MovieLens Project

Carlos Parra

12/17/2021

Movie Ratings Prediction

Introduction

This document includes the information (data and algorithms) required to create a movie recommendation system using the MovieLens dataset generated in section *Data*, and machine learning models described in the section *Movie Prediction Algorithm*

Libraries

Next libraries were used to generate the rating prediction:

- library(tidyverse)
- library(caret)
- library(data.table)
- library(lubridate)

Data

the edx and validation datasets are generated using the code provided in the *Capstone Introduction section* using Movielens 10M dataset; the data set contains next variables:

- userId - MovieLens users were selected at random for inclusion. Their ids have been anonymized.
- movieId - Only movies with at least one rating or tag are included in the dataset.
- rating - Each line of this file represents one rating of one movie by one user.
- timestamp - Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- title - Title of the movie.
- genres - Genres are a pipe-separated list, and are selected from the following: Action, Adventure, Animation, ,Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, and (no genres listed)

The dataset edx is then splitted into train_set and test_set to develop and test the prediction algorithm; the validation dataset will be then used to evaluate the Root Mean Squared Error (RMSE) of the final algorithm. .

```
# Note: this process could take a couple of minutes

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

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)

# 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)

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")


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)
```

Split edx dataset into separate training and test sets.

```
# split edc into test and tratin datasets
test_index <- createDataPartition(edx$rating, times = 1, p = 0.5, list =  FALSE)
train_set <- edx %>% slice(-test_index)
test_set <- edx %>% slice(test_index)
```

Baseline Predictors

Next variables were chosen as predictors:
1. movieId
2. userId
3. frequency rating per user
According to Koren (2) the number of ratings a user gave on a specific time explains a portion of the variability of the data during that time. The Frequency F_{ut} is the overall number of ratings that user u gave on day t. section 1.4 provides the formula to calculate the frequency rating per user.
4. time (day)
Variable generated using TimeStamp. The trend line shows that the rates decline over time

In the selection of predictors we avoided:

- Predictors that are highly correlated with other predictors
- Predictors have very few non-unique values
- Predictors that have close to zero variation.

```
# generate predictor day
train_set<- train_set %>%
  mutate(day = round_date(as_datetime(timestamp), unit = "day"))
```

The number of unique userIds (n_users), movieIds (n_movies), and days (n_days) are:

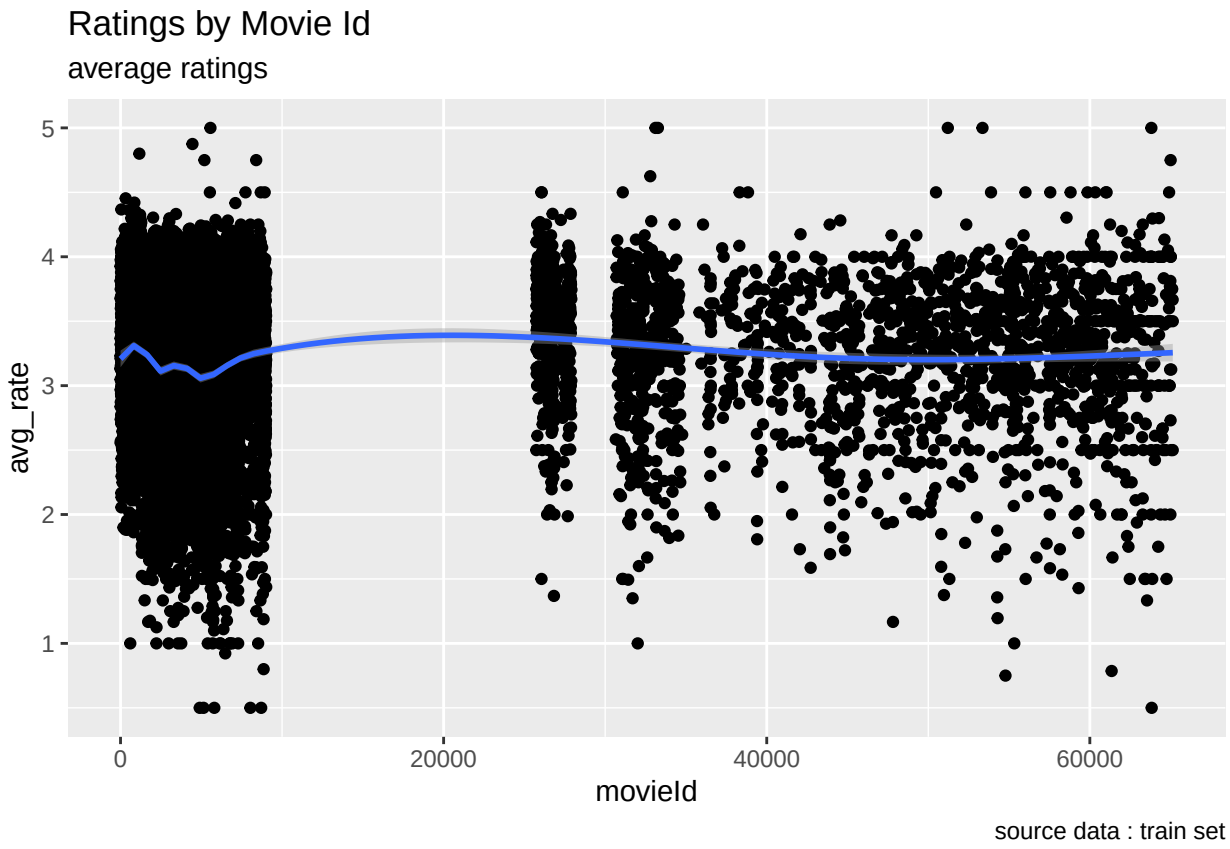
```
# Identify unique number of predictors: movie, user, time (days)

train_set %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId),
           n_days = n_distinct(day))
```

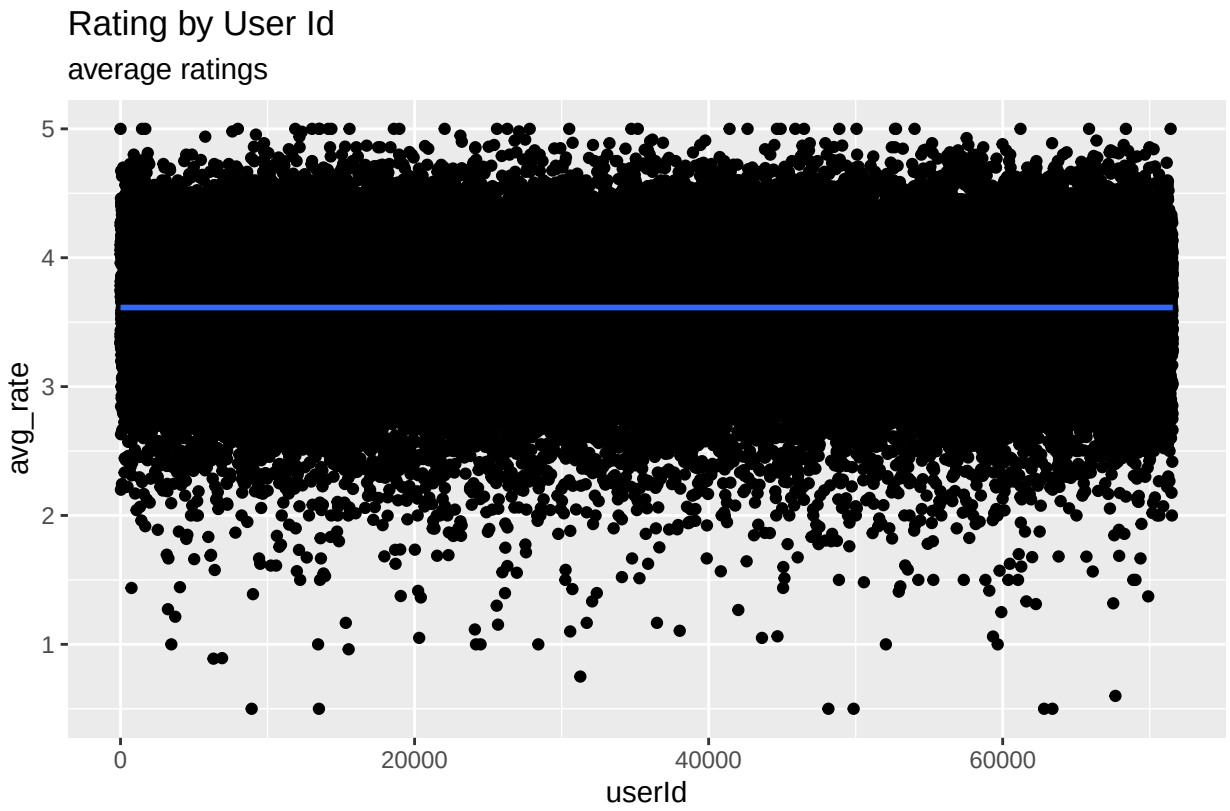
```
##   n_users n_movies n_days
## 1   69878    10527   4629
```

The interactions between users, movies and time (days) produce different rating values due to the effect associated with the movie, user or day independently. Next graphics represent the dispersion of the average rate (avg_rate) versus the predictors.

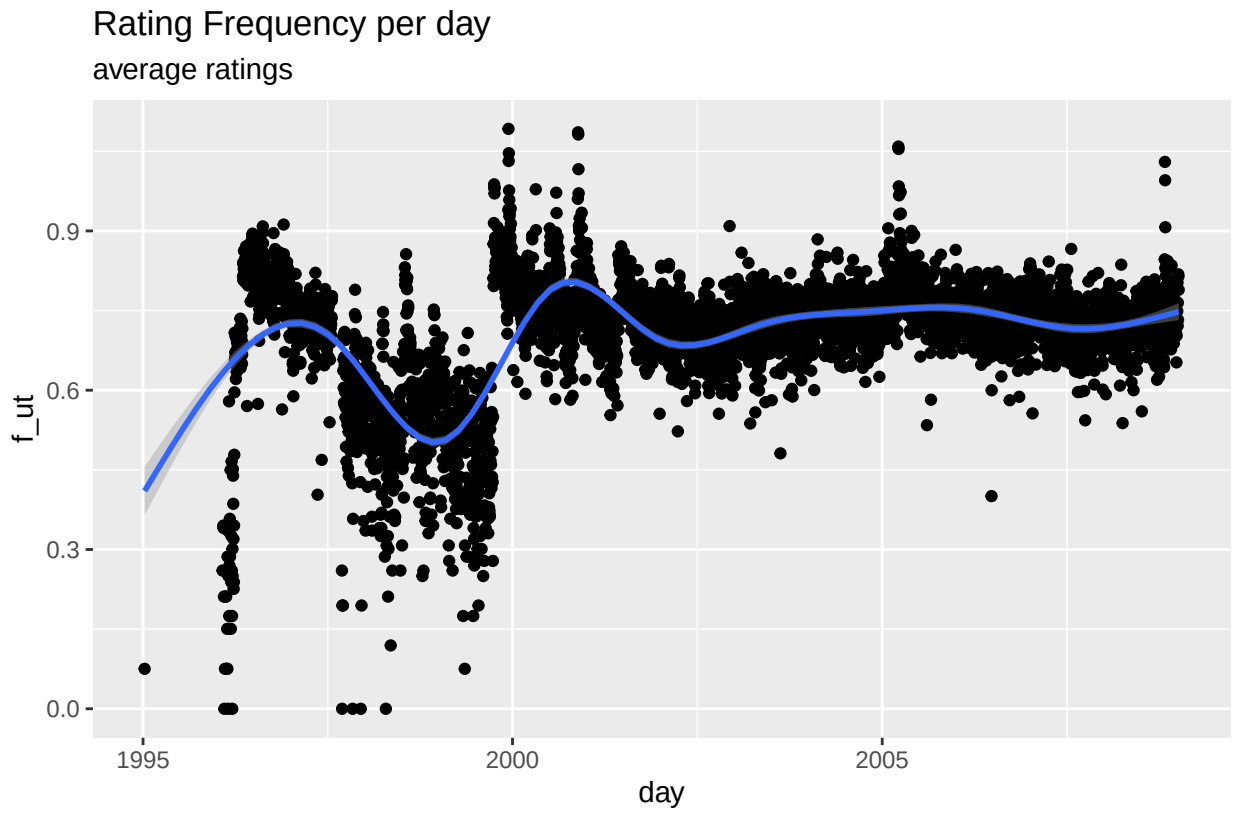
```
# graphic avg_rate versus predictors
train_set %>%
  group_by(movieId) %>%
  summarize(avg_rate=mean(rating)) %>%
  arrange(desc(avg_rate)) %>%
  ggplot(aes(movieId,avg_rate)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Ratings by Movie Id")+
  labs(subtitle = "average ratings",
       caption = "source data : train set")
```



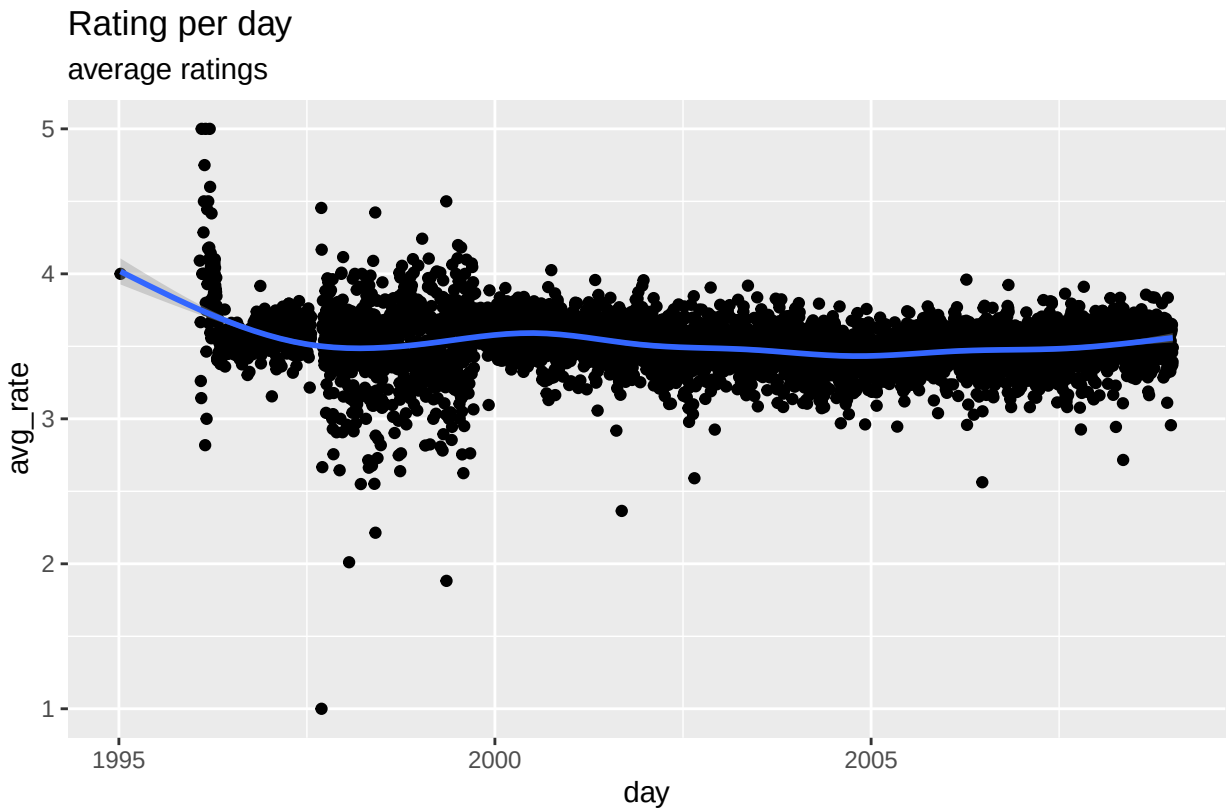

```
train_set %>%
  group_by(userId) %>%
  summarize(avg_rate=mean(rating)) %>%
  arrange(desc(avg_rate)) %>%
  ggplot(aes(userId,avg_rate)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Rating by User Id")+
  labs(subtitle = "average ratings",
       caption = "source data : train set")
```




```
#Calculate User rating fequency per day
f_ut<-train_set %>%
  group_by(userId) %>%
  group_by(day) %>%
  summarize(f_ut=abs(log(n(),base=10000)))
f_ut %>%
  ggplot(aes(day,f_ut)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Rating Frequency per day")+
  labs(subtitle = "average ratings",
       caption = "source data : train set")
```




```
train_set %>%
  group_by(day) %>%
  summarize(avg_rate=mean(rating)) %>%
  arrange(desc(avg_rate)) %>%
  ggplot(aes(day,avg_rate)) +
  geom_point()+
  geom_smooth() +
  ggtitle("Rating per day")+
  labs(subtitle = "average ratings",
  caption = "source data : train set")
```



Movie Prediction Algoritm

In order to optimize the movie prediction a combination of Machine Learning models will be used to optimize the result (minimize **root mean squared error- RMSE** between the predicted rating and the actual rating) RMSE formula is described next, where N is the number of user-movie combinations, y_{ui} is the rating for movie i by user u, and $\overline{y}_{u,i}$ is the prediction.

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

```
#function that computes the Residual Means Squared Error for a vector
# of ratings and their corresponding predictors
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

1. **Building the Recommendation Algorithm** RMSE progressive approach

1.1 **Naive Method:** assume the same rating for all movies and all users.

Estimate μ as the average rating for all movies and users

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

Epsilon represents independent errors sampled for the same distribution centered at zero; it explains the remaining variability.

Prediction using the same rating for all movies using an average rating μ for all movies and users.

```
# Estimate RMSE using Mu (average rating of all movies across all users)
mu<-mean(train_set$rating)
predicted_ratings<-mu
test_set <- test_set %>%
     semi_join(train_set, by = "movieId") %>%
     semi_join(train_set, by = "userId")
rmse1<-RMSE(test_set$rating,predicted_ratings)
cat("Baseline/average rating across all movies and users (mu):",mu)
```

```
## Baseline/average rating across all movies and users (mu): 3.512476
```

```
difference<-0
rmse_results <- tibble(method = "Baseline/average:", RMSE = rmse1, Difference = difference)
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.060407	0

1.2. Add the **Movie effect** b_i.

b_i is estimated as the average rating for movie i.

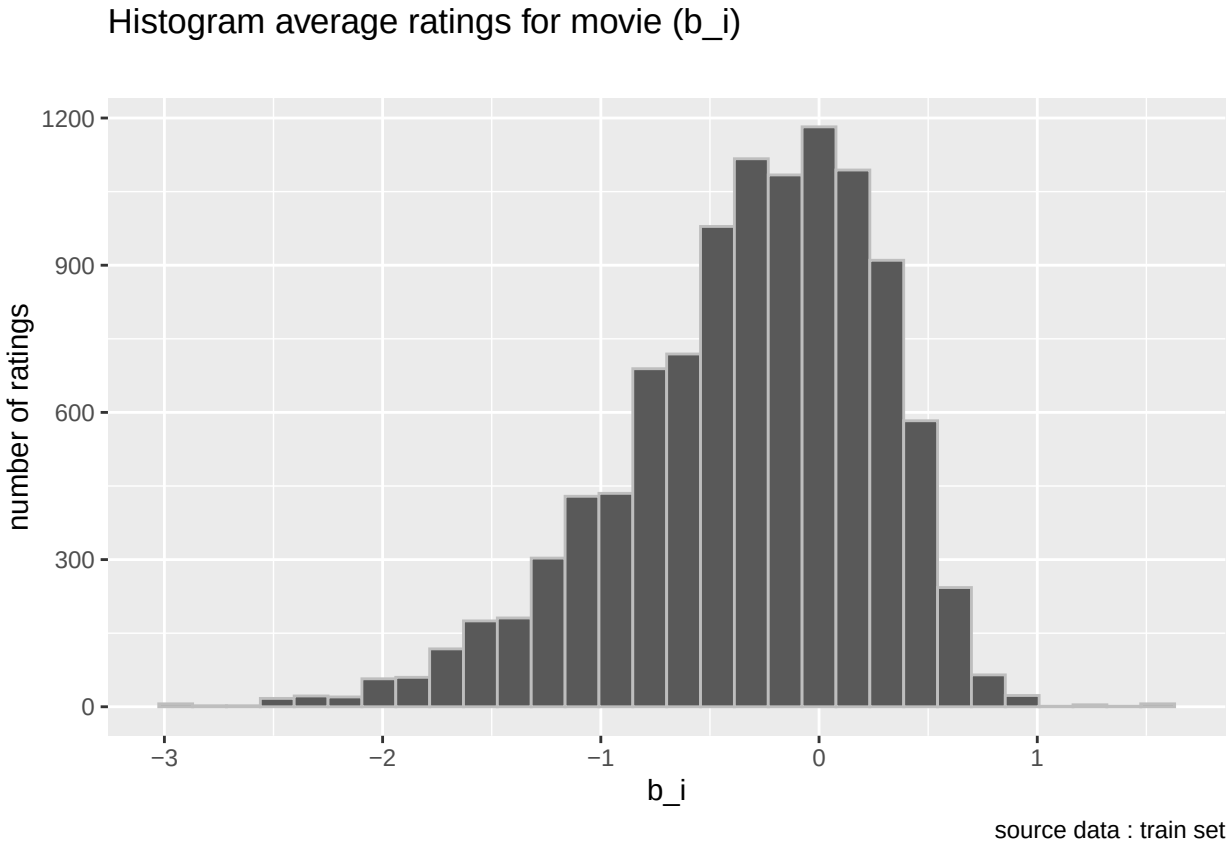
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Calculate b_i average rating per movie i.

$$b_i = Y_{u_i} - \mu$$

```
# Estimate RMSE using b_i and mu
b_i <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - mu))

b_i%>% ggplot(aes(b_i)) +
  geom_histogram( bins=30,color="gray") +
  ggtitle("Histogram average ratings for movie (b_i)") +
  labs(subtitle  ="",
       x="b_i" ,
       y="number of ratings",
       caption ="source data : train set")
```




```
predicted_ratings<-test_set %>%
  left_join(b_i, by = "movieId") %>%
  mutate(pred =  b_i + mu) %>%
    .$pred

rmse2<-RMSE(test_set$rating,predicted_ratings)
difference<-rmse2-rmse1
rmse_results <-bind_rows(rmse_results,tibble(method = "Movie Effect Model", RMSE = rmse2, Difference = d
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.0604071	0.0000000
Movie Effect Model	0.9441335	-0.1162737

1.3 Add the **user specific effect** b_u. b_u is estimated by taking the average of the residuals obtained after removing the overall mean and the movie effect from the rating y_{ui}

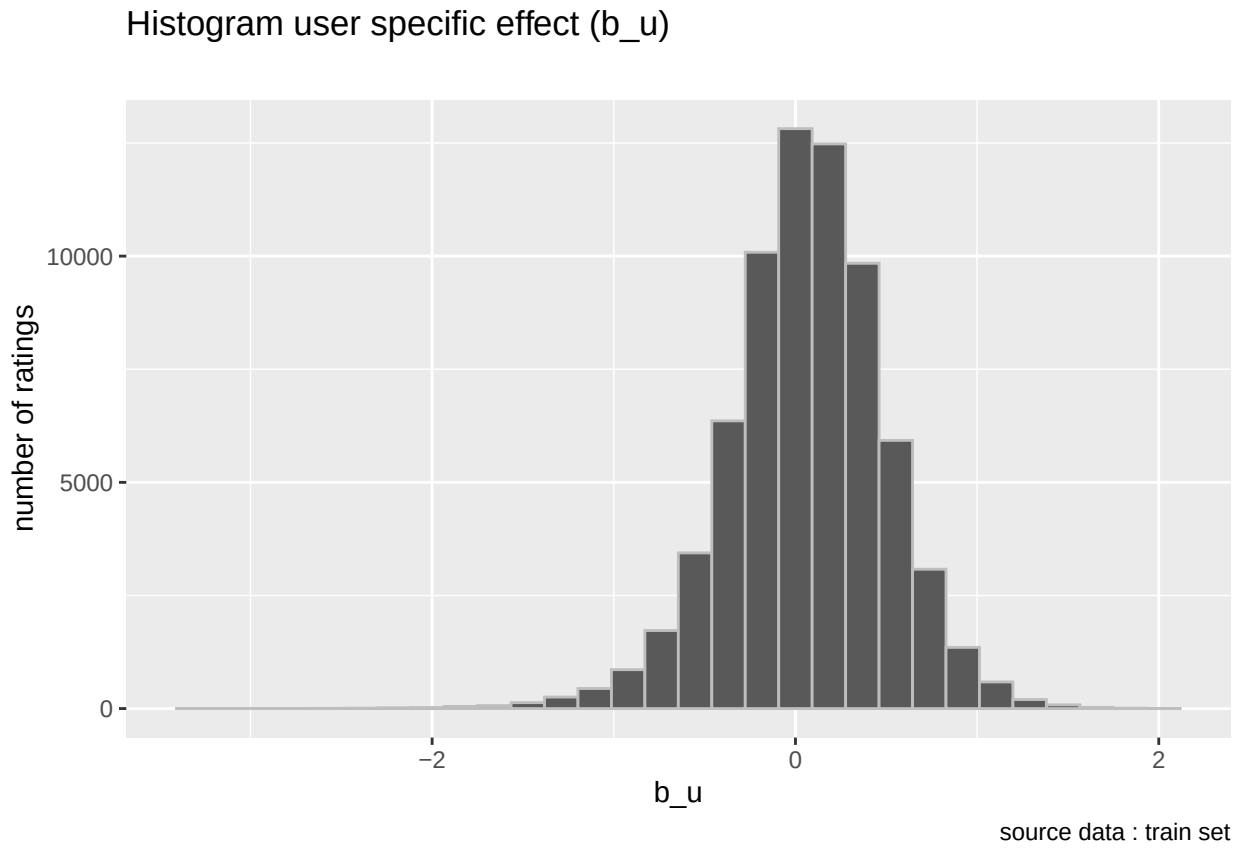
$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

The user specific effect b_u per user u

$$b_u = Y_{u_i} - \mu - b_i$$

```
# Estimate RMSE using b_i and b_u
b_u <- train_set %>%
     left_join(b_i, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_i))

b_u%>% ggplot(aes(b_u)) +
  geom_histogram( bins=30, color = "gray") +
  ggtitle("Histogram user specific effect (b_u)") +
  labs(subtitle  ="",
       x="b_u" ,
       y="number of ratings",
       caption ="source data : train set")
```




```
predicted_ratings<-test_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
rmse3<-RMSE(test_set$rating,predicted_ratings)
difference<-rmse3-rmse2
rmse_results <-bind_rows(rmse_results,tibble(method = "Movie and User Effect Model", RMSE = rmse3, Diff
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.0604071	0.0000000
Movie Effect Model	0.9441335	-0.1162737
Movie and User Effect Model	0.8696665	-0.0744670

1.4 Include the **frequency effect** f_{ut} The frequency is the number of ratings a user gave on a specific time denoted as F_{ut}. I will use a Koren's (2) rounded logarithm of F_{ut}, denoted as:

$$f_{ut} = [\log_a F_{ut}]$$

F_{ut} is the frequency: overall number of ratings that user u gave on day t.

"Interestingly, even though f_{ut} is solely driven by user u, it will influence the item-biases, rather than the user-biases." (Koren 3) I experimented by adding a factor that multiplies the item_biases with the frequency

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

$$b_i = (Y_{u_i} - \mu)/(1 + f_{ut})$$

Notes on f_{ut}
The formula was fined tuned (testing power of 10 to identify the inflexion point) to optimize RMSE:
The selected base for the logarithm is 10000 and the denominator of f_{ut} *to estimated* b_i *is -1000.*

```
# Estimate RMSE using b_i andjusted by f_ut. and b_u
b_i <- train_set %>%
  left_join(f_ut,by="day") %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)/(1 - mean(f_ut )/1000))
b_u <- train_set %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings<-test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
rmse4<-RMSE(test_set$rating,predicted_ratings)
difference<-rmse4-rmse3
rmse_results <-bind_rows(rmse_results,tibble(method = "Movie adjusted by frequency and  User Model", RMS
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.0604071	0.0000000
Movie Effect Model	0.9441335	-0.1162737
Movie and User Effect Model	0.8696665	-0.0744670
Movie adjusted by frequency and User Model	0.8696660	-0.0000005

The predictor frequency f_{ut} is discarded because the improvement was negligible; there could be other potential ways to combine frequency with movie effect that produce better results.

1.5 Add the **time effect** b_t.
The graphic in the section Predictors shows that ratings reduce over time, I have added the variable b_t to capture this effect

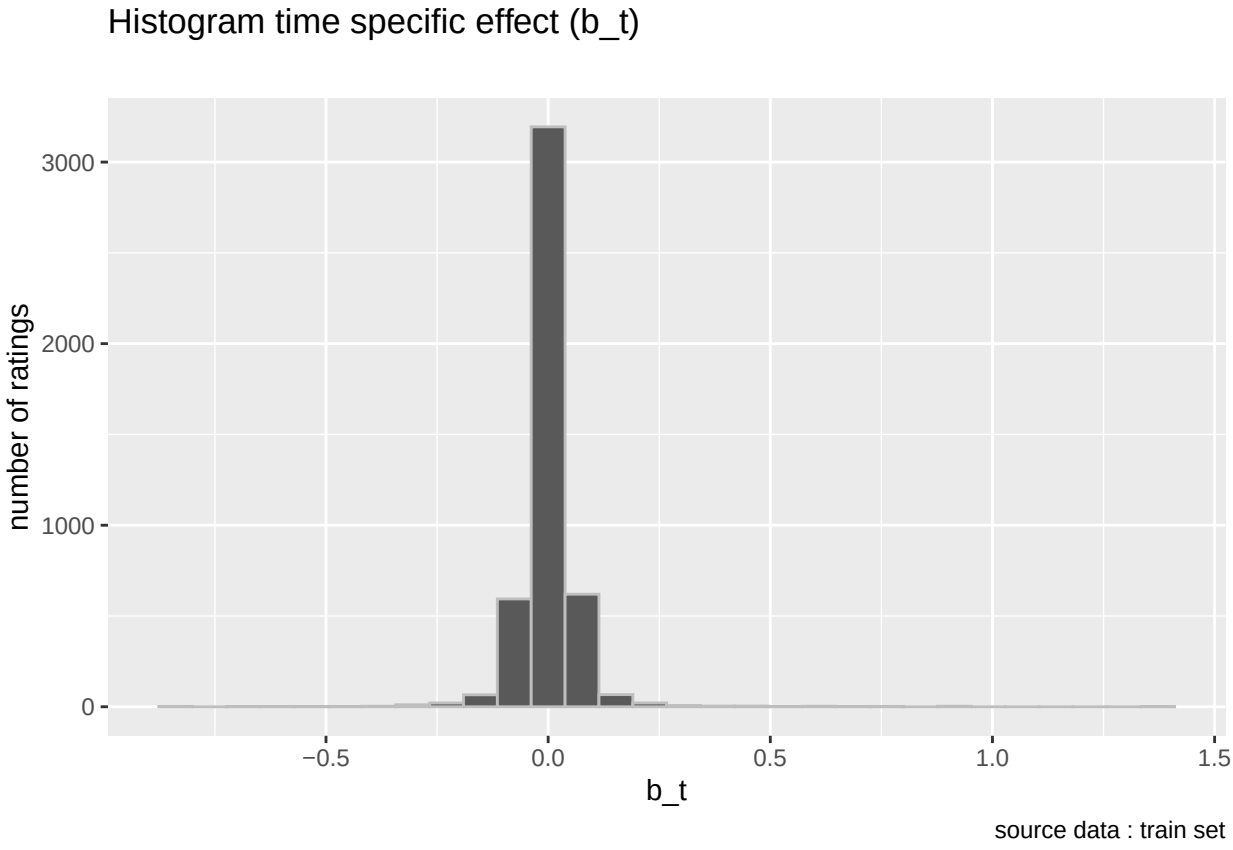
$$Y_{u,i} = \mu + b_i + b_u + b_t + \epsilon_{u,i}$$

The time specific effect b_t is

$$b_t = Y_{u_i} - \mu - b_i - b_u$$

Calculate the time effect b_t (per day)

```
# Estimate RMSE using b_t, b_u, b_i and mu
b_i <- train_set %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - mu))
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - b_i - mu))
# add day variable to tes_set to generate b_t
test_set<- test_set %>%
  mutate(day = round_date(as_datetime(timestamp), unit = "day"))
b_t <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(day) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
b_t%>% ggplot(aes(b_t)) +
  geom_histogram( bins=30, color = "gray") +
  ggtitle("Histogram time specific effect (b_t)") +
  labs(subtitle  ="",
       x="b_t" ,
       y="number of ratings",
       caption ="source data : train set")
```




```
predicted_ratings <-  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_t, by = "day") %>%
  mutate(pred = ifelse(is.na(mu + b_i + b_u + b_t),mu + b_i +b_u,mu + b_i + b_u + b_t   )) %>%
  .$pred
rmse5 <- RMSE(predicted_ratings, test_set$rating)
difference<-rmse5-rmse3
rmse_results <-bind_rows(rmse_results,tibble(method = "Movie,User, and Time Model", RMSE = rmse5, Diffe
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.0604071	0.0000000
Movie Effect Model	0.9441335	-0.1162737
Movie and User Effect Model	0.8696665	-0.0744670
Movie adjusted by frequency and User Model	0.8696660	-0.0000005
Movie,User, and Time Model	0.8692919	-0.0003746

2. **Regularization** -
   Regularization penalizes large estimates that come from small sample sizes. The algorithms described in prior section give the same weight to any estimate regardless of the size of the sample. The total variability generated by the effect of the size can be inhibited by penalizing large estimates that come from sample samples, similar to th Bayesian approach to shrunk predictions. A penalty term is added

to the equation:

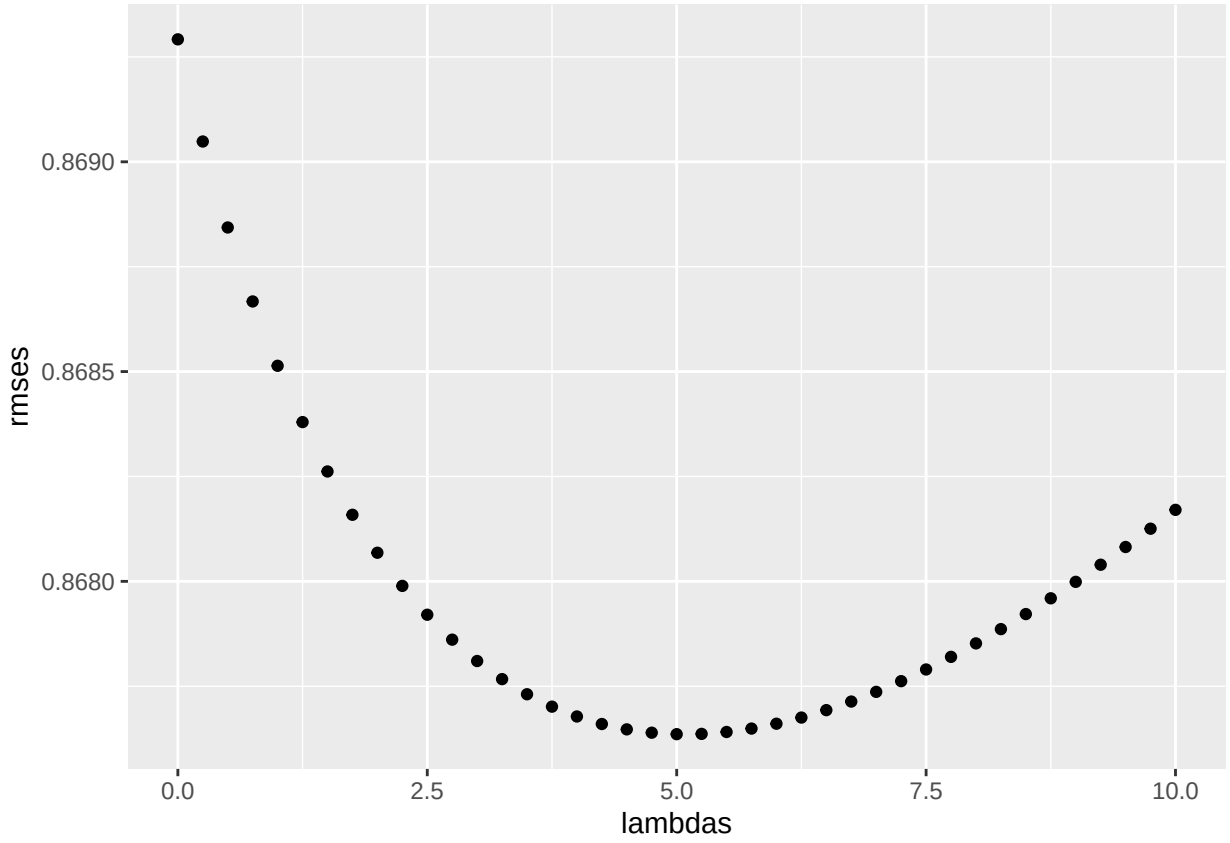
$$\frac{1}{n}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u-b_t\right)^2=\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_t b_t^2\right)$$

Next routine will identify the optimal lambda in the range 0, 10 with 0.25 increments

```
lambdas <- seq(0,10, 0.25)
rmses <- sapply(lambdas, function(l){
  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))

  b_t <- train_set %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(day) %>%
    summarize(b_t = sum(rating - mu - b_i - b_u)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "day") %>%
    mutate(pred = ifelse(is.na(mu + b_i + b_u + b_t),mu + b_i +b_u,mu + b_i + b_u + b_t)) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
cat("Optimal Lambda:", lambda)
```

```
## Optimal Lambda: 5
```

```
rmse6<- min(rmses)
difference<-rmse6-rmse5
rmse_results <-bind_rows(rmse_results,tibble(method = "Regularized Movie, User, and and Time Model", RMS
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.0604071	0.0000000
Movie Effect Model	0.9441335	-0.1162737
Movie and User Effect Model	0.8696665	-0.0744670
Movie adjusted by frequency and User Model	0.8696660	-0.0000005
Movie,User, and Time Model	0.8692919	-0.0003746
Regularized Movie, User, and and Time Model	0.8676358	-0.0016560

Validation

The final step is to validate the algorithm using the validation dataset generated in the *Data section*

```
edx <- edx %>%
    mutate(day = round_date(as_datetime(timestamp), unit = "day"))

validation <- validation %>%
  mutate(day = round_date(as_datetime(timestamp), unit = "day"))

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

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

 b_t <- edx %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(day) %>%
    summarize(b_t = sum(rating - mu - b_i - b_u)/(n()+lambda))
 predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "day") %>%
    mutate(pred = ifelse(is.na(mu + b_i + b_u + b_t),mu + b_i +b_u,mu + b_i + b_u + b_t   )) %>%
    .$pred
rmse7<-RMSE(validation$rating,predicted_ratings)
difference<-rmse7-rmse6
rmse_results <-bind_rows(rmse_results,tibble(method = "Validation - Regularized Movie, User, and and Ti
rmse_results %>% knitr::kable()
```

method	RMSE	Difference
Baseline/average:	1.0604071	0.0000000
Movie Effect Model	0.9441335	-0.1162737
Movie and User Effect Model	0.8696665	-0.0744670
Movie adjusted by frequency and User Model	0.8696660	-0.0000005
Movie,User, and Time Model	0.8692919	-0.0003746
Regularized Movie, User, and and Time Model	0.8676358	-0.0016560
Validation - Regularized Movie, User, and and Time Model	0.8642773	-0.0033585

Result

The Optimal RMSE obtained with the method described in this document is:

```
## OPTIMAL RMSE: 0.8642773
```

Conclusion

The chosen algorithm uses the predictors movie, user, and time with regularization to optimize the predicted movie rating accuracy.

The user rating frequency was explored as potential predictor without significant improvements, it is possible that adjusting the formula this predictor could provide better results.

Computer intense methods like Matrix Factorization or Singular Value Decomposition were not explored due to System requirements. There is literature available in the web to apply the matrix factorization method by using the R package *recosystem*; this method generates RMSE lower than 0.8

References

1. Irizarry, Rafael A (2021-07-03). Introduction to Data Science Data Analysis and Prediction Algorithms with R
2. Koren, Yehuda (August 2009). The Bellkor Solution to the Netflix Grand Prize
3. Chen, Edwin. Winning the Netflix Prize: A Summary
4. Wickham, Hadley and Grolemund, Garrett. R for Data Science Visualize, Model, Transform, Tidy and Import Data