Movielens

Shriya

2020/12/31

Contents

1 Introduction

The goal of the Movielens dataset is to create an machine learning algorithm which predicts the ratings of the movies as compared to the actual ratings. It checks the accuracy of our algorithm to predict the movie ratings.

1.0.1 Rating prediction for movies on MovieLens dataset using caret package

First we need to install all the packages and the files required to start with the prediction of the ratings of the Movielens dataset.

2 MovieLens 10M dataset:

Download the dataset from grouplens website.

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

3 Ratings table with the columns UserId, MovieId, Rating and Timestamp

Movies table where the columns are shown separately with names of columns in Movies table showing Movies Table with MovieId, Title and Genres with Left join by MovieId in movielens dataset.

```
ratings <- fread(text = gsub("::", "\t",
                             readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

4 'Validation' 'Training' 'Testing'

Validation Set will be 10% of MovieLens data. Then Separating the Training and Testing Dataset.

- Make sure userId and movieId in 'validation' set are also in 'edx' set.
- Add rows removed from 'validation' set back into 'edx' set

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

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

```
test_index <- createDataPartition(y = movielens$rating,
                                  times = 1, p = 0.1, list = FALSE)

edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

```
removed <- anti_join(temp, validation)
```

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

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

4.0.1 Purge temp columns

After downloading the required packages and files, we will make the ratings and movies tables with some specific columns and then remove temporary datasets.

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

#Machine Learning Adding the index for the Test Dataset. Then Training and Testing Dataset. Make sure userId and movieId in test set are also in train set. Add rows removed from test set back into train set.

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

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

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

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

removed <- anti_join(temp, test_set)
```

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

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

4.0.2 Purge temp columns

Removing temporary datasets.

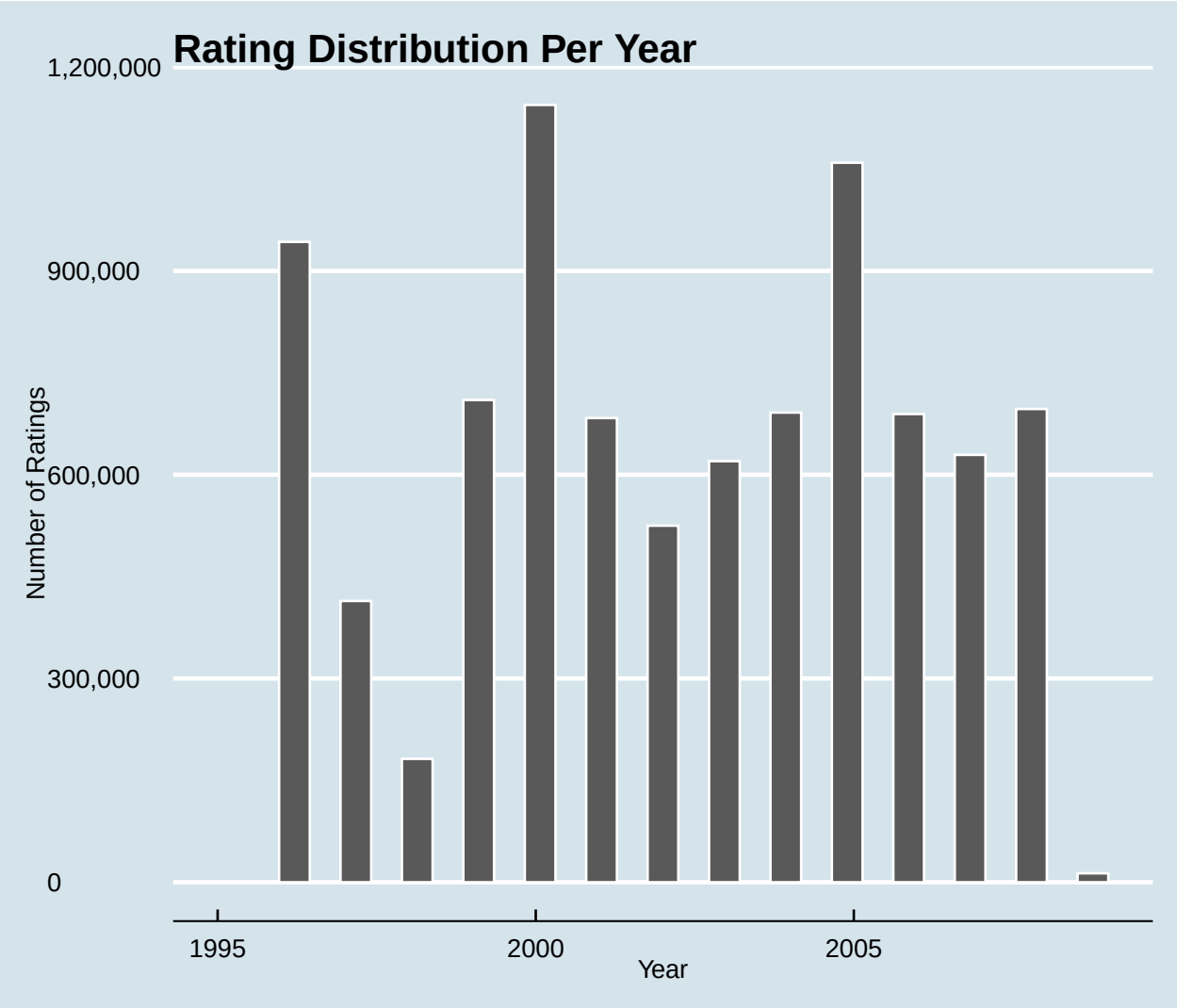
```
rm(test_index, temp, removed)
```

5 Plot the histogram

A Histogram is plotted called the Rating Distribution per year where Number of Ratings is plotted on the y-axis and the Year on the x-axis I.e it shows the ratings w.r.t year.

```
edx %>% mutate(year = year(as_datetime(timestamp, origin="1970-01-01"))) %>%
  ggplot(aes(x=year)) +
    geom_histogram(color = "white") +
    ggtitle("Rating Distribution Per Year") +
    xlab("Year") +
    ylab("Number of Ratings") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It makes a table with the top 10 movies with maximum count dated after 1970-01-01.

```
edx %>% mutate(date = date(as_datetime(timestamp, origin="1970-01-01"))) %>%
  group_by(date, title) %>%
  summarise(count = n()) %>%
  arrange(-count) %>%
  head(10)
```

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

date	title	count
1998-05-22	Chasing Amy (1997)	322
2000-11-20	American Beauty (1999)	277
1999-12-11	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	254
1999-12-11	Star Wars: Episode V - The Empire Strikes Back (1980)	251
1999-12-11	Star Wars: Episode VI - Return of the Jedi (1983)	241
2005-03-22	Lord of the Rings: The Two Towers, The (2002)	239
2005-03-22	Lord of the Rings: The Fellowship of the Ring, The (2001)	227
2000-11-20	Terminator 2: Judgment Day (1991)	221

date	title	count
1999-12-11	Matrix, The (1999)	210
2000-11-20	Jurassic Park (1993)	201

We make a table with the ratings and count of each rating.

```
edx %>% group_by(rating) %>% summarize(n=n())
```

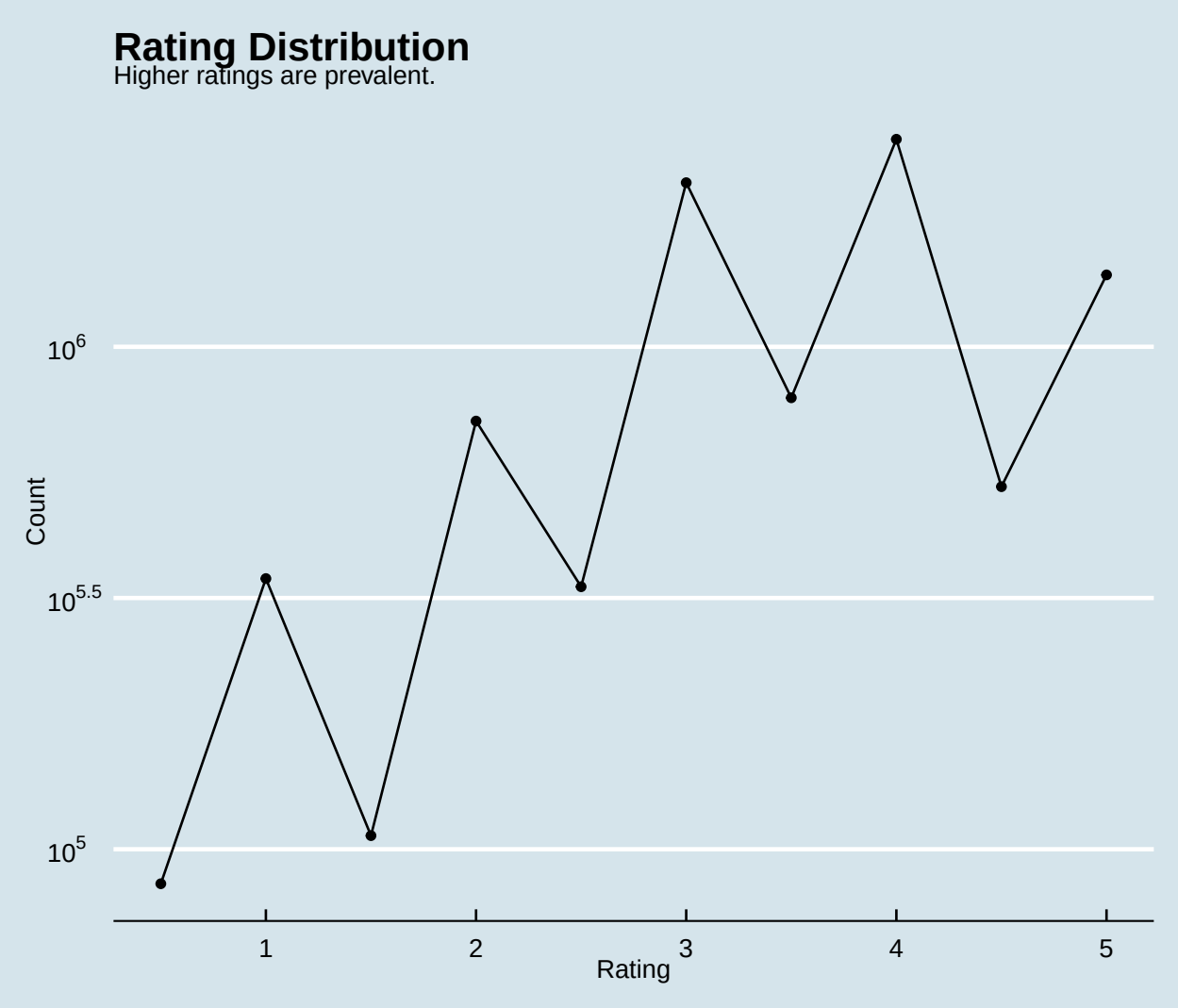
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

rating	n
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

A ratings distribution helps us visualise that higher ratings are prevalent.

```
edx %>% group_by(rating) %>%
  summarise(count=n()) %>%
  ggplot(aes(x=rating, y=count)) +
    geom_line() +
    geom_point() +
    scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                  labels = trans_format("log10", math_format(10^.x))) +
    ggtitle("Rating Distribution", subtitle = "Higher ratings are prevalent.") +
    xlab("Rating") +
ylab("Count") +
    theme_economist()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

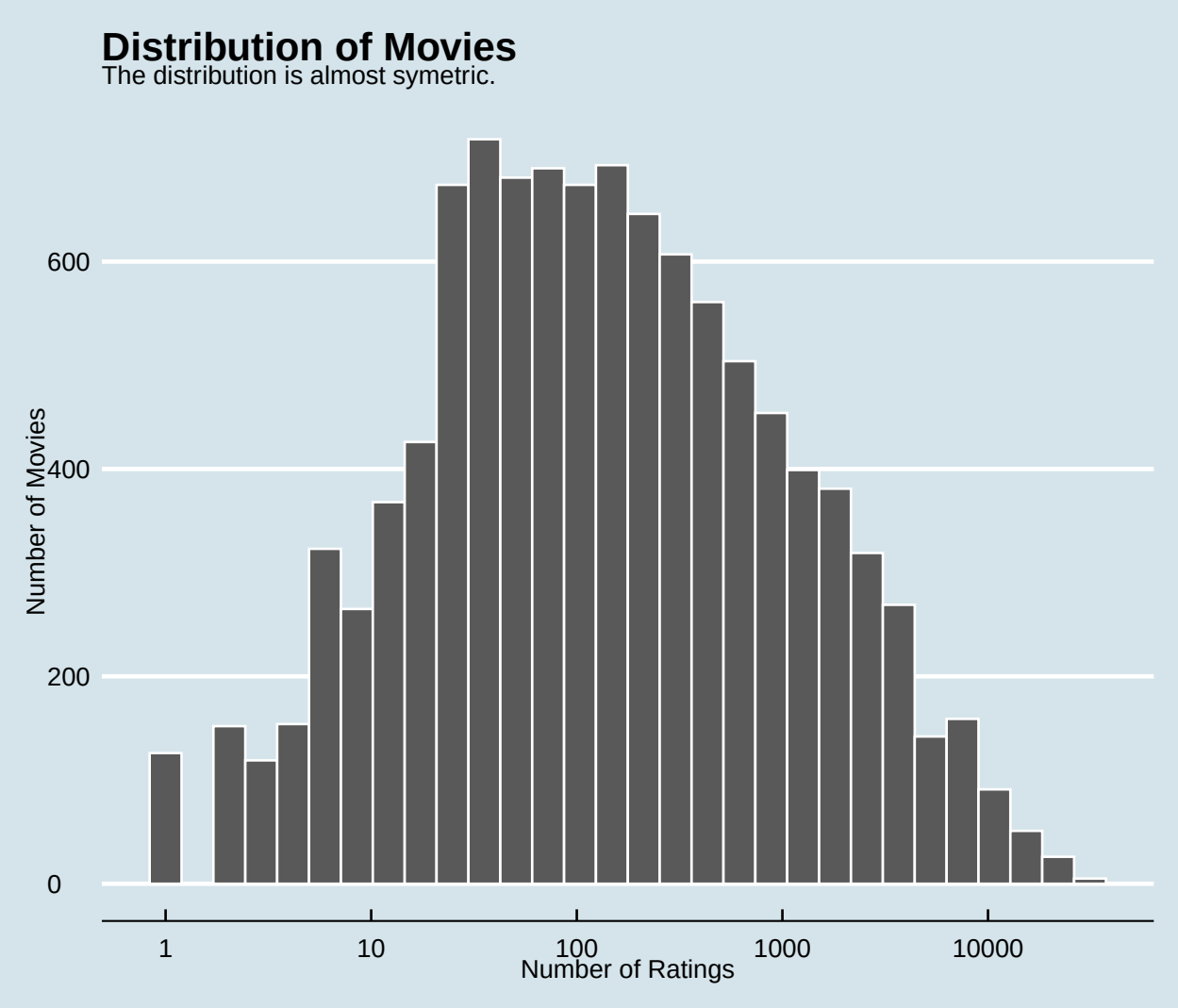


We chart a plot of the movies distribution. Here we see that it is almost symmetric.

```
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Movies",
            subtitle = "The distribution is almost symetric.") +
    xlab("Number of Ratings") +
    ylab("Number of Movies") +
    theme_economist()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We'll group by user id to get a distribution of users vs ratings

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  arrange(n) %>%
  head()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

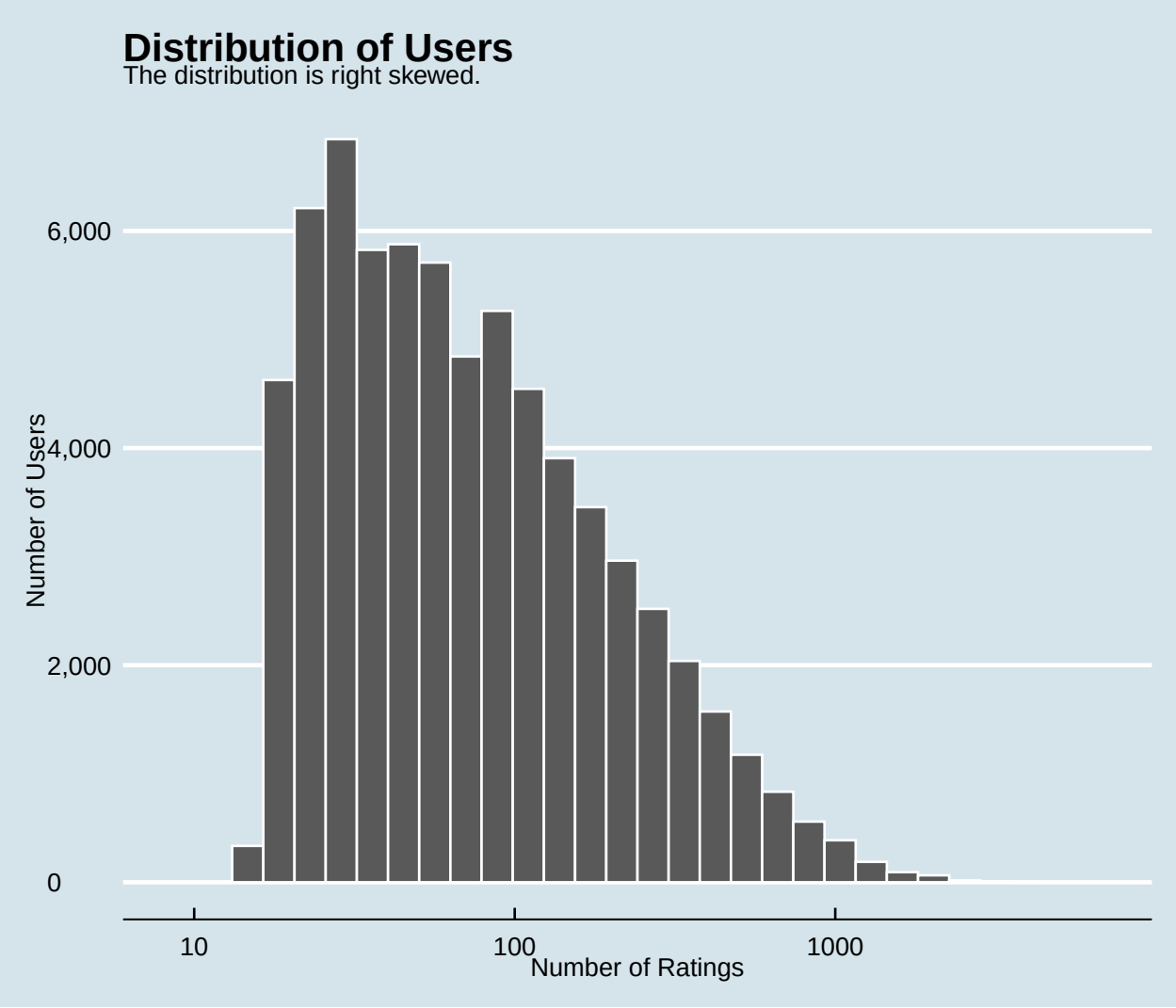
userId	n
62516	10
22170	12
15719	13
50608	13
901	14
1833	14

We can plot the graph to get a feel of how the whole thing looks.

```
edx %>% group_by(userId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
    geom_histogram(color = "white") +
    scale_x_log10() +
    ggtitle("Distribution of Users",
            subtitle="The distribution is right skewed.") +
    xlab("Number of Ratings") +
    ylab("Number of Users") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

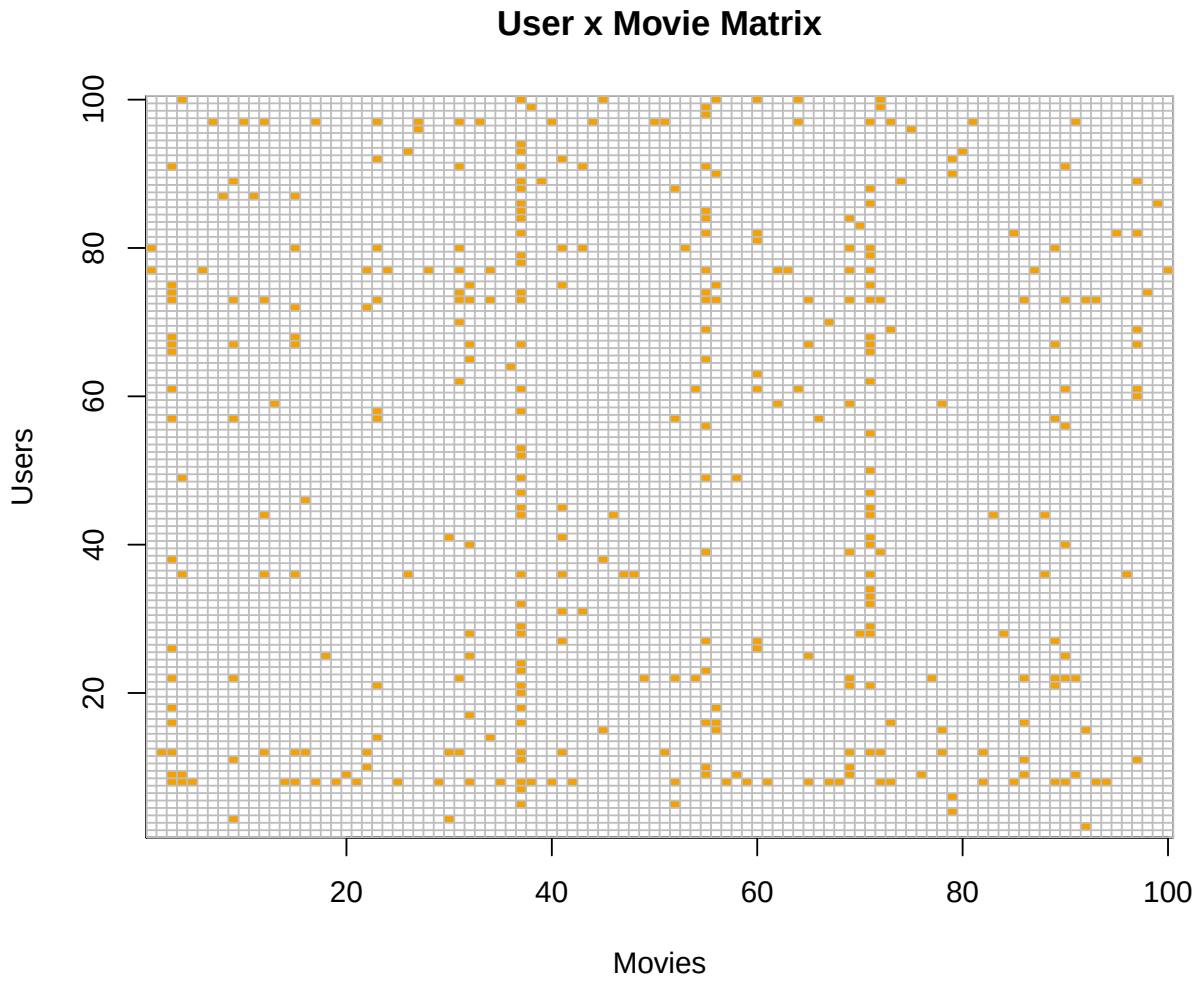
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Then we clean the dataset.

```
users <- sample(unique(edx$userId), 100)

edx %>% filter(userId %in% users) %>%
```

```
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")

abline(h=0:100+0.5, v=0:100+0.5, col = "grey")

title("User x Movie Matrix")
```



6 Train and Test set

We will partition the dataset and then divide it into training and testing dataset, so that we can train the data and then test it.

```
train_set <- train_set %>% select(userId, movieId, rating, title)
test_set  <- test_set  %>% select(userId, movieId, rating, title)
```

7 Errors

Define Mean Absolute Error (MAE) Define Mean Squared Error (MSE) Define Root Mean Squared Error (RMSE)

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

8 Probability estimation

Create the probability of each rating Estimate the probability of each rating with Monte Carlo simulation After visualizing, training and testing, we will check the accuracy of our algorithm. We will compare our predicted ratings with the actual ratings.

```
set.seed(4321, sample.kind = "Rounding")
```

```
## Warning in set.seed(4321, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
p <- function(x, y) mean(y == x)

rating <- seq(0.5,5,0.5)

B <- 10^3
M <- replicate(B, {
  s <- sample(train_set$rating, 100, replace = TRUE)
  sapply(rating, p, y= s)
})
prob <- sapply(1:nrow(M), function(x) mean(M[x,]))
```

9 RMSE

Predict random ratings. Create a table with the error results

```
y_hat_random <- sample(rating, size = nrow(test_set),
                       replace = TRUE, prob = prob)

result <- tibble(Method = "Project Goal", RMSE = 0.8649, MSE = NA, MAE = NA)
result <- bind_rows(result,
                    tibble(Method = "Random prediction",
                           RMSE = RMSE(test_set$rating, y_hat_random),
                           MSE  = MSE(test_set$rating, y_hat_random),
                           MAE  = MAE(test_set$rating, y_hat_random)))

result
```

Method	RMSE	MSE	MAE
Project Goal	0.864900	NA	NA
Random prediction	1.501245	2.253735	1.167597

Then we calculate the mean of observed values.

```
# Mean of observed values
mu <- mean(train_set$rating)

# Update the error table
result <- bind_rows(result,
                    tibble(Method = "Mean",
                           RMSE = RMSE(test_set$rating, mu),
                           MSE  = MSE(test_set$rating, mu),
                           MAE  = MAE(test_set$rating, mu)))

# Show the RMSE improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.864900	NA	NA
Random prediction	1.501245	2.253735	1.1675974
Mean	1.060054	1.123714	0.8551636

10 Movie effects (bi)

We make a table of the top 6 movieId with their average ratings

```
bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

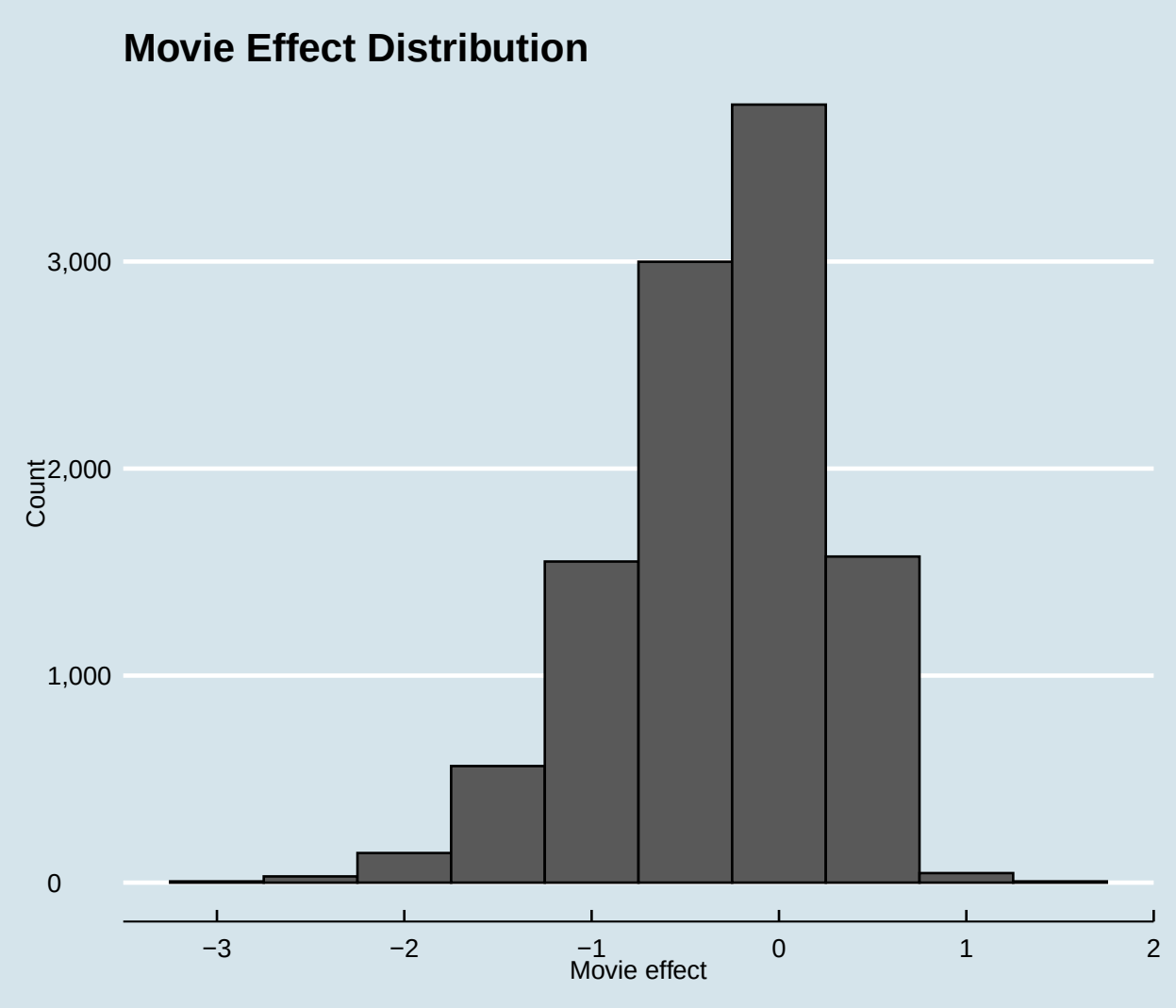
```
head(bi)
```

movieId	b_i
1	0.4150040
2	-0.3064057
3	-0.3613952
4	-0.6372808
5	-0.4416058
6	0.3018943

Plotting a histogram with the movie distribution having count in the y-axis and the movie effect on the x-axis.

```
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("black")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
```

```
  ylab("Count") +
  scale_y_continuous(labels = comma) +
  theme_economist()
```



11 RMSE calculation

Predict the rating with mean + bi and then calculate the RMSE.

```
y_hat_bi <- mu + test_set %>%
  left_join(bi, by = "movieId") %>%
  .$b_i

# Calculate the RMSE
result <- bind_rows(result,
                    tibble(Method = "Mean + bi",
                           RMSE = RMSE(test_set$rating, y_hat_bi),
                           MSE  = MSE(test_set$rating, y_hat_bi),
                           MAE  = MAE(test_set$rating, y_hat_bi)))
```

```
# Show the RMSE improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384

12 Actual prediction

Probability determination that each ratings will appear in the Movielens dataset and subsequent prediction.

```
# User effect (bu)
bu <- train_set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# Prediction
y_hat_bi_bu <- test_set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Update the results table
result <- bind_rows(result,
                    tibble(Method = "Mean + bi + bu",
                           RMSE = RMSE(test_set$rating, y_hat_bi_bu),
                           MSE  = MSE(test_set$rating, y_hat_bi_bu),
                           MAE  = MAE(test_set$rating, y_hat_bi_bu)))

# Show the RMSE improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369

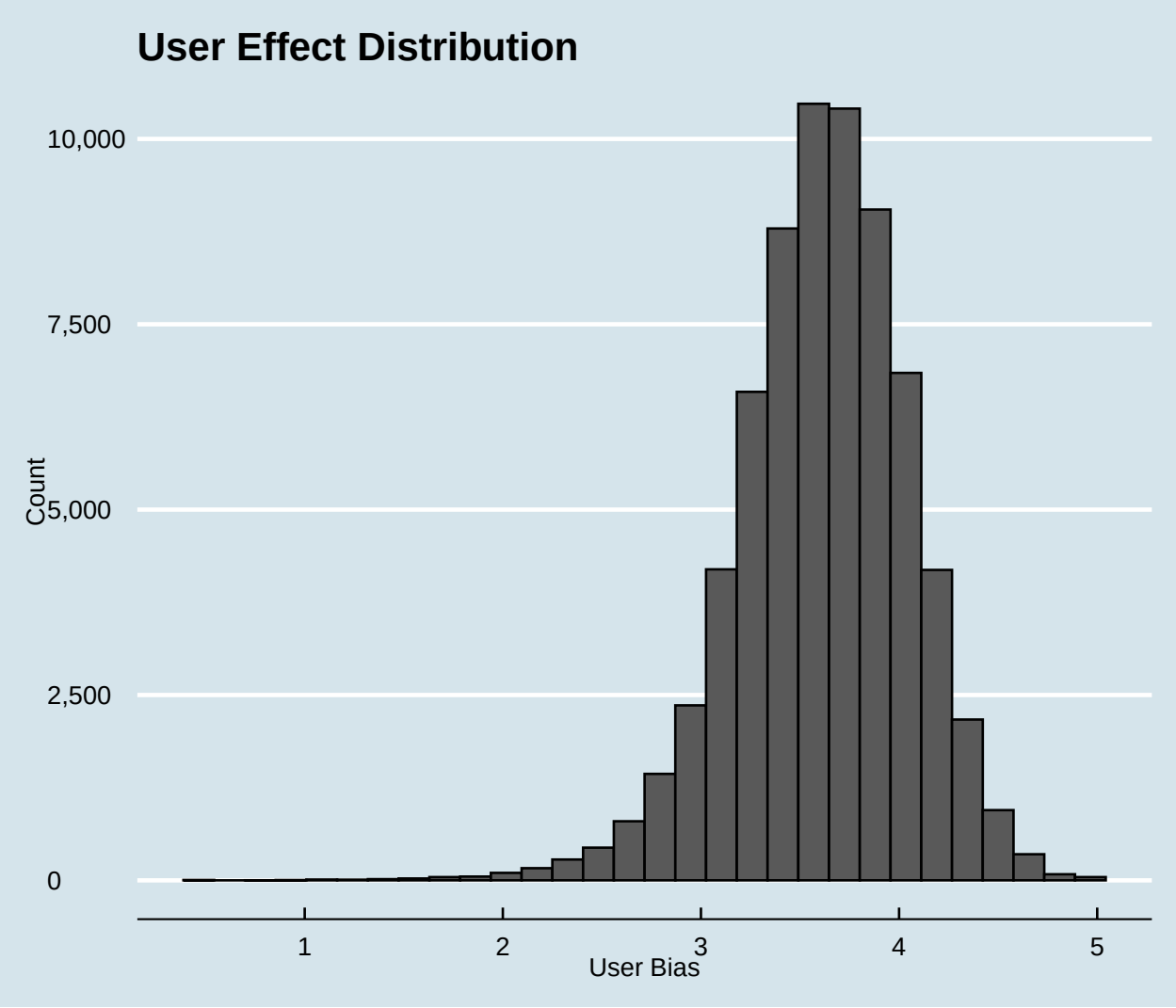
13 User distribution

Plotting a histogram showing the user distribution

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
    geom_histogram(color = "black") +
    ggtitle("User Effect Distribution") +
    xlab("User Bias") +
    ylab("Count") +
    scale_y_continuous(labels = comma) +
    theme_economist()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It makes a table with top 10 data with the coumns UserId, movieId, rating, title, b_i and residual

```
train_set %>%
  left_join(bi, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
```

```
  arrange(desc(abs(residual))) %>%
  slice(1:10)
```

userId	movieId	rating	title	b_i	residual
26423	6483	5.0	From Justin to Kelly (2003)	-2.638139	4.125683
5279	6371	5.0	Pokémon Heroes (2003)	-2.472133	3.959677
57863	6371	5.0	Pokémon Heroes (2003)	-2.472133	3.959677
2507	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567
7708	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567
9214	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567
9568	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567
9975	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567
10749	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567
13496	318	0.5	Shawshank Redemption, The (1994)	0.944111	-3.956567

```
# It shows the data of the MovieId and Title separately
titles <- train_set %>%
  select(movieId, title) %>%
  distinct()

# It shows the top 6 titles
bi %>%
  inner_join(titles, by = "movieId") %>%
  arrange(-b_i) %>%
  select(title) %>%
  head()
```

title
Hellhounds on My Trail (1999)
Satan's Tango (Sátántangó) (1994)
Shadows of Forgotten Ancestors (1964)
Fighting Elegy (Kenka erejii) (1966)
Sun Alley (Sonnenallee) (1999)
Blue Light, The (Das Blaue Licht) (1932)

A table with the top 10 movies with their titles and count

```
# It shows top 6 titles
bi %>%
  inner_join(titles, by = "movieId") %>%
  arrange(b_i) %>%
  select(title) %>%
  head()
```

title
Besotted (2001)
Hi-Line, The (1999)
Accused (Anklaget) (2005)
Confessions of a Superhero (2007)
War of the Worlds 2: The Next Wave (2008)

title
SuperBabies: Baby Geniuses 2 (2004)

14 Lambda Tuning

Calculate, tune and plot lambda parameters.

```
train_set %>%
  left_join(bi, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  group_by(title) %>%
  summarise(n = n()) %>%
  slice(1:10)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

title	n
...All the Marbles (a.k.a. The California Dolls) (1981)	17
...And God Created Woman (Et Dieu... créa la femme) (1956)	68
...And God Spoke (1993)	19
...And Justice for All (1979)	500
'burbs, The (1989)	1201
'night Mother (1986)	178
'Round Midnight (1986)	40
'Til There Was You (1997)	242
"Great Performances" Cats (1998)	4
[Rec] (2007)	59

```
# Selects columns into vector
train_set %>% count(movieId) %>%
  left_join(bi, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1] 1 1 1 1 1 1 4 2 4 4
```

```
regularization <- function(lambda, trainset, testset){
  # Mean
  mu <- mean(trainset$rating)

  # Movie effect (bi)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # User effect (bu)
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
```

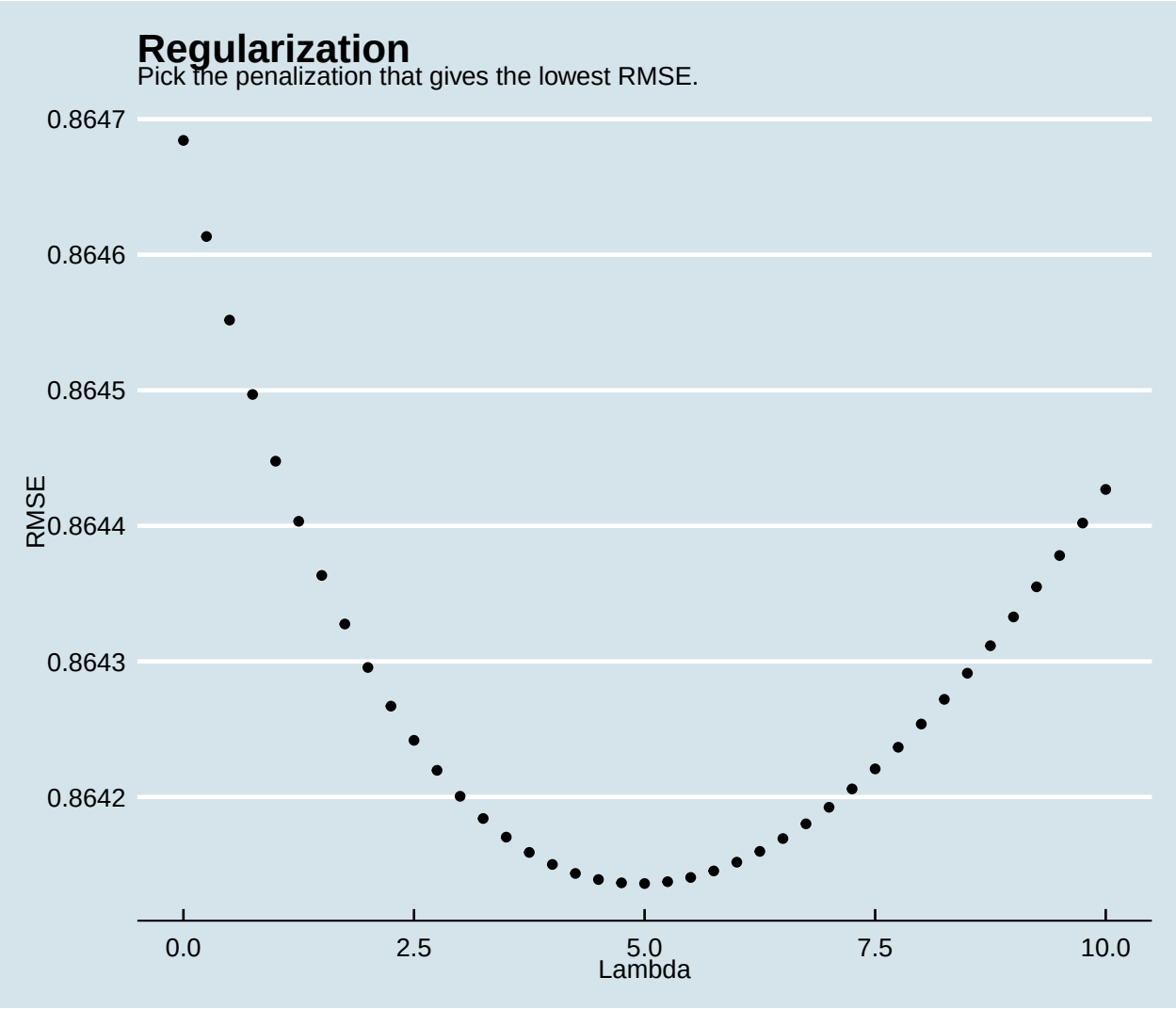
```r
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Prediction: mu + bi + bu
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}

# Define a set of lambdas to tune
lambdas <- seq(0, 10, 0.25)

# Tune lambda
rmses <- sapply(lambdas,
                regularization,
                trainset = train_set,
                testset = test_set)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# Plot the lambda vs RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
```

```
    geom_point() +
    ggtitle("Regularization",
            subtitle = "Pick the penalization that gives the lowest RMSE.") +
    theme_economist()
```



We pick the lambda that returns the lowest RMSE. Then, we calculate the predicted rating using the best parameters achieved from regularization. We can observe that Regularization made a small improvement in RMSE.

```
lambda <- lambdas[which.min(rmses)]
mu <- mean(train_set$rating)

# Movie effect (bi)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# User effect (bu)
b_u <- train_set %>%
```

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# Prediction
y_hat_reg <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Update the result table
result <- bind_rows(result,
                    tibble(Method = "Regularized bi and bu",
                           RMSE = RMSE(test_set$rating, y_hat_reg),
                           MSE  = MSE(test_set$rating, y_hat_reg),
                           MAE  = MAE(test_set$rating, y_hat_reg)))

# Display the improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070

Shows training data in the matrix format

```
train_data <- train_set %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
```

15 Model creation

Convert the train and test sets into recosystem input format

```
set.seed(123, sample.kind = "Rounding") # This is a randomized algorithm
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
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))
```

```
# Create the model object
r <-  recosystem::Reco()

# Select the best tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = 4, niter = 10))
```

Train the algorithm and calculate the predicted values

```
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9825   1.1041e+07
##    1       0.8748   8.9747e+06
##    2       0.8422   8.3343e+06
##    3       0.8212   7.9600e+06
##    4       0.8052   7.6991e+06
##    5       0.7927   7.5058e+06
##    6       0.7825   7.3597e+06
##    7       0.7739   7.2446e+06
##    8       0.7665   7.1504e+06
##    9       0.7601   7.0721e+06
##   10       0.7542   7.0025e+06
##   11       0.7490   6.9421e+06
##   12       0.7445   6.8952e+06
##   13       0.7404   6.8507e+06
##   14       0.7365   6.8102e+06
##   15       0.7330   6.7750e+06
##   16       0.7299   6.7459e+06
##   17       0.7269   6.7168e+06
##   18       0.7242   6.6914e+06
##   19       0.7218   6.6671e+06
```

```
y_hat_reco <-  r$predict(test_data, out_memory())
head(y_hat_reco, 10)
```

```
##  [1] 4.924591 3.791518 3.228763 3.040660 3.586447 3.729566 3.675250 3.269653
##  [9] 3.967616 3.187514
```

16 Result

It makes a result table with the columns – Method, RMSE, MSE, MAE

```
result <- bind_rows(result,
                    tibble(Method = "Matrix Factorization - recosystem",
                           RMSE = RMSE(test_set$rating, y_hat_reco),
                           MSE  = MSE(test_set$rating, y_hat_reco),
                           MAE  = MAE(test_set$rating, y_hat_reco)))
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070
Matrix Factorization - recosystem	0.7859317	0.6176887	0.6054374

17 Movie Effect

Calculate user effect on prediction and subsequent RMSE improvement.

```
mu_edx <- mean(edx$rating)

# Movie effect (bi)
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# User effect (bu)
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# Prediction
y_hat_edx <- validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  pull(pred)

# Update the results table
result <- bind_rows(result,
                    tibble(Method = "Final Regularization (edx vs validation)",
                           RMSE = RMSE(validation$rating, y_hat_edx),
                           MSE  = MSE(validation$rating, y_hat_edx),
                           MAE  = MAE(validation$rating, y_hat_edx)))

# Show the RMSE improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369

Method	RMSE	MSE	MAE
Regularized bi and bu	0.8641362	0.7467313	0.6687070
Matrix Factorization - recosystem	0.7859317	0.6176887	0.6054374
Final Regularization (edx vs validation)	0.8648177	0.7479097	0.6693494

Then show the top 10 titles of the Validation set

```
validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  arrange(-pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Usual Suspects, The (1995)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Eternal Sunshine of the Spotless Mind (2004)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Schindler's List (1993)
Donnie Darko (2001)
Star Wars: Episode VI - Return of the Jedi (1983)
Schindler's List (1993)

Join `validation` dataset with `b_i_edx`

```
validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Battlefield Earth (2000)
Police Academy 4: Citizens on Patrol (1987)
Karate Kid Part III, The (1989)
Pokémon Heroes (2003)
Turbo: A Power Rangers Movie (1997)
Kazaam (1996)
Pokémon Heroes (2003)
Free Willy 3: The Rescue (1997)
Shanghai Surprise (1986)
Steel (1997)

Convert `edx_reco` and `validation_reco` sets to recosystem input format

```
set.seed(1234, sample.kind = "Rounding")
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating = rating))
validation_reco  <-  with(validation, data_memory(user_index = userId,
                                                  item_index = movieId,
                                                  rating = rating))

# Create the model object
r <-  recosystem::Reco()

# Tune the parameters
opts <-  r$tune(edx_reco, opts = list(dim = c(10, 20, 30),
                                      lrate = c(0.1, 0.2),
                                      costp_l2 = c(0.01, 0.1),
                                      costq_l2 = c(0.01, 0.1),
                                      nthread  = 4, niter = 10))
```

18 Model Training

Train the model.

```
r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9729   1.1998e+07
##    1       0.8724   9.8901e+06
##    2       0.8381   9.1617e+06
##    3       0.8162   8.7448e+06
##    4       0.8007   8.4687e+06
##    5       0.7889   8.2682e+06
##    6       0.7794   8.1191e+06
##    7       0.7715   8.0011e+06
##    8       0.7649   7.9048e+06
##    9       0.7591   7.8266e+06
##   10       0.7541   7.7599e+06
##   11       0.7496   7.7018e+06
##   12       0.7455   7.6557e+06
##   13       0.7417   7.6101e+06
##   14       0.7383   7.5729e+06
##   15       0.7352   7.5372e+06
##   16       0.7323   7.5053e+06
##   17       0.7296   7.4778e+06
##   18       0.7272   7.4527e+06
##   19       0.7249   7.4314e+06
```

```
# Calculate the prediction
y_hat_final_reco <-  r$predict(validation_reco, out_memory())
```

```r
# Update the result table
result <- bind_rows(result,
                    tibble(Method = "Final Matrix Factorization - recosystem",
                           RMSE = RMSE(validation$rating, y_hat_final_reco),
                           MSE  = MSE(validation$rating, y_hat_final_reco),
                           MAE  = MAE(validation$rating, y_hat_final_reco)))

# Show the RMSE improvement
result
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5012445	2.2537350	1.1675974
Mean	1.0600537	1.1237139	0.8551636
Mean + bi	0.9429615	0.8891764	0.7370384
Mean + bi + bu	0.8646843	0.7476789	0.6684369
Regularized bi and bu	0.8641362	0.7467313	0.6687070
Matrix Factorization - recosystem	0.7859317	0.6176887	0.6054374
Final Regularization (edx vs validation)	0.8648177	0.7479097	0.6693494
Final Matrix Factorization - recosystem	0.7830406	0.6131525	0.6033935

Top 10 movie titles from validation set

```r
tibble(title = validation$title, rating = y_hat_final_reco) %>%
  arrange(-rating) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Rhyme & Reason (1997)
Lord of the Rings: The Return of the King, The (2003)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Schindler's List (1993)
Shawshank Redemption, The (1994)
Lone Ranger, The (1956)
Shawshank Redemption, The (1994)
Cats Don't Dance (1997)
Shawshank Redemption, The (1994)
Godfather, The (1972)

Create the final validation dataset.

```r
tibble(title = validation$title, rating = y_hat_final_reco) %>%
  arrange(rating) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Time Walker (a.k.a. Being From Another Planet) (1982)

title
Dead Girl, The (2006)
Alien from L.A. (1988)
Madhouse (1990)
Giant Gila Monster, The (1959)
Zombie Lake (Le Lac des morts vivants) (1981)
Free Willy 2: The Adventure Home (1995)
Daddy Day Camp (2007)
Beast of Yucca Flats, The (1961)
Kiss Them for Me (1957)

#Conclusion We are able to predict the ratings of the Movielens dataset by training and testing our machine learning algorithm. By comparing the predicted ratings with the actual ratings, we can conclude that our predicted rating is quite close to the actual ratings though not exactly the same. Thus, our algorithm is successful in predicting the movie ratings.