HarvardX Movielens Report

Rhys van den Handel

27/10/2020

Executive Summary

As part of the HarvardX data science certificate, a recommendation system for movies using GroupLens' Movielens 10M dataset was built. The project required an intense data analysis to be conducted. The results of the analysis allowed for a recommendation system to be accurately built. The target for the system was an RMSE of < 0.86490.

The system was built by separating the 10M dataset into a building set called edx and a validation set. The model was then created by working from simple mean value prediction systems through to more complex regularization. As expected the simple prediction systems resulted in poorer RMSE with the most complex ones getting closer to the target RMSE.

The quality of the data as well as skewing factors such as low numbers of ratings on movies and users rating few movies resulted in higher than expected RMSE values. Cleaner datasets revealed that the approaches used have the ability to accurately predict RMSE values lower than the target. The final result on the training data set however only yielded an RMSE of 0.8740286 for the training model. The project was also limited by the size of the dataset and computing power available.

When building the rating system with a data optimized process final RMSE of 0.9200898 was achieved. However, as this does not evaluate outlive in the dataset the final accepted RMSE must be 0.9345861

1. Introduction

The HarvardX data science certificate takes part over 9 Courses. This is the final Capstone project for all learners. The project is a recommendation system for movies based on the Movielens data. The purpose is to take user given ratings on different movies across many users and movies. The project will then give a prediction on the movie based on the title, and the users prior preference thus recommending a movie.

These systems are common in daily lives. Many streaming services use very similar systems to predict what a user would rate a movie as and therefore recommend high rated movies to the user to encourage more screen time and thus more exposure to the platform and potentially advertising.

This project looks at many movies and user ratings from the movielens dataset. Ratings will be scored on a scale of 1 to 5 and used to predict a score. This score will be compared against the known hidden score to determine the RMSE which is used to quantify the models performance.

1.1 Movielens

The Movielens data science project is has been developed by GroupLens,from the University of Minnesota. There are a few different datasets that have been developed. For this project we will be using the Movielens 10M dataset. Within the dataset we will be looking at 2 specific data sets Ratings.dat and Movies.dat. They are each comprised of the following:

Ratings.dat

- userId: a unique identifying code allocated to individual users
- movieId: a unique identifying code allocated to individual movies
- rating: The rating given by the user for a specific movie
- timestamp: Time of tagging in seconds since midnight UTC of January 1, 1970

Movies.dat

- movieId: a unique identifying code allocated to individual movies
- title: The title of the movie with (year) of production at the end of the title
- genres: All the relevant genres to the movie delimited by a |

1.2 Limitations

The size of the datasets means that certain operations cannot be used to run machine learning such as the caret train. The machine used was an i7 with 8gb of RAM however this was not sufficient to compute the train matrices which were over 100gb. This limits the analysis to more simple methods such as mean prediction, effect modeling and regularization. Given the computing power restrictions the methodology was decided upon as shown bellow.

1.3 Evaluation Method

"The RMSE is a quadratic scoring rule which measures the average magnitude of the error. The equation for the RMSE is given in both of the references. Expressing the formula in words, the difference between forecast and corresponding observed values are each squared and then averaged over the sample. Finally, the square root of the average is taken. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE is most useful when large errors are particularly undesirable." - http://www.eumetrain.org

As well put by eumetrain.org RMSE penalizes high errors. This makes it an ideal evaluation method for recommendation systems. A lower RMSE means that there is a lower likelihood of a very poor recommendation being made which could cause a user to not select the title or worse, leave the application.

2. Methodology

The methodology used to evaluate the project is shown and discussed bellow:

2.1 Data Import

This initial piece of the project was given and is therefore applicable to all. The purpose is to fetch the data from *http://files.grouplens.org/datasets/movielens/ml-10m.zip*. The data is then split into two separate datasets.

edx: The Edx dataset contains the 90% of the ratings.dat and movies.dat datasets. They are joined on movieid. The primary purpose of the edx dataset is to build and test the system. We will also be using the edx dataset to perform the data analysis. validation: The validation dataset contains 10% of the ratings.dat and movies.dat datasets joined on movieid. Validation has been semi-joined to ensure all the userid's and movieid's that are present in the validation set are represented in the edx dataset.

2.2 Data Analysis

The data analysis allows for a better understanding of the data components and their interactions. The process of running the data analysis was primarily summarizing into movie and user groups. The key analysis variables were the counts and the ratings. The following was investigated:

1. edx dataset

- view the data
- check for nulls

2. Movies (group by movieid)

- Summary statistics
- Most rated movies
- Best rated movies
- mean rating by year and number of movies

3. Users (group by userid)

- Summary statistics
- Users with most ratings
- Users with best ratings
- Users with worst ratings

4. Genres (group by genres)

- Summary statistics
- Genres with most ratings
- Genres with best ratings
- Genres with worst ratings

The purpose of the data analysis would be to allow for informed data cleaning decisions that would have a positive outcome on the final results. The data cleaning was used as part of the predictive model as discussed in section 2.3 bellow.

2.3 Predictive Model

Due to the limitations discussed in section 1.2 *Limitations* above the predictive model was built with a bottom up complexity method. Therefore the outcome would use the most simple method that would still give an acceptable result.

2.3.1 Splitting Data For each of the key analysis forms the data was split into a training and testing set. The partition is 25% this allows for a large proportion of testing data. If the model can be accurate using a small training proportion to the validation model it is more likely to yield good results at the validation stage.

2.3.2 Prediction Models As introduced the the model will be build from a simple to more complex model:

- Average Model:
  - Using the simple mean of the dataset
- Movie Effect:
  - Using the grouping of the movie to calculate the mean rating for each movie
- Movie Regularization:
  - Using a penalty term on the number of movies to weight the mean
  - Optimized for the best penalty term
- User Effect:
  - Using the grouping of the users to calculate the mean rating for each user
- User Regularization:
  - Using a penalty term on the number of users to weight the mean
  - Optimized for the best penalty term
- Genre Effect:
  - Using the grouping of the users to calculate the mean rating for each user
- Genre Regularization:
  - Using a penalty term on the number of genres to weight the mean
  - Optimized for the best penalty term
- Combined gene, user and movie regularization:
  - Using a penalty term on the number of ratings for each movie as well as the number of ratings for each user and genre to adjust the mean

2.3.3 Iterations The prediction models would be iterated over the default dataset. After the results of the default data have been shown optimization through data cleaning would occur. The order of testing will be:

- Default
- Limit by median

2.4 Validation

Once an acceptable RMSE has been identified the final iteration would be the validation, run through the final model.

3. Results and Discussion

This sections presents the results of the methodology.

3.1 Data Import and Preparation

As stated previously, this piece of code has been provided by the edx team.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################
# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

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

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
## 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,]

# 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") %>%
  semi_join(edx, by = "genres")
```

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

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

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

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

The edx and Validation datasets were created successfully.

3.2 Data Analasis

```
#View dataset
head(edx)
```

3.2.1 Edx dataset

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
#Check nulls
any(is.na(edx))
```

```
## [1] FALSE
```

The investigation proved that the columns are presented as expected in the format expected. It also concludes that there are no nulls in the edx dataset.

```
#Create Smaller data sets Average Movie
mean_movie <- edx %>% group_by(movieId, title, genres) %>%
  summarize(n=n(), mean_rating=mean(rating))%>%
  select(movieId, title, genres,n,mean_rating)
```

3.2.1 Movies

```
## `summarise()` regrouping output by 'movieId', 'title' (override with `.groups` argument)
```

```
#Count number of movies
nrow(mean_movie)
```

```
## [1] 10677
```

```
#movie stats
summary(mean_movie$n)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    30.0   122.0   842.9   565.0 31362.0
```

```
summary(mean_movie$mean_rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   2.844   3.268   3.192   3.609   5.000
```

As shown there are 10677 movies. The summaries have been produced on both the number of ratings per movie and the average rating of each movie.

```
#10 most rated movies
mean_movie %>% arrange(desc(n)) %>%
  top_n(10,n)
```

```
## # A tibble: 10,677 x 5
## # Groups:   movieId, title [10,677]
##    movieId title                               genres                n mean_rating
##      <dbl> <chr>                               <chr>             <int>       <dbl>
##  1     296 Pulp Fiction (1994)                 Comedy|Crime|Drama 31362        4.15
##  2     356 Forrest Gump (1994)                 Comedy|Drama|Roma~ 31079        4.01
##  3     593 Silence of the Lambs, The (1991)    Crime|Horror|Thri~ 30382        4.20
##  4     480 Jurassic Park (1993)                Action|Adventure|~ 29360        3.66
##  5     318 Shawshank Redemption, The (1994)    Drama              28015        4.46
##  6     110 Braveheart (1995)                   Action|Drama|War   26212        4.08
##  7     457 Fugitive, The (1993)                Thriller           25998        4.01
##  8     589 Terminator 2: Judgment Day (199~    Action|Sci-Fi      25984        3.93
##  9     260 Star Wars: Episode IV - A New H~    Action|Adventure|~ 25672        4.22
## 10     150 Apollo 13 (1995)                    Adventure|Drama    24284        3.89
## # ... with 10,667 more rows
```

```
#10 best rated movies
mean_movie %>% arrange(desc(mean_rating)) %>%
  top_n(10,mean_rating)
```

```
## # A tibble: 10,677 x 5
## # Groups:   movieId, title [10,677]
##    movieId title                                   genres            n mean_rating
##      <dbl> <chr>                                   <chr>         <int>       <dbl>
##  1    3226 Hellhounds on My Trail (1999)           Documentary       1           5
##  2   33264 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)    Drama             2           5
##  3   42783 Shadows of Forgotten Ancestors (1964)  Drama|Roman~      1           5
##  4   51209 Fighting Elegy (Kenka erejii) (1966)   Action|Come~      1           5
##  5   53355 Sun Alley (Sonnenallee) (1999)         Comedy|Roma~      1           5
##  6   64275 Blue Light, The (Das Blaue Licht) (19~ Drama|Fanta~      1           5
##  7    5194 Who's Singin' Over There? (a.k.a. Who~ Comedy            4           4.75
##  8   26048 Human Condition II, The (Ningen no jo~ Drama|War         4           4.75
##  9   26073 Human Condition III, The (Ningen no j~ Drama|War         4           4.75
## 10   65001 Constantine's Sword (2007)             Documentary       2           4.75
## # ... with 10,667 more rows
```

```
#10 best rated movies > 122 ratings (median)
mean_movie %>% filter(n>122) %>%
  arrange(desc(mean_rating)) %>%
  top_n(10,mean_rating)
```

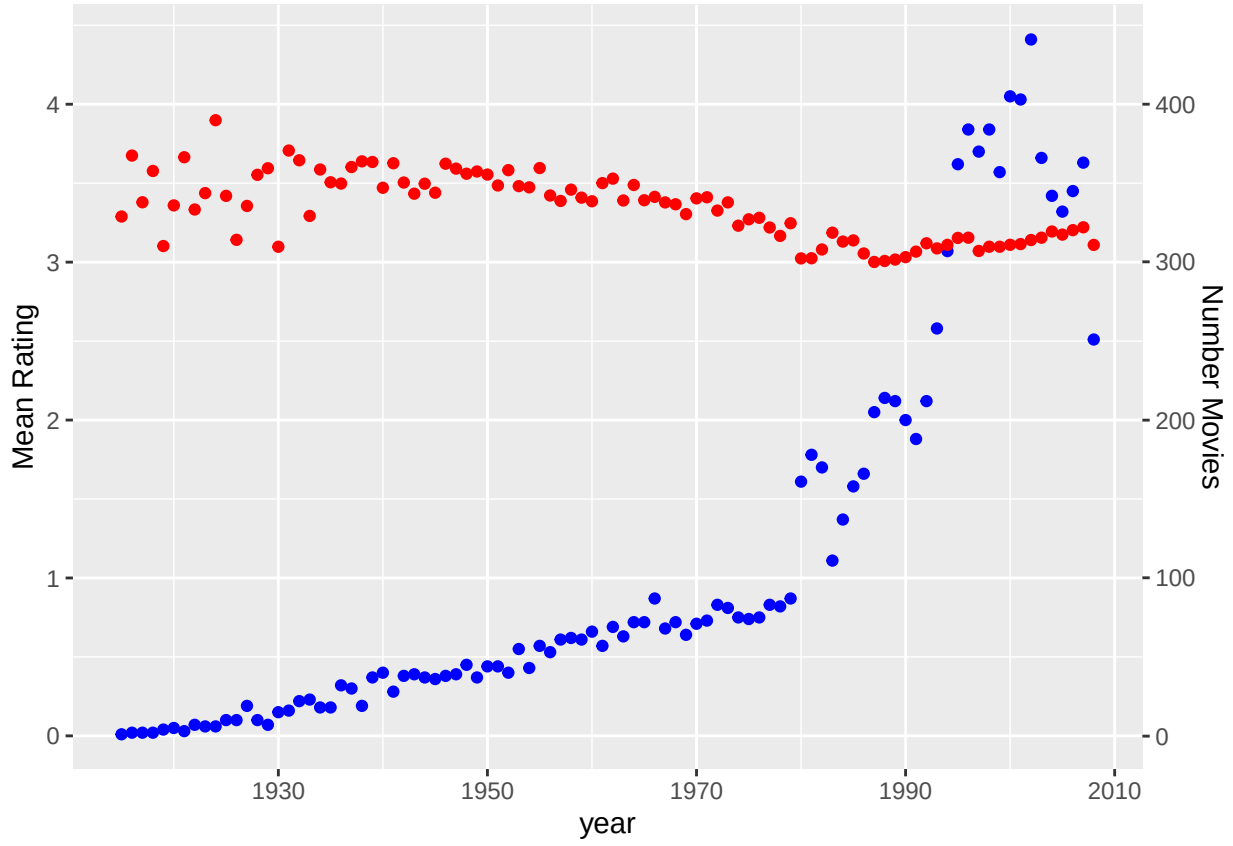
```
## # A tibble: 5,330 x 5
## # Groups:   movieId, title [5,330]
##    movieId title                               genres                n mean_rating
##      <dbl> <chr>                               <chr>             <int>       <dbl>
##  1     318 Shawshank Redemption, The (199~     Drama             28015        4.46
##  2     858 Godfather, The (1972)               Crime|Drama       17747        4.42
##  3      50 Usual Suspects, The (1995)          Crime|Mystery|Thri~ 21648      4.37
##  4     527 Schindler's List (1993)             Drama|War         23193        4.36
##  5     912 Casablanca (1942)                   Drama|Romance     11232        4.32
##  6     904 Rear Window (1954)                  Mystery|Thriller   7935        4.32
```

```
##  7     922 Sunset Blvd. (a.k.a. Sunset Bo~ Drama|Film-Noir|Ro~  2922        4.32
##  8    1212 Third Man, The (1949)           Film-Noir|Mystery|~  2967        4.31
##  9    3435 Double Indemnity (1944)         Crime|Drama|Film-N~  2154        4.31
## 10    1178 Paths of Glory (1957)           Drama|War            1571        4.31
## # ... with 5,320 more rows
```

After looking at the most rated and best rated movies we can see that there is a clear benefit to looking at the movies with a minimum number of ratings to get more accurate and expected results. A breakdown of the movies and average ratings per year can be shown.

```
## # A tibble: 10,677 x 6
## # Groups:   movieId, title [10,677]
##    movieId title                genres                      n mean_rating  year
##      <dbl> <chr>                <chr>                   <int>       <dbl> <dbl>
##  1       1 Toy Story (1995)     Adventure|Animation|Chi~ 23790        3.93  1995
##  2       2 Jumanji (1995)       Adventure|Children|Fant~ 10779        3.21  1995
##  3       3 Grumpier Old Men (1~ Comedy|Romance            7028        3.15  1995
##  4       4 Waiting to Exhale (~ Comedy|Drama|Romance      1577        2.86  1995
##  5       5 Father of the Bride~ Comedy                    6400        3.07  1995
##  6       6 Heat (1995)          Action|Crime|Thriller    12346        3.82  1995
##  7       7 Sabrina (1995)       Comedy|Romance            7259        3.36  1995
##  8       8 Tom and Huck (1995)  Adventure|Children         821        3.13  1995
##  9       9 Sudden Death (1995)  Action                    2278        3.00  1995
## 10      10 GoldenEye (1995)     Action|Adventure|Thrill~ 15187        3.43  1995
## # ... with 10,667 more rows
```

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



AS shown there is a larger variation in the movie ratings before 1950 this can be attributed to the low number of ratings. Therefore, it can be concluded that movies before 1950 and movies with less than the median number of reviews may skew the results.

```
#Create Smaller data sets Average user
mean_user <- edx_yr %>% group_by(userId) %>%
  summarize(n=n(), mean_rating=mean(rating))%>%
  select(userId,n,mean_rating)
```

3.2.2 Users

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

```
#Count number of users
nrow(mean_user)
```

```
## [1] 69878
```

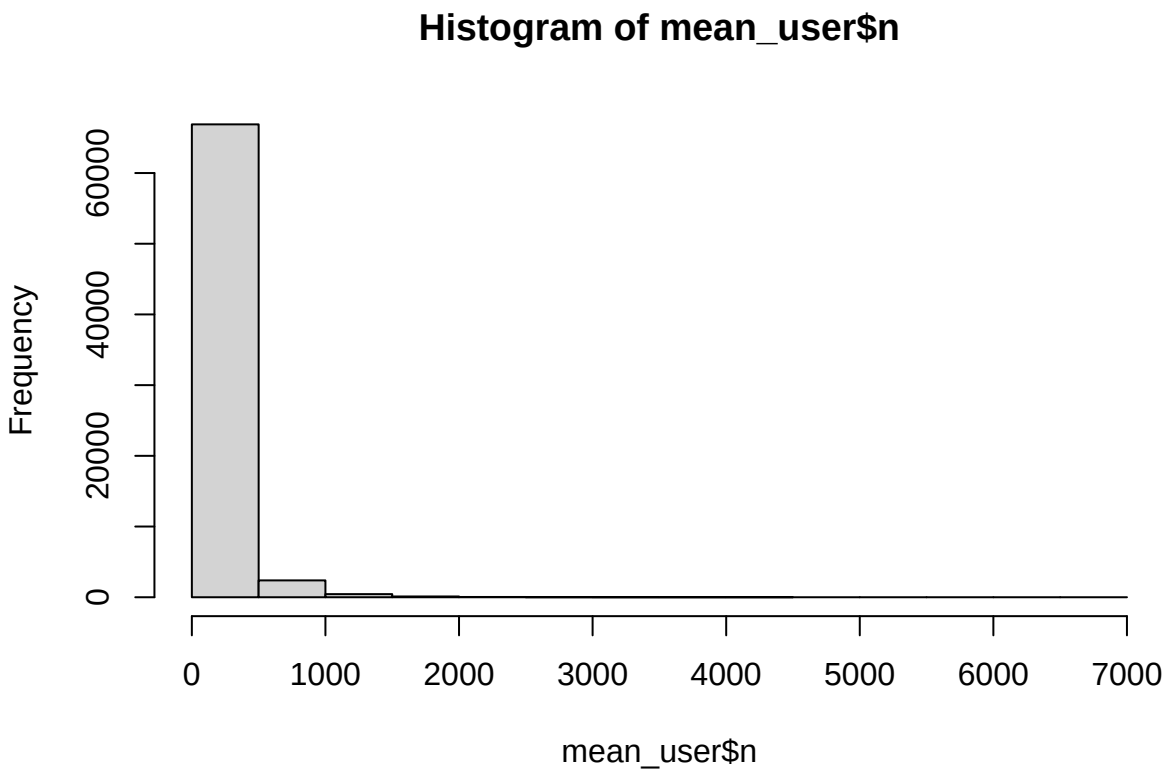
```
#user stats
summary(mean_user$n)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    32.0    62.0   128.8   141.0  6616.0
```

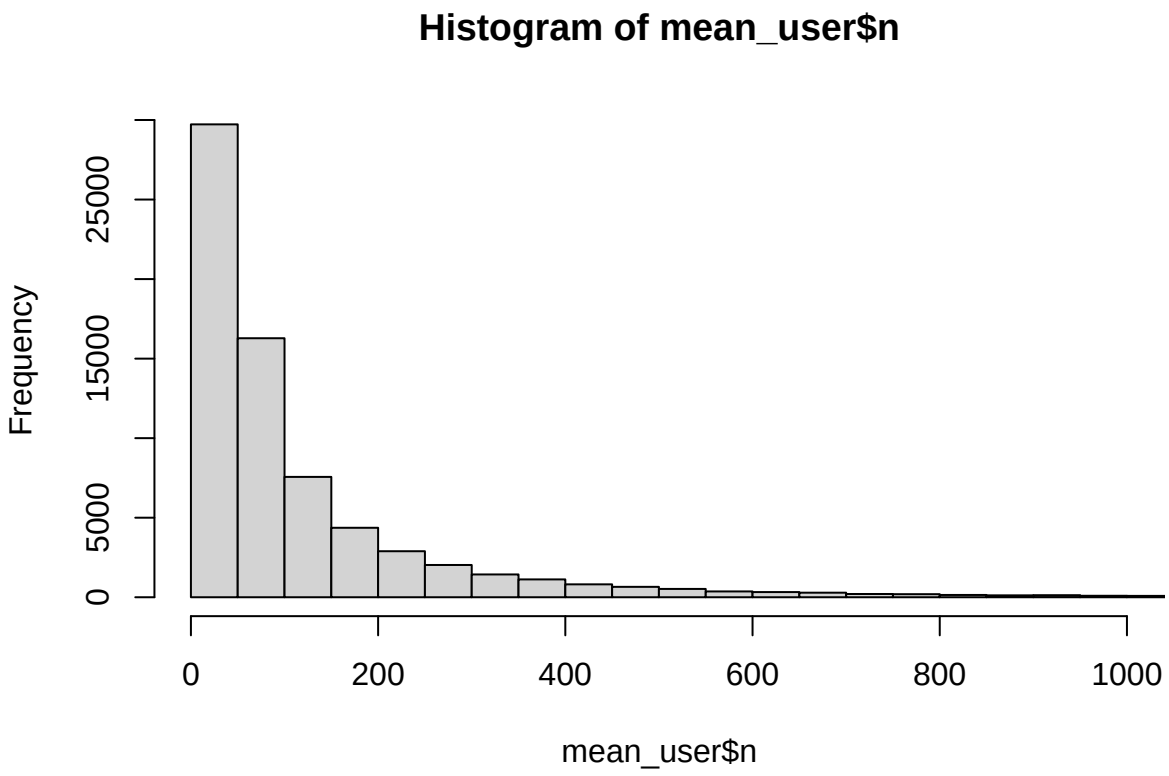
```
summary(mean_user$mean_rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.357   3.635   3.614   3.903   5.000
```

As shown there are 69878 users The summaries have been produced on both the number of ratings per user and the average rating of each user.



As shown there is a massive skew in that data due to a few users rating a large number of movies. By removing these users we get the following:



This more accurately shows the distribution of the users number of ratings.

```
#10 users with most ratings
mean_user %>% arrange(desc(n)) %>%
  top_n(10,n)
```

```
## # A tibble: 10 x 3
##    userId     n mean_rating
##     <int> <int>       <dbl>
##  1  59269  6616        3.26
##  2  67385  6360        3.20
##  3  14463  4648        2.40
##  4  68259  4036        3.58
##  5  27468  4023        3.83
##  6  19635  3771        3.50
##  7   3817  3733        3.11
##  8  63134  3371        3.27
##  9  58357  3361        3.00
## 10  27584  3142        3.00
```

```
#10 users with best ratings
mean_user %>% arrange(desc(mean_rating)) %>%
  top_n(10,mean_rating)
```

```
## # A tibble: 20 x 3
##    userId     n mean_rating
##     <int> <int>       <dbl>
##  1      1    19           5
```

```
##  2   7984    17           5
##  3  11884    18           5
##  4  13027    29           5
##  5  13513    17           5
##  6  13524    20           5
##  7  15575    29           5
##  8  18965    49           5
##  9  22045    18           5
## 10  26308    17           5
## 11  27831    20           5
## 12  30519    17           5
## 13  35184    23           5
## 14  42649    20           5
## 15  52674    14           5
## 16  52749    85           5
## 17  54009    27           5
## 18  65873    19           5
## 19  68379    56           5
## 20  71422    16           5
```

```
#10 users with best ratings > 62 ratings (median)
mean_user %>% filter(n>62) %>%
  arrange(desc(mean_rating)) %>%
  top_n(10,mean_rating)
```

```
## # A tibble: 10 x 3
##    userId     n mean_rating
##     <int> <int>       <dbl>
##  1  52749    85        5
##  2  27098    87        4.99
##  3  36022    91        4.96
##  4  12330    86        4.95
##  5   5763   214        4.93
##  6  59987   202        4.90
##  7  36896   149        4.89
##  8   7605    92        4.89
##  9  24125    77        4.86
## 10   9210    94        4.85
```

```
#10 users with worst ratings > 62 ratings (median)
mean_user %>% filter(n>62) %>%
  arrange(mean_rating)
```

```
## # A tibble: 34,810 x 3
##    userId     n mean_rating
##     <int> <int>       <dbl>
##  1  24176   131        1
##  2  59342   711        1.06
##  3  25674    67        1.22
##  4   4043   232        1.31
##  5  69898    89        1.31
##  6  26150   359        1.34
##  7  20240   103        1.37
##  8  45157   180        1.44
##  9  25969    71        1.46
```

```
## 10   9008   385        1.49
## # ... with 34,800 more rows
```

After looking at the most rated, best and worst rated users there is a clear benefit to looking at the users with a minimum number of ratings to get more accurate and expected results. This aligns to the movies analysis.

```
#Create Smaller data sets Average Movie
mean_genre <- edx %>% group_by(genres) %>%
  summarize(n=n(), mean_rating=mean(rating))%>%
  select(genres,n,mean_rating)
```

3.2.3 Genres

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

```
#Count number of movies
nrow(mean_genre)
```

```
## [1] 797
```

As shown there are 797 genres.

```
#Create Smaller data sets Average Movie
#10 most rated genres
mean_genre %>% arrange(desc(n)) %>%
  top_n(10,n)
```

```
## # A tibble: 10 x 3
##    genres                        n mean_rating
##    <chr>                     <int>       <dbl>
##  1 Drama                    733296        3.71
##  2 Comedy                   700889        3.24
##  3 Comedy|Romance           365468        3.41
##  4 Comedy|Drama             323637        3.60
##  5 Comedy|Drama|Romance     261425        3.65
##  6 Drama|Romance            259355        3.61
##  7 Action|Adventure|Sci-Fi  219938        3.51
##  8 Action|Adventure|Thriller 149091       3.43
##  9 Drama|Thriller           145373        3.45
## 10 Crime|Drama              137387        3.95
```

```
#10 best rated genres
mean_genre %>% arrange(desc(mean_rating)) %>%
  top_n(10,mean_rating)
```

```
## # A tibble: 10 x 3
##    genres                                  n mean_rating
##    <chr>                               <int>       <dbl>
##  1 Animation|IMAX|Sci-Fi                   7        4.71
##  2 Drama|Film-Noir|Romance              2989        4.30
##  3 Action|Crime|Drama|IMAX              2353        4.30
##  4 Animation|Children|Comedy|Crime      7167        4.28
##  5 Film-Noir|Mystery                    5988        4.24
##  6 Crime|Film-Noir|Mystery              4029        4.22
##  7 Film-Noir|Romance|Thriller           2453        4.22
##  8 Crime|Film-Noir|Thriller             4844        4.21
##  9 Crime|Mystery|Thriller              26892        4.20
```

```
## 10 Action|Adventure|Comedy|Fantasy|Romance 14809        4.20
```

There is a clear differentiation between ratings and the associated genres.

3.3 Prediction model

Building out the actual prediction model

3.3.1 Default Data Running through the methodology using the default edx dataset with no cleaning of data.

3.3.1.1 Splitting the data Splitting the data into the testing and training data using a 10% split. Importantly a semijoin is used to ensure that the users and movies from the test est exist in the train set.

```
#Create training and test data: Split on a 10% for testing
set.seed(1, sample.kind = "Rounding")
```

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

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

test_set <- edx[test_index,]
train_set <- edx[-test_index,]

#Due to regularization issues we need to ensure users and movies in test set are covered in the trainin

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

nrow(test_set)
```

```
## [1] 899990
```

```
nrow(train_set)
```

```
## [1] 8100048
```

3.3.1.2 Simple Average Prediction Running an RMSE on setting the prediction as the mean of the dataset.

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

RMSE_mu <- sqrt(mean((test_set$rating-mu)^2))
RMSE_mu
```

```
## [1] 1.060054
```

As shown this is still far near the goal RMSE of < 0.86490.

3.3.1.3 Movie Effect Prediction Running an RMSE on setting the prediction as the mean plus the average variation per movie

```
#Create the regularized for sum and mean
train_movie <- train_set %>% group_by(movieId) %>%
  summarize(n=n(),rmean=mean(rating-mu),rsum=sum(rating-mu))
```

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

```
#Apply to test and training set
test_movie <- test_set %>% left_join(train_movie,by='movieId')
```

```r
any(is.na(test_movie))
```

```
## [1] FALSE
```

```r
any(is.na(train_movie))
```

```
## [1] FALSE
```

```r
#Sample size regularisation accounting for sample size n
test_movie <- test_movie %>% mutate(reg=rsum/n)

RMSE_mov_nopen <- sqrt(mean((test_set$rating-(mu+test_movie$reg))^2))
RMSE_mov_nopen
```

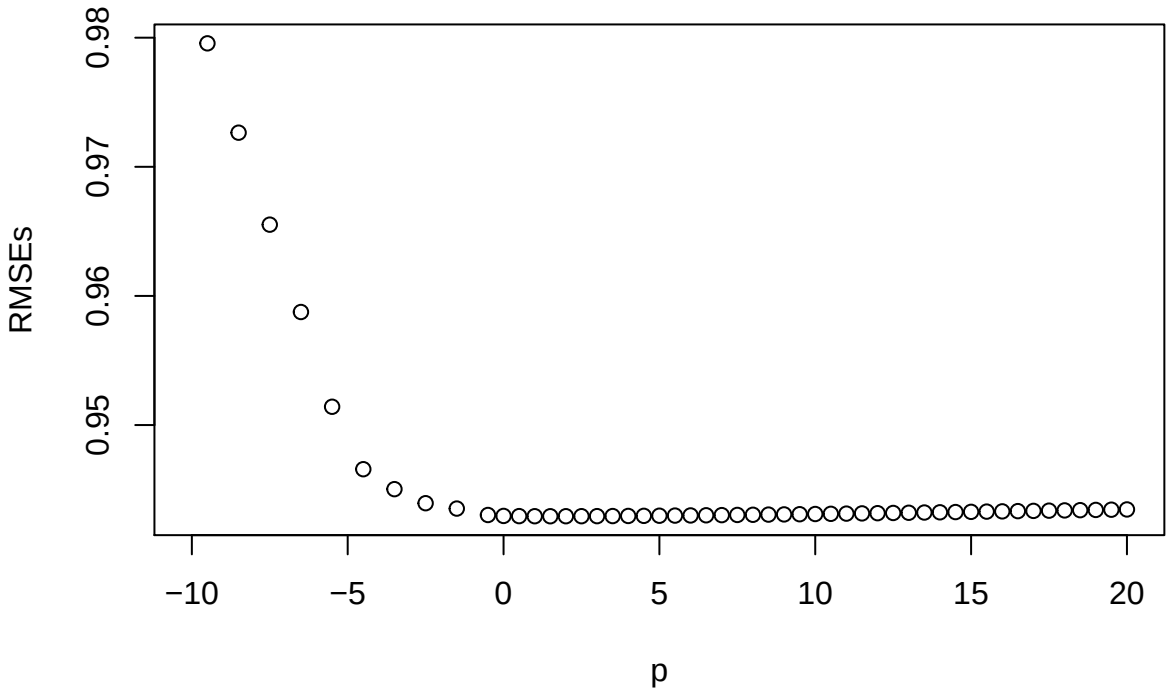
```
## [1] 0.9429615
```

This method shows a marked improvement on the average prediction but us still well above the goal RMSE of < 0.86490. AS shown there is no change from the movie effect as this just sum/count results in the average. Therefore, a penalty term is introduced. Finding the optimal penalty term was achieved by initially running large steps for a large range then more accurate steps around the optimized point.

3.3.1.4 Movie Regularization Prediction

This method involves regularization of the movie effect.

```r
#regularisation optimised with penalty term p
p <- seq(-10,20,0.5)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_movie <- train_movie %>% mutate(pen=rsum/(n+p))
  tune_movie <- test_set %>% left_join(train_movie,by='movieId')
  sqrt(mean((test_set$rating-(mu+tune_movie$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
#Find optimal penalty term
pmin_mov <- p[which.min(RMSEs)]
pmin_mov
```

```
## [1] 1.5
```

```
#final optimised output
test_movie <- test_movie %>% mutate(reg=rsum/(n+pmin_mov))

RMSE_mov <- sqrt(mean((test_set$rating-(mu+test_movie$reg))^2))
RMSE_mov
```

```
## [1] 0.942937
```

This method results in a small improvement from the movie effect.

3.3.1.5 User Effect Prediction Running an RMSE on setting the prediction as the mean plus the average variation per user.

```
#Create the regularized for sum and mean
train_user <- train_set %>% group_by(userId) %>%
  summarize(n=n(),rmean=mean(rating-mu),rsum=sum(rating-mu))
```

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

```
test_user <- test_set %>% left_join(train_user,by='userId')

any(is.na(test_user))
```

```
## [1] FALSE
```

```
any(is.na(train_user))
```

```
## [1] FALSE
```

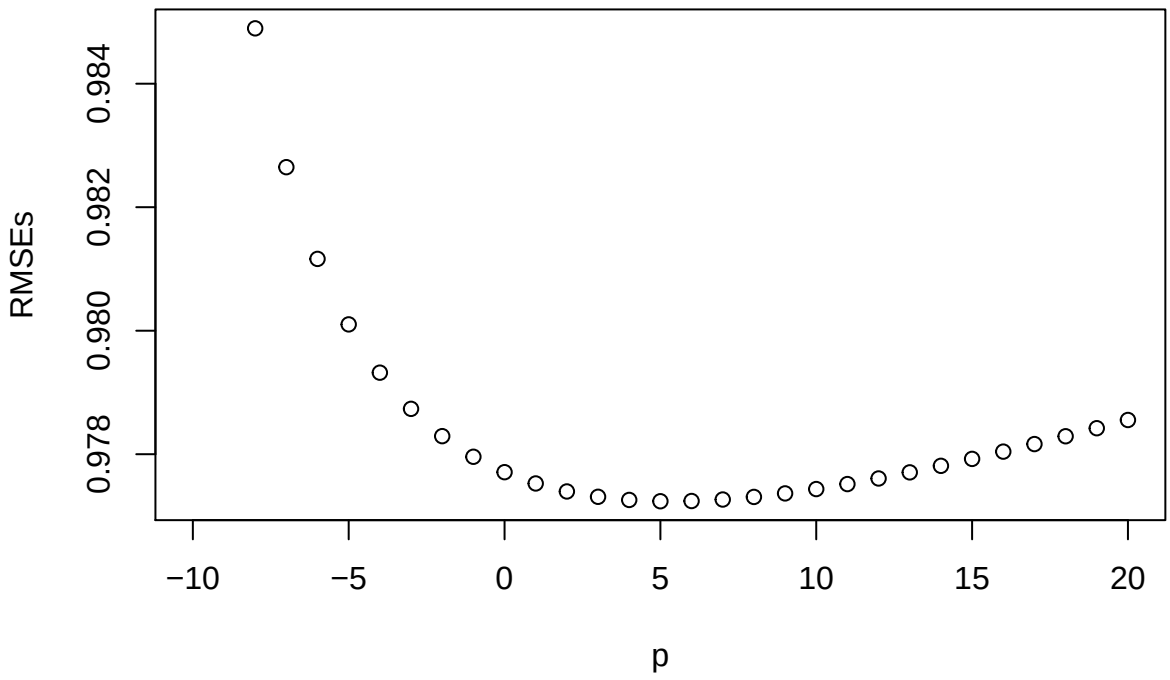
```
#Sample size regularization accounting for sample size n
test_user <- test_user %>% mutate(reg=rsum/n)

RMSE_use_nopen <- sqrt(mean((test_set$rating-(mu+test_user$reg))^2))
RMSE_use_nopen
```

```
## [1] 0.9777091
```

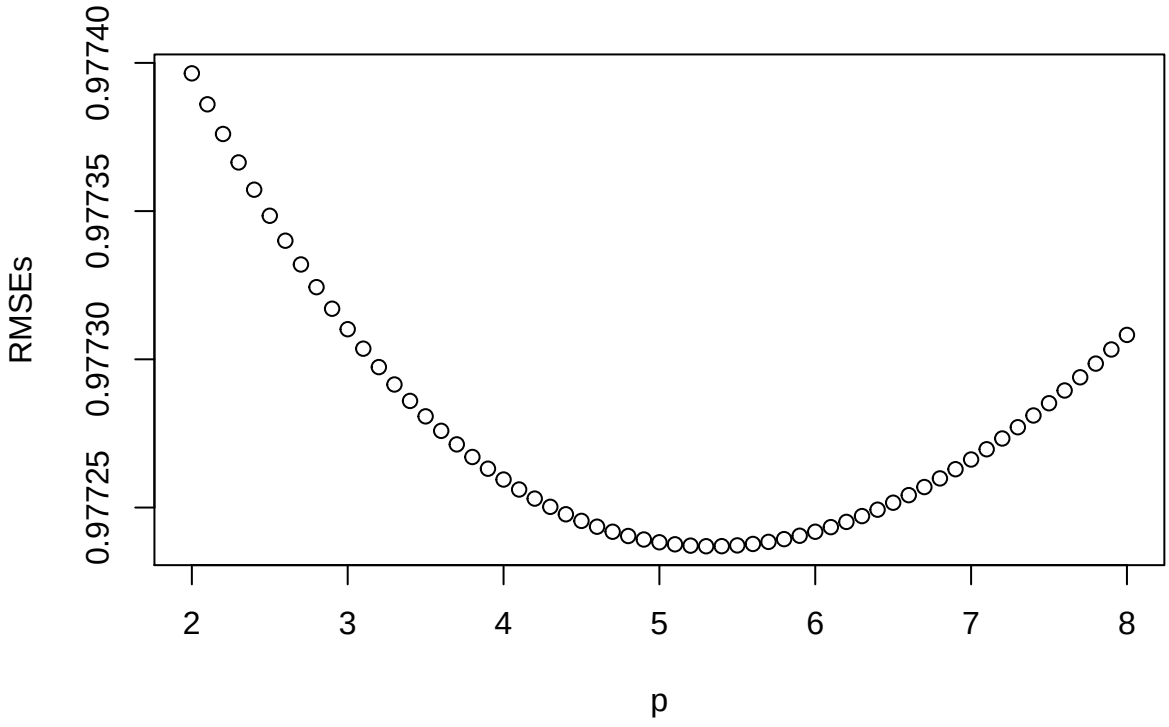
This method shows a marked improvement on the average prediction but is less effective than the movie effect approach.

3.3.1.6 User Regularization Prediction This method involves regularization of the user effect. Once again a penalty term is used and optimized with the same approach as before.

```
#regularization optimized with penalty term p
p <- seq(-10,20)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_user <- train_user %>% mutate(pen=rsum/(n+p))
  tune_user <- test_set %>% left_join(train_user,by='userId')
  sqrt(mean((test_set$rating-(mu+tune_user$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
#Improve the accuracy
p <- seq(2,8,0.1)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_user <- train_user %>% mutate(pen=rsum/(n+p))
  tune_user <- test_set %>% left_join(train_user,by='userId')
  sqrt(mean((test_set$rating-(mu+tune_user$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
pmin_use <- p[which.min(RMSEs)]

#final optimized output
test_user <- test_user %>% mutate(reg=rsum/(n+pmin_use))

RMSE_use <- sqrt(mean((test_set$rating-(mu+test_user$reg))^2))
RMSE_use
```

```
## [1] 0.9772369
```

This method results in a small improvement from the user effect but is still higher than the movie effect.

3.3.1.7 Genre Effect Prediction Running an RMSE on setting the prediction as the mean plus the average variation per user.

```
#Create the regularized for sum and mean
train_genre <- train_set %>% group_by(genres) %>%
  summarize(n=n(),rmean=mean(rating-mu),rsum=sum(rating-mu))
```

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

```
#Apply to test and training set
test_genre <- test_set %>% left_join(train_genre,by='genres')

any(is.na(test_genre))
```

```
## [1] FALSE
```

```r
any(is.na(train_genre))
```

```
## [1] FALSE
```

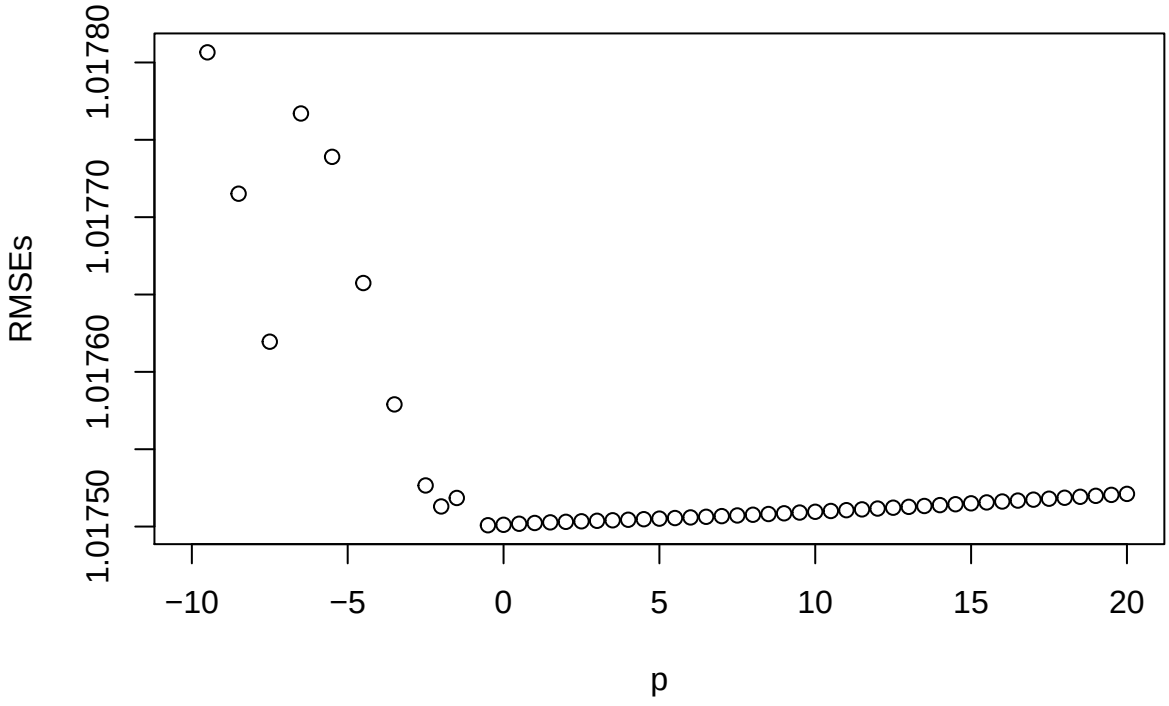
```r
#Sample size regularisation accounting for sample size n
test_genre <- test_genre %>% mutate(reg=rsum/n)

RMSE_gen_nopen <- sqrt(mean((test_set$rating-(mu+test_genre$reg))^2))
RMSE_gen_nopen
```

```
## [1] 1.017501
```

This method shows a very slight improvement on the average prediction but is less effective than the movie and user effect approaches.

3.3.1.8 Genre Regularization Prediction This method involves regularization of the genre effect. Once again a penalty term is used and optimized with the same approach as before.

```r
#regularisation optimised with penalty term p
p <- seq(-10,20,0.5)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_genre <- train_genre %>% mutate(pen=rsum/(n+p))
  tune_genre <- test_set %>% left_join(train_genre,by='genres')
  sqrt(mean((test_set$rating-(mu+tune_genre$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
#Find optimal peanalty term
pmin_gen <- p[which.min(RMSEs)]

#final optimised output
test_genre <- test_genre %>% mutate(reg=rsum/(n+pmin_gen))

RMSE_gen <- sqrt(mean((test_set$rating-(mu+test_genre$reg))^2))
RMSE_gen
```

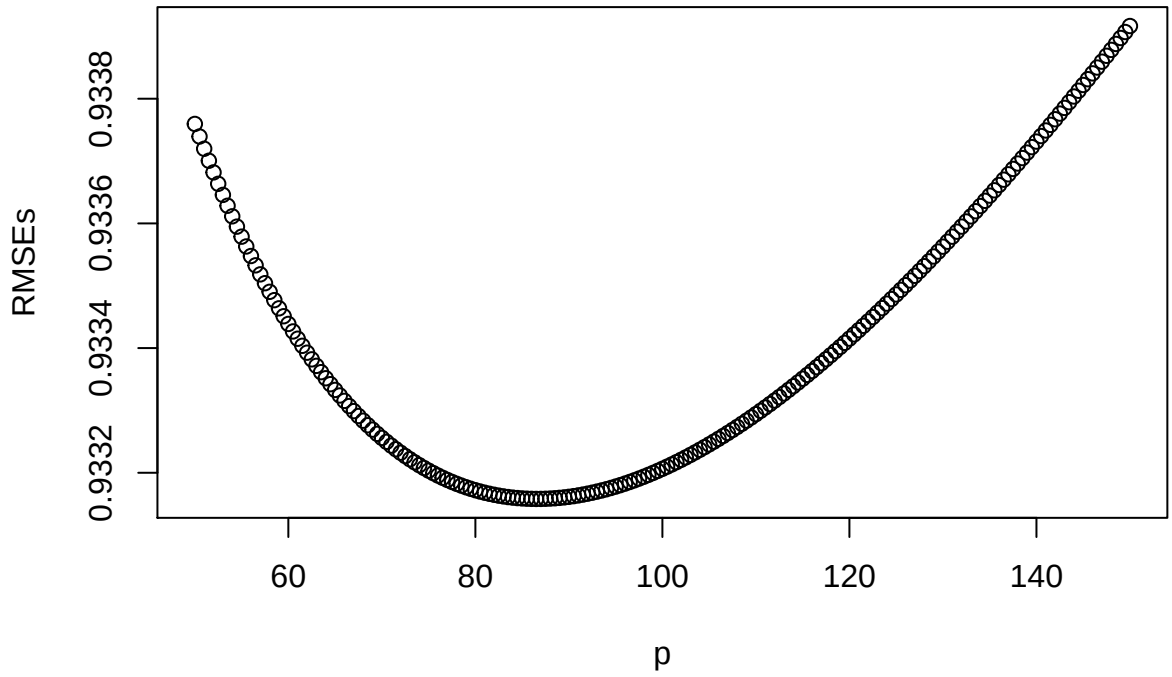
```
## [1] 1.017501
```

This method results in a small improvement from the genre effect but was still higher than both the movie and user effect. This was the least useful model. However, since there was still improvement it was used in the combined model

3.3.1.9 Genre, User and Movie Regularization Prediction Results Combining the genre, user and movie regularization we get the following:

```
#Improve the accuracy
p <- seq(50,150,0.5)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_user <- train_user %>% mutate(pen_u=rsum/(n+p))
  train_movie <- train_movie %>% mutate(pen_m=rsum/(n+p))
  train_genre <- train_genre %>% mutate(pen_g=rsum/(n+p))
  tune_comb <- test_set %>%
```

```
    left_join(train_user,by='userId') %>%
    left_join(train_movie,by='movieId')%>%
    left_join(train_genre,by='genres')
  sqrt(mean((test_set$rating-(mu+tune_comb$pen_u+tune_comb$pen_m+tune_comb$pen_g))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
pmin_comb <- p[which.min(RMSEs)]

#final optimized output
test_user <- test_user %>% mutate(reg=rsum/(n+pmin_comb))
test_movie <- test_movie %>% mutate(reg=rsum/(n+pmin_comb))

RMSE_comb <- sqrt(mean((test_set$rating-(mu+test_user$reg+test_movie$reg))^2))
RMSE_comb
```

```
## [1] 0.8849991
```

This is a much better result and is acceptable. However, it is still not below the target RMSE of < 0.86490. Therefore we will look to data cleaning to improve the result.

The final results are as follows:

```
#Build results table
results_table <- data.frame(Method='Default Mean', RMSE = RMSE_mu)
results_table <- bind_rows(results_table,data.frame(Method='Default Movie Mean', RMSE = RMSE_mov_nopen)
```

```
results_table <- bind_rows(results_table,data.frame(Method='Default Movie tuned', RMSE = RMSE_mov))
results_table <- bind_rows(results_table,data.frame(Method='Default User Mean', RMSE = RMSE_use_nopen))
results_table <- bind_rows(results_table,data.frame(Method='Default User tuned', RMSE = RMSE_use))
results_table <- bind_rows(results_table,data.frame(Method='Default Genre Mean', RMSE = RMSE_gen_nopen))
results_table <- bind_rows(results_table,data.frame(Method='Default Genre tuned', RMSE = RMSE_gen))
results_table <- bind_rows(results_table,data.frame(Method='Default Combined Model', RMSE = RMSE_comb))
results_table %>% knitr::kable()
```

Method	RMSE
Default Mean	1.0600537
Default Movie Mean	0.9429615
Default Movie tuned	0.9429370
Default User Mean	0.9777091
Default User tuned	0.9772369
Default Genre Mean	1.0175011
Default Genre tuned	1.0175009
Default Combined Model	0.8849991

```
summary_table <- data.frame(Method='Default Combined Model', RMSE = RMSE_comb)
```

There is an improvement in our model.

3.3.2 Median Limited Data

Running through the methodology using the cleaned data by median counts.

3.3.2.1 Cleaning and Splitting the data

The data will be cleaned using the median count of n=122 for movies and n=62 for users. Splitting the data into the testing and training data using a 10% split. Importantly a semi-join is used to ensure that the users and movies from the test est exist in the train set.

```
#---- Limit movies users and genres by median ----

#Get the mean counts
edx_movie <- edx %>% group_by(movieId) %>%
  summarize(n=n())
```

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

```
edx_user <- edx %>% group_by(userId) %>%
  summarize(n=n())
```

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

```
edx_genre <- edx %>% group_by(genres) %>%
  summarize(n=n())
```

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

```
median_mov_n <- median(edx_movie$n)
median_use_n <- median(edx_user$n)
median_gen_n <- median(edx_genre$n)

#Limit the by the mean

lim_movie <- edx_movie %>% filter(n>median_mov_n)
lim_user <- edx_user %>% filter(n>median_use_n)
lim_genre <- edx_genre %>% filter(n>median_gen_n)

#Apply to edx
lim_edx <- edx %>% semi_join(lim_movie, by = "movieId") %>%
  semi_join(lim_user, by = "userId") %>%
  semi_join(lim_genre, by = "genres")


head(lim_edx)
```

```
##    userId movieId rating timestamp
## 1:      5       1      1 857911264
## 2:      5       7      3 857911357
## 3:      5      25      3 857911265
## 4:      5      28      3 857913507
## 5:      5      30      5 857911752
## 6:      5      32      5 857911264
##                                                  title
## 1:                                     Toy Story (1995)
## 2:                                       Sabrina (1995)
## 3:                                Leaving Las Vegas (1995)
## 4:                                      Persuasion (1995)
## 5: Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)
## 6:                    12 Monkeys (Twelve Monkeys) (1995)
##                                        genres
```

```
## 1: Adventure|Animation|Children|Comedy|Fantasy
## 2:                                Comedy|Romance
## 3:                                 Drama|Romance
## 4:                                 Drama|Romance
## 5:                                   Crime|Drama
## 6:                               Sci-Fi|Thriller
```

```
nrow(lim_edx)
```

```
## [1] 7464825
```

```
#Create training and test data: Split on a 10% for testing
set.seed(1, sample.kind = "Rounding")
```

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

```
lim_test_index <- createDataPartition(lim_edx$rating, times = 1, p = 0.1, list=FALSE)

lim_test_set <- lim_edx[lim_test_index,]
lim_train_set <- lim_edx[-lim_test_index,]

#Due to regularization issues we need to ensure users and movies in test set are covered in the trainin

lim_test_set <- lim_test_set %>% semi_join(lim_train_set, by = "movieId") %>%
  semi_join(lim_train_set, by = "userId") %>%
  semi_join(lim_train_set, by = "genres")

nrow(lim_test_set)
```

```
## [1] 746484
```

```
nrow(lim_train_set)
```

```
## [1] 6718341
```

3.3.2.2 Simple Average Prediction Running an RMSE on setting the prediction as the mean of the dataset.

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

RMSE_mu <- sqrt(mean((lim_test_set$rating-mu)^2))
RMSE_mu
```

```
## [1] 1.053038
```

Already there is an improvement towards the goal of < 0.86490.

3.3.2.3 Movie Effect Prediction Running an RMSE on setting the prediction as the mean plus the average variation per movie

```
#Create the regularized for sum and mean
train_movie <- lim_train_set %>% group_by(movieId) %>%
  summarize(n=n(),rmean=mean(rating-mu),rsum=sum(rating-mu))
```

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

```
#Apply to test and training set
test_movie <- lim_test_set %>% left_join(train_movie,by='movieId')
```

```r
any(is.na(test_movie))
```

```
## [1] FALSE
```

```r
any(is.na(train_movie))
```

```
## [1] FALSE
```

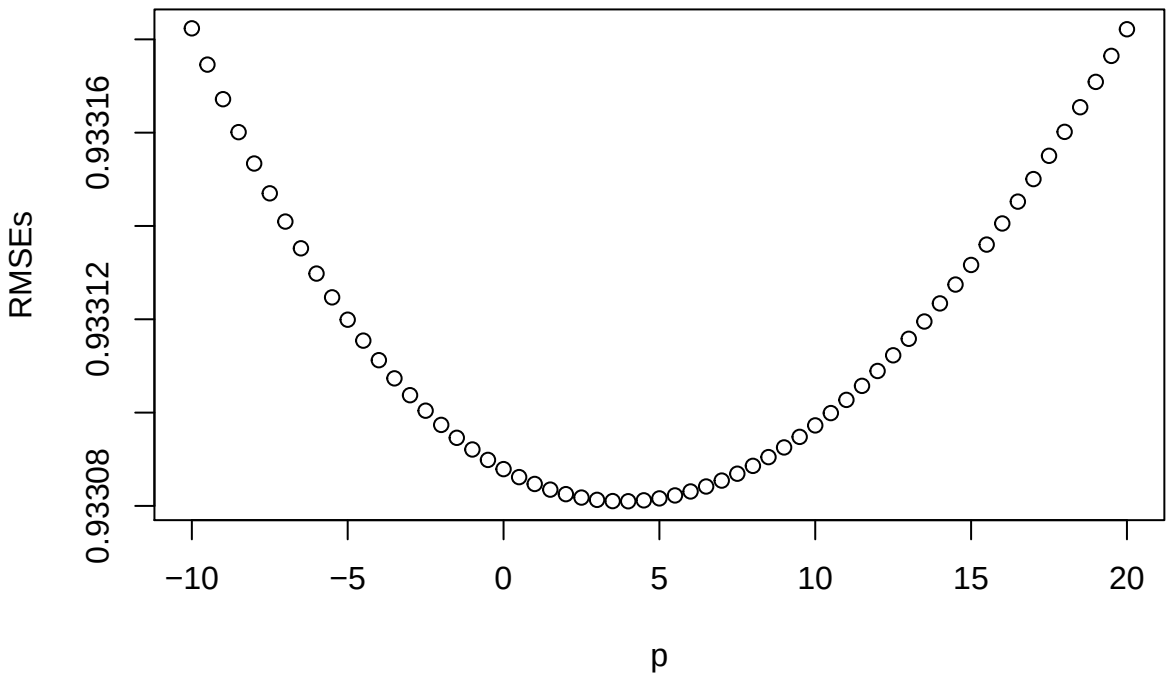
```r
#Sample size regularisation accounting for sample size n
test_movie <- test_movie %>% mutate(reg=rsum/n)

RMSE_mov_nopen <- sqrt(mean((lim_test_set$rating-(mu+test_movie$reg))^2))
RMSE_mov_nopen
```

```
## [1] 0.9330879
```

This method shows an improvement on the average prediction but is still above the goal RMSE of < 0.86490.

3.3.2.4 Movie Regularization Prediction This method involves regularization of the movie effect. Optimizing a penalty term using the same method as before.

```r
#regularisation optimised with penalty term p
p <- seq(-10,20,0.5)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_movie <- train_movie %>% mutate(pen=rsum/(n+p))
  tune_movie <- lim_test_set %>% left_join(train_movie,by='movieId')
  sqrt(mean((lim_test_set$rating-(mu+tune_movie$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
#Find optimal peanalty term
pmin_mov <- p[which.min(RMSEs)]

#final optimised output
test_movie <- test_movie %>% mutate(reg=rsum/(n+pmin_mov))

RMSE_mov <- sqrt(mean((lim_test_set$rating-(mu+test_movie$reg))^2))
RMSE_mov
```

```
## [1] 0.933081
```

This method results in a small improvement from the movie effect. and an improvement when compared to the initial default data run

3.3.2.5 User Effect Prediction Running the RMSE on setting the prediction as the mean plus the average variation per user.

```
#Create the regularized for sum and mean
train_user <- lim_train_set %>% group_by(userId) %>%
  summarize(n=n(),rmean=mean(rating-mu),rsum=sum(rating-mu))
```

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

```
test_user <- lim_test_set %>% left_join(train_user,by='userId')

any(is.na(test_user))
```

```
## [1] FALSE
```

```r
any(is.na(train_user))
```

```
## [1] FALSE
```

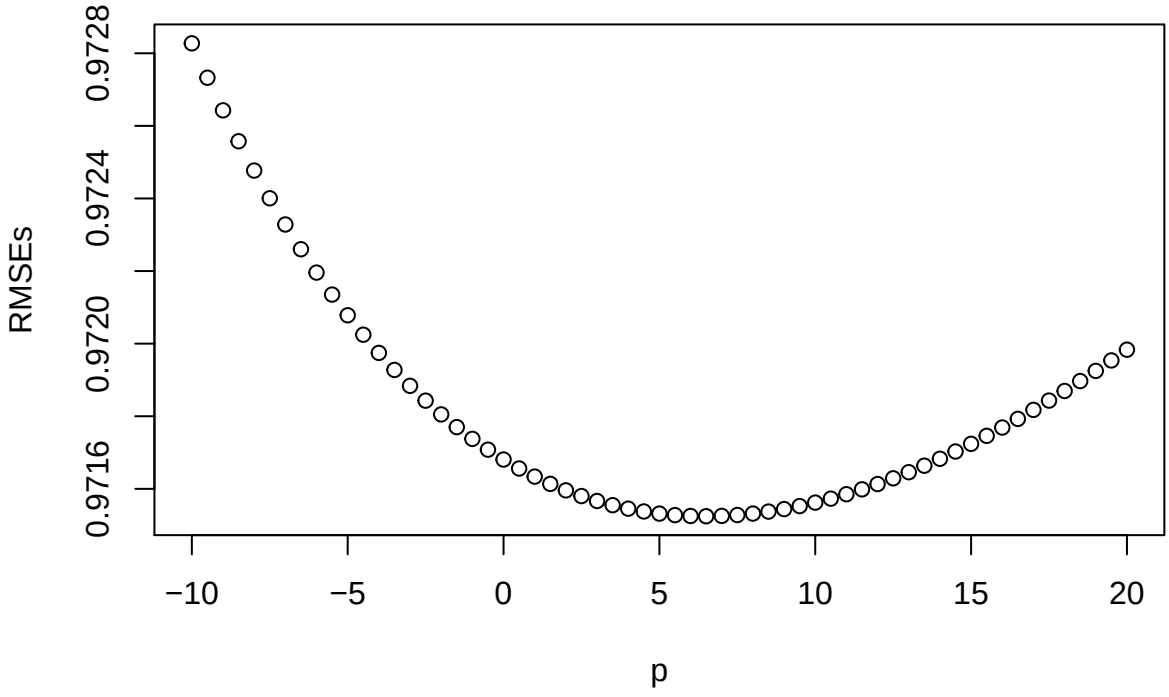
```r
#Sample size regularization accounting for sample size n
test_user <- test_user %>% mutate(reg=rsum/n)

RMSE_use_nopen <- sqrt(mean((lim_test_set$rating-(mu+test_user$reg))^2))
RMSE_use_nopen
```

```
## [1] 0.9716807
```

This method shows an improvement from the default run but still performs worse than the default movie effect approach.

3.3.2.6 User Regularization Prediction This method involves regularization of the user effect. Once again a penalty term is used and optimized with the same approach as before.

```r
#regularization optimized with penalty term p
p <- seq(-10,20,0.5)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_user <- train_user %>% mutate(pen=rsum/(n+p))
  tune_user <- lim_test_set %>% left_join(train_user,by='userId')
  sqrt(mean((lim_test_set$rating-(mu+tune_user$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
pmin_use <- p[which.min(RMSEs)]

#final optimized output
test_user <- test_user %>% mutate(reg=rsum/(n+pmin_use))

RMSE_use <- sqrt(mean((lim_test_set$rating-(mu+test_user$reg))^2))
RMSE_use
```

```
## [1] 0.9715245
```

3.3.2.7 Genre Effect Prediction Running the RMSE on setting the prediction as the mean plus the average variation per genre.

```
#Create the regularized for sum and mean
train_genre <- lim_train_set %>% group_by(genres) %>%
  summarize(n=n(),rmean=mean(rating-mu),rsum=sum(rating-mu))
```

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

```
#Apply to test and training set
test_genre <- lim_test_set %>% left_join(train_genre,by='genres')

any(is.na(test_genre))
```

```
## [1] FALSE
```

```
any(is.na(train_genre))
```

```
## [1] FALSE
```

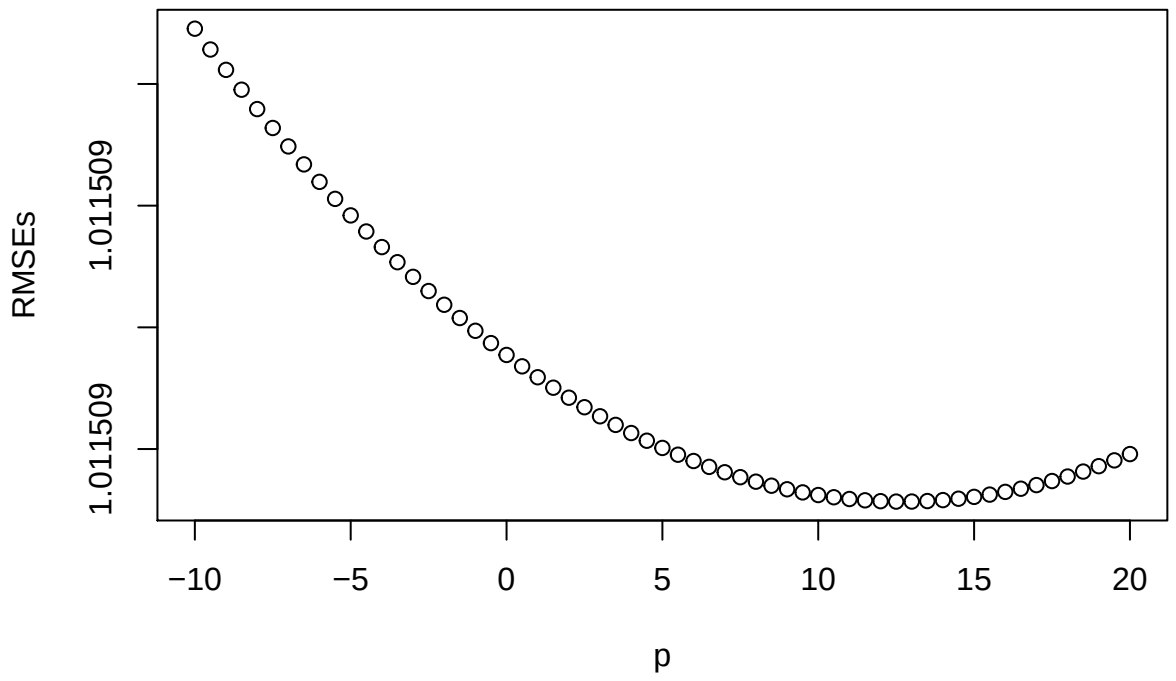
```
#Sample size regularisation accounting for sample size n
test_genre <- test_genre %>% mutate(reg=rsum/n)

RMSE_gen_nopen <- sqrt(mean((lim_test_set$rating-(mu+test_genre$reg))^2))
RMSE_gen_nopen
```

```
## [1] 1.011509
```

This method shows an improvement from the default run but still performs worse than the default movie and user effect approaches.

3.3.2.8 Genre Regularization Prediction This method involves regularization of the user effect. Once again a penalty term is used and optimized with the same approach as before.

```
#regularisation optimised with penalty term p
p <- seq(-10,20,0.5)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_genre <- train_genre %>% mutate(pen=rsum/(n+p))
  tune_genre <- lim_test_set %>% left_join(train_genre,by='genres')
  sqrt(mean((lim_test_set$rating-(mu+tune_genre$pen))^2))
})

#plot outputs
plot(p,RMSEs)
```



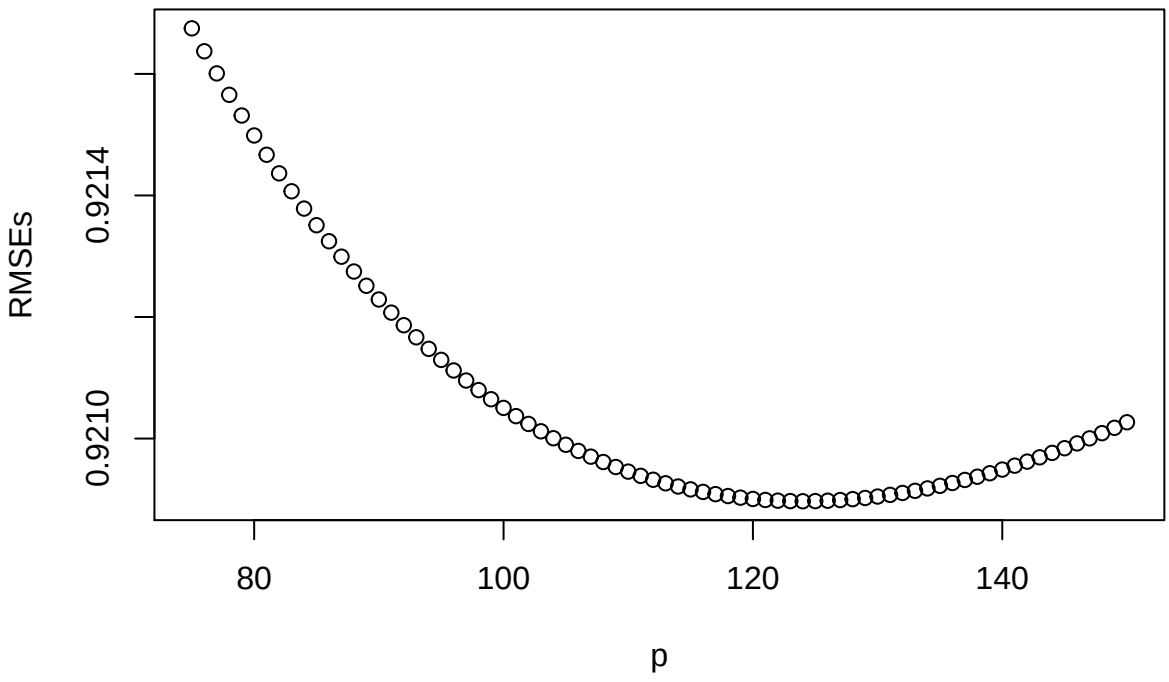
```
#Find optimal peanalty term
pmin_gen <- p[which.min(RMSEs)]

#final optimised output
test_genre <- test_genre %>% mutate(reg=rsum/(n+pmin_gen))

RMSE_gen <- sqrt(mean((lim_test_set$rating-(mu+test_genre$reg))^2))
RMSE_gen
```

```
## [1] 1.011509
```

Similarly there is improvement but it is very small. Genre remains the poorest performing model.

3.3.2.9 Genre, User and Movie Regularization Prediction Combining the genre, user and movie regularization we get the following:

```
#Improve the accuracy
p <- seq(75,150,1)

#sapply the terms
RMSEs <- sapply(p,function(p){
  train_user <- train_user %>% mutate(pen_u=rsum/(n+p))
  train_movie <- train_movie %>% mutate(pen_m=rsum/(n+p))
  train_genre <- train_genre %>% mutate(pen_g=rsum/(n+p))
  tune_comb <- lim_test_set %>%
    left_join(train_user,by='userId') %>%
    left_join(train_movie,by='movieId')%>%
    left_join(train_genre,by='genres')
  sqrt(mean((lim_test_set$rating-(mu+tune_comb$pen_u+tune_comb$pen_m+tune_comb$pen_g))^2))
})

#plot outputs
plot(p,RMSEs)
```




```
pmin_comb_lim <- p[which.min(RMSEs)]

#final optimized output
test_user <- test_user %>% mutate(reg=rsum/(n+pmin_comb_lim))
test_movie <- test_movie %>% mutate(reg=rsum/(n+pmin_comb_lim))

RMSE_comb <- sqrt(mean((lim_test_set$rating-(mu+test_user$reg+test_movie$reg))^2))
RMSE_comb
```

```
## [1] 0.8740286
```

This is a much better result and a step better than the default approach. However, it is still not below the target RMSE of < 0.86490. This means that in order to recommend a good enough system it would be best to recommend movies that have more than the median of the number of ratings. This means that small niche shows and movies will likely get removed. However, It would push for more watched movies to be seen.

There can still be other effects that are skewing the data. The processing power limitation now forms a serious block as the model cannot be made any more complex. The results are now as follows:

```
#Append to results table
results_table <- bind_rows(results_table,data.frame(Method='Limited Mean', RMSE = RMSE_mu))
results_table <- bind_rows(results_table,data.frame(Method='Limited Movie Mean', RMSE = RMSE_mov_nopen))
results_table <- bind_rows(results_table,data.frame(Method='Limited Movie tuned', RMSE = RMSE_mov))
results_table <- bind_rows(results_table,data.frame(Method='Limited User Mean', RMSE = RMSE_use_nopen))
results_table <- bind_rows(results_table,data.frame(Method='Limited User tuned', RMSE = RMSE_use))
results_table <- bind_rows(results_table,data.frame(Method='Limited Genre Mean', RMSE = RMSE_gen_nopen))
results_table <- bind_rows(results_table,data.frame(Method='Limited Genre tuned', RMSE = RMSE_gen))
results_table <- bind_rows(results_table,data.frame(Method='Limited Combined Model', RMSE = RMSE_comb))
```

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

Method	RMSE
Default Mean	1.0600537
Default Movie Mean	0.9429615
Default Movie tuned	0.9429370
Default User Mean	0.9777091
Default User tuned	0.9772369
Default Genre Mean	1.0175011
Default Genre tuned	1.0175009
Default Combined Model	0.8849991
Limited Mean	1.0530384
Limited Movie Mean	0.9330879
Limited Movie tuned	0.9330810
Limited User Mean	0.9716807
Limited User tuned	0.9715245
Limited Genre Mean	1.0115088
Limited Genre tuned	1.0115085
Limited Combined Model	0.8740286

3.4 Validation

3.4.1 Default Dataset Run the validation using the method lined out without data cleaning. To show the pure results of the model.

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

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

```
## [1] FALSE
```

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

```
## [1] FALSE
```

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

```
## [1] FALSE
```

```
#Final validated RMSE
RMSE_Val <- sqrt(mean((validation$rating-(mu+val_user$reg+val_movie$reg +val_genre$reg))^2))
RMSE_Val
```

```
## [1] 0.9345861
```

```
#Append to results tables
results_table <- bind_rows(results_table,data.frame(Method='Default Validation', RMSE = RMSE_Val))
summary_table <- bind_rows(summary_table,data.frame(Method='Default Validation', RMSE = RMSE_Val))
```

This shows a very respectable final RMSE. However, this is above the target RMSE of 0.86490. Knowing that data issues form part of this. If chosen to only report on the items with more ratings than the mean number of ratings, this can improve the accuracy of the results.

3.4.2 Final Output

The final output using cleaner data

3.4.2.1 Cleaner data

Cleaning the data by mean number of ratings for both movies and users. A limit is applied to the validation set essentially throwing out the unique and different values that could negatively influence the recommendation.

```
############################################################
# VALID CLEAN: - CLEAN DATA
############################################################
#Get the mean counts
edx_movie <- edx %>% group_by(movieId) %>%
  summarize(n=n())
```

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

```
edx_user <- edx %>% group_by(userId) %>%
  summarize(n=n())
```

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

```
edx_genre <- edx %>% group_by(genres) %>%
  summarize(n=n())
```

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

```
median_mov_n <- median(edx_movie$n)
median_use_n <- median(edx_user$n)
median_gen_n <- median(edx_genre$n)

#Limit the by the mean

lim_movie <- edx_movie %>% filter(n>median_mov_n)
lim_user <- edx_user %>% filter(n>median_use_n)
lim_genre <- edx_genre %>% filter(n>median_gen_n)

#Apply to edx
lim_edx <- edx %>% semi_join(lim_movie, by = "movieId") %>%
  semi_join(lim_user, by = "userId") %>%
  semi_join(lim_genre, by = "genres")


lim_valid <- validation %>% semi_join(lim_edx, by = "movieId") %>%
  semi_join(lim_edx, by = "userId") %>%
  semi_join(lim_edx, by = "genres")

any(is.na(lim_valid))
```

```
## [1] FALSE
```

There are no NA's therefore the dataset is good.

3.4.2.2 Final RMSE

Running the code as per the same method as above.

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

```
## [1] FALSE
```

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

```
## [1] FALSE
```

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

```
## [1] FALSE
```

```
############################################################
# VALID CLEAN: RMSE - REGULARIZATION USER AND MOVIE EFFECT
############################################################

#Final validated RMSE
RMSE_Val <- sqrt(mean((lim_valid$rating-(mu+val_user$reg+val_movie$reg +val_genre$reg))^2))
RMSE_Val
```

```
## [1] 0.9200898
```

```
#Rated Items:
nrow(lim_valid)/nrow(validation)
```

```
## [1] 0.8265718
```

```
#Append to results table
results_table <- bind_rows(results_table,data.frame(Method='Limited Validation', RMSE = RMSE_Val))
summary_table <- bind_rows(summary_table,data.frame(Method='Limited Validation', RMSE = RMSE_Val))
```

The final RMSE is an improvement from the mean estimation. which is well below the target of < 0.86490. This is a very positive result. However only 82% of the validation set has been evaluated. Therefore the high accuracy comes at a very large cost.

The final results are therefore:

Method	RMSE
Default Mean	1.0600537
Default Movie Mean	0.9429615
Default Movie tuned	0.9429370
Default User Mean	0.9777091
Default User tuned	0.9772369
Default Genre Mean	1.0175011
Default Genre tuned	1.0175009
Default Combined Model	0.8849991
Limited Mean	1.0530384
Limited Movie Mean	0.9330879
Limited Movie tuned	0.9330810
Limited User Mean	0.9716807
Limited User tuned	0.9715245
Limited Genre Mean	1.0115088
Limited Genre tuned	1.0115085
Limited Combined Model	0.8740286
Default Validation	0.9345861
Limited Validation	0.9200898

4. Conclusion

The quality of the data as well as skewing factors such as low numbers of ratings on movies and users rating few movies resulted in higher than expected RMSE values. Cleaner datasets revealed that the approaches used have the ability to predict better RMSE values but not lower than the target.

When building the rating system with a data optimized process an RMSE of lower than the default was achieved. However, as this does not evaluate outliers in the dataset the final accepted RMSE must be the default RMSE as shown below:

Method	RMSE
Default Combined Model	0.8849991
Default Validation	0.9345861
Limited Validation	0.9200898

Therefore the final RMSE is worse than the target which is a poor outcome. The model is not sufficient. The RMSE metric is optimized as best achievable with the data available but can be improved with more complex classifiers, for example K Nearest Neighbors approach. This project was limited by the large size of the dataset and not enough computing power.