Report: HarvardX, PH125.8x. Data Science: Machine Learning

Rumbidzai Sylviah Pamacheche

10/04/2019

1. Introduction

Recommendation systems are information filtering systems that seek to predict the rating a user would give an item. We can use a recommendation system in a variety of areas such as movies, music, books and products in general. Companies like Amazon that sell many products and require customers for ratings. Amazon is then able to collect large amounts of data which allows them to predict ratings for particular users on specific items. As susch highly rated items for a given user are then recommended to that user.

Netflix an online streaming service, uses a recommendation system to predict the number of stars a user will give a specific movie. One star suggests that it is a bad movie, whereas five stars suggest it is an excellent movie. In 2006, Netflix offered a challenge to the data science community. They asked the community to improve their recommendation algorithm by 10%, and promised to award the winner a million dollar.

In accordance to the HarvardX: PH125.9x Data Science: Capstone, the goal of this report is to demonstrate that as a data science student is able to utilise skills in R programming language to solve real world problems. The task is to analyse a data set called ‘Movie lens’ with over 20 million ratings for over 27,000 movies by more than 138,000 users.

The key steps for this analysis will be to: 1). generate test and validation sets on MovieLens, 2). to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set, 3). test the algorithms, 4).and use the RMSE to evaluate how close our predictions are to the true values in the validation set.

2. Methods

\(We will first create our Test and Validation Sets\)

```r
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")

# 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 <- read.table(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")
head(movielens)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

The above movie lens table is in tidy format and contains thousands of rows, where each row represents a rating given by one user to one movie.

Validation set will be 10% of Movie Lens data

```
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)
```

Under this section we will inspect the *training* and *test* data sets, build the recommendation system, calculate the residual mean squared error (RMSE) for each model and choose the best RMSE for predicting the rating system in the validation data set.

We can inspect the edx data set with some functions and plots

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4   Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6  Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```
dim(edx)
```

```
## [1] 9000055       6
```

edx has 9000055 lines and 6 columns, which represents 90% of the *movielens* 10M subset.

```
summary(edx)
```

```
##      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              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Ratings have a minimum 0.5, mean 4 and maximum of 5 ratings.

The code below shows the number of unique users that provide ratings and for how many unique movies they provided.

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

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

The data set has 9.000.063 rows, but if we multiply those two numbers, we get 746.087.406 movies. This suggests that not every user rated every movie. So we can think of this data as a very large matrix with users on the rows and movies on the columns with many empty cells.

```
n_distinct(edx$movieId)* n_distinct(edx$userId)
```

```
## [1] 746087406
```

As shown below, *Pulp Fiction (1994)* has the greatest number of ratings of 31362, followed by *Forrest Gump (1994)* with 31079 ratings.

```
  edx %>% group_by(title) %>%
    summarize(count=n()) %>%
    top_n(5) %>%
    arrange(desc(count))
```

```
## # A tibble: 5 x 2
##   title                             count
##   <chr>                             <int>
## 1 Pulp Fiction (1994)               31362
## 2 Forrest Gump (1994)               31079
## 3 Silence of the Lambs, The (1991) 30382
## 4 Jurassic Park (1993)              29360
## 5 Shawshank Redemption, The (1994) 28015
```

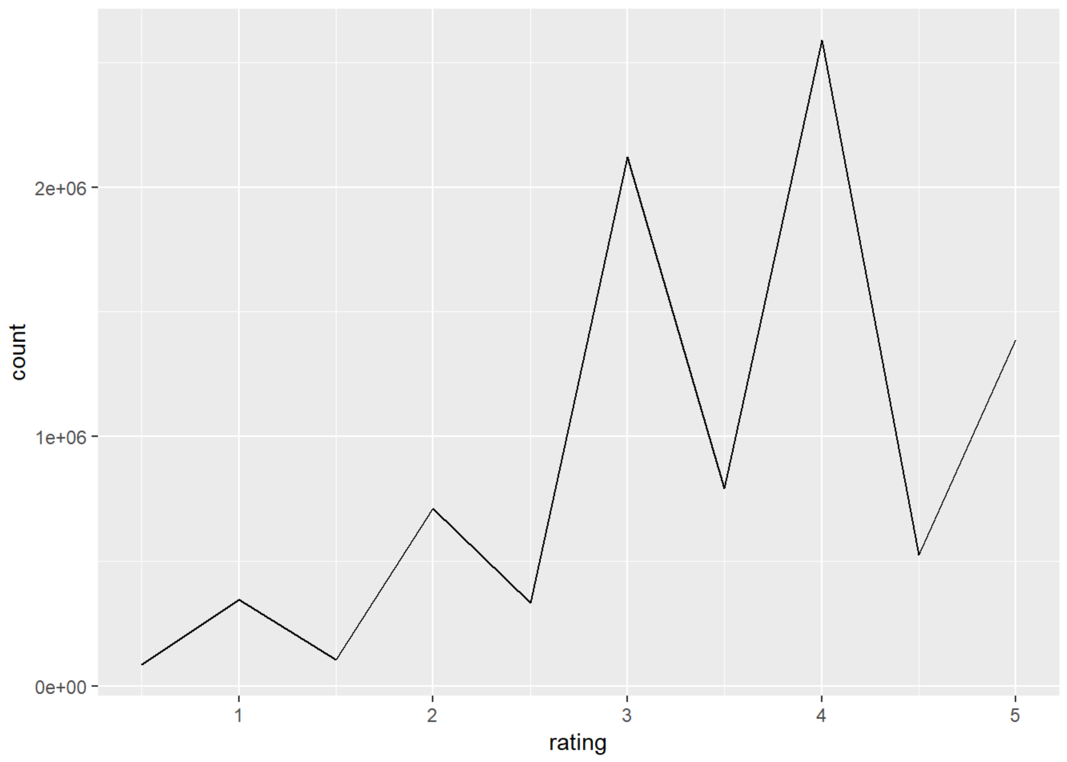
The five most given ratings in order from most to least are *4*, *3*, *5*, *3.5* and *2*, shown in the code below.

```
edx %>% group_by(rating) %>%
    summarize(count=n()) %>%
    top_n(5) %>%
    arrange(desc(count))
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

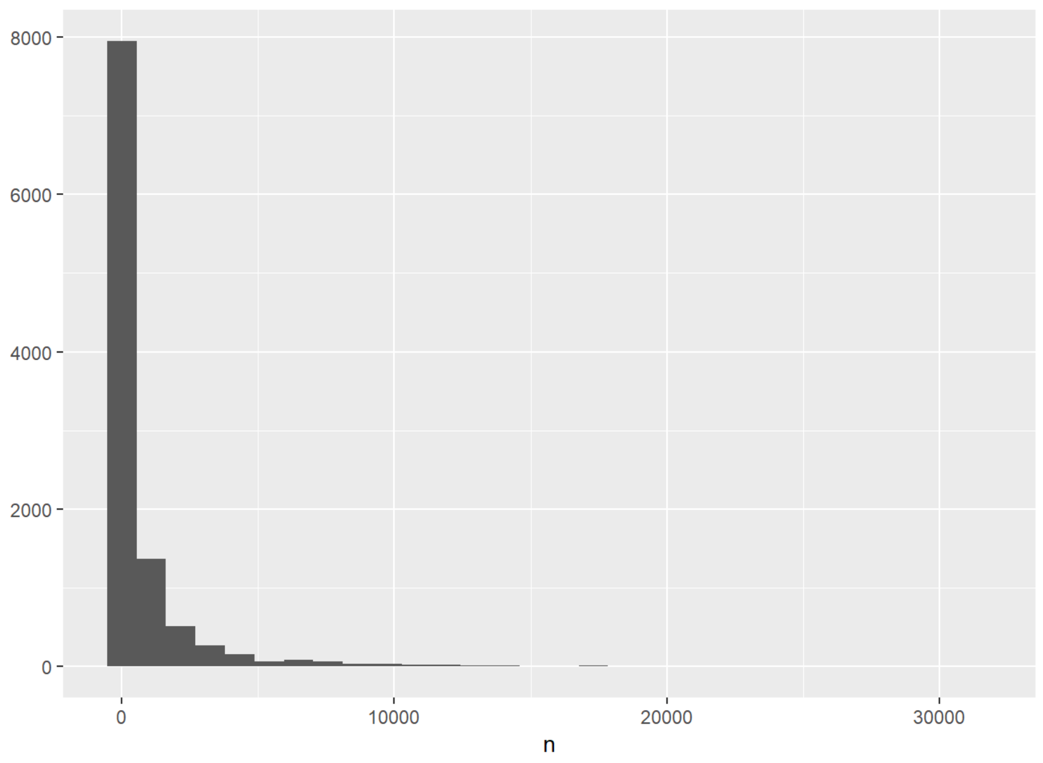
The plot shows that there are fewer half ratings than there are full ratings i.e. there are fewer 3.5 than there are ratings of 3

```
edx %>% group_by(rating) %>%
    summarize(count=n()) %>%
    ggplot(aes(x=rating, y=count))+
    geom_line()
```



If we look at some of the general properties of the data set , the first thing we notice is that some movies are rated more than others. Below is the distribution.

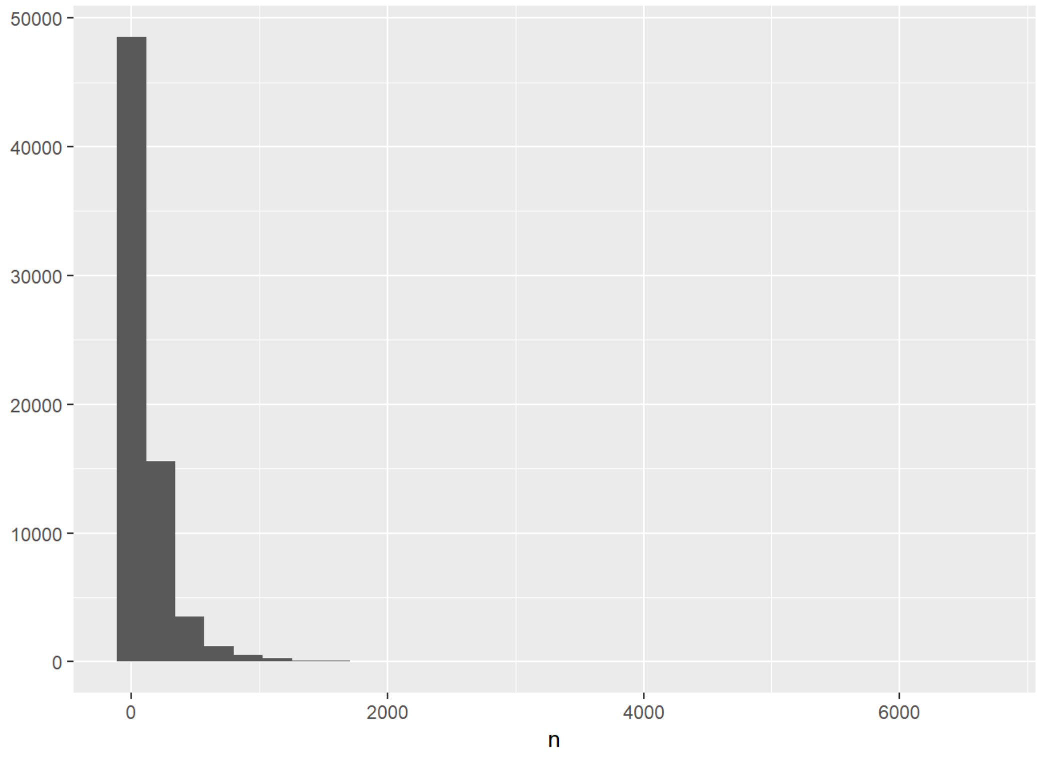
```
edx %>% group_by(movieId) %>%
  summarize(n=n()) %>%
  qplot(n, data= ., geom="histogram")
```



This is not surprising considering that there are blockbuster movies that are watched by millions and independent movies watched by a few.

A second observation is that some users are more active than others at rating movies.

```
edx %>% group_by(userId) %>%
  summarize(n=n()) %>%
  qplot(n, data= ., geom="histogram")
```



Model

We will now compare different models to see how well we're doing compared to a baseline model, we will need to use a loss function to quantify what it means to do well. The Netflix challenge used the typical error and decided on a winner based on the residual mean squared error on a test set.

We will define yui: as the rating for movie i by user u and y hat ui as our prediction, then the residual mean squared error is defined as follows.

\[RMSE = \sqrt{(\frac{1}{N} \sum_{u,i}(\hat{Y}_{u,i} - Y_{u,i})^2}\]

Here n is a number of user movie combinations and the sum is occurring over all these combinations. We can interpret the residual mean squared error similar to standard deviation.

If this number is much larger than one, we're typically missing by one or more stars rating which is not very good. So below we write a function that computes this 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))  }
```

If RMSE>1,it is not a good prediction. The smaller the RMSE, the better the prediction.

Building the Recommendation System

The winners of the Netflix challenge used two general classes of models.One model was similar to k-nearest neighbors, where they found similar movies and users.The second model was based on an approach called matrix factorization, which we will focus on.

We can start by building the simplest recommendation system. We're going to predict the same rating for all movies, regardless of the user and movie. We will use a model-based approach. So a model that assumes the same rating for all movies and all users, with all the differences explained by random variation would look something like this.

\[\ Y_{u,i} = u + \epsilon_{u,i} \]

mu represents the true rating for all movies and users. Epsilon represents independent errors sampled from the same distribution centered at zero

We know that the estimate that minimizes the residual mean squared error is the least squared estimate of mu.In this case, that's the average of all the ratings. We compute it like this:

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

```
## [1] 3.512465
```

3.51 is the average rating of all movies across all users. So now we can see how well this movie does. We will first compute this average on the training data, and then compute the residual mean squared error on the test set data.

```
naive_rmse<-RMSE(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

So when we predict all unknown ratings with this average. We get a residual mean squared error of about 1.05.That is big. If we plug in any other number, we will get a higher RMSE. That's what's supposed to happen, because we know that the average minimizes the residual mean squared error when using this model. This is the shown in the code below.

```
predictions <- rep(2.5, nrow(validation))
RMSE(validation$rating, predictions)
```

```
## [1] 1.46641
```

So as we go along we will be comparing different approaches. We will store the results in a table called rmse_results.

```
rmse_results <- data_frame(method="Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

The rmse result for 'Just the average' is 1.06. And we know from experience that some movies are just generally rated higher than others. So we can adjust our previous model by adding a term, b i, to represent the average rating for movie i. These b's are called effects.

\[\ Y_{u,i} = u + b_i + \epsilon_{u,i} \]

However, in this particular situation, we know that the least squared estimate, b hat i, is just the average of yui minus the overall mean for each movie, i. So we can compute them using this code:

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

movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i=mean(rating -  mu))
```

We can see that our residual mean squared error has dropped a little bit.

```
predicted_ratings <- mu + validation %>%
    left_join(movie_avgs, by='movieId')%>%
    .$b_i

model_rmse1 <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                           data_frame(method="Movie Effect Model",
                                      RMSE = model_rmse1))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087

We can see an improvement. In order to make it better and taken into account differences between users and how they rating movies. We can include a term, bu, which is the user-specific effect.

\[\ Y_{u,i} = u + b_i + b_u + \epsilon_{u,i} \]

We can compute our approximation by computing the overall mean (u-hat), the movie effects (b-hat i), and then estimate the user effects (b u-hat), by taking the average of the residuals obtained after removing the overall mean and the movie effect from the ratings yui.

```
user_avgs <- validation %>%
    left_join(movie_avgs, by='movieId')%>%
    group_by(userId) %>%
    summarize(b_u=mean(rating - mu - b_i))
```

We can now see how “well” this new model does by predicting values and computing the residual mean squared error.

```
predicted_ratings <- validation %>%
    left_join(movie_avgs, by='movieId')%>%
    left_join(user_avgs, by='userId')%>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

model_rmse2 <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                            data_frame(method="Movie + User Effects Model",
                                       RMSE = model_rmse2))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8292477

We can see that we have obtained a further improvement. Our residual mean squared error dropped down to about 0.82.

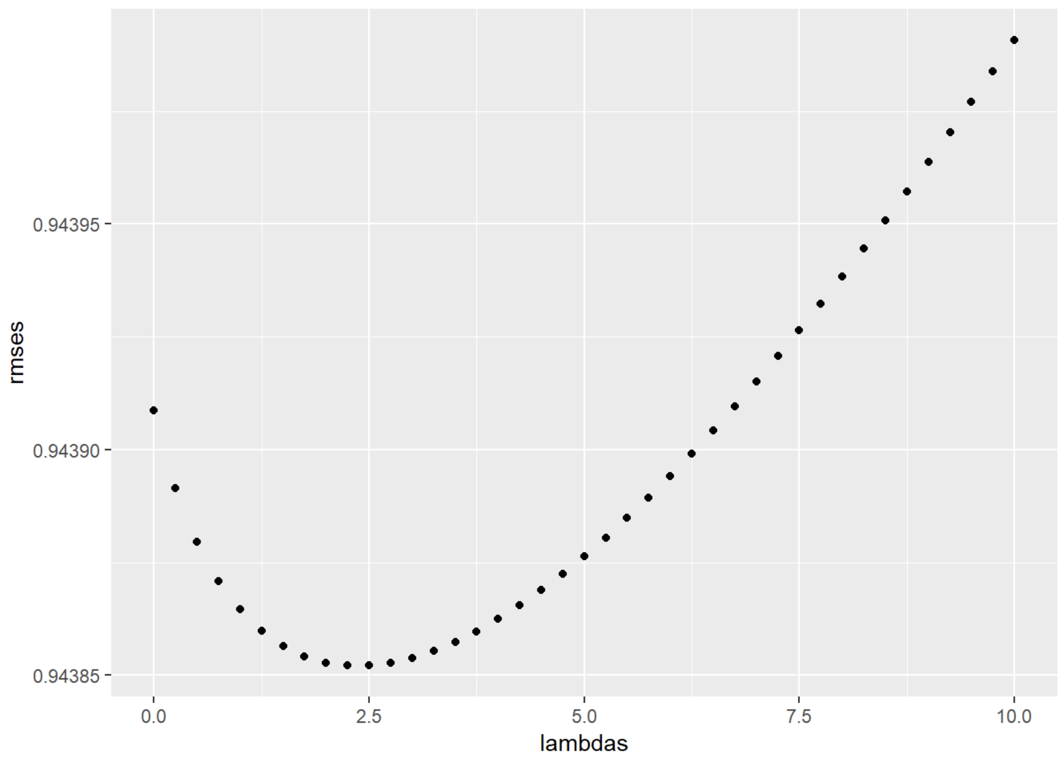
We can also use regularization, which allows us to penalize large estimates that come from small sample sizes. It has similarities to the Bayesian approaches which shrink predictions. The general idea is to add a penalty for large values of b to the sum of squares equations that we minimize. Take note that lambda is a tuning parameter.

```
  lambda <- 3
  mu <- mean(edx$rating)
  movie_reg_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_i=sum(rating-mu)/(n()+lambda), n_i = n())

  predicted_ratings <- validation %>%
    left_join(movie_reg_avgs, by='movieId') %>%
    mutate(pred=mu + b_i) %>%
    .$pred

  model_rmse3 <- RMSE(predicted_ratings, validation$rating)
  rmse_results <- bind_rows(rmse_results,
                            data_frame(method="Regularized Movie Effect Model",
                                       RMSE = model_rmse3))

  rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8292477
Regularized Movie Effect Model	0.9438538

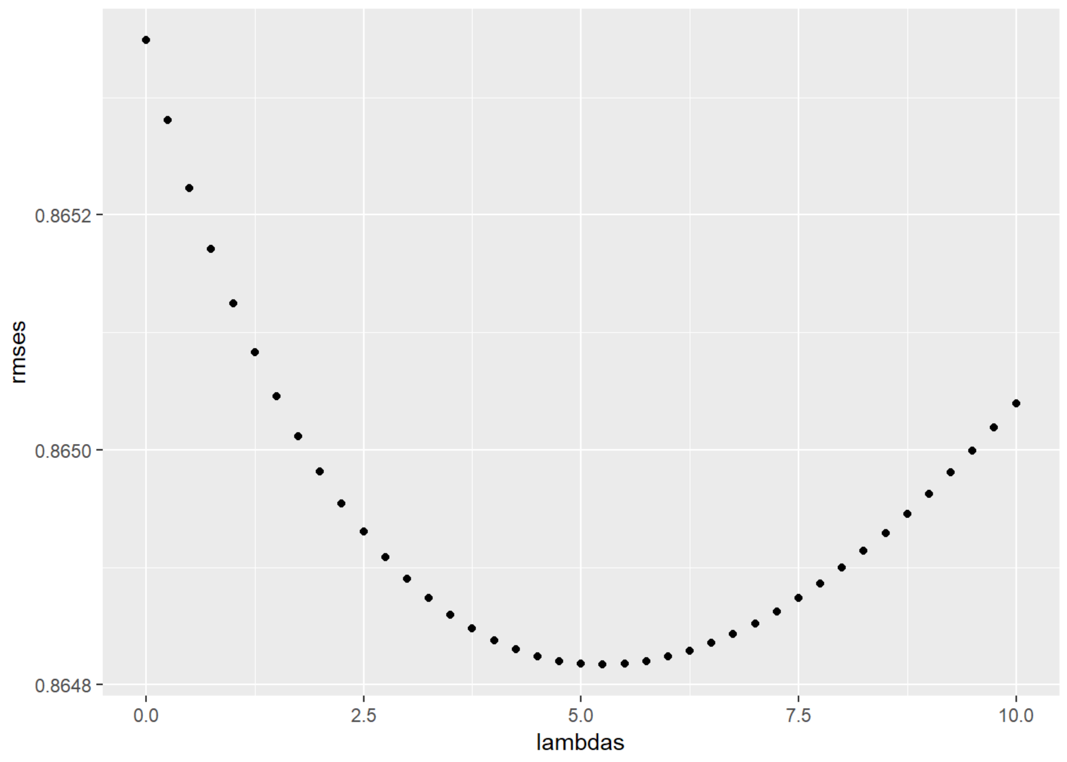
Since lambda is our tuning parameter, we can use cross-fertilization to choose it. We do this in the code below:

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(edx$rating)
just_the_sum <-edx %>%
    group_by(movieId) %>%
    summarize(s=sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
    predicted_ratings <- validation %>%
      left_join(just_the_sum, by='movieId') %>%
      mutate(b_i = s/(n_i+l)) %>%
      mutate(pred = mu + b_i) %>%
      .$pred
    return(RMSE(predicted_ratings, validation$rating))
  })

qplot(lambdas,rmses)
```



The minimized estimates can be found by using cross-validation to pick lambda, again. The code below does this, and we see what lambda minimizes our equation.

```
 lambdas <- seq(0, 10, 0.25)
  rmses <- sapply(lambdas, function(l){
    mu <- mean(edx$rating)

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

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

    predicted_ratings <- validation %>%
      left_join(b_i, by='movieId') %>%
      left_join(b_u, by='userId') %>%
      mutate(pred = mu + b_i + b_u) %>%
      .$pred

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

  qplot(lambdas,rmses)
```



It seems the full model including movie and user effects, the optimal lambda is 5.25, determined below.

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

```
## [1] 5.25
```

3. Results

Now that we know which is the best \(y\) for the model, we can calculate the *Regularized Movie Effect Model - adjusted*.

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

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))

predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

model_rmse_adjusted <- RMSE(predicted_ratings, validation$rating)
model_rmse_adjusted
```

```
## [1] 0.864817
```

The RMSE for the *Regularized Movie Effect Model - adjusted* is **0.86**

4. Conclusion

The variables in this data set have sufficient predictive power to allow us to predict how a user will rate a movie. For this report, we were able to create a movie recommendation system about movies for specific users for an online streaming service. We were able to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. Through trial of some machine learning techniques, we were able to obtain an acceptable RSME equal to 0.86.