. Each k-elements column of matrix Q represents each item.

Matrix Factorization has a ready made Package in R called recosystem which use only rating and effects of User and Movie into consideration require to have inputs serialized to a Text files as the following:

```r
if (!file.exists("mtx_fact_pred_ratings.rda")) {
    # select only required fields for the algorithm
    train_set <- edx %>% select(movieId, userId, rating)
    vald_set <- validation %>% select(movieId, userId, rating)

    # Convert datasets to matrix format, so it would be ready to be serialized
    train_set_mat <- as.matrix(train_set)
    vald_set_mat <- as.matrix(vald_set)

    # Create text file for Training Set
    write.table(train_set_mat, file = "trainingset.txt", sep = " ", row.names = FALSE,
        col.names = FALSE)
    # Create text file for Test Set
    write.table(vald_set_mat, file = "validationset.txt", sep = " ", row.names = FALSE,
        col.names = FALSE)

    # Upload Training Set in the required fromat
    train_set <- data_file("trainingset.txt")
    # Upload Test Set in the required fromat
    vald_set <- data_file("validationset.txt")
}
```

Now we could start by creating Recommender object and tune parameters of Recommender:

- dim: number of latent factors
- lrate: gradient descend step
- cost: penalty parameter to avoid overfitting

To make this easier and faster to run, cost could be set to a small value and dim will adopt to it while tuning:

```r
if (!file.exists("mtx_fact_pred_ratings.rda")) {
    r = Reco()
    # set tune parameters
    opts = r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1,
        0.2), costp_l1 = 0, costq_l1 = 0, nthread = 1, niter = 10))
    # train model
    r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20, verbose = FALSE))
}
```

After training, it is time for prediction.

```r
# Create file to write predicted values
if (!file.exists("mtx_fact_pred_ratings.rda")) {
    pred_file = tempfile()

    # Predict on test set
    r$predict(vald_set, out_file(pred_file))

    # Read Actual rating
    actual_ratings <- read.table("validationset.txt", header = FALSE, sep = " ")$V3

    # Read Predict rating rating
    mtx_fact_pred_ratings <- scan(pred_file)

    save(actual_ratings, file = "./actual_ratings.rda")
    save(mtx_fact_pred_ratings, file = "./mtx_fact_pred_ratings.rda")
} else
{
    load("mtx_fact_pred_ratings.rda")
    load("actual_ratings.rda")
}
```

Finally, let us check RMSE

```r
result <- RMSE_Custom(actual_ratings, mtx_fact_pred_ratings)

rmse_pesults_tb <- bind_rows(rmse_pesults_tb, tibble(method = "Matrix Factorization",
    RMSE = result))
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398
User & Movie Effect	0.86590
User & Movie & Genres Effect	0.86560
User & Movie & Genres & Year Effect	0.86535
User & Movie & Genres & Year Regularized Effect	0.86477
Matrix Factorization	0.78282

RMSE decreased by a noticeable value, hence we are in a right way.

3.6 Singular Value Decomposition

Singular value decomposition (SVD). SVD is a mathematical result that is widely used in machine learning, both in practice and to understand the mathematical properties of some algorithms.

The SVD tells us that we can decompose an $N \times p$ matrix Y with $p < N$ as

$$Y = UDV^{\top}$$

with U and V orthogonal of dimensions $N \times p$ and $p \times p$ respectively and D a $p \times p$ diagonal matrix with the values of the diagonal decreasing:

$$d_{1,1} \geq d_{2,2} \geq \dots d_{p,p}$$

To achieve that we will do the next steps:

- Split Training (edx) set to Training and Validation sets.
- Check that all Movies and Users in Validation and Test sets are also existing in Training set.
- Make sure that training set doesn't have Users or Movies that are not exist in validation and test datasets to avoid any matrix dimensions issues.
- Convert Training, Validation and Test sets to Sparse Matrix format for better memory management.
- Replace all missing Rating with average rating.
- Normalize matrix by subtracting users averages (calculated based on initial rating matrix, not filled-in with average).
- Perform Singular Value Decomposition for Y.
- Keeping only first p rows of matrix U , p rows and p columns of matrix D and p columns of matrix V , to reconstruct matrix Y:

```r
if (!file.exists("decomposed_matrix.rda")) {
    # As all SVD is memeory consuming let us use PC virtual memory
    # memory.size(max=80000) Only keep fields that are required by the algorithm
    edx <- edx %>% select(userId, movieId, rating)

    # Only keep fields that are required by the algorithm
    SVD_test_set <- validation %>% select(userId, movieId, rating)

    # Split Training set to Training and Validation sets
    set.seed(1)
    vald_index <- createDataPartition(y = edx$rating, times = 1, p = 0.111,
        list = FALSE)

    SVD_train_set <- edx[-vald_index, ]
    SVD_vald_set <- edx[vald_index, ]

    # Remove rows from training that are not exsit in validation and test sets
    SVD_train_set <- SVD_train_set %>% filter(userId %in% SVD_vald_set$userId &
        userId %in% SVD_test_set$userId) %>% filter(movieId %in% SVD_vald_set$movieId &
        movieId %in% SVD_test_set$movieId)

    # Remove rows from validation that are not exsit in training set
    SVD_vald_set <- SVD_vald_set %>% semi_join(SVD_train_set, by = "movieId") %>%
        semi_join(SVD_train_set, by = "userId")

    # Remove rows from test that are not exsit in training set
    SVD_test_set <- SVD_test_set %>% semi_join(SVD_train_set, by = "movieId") %>%
        semi_join(SVD_train_set, by = "userId")
}
```

Now, we will have out datasets as sparse matrices.

```r
if(!file.exists("decomposed_matrix.rda")){
SVD_train_set$userId <- as.factor(SVD_train_set$userId)
SVD_train_set$movieId <- as.factor(SVD_train_set$movieId)

SVD_vald_set$userId <- as.factor(SVD_vald_set$userId)
SVD_vald_set$movieId <- as.factor(SVD_vald_set$movieId)


SVD_test_set$userId <- as.factor(SVD_test_set$userId)
SVD_test_set$movieId <- as.factor(SVD_test_set$movieId)

l1 <- length(unique(SVD_train_set$userId))
l2 <- length(unique(SVD_train_set$movieId))

# Convert Training to Sparse Matrix
train_matrix <- sparseMatrix(
  i = as.numeric(SVD_train_set$userId) ,
  j = as.numeric(SVD_train_set$movieId),
  x = SVD_train_set$rating,
  dims = c(l1,l2),
  dimnames = list(paste("u", 1:l1, sep = ""),
  paste("m", 1:l2, sep = "")))
# Convert Validation to Sparse Matrix
valid_matrix <- sparseMatrix(
  i = as.numeric(SVD_vald_set$userId) ,
  j = as.numeric(SVD_vald_set$movieId),
  x = SVD_vald_set$rating,
  dims = c(l1,l2),
  dimnames = list(paste("u", 1:l1, sep = "") ,
  paste("m", 1:l2, sep = "")))

# Convert Test to Sparse Matrix
test_matrix <- sparseMatrix(
  i = as.numeric(SVD_test_set$userId) ,
  j = as.numeric(SVD_test_set$movieId),
  x = SVD_test_set$rating,
  dims = c(l1,l2),
  dimnames = list(paste("u", 1:l1, sep = "") ,
  paste("m", 1:l2, sep = "")))

save(test_matrix,file="./test_matrix.rda")

} else ( load("test_matrix.rda")) # To be used in Calculating RMSE on Test set.
```

To have easy calculation for Number of Users and Number of Movies. In addition to easily manipulate the missing values We will build a corresponding binary matrix for each dataset.

In binary matrix, one will be corresponding to any value and zero for any missing value.

```
if (!file.exists("decomposed_matrix.rda")) {
    train_matrix_ones <- as(binarize(new("realRatingMatrix", data = train_matrix),
        minRating = 0), "dgCMatrix")

    valid_matrix_ones <- as(binarize(new("realRatingMatrix", data = valid_matrix),
        minRating = 0), "dgCMatrix")

    test_matrix_ones <- as(binarize(new("realRatingMatrix", data = test_matrix),
        minRating = 0), "dgCMatrix")

    save(test_matrix_ones, file = "./test_matrix_ones.rda")

} else (load("test_matrix_ones.rda"))  # To be used in Calculating RMSE on Test set.
```

Now, let us start by replacing any missing rating value on the Training Matrix by Movie average rating.

```
if(!file.exists("decomposed_matrix.rda")){
# Calculate Movie average
 movie_avg_matrix <- colSums(train_matrix) / colSums(train_matrix_ones)

 # Replace any mssing average movie rating by average rating for all mvoies
 na_Index <- which(is.na(movie_avg_matrix) == TRUE)
 movie_avg_matrix[na_Index] <- sum(train_matrix) /sum(train_matrix_ones)

  # Replace any mssing User rating by average movie rating
train <- train_matrix +
  (matrix(rep(movie_avg_matrix,
              nrow(train_matrix)),
          nrow = nrow(train_matrix), byrow = TRUE) * (1 - train_matrix_ones))
}
```

For better memory management, Remove unwanted objects.

```
if (!file.exists("decomposed_matrix.rda")) {
    rm(edx, SVD_train_set, SVD_vald_set, SVD_test_set, vald_index, validation,
        train_set)
}
```

Next, let us Normalize Training Matrix by subtracting User average rating for each user from actual rating.

```
if (!file.exists("decomposed_matrix.rda")) {

    # Calculate User average
    user_avg_vector <- rowSums(train_matrix)/rowSums(train_matrix_ones)
    # Normalize Training Matrix
    train <- sweep(x = train, MARGIN = 1, STATS = user_avg_vector, FUN = "-")

    save(user_avg_vector, file = "./user_avg_vector.rda")

} else (load("user_avg_vector.rda"))  # To be used in Calculating RMSE on Test set.
```

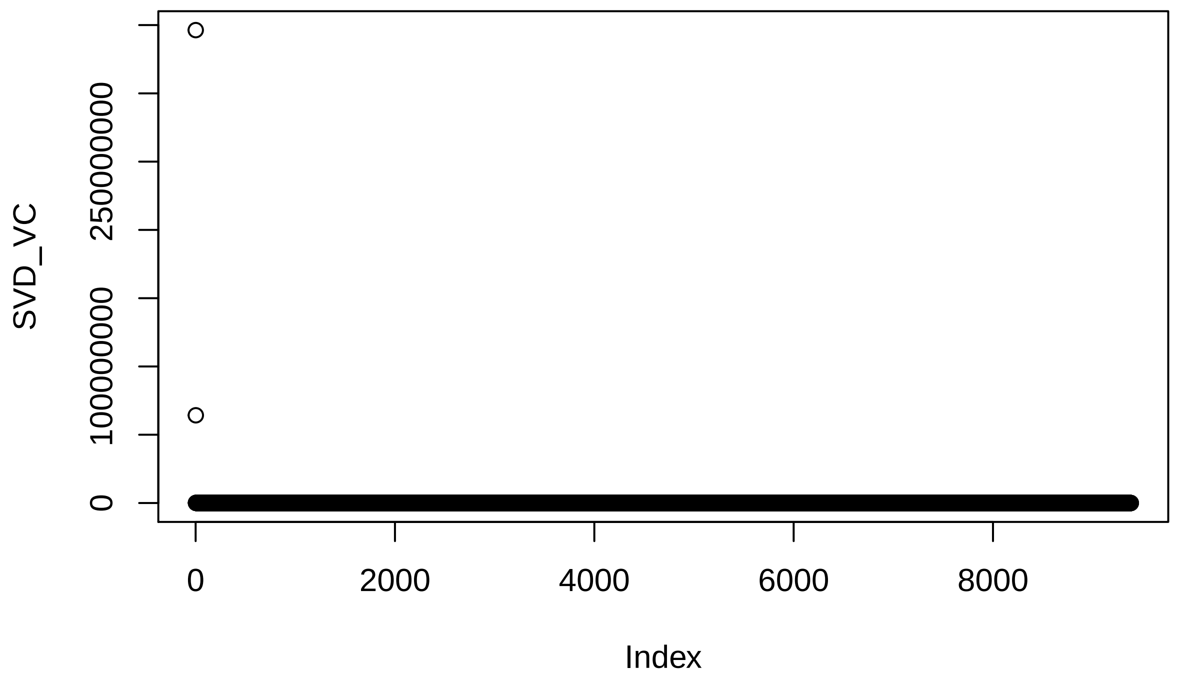
```
## [1] "user_avg_vector"
```

After our Training Matrix is ready and Normalized we can compute SVD components.

```
if (!file.exists("decomposed_matrix.rda")) {
    decomposed_matrix <- svd(train)

    save(decomposed_matrix, file = "./decomposed_matrix.rda")
}
```

Let us check the SVD variance components.

```
if (!file.exists("SVD_VC.rda")) {
    SVD_VC <- sweep(decomposed_matrix$u, 2, decomposed_matrix$d, FUN = "*")
    SVD_VC <- apply(SVD_VC^2, 2, sum)
    plot(SVD_VC)
    save(SVD_VC, file = "./SVD_VC.rda")
} else {
    load("SVD_VC.rda")
    plot(SVD_VC)
}
```



It is so obvious from the plot that with only 3-4 SVD Components, we achieved most of the variance and we can reconstruct training matrix with minimum loss, hence good value for RMSE.

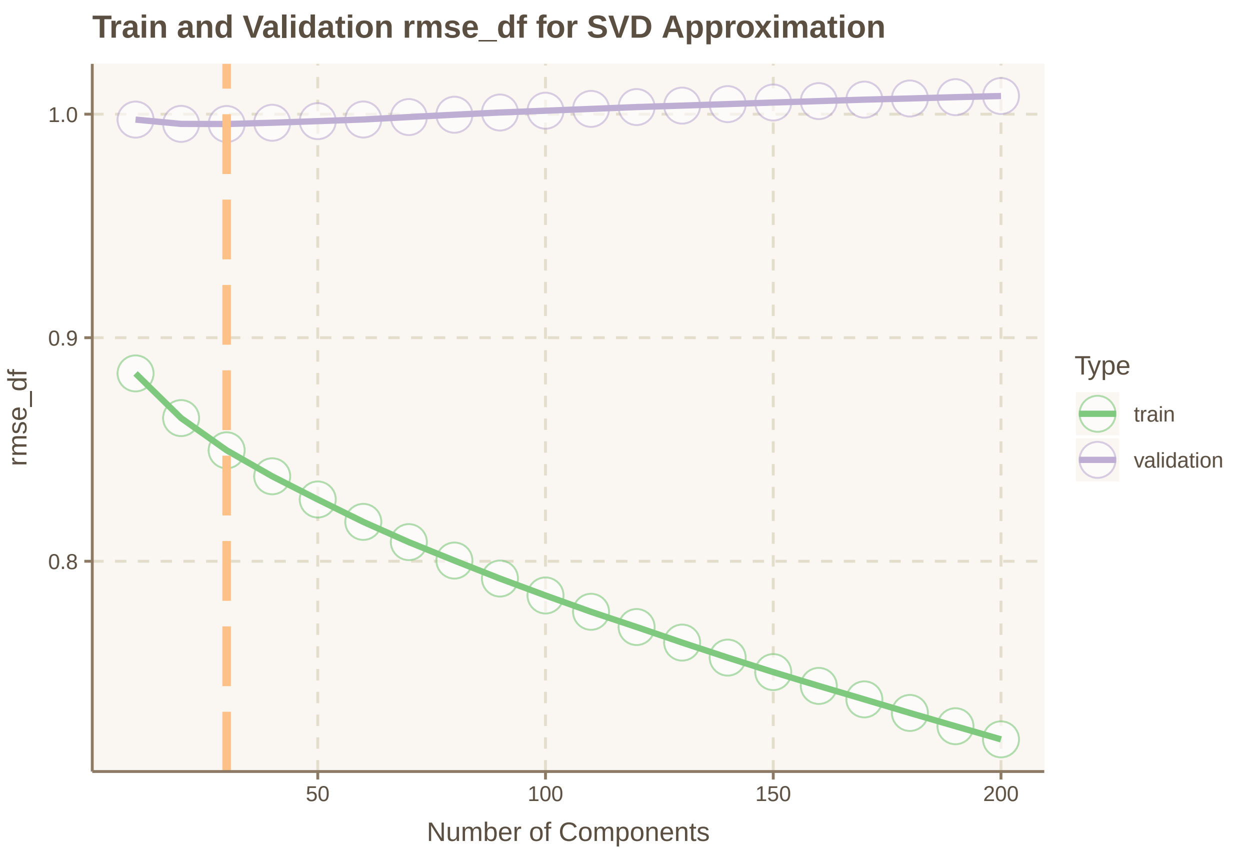
However, let us do the calculation to check the best value for number of SVD Components p.

Now, to figure out the *p* value that give us best performance (Minimum RMSE) and not proceed overfitting issue. We will calculate *p* value that achieve the best performance on validation set and then use the same *p* value to calculate RMSE the test set.

```
if (!file.exists("rmse_df.rda")) {
    library(SMUT)
    k <- seq(10, 200, 10)
    # Constucte a dataframe to keep trak of RMSE
    rmse_df <- data.frame(p = integer(), train = numeric(), validation = numeric())

    sum_train <- sum(train_matrix_ones)
    sum_vald <- sum(valid_matrix_ones)

    for (p in k) {

        # a = UD
        a <- with(decomposed_matrix, sweep(u[, 1:p, drop = FALSE], 2, d[1:p,
            drop = FALSE], FUN = "*"))
        # b = UDVt
        b <- eigenMapMatMult(a, t(decomposed_matrix$v[, 1:p]))

        # Reconstruct Matrix Y= UDVt
        pred <- sweep(x = b, MARGIN = 1, STATS = user_avg_vector, FUN = "+")
        # RMSE corresponding to selected p
        rmse_p <- NULL

        # RMSE on Training
        rmse_p <- c(rmse_p, sqrt(sum((pred * train_matrix_ones - train_matrix)^2)/sum_train))

        # RMSE on Validation
        rmse_p <- c(rmse_p, sqrt(sum((pred * valid_matrix_ones - valid_matrix)^2)/sum_vald))


        # Add RMSE values to RMSE dataframe
        rmse_df <- rbind(rmse_df, data.frame(p = p, train = rmse_p[1], validation = rmse_p[2]))

        cat("Model is evaluated for p =", p, "\n")
    }
    save(rmse_df, file = "./rmse_df.rda")
} else {
    load("rmse_df.rda")
}
```

As we have calculated RMSE on Training and Test sets for different values of p, Let us now plot RMSE vs p values to chooses the best p value to avoid overfitting.

```
    color_pallet <- "Accent"

    data_plot <- reshape2::melt(rmse_df, id.vars = "p")
    names(data_plot) <- c("p", "Type", "Value")

    cols <- brewer.pal(3, color_pallet)

    ggplot(data = data_plot,aes(x = p, y = Value, color = Type)) +
      geom_point(shape = 21, size = 6, fill="white", alpha=6/10) +
      geom_line(size = 1.1) +
      geom_vline(xintercept = rmse_df[which.min(rmse_df[,3]),1],
                 linetype = "longdash", colour = cols[3], size=1.5) +
            scale_color_manual(values=cols[1:2]) +
      labs(title = "Train and Validation rmse_df for SVD Approximation",
          x = "Number of Components", y = "rmse_df") +
      theme(axis.text = element_text(size=8),
           axis.title = element_text(size=10),
           title=element_text(size=10),
           legend.text = element_text(size=8))
```



We can clearly see from the previous graph that with only 30 components (p value), we are getting a good value for RMSE around 0.90.

Let us now, calculate RMSE on Test dataset which is unseen data.

```
if (!file.exists("SVD_result.rda")) {
    # Get best value for p
    p_best = rmse_df[which.min(rmse_df[, 3]), 1]
    p_best

    # Calculate Y Matrix
    a <- with(decomposed_matrix, sweep(u[, 1:p_best, drop = FALSE], 2, d[1:p_best,
        drop = FALSE], FUN = "*"))

    b <- eigenMapMatMult(a, t(decomposed_matrix$v[, 1:p_best]))

    pred <- sweep(x = b, MARGIN = 1, STATS = user_avg_vector, FUN = "+")

    # Calculate RMSE
    SVD_result <- sqrt(sum((pred * test_matrix_ones - test_matrix)^2)/sum(test_matrix_ones))

    save(SVD_result, file = "./SVD_result.rda")
} else {
    load("SVD_result.rda")
}
```

Finally, let us Compare RMSE from all previous algorithms.

```r
# Add to rmse_pesults_tb
rmse_pesults_tb <- bind_rows(rmse_pesults_tb, tibble(method = "Singular Value Decomposition",
    RMSE = SVD_result))
kable(rmse_pesults_tb, "latex", booktabs = T) %>% kable_styling(latex_options = "striped")
```

method	RMSE
Just the average	1.06120
Movie Effect	0.94398
User & Movie Effect	0.86590
User & Movie & Genres Effect	0.86560
User & Movie & Genres & Year Effect	0.86535
User & Movie & Genres & Year Regularized Effect	0.86477
Matrix Factorization	0.78282
Singular Value Decomposition	0.97364

4 Results

From the above summary of the results from all algorithms, we can conclude that Matrix Factorization give us the lowest value for RMSE around 0.78 using only two features UserId and MovieId.However the strongest power behind Matrix Factorization is the way of calculating factors for movies and users, as there is a hidden groups and correlations for users and movies reside inside the data.

For example users who like part1 of a movie is expected to like part2 of the same movie. Follow the same concept, by intuition .For example, there should be a group if users who like action movies and most of the time, they will give a high rate for action movies and average or low rate for romance.

Matrix Factorization is discovering these hidden correlations. First by assuming them as coefficients for movies and Users matrix,then predict the value of the ratings. Second, find the difference between predicted vs actual value. Next, calculate the gradient to change the coefficient value and do recalculation of the rating value. Finally, iterating till converge to a certain minimum error. Surprisingly, the above mentioned flow is the basic of first machine learning algorithm gradient descent.

On the other hand, Regression and Regression with Regularization done with basic calculation and using cross validation gave a good RMSE around 0.80 which is not bad and easy to interpret. As both algorithms take into consideration most of the features like movie renege and year of production.

Finally, SVD - Singular Value Decomposition was the lowest performance by RMSE around 0.90. Although,SVD use the same idea like Matrix Factorization. However, Matrix Factorization had achieved the best performance as it take into consideration the factors for users and movies.

5 Conclusion

We have examined different Machine Learning algorithm to predict movie ratings for the 10M version of the Movielens data.

All algorithms gave a better performance or better RMSE results than average rating or guessing. We have seen also that many predicted rating is almost similar to actual rating.

These achieved results, drop the light on the value of Machine Learning for many industries that a big part of their business and revenue depends on do accurate recommendation for their customers about products or services and to deliver that recommendation in a personalized way that actually fulfill personalized customer need. In return, business expect that the customers will buy the recommended products or subscribe in recommended services.

The presented algorithms were implemented on basic PC with 16 GB of RAM. Better results could be achieved using high end PC or server that allows more iterations of cross validation.

With high-end server, there is a possibility to do future work like combining prediction from different algorithms and choose the best performance by Ensembles. This will allow us to check the best results on each movie level and get the most accurate one.

Finally, the value of Machine Learning to many business areas is crucial and required to be utilized effectively. Hence, business can speed growth process and gain more profit through predicting customer needs.

On the other hand, customers will get a personalized products and services that acutely fulfill their needs.

6 References

- Irizzary,R.,2018,Introduction to Data Science,https://rafalab.github.io/dsbook/
- SVDApproximation https://github.com/tarashnot/SVDApproximation
- https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

7 GitHub

- https://github.com/sherif-allam/Recommendation_System_Movielens