Capstone - Movielens

Renz Marvin Asprec

2024-10-23

Executive Summary

This project aims to produce a movie recommendation system, based on the MoiveLens Dataset, through different data analysis methods via R.

The Movielens data set contains several ratings applied to different movies by many users of the online movie recommender service MovieLens.

This project explored 5 models in total. The first model is a naive approach which just predicts the mean of all the ratings. The second and third models investigated the effect of the movie and user effects. The fourth model used regularization to penalize the effects from small samples. The final model used matrix factorization.

Each approach was evaluated using the root mean squared (RMSE) loss function.

The model which resulted to the best RMSE is the matrix factorization model, with an RMSE of 0.795 using the test data.

The RMSE obtained using the validation data is 0.795.

Introduction

This project aims to produce a movie recommendation system through different data analysis methods via R. The project used the MovieLens data set.

MovieLens Data Set

MovieLens data set is a movie rating database generated by GroupLens research lab. It is a collection of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users.

Recommendation systems use ratings that *users* have given *items* to make specific recommendations. [1]

Evaluation Metrics

The recommendation system models for this project were evaluated using a loss function of root mean squared error (RMSE). The RMSE is defined as:

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

where $y_{u,i}$ is the actual rating for user u, and movie i, and $\hat{y}_{u,i}$ is the predicted rating.

Methodology

Exploratory Data Analysis

The movielens data was extracted from the grouplens website.[2] The data was partitioned to a 90% training data set, *edx*, and a 10% final hold-out data set, *validation*. (See Appendix)

The *edx* training data is shown below in a tidy format.

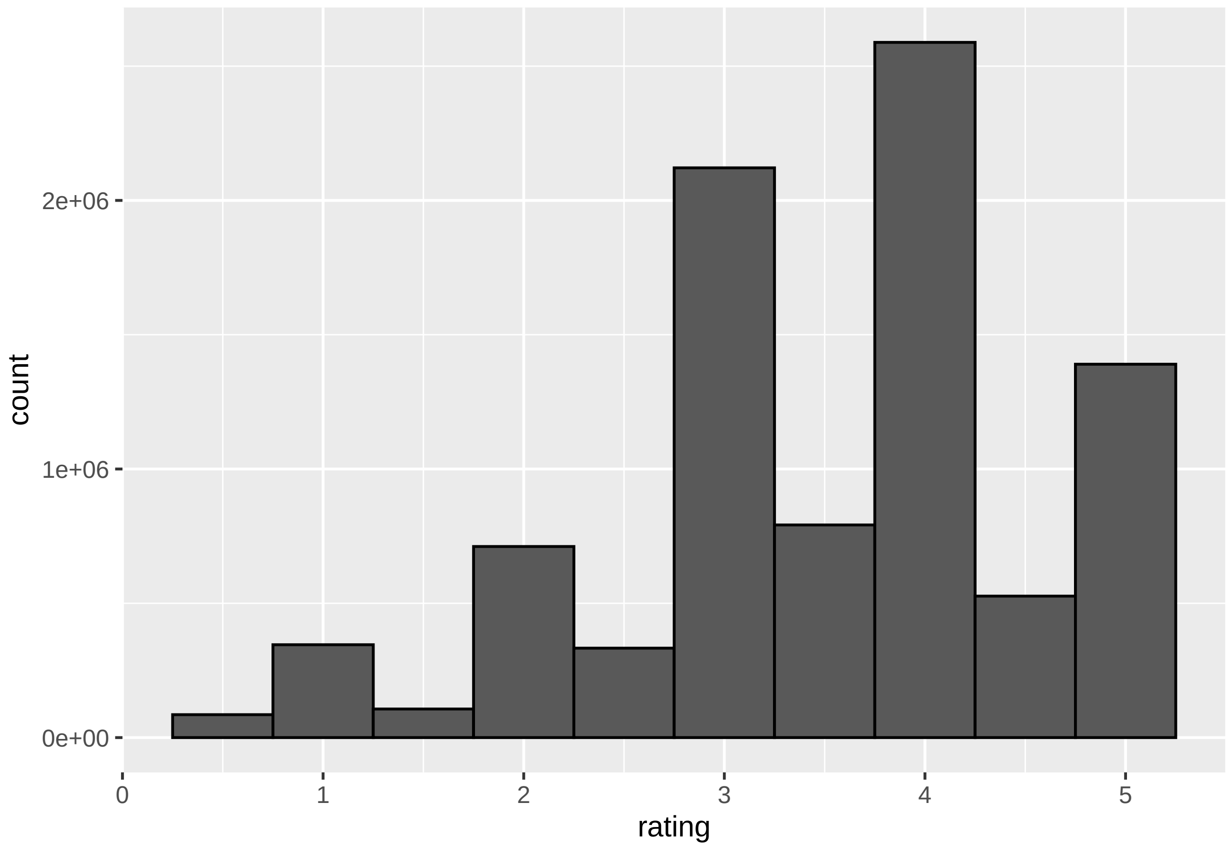
```
##    userId movieId rating timestamp                         title
##     <int>   <num>  <num>     <int>                        <char>
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      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
##                           <char>
## 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
```

From the data frame above, we can see that the data contains the following details in the columns:

- userId - unique identification for a specific user
- movieId - unique identification for a specific movie
- rating - rating provided by a user for a specific movie. The rating is a 0-5 scale
- timestamp - timestamp of the rating
- title - title of the movie corresponding to a movieId
- genres - genres associated with the movie

Exploration of the data shows that there are 9000055 rows and 6 columns. There are 69878 unique users, 10677 unique movies.

The ratings are in a scale of 0-5 stars. Below is a distribution of the ratings



As shown above, whole star ratings are more common than half star ratings.

Exploration of the data showed that the ratings are within the years 1995 and 2009.

Models

Naive Approach - Mean

A naive estimate that would reduce the value of the RMSE would be to estimate the mean of all the ratings. A model that predicts the mean would look like this:

$$\mathrm{Y}_{u,i} = \mu + \epsilon_{u,i}$$

with μ as the average, and $\epsilon_{u,i}$ as the independent errors

Movie Effect

The second model assumes that there is a bias for each movies. Some movies are just generally rated higher or lower than other movies. A model that takes into account a movie effect is shown below:

$$\mathrm{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

where b_i is the bias for each movie, i.

Movie and User Effect

The third model assumes that there is a bias for each user. Some users generally rate lower or higher compared to other users. A model that takes into account both movie and user effects is shown below:

$$\mathrm{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is the bias for each user, u.

Regularized Movie and User Effect

The fourth model penalizes large estimates of the bias effects from small sample sizes. The previous models account for the bias effects across all sample sizes. However, movie and user effects from few ratings have more uncertainties.The resulting models in this project would be evaluated using the loss function, root mean squared error or RMSE.The resulting models in this project would be evaluated using the loss function, root mean squared error or RMSE.The resulting models in this project would be evaluated using the loss function, root mean squared error or RMSE.

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

and

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

Here, λ is an optimization parameter.

Matrix Factorization

A popular technique in solving a recommendation system is to use matrix factorization. The idea in matrix factorization is to estimate the whole matrix of rating R_{mxn} by the product of two matrices with lower dimensions, P_{kxm} and Q_{kxn}, such that

$$\mathrm{R} \approx P'Q$$

Let p_u be the u-th column of P, and q_v be the v-th column of Q, then the rating given by the user u on the movie v would be predicted as $p'_u q_v$.[3]

For this model, the recommender system under the *recosystem* package by Yixuan Qiu [3] was used.

Results

Model Training

In order to train the models, the edx data set was partitioned into 80% training, and 20% testing sets. For all the models trained, the RMSE would be evaluated using the test set.

Naive Approach - Mean

For this model, the estimates are just equal to the mean of the data set. The average, mu, of the data is:

```
## [1] 3.51
```

The RMSE for this model is

```
## [1] 1.06
```

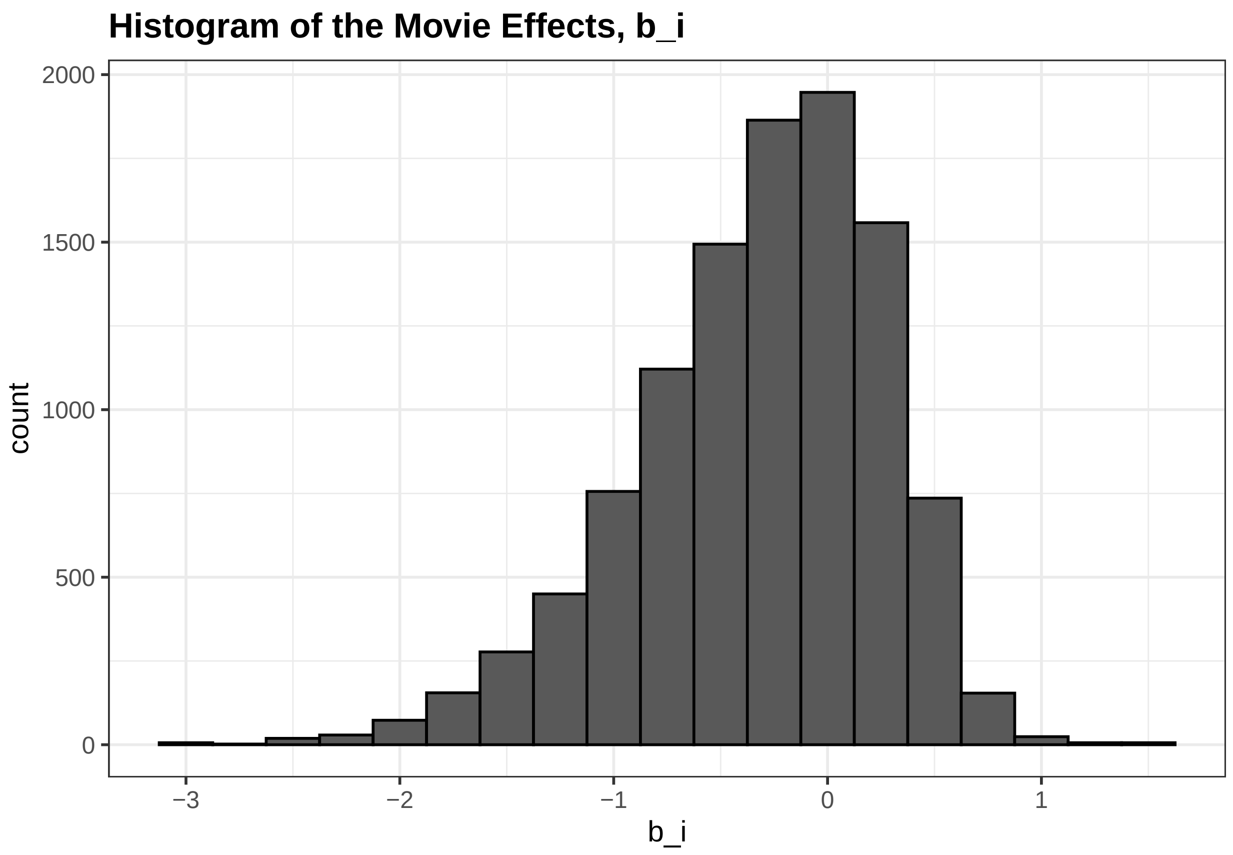
With an RMSE of 1.06, the error would be larger than a star of rating.

model	RMSE
mean	1.06

Movie Effect

b_i were computed by grouping by movies and then getting the mean of the ratings after subtracting μ.

The histogram of the b_i 's are shown below.



We can see that there is a distribution of the b_i 's.

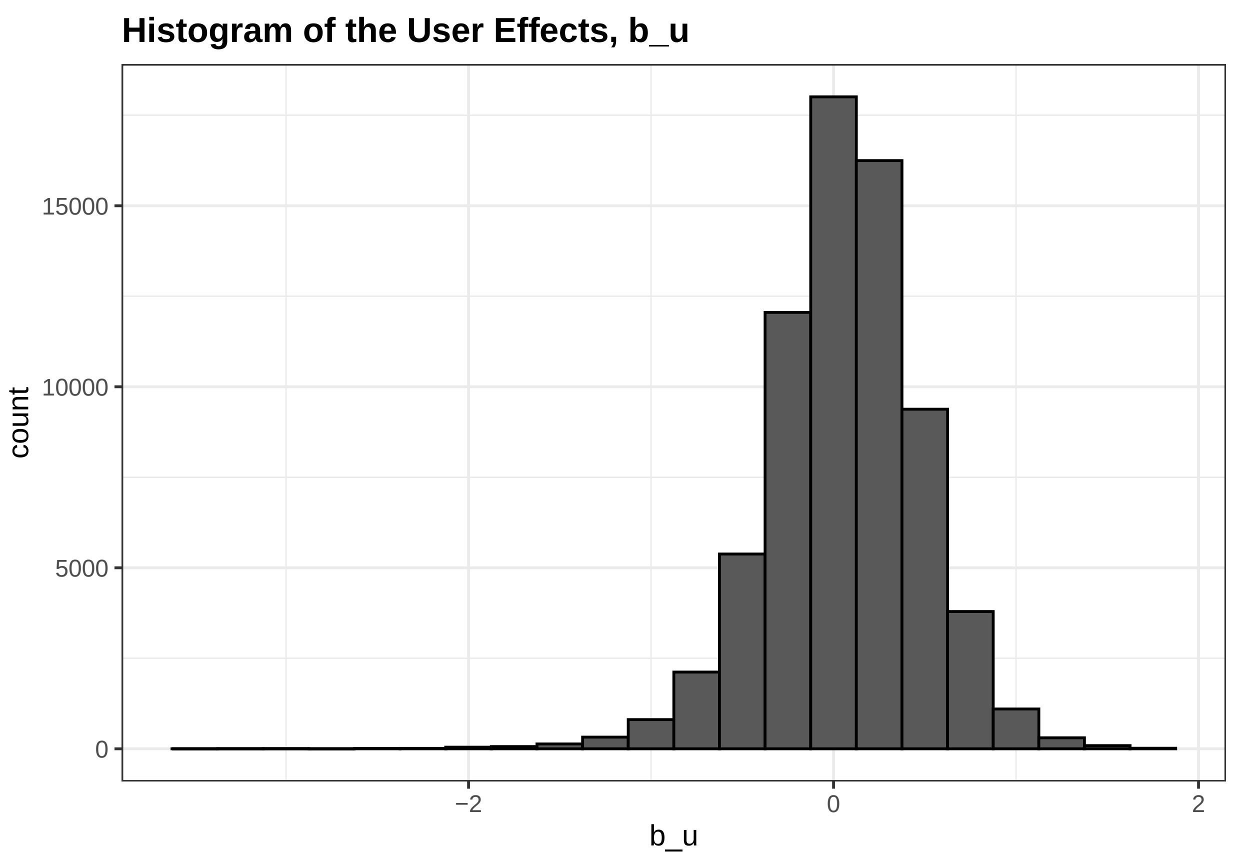
The RMSE for this model is 0.943. This is an improvement compared to the previous model.

model	RMSE
mean	1.060
movie effect	0.943

Movie and User Effect

b_u were computed by grouping by users and then getting the mean of the ratings after subtracting μ and b_i.

The histogram of the b_u 's are shown below.



We can see that there is a distribution of the b_u 's.

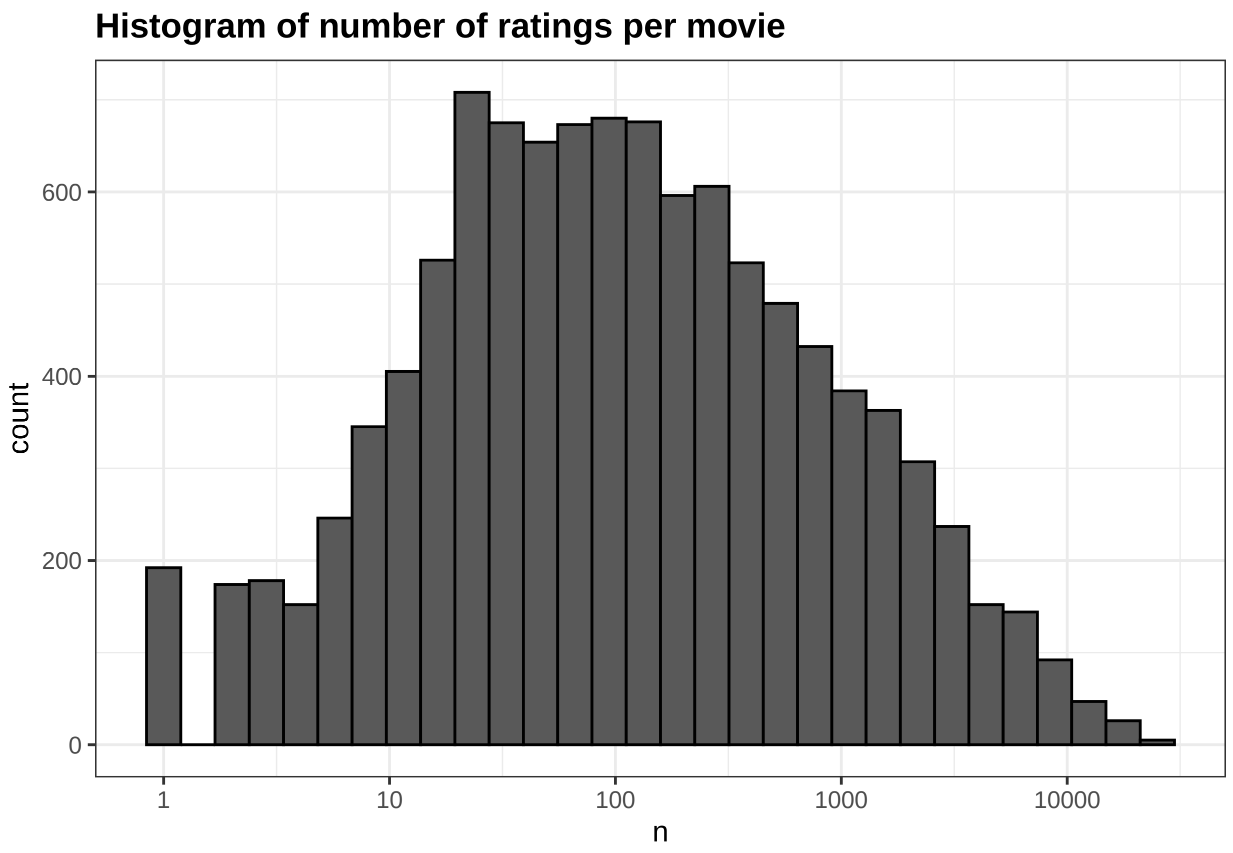
```
## [1] 0.866
```

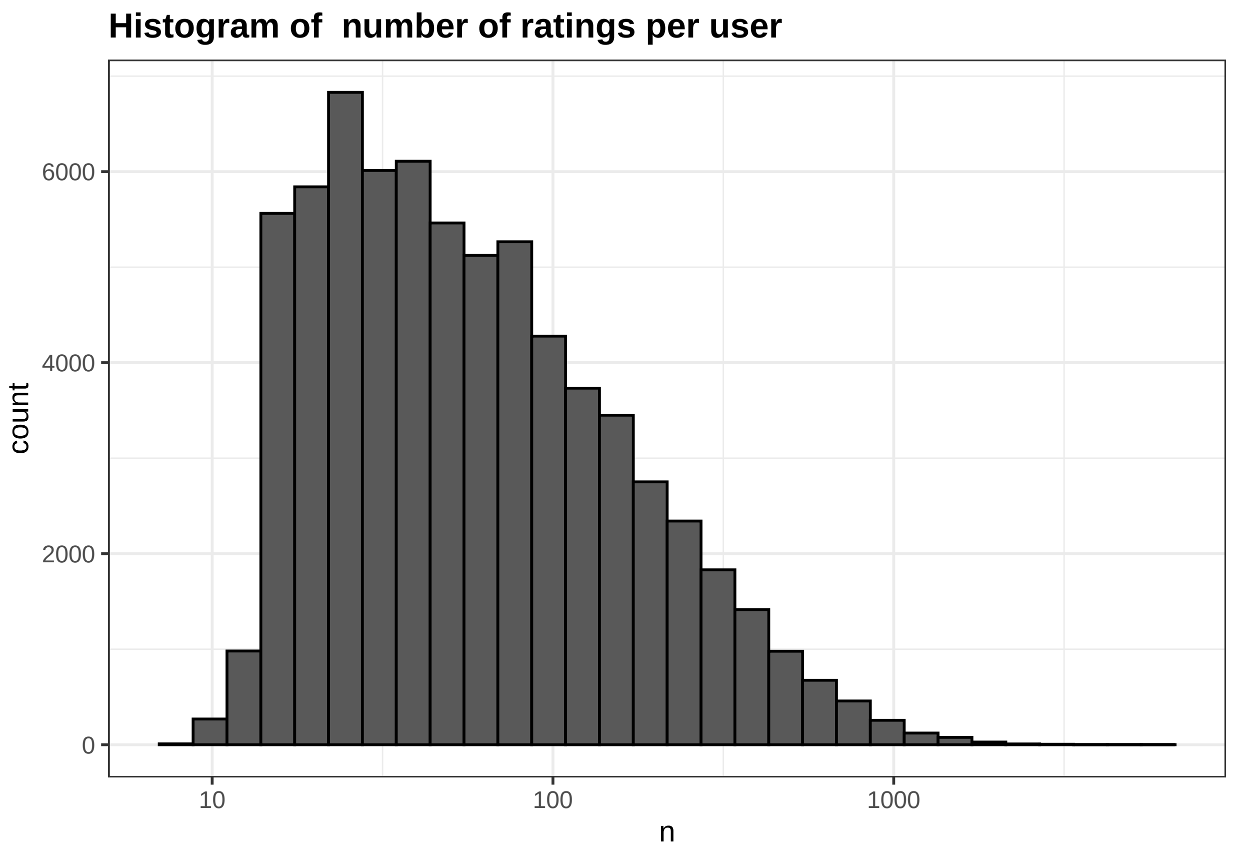
The RMSE for this model is 0.866. This is an improvement compared to the two previous models.

model	RMSE
mean	1.060
movie effect	0.943
movie and user effect	0.866

Regularized Movie and User Effect

Some movies are rated more than other movies. Similarly, some users rate more than other users. Below are the histograms for both the number of ratings per movie, and per user, respectively.

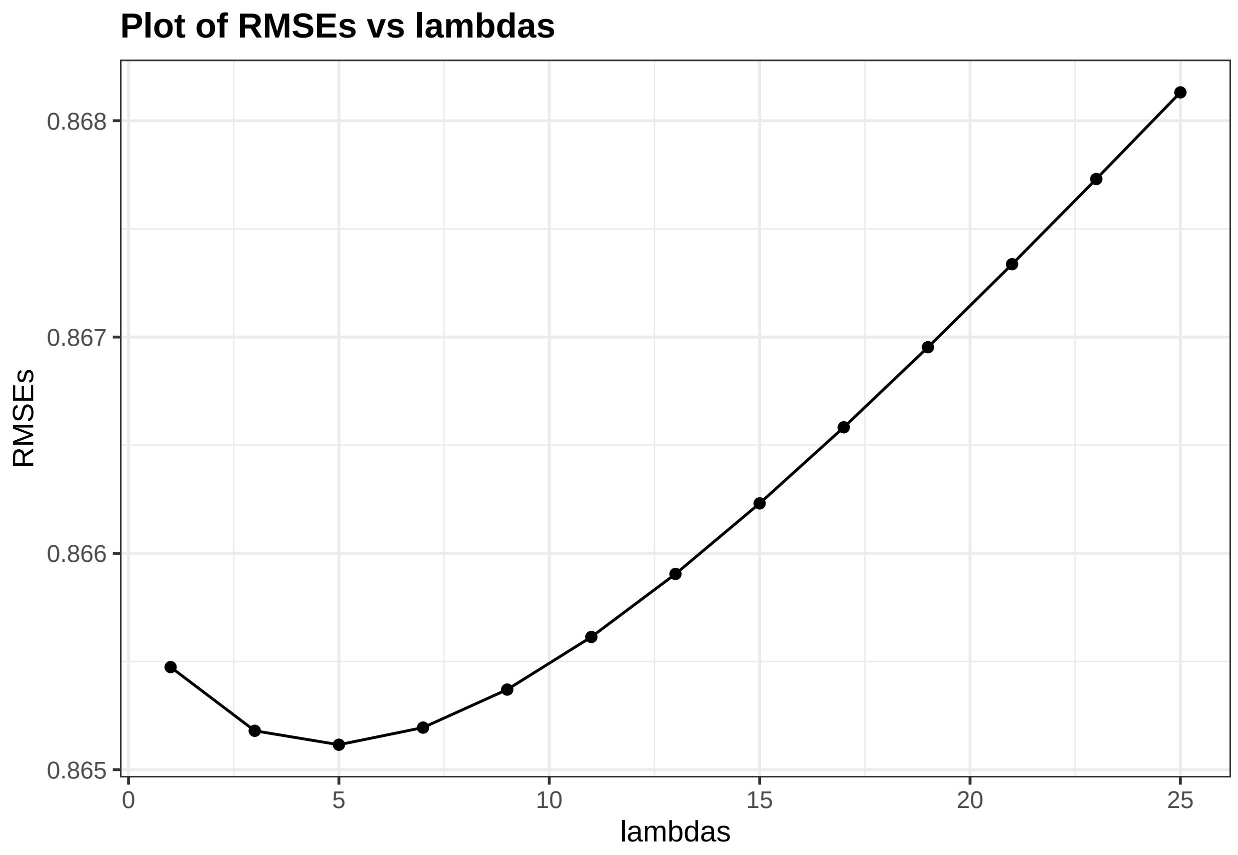

Histogram of number of ratings per movie
count
600
400
200
0
1
10
100
1000
10000
n



To penalize large movie and user effects, *lambda* was used to optimize the model. The range of lambdas used for optimization was:

```
## [1]  1  3  5  7  9 11 13 15 17 19 21 23 25
```

The plot below shows the corresponding RMSE for each lambda.



The lambda which minimizes the RMSE of the model is 5. The final model was generated using this lambda.

```
## [1] 0.865
```

The RMSE for this model is 0.865. Compared to the non regularized movie and user effects model, this model did not improve the RMSE significantly.

model	RMSE
mean	1.060
movie effect	0.943
movie and user effect	0.866
regularized movie and user effect	0.865

Matrix Factorization

Using the *recosystem* package, firstly, a set of tuning parameters were obtained. These tuning parameters were used to train the model.

```
## iter      tr_rmse          obj
##    0       0.9881   9.9441e+06
##    1       0.8791   8.0769e+06
##    2       0.8495   7.5217e+06
##    3       0.8314   7.2094e+06
##    4       0.8152   6.9752e+06
##    5        0.8020   6.7869e+06
##    6        0.7920   6.6531e+06
##    7        0.7839   6.5455e+06
##    8        0.7772   6.4592e+06
##    9        0.7716   6.3881e+06
##   10        0.7668   6.3291e+06
##   11        0.7626   6.2786e+06
##   12        0.7589   6.2377e+06
##   13        0.7555   6.1975e+06
##   14        0.7525   6.1643e+06
##   15        0.7498   6.1349e+06
##   16        0.7473   6.1091e+06
##   17        0.7451   6.0847e+06
##   18        0.7431   6.0643e+06
##   19        0.7412   6.0444e+06
```

The RMSE for this model is 0.795. This is a significant improvement from all the previous models.

model	RMSE
mean	1.060
movie effect	0.943
movie and user effect	0.866
regularized movie and user effect	0.865
recommender System	0.795

Validation

For this part, the validation set was used to evaluate the RMSEs. The summary of the RMSEs for each model is shown below.

model	RMSE
mean	1.061
movie effect	0.944
movie and user effect	0.866
Regularized movie and user effect	0.866
Recommender System	0.795

With the validation set, using matrix factorization, an RMSE of 0.795 was obtained.

Conclusion

This project explored 5 models in total. The first model is a naive approach which just predicts the mean of all the ratings. The second and third models investigated the effect of the movie and user effects. These two models improved the RMSE.

The fourth model aimed to penalize the effects from small samples. This model did not improve the RMSE as compared to the non-regularized one.

The final model used matrix factorization using the *recosystem* package. This model was the most computer-intenstive but also produced the best RMSE of 0.795 using the test data.

Evaluating the validation set using this model resulted to an RMSE of 0.795.

Limitations

The recommendation system methods used in this project does not account for new users. New users have no existing information to use as basis for recommendation. Moreover, every time a new movie or user is added, the data should be rerun to take these new data into account.

References

[1] R. Irizarry, *Introduction to Data Science: Data Analysis and Prediction Algorithms with R (2019)* [2] https://grouplens.org/ Accessed October 23, 2024 [3] https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html Accessed October 23, 2024

Appendix

Code provided by Edx:

```r
# Create train and validation sets
###### 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")
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")

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

# 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 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)
```

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

Code by User:

```r
########################### Training Data ###########################
# create training and test sets from the edx data set with 20% of the data as the test set

options(digits = 3)

set.seed(seed = 1991,sample.kind = "Rounding")

test_index<- createDataPartition(y = edx$rating,times = 1,p = 0.2,list = FALSE)
train_set<- edx[-test_index,]
test_set<- edx[test_index,]

# make sure that movieId and userId in the test set is also in the training set
temp<- test_set %>%
  semi_join(y = train_set,by = "movieId") %>%
  semi_join(y = train_set,by = "userId")

# add the removed data from test set, back to the training set
removed<- anti_join(test_set,temp)
test_set<- temp
train_set<- bind_rows(train_set,removed)

########################### Loss Function ###########################

# define a loss function to be the RMSE between the prediction and the actual rating

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

########################### First Model - Mean ###########################

# first model predicts the same rating for all movies to be equal to the average, mu
mu<- mean(train_set$rating)

# value of mu
mu

# predict with just the mu
y_hat_mean<- mu

# compute the RMSE
rmse_mean<- RMSE(predicted_ratings = y_hat_mean,actual_ratings = test_set$rating)

rmse_results<- data.frame(model="mean",RMSE=rmse_mean)

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

```r
rmse_results %>% kable()

############################ Second Model - Movie Effect ############################

# define a movie effect, b_i where the expected value of the rating, y_hat, is given by y_hat= mu + b_i

# compute for b_i by subtracting mu from each rating, then getting the mean for each movie group
b_i<- train_set %>%
  group_by(movieId) %>%
  summarize(b_i=mean(rating-mu))

# data exploration to confirm the presence of a movie effect
# histogram of movie effects
if(!require(dslabs)) install.packages("dslabs", repos = "http://cran.us.r-project.org")
library(dslabs)
ds_theme_set()
b_i %>%
  ggplot() +
  geom_histogram(aes(b_i),binwidth = 0.25, color="black") +
  labs(title = "Histogram of the Movie Effects, b_i")

# predict the ratings using this new model
y_hat_movie_effect<- test_set %>%
  left_join(y = b_i,by = "movieId") %>%
  select(movieId,rating,b_i) %>%
  mutate(predicted_rating = mu+b_i) %>%
  pull(predicted_rating)

# compute the RMSE
rmse_movie_effect<- RMSE(predicted_ratings = y_hat_movie_effect,actual_ratings = test_set$rating )

rmse_results<- rmse_results %>%
  bind_rows(data.frame(model="movie effect", RMSE=rmse_movie_effect))

rmse_results %>% kable()

############################ Third Model - Movie and User Effect ############################

# define a user effect, b_u where the expected value of the rating, y_hat, is given by y_hat= mu + b_i

# compute for b_u by subtracting mu and b_i from each rating, then getting the mean for each user group

b_u<- train_set %>%
  left_join(y = b_i,by = "movieId") %>%
  select(userId,movieId,rating,b_i) %>%
  group_by(userId) %>%
  summarize(b_u=mean(rating-mu-b_i))

# data exploration to confirm the presence of a user effect
# histogram of user effects

b_u %>%
  ggplot() +
  geom_histogram(aes(b_u),binwidth = 0.25, color="black") +
  labs(title = "Histogram of the User Effects, b_u")

# predict the ratings using this new model
y_hat_movie_and_user_effect<- test_set %>%
  left_join(y = b_i,by = "movieId") %>%
  left_join(y = b_u,by = "userId") %>%
  select(userId,movieId,rating,b_i,b_u) %>%
  mutate(predicted_rating=mu+b_i+b_u) %>%
  pull(predicted_rating)

# compute the RMSE
rmse_movie_and_user_effect<- RMSE(predicted_ratings = y_hat_movie_and_user_effect,actual_ratings = test_

rmse_results<- rmse_results %>%
  bind_rows(data.frame(model="movie and user effect",RMSE=rmse_movie_and_user_effect))

rmse_results %>% kable()

########################### Fourth Model - Regularized Movie and User Effect ##########################

# data exploration to for the motivation of using a regularized model

# distribution of the number of ratings per movie

train_set %>%
  group_by(movieId) %>%
  summarize(n=n()) %>%
  ggplot()+
  geom_histogram(aes(n), color="black") +
  # use a log10 transformation
  scale_x_continuous(transform = "log10") +
  labs(title = "Histogram of number of ratings per movie")

# distribution of the number of ratins per user

train_set %>%
  group_by(userId) %>%
  summarize(n=n()) %>%
  ggplot()+
  geom_histogram(aes(n),color="black")+
  # use a log10 transofrmation
  scale_x_continuous(transform="log10")+
  labs(title = "Histogram of  number of ratings per user")

# regularize model by adding a lambda term, l
l<- seq(1,25,2)

# optimize for lambda
results<- sapply(X = l,FUN = function(l){
  b_i_reg<- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_reg=sum(rating-mu)/(n()+l))
```

```r
  b_u_reg<- train_set %>%
    left_join(y = b_i_reg,by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg= sum(rating-mu-b_i_reg)/(n()+l))

  y_hat_reg<- test_set %>%
    left_join(y = b_i_reg,by = "movieId") %>%
    left_join(y = b_u_reg,by = "userId") %>%
    select(userId,movieId,rating,b_i_reg,b_u_reg) %>%
    mutate(predicted_rating=mu+b_i_reg+b_u_reg) %>%
    pull(predicted_rating)

  rmse= RMSE(predicted_ratings = y_hat_reg,actual_ratings = test_set$rating)
})

# plot of rmses vs lambdas
data.frame(lambda=l,rmse=results) %>%
  ggplot(aes(lambda,results)) +
  geom_point() +
  geom_line() +
  ylab("RMSEs") +
  xlab("lambdas") +
  labs(title = "Plot of RMSEs vs lambdas")

# lambda, l, which minimizes the rmses of the test_set
best_l<-l[which.min(results)]
best_l

# model based on the optimized lambda

b_i_reg<- train_set %>%
  group_by(movieId) %>%
  summarize(b_i_reg=sum(rating-mu)/(n()+best_l))

b_u_reg<- train_set %>%
  left_join(y = b_i_reg,by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u_reg= sum(rating-mu-b_i_reg)/(n()+best_l))

y_hat_reg<- test_set %>%
  left_join(y = b_i_reg,by = "movieId") %>%
  left_join(y = b_u_reg,by = "userId") %>%
  select(userId,movieId,rating,b_i_reg,b_u_reg) %>%
  mutate(predicted_rating=mu+b_i_reg+b_u_reg) %>%
  pull(predicted_rating)

rmse_reg_movie_and_user_effect<- RMSE(predicted_ratings = y_hat_reg,actual_ratings = test_set$rating)

rmse_reg_movie_and_user_effect
rmse_results<- rmse_results %>%
  bind_rows(data.frame(model="regularized movie and user effect",RMSE=rmse_reg_movie_and_user_effect))
rmse_results %>% kable()
```

```r
########################### Fifth Model - Recommender System ###########################


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

r<- Reco()
train_src<- with(train_set,data_memory(user_index = userId,item_index = movieId,rating = rating))
test_src<- with(test_set,data_memory(user_index = userId,item_index = movieId,rating = rating))

set.seed(seed = 1991,sample.kind = "Rounding")

res<- r$tune(train_data = train_src)
r$train(train_data=train_src, opts=res$min)
y_hat_reco<- r$predict(test_src)

rmse_reco<- RMSE(predicted_ratings = y_hat_reco,actual_ratings = test_set$rating)

rmse_results<- rmse_results %>%
  bind_rows(data.frame(model="recommender System",RMSE=rmse_reco))

rmse_results %>% kable()

########################### RMSE with the Validation sets ###########################
#### First Model - Mean ####
# predict with just the mu
y_hat_mean_validation<- mu

# compute the RMSE
rmse_mean_validation<- RMSE(predicted_ratings = y_hat_mean_validation,actual_ratings = validation$ratin

rmse_results_validation<- data.frame(model="mean",RMSE=rmse_mean_validation)

#### Second Model - Movie Effect ####
# predict using the movie effect, b_i
y_hat_movie_effect_validation<- validation %>%
  left_join(y = b_i,by = "movieId") %>%
  select(movieId,rating,b_i) %>%
  mutate(predicted_rating = mu+b_i) %>%
  pull(predicted_rating)

# compute the RMSE
rmse_movie_effect_validation<- RMSE(predicted_ratings = y_hat_movie_effect_validation,actual_ratings = 

rmse_results_validation<- rmse_results_validation %>%
  bind_rows(data.frame(model="movie effect", RMSE=rmse_movie_effect_validation))

#### Third Model - Movie and User Effect ####
# predict using the movie and user effects, b_i and b_u, respectively
y_hat_movie_and_user_effect_validation<- validation %>%
  left_join(y = b_i,by = "movieId") %>%
  left_join(y = b_u,by = "userId") %>%
  select(userId,movieId,rating,b_i,b_u) %>%
  mutate(predicted_rating=mu+b_i+b_u) %>%
  pull(predicted_rating)

# compute the RMSE
rmse_movie_and_user_effect_validation<- RMSE(predicted_ratings = y_hat_movie_and_user_effect_validation

rmse_results_validation<- rmse_results_validation %>%
  bind_rows(data.frame(model="movie and user effect",RMSE=rmse_movie_and_user_effect_validation))

#### Fourth Model - Regularized Movie and User Effects  ####
# predict with the regularized movie and user effects
y_hat_reg_validation<- validation %>%
  left_join(y = b_i_reg,by = "movieId") %>%
  left_join(y = b_u_reg,by = "userId") %>%
  select(userId,movieId,rating,b_i_reg,b_u_reg) %>%
  mutate(predicted_rating=mu+b_i_reg+b_u_reg) %>%
  pull(predicted_rating)

rmse_reg_movie_and_user_effect_validation<- RMSE(predicted_ratings = y_hat_reg_validation,actual_rating

rmse_reg_movie_and_user_effect_validation
rmse_results_validation<- rmse_results_validation %>%
  bind_rows(data.frame(model="Regularized movie and user effect",RMSE=rmse_reg_movie_and_user_effect_val

#### Fifth Model - Recommender System  ####

# create the required data source for reco system
validation_src<- with(validation, data_memory(user_index = userId, item_index = movieId, rating = ratin

# predict using the trained model
y_hat_reco_validation<- r$predict(validation_src)

rmse_reco_validation<- RMSE(predicted_ratings = y_hat_reco_validation,actual_ratings = validation$ratin

rmse_results_validation<- rmse_results_validation %>%
  bind_rows(data.frame(model="Recommender System",RMSE=rmse_reco_validation))

rmse_results_validation %>% kable()
```