Movielens

Arantxa Sanchis

21/06/2020

1. Executive summary

In today's hi-tech digital environment and advanced technology there is a wide variety of products and services available in the market. The flood of these products and services in the market makes it complex for the user to choose. Particularly the movie industry has made tremendous progress and produces a large number of movies of all genres namely – Comedy, Thriller, Action, Drama, Romance, Horror, Sci-fi, Animation etc. in a very short time creating a huge basket of movies for the viewer.

In order to assist the viewer in deciding which movie to see, there is a requirement of developing a recommendation system comprised of ratings given to the movies based on various aspects such as past user preferences, popularity, categories of movies- whether blockbusters, artsy, nerd favorites and so on.

1.1 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 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.

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 suggest it is an excellent movie. We explain the basics of how these recommendations are made, guided by some of the approaches taken by the winners of the Netflix challenges.

In October 2006, Netflix offered a challenge to the data science community: improve our recommendation algorithm by 10% and win a million dollars. In September 2009, the winners were announced.

Summary as below:

http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/ http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

We will now delve into some of the data analysis strategies used by the winning team.

1.2 Goal of the project

To develop a movie recommendation systems that predicts movie ratings using various machine learning algorithms to build efficient models assisted by teachings in the Harvard Course – Data Science & Machine Learning.

1.3 Approach Used

The approach in this project is to train a machine learning algorithm that predicts user ratings using the inputs of a subset "edx" of the MovieLens dataset, in order to predict movie ratings in a "validation" set. (both datasets provided by the teaching assistant). The ratings range from 0.5 to 5 stars.

A well-fitted model results in predicted values close to the observed data values. The metric used to evaluate the algorithm performance is the Root Mean Square Error, or RMSE. RMSE is one of the most widely used measures computing differences between predicted values generated by a model and the observed values.

Theoretically, Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

A lower value of RMSE is considered a better fit. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers.

We will be fit four models which will be compared using their resulting RMSE's in order to evaluate their performance. The evaluation standard for this algorithm is an RMSE of below 0.86490. The most efficient model will be used to predict the movie ratings.

The mathematical notation for the RMSE for a vector of ratings and their respective predictors is:

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

Where $y_{u,i}$ is the outcome having a different set of predictors of ratings for movie i by user u and N is the sample size. $\hat{y}_{u,i}$ is the estimated outcome predicted by the model.

1.4 Key steps

Steps to build the movie recommendation model are broadly summarized as below.

1) Obtain the original Movielens database and split it into two subsets-
    a) "edx": a training subset to train the algorithm
    b) "validation" : a test subset to assess the accuracy of the models we develop

This is done in order to accurately predict the movie rating of the users that haven't seen the movie yet.

2) Further subset the "edx" dataset as below-
    a) a training set "edx_train"
    b) a test set "edx_test"
3) Explore and analyze the data to study the features of the dataset.
4) Train a number of models on the "edx_train" dataset and test the models first on the "edx_test" dataset. Models can be varied based on features/effects.

NOTE: The validation data will NOT be used for training the algorithm and will ONLY be used for evaluating the RMSE of the final algorithm.

5) Compute the RMSE for each model to find the best model (lower RMSE implies higher accuracy).
6) We can use the best model to generate the final results on the "validation" dataset and obtain the final RMSE.

1.5 Dataset

The Netflix data is not publicly available, but the GroupLens research lab generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. We make a small subset of this data.

The MovieLens dataset is available as below:

- [MovieLens 10M dataset] https://grouplens.org/datasets/movielens/10m/
- [MovieLens 10M dataset - zip file] http://files.grouplens.org/datasets/movielens/ml-10m.zip

Note: Due to system constraints in downloading and splitting this very large database, we have been provided with the "edx" training set and "validation" test set by the Harvard teaching staff.

We have uploaded these datasets to "Google Drive" and made it "publicly available to all". We will download them from "Google Drive" into a local folder on our system using the code below. We can load the datasets into R as below:

```r
################################
# Create edx set, validation set
################################
#Installation of R packages which support the project
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(googledrive)) install.packages("googledrive", repos = "http://cran.us.r-project.org")
if(!require(httpuv)) install.packages("httpuv", repos = "http://cran.us.r-project.org")

#Deauthorize i.e do not request for any login credentials
drive_deauth()
drive_user()

#Download datasets from Google drive as below
#edx dataset
downloaded_file_edx <- drive_download(as_id("15Kd9GctAIx4Yl0yy0huY0EobMqgKEyRq"), overwrite = TRUE)
google_file_edx <- downloaded_file_edx$local_path

#Read dataset into RStudio
edx<-readRDS(google_file_edx)

#validation dataset
downloaded_file_val <- drive_download(as_id("10dUZ4iymvcvq6n36pFDgiTmgMgUxNDFQ"), overwrite = TRUE)
google_file_val <- downloaded_file_val$local_path

#Read dataset into RStudio
validation<-readRDS(google_file_val)
```

2. Analysis

2.1 Data Cleaning

Data Cleaning is a process of detecting and correcting (or removing) corrupt or inaccurate records/data from a record set, table or database and refers to identifying incomplete, incorrect, inaccurate parts of the data and then replacing, modifying or deleting the dirty or coarse data. This improves the data quality and in doing so increases overall productivity. We can see that the Movielens table is in tidy format and contains thousands of rows. Each row represents a rating given by one user to one movie.

Check for NAs in the dataset:

```r
na_check <-is.na(edx)
sum(na_check)
```

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

We see that there are no missing values.

2.2 Data Analysis

We can see the first six rows ("head" function) of the dataset "edx" as below. There are six variables – a) "userID" b) "movieID" c) "rating" d) "timestamp" e) "title" f) "genres"

The ratings of a single movie given by a user is depicted by a unique row.

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

We would need to carry out a check that the subset is complete in all aspects. For this we see the summary function to give a "summary" of the subset as below.

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

We will also use the “str” function to view the class of the objects as below.

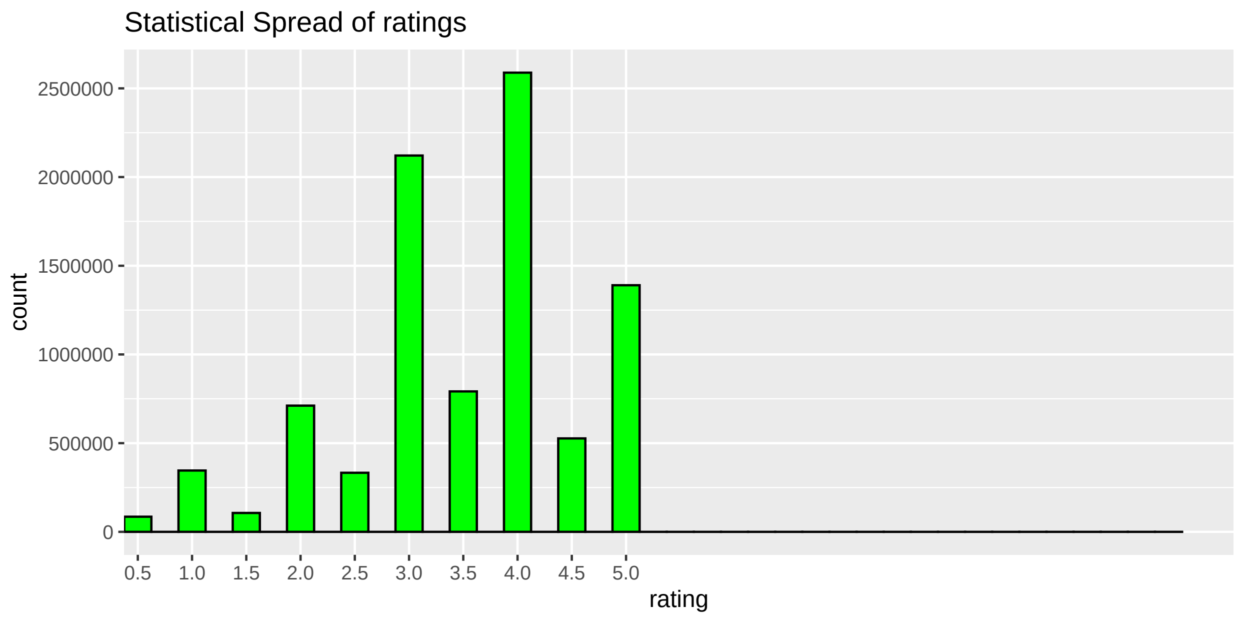
```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

Since in the dataset a user can rate one or more movies implying that a single movie can be rated by many users we would need to ascertain the number of users and the number of movies in the edx subset as below:

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

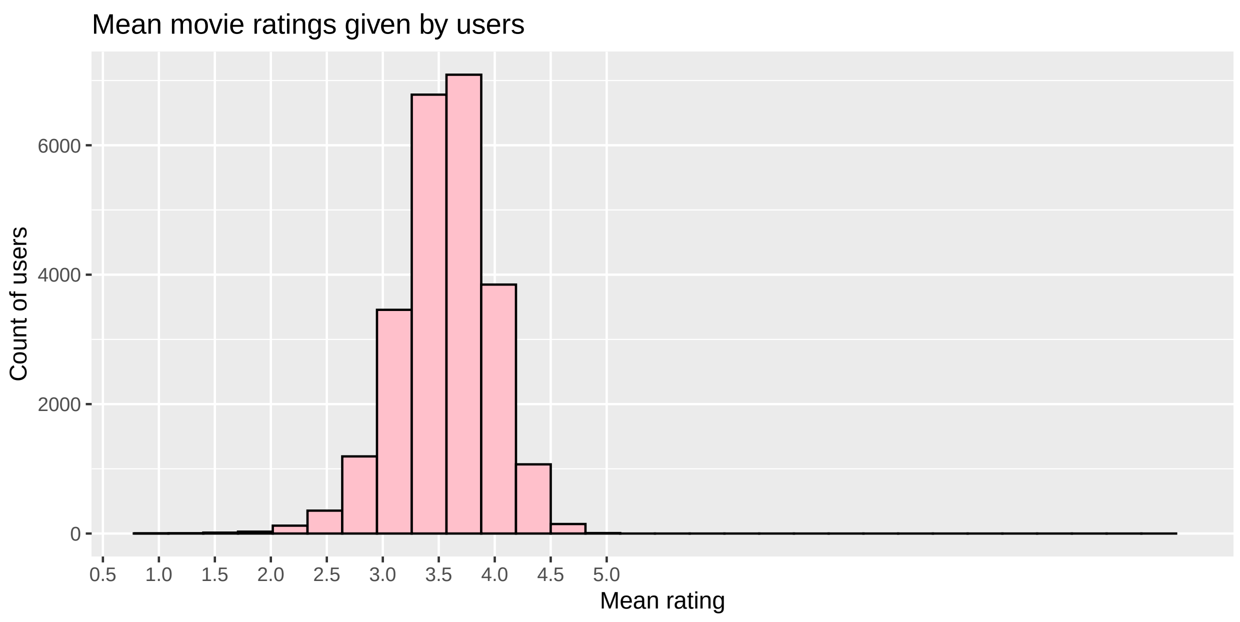
If we study the ratings in the whole dataset compromising of 9,000,055 ratings we can see that users have a tendency to rate movies rather higher than lower. Accordingly it can be seen that the rating value of 4 is the most common and there are very few movies that are rated with a value of 0.5. The ratings are generally in whole numbers such as 1,2,3,4,5. The statistical spread of the ratings is shown below.

```r
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "black", fill = "green") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Statistical Spread of ratings") + theme_gray()
```



If we further examine the distribution for users that have rated at least 100 movies, we see the extremities of user behavior. Some users are critical with their ratings and tend to underrate the movies in comparison to the average. However, on the other hand there are users who tend to overrate the movies in comparison to the average as seen in the below graph. In case these numbers are equal it would balance out else it would need to be given some penalty.

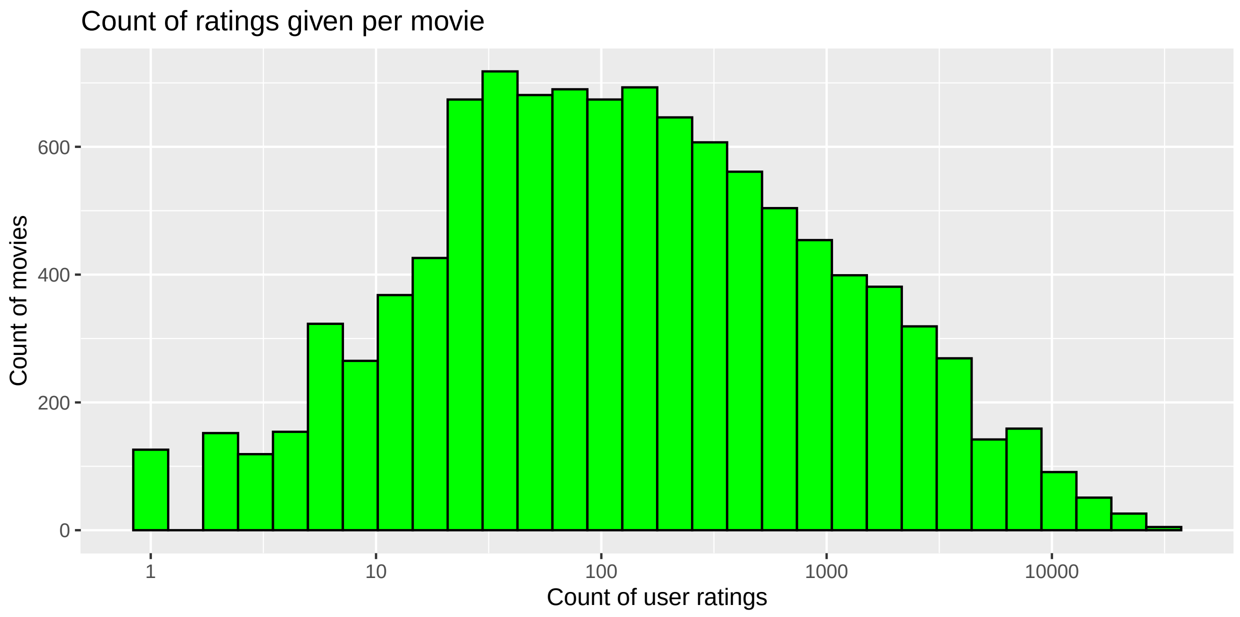
```r
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black", fill="pink") +
  xlab("Mean rating") +
  ylab("Count of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_gray()
```



A study of the number of ratings given per movie show that some movies have been rated much often than others while some have very few ratings. Typically, around 125 movies have received only one rating. This skewness can become misleading and affect the estimate for our predictions.

We will implement regularisation and a penalty term will be applied to the models in this project. Regularizations are techniques used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting (the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably). Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

```r
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "green") +
  scale_x_log10() +
  xlab("Count of user ratings") +
  ylab("Count of movies") +
  ggtitle("Count of ratings given per movie") +
  theme_gray()
```



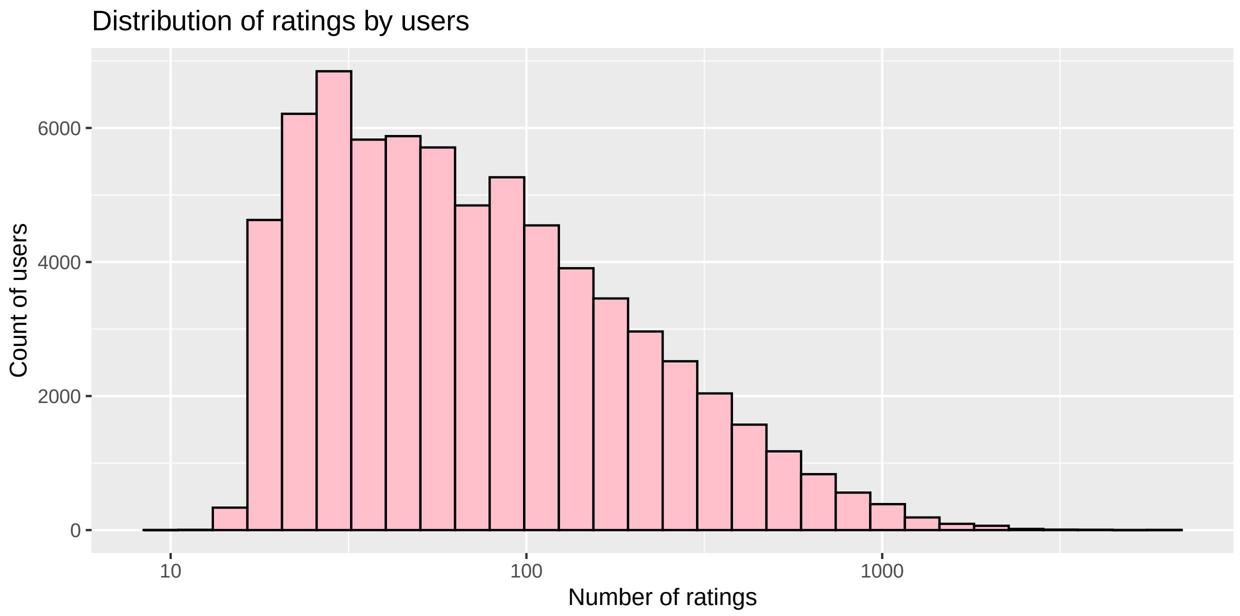
We further explore that the 125 movies that were rated only once appear to be obscure and in our model since they form the opinion of a single person (the rater) the predictions would be misleading. The top 10 best rated by only one user are shown below:

```r
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title)  %>%
  summarize(rating = rating, no_of_ratings = count) %>%
  arrange(desc(rating)) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	rating	no_of_ratings
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Fighting Elegy (Kenka erejii) (1966)	5.0	1
Hellhounds on My Trail (1999)	5.0	1
Shadows of Forgotten Ancestors (1964)	5.0	1
Sun Alley (Sonnenallee) (1999)	5.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Demon Lover Diary (1980)	4.5	1
Kansas City Confidential (1952)	4.5	1
Ladrones (2007)	4.5	1
Man Named Pearl, A (2006)	4.5	1

As depicted in the below graph, a majority of users have rated only between 30 to 100 movies although there are 10,677 movies in the database. Accordingly, a suitable penalty needs to be imposed on for this user effect in our models.

```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill="pink") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Count of users") +
  ggtitle("Distribution of ratings by users") +
  theme_gray()
```



2.3 Modelling approach

The goal of a model is to provide a simple low dimensional summary of a dataset. There are two parts to a model- to define a family of models that are appropriate for our application and finding the best model that fits our application.

2.3.1 Terminology:

a) Model: A machine learning model can be a mathematical representation of a real-world process. To generate a machine learning model we need to provide training data to a machine learning algorithm to learn from.

b) Algorithm: Machine Learning algorithm is the hypothesis set that is taken at the beginning before the training starts with real-world data. For example, when we say Linear Regression algorithm, it means a set of functions that define similar characteristics as defined by Linear Regression and from those set of functions we will choose one function that fits the most by the training data.

c) Training: While training for machine learning, you pass an algorithm with training data. The learning algorithm finds patterns in the training data such that the input parameters correspond to the outcome. The output of the training process is a machine learning model which you can then use to make predictions. This process is also called "learning".

d) Regression: Regression techniques are used when the output is real-valued based on continuous variables. For example, any time series data. This technique involves fitting a line.

e) Classification: In classification, you will need to categorize data into predefined classes. For example, an email can either be 'spam' or 'not spam'.

f) Outcome: The outcome is whatever the output of the input variables. It could be the individual classes that the input variables maybe mapped to in case of a classification problem or the output value range in a regression problem. If the training set is considered then the outcome is the training output values that will be considered.

g) Feature: Features are individual independent variables that act as the input in your system. Prediction models use features to make predictions.

h) Overfitting: An important consideration in machine learning is how well the approximation of the outcome function that has been trained using training data, generalizes to new independant data. Generalization works best if the signal or the sample that is used as the training data has a high signal to noise ratio. If that is not the case, generalization would be poor and we will not get good predictions. A model is overfitting if it fits the training data too well and there is a poor generalization of new data.

i) Regularization: Regularization is the method to estimate a preferred complexity of the machine learning model so that the model generalizes and the over-fit/under-fit problem is avoided. This is done by adding a penalty on the different parameters of the model thereby reducing the freedom of the model.

j) Parameter: Parameters are configuration variables that can be thought to be internal to the model as they can be estimated from the training data. Algorithms have mechanisms to optimize parameters.

These are some key machine learning terms that I thought are important and should be looked into when studying this project.

2.3.2 RMSE

We will be using regression in the models that we are generating. The use of RMSE (as referred earlier) is our measure of model accuracy that will be resorted to in the models.

The loss-function that computes the RMSE is defined as follows:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If it is larger than 1, it implies that our typical error is larger than one star indicating a low accuracy. As such the evaluation standard has been set to < 0.86490 for better accuracy.

The function to compute the RMSE for vectors of ratings and their corresponding predictions is:

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

2.3.3 Split "edx" dataset into train and test sets to train our algorithms

We will split the edx dataset into train and test sets as below:

```r
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-test_index,]
temp <- edx[test_index,]

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

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

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

```r
edx_train <- rbind(edx_train, removed)

rm(test_index, temp, removed)
```

2.4 Models

I. Movie Rating Model based on the Average

The average is a measure of central tendency and is one of the most commonly used parameter that gives one a fair idea of the population. The rest of the values normally converge towards the mean as would be expected in this case.

The simplest possible model predicts the same rating for all movies which is the dataset's mean rating. A model that assumes the same rating for all movies and all users, with all the differences explained by random variation. We start by building the simplest possible recommendation system by predicting the same rating for all movies regardless of user who give it. The expected rating of the underlying data set is between 3 and 4.

The notation is as below:

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

here, $Y_{u,i}$ represents the expected outcome of user 'u' and movie 'i' μ is the "true" (overall average) rating for all movies $\epsilon_{u,i}$ is the independent sampling error from the same distribution (Standard Normal) centered at 0 with Standard Deviation of 1.

This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone.

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

The expected rating of the underlying dataset lies between 3 and 4 as seen below.

```r
mu <- mean(edx_train$rating)
mu
```

```
## [1] 3.512356
```

So we predict all unknown ratings with this average μ or mu and obtain the first naive RMSE (very basic):

```r
naive_rmse <- RMSE(edx_test$rating, mu)
naive_rmse
```

```
## [1] 1.0597
```

We obtain an RMSE of 1.06 which is certainly large compared to our evaluation standard. We will use this naive RMSE to compare with the succeeding models and see how we can improve the RMSE by including some of the imperatives and deductions drawn during our data analysis.

Now we proceed to represent the results table with the first RMSE obtained:

```r
#Store the RMSE results of the first model
rmse_results <- data_frame(Method = "Movie Rating Model based on the Average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
rmse_results %>% knitr::kable()
```

Method	RMSE
Movie Rating Model based on the Average	1.0597

II. Movie Rating Model based on the Movie Effects

We know from experience that some movies are just generally rated higher than others. Higher ratings are mostly linked to popularity of the movies among users and the opposite is true for unpopular movies which usually receive lower ratings. We also see that rating given by professional critics, the cast (popularity of actors and actresses), the credibility of directors and producers and the amount spent on advertisements do have a considerable effect on the movie's ratings.

We can augment our previous model and compute the estimated deviation of each movie's mean rating from the total mean of all the movies μ.

The notation is as below:

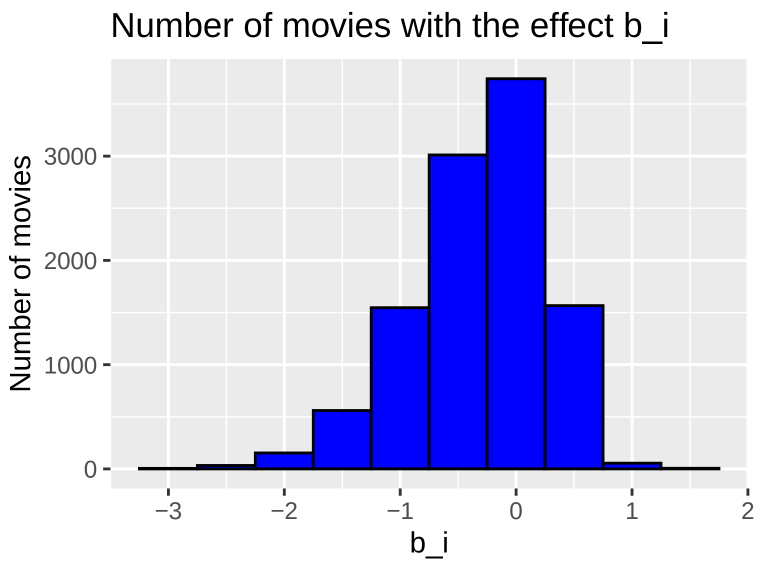
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

The resulting variable is b_i where "b" is referred to as the "bias" (or "b effects" in statistical terms) for each movie "i".

b_i represents the average rating for movie i as each movie gets one parameter, one estimate.

We can obtain a histogram where we see that many movies have negative effects based on the skewness.

```
movie_averages <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_averages %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"),fill=I("blue")
ylab = "Number of movies", main = "Number of movies with the effect b_i")
```



This is because of the penalty term introduced to cater for movie effect.

We can test our model below and will notice that our prediction has improved considerably using this model.

```
predicted_ratings <- mu +  edx_test %>%
  left_join(movie_averages, by='movieId') %>%
  pull(b_i)
model_movie_effect_rmse <- RMSE(edx_test$rating, predicted_ratings)

#Store the RMSE results of the second model
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie Rating Model based on the Movie Effects",
                                     RMSE = model_movie_effect_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Movie Rating Model based on the Average	1.0597002
Movie Rating Model based on the Movie Effects	0.9430962

Our predicted movie rating is based on the fact that movies are rated differently by adding the effect b_i to μ. If an individual movie is on average rated worse than the average rating of all movies μ , we predict that it will be rated lower than μ by effect b_i which is the difference of the individual movie average from the total average.

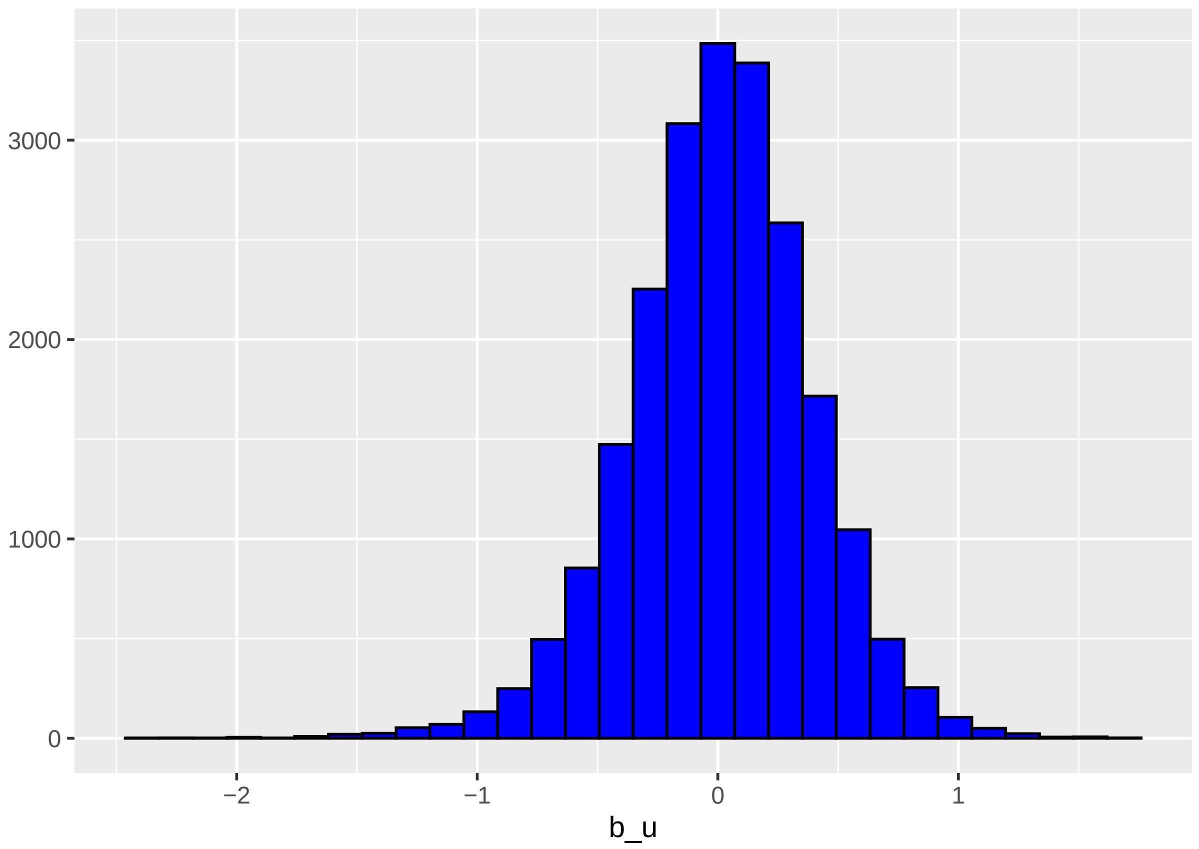
Though our RMSE has improved to 0.943 from before, yet we also need to consider adding the individual user rating effect which will be our next model.

III. Movie Rating Model based on the Movie & User Effects

To explore the user behavior we can calculate the average rating for user μ, for those that have rated over 100 movies and consider this as the penalty term 'user effect'.

We can observe the distribution of 'user effects' from the histogram below:

```
user_averages<- edx_train %>%
  left_join(movie_averages, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_i))
user_averages %>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"), fill = I("blue
```



We see that even users affect the ratings positively or negatively. There is considerable variability across users as well- some users are very cranky and some love every movie. Most of the others are somewhere in the middle.

We can add the user effect to our model as below:

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

where b_u is the user-specific effect. If a cranky user with a (negative b_u) rates a great movie having (positive b_i), the effects then counter each other. We might be able to correctly predict that this user gave a great movie say a 3 rather than a 5.

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

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

```r
user_averages <- edx_train %>%
  left_join(movie_averages, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

We can add this effect to our model to construct the predictors.

```r
predicted_ratings <- edx_test %>%
  left_join(movie_averages, by='movieId') %>%
  left_join(user_averages, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#Store the RMSE results of the third model
model_movie_and_user_effects_rmse <- RMSE(edx_test$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie Rating Model based on the Movie & User Effects",
                                     RMSE = model_movie_and_user_effects_rmse))
rmse_results %>% knitr::kable()
```

Method	RMSE
Movie Rating Model based on the Average	1.0597002
Movie Rating Model based on the Movie Effects	0.9430962
Movie Rating Model based on the Movie & User Effects	0.8646243

The addition of the user-specific effect has improved the RMSE further to 0.865.

We can see the shortcomings in our first model (with movie effects) as below:

- The supposed "best" and "worst" movies were rated by few users, in most cases by just one user implying these were obscure movies.
- As such with a few users, we have more uncertainty.
- As a result we obtained a large estimate of b_i (negative or positive) and thus our RMSE was high.
- These are basically noisy estimates that we should not trust, especially when it comes to prediction.

All our models till now were based on standard error and confidence intervals to account for different levels of uncertainty.

However, when making predictions, we need one number, one prediction, not an interval.

This can be resolved with the help of "regularization".

IV. Regularized Movie Rating Model based on the Movie & User Effects

As discussed previously in the terminology, regularization permits us to penalize large estimates that come from small sample sizes. The general idea is to add a penalty for large values of b_i to the sum of squares equation that we minimize. So having many large b_i, make it harder to minimize.

Regularization is a method used to reduce the effect of overfitting.

So estimates of b_i and b_u are caused by movies with very few ratings and by few users that only rated a very small number of movies. Hence this can strongly influence the prediction. The use of the regularization permits to penalize these aspects.

A tuning parameter (lambda), sometimes called a penalty parameter, controls the strength of the penalty term. It is basically the amount of shrinkage, where data values are shrunk towards a central point, like the mean.

To estimate the b 's, we will now minimize this equation, which contains a penalty term.

$$1/N \sum_{u,i} (y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

The first term is the mean squared error and the second is a penalty term that gets larger when many b 's are large.

The values of b that minimize this equation are given by:

$$\hat{b}_i(\lambda) = 1/(\lambda + n_i) \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

where n_i is a number of ratings b for movie i .

The larger λ is, the more we shrink. λ is a tuning parameter, so we can use cross-validation to choose it. We should be using full cross-validation on just the training set, without using the test set until the final assessment.

We can also use regularization to estimate the user effect. We will now minimize this equation:

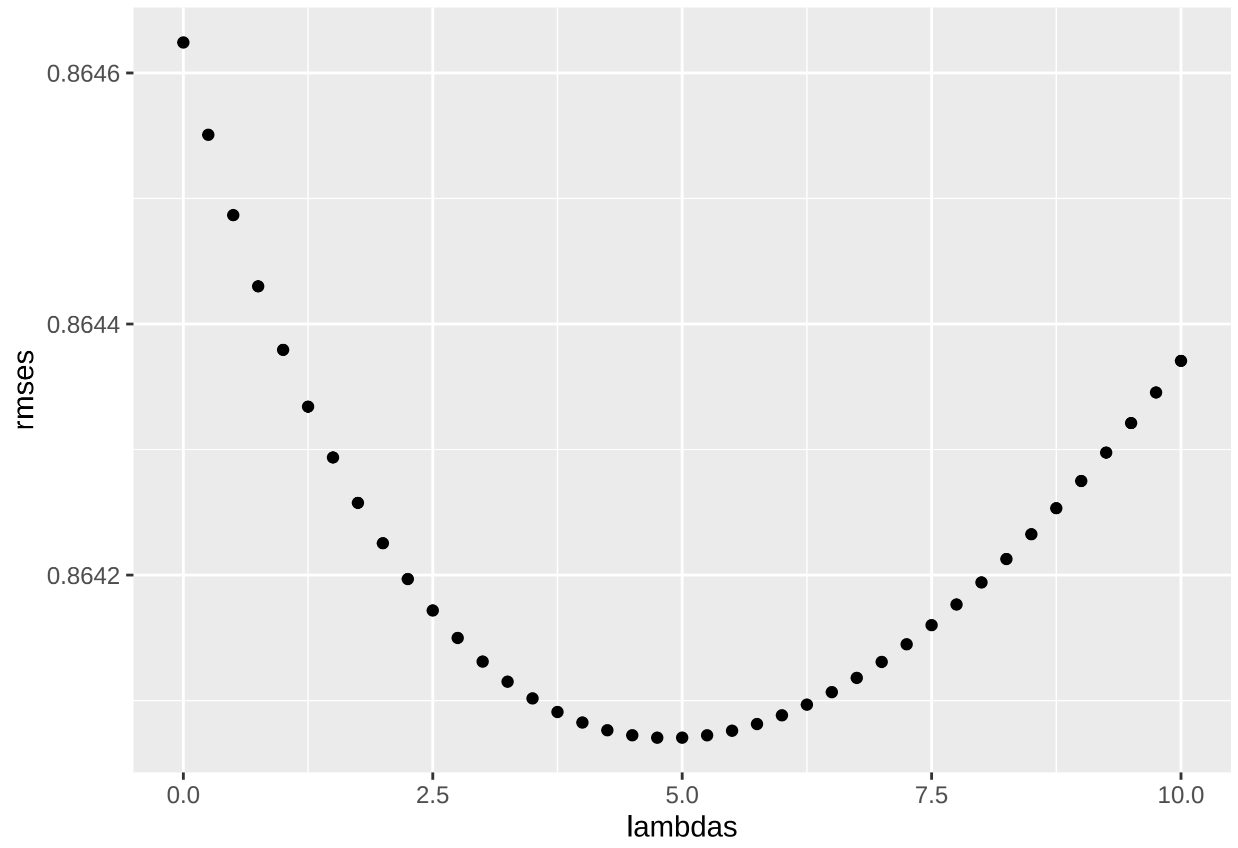
$$1/N \sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

We should find the value of   that will minimize the RMSE. This shrinks the b_i and b_u in case of small number of ratings.

```r
#Try out a range of lambda values
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

  mu <- mean(edx_train$rating)

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

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

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

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

We plot RMSE vs lambdas ('s) to select the optimal lambda ( ).

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



For the full model, the optimal lambda which minimizes the RMSE can be obtained as below:

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

```
## [1] 4.75
```

Next, we apply the optimal lambda to make the prediction.

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

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

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


  regularized_model_movie_and_user_effects_rmse <- RMSE(edx_test$rating, predicted_ratings)

  #Store the RMSE results of the fourth model
rmse_results <- bind_rows(rmse_results,
                data_frame(Method="Regularized Movie Rating Model based on the Movie & User Effects",
rmse_results %>% knitr::kable()
```

Method	RMSE
Movie Rating Model based on the Average	1.0597002
Movie Rating Model based on the Movie Effects	0.9430962
Movie Rating Model based on the Movie & User Effects	0.8646243
Regularized Movie Rating Model based on the Movie & User Effects	0.8640705

The Regularized Movie Rating Model has improved the RMSE even further to 0.864.

V. Matrix Factorization Model

Matrix factorization is a significant approach in many applications. There are supercomputers built to do matrix factorizations. Matrix factorization came into the limelight after the Netflix competition (2006) when Netflix announced a prize money of $1 million to those who will improve its RMSE performance by 10%. As studied in the course, the data can be converted into a matrix such that each user is in a row, each movie is in a column and the rating is in the cell, then the algorithm attempts to fill in the missing values.

Example:

	movie 1	movie 2	movie 3	movie 4	movie 5
user 1	?	3	3	4	?
user 2	3	?	2	2	?
user 3	5	?	4	4	5
user 4	3	3	?	2	?
user 5	4	4	?	3	5

The idea is to approximate a large rating matrix $R_{m\times n}$ into the product of two lower dimension matrices $P_{k\times m}$and $Q_{k\times n}$, such that

$$R \approx P'Q$$

We first need to convert the data in the train set (having one observation per row) to the user-movie matrix before using matrix factorization.

This can be obtained as below however it is difficult to run on regular configuration laptops (code below only for reference).

```r
#train_data <- edx_train %>%
#  select(userId, movieId, rating) %>%
#  spread(movieId, rating) %>%
#  as.matrix()
```

As such we will use the R package “recosystem” which provides ways to decompose the rating matrix and estimate the user rating with the help of parallel matrix factorization.

a) Install “recosystem” package and create a model object by calling “Reco()”
b) We can invoke the “$tune()” method to select the best tuning parameters along a set of possible values.
c) Then we train the model by calling the “$train()"method.Anumberofparameterscanbesetinsidethefunction, saytheposs$
d) Export the model via “$output()”. Here we generate the factorization matrices P and Q into files or return them as R objects.
e) Finally use the “$predict()” method to compute the predicted values.

```r
if(!require(recosystem))
install.packages("recosystem", repos = "http://cran.us.r-project.org")
set.seed(123)

# Convert the train and test sets into recosystem input format
train_data <-  with(edx_train, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_data  <-  with(edx_test,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
# Create the model object
r <-  recosystem::Reco()

# Select the best tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = 4, niter = 10))

# Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9830   1.1058e+07
##    1       0.8749   8.9794e+06
##    2       0.8426   8.3363e+06
##    3       0.8213   7.9569e+06
##    4       0.8052   7.6975e+06
##    5       0.7930   7.5114e+06
##    6       0.7828   7.3649e+06
##    7       0.7741   7.2485e+06
##    8       0.7664   7.1491e+06
##    9       0.7599   7.0698e+06
##   10       0.7541   7.0008e+06
##   11       0.7490   6.9423e+06
##   12       0.7445   6.8933e+06
##   13       0.7404   6.8502e+06
##   14       0.7366   6.8103e+06
##   15       0.7333   6.7785e+06
##   16       0.7301   6.7457e+06
##   17       0.7272   6.7161e+06
##   18       0.7245   6.6929e+06
##   19       0.7221   6.6704e+06
```

```
# Calculate the predicted values
y_hat_reco <-  r$predict(test_data, out_memory())
head(y_hat_reco, 10)
```

```
##  [1] 4.690784 3.997152 3.731637 4.193257 3.502854 3.667658 4.304418 4.425198
##  [9] 4.245836 4.200690
```

```
RMSE_matrix_factorization <- RMSE(edx_test$rating, y_hat_reco)

#Store the RMSE results of the fifth model
rmse_results <- bind_rows(rmse_results,
                data_frame(Method="Matrix Factorization Movie Rating Model",
                           RMSE = RMSE_matrix_factorization))
rmse_results %>% knitr::kable()
```

Method	RMSE
Movie Rating Model based on the Average	1.0597002
Movie Rating Model based on the Movie Effects	0.9430962
Movie Rating Model based on the Movie & User Effects	0.8646243
Regularized Movie Rating Model based on the Movie & User Effects	0.8640705
Matrix Factorization Movie Rating Model	0.7861938

Matrix factorization has substantially improved the RMSE to 0.786.

The training of our algorithms is complete.

2.5 Validation

We have trained five algorithms and found the most suitable one to be the Matrix Factorization Model as it gave us the lowest RMSE.

We will now use the complete "edx" dataset to calculate the RMSE in the "validation" dataset. This will determine the accuracy of the prediction of the movie rating. The project goal is achieved if the RMSE stays below the target.

```r
set.seed(1234)

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

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

#Tune the parameters
opts <-  r$tune(edx_reco, opts = list(dim = c(10, 20, 30),
                                     lrate = c(0.1, 0.2),
                                     costp_l2 = c(0.01, 0.1),
                                     costq_l2 = c(0.01, 0.1),
                                     nthread  = 4, niter = 10))

#Train the model
r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9728   1.2037e+07
##    1       0.8723   9.8842e+06
##    2       0.8402   9.1912e+06
##    3       0.8185   8.7762e+06
##    4       0.8026   8.4892e+06
##    5       0.7906   8.2880e+06
##    6       0.7807   8.1326e+06
##    7       0.7724   8.0133e+06
##    8       0.7653   7.9132e+06
##    9       0.7593   7.8310e+06
##   10       0.7540   7.7643e+06
##   11       0.7493   7.7026e+06
##   12       0.7451   7.6532e+06
##   13       0.7413   7.6093e+06
##   14       0.7378   7.5688e+06
##   15       0.7347   7.5349e+06
##   16       0.7317   7.5031e+06
##   17       0.7291   7.4744e+06
##   18       0.7267   7.4513e+06
##   19       0.7244   7.4287e+06
```

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

RMSE_matrix_factorization_val <- RMSE(validation$rating, y_hat_final_reco)

# Update the result table
rmse_results_final <- data_frame(Method="Matrix Factorization Movie Rating Model",
                                 RMSE = RMSE_matrix_factorization_val)
```

```r
rmse_results_final %>% knitr::kable()
```

Method	RMSE
Matrix Factorization Movie Rating Model	0.7828538

3. Results

We therefore found the lowest value of RMSE.The RMSE value of the final model "Matrix Factorization Movie Rating Model" is as below:

Method	RMSE
Matrix Factorization Movie Rating Model	0.7828538

3.1 Discussion

So we can confirm that the final model "Matrix Factorization Movie Rating Model" for our project is the following:

To approximate a large rating matrix $R_{m\times n}$ into the product of two lower dimension matrices $P_{k\times m}$and $Q_{k\times n}$, such that

$$R \approx P'Q$$

The low value of the RMSE is indicative that we are very near to the true regression line and the error is negligible. Accordingly, the accuracy of the model in predicting movie ratings is quite high.

However, in case we wish to refine it further we can train on more complex models like knn, random forest etc to improve the accuracy.

4. Conclusion

We have developed a systematic step-by-step process to build a robust machine learning algorithm to predict movie ratings with fairly high accuracy from the Movielens dataset.

The "Matrix Factorization Movie Rating Model" is sufficiently robust as it is characterized by a low RMSE value and is hence the optimal model to use for the current project.

The optimal model as seen is characterized by the lowest RMSE value 0.783 which is lower than the initial evaluation criteria (0.86490) given by the goal of the project.

4.1 Future Work

We could propose that improvements in the RMSE could be achieved by adding other effects such as genre, year, age of the movie etc. Other complex machine learning models like knn, random forest etc could also improve the results further, but come with constrains of system limitations, as the RAM, processing speed and storage.

With minor modifications to the R script, we may be able to apply the model to any other large database of movies.