MovieLens_Analysis_SA

SatyaAvala

8/2/2020

Introduction

This report describes a "Recommendation Systems" approach for Collaborative Filtering scenarios to develop a predictive model. Specific requirement is to develop and test a model which can predict ratings of movies using a given "MovieLens" dataset containing a list of users, their ratings of movies, movie classification and the time of rating. Goal for the model is to meet a specific criterion in terms of target Root Mean Square Error (RSME) of predicted results compared to the true ratings in the validation data set. Source of MovieLens data including the validation data set is "GroupLens Research Lab" at http://files.grouplens.org/datasets/movielens/ml-10m.zip

Key steps performed:

a. Prepare a train_set "edx" and tes_set for final testing, "validation" data following the instructions with 90/10 split
b. Create an internal train_set and test_set from edx data with 80/20 split to be used for developing and testing the target model.
c. Add columns required for all the train_set and test_set data.
d. Explore data to understand different variables. Use the findings to select the model
e. Identify and define a suitable model with parameters to be calculated from the test data.
f. Calculate the parameters which satisfy the target model criteria . Leverage internal data sets created in step "b" for this purpose
g. Perform final test of the model using the data sets created in step "a".
h. Summarize the findings and limitations of the model with calculated parameters.

Method/Analysis:

A model is representation of system we want to understand and predict as in our present context. For this purpose, any selected model needs to focus on the properties that matter. In general, a model is considered complex if it considers more parameters to resolve requiring more data/resources to process or the structure itself is complex. In some scenarios a simple model is more preferred for the expected accuracy of result compared to a more complex model. Preparing data, analyzing various data elements and their relation, constructing a suitable model to represent the insights gained are various phases of our method.

Data cleansing:

Data preparation for our project is relatively simple. Given data set "MovieLens" is partitioned in to two separate sets "edx" for training the model and "validation" to test the final result of the model. As we need to develop and test the model to meet the specifications in term of RSME computed against "validation" data set , the "edx" data set is further portioned in to a "train_set" and "test_set".

```r
knitr::opts_chunk$set(warning = FALSE)
# Load edx and partition Temp
# Create edx set, validation set (final hold-out test set)
# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse

## -- Attaching packages --------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

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

```
## Loading required package: recommenderlab

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loading required package: arules

##
## Attaching package: 'arules'
## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: proxy

##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist

## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

##
## Attaching package: 'recommenderlab'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

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

```
## Loading required package: lubridate

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following object is masked from 'package:base':
##
##     date
```

```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")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

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

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

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


## Partition edx in to test & train
# Test set is 20% of edx

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

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index1 <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                   list = FALSE)
train_set <- edx[-test_index1,]
temp1 <- edx[test_index1,]
```

```
# Make sure userId and movieId in test set are also in train_set

test_set <- temp1 %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Add rows removed from test_set set back into train_set

removed1 <- anti_join(temp1, test_set)
```

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

```
train_set <- rbind(train_set, removed1)

rm(test_index1, temp1,removed1)
```

Additional columns derived from the field “timestamp” were added to the data sets. For convenience two vesrsions of data sets with a column “year” of timestamp and “week” of timestamp were created.

```
knitr::opts_chunk$set(warning = FALSE)

# Create a copy of train_set and test_set  with an additional column of "year" of rating derived from "

train_set_s <- train_set %>% mutate(year = year(as_datetime(timestamp, origin = lubridate:: origin)))
 test_set_s <- test_set %>% mutate(year = year(as_datetime(timestamp, origin = lubridate:: origin)))

#  Create a copy of train_set and test_set with an additional column of "week" of rating derived from "

train_set_m <- train_set  %>% mutate (date = round_date(as_datetime(timestamp), unit = "week"))
test_set_m <- test_set %>% mutate (date = round_date(as_datetime(timestamp), unit = "week"))

#  Create a copy of "edx" and "validation"  with an additional column of "week" of rating derived from

edx_m <- edx  %>% mutate (date = round_date(as_datetime(timestamp), unit = "week"))
validation_m <- validation %>% mutate (date = round_date(as_datetime(timestamp), unit = "week"))

knitr::opts_chunk$set(echo = TRUE)
```

To understand the details of data elements, structure was analysed.

```
str(edx)
```

```
## '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 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
```

We have 6 variables of which userID,movieID,timestamp and genres all can have impact on how the rating is given. For developing a predictive model these variables are important.

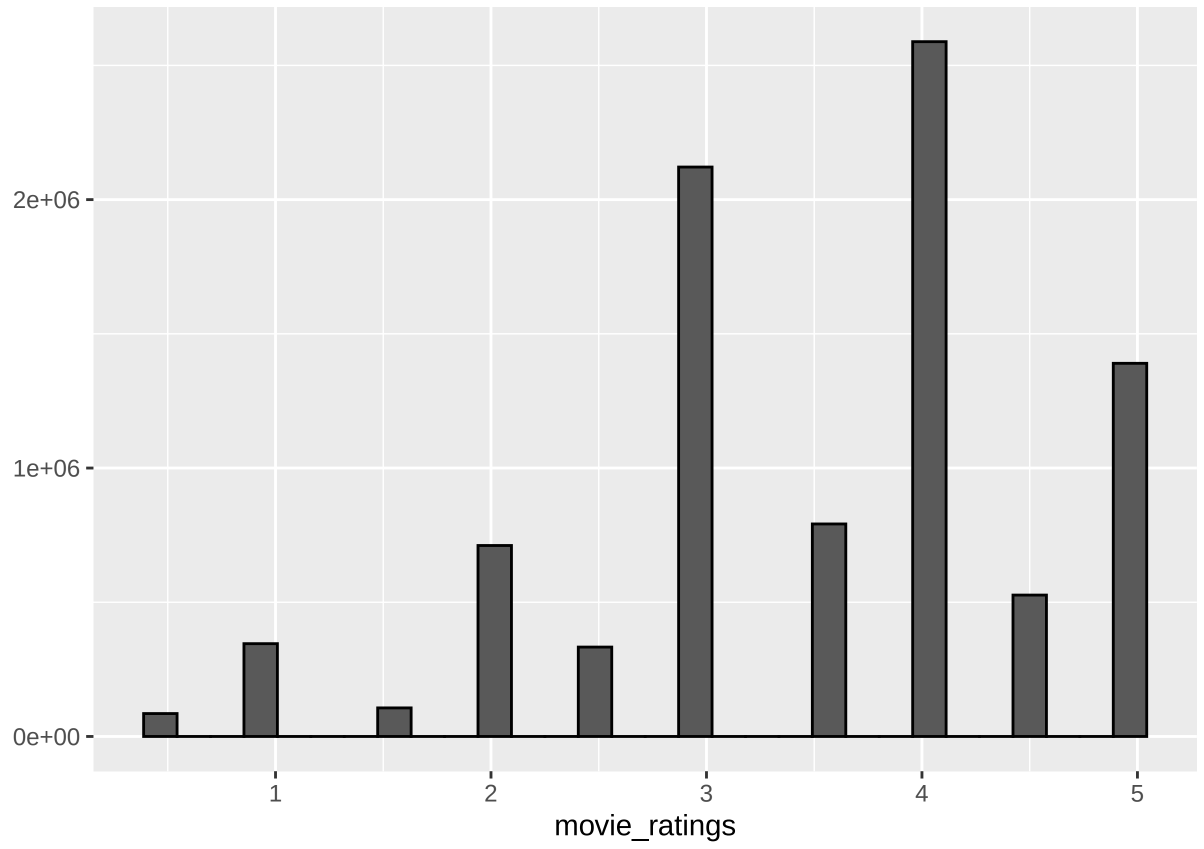
Requirement for the target model is to get a “Root Mean Square Error” (RSME) of less than 0.8649. We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y_{u,i}}$. The RMSE is then defined as: $RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$ with N being the number of user/movie combinations and the sum occurring over all these combinations A function was created to calculate RSME.

```r
# RSME Function

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

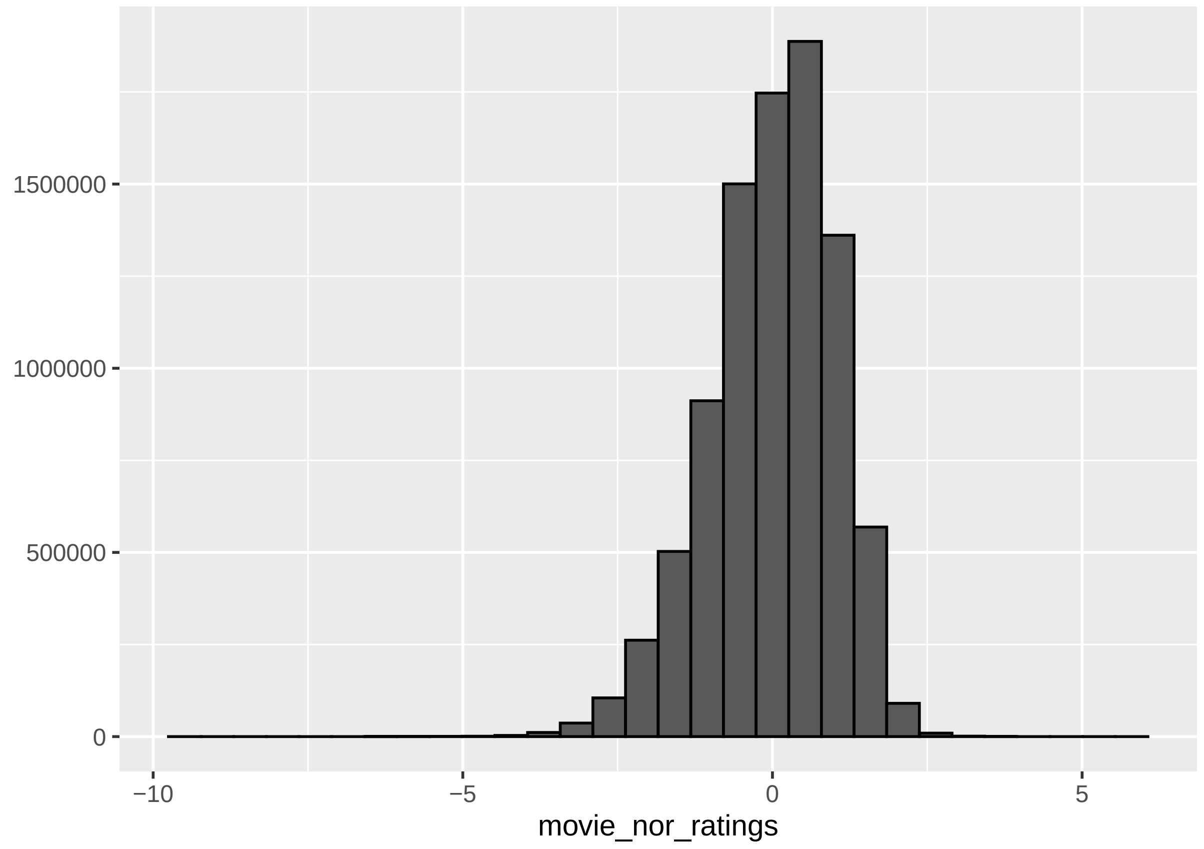
Data exploration:

edx data set has 9000055 obbservations (rows) and 6 variables (columns). An additional column for "week" of the rating dervived from rating field is added part of analysis. There are more ratings between 3 & 4. Number of whole ratings like 5,4,3 are mre compared to fractions. Histograms for ratings and normalized ratings by number of movies show the distribution of the ratings.

```r
# Plot ratings
library(recommenderlab)
userid_factor<- as.factor(edx_m$userId)
movieid_factor<- as.factor(edx_m$movieId)
movies_sm<- sparseMatrix(i = as.numeric(userid_factor), j =
                            as.numeric(movieid_factor), x = as.numeric(edx_m$rating))
movies_rrm<- new("realRatingMatrix", data = movies_sm)
colnames(movies_rrm) <- levels(movieid_factor)
rownames(movies_rrm) <- levels(userid_factor)
movie_ratings <- getRatings(movies_rrm)
movie_nor_ratings <- getRatings(normalize(movies_rrm, method ="Z-score"))
qplot(movie_ratings, bins = 30, color = I("black"))
```



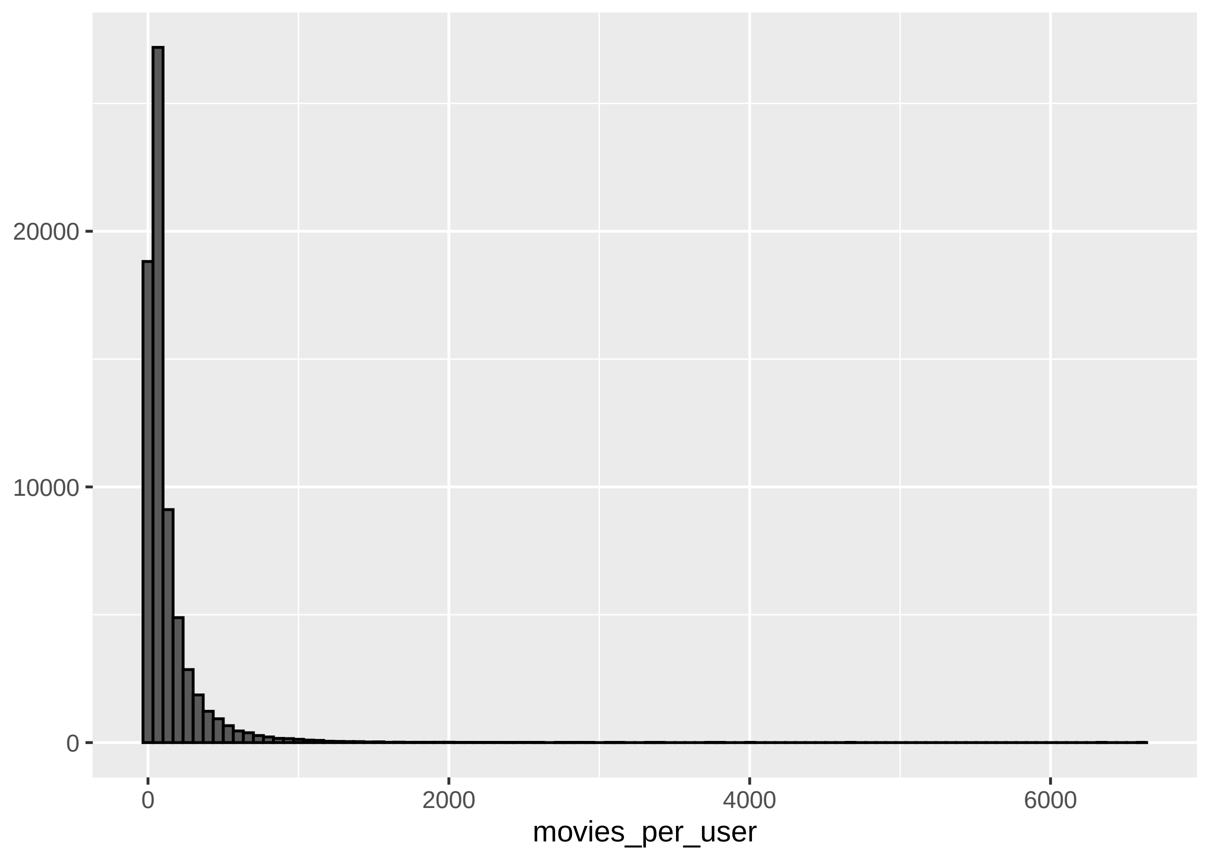
```r
qplot(movie_nor_ratings, bins = 30, color = I("black"))
```



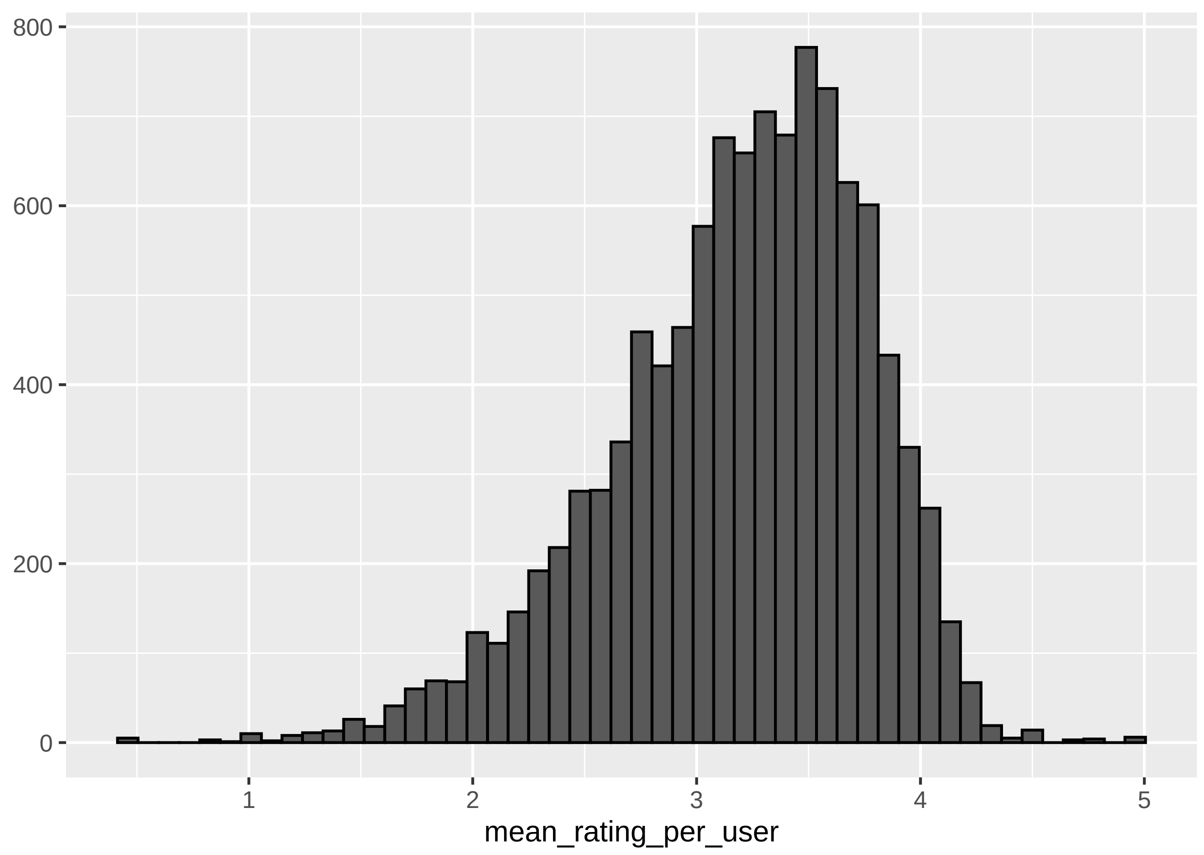
```
echo=FALSE
```

There are some core users who have rated more movies and there are many movies rated by small number of users. Average rating per movie for more number of movies is higher. Both these are plotted.

```r
# Plot rated per user
movies_per_user <- rowCounts(movies_rrm)
qplot(movies_per_user, bins = 100, color = I("black"))
```



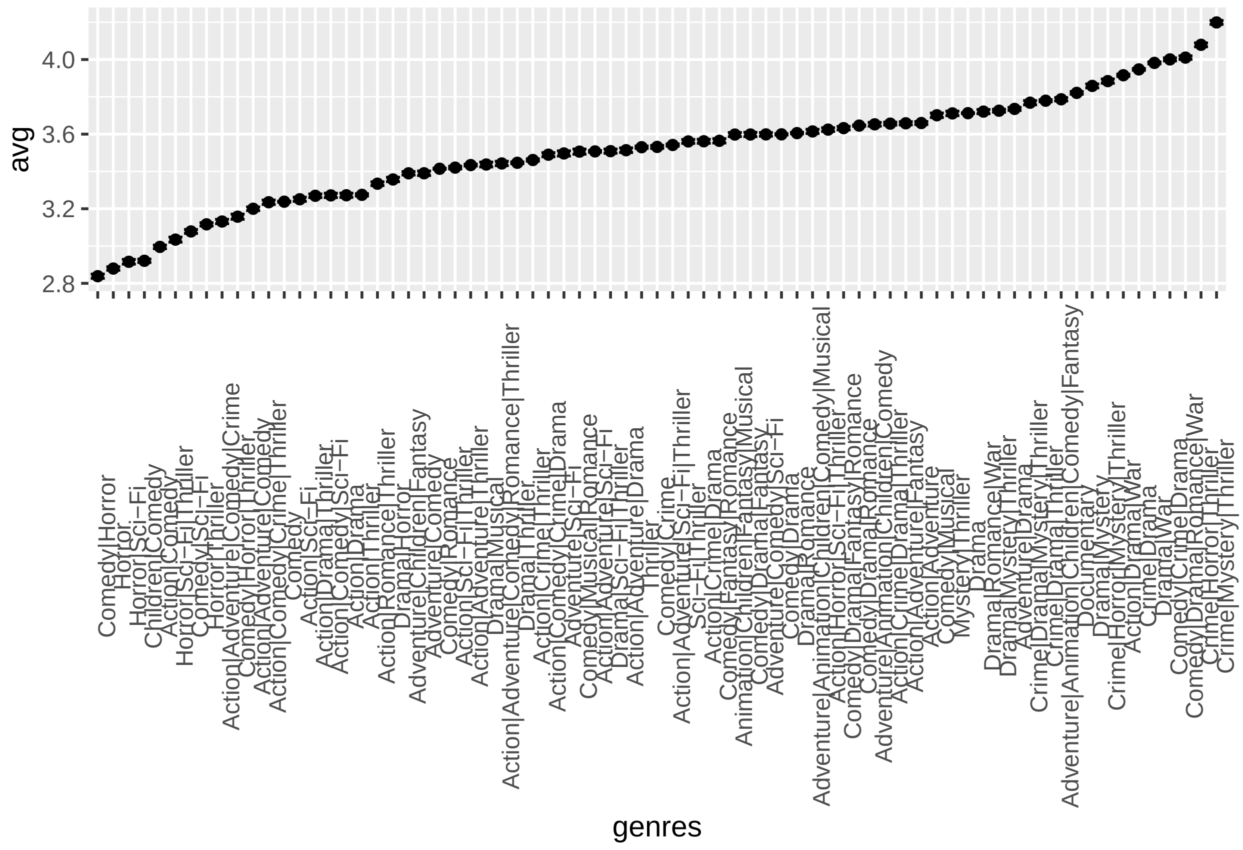
```r
mean_rating_per_user <- colMeans(movies_rrm)
qplot(mean_rating_per_user, bins = 50, color = I("black"))
```



```r
avg_movies_rated_per_user <- mean(rowCounts(movies_rrm))
avg_moview_rating <- mean(colMeans(movies_rrm))
```

There is a relation how users have rated the movies and the genres of movie. This could contribute to the user bias in opting to rate or rate at a specific level.

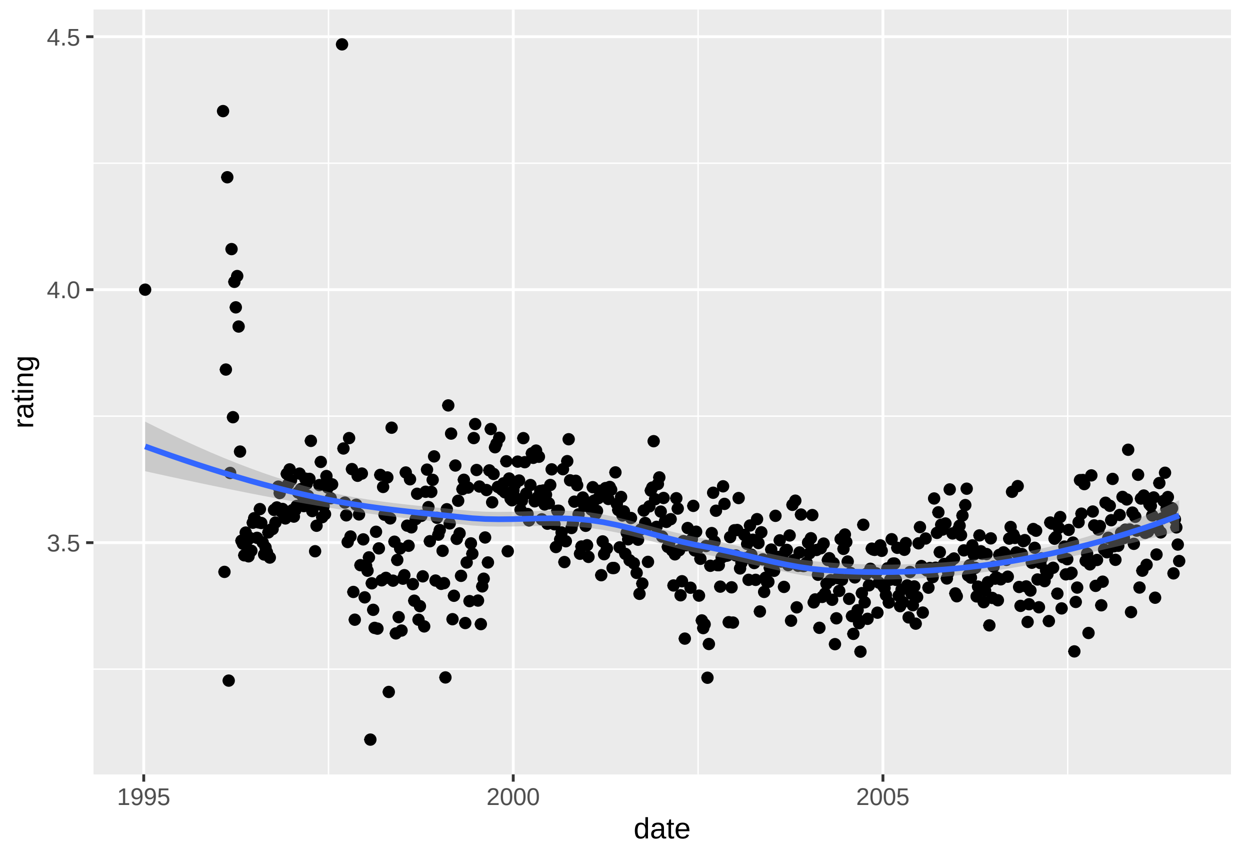
```r
# Plot genre
edx_m %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 25000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = .5))
```



There were two variants of data created with addiional column derived from timestamp data. One aggregared based on “year” other day of the “week”. There is some impact of when the rating is given by the user. This could impact by movie and user depending on how popular or not the movie had been at the time of rating and also user bias. Attempts were made consider this impact in the model.

```r
## Plot b_day
edx_m %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```

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



Insights gained:

There are user specific preferences based on genres of movies which are reflected in the ratings. Also timing of rating impacts it. Both these need to be included in the model in addition to user impacts and movie impact.

Modeling approach:

Considering different options and available resources to process the model, a penalized least squares approach was used for the model. Rating of the movie "i" per user "u" is considered to be a combination of average rating across the total pupulation plus user based bias, movie based bias, genre based bias and timing impact. This can be represented as:

$Y_{u,i} = \mu + b_i + b_u + b_{ge} + b_t + \epsilon_{u,i} + \epsilon_{u,ge} + \epsilon_{i,t}$

The problem is converted to minimizing the following:

$\frac{1}{N} \sum_{u,i} \left(y_{u,i} - \mu - b_i - b_u - b_{ge} - b_t\right)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \ + \sum_{ge} b_{ge}^2 + \sum_t b_t^2\right)$

Value of bias to minimize the above equation is to solve for each of the bias items using the following equation:

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

Solving the minimum value as measured in terms of RSME "Lamba" value is calculated for the model. In a iterartive manner "Lambda" values are recalculated for each bias item to refine the model as under:

$\frac{1}{N} \sum_{u,i} \left(y_{u,i} - \mu - b_i - b_u - b_{ge} - b_t\right)^2 + \lambda 1 \left(\sum_i b_i^2\right) + \lambda 2 \left(\sum_u b_u^2\right) + \lambda 3 \left(\sum_{ge} b_{ge}^2\right) + \lambda 4 \left(\sum_t b_t^2\right)$

Results:

Modeling results:

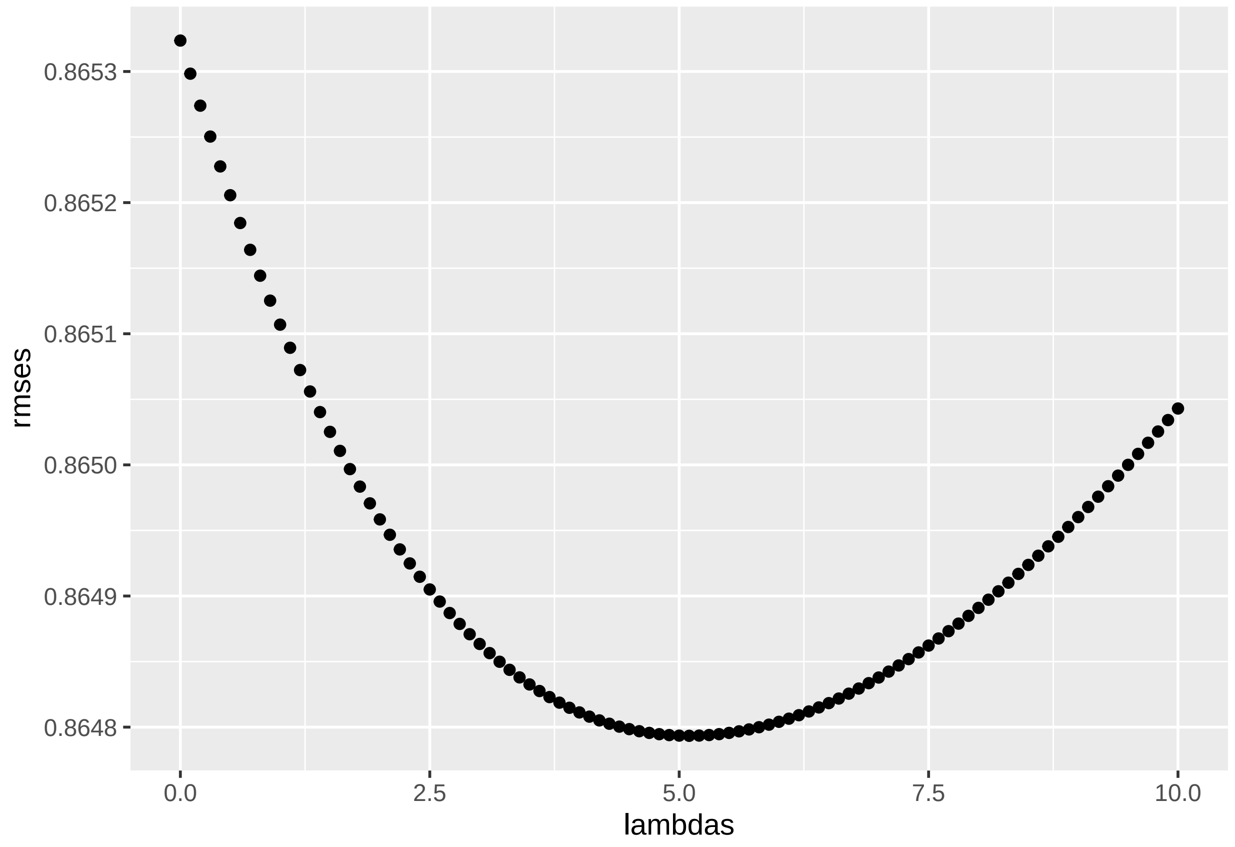
Model parameters are calculated in different combinations. Reference model was used with user related bias and movie related bias as the base model and subsequent improvements calculated adding other bias items. Final version of the model has all four "bias" items with different "Lambdas" to minimize the RSME. Results from different iterations are compiled in the table under:

Run	Bias Used	Lambda	Dataset	RSME
*1	user,movie	4.95	train_set_m	0.8652
2	user,movie,genre	4.95	train_set_m	0.8649
3	user,movie,genre,year	4.95	train_set_s	0.8649
4	user,movie,genre,week	4.95	train_set_m	0.8647
5	user,movie,genre,week	4.95	train_set_m	0.8648
6	user	5.1	train_set_m	0.8647
6	movie	4.95		
6	genre	2.5		
6	week	1.9		
7	user	5.1	edx_m	0.8643
7	movie	4.95		
7	genre	2.5		
7	week	1.9		

Using an aggregated timestamp variable at "week" level compared to "year" level has given improved result. There is not much improvement by using individual tuning parameters(lambdas) but there is change in digits which made difference when we have done final test with "validation" data set.

Following routine was used to calculate "lambdas" to minimize the RSME iteratively to get the values tabulated above

```r
# Tuning Lambda an example
set.seed(1, sample.kind="Rounding")
library(lubridate)
library(tidyverse)
lambdas <- seq(0, 10, 0.1)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set_m$rating)
  b_i <- train_set_m %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+4.95))

  b_u <- train_set_m %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_day <- train_set_m %>%
    left_join(b_i, by="movieId") %>% left_join(b_u, by="userId")%>%
    group_by(date) %>%
    summarize(b_day = sum(rating - b_i - b_u -mu)/(n()+2.5))
  b_ge <- train_set_m %>%
    left_join(b_i, by="movieId") %>% left_join(b_u, by="userId")%>%left_join(b_day, by="date")%>%
    group_by(genres) %>%
    summarize(b_ge = sum(rating - b_i - b_u -b_day-mu)/(n()+1.9))

  predicted_ratings <-
    test_set_m %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_day,by="date")%>%
    left_join(b_ge,by="genres") %>%
    mutate(pred = mu + b_i + b_u + b_day+b_ge) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set_m$rating))
})
qplot(lambdas, rmses)
```



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

```
## [1] 5.1
```

Model performance:

Final Result of the model with “validation” data set returns a RSME of 0.8643

```
# Final run with different lambdas
```

```r
set.seed(1, sample.kind="Rounding")
library(lubridate)
library(tidyverse)

rmses <- sapply(4.95, function(l){

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

  b_u <- edx_m %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+5.1))
  b_day <- edx_m %>%
    left_join(b_i, by="movieId") %>% left_join(b_u, by="userId")%>%
    group_by(date) %>%
    summarize(b_day = sum(rating - b_i - b_u -mu)/(n()+2.5))
  b_ge <- edx_m %>%
    left_join(b_i, by="movieId") %>% left_join(b_u, by="userId")%>%left_join(b_day, by="date")%>%
    group_by(genres) %>%
    summarize(b_ge = sum(rating - b_i - b_u -b_day-mu)/(n()+1.9))

  predicted_ratings <-
    validation_m %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_day,by="date")%>%
    left_join(b_ge,by="genres") %>%
    mutate(pred = mu + b_i + b_u + b_day+b_ge) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation_m$rating))
})
rmses
```

```
## [1] 0.8643108
```

Conclusion:

Penalized least squares approach was used to develop a model for predicting ratings of movies using the MovieLens data set. Addition of bias from different variables including user,movie,genre,timestamp of rating to the base model and solving for respective tuning parameters (lambdas) resulted in a model that met the target RSME. Different iterations with combination of bias from the variables and impact of timestamp related aggregated data is used for developing the model.Final RSME from the selected model is 0.8643.

Model can be improved by detailing the impact of genre and timestamp variables using Matrix factorization techniques as they have specic dependencies on other variables. Due to limitation of available compute resources some of those options were not attempted and these variables are included in the base model here.