A Regression Analysis for Prediction of Movie Ratings

Nirupa Gadi

June/2020

Purpose

The purpose of this study is to develop a recommendation model that can predict ratings for movies using the well-known MovieLens dataset. A successful model is defined as one that can achieve a Root Mean Square Error (RMSE) below 0.8572.

Background

How does Instagram know what posts to load on the discover page? How do Youtube and Tiktok know what videos to suggest? How does Amazon select products “inspired by your shopping trends”?

Recommendation systems are a subset of machine learning algorithms that can propose relevant suggestions to users. This relevancy is determined by the encoded algorithm which is based on past data of user ratings. Products which are predicted to be highly rated for a user will then be suggested to them for purchase. It is paramount that these algorithms are accurate to a wide range of users, as recommendation systems determine how likely a consumer is to utilise a product.

MovieLens is a dataset of movie ratings developed by GroupLens Research at the University of Minnesota, USA. This dataset contains several millions of 5-star-based ratings across a variety of genres and users. This dataset will be partitioned into training and validation subsets. The training set will be utilised to develop a recommendation algorithm. The validation set will be used afterwards to evaluate the accuracy of this system.

The accuracy of the model will be assessed via computation of the Root Mean Square Error (RMSE) which here in **Eq. 1** is defined as:

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

where $\hat{y}$ is the algorithm-predicted rating for movie i and y is the rating given by the user for that movie in all consumer and movie combinations N.

The Netflix prize is an open competition for the top collaborative filtering algorithm that can improve Netflix’s recommendation system by 10%. In order to win a million-dollar grand prize, a team must achieve an RMSE of 0.8572. In this analysis, scoring an error below this threshold will be considered a “success.”

Exploratory Data Analysis

Setup

Below is information to detail the operating system.

```
version
```

```
##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.0
## year           2020
## month          04
## day            24
## svn rev        78286
## language       R
## version.string R version 4.0.0 (2020-04-24)
## nickname       Arbor Day
```

The code to install the MovieLens dataset was provided by the PH125.9x teaching fellows. The code was modified to fit R 4.0.0.

```
##############################
# Create edx set, validation 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")

# 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")
 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")
```

Now, the provided data set will be partitioned in a 9:1 ratio to create the training and validation data sets, respectively. The partition in this analysis will not be randomised, but completed using a seed. The training set will be called `edx` and the test set for validation will be called `temp` .

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
 edx <- movielens[-test_index,]
 temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
 edx <- rbind(edx, removed)

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

Data Manipulation

The basic structure of the training and test data sets should first be studied.

```
str(edx)
```

```
## '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 838983
707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Advent
ure|Sci-Fi" ...
```

```
str(validation)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200 8444
16936 844417070 ...
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)" ...
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Romance|Wa
r" ...
```

It seems that both datasets are in tidy format. The `edx` dataset has 9000055 observations, while the test set `validation` has 999999 observations. Both subsets are of the same six variables, which are described below.

`userId` is of the integer class and represents each user who rated atleast one movie. `movieId` is of the numeric class and represents each movie that has atleast one rating. `rating` is of the numeric class. It is a number between 0.5 and 5 stars with 0.5 increments that is decided by each user for each movie that is rated. This is the variable the algorithm is interested in predicting. `timestamp` is of the integer class that signifies the exact time a review was made. `title` is of the character class and contains the name and year of release for each movie. `genres` is of the character class and contains one or several genres that the movie is classified into.

Inspection of both datasets reveals that the `timestamp` and `title` columns are problematic. `timestamp` is in a strange format based on the number of seconds since 1970, and `title` actually has two columns (movie title and release year) within itself. These columns should be modified in order to proceed.

```
# Modify the timestamp column

library(lubridate)
edx$timestamp <- as_datetime(edx$timestamp)
validation$timestamp <- as_datetime(validation$timestamp)

# Modify the title column

edx<- edx %>% separate(title, c("name", "year"), sep="\\s*\\((?=\\d+\\)$)|\\)$", convert=TRUE)
validation<- validation %>% separate(title, c("name", "year"), sep="\\s*\\((?=\\d+\\)$)|\\)$", convert=TRUE)
```

Additionally, the `genres` column could also be amended, as there are currently many genres listed for each movie. However, due to the large size of the dataset, replicating each observation for each genre that it contains may be too time consuming.

If the data set is looked at again,

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  7 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: POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
##  $ name     : chr  "Boomerang" "Net, The" "Outbreak" "Stargate" ...
##  $ year     : int  1992 1995 1995 1994 1994 1994 1994 1994 1994 1994 ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Advent
ure|Sci-Fi" ...
```

Now, `timestamp` is of the class `"POSIXct" "POSIXt"`, and `title` has been replaced with the character `name` and integer `year` categories.

Missing values are quite common in real-life datasets, so it is imperative to inspect the sets for this.

```
sum(is.na(c(edx,validation)))
```

```
## [1] 0
```

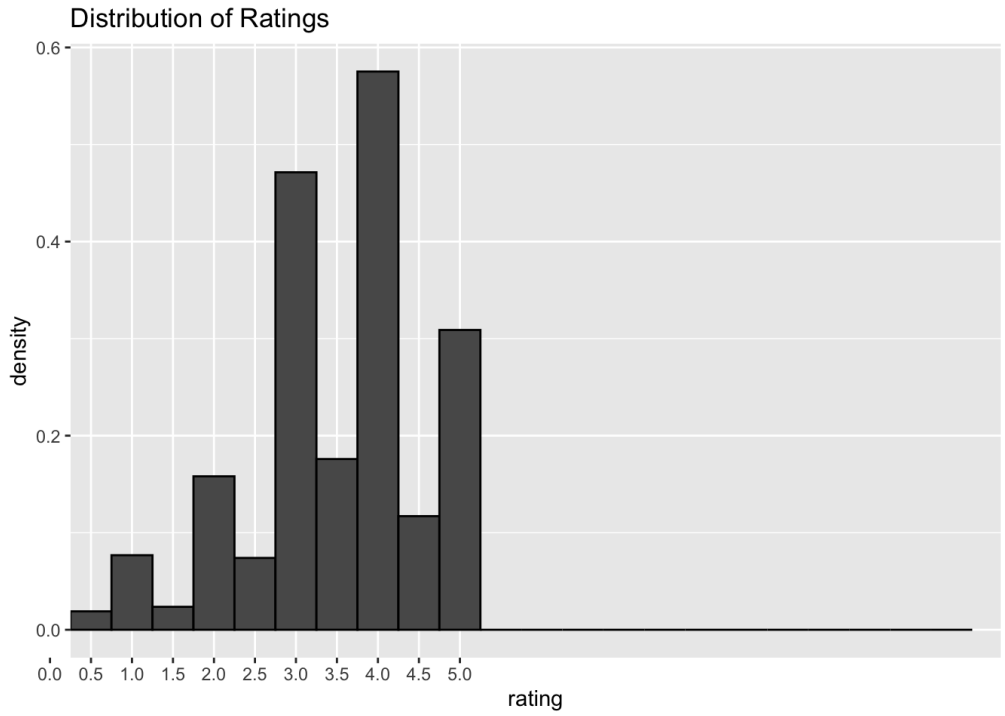
Luckily, there are no missing values in either set!

Visualisation

It is vital to understand trends in the training data in order to make accurate theories for modelling. These can be well-understood via data visualisation techniques.

First, it may be useful to know the distribution of rates in the `edx` dataset. Later, this will help develop a mathematical model for recommendation that is based on randomness. Here in **Fig. 1**,

```
edx %>% ggplot(aes(x=rating,y=..density..)) + geom_histogram(binwidth = 0.5, colour = "black") + scale_x_dis
crete(limits = c(seq(0,5,0.5))) + ggtitle("Distribution of Ratings")
```



From the histogram, one can see that users are more likely to select integer ratings rather than non-integer ratings. With this in mind, it appears that the mean rating for a movie is 3.5 stars with a mode of 4 stars, and that this distribution is skewed left. This means that consumers are more likely to rate a movie as having been “good” rather than “bad”.

On further analysis, it appears that some movies are watched and therefore rated much more often than others.

```
edx %>% count(name)%>%arrange(desc(n))%>%head(6)
```

```
## # A tibble: 6 x 2
##   name                          n
##   <chr>                     <int>
## 1 Pulp Fiction              31362
## 2 Forrest Gump              31079
## 3 Silence of the Lambs, The 30382
## 4 Jurassic Park             29360
## 5 Shawshank Redemption, The 28015
## 6 Braveheart                26212
```

```
edx %>% count(name)%>%arrange(-desc(n))%>%head(6)
```

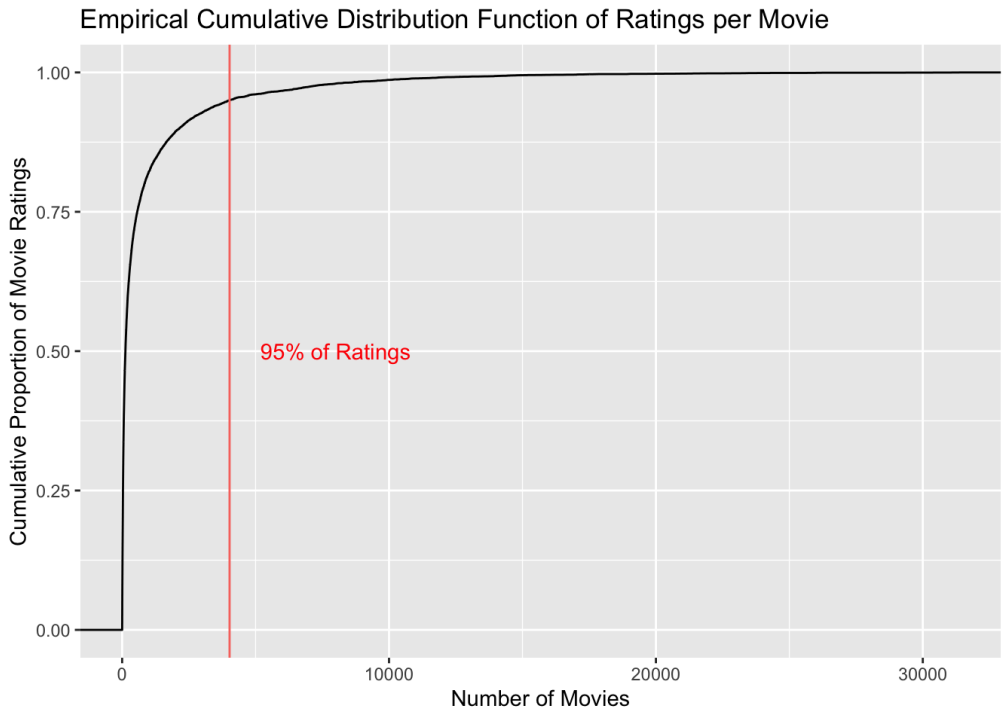
```
## # A tibble: 6 x 2
##   name                                    n
##   <chr>                               <int>
## 1 1, 2, 3, Sun (Un, deuz, trois, soleil)     1
## 2 100 Feet                                   1
## 3 4                                          1
## 4 Accused (Anklaget)                         1
## 5 Ace of Hearts                              1
## 6 Ace of Hearts, The                         1
```

In graphical form for **Fig. 2**,

```
edx %>% count(movieId) %>% ggplot(aes(n))+ geom_density(fill = 'black',colour = 'black')+ ggtitle('Distribut
ion of ratings per movie')+ labs(x = "Ratings per movie", y = "Density") + scale_x_log10() + geom_vline(aes(
xintercept=mean(n)), colour="red", linetype="solid", size=1) + geom_vline(aes(xintercept=median(n)),color="b
lue", linetype="solid", size=1)+annotate("text", x = 1500, y = 0.12, label = "Mean", color="Red") + annotat
e("text", x = 50, y = 0.12, label = "Median", colour="Blue")
```

An empirical cumulative distribution function is useful in this case, as it draws light to how different the rating counts are for some movies compared to other better than a density curve. Described is **Fig. 3**,

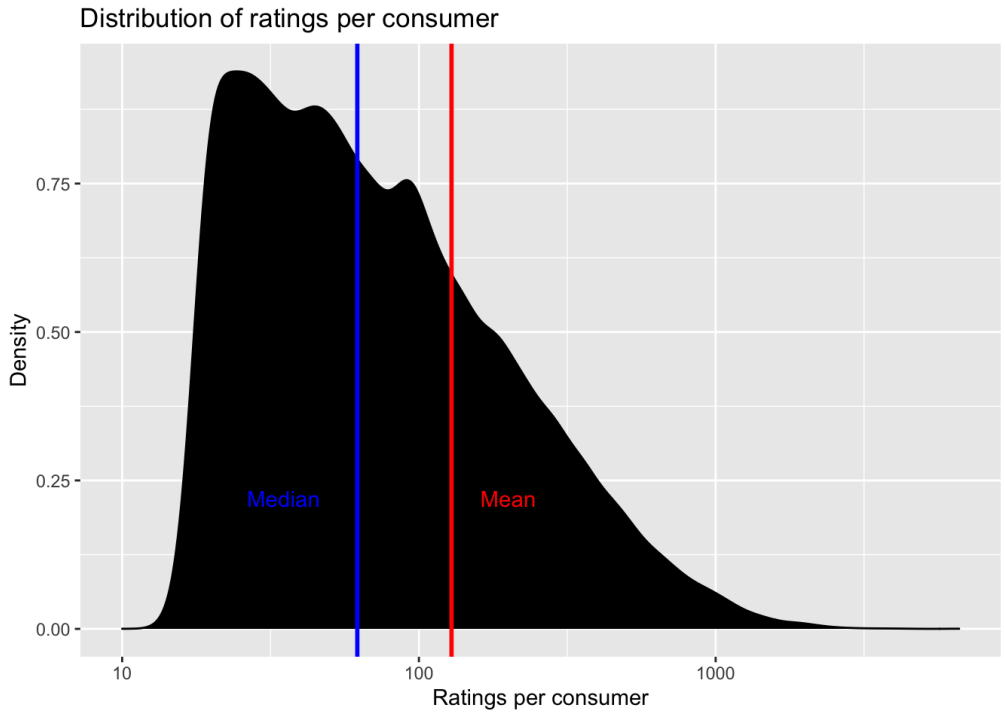
```
movie_total_counts <- edx %>% group_by(movieId) %>% summarise(total_ratings=n())
quantiles_1 <- quantile(movie_total_counts$total_ratings, probs=0.95)
movie_total_counts %>% ggplot(aes(total_ratings)) + stat_ecdf(geom = "step") + labs(title = "Empirical Cumul
ative Distribution Function of Ratings per Movie",x = "Number of Movies",y = "Cumulative Proportion of Movie
Ratings") + geom_vline(aes(xintercept=quantiles_1, colour = "red",linetype="solid")) + annotate("text", x =
8000, y = 0.50, label = "95% of Ratings", colour="red") + theme(legend.position = "none")
```



This graph is incredibly powerful, as it shows that `quantiles_1` movies contain 95% of the ratings in the `edx` dataset. This leaves the leftover `movie_total_counts - quantiles_1` movies to occupy only 5% of the ratings. In the remainder, there are actually hundreds of movies that have only one reviewer. In a simple recommendation system, this one rating is given excess “weight,” resulting in poor prediction of how a consumer would actually rate the movie, leading to inaccurate recommendations.

This bias in how often a movie is rated also exists in terms of the consumer. To elucidate, some consumers rate movies much more often than others. In a similar code to the one before, **Fig. 4** describes:

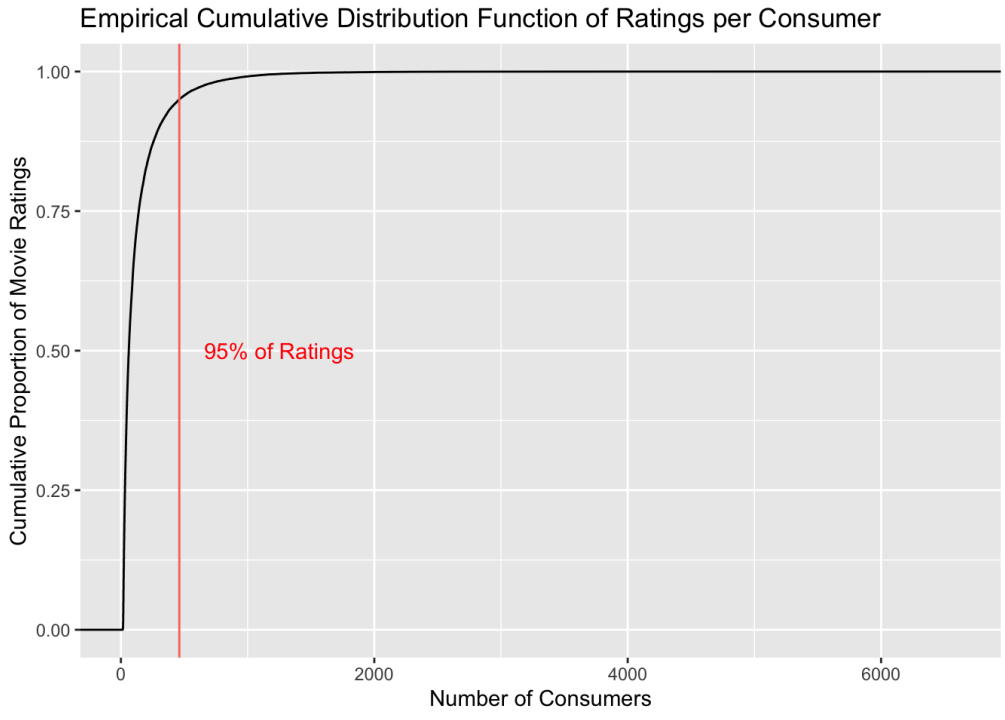
```
edx %>% count(userId) %>% ggplot(aes(n))+ geom_density(fill = 'black', colour = 'black')+ ggtitle('Distribution of ratings per consumer')+ labs(x = "Ratings per consumer", y = "Density") + scale_x_log10() + geom_vline(aes(xintercept=mean(n)),colour="red", linetype="solid", size=1) + geom_vline(aes(xintercept=median(n)),colour="blue", linetype="solid", size=1) + annotate("text", x = 200, y = 0.22, label = "Mean", color="Red") + annotate("text", x = 35, y = 0.22, label = "Median", colour="Blue")
```



Based on the fact that the mean is larger than the median, it is true that the density graph is skewed right, showing that few reviewers are very active on the rating platform. In fact, the most active consumer for rating has reviewed over 6,000 movies!

An ECDF for ratings per consumer can also be computed for **Fig. 5**.

```
user_total_counts <- edx %>% group_by(userId) %>% summarise(total_ratings=n())
quantiles_2 <- quantile(user_total_counts$total_ratings, probs=0.95)
user_total_counts %>% ggplot(aes(total_ratings)) + stat_ecdf(geom = "step") + labs(title = "Empirical Cumulative Distribution Function of Ratings per Consumer", x = "Number of Consumers", y = "Cumulative Proportion of Movie Ratings") + geom_vline(aes(xintercept=quantiles_2, colour = "red",linetype="solid")) + annotate("text", x = 1250, y = 0.50, label = "95% of Ratings", colour="red") + theme(legend.position = "none")
```



An ECDF is powerful here as well, showing that `quantiles_2` users complete 95% of the reviews. The remaining `user_total_counts-quantiles_2` users make up the remaining 5% of reviews.

Modelling

Different modelling approaches will be tested in order to estimate the lowest Root Mean Square Error, which is shown in Eq. 1. The RMSE is analogous to a standard deviation, as it calculates the square residual, or the positive absolute error in the number of stars difference between the predicted and actual values. The goal is to attempt modelling combinations to reach an error as low as RMSE = 0.8572, the amount needed to win the grand Netflix prize.

First, the RMSE needs to be redefined into a function format that R can understand.

```
RMSE <- function(actual_ratings, predicted_ratings){
     sqrt(mean((actual_ratings - predicted_ratings)^2, na.rm=TRUE))}
```

Next, the `edx` dataset needs to be split into training and test sets, as was suggested by the PH125.9x TFs. The `validation` set is to be used for the final, optimised regression. The expected value will be ascertained from the training set, and the RMSE will be predicted by using the test set. This will avoid overfitting the model to the training set. The 50-50 split between training and testing sets was decided for a balance between parameter estimates and performance statistics.

```
test_index <- createDataPartition(edx$rating, times = 1, p = 0.5, list = FALSE)
edx_test<- edx[test_index,]
edx_train<- edx[-test_index,]
```

Baseline Naïve Model

This simple model will form a baseline from where the linear regression can be enhanced to become more accurate. This baseline is constructed on randomised recommendations off the training set expected value, regardless of any details of the media and consumer. Here is **Eq. 2**:

$$Y_{y,i} = \hat{\mu} + \varepsilon_{y,i}$$

`\hat{\mu}` is the expected value of a rating, as described previously in Figure 1. Using `\hat{\mu}` for the initial prediction of the regression is a wise choice, as the least squares estimate of this variable will minimize the RMSE of the test set compared to a random variable. _{y,i}$$ is a variable that describes the random, independent errors with mean 0 that are within this same distribution.

```
mu_hat_train<- mean(edx_train$rating)
rmse_1 <- RMSE(edx_test$rating, mu_hat_train)
reg_1 <- data.frame(Approach = "Naive Baseline from Expected Value", RMSE = rmse_1)
reg_1 %>% knitr::kable()
```

Approach	RMSE
Naive Baseline from Expected Value	1.060415

An RMSE of `rmse_1` is decent, but this regression can be improved.

Movie-Specific Effect

From the density curve in Fig. 2 and ECDF seen in Fig. 3, it can be ascertained that `movieID` should be a factor in the regression. Some movies are higher quality than others and tend to get high ratings across the board. Therefore, there is a movieId-specific effect that should be represented as a bias, as this will likely lower the RMSE.

Y_{y, i} = \hat{μ} +b_{i}+ \varepsilon_{y, i}

Here in **Eq. 3**, all components are kept the same as in Eq. 2, but an extra term `b_{i}` is used to account for this movie-specific bias. Technically, `b_{i}` is the average of `Y_{y, i}-\hat{μ}` for each movie that is rated.

```
actual_movie_ratings <- edx_train %>% group_by(movieId) %>% summarise(b_i = mean(rating - mu_hat_train))

predicted_movie_ratings<-  mu_hat_train + edx_test %>% left_join(actual_movie_ratings, by= "movieId", na.rm=
TRUE) %>% .$b_i

predicted_movie_ratings<- as.numeric(predicted_movie_ratings)

rmse_2 <- RMSE(edx_test$rating, predicted_movie_ratings)

reg_2 <- data.frame(Approach = "Regression Model with Movie Effect",RMSE = rmse_2)

reg_2 %>% knitr::kable()
```

Approach	RMSE
Regression Model with Movie Effect	0.9441335

`rmse_2` is an improved RMSE to `rmse_1` of the naïve regression. This demonstrates that considering important variables such as movie bias can improve the accuracy of the recommendation system.

Consumer-Specific Effect

Analogous to the movie-specific effects, from the density curve in Fig. 4 and the ECDF seen in Fig. 4, `userId` should also be a factor in the linear regression. Some users tend to rate movies consistently higher than other users. Therefore, there is a userId-specific effect that should be represented as a bias, as this will likely further lower the RMSE.

$$Y_{y, i} = \hat{μ} +b_{i}+ b_{y}+ \varepsilon_{y, i}$$

Here in **Eq. 4**, all components are kept the same as in Eq. 3, but an extra term `b_{y}` is used to account for this consumer-specific bias. `b_{i}`.

```
actual_user_ratings <- edx_train %>% left_join(actual_movie_ratings, by='movieId') %>%
group_by(userId) %>% summarise(b_y = mean(rating - mu_hat_train - b_i))

predicted_user_ratings<-  edx_test %>% left_join(actual_movie_ratings, by= "movieId") %>% left_join(actual_u
ser_ratings, by= "userId") %>% mutate(pred = mu_hat_train + b_i + b_y) %>% .$pred

predicted_user_ratings<- as.numeric(predicted_user_ratings)

rmse_3 <- RMSE(edx_test$rating, predicted_user_ratings)

reg_3 <- data.frame(Approach = "Regression Model with Movie and User Effects", RMSE = rmse_3)

reg_3 %>% knitr::kable()
```

Approach	RMSE
Regression Model with Movie and User Effects	0.8696665

Factoring in the consumer-specific effect along with the previous movie-specific effect produced an RMSE of `rmse_3`! This is very close to our original goal of 0.8572. Is it possible to get an even lower RMSE?

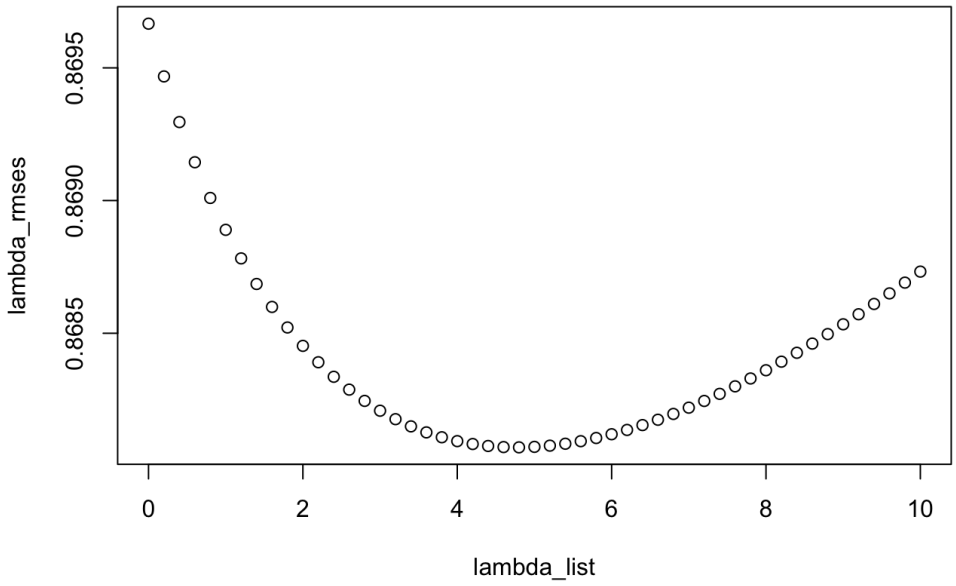
Regularised model

Biases in terms of movie quality or rating rigor by consumers have already been discussed and accounted for by `b_{i}` and `b_{y}`, respectively. However, another related bias has to do with the frequency of rating. Examples are described in the previous section. Some movies are more popular, and thus get rated more often than unpopular movies. To follow, some users are more active on the review platform than others. This creates an imbalance, as these small sample sizes of reviews are heavily weighted and create high errors. In this section, a method called regularisation will be used to attenuate these occurrences. First, a tuning parameter called lambda will be ascertained. This parameter will optimally minimise `b_{i}` and `b_{y}` in the case of small sample sizes. Accounting for this bias should further lower the RMSE.

```
lambda_list<- seq(0,10,0.2)
lambda_rmses <- sapply(lambda_list, function(l){

b_i <- edx_train %>%  group_by(movieId) %>% summarise(b_i = sum(rating - mu_hat_train)/(n()+l))

b_y <- edx_train %>% left_join(b_i, by="movieId") %>% group_by(userId) %>% summarise(b_y = sum(rating - b_i
- mu_hat_train)/(n()+l))

predicted_ratings <- edx_test %>% left_join(b_i, by = "movieId") %>% left_join(b_y, by = "userId") %>% mutat
e(pred = mu_hat_train + b_i + b_y) %>% .$pred

return(RMSE(edx_test$rating, predicted_ratings))

})
```

While it seems complicated, the coding for the lambda function is just an amalgamation of coding that was previously developed to account for the biases. These lambda values can be plotted to decide the optimal value.

```
lambda_plot<- plot(lambda_list, lambda_rmses)
```




```
lambda<- lambda_list[which.min(lambda_rmses)]
lambda
```

```
## [1] 4.8
```

It appears that the optimal value of the parameter lambda is `lambda`. Before and after this vertex, the RMSEs increase. This lambda value must be incorporated into the regression model with `b_{i}` and `b_{y}` in order to achieve a lower RMSE.

```
reg_movie_ratings <- edx_train %>% group_by(movieId) %>% summarise(b_i = sum(rating - mu_hat_train)/(n()+lam
bda), n_i = n())

user_avgs_reg <- edx_train %>% left_join(reg_movie_ratings, by="movieId") %>% group_by(userId) %>% summarise
(b_y = sum(rating - mu_hat_train - b_i)/(n()+lambda), n_y = n())

reg_user_ratings <- edx_train %>% left_join(reg_movie_ratings, by="movieId") %>% group_by(userId) %>% summar
ise(b_y = sum(rating - mu_hat_train - b_i)/(n()+lambda), n_y = n())

reg_predicted_ratings <- edx_test %>% left_join(reg_movie_ratings, by="movieId") %>% left_join(reg_user_rati
ngs, by="userId") %>% mutate(pred = mu_hat_train + b_i + b_y) %>%  .$pred

rmse_4 <- RMSE(edx_test$rating,reg_predicted_ratings)

reg_4 <- data_frame(Approach="Regression Model with Effects and Regularisation", RMSE = rmse_4 )

reg_4 %>% knitr::kable()
```

Approach	RMSE
Regression Model with Effects and Regularisation	0.86807

Although the improvement in `rmse_4` is slight when compared to `rmse_3`, the total value has still decreased. So far, the regression models have been tested on `edx_test`, but now the current regularised model will be tested on the validation set to analyse its consistency.

```
reg_movie_ratings_val <- edx %>% group_by(movieId) %>% summarise(b_i = sum(rating - mu_hat_train)/(n()+lambda), n_i = n())

user_avgs_reg_val <- edx %>% left_join(reg_movie_ratings_val, by="movieId") %>% group_by(userId) %>% summarise(b_y = sum(rating - mu_hat_train - b_i)/(n()+lambda), n_y = n())

reg_user_ratings_val <- edx %>% left_join(reg_movie_ratings_val, by="movieId") %>% group_by(userId) %>% summarise(b_y = sum(rating - mu_hat_train - b_i)/(n()+lambda), n_y = n())

reg_predicted_ratings_val <- validation %>% left_join(reg_movie_ratings_val, by="movieId") %>% left_join(reg_user_ratings_val, by="userId") %>% mutate(pred = mu_hat_train + b_i + b_y) %>%  .$pred

rmse_5 <- RMSE(validation$rating,reg_predicted_ratings_val)

reg_5 <- data_frame(Approach="Regression Model with Regularisation on Validation Set", RMSE = rmse_5 )

reg_5 %>% knitr::kable()
```

Approach	RMSE
Regression Model with Regularisation on Validation Set	0.8648195

The RMSE has decreased slightly to `rmse_5` when testing the algorithm on the validation set. This error value is not low enough to win the Netflix challenge, but it is still a satisfactory conclusion.

Discussion

The results of this study have important implications for media platforms, as being able to predict a consumer's reaction to a piece is incredibly powerful in the supply and demand chain. In this example, the media company would be able to better recommend movies to users, increasing their chances success in terms of subscribers and profit.

The final model for the regression is the same as Eq. 4:

$$Y_{y, i} = \hat{\mu} +b_{i}+ b_{y}+ \varepsilon_{y, i}$$

The RMSE results of the four completed models and five tests are shown below.

```
total_reg<- data.frame(c("Naive", "Movie Effect", "Movie and User Effects", "Regularisation", "Regularisation on Validation"), c(rmse_1, rmse_2, rmse_3, rmse_4, rmse_5))

colnames(total_reg)<-(c("Regression Approach","RMSE"))

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

Regression Approach	RMSE
Naive	1.0604153
Movie Effect	0.9441335
Movie and User Effects	0.8696665
Regularisation	0.8680700
Regularisation on Validation	0.8648195

It seems that with increasing accountability for biases in the reviews, the RMSE decreases as the actual and predicted ratings unify. Additionally, the RMSE was actually lower in the `validation` test set, which speaks to the accuracy of the model between training and test sets. The RMSE developed can be further improved by accounting for effects in movie genre, although this would have greatly increased the size of the dataset as well as computation time. Other factors to consider in optimisation of the regression model would be ew time, etc. Considering these factors may bring the RMSE low enough to actually win the Netflix challenge.