Capstone : Movie Recommendation System

Pratyush Singh

5/5/2021

Contents

Introduction

The world is going virtual agressively and that means everyone is constantly surrounded by products . The biggest marketplace is now accesible with a click . All of these entertainment websites and online store has abundance of choices and products , the challenge is to find the right product for the right customer . That is where Recommendation system becomes important.

Inspired by the Netflix Challenge of 2006 (Where Netflix offered a one million dollar prize for the person or group that could improve their recommendation system by at least 10%.) and its winning model, We will try to find the best model that recommendation model that can help a user find a movie based on their preference and other factors.

We will use the **10M version of the MovieLens dataset** and methods learned throughout HarvardX's Data Science Professional Certificate program.

The Dataset is Divided into two parts edx and validation for training and testing the final model respectively . The edx dataset is further divided into train and test set to evaluate multiple models and select the best one.

For Evaluation RMSE(Root Mean Squared Error) will be used , our goal will be to minimize RMSE value for the train - test set and select the models with least RMSE.

Analysis

We will load required libraries for the project

```r
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(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")
if(!require(ggrepel)) install.packages("ggrepel", repos = "http://cran.us.r-project.org")
library(ggrepel)
library(recosystem)
library(lubridate)
library(tidyverse)
library(caret)
library(data.table)
library(ggthemes)
options(ggrepel.max.overlaps = Inf)
```

```r
load("movielens.RData")
```

MovieLens Dataset

The complete MovieLens dataset consists of 27 million ratings of 58,000 movies by 280,000 users. The research presented in this paper is based in a subset of this dataset with 10 million ratings on 10,000 movies by 72,000 users.This Dataset was generated by GroupLens research lab.

We will first Download the MovieLens Dataset and then Prepare it by creating two datasets edx and validation. edx will be 90% of Movielens dataset while validation will be 10%.

```r
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

```r
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")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

```r
dim(edx)
```

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

```r
dim(validation)
```

```
## [1] 999999      6
```

We will divide the edx dataset into train test set to evaluate different model and generate final RMSE using cross validation. The train set will be 80% of edx while test set will be 20% .

```r
#Train / Test split of edx dataset
test_index <- createDataPartition(y = edx$rating , times = 1 , p = 0.2 ,
                                  list = F)
test_set <- edx %>% slice(test_index)
train_set <- edx %>% slice(-test_index)
```

```r
# Make sure userId and movieId in test set are also in train set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

```r
dim(train_set)
```

```
## [1] 7200043       6
```

```r
dim(test_set)
```

```
## [1] 1799967       7
```

Exploration

```r
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

We see that edx has 6 columns among which we have 5 predictors and rating column is our target outcome(y). The dataset has timestamp , genres , title for each movie and user identified by movieId and userId

We will now explore each of these columns to gain insight .

```r
length(unique(edx$movieId))
```

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

```r
#Distribution of Ratings by Movies
edx %>% group_by(movieId) %>%
  summarise(n = n())%>% ggplot(aes(n) ) + geom_histogram(bins =30 ,color = 'black' , fill = 'orange') +
  scale_x_log10()  + scale_color_hc() + xlab('No. of Ratings') +
  ylab('No. of Movies') + ggtitle("Distribution of Ratings by Movies") + theme_hc()
```

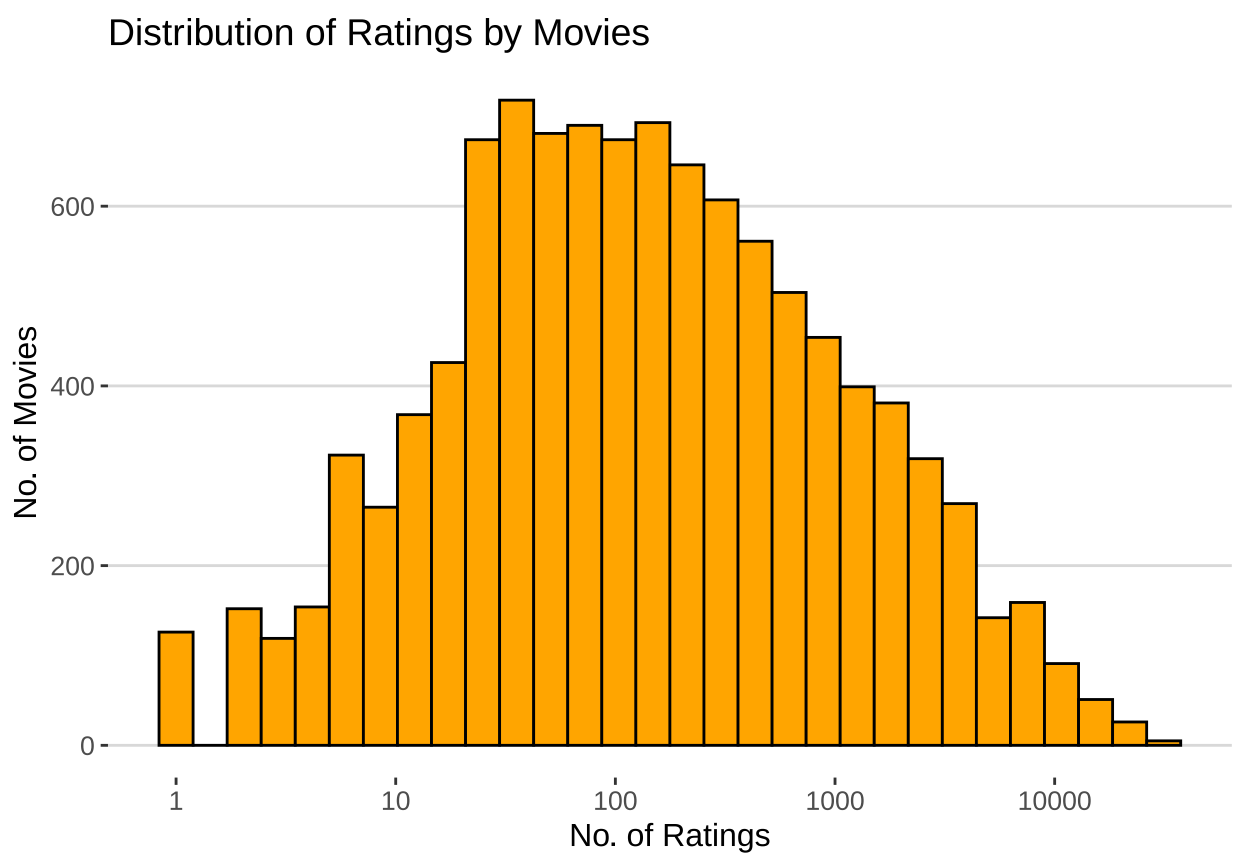


Figure 1: Movies

The Distribution of movies is almost symmetric . We can clearly gather that different movies aren't rated similarly with more than 10% of the movies rated less than 10 times. It is evident that there is movie specific effect on ratings

```
length(unique(edx$userId))
```

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

```
#User Specific effect on ratings(b_u)
edx %>% group_by(userId) %>%
  summarise(n = n()) %>% ggplot(aes(n)) +
  geom_histogram(color = 'black' , fill = "skyblue" , bins = 30) +
  scale_x_log10()  + xlab('No. of Ratings') + ylab('No of Users') + ggtitle("Distribution of Ratings by
  theme_hc()
```

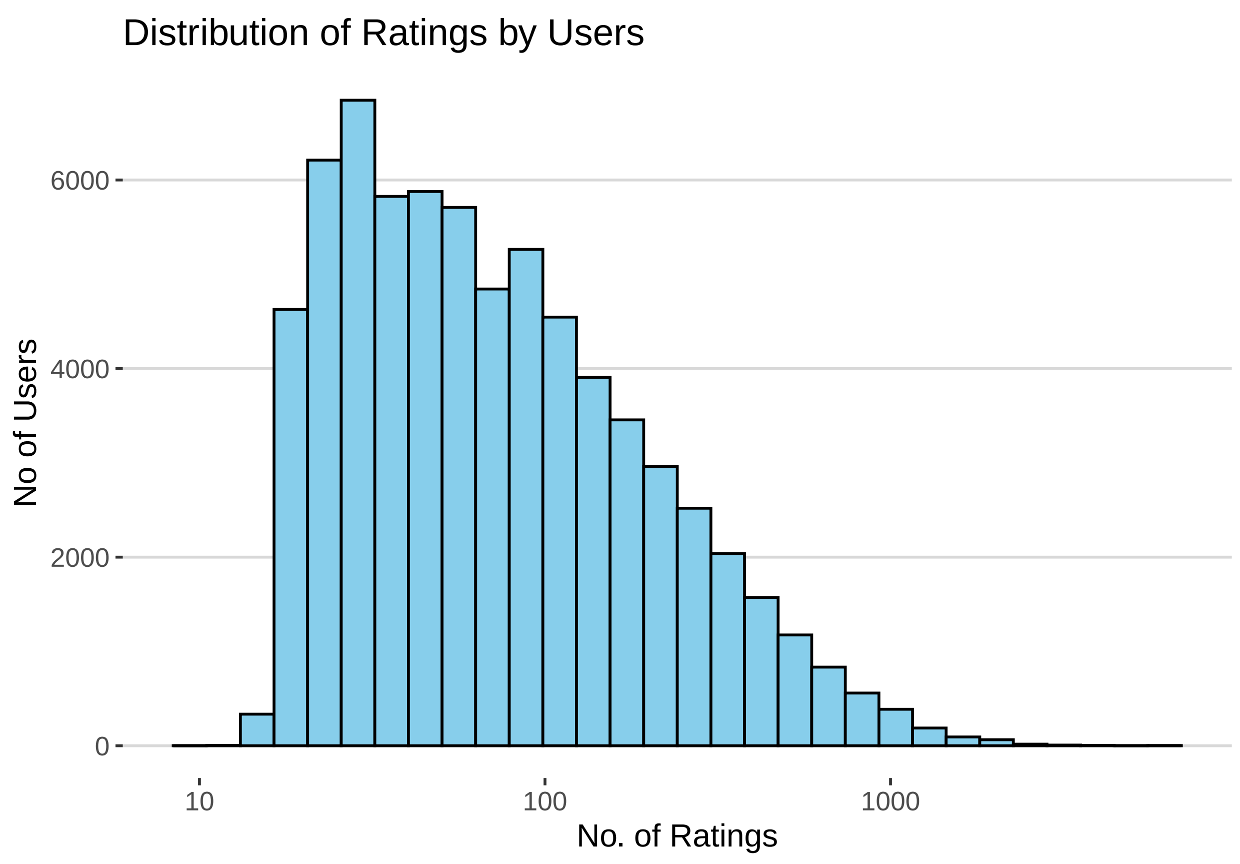


Figure 2: Users

Some users are more active that others with only around 10% users rating more that 1000 movies and 5% users rating more less than 20 movies . This shows user effect on ratings

We will convert timestamp into Datetime format and create a new column named date in the edx set for further analysis .

```r
edx <- mutate(edx, date = as_datetime(timestamp))
```

```r
edx %>%
  mutate(year = round_date(date  , unit = 'year')) %>%
  group_by(movieId , year) %>%
  summarise(n = n()) %>%
  ggplot() + geom_boxplot(aes( year , n , group = year)) +
  scale_y_sqrt() + ylab("No. of Ratings")+ ggtitle("Average Number of Ratings per   Year")
```

```
## `summarise()` has grouped output by 'movieId'. You can override using the `.groups` argument.
```

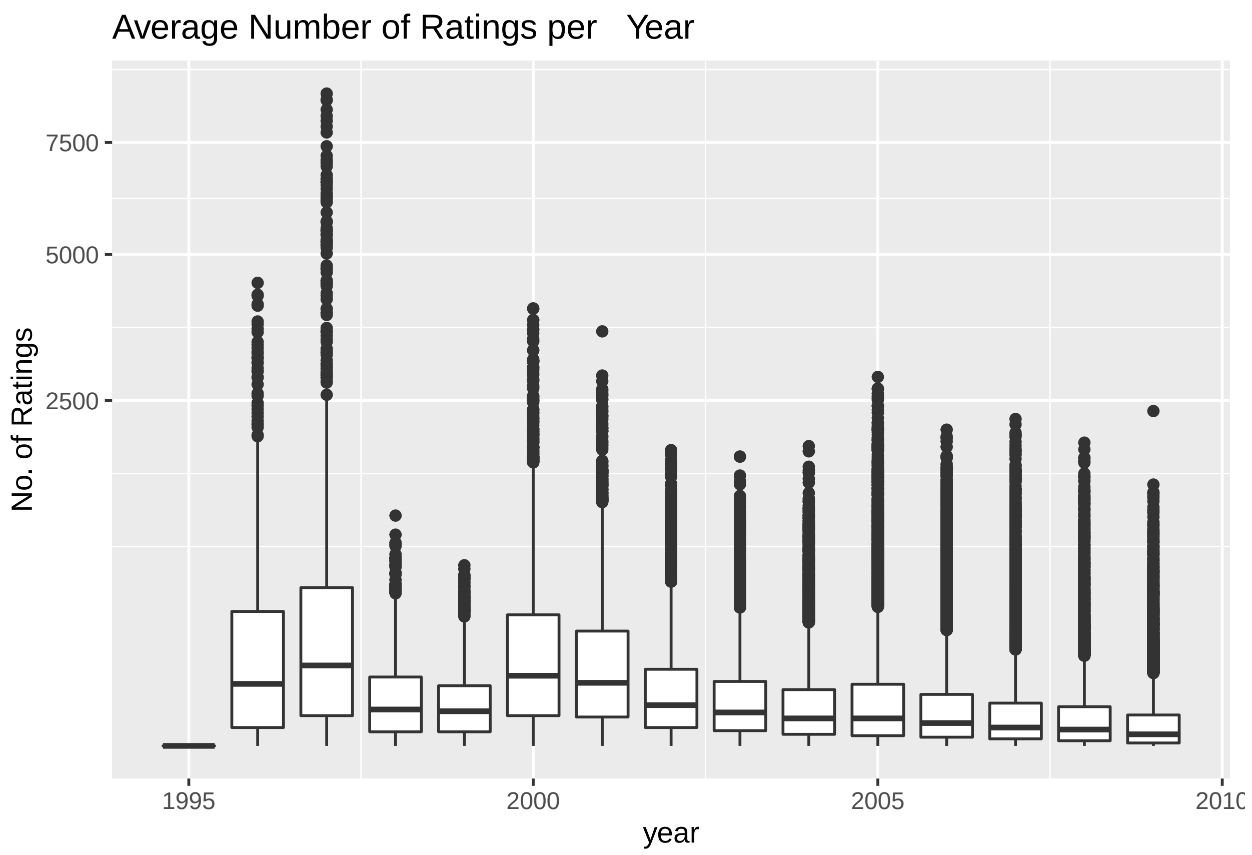


Figure 3: No of Ratings by Year

There does not seem to be an effect of year on ratings as movies that released after 2005 will have lower rating beacuse of lesser reviews (due to less time after release) and movies before 1990 also have less reviews.

```
edx %>% mutate(date = round_date(date  , unit = 'week' , week_start = getOption('lubridate.week.start',
  group_by(date) %>% summarise(n = n()) %>%
  ggplot(aes(date , n)) + geom_point(color = 'darkgreen') +
  geom_smooth() + theme_hc() + xlab('Date(unit - week)') + ylab('No. of Ratings') + ggtitle("Weekly Eff
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

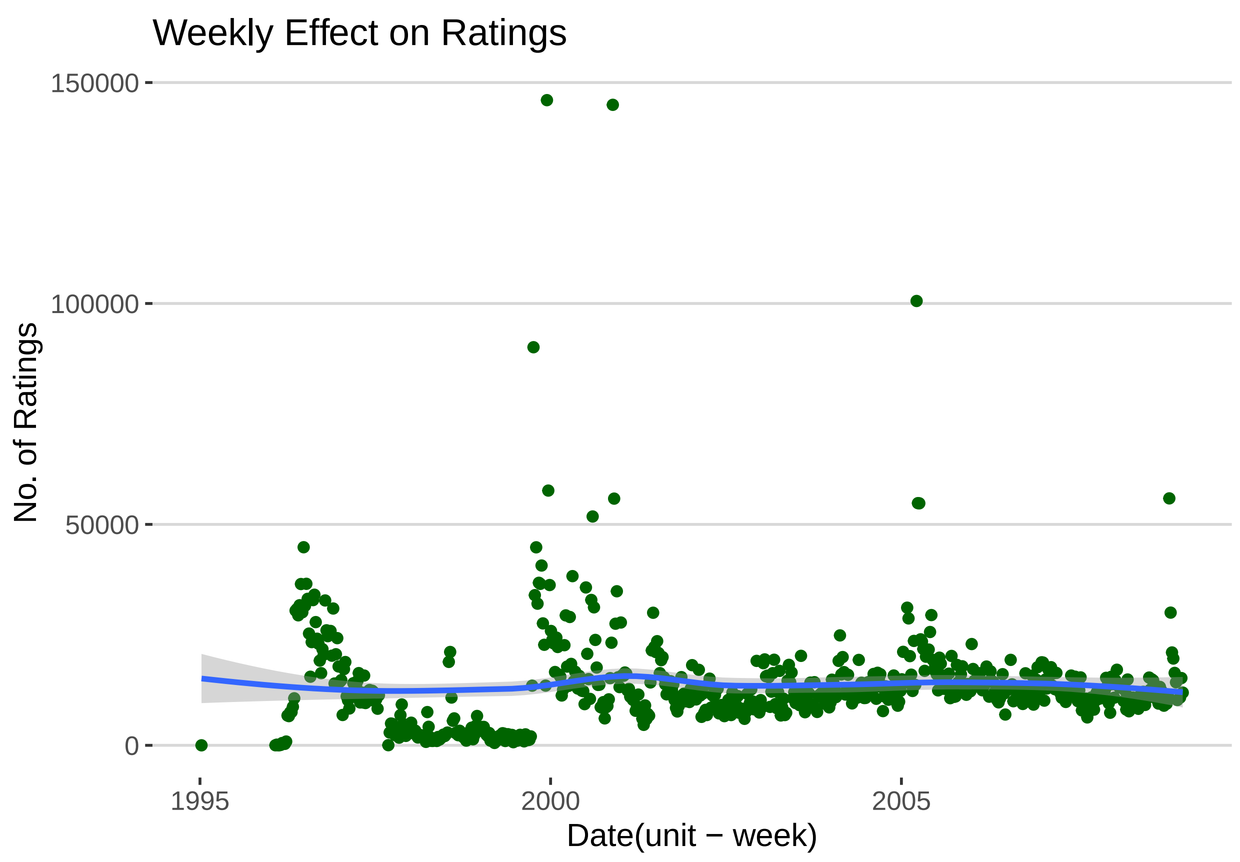


Figure 4: Ratings by year

It is seen that weekly , there is a small effect of time on ratings .Later on we can see how much this effects improves on overall RMSE.

```
edx %>%
  group_by(rating) %>%
  summarise(n = n()) %>% arrange(desc(n))
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

```
edx %>%
  ggplot(aes(rating)) + geom_histogram(color = 'Black' , fill = 'yellow' ,    bins = 30) + theme_hc() +
```

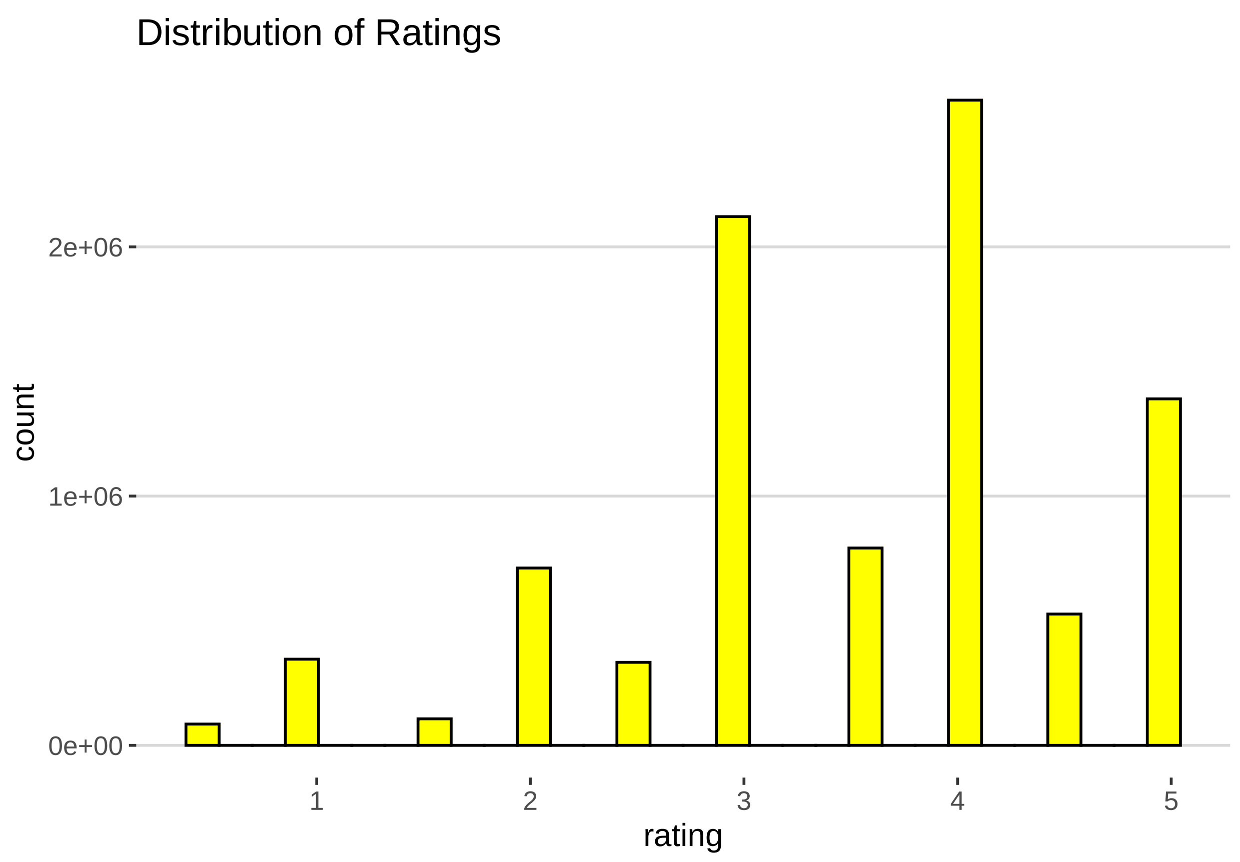
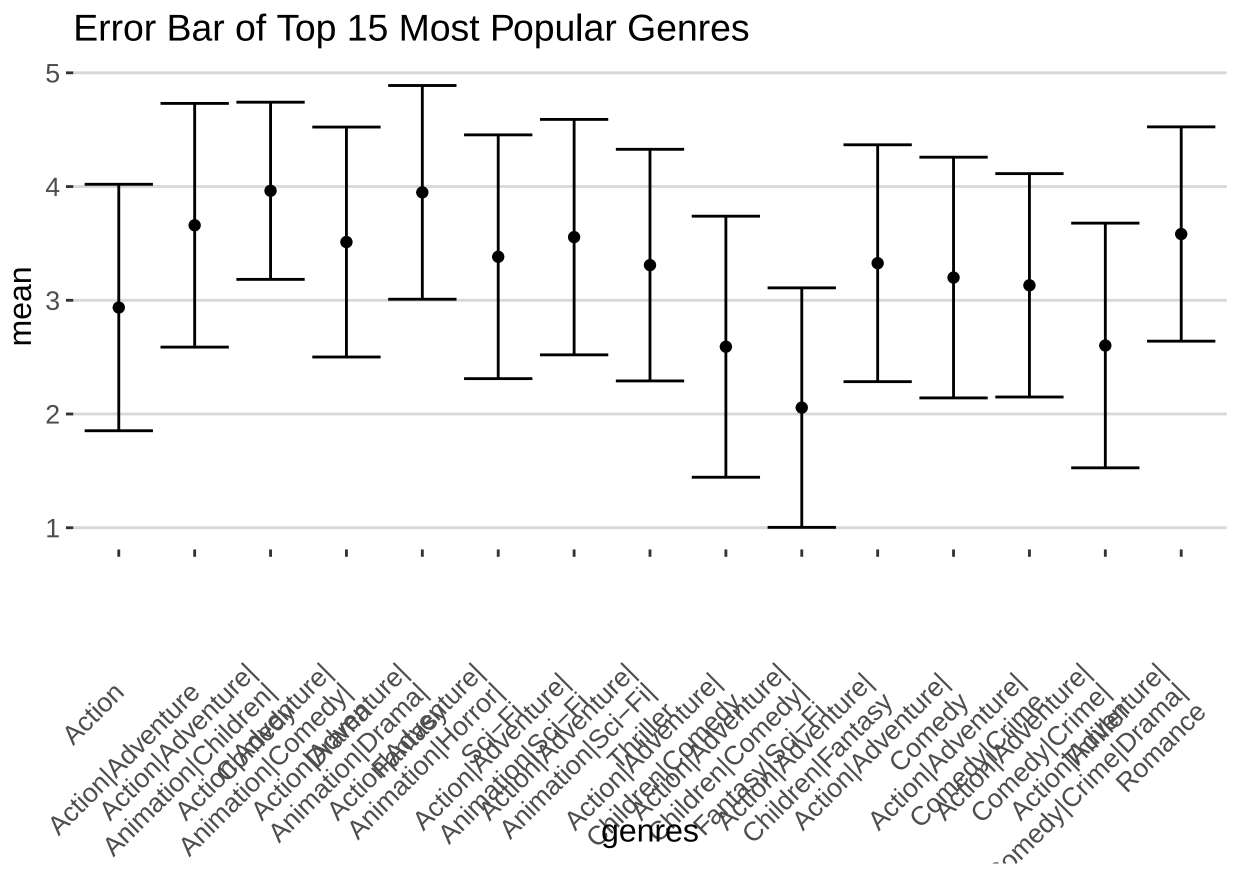


Figure 5: Ratings

Users rate round numbers more than half ratings and no user gives 0 ratings . Also 4 stars rating is the most popular among the lot.

```
genres_df <- edx %>%
  group_by(genres) %>%
  summarise(n = n()) %>% filter(n > 1000)

edx %>% filter(genres %in% genres_df$genres) %>%
  group_by(genres) %>%
  summarise(mean = mean(rating) , sd = sd(rating)) %>% slice(1:15) %>%
  mutate(genres = str_wrap(genres , width = 20)) %>%
  ggplot(aes(genres , mean)) + geom_point() +
  geom_errorbar(aes(ymin = mean - sd , ymax = mean+sd)) +
  theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=1)) +
  theme_hc() + ggtitle("Error Bar of Top 15 Most Popular Genres")
```



Genres also seems to have strong effect on ratings as some genres are rated higher than others on average.

Principal Component Analysis

The Decomposition in matrix factorization is very much related to PCA . We will divide a small subset of our data into different components to understand how it divides our train set into different components explaining variability

```
# subsetting train set
train_small <- train_set %>%
  group_by(movieId) %>%
  filter(n() >= 500 ) %>% ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 500 ) %>% ungroup()

#creating a matrix for PCA
y <- train_small %>%
  select(movieId , userId , rating) %>%
  spread(movieId , rating) %>% as.matrix()

rownames(y) <- y[,1]
y <- y[,-1]

movie_titles <- train_small %>%
  select(movieId , title) %>% distinct()
```

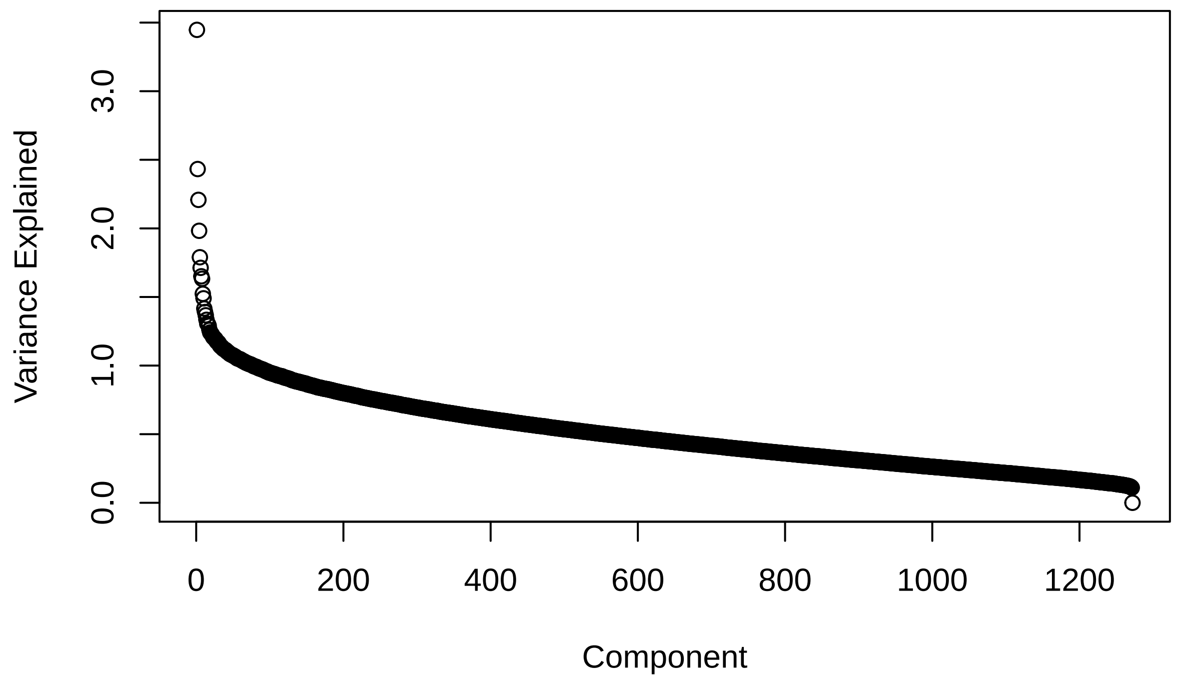
```
colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])

# We subtract the row wise and column wise average of each user movie pair to normalize data
y <- sweep(y , 2 , colMeans(y ,na.rm = T))
y <- sweep(y , 1 , rowMeans(y ,na.rm = T))

#converting Nas to 0
y[is.na(y)] <- 0

pca <- prcomp(y)
```

```
plot(pca$sdev , ylab = "Variance Explained" , xlab = "Component")
```



The matrix is decomposed to over 1200 components . But the variance decreases significantly after only a few components.

```
var_exp <- cumsum(pca$sdev^2/sum(pca$sdev^2))
plot(var_exp , cex = 1)
lines(x = c(0,1300), y = c(.90, .90))
```

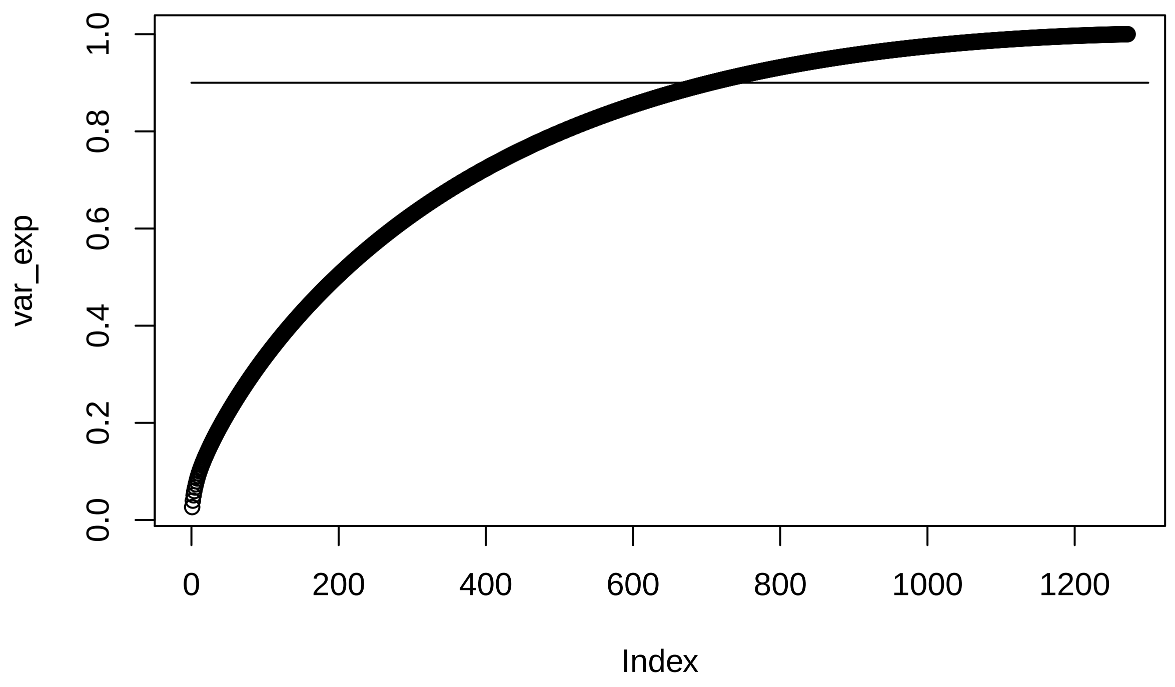


Figure 6: Variance Explained

We can see that 90% of the variance is explained by around 700 components out of 1200 . This number will further decrease if complete dataset is used .

```
pcs <- data.frame(pca$rotation, name = colnames(y))
pcs %>%
  ggplot(aes(PC1 , PC2 )) + geom_point() +
  geom_text_repel(aes(PC1 , PC2 , label = name , color = 'green') ,
                data = filter(pcs ,
                              PC1 < -0.1 | PC1 > 0.1 |
                                PC2 < -0.075 | PC2 > 0.05)) +
  theme_hc() + guides(color = F) + ggtitle("Principal Components Analysis")
```

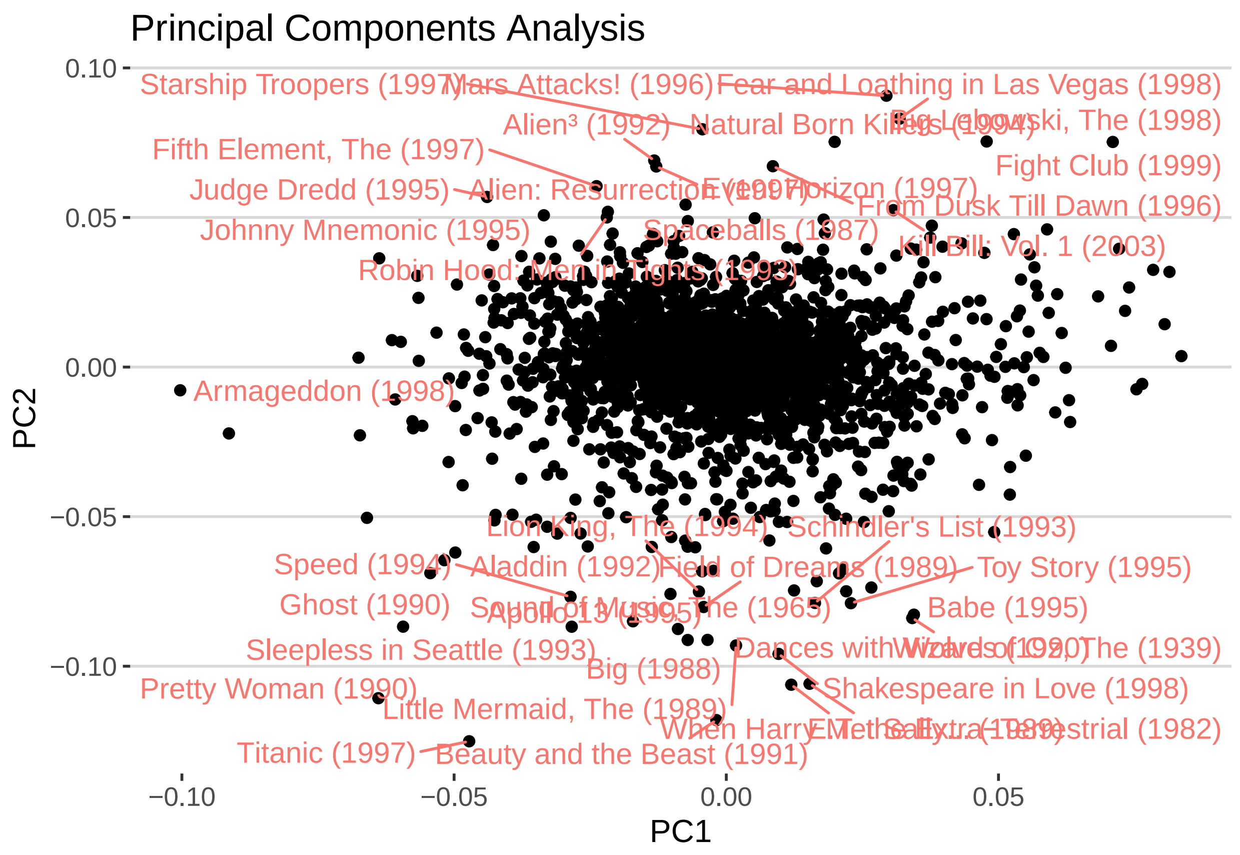


Figure 7: PC1 v/s PC2

Examining the first two components we can see that the first component divides critically acclaimed movie to those who weren't recieved that well while the second component divides blockbusters to smaller earning movies.

Modeling

Linear Model

Our first and simplest model predicts the same rating for all movies regardless of user. A model that assumes the same rating for all movies and users with all the differences explained by random variation is:

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

Where

$$\epsilon_{u,i}$$

is the independent errors and

$$\nu$$

is the "True" prediction for all movies .

We know from our data exploration that there is a strong movie specific effect on the ratings (i.e, different movies are rated differently). We will alter our previous model to take this effect in account

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

Where

$$b_i$$

is the movie effect .

We have also found out that different users rate movies differently and so we can add user effect to our model .

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

here

$$b_u$$

is the user effect .

Similarly we can adjust our model to account for other effects like genres and time

$$Y_{u,i} = \nu + b_i + b_u + f(d_{u,i}) + \epsilon_{u,i}$$

here

$$d_{u,i}$$

is the time effect (rounded timestamp to weekly in the project) with a smooth function over it.

Although there is a strong effect of genres on ratings which is complex to account for , we do not use genre effect in our model.

Regularization

Movies are rated with high variablity i.e, there are alot of movies which are rated only once or twice and users who have rated only one or two movies . This can cause in low precision due to predictions made from these observations.

The general idea behind regularization is to constrain the total variability of the effect sizes.

Penalized regression can control the total variability of the movie effects and user effects and reduced estimate error in prediction.

$$b_i = \frac{1}{n+l} \sum_{i=1}^{n_i} (y_{u,1} - u)$$

$$b_u = \frac{1}{n+l} \sum_{i=1}^{n_i} (y_{u,1} - b_i - u)$$

This approach will have our desired effect: when our sample size

$$n_i$$

is very large, a case which will give us a stable estimate, then the penalty

$$\lambda$$

is effectively ignored since

$$n_i + \lambda \sim n_i$$

However, when the

$$n_i$$

is small, then the estimate

$$b_i(\lambda)$$

is shrunken towards 0. The larger

$$\lambda$$

, the more we shrink. We will use regularization on both our movie and user effects to improve our RMSE .

Matrix Factorization

Our model accounts for movie differences and user differences. But this model leaves out an important source of variation related to the fact that groups of movies have similar rating patterns and groups of users have similar rating patterns as well .

matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. One matrix can be seen as the user matrix where rows represent users and columns are latent factors. The other matrix is the item matrix where rows are latent factors and columns represent items.

The general idea of matrix factorization is to decompose the matrix

$$R_{mXn}$$

into two matrices of lower dimension

$$P_{kXm}$$

and

$$Q_{kXn}$$

such that :

$$R \sim P^{'}Q$$

In this report we will use recosystem package in R which decompose the rating matrix and estimate the user rating, using parallel stochastic gradient descent .

Evaluation

For evaluation of different models in this project we will use Root Mean Squared Error method to select the best model.

$$RMSE = \sqrt{\frac{1}{N}\sum u,1(\widehat{y_{u,i}} - y_{u,i})}$$

The model which minimizes this equation would be the best and hence used to generate final model on validation set

```r
# RMSE function for evaluation throughout the project
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Results

Linear Models

```
#overall mean of ratings for train set
mu <- mean(train_set$rating)

# RMSE using just the overall average of train set
naive_rmse <- RMSE(test_set$rating , mu)
naive_rmse
```

```
## [1] 1.060289
```

```
# Movie effect on ratings in the train set
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

# adding movie effec(b_i) on previous model
y_hat_b_i <- mu + test_set %>%
  left_join(movie_effect , by = 'movieId') %>%
  .$b_i

model1_rmse <- RMSE(test_set$rating , y_hat_b_i)
model1_rmse
```

```
## [1] 0.9440672
```

```
# user effect on ratings in train set
user_effect <- train_set %>%
  left_join(movie_effect , by= 'movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

# Adding combined effect of b_u & b_i to linear model
y_bi_bu <- test_set %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>% .$pred

model2_rmse <- RMSE(test_set$rating , y_bi_bu)
model2_rmse
```

```
## [1] 0.8665681
```

Adding user and movie effect significantly improve the RMSE over our naive model.

```
time_effect <- train_set %>%
  left_join(user_effect , by = 'userId') %>%
  left_join(movie_effect , by = 'movieId') %>%
  mutate(date = round_date(as_datetime(timestamp)  , unit = 'week' )) %>%
  group_by(date) %>%
  summarise(b_t = mean(rating - mu - b_i - b_u))

#Combined effect mu + b_u + b_i + b_t
test_set <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))


y_hat_buit <- test_set %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  left_join(time_effect , by= 'date') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>% .$pred

# Not much improvement in RMSE considering time effect
model3_rmse <- RMSE(test_set$rating , y_hat_buit)
model3_rmse
```

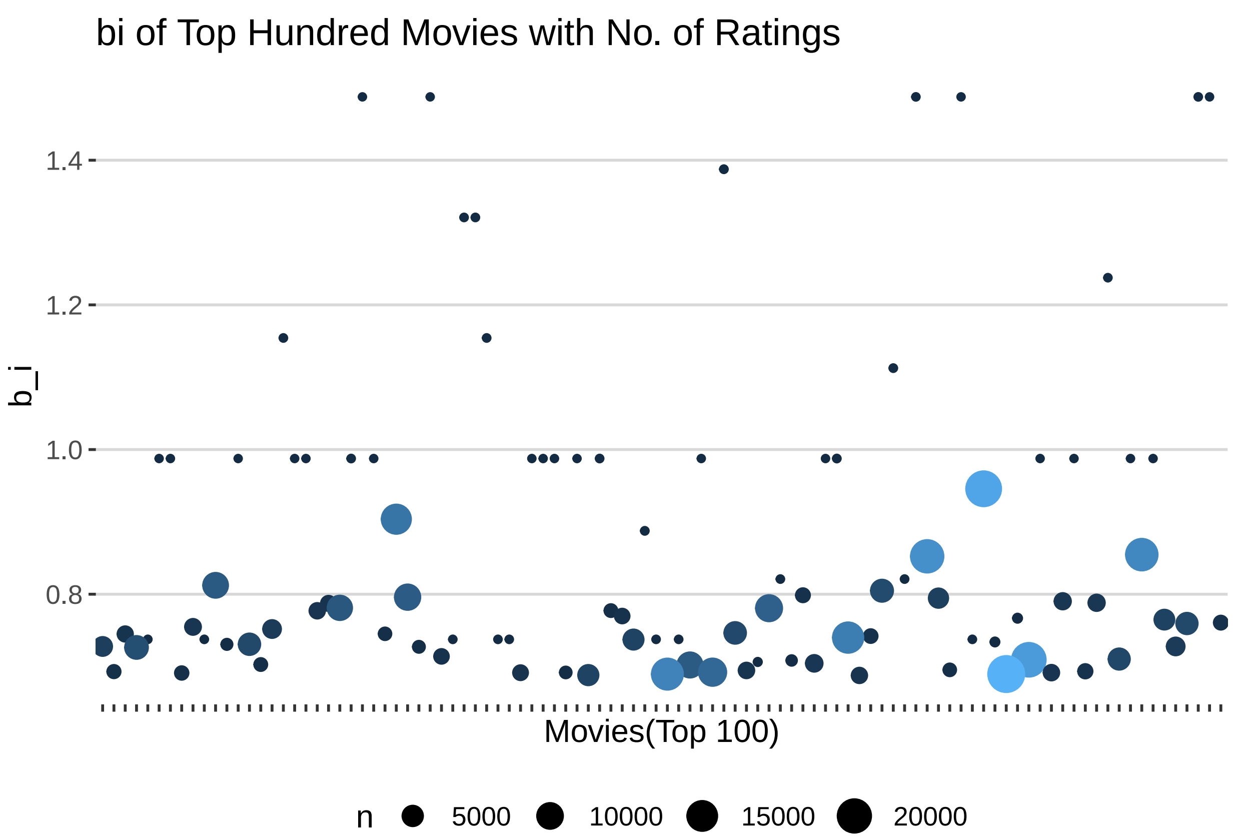
```
## [1] 0.8664991
```

As we imagined , time effect only improved the RMSE of the previous model very slightly (by 0.0001)

Regularization

```
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_effect) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:100) %>%
  ggplot(aes(title , b_i)) + geom_point(aes(size = n , color = n)) +
  theme(axis.text.x=element_blank()) + guides(color = F) +theme_hc() +
  xlab('Movies(Top 100)') + ggtitle("bi of Top Hundred Movies with No. of Ratings")
```

```
## Joining, by = "movieId"
```



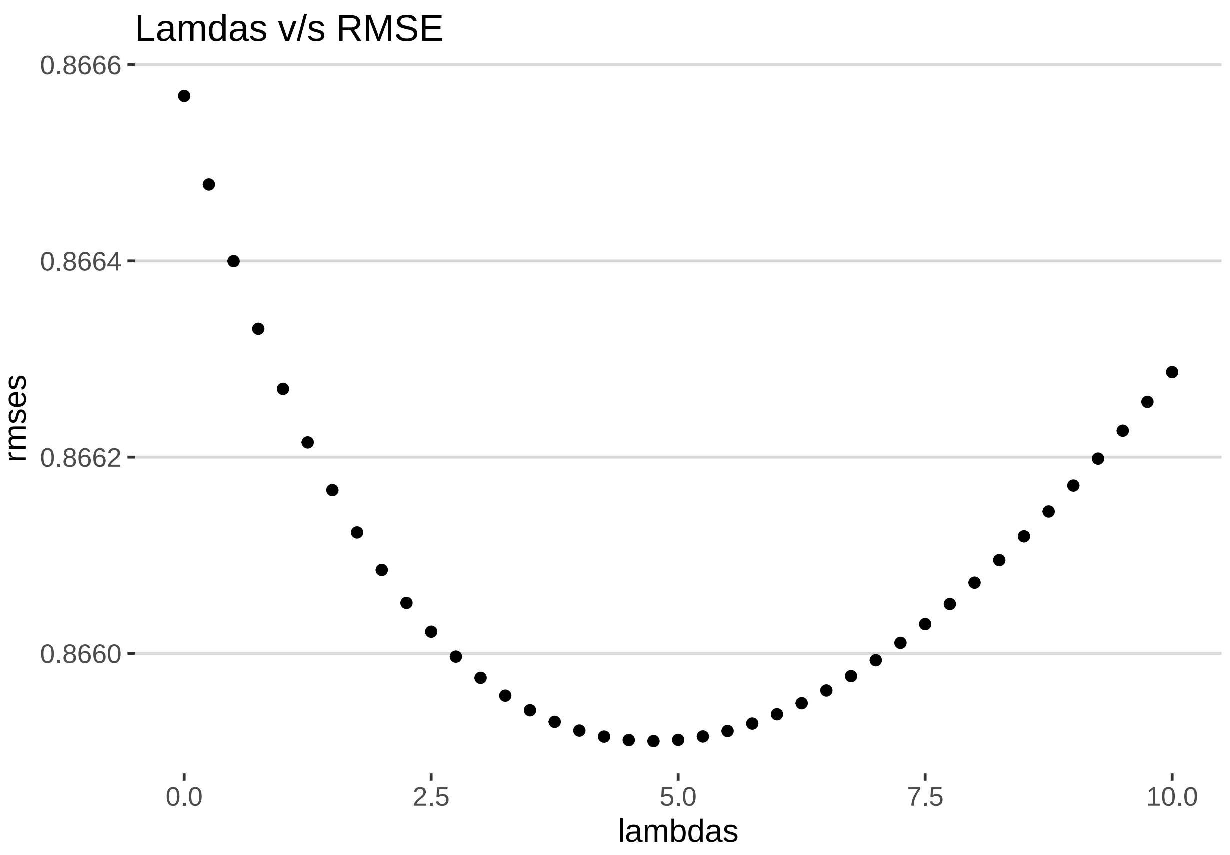
The need of penalized effect is obvious as the movies rated highest are the ones with only few reviews which explaines its high variance.

We will use regularization on linear model with movie and user effects . For this model we will tune lambdas from a seq(0 ,25, 0.25) .

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

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  pred <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(test_set$rating , pred))
})
```

```
qplot(lambdas , rmses ) + theme_hc() + ggtitle("Lamdas v/s RMSE")
```



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

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

We have obtained lambda that minimizes the RMSE . We will use this lambda for our final model .

```r
rmse_results <- data_frame(methods=c("Just the average" ,"movie effect","movie + user effects",
                                     "movie + user + time effects"),
                          RMSE = c(naive_rmse , model1_rmse, model2_rmse,model3_rmse))
```

```
## Warning: ‘data_frame()‘ was deprecated in tibble 1.1.0.
## Please use ‘tibble()‘ instead.
```

```r
rmse_results <- bind_rows(rmse_results,
                          data_frame(methods="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

methods	RMSE
Just the average	1.0602892
movie effect	0.9440672
movie + user effects	0.8665681
movie + user + time effects	0.8664991
Regularized Movie + User Effect Model	0.8659106

By comparing RMSE of different model we find out that a Regularized linear model with time and user effect minimizes RMSE and thus will be ideal for edx vs validation analysis.

Matrix Factorization

We will use recosystem package to perform matrix factorization on our train set .

In the recosystem package :

- We create an object by calling reco() function
- We use $tune() functions to tune different parameters. Here we will only tune dim(No. of latent factors) and lrate(learning rate/step size in gradient descent) parameters. We will set other parameters (costp__11 , costq__11 , nthread , niter) as static to reduce runtime.

```
#We will turn our datasets into large datasource to avoid working with matrix of very large size
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))
test_data <- with(test_set, data_memory(user_index = userId,
                                         item_index = movieId,
                                         rating     = rating))
# creating model object r
r <- Reco()

# optimizing parameters
options <- r$tune(train_data, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                         costp_l1 = 0, costq_l1 = 0,
                                         nthread = 1, niter = 10))
# training model on train data
r$train(train_data, opts = c(options$min, nthread = 1, niter = 20))

#making predictions (output can be acesses using out_memory() for $predict in recosystem)
y_hat_recomm <- r$predict(test_data, out_memory())

model4_rmse <- RMSE(test_set$rating , y_hat_recomm)
```

```
model4_rmse
```

```
## [1] 0.7911176
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(methods="Matrix factorization Model",
                                     RMSE = model4_rmse))
rmse_results %>% knitr::kable()
```

methods	RMSE
Just the average	1.0602892
movie effect	0.9440672
movie + user effects	0.8665681
movie + user + time effects	0.8664991
Regularized Movie + User Effect Model	0.8659106
Matrix factorization Model	0.7911176

Matrix factorization reduces the RMSE drastically and thus is ideal for final evaluation.

Final Model and Evaluation

We will use the two best models i.e, Regularised linear model and matrix factorization model on edx and validation set.

Regularized Linear Model

For final Regularized model we will also include time effect which only slightly improves the RMSE

```r
validation <- validation %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))

#overall average of edx set
mu_edx <- mean(edx$rating)
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_edx = sum(rating - mu_edx)/(n()+lambda))
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_edx = sum(rating - b_i_edx - mu_edx)/(n()+lambda))
b_t_edx <- edx %>%
  left_join(b_u_edx , by = 'userId') %>%
  left_join(b_i_edx , by = 'movieId') %>%
  mutate(date = round_date(as_datetime(timestamp)  , unit = 'week' )) %>%
  group_by(date) %>%
  summarise(b_t_edx = mean(rating - mu_edx - b_i_edx - b_u_edx))
pred_edx <-
  validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  left_join(b_t_edx , by = 'date')%>%
  mutate(pred_edx = mu_edx + b_i_edx + b_u_edx + b_t_edx) %>%
  .$pred_edx

rmse1_final <- RMSE(validation$rating , pred_edx)
```

```r
rmse1_final
```

```
## [1] 0.8647015
```

Matrix Factorization

```r
edx_data <-  with(edx, data_memory(user_index = userId,
                                    item_index = movieId,
                                    rating     = rating))
validation_data <- with(validation, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating     = rating))

r$train(edx_data, opts = c(options$min, nthread = 1, niter = 20))

recomm_edx <- r$predict(validation_data , out_memory())

rmse2_final <- RMSE(validation$rating , recomm_edx)
```

```
rmse2_final
```

```
## [1] 0.7827928
```

```
rmse_final <- data_frame(methods=c("Regularized movie + user + time effect (edx v/s validation)" ,"Matr
                                         RMSE = c(rmse1_final , rmse2_final))
rmse_final %>% knitr::kable()
```

methods	RMSE
Regularized movie + user + time effect (edx v/s validation)	0.8647015
Matrix factorization (edx v/s validation)	0.7827928

Conclusion

We can see that both the regularized linear model and the matrix factorization model give RMSE that are lower than a target of 0.86490 . The linear model can be significantly improved by accounting for genre effect , which is not done in this report . Using more predictors like labels , actors , etc can also significantly improve model results

The recosystem model also improved RMSE significantly by implementing Matrix Factorization using stochastic gradient descent . There are other methods of implementing matrix factorization using recommendorlab (content-based and collaborative filtering) which are not discussed here .

Apart from this there are other libraries in R which can be used for Recommendation systems.

We can also use Ensemble models in future to improve our RMSE results . These models can be quite difficult to run on commodity laptop or PC , so they were not included in the project.