MovieLens Project Submission

Vimal Thomas Joseph

Introduction

The objective of this project is to create a recommendation model based on past rating of users and movies. The focus is to analyze the dataset for correlation, anomalies, biases and relationships present in the data and then build an array of machine learning models by utilizing features present in the dataset and applying machine learning techniques.

The Model generation code consists of the following modules

1) Library Management
2) Function Loading
3) Master Data Preparation
4) Data Analysis and Visualization
5) Creation of training and testing datasets
6) Machine Learning Models based on feature effects
7) Creation of Ensemble

Master Data Creation

Data for the model has been downloaded from grouplens.org. The downloaded dataset consists of the following.

1. User Id - Id of the user.
2. Movie Id - Id of the Movie.
3. Rating - Rating from 0.5 to 5 (in 0.5 interval).
4. Timestamp - Timestamp of the rating in UNIX format.
5. Title - Title of the movie with release year in brackets.
6. Genres - Genres of the movie delimited by pipe.

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

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

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

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

#Validaition Set Creation

set.seed(1, sample.kind="Rounding")

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

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

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

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

```
RMSE<-function(true_rating,predicted_rating){
  sqrt(mean((true_rating-predicted_rating)^2))

}
```

Data Analysis and visualization

Now that we have downloaded dataset and created a validation set, let us take a look at the dataset and analyze the data set for patterns, cleansing requirements and relationships.

This section includes data wrangling, cleansing and gaining insights from visualizations.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

Following data wrangling activities are done in the order.

1) As a first step of data cleaning activity, let us try to get the count of genres present for each movie. The way to acquire number of genres per movie, is to create separate rows per genres of the movie and then join the original table with the genres count dataset.

2) Create a new feature for the movie release year

3) Convert Unix timestamp from the timestamp feature into proper date time in the format of yyyy-mm-dd hh-mi-ss. Add this as a new feature column.

```
#To create multiple rows based on genres data for each movie rating
edx_wrangled<-edx%>%separate_rows("genres",sep = "\\|")

#Adding number of genres for each movie rating
edx_gen_count<-edx_wrangled%>%group_by(movieId,userId)%>%
  summarize(genres_count=n())
```

```
## `summarise()` regrouping output by 'movieId' (override with `.groups` argument)
```

```
#Converting timestamp to proper date format
edx_wrangled1<-edx%>%
  inner_join(edx_gen_count,by = c("movieId","userId"))%>%
  mutate(date = as_datetime(timestamp),
         year=substring(str_extract(title,'\\([0-9]{4}'),2,5))

edx_wrangled<-edx_wrangled1
```

Let us take a look at the data before and after cleansing.

Before Cleanup:

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

After Cleanup:

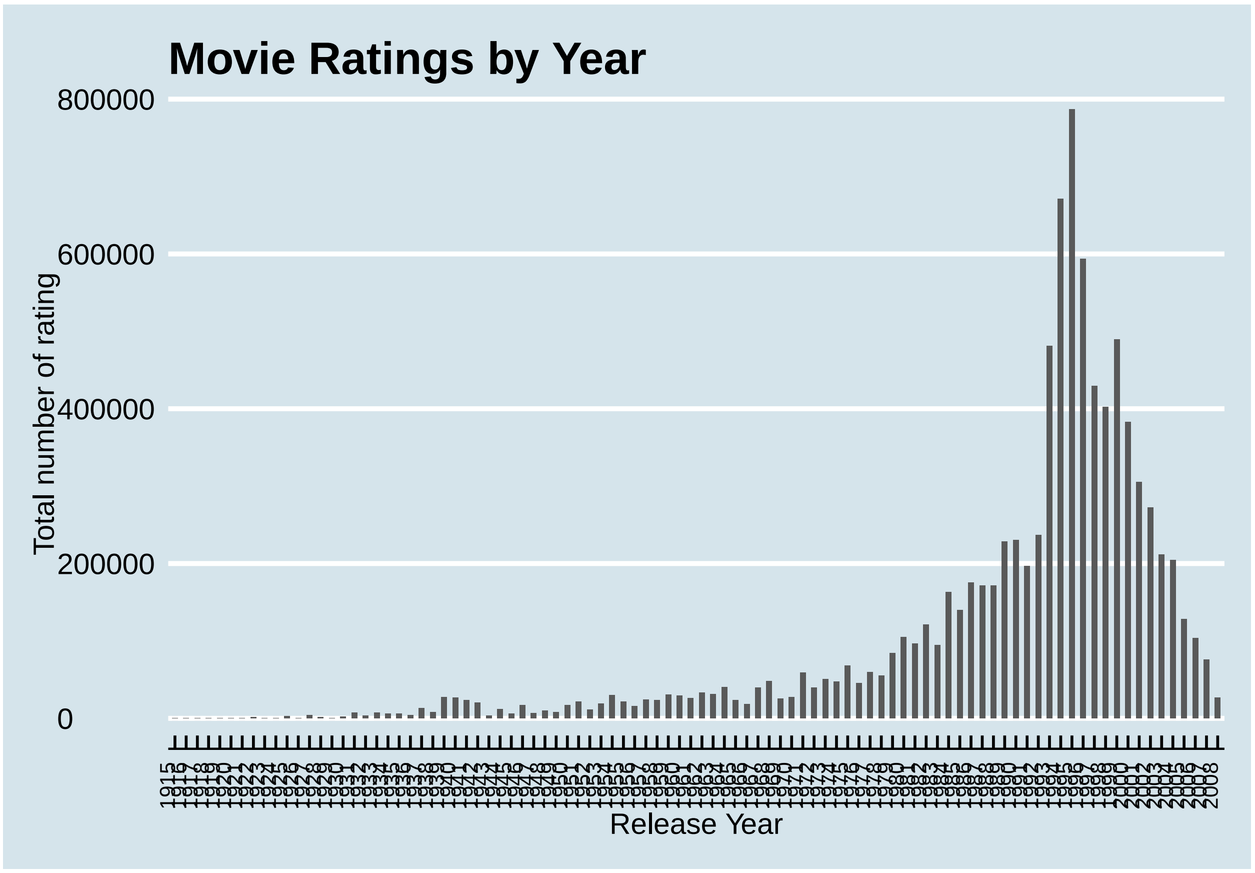
```
##     userId movieId rating                date
##  1:      1     122      5 1996-08-02 11:24:06
##  2:      1     185      5 1996-08-02 10:58:45
##  3:      1     292      5 1996-08-02 10:57:01
##  4:      1     316      5 1996-08-02 10:56:32
##  5:      1     329      5 1996-08-02 10:56:32
##  6:      1     355      5 1996-08-02 11:14:34
##  7:      1     356      5 1996-08-02 11:00:53
##  8:      1     362      5 1996-08-02 11:21:25
##  9:      1     364      5 1996-08-02 11:01:47
## 10:      1     370      5 1996-08-02 11:16:36
##                                        title
##  1:                         Boomerang (1992)
##  2:                          Net, The (1995)
##  3:                          Outbreak (1995)
##  4:                          Stargate (1994)
##  5:             Star Trek: Generations (1994)
##  6:                   Flintstones, The (1994)
##  7:                       Forrest Gump (1994)
##  8:                   Jungle Book, The (1994)
##  9:                     Lion King, The (1994)
## 10: Naked Gun 33 1/3: The Final Insult (1994)
##                                        genres year genres_count
##  1:                            Comedy|Romance 1992            2
##  2:                     Action|Crime|Thriller 1995            3
##  3:              Action|Drama|Sci-Fi|Thriller 1995            4
##  4:                   Action|Adventure|Sci-Fi 1994            3
##  5:             Action|Adventure|Drama|Sci-Fi 1994            4
##  6:                   Children|Comedy|Fantasy 1994            3
##  7:                  Comedy|Drama|Romance|War 1994            4
##  8:                Adventure|Children|Romance 1994            3
##  9: Adventure|Animation|Children|Drama|Musical 1994            5
## 10:                             Action|Comedy 1994            2
```

Insights Gained from Data Analysis:

Looking at the summary and wrangled data, some of the points to note are,

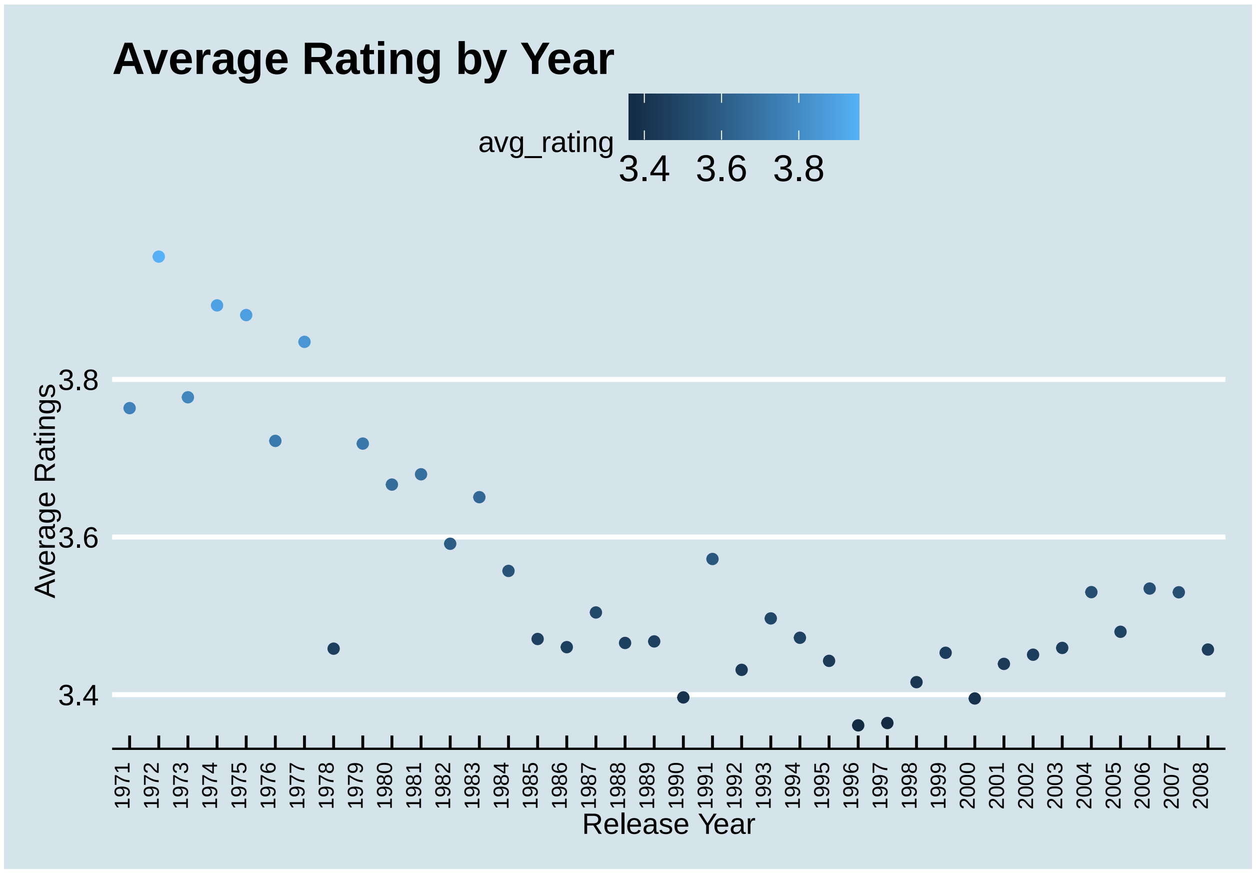
1) average rating is above 3.5.
2) There are no 0 ratings for any movie.
3) There are multiple genres for most of the movies delimited with a pipe.
4) The release year of the movie is embadded in the title.
5) The rating date is given in a timestamp format which should be converted.

To further Analyze the dataset, let us first look at the full picture of these movie ratings.

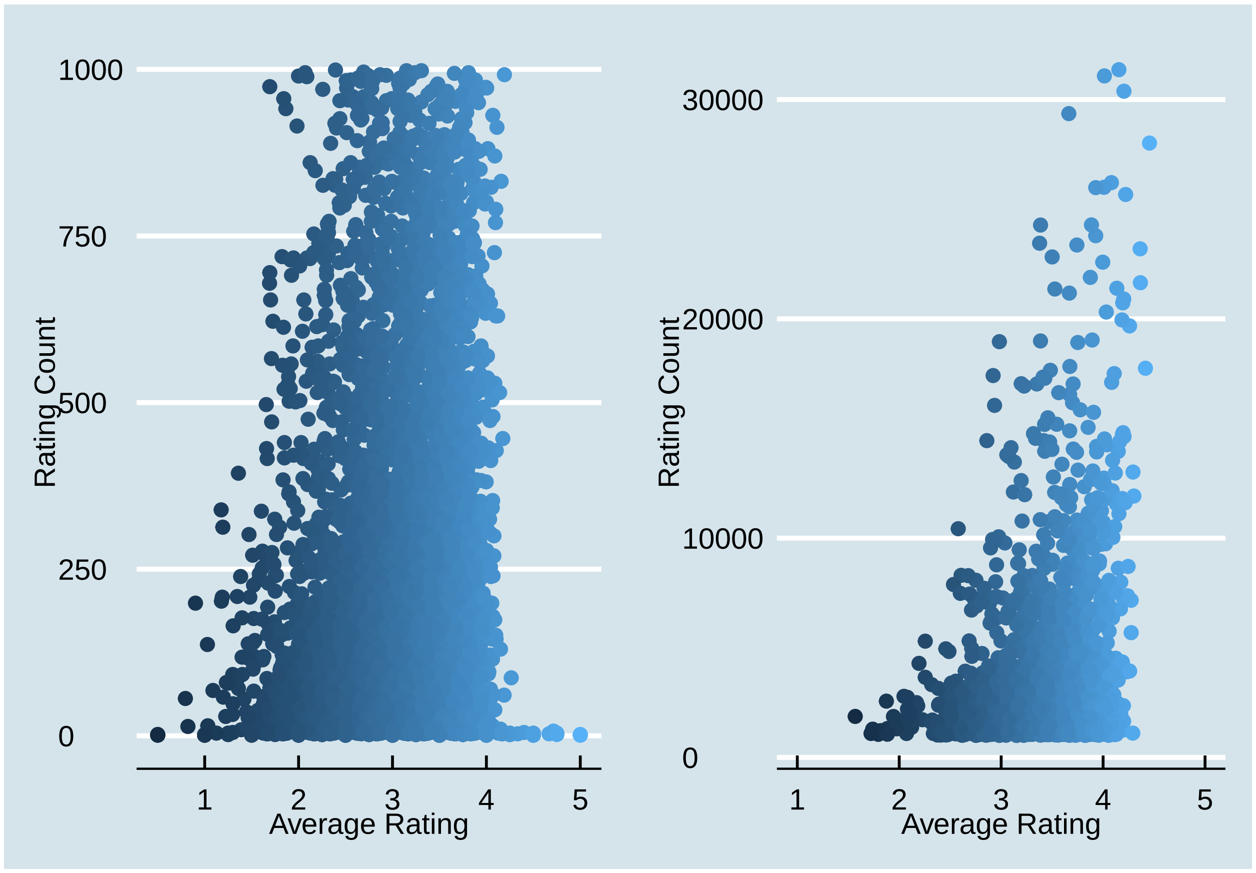


This clearly states that the count of ratings has increased between 1970 and 2000. The downtrend starts from 2000 again. The next look at this dataset would be based on the average rating by year between the specific time period where there is an increase in the number of movie rating 1970 and 2008.

This filter will be applied on the graphs below.



A side by side view of effects based on movies with more user ratings and less user rating.



Model approach

Based on the previous data analysis, it is evident that the ratings are impacted by number of times a movie is rated and number of users that a specific user has rated a number of movies. The model approach is to star adding these biases aka effects into the mean rating of the movies. Hence, let us begin with creating most simple model of the recommendation system

Before building any of the models, let us split edx data into training and testing datasets.

```
set.seed(75, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx_wrangled$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx_wrangled[-test_index,]
test_set <- edx_wrangled[test_index,]

train_set<-train_set%>%mutate(train_set, date = as_datetime(timestamp))
test_set<-test_set%>%mutate(test_set, date = as_datetime(timestamp))

test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set,by='date')%>%
  semi_join(train_set,by='genres')
#dim(train_set)
#dim(test_set)
```

Model 1: Average_Rating_System

```r
mu_hat<-mean(train_set$rating)


#Calculating RMSE for Model 1
model_rmse<-(RMSE(mu_hat,test_set$rating))

RMSE_Table<-data.frame(Model_Name = 'Basic_Average',RMSE_Value=model_rmse)
RMSE_Table
```

```
##      Model_Name RMSE_Value
## 1 Basic_Average   1.088765
```

Model 2 - Average Rating + Movie Effect

```r
mu<-mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu ),.groups = "keep")


prediction_values<-test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i ) %>%
  .$pred

model_rmse<-RMSE(test_set$rating, prediction_values)

RMSE_Table<-rbind(RMSE_Table,data.frame(Model_Name = 'Basic_with_Movie_Effect',RMSE_Value=model_rmse))
RMSE_Table
```

```
##                Model_Name RMSE_Value
## 1           Basic_Average  1.0887650
## 2 Basic_with_Movie_Effect  0.9722858
```

Model 3 - Average Rating + Movie Effect + User Effect

```r
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i),.groups = "keep")

prediction_values <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_rmse<-RMSE(test_set$rating, prediction_values)

RMSE_Table<-rbind(RMSE_Table,data.frame(Model_Name = 'Basic_with_Movie_and_user_effect',RMSE_Value=model
RMSE_Table
```

```
##                         Model_Name RMSE_Value
## 1                    Basic_Average  1.0887650
## 2          Basic_with_Movie_Effect  0.9722858
## 3 Basic_with_Movie_and_user_effect  0.8961433
```

Model 4: Average Rating + Movie Effect + User Effect + Genres Effect

```
#introducing genres effect
genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(g_u = mean(rating - mu - b_i - b_u),.groups = "keep")

prediction_values <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs,by='genres')%>%
  mutate(pred = mu + b_i + b_u + genres_count*g_u) %>%
  .$pred


model_rmse<-RMSE(test_set$rating, prediction_values)

RMSE_Table<-rbind(RMSE_Table,data.frame(Model_Name = 'Basic_with_Movie_User_and_genres_effect',RMSE_Valu
RMSE_Table
```

```
##                                Model_Name RMSE_Value
## 1                           Basic_Average  1.0887650
## 2                 Basic_with_Movie_Effect  0.9722858
## 3        Basic_with_Movie_and_user_effect  0.8961433
## 4 Basic_with_Movie_User_and_genres_effect  0.8986599
```

Model 5: Introducing date effect.

Let us see if the model improves if we add the date effect.

```r
week_avg <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs,by='genres')%>%
  group_by(date) %>%
  summarize(d_ui = mean(rating - mu - b_i - b_u - g_u),.groups = "keep")

week_avg
```

```
## # A tibble: 5,439,266 x 2
## # Groups:   date [5,439,266]
##    date                  d_ui
##    <dttm>               <dbl>
##  1 1995-01-09 11:46:49  0.285
##  2 1996-01-29 00:00:00  0.520
##  3 1996-02-01 14:33:16  0.966
##  4 1996-02-01 14:33:17  1.01
##  5 1996-02-01 14:33:18  0.172
##  6 1996-02-01 14:33:20 -0.485
##  7 1996-02-01 14:33:23 -1.40
##  8 1996-02-01 14:33:24  1.32
##  9 1996-02-01 14:33:25 -2.68
## 10 1996-02-01 14:33:26  0.130
## # ... with 5,439,256 more rows
```

```r
prediction_values <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs,by='genres')%>%
  left_join(week_avg, by='date')%>%
  mutate(pred = mu + b_i + b_u + genres_count*g_u + d_ui) %>%
  .$pred


model_rmse<-RMSE(test_set$rating, prediction_values)

RMSE_Table<-rbind(RMSE_Table,data.frame(Model_Name = 'Basic_with_Movie_user_genres_and_date_effect',RMS
RMSE_Table
```

```
##                                     Model_Name RMSE_Value
## 1                                Basic_Average  1.0887650
## 2                    Basic_with_Movie_Effect  0.9722858
## 3              Basic_with_Movie_and_user_effect  0.8961433
## 4       Basic_with_Movie_User_and_genres_effect  0.8986599
## 5 Basic_with_Movie_user_genres_and_date_effect  1.0751457
```

Model 6: Matrix Factorization with recosystem package

Before diving into the next model, I would like to explain, that the models above are manually trained and predicted based on effects each feature could have on the predicted values.

Another limitation is that the memory and compute threshold our individual systems have is very limited. Due to this, we are unable to auto train these data features using methods like glm, etc.

While, looking for memory and compute efficient r packages, I came across this recomender package, that enabled computation of training and scoring of this dataset within minutes.

The documentation is available at https://github.com/yixuan/recosystem

"recosystem is an R wrapper of the LIBMF library developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin (http://www.csie.ntu.edu.tw/~cjlin/libmf/), an open source library for recommender system using parallel matrix factorization."

```r
#Matrix Factorization with recosystem



recommender<-Reco()

#converting user id and movie id as numeric indexes
train_small<-train_set%>%mutate(userId=as.numeric(userId))%>%select(userId,movieId,rating)
test_small<-test_set%>%mutate(userId,as.numeric(userId))%>%select(userId,movieId)


#creating matrix for train and test sets.
train_small_m<-as.matrix(train_small)
test_set_m<-as.matrix(test_small)

#writing data to an operating system file.
write.table(train_small_m,
            file = "train_set.txt",
            sep = " ",
            row.names = FALSE,
            col.names = FALSE)
write.table(test_set_m,
            file = "test_set.txt",
            sep = " ",
            row.names = FALSE,
            col.names = FALSE)

#reading from the files written
training_dataset <- data_file("train_set.txt")
test_dataset <- data_file("test_set.txt")
predict_dataset<-tempfile()

#training - using default parameters.
recommender$train(training_dataset,opts = c(costp_12=0.1,costq_12=0.1,lrate=0.1,niter=100,nthread=6,verl

#predicting
recommender$predict(test_dataset, out_file(predict_dataset))
```

```
## prediction output generated at /var/folders/3w/ntjy1j594dn2bbfxs3ykybn80000gn/T//RtmpUob8pG/file12fd
```

```r
#write predicted values into pred_rating
pred_rating<-scan(predict_dataset)

model_rmse<-RMSE(test_set$rating, round(pred_rating,1))
```

```
RMSE_Table<-rbind(RMSE_Table,data.frame(Model_Name = 'Matrix Factorization with recosystem',RMSE_Value=

RMSE_Table
```

```
##                                   Model_Name RMSE_Value
## 1                              Basic_Average  1.0887650
## 2                    Basic_with_Movie_Effect  0.9722858
## 3           Basic_with_Movie_and_user_effect  0.8961433
## 4       Basic_with_Movie_User_and_genres_effect  0.8986599
## 5 Basic_with_Movie_user_genres_and_date_effect  1.0751457
## 6         Matrix Factorization with recosystem  0.8605032
```

Model Performances and Fine-Tuning.

Based on the RMSEs for each models, it is evident that adding date effect worsen the RMSE expectation. For now, let us remove date effect from our model list and continue fine-tuning part.

How can we fine-tune the effect-based models?

If we look at the movie rating, there are movies that are rated 1000 times and there are movies that are rated only once. Comparing a movie that gets rated only once and a movie that is rated multiple times on the same scale, introduces additional biases. Similarly, managing user effects where some users have rated more movies, and some have rated only very few on the same scale introduces a similar problem.

Hence, penalizing the effects based on the sample size could solve this problem. However, before settling on one penalizing value - let us call it lambda. We need to determine a lambda that produces a lower RMSE.

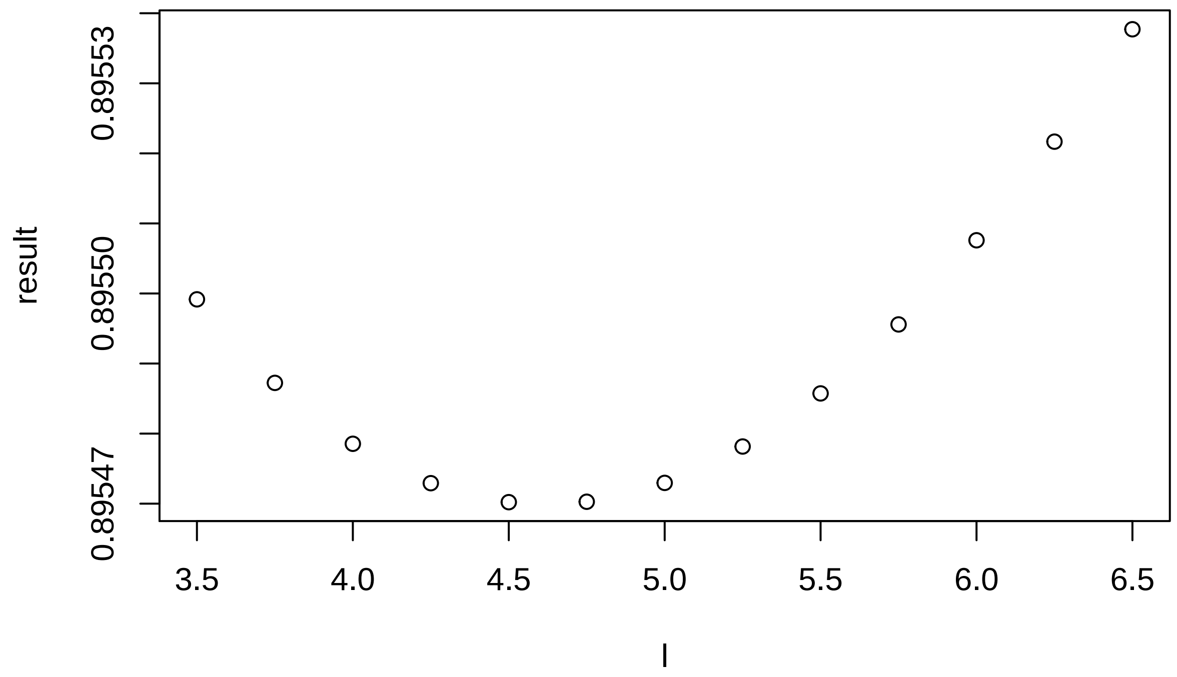
let us write a function that produces a list of lambdas and their RMSEs before settling on one of the best.

The lambda values are going to be chosen from a list of 3.5 to 6.5 with a 0.25 interval values.

```
# determining lamda
l<-seq(3.5,6.5,0.25)

#function to loop through lambda values
result<-sapply(l,function(lam){
  mu<-mean(train_set$rating)
  movie_avgs <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu )/(n()+lam),.groups = "keep")


  #introducing userid effect
  user_avgs <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+lam),.groups = "keep")


  #introducing genres effect
  genres_avgs <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(g_u = sum(rating - mu - b_i - b_u)/(n()+lam),.groups = "keep")

  predicted_values <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    mutate(pred = mu + b_i + b_u ) %>%
    .$pred

  return(RMSE(test_set$rating, predicted_values))
})
```

Let us plot the lamda values and their results

```
plot(l,result)
```



Running finalized models with validation set.

Now that we have seen the optimum lambda value, let us apply that on the validation set.

Final Model based on feature effects with validation set.

```
# determining lamda
lam<-4.75

#function to loop through lambda values

  mu<-mean(edx_wrangled$rating)
  movie_avgs <- edx_wrangled %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu )/(n()+lam),.groups = "keep")


  #introducing userid effect
  user_avgs <- edx_wrangled %>%
    left_join(movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+lam),.groups = "keep")


  #introducing genres effect
  genres_avgs <- edx_wrangled %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(g_u = sum(rating - mu - b_i - b_u)/(n()+lam),.groups = "keep")

  predicted_values <- validation %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genres_avgs,by='genres')%>%
    mutate(pred = mu + b_i + b_u + g_u ) %>%
    .$pred

 RMSE(validation$rating, predicted_values)
```

```
## [1] 0.8644514
```

Additional Final Model 2 with recommender package and validation dataset.

The moderator or peer reviews need not take this RMSE into consideration but I was happy to include this here since I was able to find one automated training and prediction method that does not crash the application for system with smaller memory configuration.

```
recommender<-Reco()

#creating training and test set based on edx and validation set.
train_small<-edx_wrangled%>%mutate(userId=as.numeric(userId))%>%select(userId,movieId,rating)
test_small<-validation%>%mutate(userId,as.numeric(userId))%>%select(userId,movieId)
```

```
#Converting the datasets into matrix
train_small_m<-as.matrix(train_small)
test_set_m<-as.matrix(test_small)

#writing the matrices as files in the operating system to be used in the training and prediction.
write.table(train_small_m,
            file = "train_set.txt",
            sep = " ",
            row.names = FALSE,
            col.names = FALSE)
write.table(test_set_m,
            file = "test_set.txt",
            sep = " ",
            row.names = FALSE,
            col.names = FALSE)

#referring the datasets from file.
training_dataset <- data_file("train_set.txt")
test_dataset <- data_file("test_set.txt")
predict_dataset<-tempfile()

#training using edx data
recommender$train(training_dataset,opts = c(costp_l2=0.1,costq_l2=0.1,lrate=0.1,niter=100,nthread=6,verl

#predicting using validation set
recommender$predict(test_dataset, out_file(predict_dataset))
```

```
## prediction output generated at /var/folders/3w/ntjy1j594dn2bbfxs3ykybn80000gn/T//RtmpUob8pG/file12fd
```

```
pred_rating<-scan(predict_dataset)

RMSE(validation$rating, round(pred_rating,1))
```

```
## [1] 0.8290184
```

Conclusion

The goal of this project is to identify different features associated with the rating and add effects of these features so that the ratings can be predicted. The scale on the model is how low the RMSE can be by adding and fine-tuning the effects.

The pending work on this model is to further enhance the model by

1) performing cross validations,
2) SVDs
3) increasing the system capacity to train on methods from pacakges like caret.