Movie Recommendation System | Machine Learning Project

HarvardX: PH125.9x Data Science Capstone

Ian Mathers | February 22, 2021

Introduction

The goal of this project is to build a movie recommendation system using machine learning. The first step will be to look at the structure of the data, visualize it and then progressively build a model that will reach the targeted accuracy.

The code and a PDF version are available on GitHub. The report is published online on RPubs.

MovieLens Dataset

The data was collected and provided by GroupLens. A research lab at the University of Minnesota that specializes in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries and local geographic information systems. They have collected millions of movie reviews and offer these data sets in a range of sizes. For this project the 10M version will be used. It contains 10 million ratings on 10,000 movies by 72,000 users. It was released in 2009.

Data Loading

```
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)

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

Training and Testing of the Algorithm

The 10M dataset is divided into two sets: edx and validation. The former is further split into two where the algorithm will be built and tested. Its final accuracy will then be tested using the validation set.

```
# Create train and test sets
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Matching userId and movieId in both train and test sets
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Adding back rows into train set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

Exploratory Data Analysis

We start by analyzing the data structure. There are 9,000,055 observations and 6 columns. Each observation represents a rating given by one user for one movie. Columns include userId, movieId, rating, timestamp, title and genres. Timestamps represent seconds since midnight UTC January 1, 1970.

```
str(edx)
```

```
## Classes 'data.table' and '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
##  - attr(*, ".internal.selfref")=<externalptr>
```

The summary function provides a statistical summary of the data.

```
edx %>% select(-genres) %>% summary()
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title
##  Length:9000055
##  Class :character
##  Mode  :character
##
##
##
```

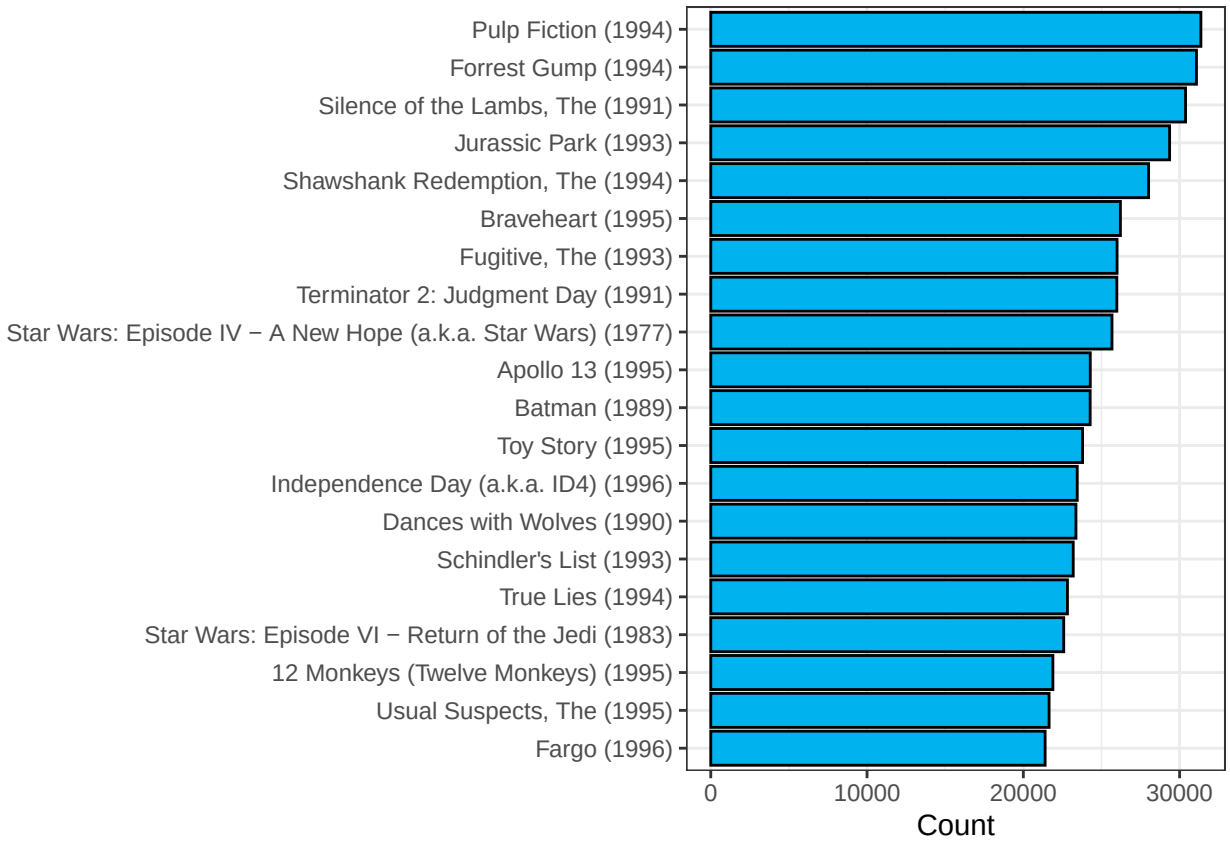
There are 69,878 unique users and 10,677 movies.

```
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

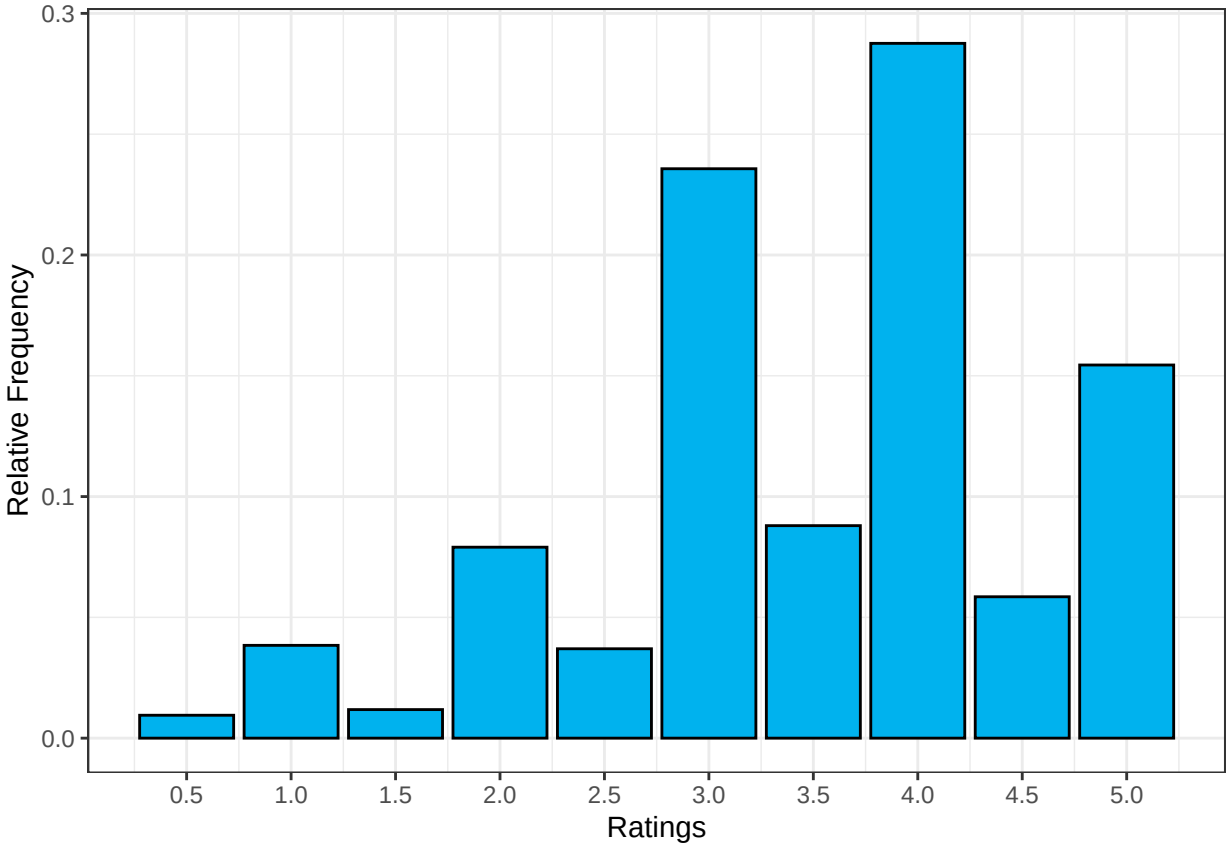
The most popular movies in this dataset.

```
edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange(-count) %>%
  top_n(20, count) %>%
  ggplot(aes(count, reorder(title, count))) +
  geom_bar(color = "black", fill = "deepskyblue2", stat = "identity") +
  xlab("Count") +
  ylab(NULL) +
  theme_bw()
```



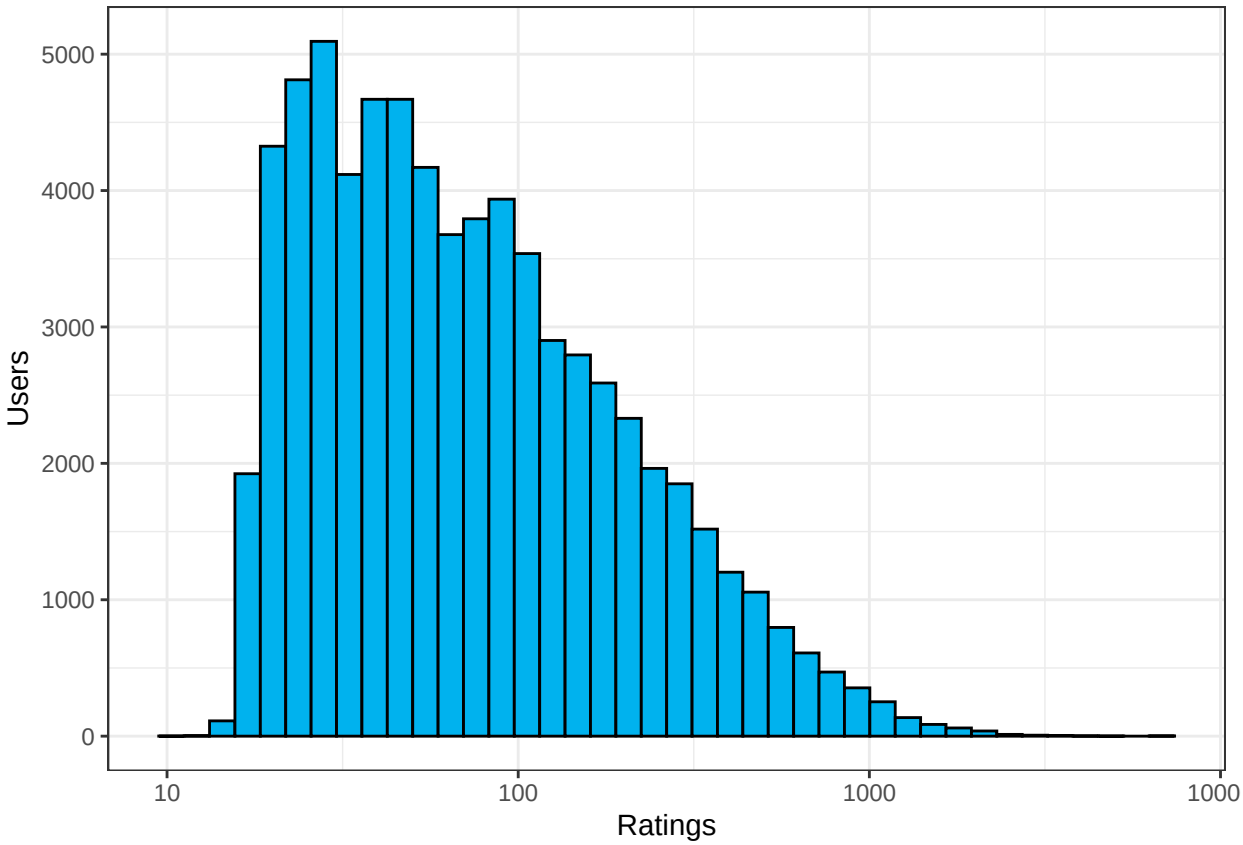
The distribution of the movie ratings shows a range of 0.5 to 5 with whole numbers used more often.

```
edx %>%
  ggplot(aes(rating, y = ..prop..)) +
  geom_bar(color = "black", fill = "deepskyblue2") +
  labs(x = "Ratings", y = "Relative Frequency") +
  scale_x_continuous(breaks = c(0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)) +
  theme_bw()
```



The number of ratings VS users shows a right skew in its distribution.

```
edx %>% group_by(userId) %>%
  summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color = "black", fill = "deepskyblue2", bins = 40) +
  xlab("Ratings") +
  ylab("Users") +
  scale_x_log10() +
  theme_bw()
```



Approach and Evaluation

Several models will be assessed starting with the simplest. Accuracy will be evaluated using the residual mean squared error (RMSE):

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

N is defined as the number of user/movie combinations, $y_{u,i}$ as the rating for movie i by user u with the prediction as $\hat{y}_{u,i}$. The RMSE is a commonly used loss function that simply measures the differences between predicted and observed values. It can be interpreted similarly to a standard deviation. For this exercise if the number is larger than 1 it means our typical error is larger than one star. The goal is to reduce this error below 0.8649. Accuracies will be compared across all models using the code below.

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

Model 1

The first model is the simplest approach we could take in making a prediction by only using the average of all movie ratings. With differences explained by random variation.

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

$\epsilon_{u,i}$ is defined as the independent sample errors and μ the true rating for all movies. Statistical theory tells us that the estimate that minimizes the RMSE is the least squares estimate of μ. For this exercise it is the average of all ratings.

```
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512456
```

If we base our prediction using this mean we obtain the following RMSE:

```
naive_rmse <- RMSE(test_set$rating, mu_hat)
results <- tibble(Method = "Model 1: Simply the mean", RMSE = naive_rmse)
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.060054

Our typical error is larger than 1 star. It was a first simple attempt that lacks accuracy. Our next model will build on this.

Model 2

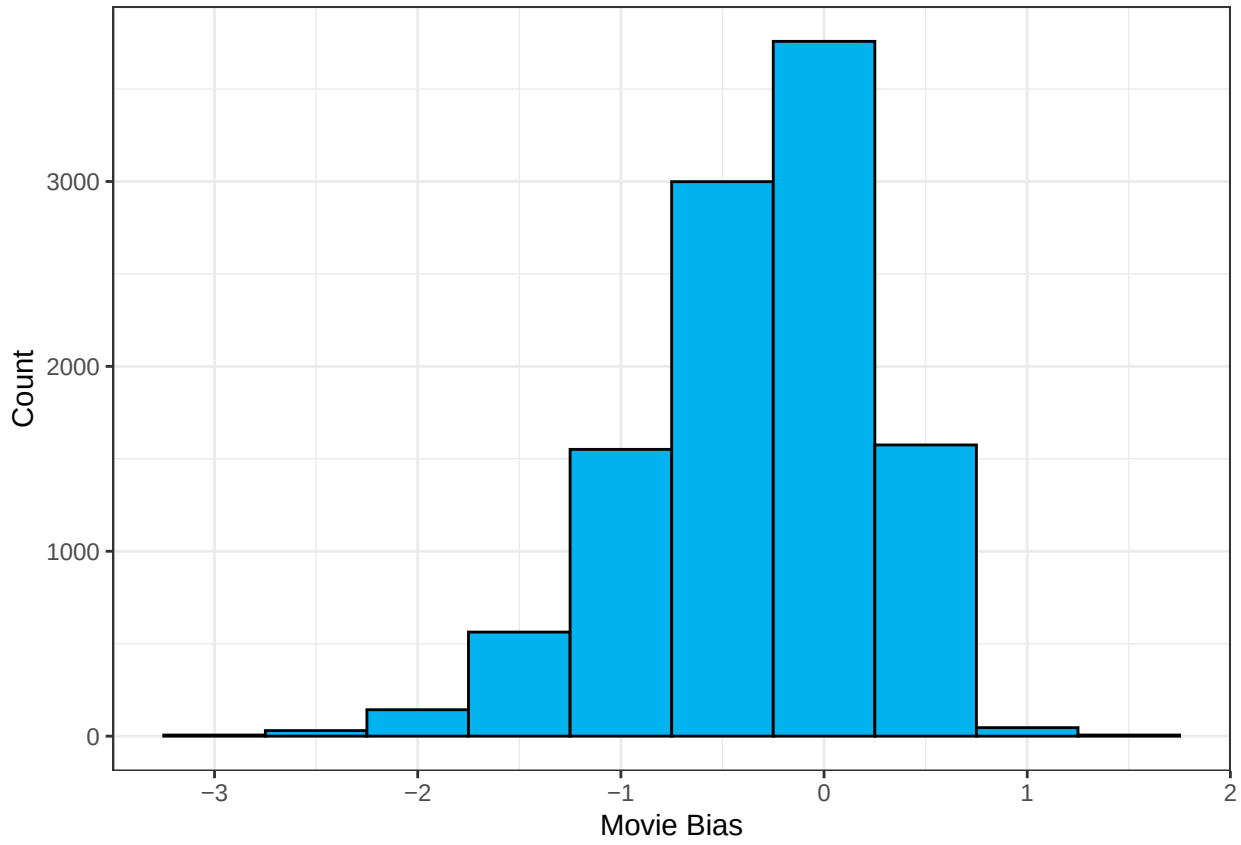
Our first model can be improved on by taking into account movie bias. We know from experience, and data confirms this, that some movies are more popular than others and receive higher ratings. We can add the term b_i to reflect this. It is the average of $Y_{u,i} - \hat{\mu}$ for each movie i.

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

We can see this bias through this distribution. The mean is at 0 so a b_i of 1.5 reflects a 5 star rating.

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

```
bi %>% ggplot(aes(b_i)) +
  geom_histogram(color = "black", fill = "deepskyblue2", bins = 10) +
  xlab("Movie Bias") +
  ylab("Count") +
  theme_bw()
```



Here is the impact of adding this bias to our model by running a RMSE test:

```
predicted_ratings <- mu_hat + test_set %>%
  left_join(bi, by = "movieId") %>%
  pull(b_i)
m_bias_rmse <- RMSE(predicted_ratings, test_set$rating)
results <- bind_rows(results, tibble(Method = "Model 2: Mean + movie bias", RMSE = m_bias_rmse))
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.0600537
Model 2: Mean + movie bias	0.9429615

Model 3

Bias can be found in users as well. Some tend to rate more positively and others negatively. We can add this effect to the model as b_u.

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

We can estimate $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$.

```
bu <- train_set %>%
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

A RMSE test will show how much we can reduce our typical error by adding this bias:

```
predicted_ratings <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
u_bias_rmse <- RMSE(predicted_ratings, test_set$rating)
results <- bind_rows(results, tibble(Method = "Model 3: Mean + movie bias + user effect", RMSE = u_bias
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.0600537
Model 2: Mean + movie bias	0.9429615
Model 3: Mean + movie bias + user effect	0.8646843

By simply factoring in movie and user biases we've managed to lower our error to 0.8647. We'll continue to improve on that.

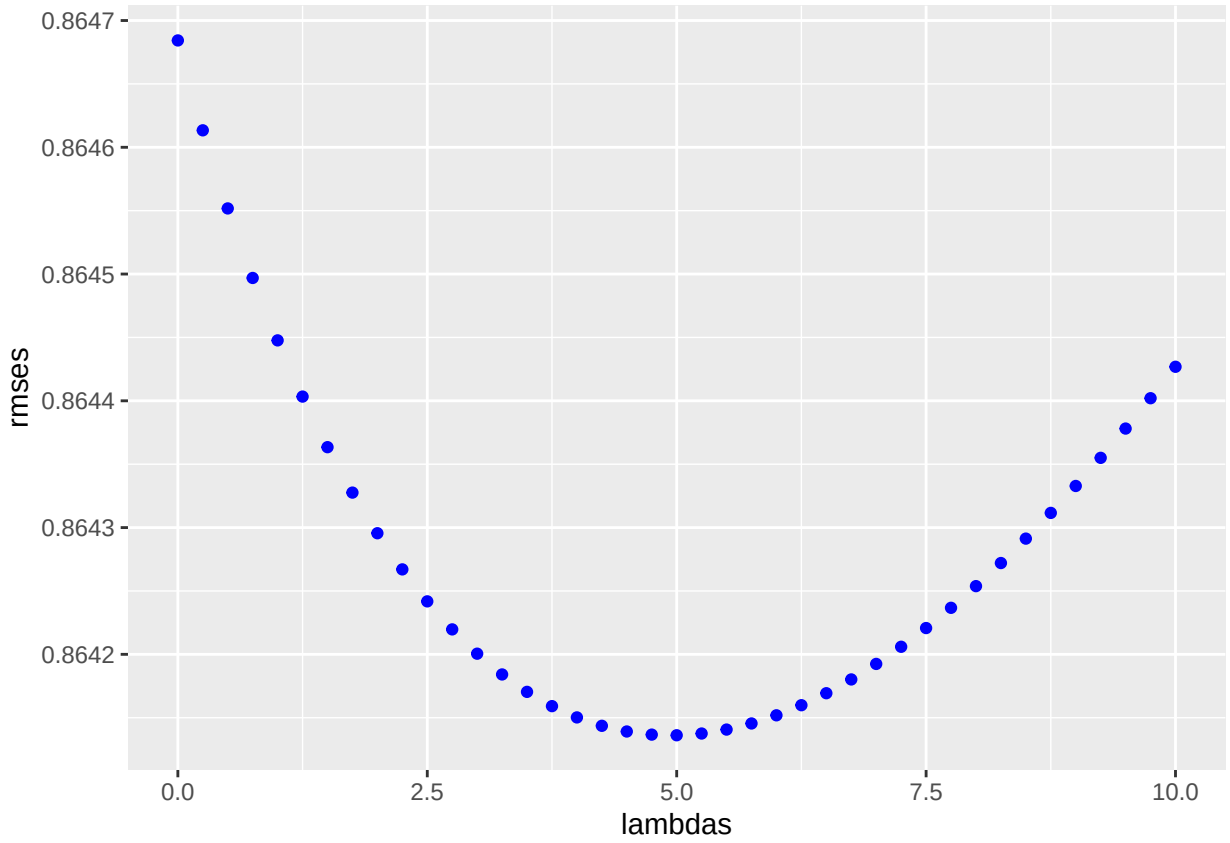
Model 4

Some of the data is noisy. For example ratings on obscure or niche movies by only a few users. This adds variability and can increase RMSE. We can use regularization to penalize large estimates formed by small sample sizes to reduce this effect. The optimal penalty to use, λ, can be found using cross-validation and applied to our model.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(x){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+x))
  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat)/(n()+x))
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

Plot that shows a range of lambdas VS RMSE. The optimal setting provides the lowest error.

```
qplot(lambdas, rmses, color = I("blue"))
```




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

```
## [1] 5
```

RMSE result:

```
results <- bind_rows(results, tibble(Method = "Model 4: Regularized movie and user effects", RMSE = min
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.0600537
Model 2: Mean + movie bias	0.9429615
Model 3: Mean + movie bias + user effect	0.8646843
Model 4: Regularized movie and user effects	0.8641362

The RMSE has improved however it is a very small gain in accuracy. It will take a different approach to improve it significantly.

Model 5

Recommender systems use historical data to make predictions. One of the most popular approaches in achieving this is collaborative filtering. It is based on historical behavior by its users. So far we have approached a dataset that features sparsity and biases with models that account for these effects with decent accuracy. To get better results we turn to a more advanced method called matrix factorization. Our user data is processed as a large and sparse matrix, then decomposed into two smaller dimensional matrices with latent features and less sparsity. To make the process more efficient the recosystem packgage will be used. For more information on it click here. We start by converting data into the recosystem format, find the best tuning parameters, train and finally test it.

```
library(recosystem)
set.seed(1, sample.kind="Rounding")
train_reco <- with(train_set, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_reco <- with(test_set, data_memory(user_index = userId, item_index = movieId, rating = rating))
r <- Reco()

para_reco <- r$tune(train_reco, opts = list(dim = c(20, 30),
                                            costp_l2 = c(0.01, 0.1),
                                            costq_l2 = c(0.01, 0.1),
                                            lrate = c(0.01, 0.1),
                                            nthread = 4,
                                            niter = 10))

r$train(train_reco, opts = c(para_reco$min, nthread = 4, niter = 30))
results_reco <- r$predict(test_reco, out_memory())
```

With the algorithm trained we now test it to see the resulting RMSE:

```
factorization_rmse <- RMSE(results_reco, test_set$rating)
results <- bind_rows(results, tibble(Method = "Model 5: Matrix factorization using ricosystem", RMSE = f
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.0600537
Model 2: Mean + movie bias	0.9429615
Model 3: Mean + movie bias + user effect	0.8646843
Model 4: Regularized movie and user effects	0.8641362
Model 5: Matrix factorization using ricosystem	0.7846547

The improvement is significant. We have our model.

Final Validation

Having found our model with the lowest RMSE using matrix factorization, the final step is to train it using the edx set and then test its accuracy on the validation set:

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

```
r <- Reco()

para_reco <- r$tune(edx_reco, opts = list(dim = c(20, 30),
                                          costp_l2 = c(0.01, 0.1),
                                          costq_l2 = c(0.01, 0.1),
                                          lrate = c(0.01, 0.1),
                                          nthread = 4,
                                          niter = 10))

r$train(edx_reco, opts = c(para_reco$min, nthread = 4, niter = 30))

final_reco <- r$predict(validation_reco, out_memory())
```

```
final_rmse <- RMSE(final_reco, validation$rating)
results <- bind_rows(results, tibble(Method = "Final validation: Matrix factorization using recosystem"
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.0600537
Model 2: Mean + movie bias	0.9429615
Model 3: Mean + movie bias + user effect	0.8646843
Model 4: Regularized movie and user effects	0.8641362
Model 5: Matrix factorization using ricosystem	0.7846547
Final validation: Matrix factorization using recosystem	0.7809950

Conclusion

Our final RMSE is 0.7811. Significantly below our target of 0.8649. We built and tested several models and achieved our best accuracy using matrix factorization which was simplified through the recosystem package.