MovieLens_Project

Movie Recommendation System using MovieLens Dataset

As a part of the course on Data Science this is the project called "Movie Recommendation System" to be submitted under the Capstone Project course. The project requires coding in R. Different algorithms will be used and the evaluation will be done using RMSE. The project report has been covered under four major heads, viz:

```
Introduction
Analysis
Results
Conclusion
```

1. Introduction

In daily life we come across recommendations made to us through emails, WhatsApp messages, prompts in social media sites like YouTube, Facebook etc.. We often wonder as to how these guys understand our taste and make such recommendations. A very relevant example is that of movie recommendations. In fact, the recommendations are based on the analysis of the data collected while we browse through the various applications or make enquiries in the sites. These data are then matched with the parameters available in the data set in respect of the movies and based on the accuracy of the matching parameters the recommendations are made.

1.1 Data Source

Context We will use the data set provided throught the link given in the course material. The version of movielens included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. The entire latest MovieLens dataset can be found at https://grouplens.org/datasets/movielens/latest/. We will use the 10M version of the MovieLens dataset to make the computation a little easier. MovieLens 10M version data set contains 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

Users were selected at random for inclusion. All users selected had rated at least 20 movies. Each user is represented by an id.

The data are contained in three files, movies.dat, ratings.dat and tags.dat. Also included are scripts for generating subsets of the data to support five-fold cross-validation of rating predictions. More details about the contents and use of all these files follows.

The first step will be to look at the structure of the data, visualize it and then progressively build a model.

This section describes the dataset and variables, and summarizes the goal of the project and key steps that were performed.

The data are contained in three files, movies.dat, ratings.dat and tags.dat. Also included are scripts for generating subsets of the data to support five-fold cross-validation of rating predictions.

User Ids Movielens users were selected at random for inclusion. Their ids have been anonymized.

Users were selected separately for inclusion in the ratings and tags data sets, which implies that user ids may appear in one set but not the other.

The anonymized values are consistent between the ratings and tags data files. That is, user id n, if it appears in both files, refers to the same real MovieLens user.

Ratings Data File Structure All ratings are contained in the file ratings.dat. Each line of this file represents one rating of one movie by one user, and has the following format:

UserID::MovieID::Rating::Timestamp

The lines within this file are ordered first by UserID, then, within user, by MovieID.

Ratings are made on a 5-star scale, with half-star increments.

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Tags Data File Structure All tags are contained in the file tags.dat. Each line of this file represents one tag applied to one movie by one user, and has the following format:

UserID::MovieID::Tag::Timestamp

The lines within this file are ordered first by UserID, then, within user, by MovieID.

Tags are user generated metadata about movies. Each tag is typically a single word, or short phrase. The meaning, value and purpose of a particular tag is determined by each user.

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Movies Data File Structure Movie information is contained in the file movies.dat. Each line of this file represents one movie, and has the following format:

MovieID::Title::Genres

MovieID is the real MovieLens id.

Movie titles, by policy, should be entered identically to those found in IMDB, including year of release. However, they are entered manually, so errors and inconsistencies may exist.

Genres are a pipe-separated list, and are selected from the following:

Action Adventure Animation Children's Comedy Crime Documentary Drama Fantasy Film-Noir Horror Musical Mystery Romance Sci-Fi Thriller War Western

1.2. Objective

The goal of the project is to build a movie recommendation system using machine learning such that the model reaches the targeted accuracy.

In order to achieve this objective, we employ the following algorithms:

We will use prediction models to train a recommendation machine learning algorithm to predict a movie recommendation by using historical ratings of users to movies in the dataset. To test the success of our recommendation system we will use the Residual Mean Square Error (RMSE) to evaluate the accuracy of the algorithm. RMSE is one of the preferred methods that measure the differences between values predicted by a model and the values observed, a lower RMSE is better than a higher one.

The prediction models that will be used are : • Simple prediction model • Prediction model with movie bias effect • Prediction model with movie and user bias effect • Regularisation model (Regularisation allows for reduced errors caused by movies with few ratings which can influence the prediction and skew the error metric. The method uses a tuning parameter, to minimise the RMSE.)

1.3. Key Steps

1.3.1 Library

In this project, we have incorporated several essential libraries to facilitate our data analysis and model building. These libraries include:

library(dplyr) library(ggplot2) library(tidyverse) library(caret) library(wordcloud) library(knitr) library(MLmetrics) library(tinytex)

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

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

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

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(tidyverse)
library(caret)
library(dplyr)
library(ggplot2)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```r
library(knitr)
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
##
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
##
## The following object is masked from 'package:base':
##
##     Recall
```

```r
library(tinytex)
```

1.3.2. Data Preparation

The Data Preparation process consists of two primary activities: Loading Data and Data Wrangling. First, we will downLoad the Data. Following this, we will perform the Data Wrangling, where we perform data cleaning, transformation, and structuring tasks, ensuring that the data is in an optimal and reliable state for further analysis and model development.

i. Load Data

The MovieLens data is downloaded using the following code to generate the datasets.

MovieLens 10M dataset:

https://grouplens.org/datasets/movielens/10m/

http://files.grouplens.org/datasets/movielens/ml-10m.zip

```r
options(timeout = 120)
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
unzip(dl, ratings_file)
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
unzip(dl, movies_file)
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE), stringsAsFa
```

```r
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>% mutate(userId = as.integer(userId), movieId = as.integer(movieId), rating = as.nu
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE), stringsAsFact
```

```r
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>% mutate(movieId = as.integer(movieId))
movielens <- left_join(ratings, movies, by = "movieId")
```

Confirming availability of combined dataset as desired

```r
str(movielens)
```

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

'data.frame': 10000054 obs. of 6 variables: $ userId : int 1 1 1 1 1 1 1 1 1 1 … $ movieId : int 122 185 231 292 316 329 355 356 362 364 … $ rating : num 5 5 5 5 5 5 5 5 5 5 … $ timestamp: int 838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 838984885 838983707 … $ title : chr "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" … $ genres : chr "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"

ii. Data Wrangling As the data type of the variable are as desired we do not need to go through the Data wrangling exercise.

2. Methods / Analysis :

2.1. Process and techniques used : Training and testing of the Algorithm :

The 10M version of the MovieLens dataset that we are using will be divided into two sets, edx and final_holdout_test. The edx will be further divided into two, one for building the algorithm and another for testing the algorithm. The accuracy will then be tested using the final_holdout_test set.

Final hold-out test set will be 10% of MovieLens data

```r
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

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

```r
# Warning message in set.seed(1, sample.kind = "Rounding") :
# non-uniform 'Rounding' sampler used
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set

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

Add rows removed from final hold-out test set back into edx set

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

```
## Joining with ‘by = join_by(userId, movieId, rating, timestamp, title, genres)‘
```

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

Joining with by = join_by(userId, movieId, rating, timestamp, title, genres)

removing variables from workspace

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

Dividing edx further into two, one for building the algorithm and another for testing the algorithm

Creating train and test sets

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

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

```
# Warning message in set.seed(1, sample.kind = "Rounding"):
# "non-uniform 'Rounding' sampler used"
test_index <- createDataPartition(y = edx$rating, times=1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]
# Matching userId and movieId in both train and test sets
test_set <- temp %>% semi_join(train_set, by = "movieId") %>% semi_join(train_set, by = "userId")
# Adding back rows into train set
removed <- anti_join(temp, test_set)
```

```
## Joining with ‘by = join_by(userId, movieId, rating, timestamp, title, genres)‘
```

Adding back rows into train set

```
train_set <- rbind(train_set, removed)
# Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

removing variables from workspace

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

2.2 Exploratory Data Analysis : We start by analysing the data structure.

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

'data.frame': 9000055 obs. of 6 variables: $ userId : int 1 1 1 1 1 1 1 1 1 1 ... $ movieId : int 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 838983707 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|Adventure|Sci-Fi"

We find that there are 9000055 observations and 6 columns. The observations can be understood to be the rating given by one user for one movie. The six columns are userId, movieId, rating, timestamp, title and genres. An understanding of the timestamp tells us that it represents seconds since midnight UTC January 1, 1970. For a further understanding of the data we will analyse based on the summary of the data other than the column "genre".

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

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

userId movieId rating timestamp

Min. : 1 Min. : 1 Min. :0.500 Min. :7.897e+08
1st Qu.:18124 1st Qu.: 648 1st Qu.:3.000 1st Qu.:9.468e+08
Median :35738 Median : 1834 Median :4.000 Median :1.035e+09
Mean :35870 Mean : 4122 Mean :3.512 Mean :1.033e+09

3rd Qu.:53607 3rd Qu.: 3626 3rd Qu.:4.000 3rd Qu.:1.127e+09
Max. :71567 Max. :65133 Max. :5.000 Max. :1.231e+09
title
Length:9000055
Class :character
Mode :character

An anlysis of the summary reveals that : i. There are 71567 users ii. There are 65133 movieid iii. The rating ranges from 0.5 to 5.0 iv. The total titles are 9000055

The difference in number of users and movies can be attributed to certain movies being rated multiple times. Finding out the unique number of users and movie can make things clear.

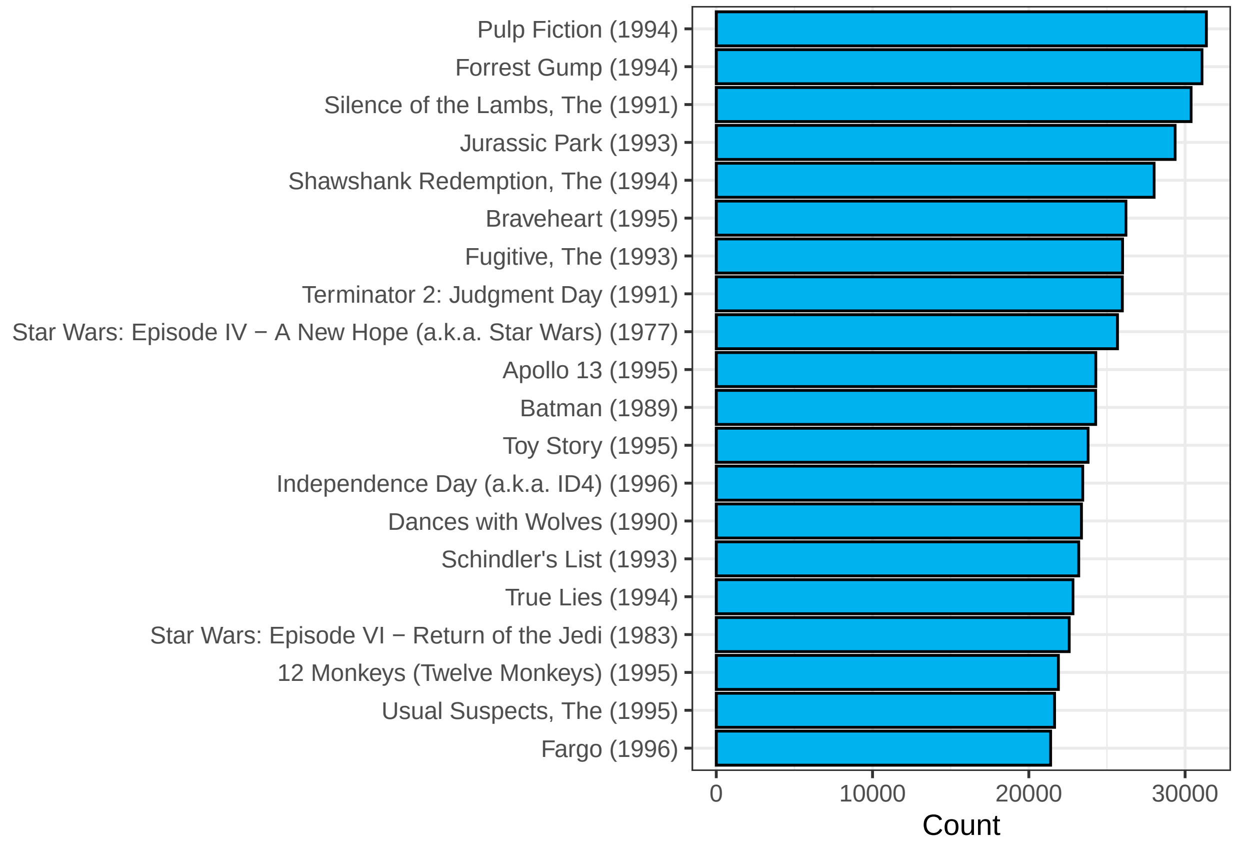
```
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

A data frame 1 * 2
n_users n_movies 69878 10677

The unique movies only being 10677 means that the movies were rated by many users. It also means that certain movies are more popular and therefore are rated by many users. We will analyse the popularity of the movies by finding out the top movies based on their popularity i.e the number of times the movies are rated. We will find out the top 20 such movies and draw a bar plot to visualize it.

```
edx %>% group_by(title) %>% summarize(count = n()) %>% arrange(-count) %>% top_n(20, count) %>% ggplot(
```

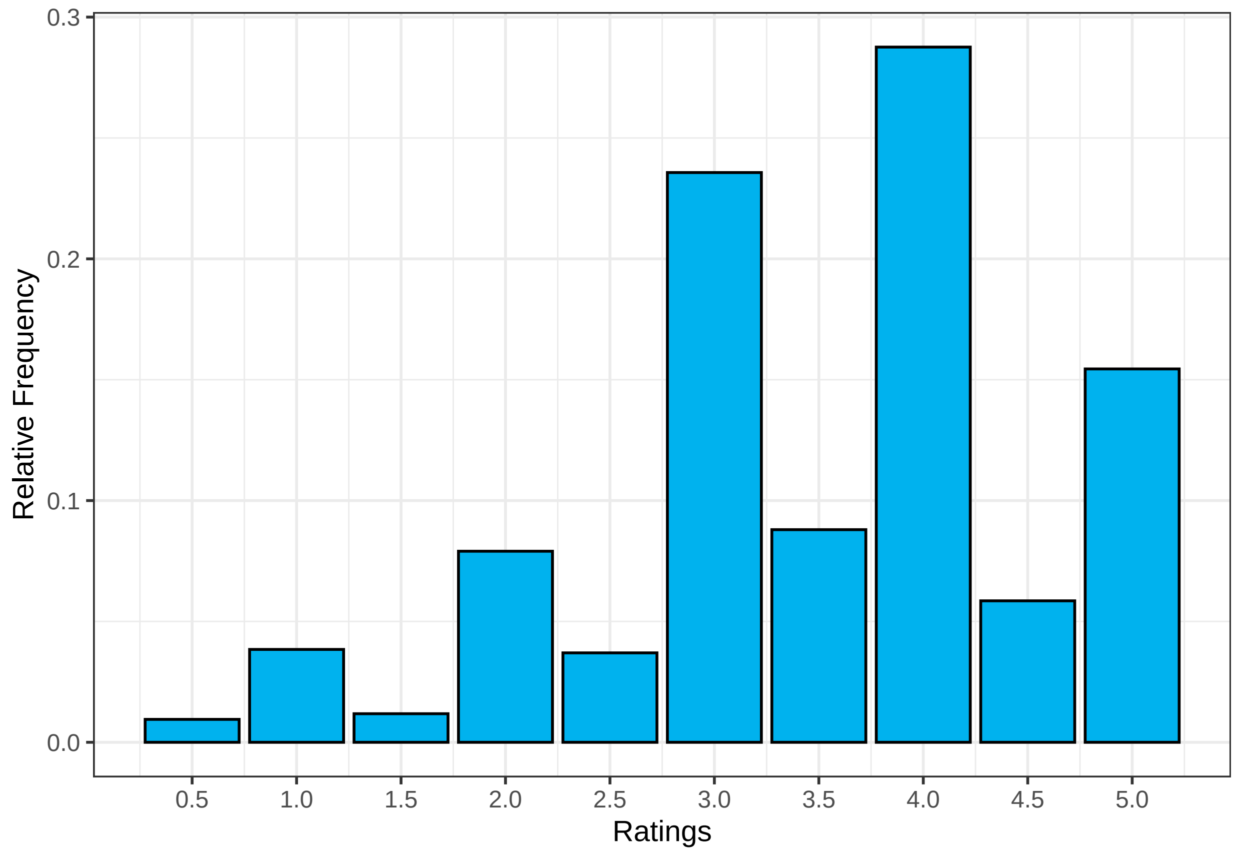


Certain movies like Pulp Fiction, Forrest Gump and Silence of the Lambs have received more than 30000 ratings.

Although while analysing the summary we have observed that the ratings range from 0.5 to 5.0 but we need to analyse what is the kind of rating given by the users i.e the most given ratings. We will draw a bar plot to visualize the pattern. As the observations are large we will use the relative frequency for the purpose of analysis.

```
edx %>% ggplot(aes(rating, y = ..prop..)) + geom_bar(color = "black", fill = "deepskyblue2") + labs(x =
```

```
## Warning: The dot-dot notation (`..prop..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(prop)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



The ratings given in the order of most to least is as under : 4.0, 3.0, 5.0, 3.5, 2.0, 4.5, 1.0, 2.5, 1.5, 0.5 No movie has been rated 0 (zero) by any of the users. Maximum users have given the rating 4.0 followed by 3.0 and 5.0. The middle ranges i.e. 0.5, 1.5, 2.5, 3.5, 4.5 have been given less. We can assume that there is a propensity of those giving rating to rate in whole numbers.

When we analysed the data structure, we found that the movies are classified under different set of genres. We will find out the movie ratings in every genres.

```
top_genr <- edx %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>% summarize(count = n()) %
top_genr
```

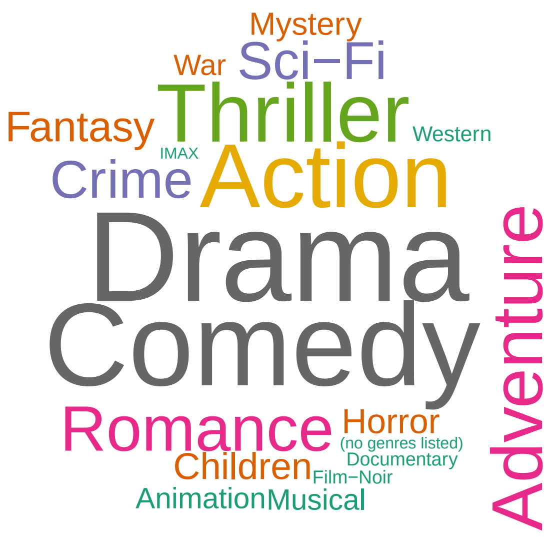
```
## # A tibble: 20 x 2
##    genres                count
##    <chr>                 <int>
##  1 Drama               3910127
##  2 Comedy              3540930
##  3 Action              2560545
##  4 Thriller            2325899
##  5 Adventure           1908892
##  6 Romance             1712100
##  7 Sci-Fi              1341183
##  8 Crime               1327715
##  9 Fantasy              925637
## 10 Children             737994
## 11 Horror               691485
## 12 Mystery              568332
## 13 War                  511147
## 14 Animation            467168
## 15 Musical              433080
## 16 Western              189394
## 17 Film-Noir            118541
## 18 Documentary           93066
## 19 IMAX                   8181
## 20 (no genres listed)        7
```

A tibble: 20 × 2

genres count Drama 3910127 Comedy 3540930 Action 2560545 Thriller 2325899 Adventure 1908892 Romance 1712100 Sci-Fi 1341183 Crime 1327715 Fantasy 925637 Children 737994 Horror 691485 Mystery 568332 War 511147 Animation 467168 Musical 433080 Western 189394 Film-Noir 118541 Documentary 93066 IMAX 8181 (no genres listed) 7

Using word cloud we can also visualise the movie ratings with most genres.

```
pal <- brewer.pal(8, "Dark2")
top_genr %>% with(wordcloud(genres, count, max.words = 50, random.order = FALSE, colors = pal))
```



As can be seen from the word cloud also the genre "drama" is one with the most movie ratings followed by "comedy", "action", "thriller", "adventure", "romance", sci-fi.

2.3 Modelling approach : Models / Algorithms :

Several models will be assessed starting with the simplest. Accuracy will be evaluated using the residual mean squared error (RMSE) or in other words to evaluate how close our predictions are to the true rating values in the final_holdout_test set. For this, we take into account the Root Mean Square Error (RMSE).

To construct the RMSE, we need to determine the residuals. Residuals are the difference between the actual values and the predicted values. We denote them by y_hat u,i-yu,i where yu,i is the observed value for the ith observation and y_hat u,i is the predicted value. They can be positive or negative as the predicted value under or over estimates the actual value. Squaring the residuals, averaging the squares, and taking the square root gives us the RMSE. We then use the RMSE as a measure of the spread of the y values about the predicted y value.

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

2.3.1 Model 1 The first model is the simplest approach we could take in making a prediction by only using the average of all movie ratings. With differences explained by random variation. Yu,i = mu + Epsilon u,i Epsilon u,i is defined as the independent sample errors and mu the true rating for all movies. Statistical theory tells us that the estimate that minimizes the RMSE is the least squares estimate of mu. For this exercise it is the average of all ratings.

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

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

[1] **3.512456**

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

```r
model1_rmse <- RMSE(test_set$rating, mu_hat)
results = tibble(Method = "Model 1: Simply the mean", RMSE = model1_rmse)
results %>% knitr::kable()
```

Method	RMSE
Model 1: Simply the mean	1.060054

Method	RMSE
Model 1: Simply the mean	1.060054

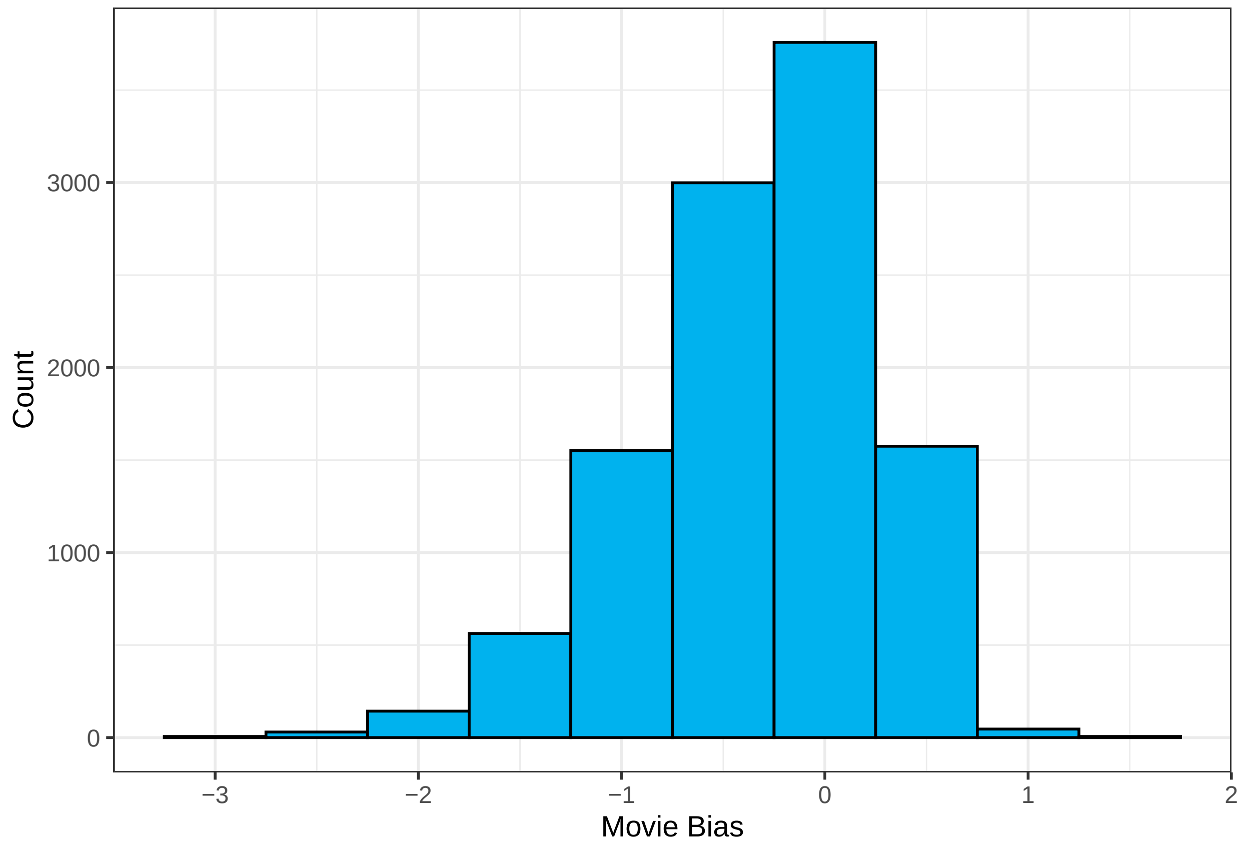
Predicting the mean gives us a rather higher RMSE over 1 which is far from ideal and does not help us in building a recommendation system. Using our earlier insights of the dataset allows us a far more advanced analysis and rating predictions with hopes of achieving a smaller error. It was a first simple attempt that lacks accuracy. Our next model will build on this.

2.3.2 Model 2 Our first model can be improved on by taking into account movie bias. We know from experience, and data confirms this, that some movies are more popular than others and receive higher ratings. We can add the term bi to reflect this. It is the average of Yu,i-mu_hat for each movie i. Yu,i = mu + bi + Epsilon u,i

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

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

```r
bi %>% ggplot(aes(b_i)) + geom_histogram(color = 'black', fill = 'deepskyblue2', bins = 10) + xlab('Movi
```



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

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

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

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

2.3.3 Model 3 Bias can be found in users as well.

Some tend to rate more positively and others negatively. We can add this effect to the model as bu Yu,i = mu + bi + bu + Epsilon u,i We can estimate b_hat u as the average of Y u,i - mu - b_hat i

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

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

```
predicted_ratings <- test_set %>% left_join(bi, by = 'movieId') %>% left_join(bu, by = 'userId') %>% mu
u_bias_rmse <- RMSE(predicted_ratings, test_set$rating)
results <- bind_rows(results, tibble(Method = 'Model 3: Mean + movie_bias + user effect', RMSE = u_bias
results %>% knitr::kable()
```

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

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

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

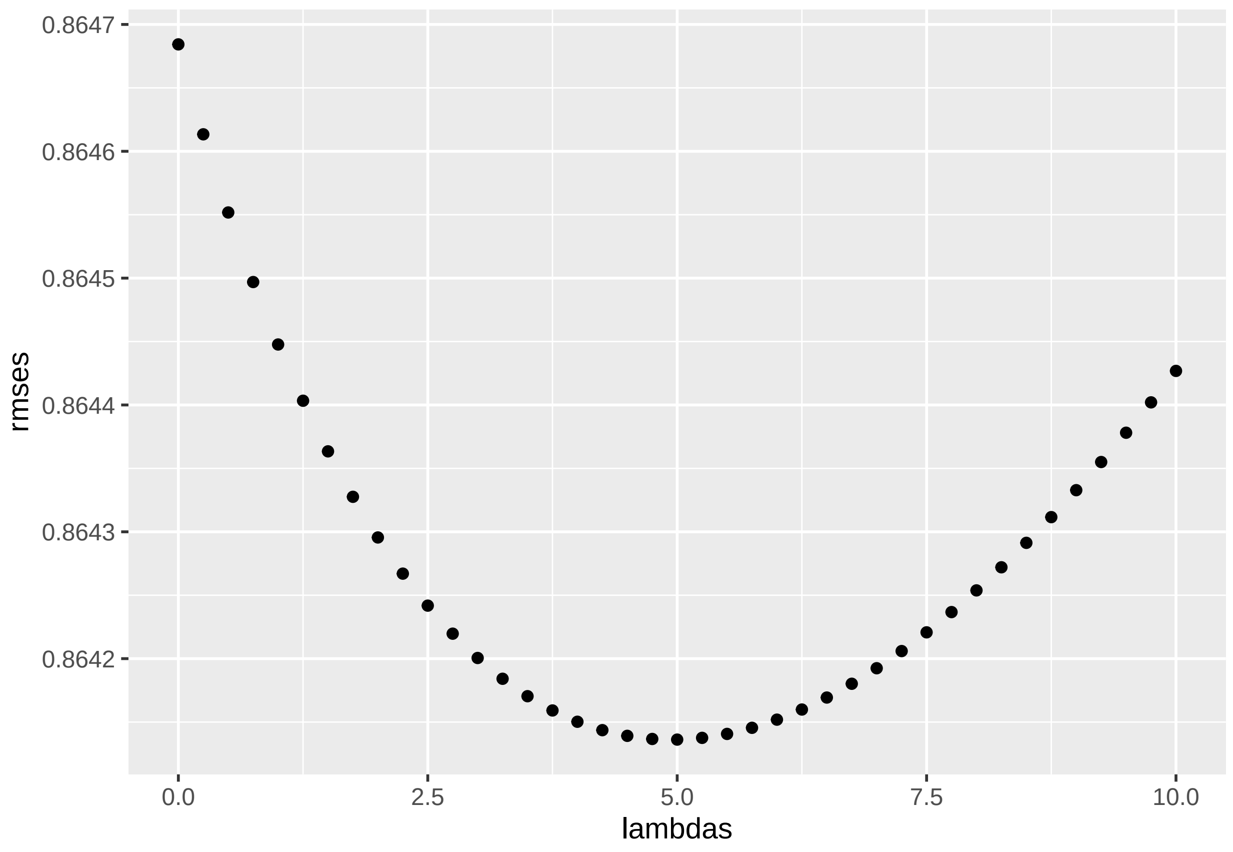
2.3.4 Model 4 Some of the data is noisy. For example ratings on obscure or niche movies by only a few users.

This adds variability and can increase RMSE. We can use regularization to penalize large estimates formed by small sample sizes to reduce this effect. The optimal penalty to use, lambda, can be found using cross-validation and applied to our model.

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

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

```
lambda_data <- data.frame(lambdas, rmses)
ggplot(data = lambda_data) +
    geom_point(aes(x = lambdas, y = rmses))
```



```r
rmse_regularisation <- min(rmses)
rmse_regularisation
```

```
## [1] 0.8641362
```

0.8641362

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

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

5

RMSE result:

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

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

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

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

3 Results :

A machine learning algorithm to predict the ratings from the MovieLens dataset was constructed. The optimal model incorporated the effect of both user and movie biases and these variables were regularised to incorporate movies with a low number of ratings. The aim of the project was to develop an algorithm with RMSE lower than 1 which was achieved by using the Movie and User effects alongwith Regularisation. The lowest value of RMSE obtained via this model was 0.8641362.

3.1 Validation of results :

Validating our preferred model using the final_holdout_test set

Now we use the Regularised, Movie and User Effects Model algorithm on the final_holdout_test set to validate the success of our findings. We compare the RMSE obtained versus 'Simply the mean' RMSE to give it some context.

Prediction based on Mean Rating on the final_holdout_test set

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

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

3.512465

```r
final_model1_rmse <- RMSE(final_holdout_test$rating, mu_hat)
final_model1_rmse
```

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

1.061202

Saving the results

```
final_rmse_results = tibble(Method = "Simply the mean", RMSE = final_model1_rmse)
final_rmse_results %>% knitr::kable()
```

Method	RMSE
Simply the mean	1.061202

Method	RMSE
Simply the mean	1.061202

Predict via regularisation, movie and user effect model.

We will use the minimum lambda value from the earlier models on the final__holdout__test set as it was shown to be the best tune.

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

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

3.512465

```
b_i <- edx %>% group_by(movieId) %>% summarise(b_i = sum(rating - mu) / (n() + min_lambda))
b_u <- edx %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>% summarise(b_u = sum(rating - b_i
```

```
predicted_ratings <- final_holdout_test %>% left_join(b_i, by = "movieId") %>% left_join(b_u, by = "use
final_rmse_model <- RMSE(final_holdout_test$rating, predicted_ratings)
final_rmse_model
```

```
## [1] 0.8648177
```

0.8648177

Save results in Data Frame

```r
final_rmse_results <- bind_rows(final_rmse_results, tibble(Method = "Regularised movie and user effects
final_rmse_results %>% knitr::kable()
```

Method	RMSE
Simply the mean	1.0612018
Regularised movie and user effects	0.8648177

Method	RMSE
Simply the mean	1.0612018
Regularised movie and user effects	0.8648177

4. Conclusion and Final thoughts

4.1 Brief summary We constructed a machine learning algorithm to predict the ratings from the MovieLens dataset. The optimal model incorporated both the user effect and movie biases and these variables were regularised to incorporate movies with a low number of ratings. The goal of the project was to build a movie recommendation system using machine learning such that the model reaches the targeted accuracy. The lowest value of RMSE obtained via this model was 0.8641362. The selected model has a RMSE of 0.8648242 using the final_holdout_test set and has therefore achieved the goal we had set. This model significantly improves upon the benchmarking model's, 'Simply the mean' RMSE of 1.06 and can be suggested as a recommended model.

4.2 Limitations and future work The RMSE has improved while moving from model_1 to model_2 and then to model_3 but on moving from model_3 to model_4 we observe a very small gain in accuracy. It will take a different approach to improve it significantly. While the final model, which was comprised of User effects, Movie bias and with Regularisation performed well, there are additional biases that could be explored to further improve the accuracy of the model, like year of release, genre, sub genres etc. More advanced models featuring Random Forests and those based on Singular Value Decomposition (made famous by Simon Funk during the Netflix Challenge) generate higher levels of accuracy, though they are also far more computationally intensive. It is likely that future iterations of models involving k-Nearest Neighbours and Collaborative Filtering with cosine similarity promise to continue making strides and improve the overall user experience for streamers everywhere.