Data Science Capstone - Movielens Recommendation Model

Carlos Yáñez Santibáñez

February 21, 2020

Abstract

Abstract

Contents

1 Introduction

This document presents a machine learning model that aim to predict (recommend) movie ratings for particular users of a streaming or review platform. This report is a capstone assignment for HarvardX's Professional Certificate in Data Science, which can be taken at the edX platform. The program is available at https://www.edx.org/professional-certificate/harvardx-data-science.

The data used in this excercise comes from the Movielens 10M Dataset. After using the donwload code provided in the course, the resulting dataframe contains observations with an individual movie rating from a particular user, each with the below attributes:

1. **userId** : Unique user identifier
2. **movieId** : Unique movie identified
3. **rating** : Rating given to this movie by the particular user.
4. **timestamp** : Timestamp indicating when the the user submitted the rating.
5. **title** : Title of the film and its release year in brackets. Please note that different movies can have the same name (e.g. remakes), thus movieId is a better unique identifier.
6. **genres** : List of all genres in which this movie can be clasiffied.

The goal of this project is to generate a model that can predict a particular movie rating for each user, as close as possible to the actual rating given to each film. In order to assess the model performance, the **Root Square Mean Error (RMSE)** will be calculated for a validation dataset. Training and validation dataset are generated using the code provided in the assignment instructions. In order to aim for a full mark, this report will target for a RMSE lower than **0.86490**.

The starting point of this project is the model presented in section 33.7 of the course's textbook. From there, the following steps are presented:

1. Analysis of the textbook's model and possible ways to improve it.
2. Improvement to the model via user clustering.
3. Tuning improved model.
4. Evaluation against validation dataset.
5. Conclusion.

The following sections present the above points in further detail.

2 Methods and Analysis

2.1 Preparing the data

As mentioned in the introduction, this reports starts with the model presented in the textbook and then explore options to improve segmenting the users. However before conducting any modelling, the data needs to be cleaned up a little bit and then split into training and testing set.

In terms of data cleaning, three operations have been considered, namely:

- Remove the year from the title and storing in a different column. This may be useful for modelling.
- For the same reasons, convert the timestamp into a year number.
- Finally, a sequential number will be added (*row_id*). This will be create a single unique ID for each record (instead of a userId and movieId combination) and maybe useful to speed up filtering, given the large size of the dataset.

For this purposes, the below function **Tidy_Up** has been created. The code for this function is available on the included R file. Using the below code the train and test sets are created.

```r
#tidy up data
edx_tidy_temp <- Tidy_Up(edx)

#divide Train and Test datasets
set.seed(200, sample.kind="Rounding")
edx_train_test <- Train_Test(edx_tidy_temp)

#remove temporary and original dataset to avoid filling up memory.
rm("edx_tidy_temp","edx")
```

Below, this is a sample of the generated training set (*edx_train_test$train*).

Table 1: Training Dataset - Sample

userId	movieId	rating	timestamp	title	genres	release_year	rating_date	rating_year	row_id
1	122	5	838985046	Boomerang	Comedy|Romance	1992	1996-08-02	1996	1
1	185	5	838983525	Net, The	Action|Crime|Thriller	1995	1996-08-02	1996	2
1	292	5	838983421	Outbreak	Action|Drama|Sci-Fi|Thriller	1995	1996-08-02	1996	3
1	316	5	838983392	Stargate	Action|Adventure|Sci-Fi	1994	1996-08-02	1996	4
1	329	5	838983392	Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	1994	1996-08-02	1996	5
1	355	5	838984474	Flintstones, The	Children|Comedy|Fantasy	1994	1996-08-02	1996	6
1	356	5	838983653	Forrest Gump	Comedy|Drama|Romance|War	1994	1996-08-02	1996	7
1	362	5	838984885	Jungle Book, The	Adventure|Children|Romance	1994	1996-08-02	1996	8
1	364	5	838983707	Lion King, The	Adventure|Animation|Children|Drama|Musical	1994	1996-08-02	1996	9
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult	Action|Comedy	1994	1996-08-02	1996	10

2.2 Textbook Model

Once completed data cleaning and split, the first step is to implement the model presented in the textbook and asses it's results. This model estimates rating by assuming that a particular rating from a particular user can be calculated as a deviation (bias) from the average rating for all movies (the *true* rating). In order to account for cases where movies and users don't have many reviews against them, weighting parametres have been added. This can be expressed by the below equation:

$$predicted\ rating = \hat{\mu} + b_i + b_u$$

where $\hat{(\mu)}$ is the average rating for all movies in the dataset, and b_i and b_u being the movie bias and user bias (respectively), defined as follows:

$$b_i = \frac{1}{\lambda_1 + m} \sum_{j=1}^{m} (rating_j - \hat{\mu})$$

$$b_u = \frac{1}{\lambda_2 + n} \sum_{j=1}^{n} (rating_j - \hat{\mu} - b_{i,j})$$

where * $rating_j$:a particular movie rating * m : the number of all ratings for a particular movie * n : the total number of ratings submitted from a particular user (one rating per movie). * $ b_{i,j}$: the *movie bias* for a particular movie * λ_1 : weighting parametre for *movie biases.* * λ_2 : weighting paramete for *user biases.*

The calculation of these parametres has been written into the function *General_Biases*. Then, these results are used to create the first set of predictions, setting the tuning parametres to zero. The results are evaluated using the provided *RMSE* function.

```r
# create biases

general_biases <- General_Biases(edx_train_test$train)

# join with dataset and predict

data_set <- edx_train_test$test  %>%
  left_join(general_biases$movie_avgs, by = "movieId") %>%
  left_join(general_biases$user_avgs, by = "userId")

first_prediction <- data_set %>%
    mutate(predicted_rating = general_biases$mu_hat + b_i + b_u, error = rating - predicted_ra
    filter(!(predicted_rating == general_biases$mu_hat))

# Calculate RMSE and put it in comparison table
rmse_value <- RMSE(first_prediction$rating,first_prediction$predicted_rating)
compare_RMSE <- tibble(Model=character(),RMSE = double(),
                       "Below Target"=logical())
compare_RMSE <- add_row(compare_RMSE,
                        Model="Textbook",
                        RMSE=rmse_value,
                        "Below Target"=(rmse_value<target_rmse))

# clean up
```

```r
general_biases_first <- general_biases
rm(first_prediction,data_set,rmse_value)
```

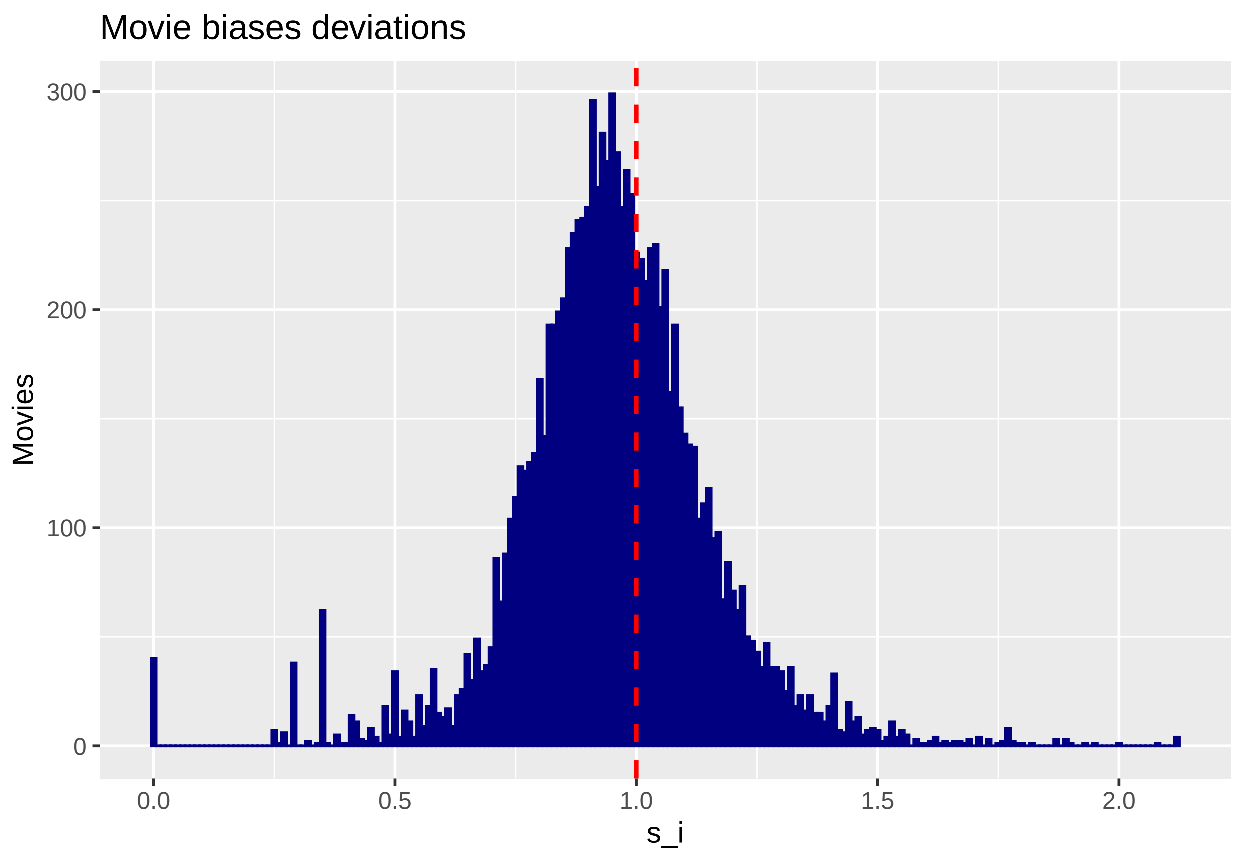
Table 2: First Prediction

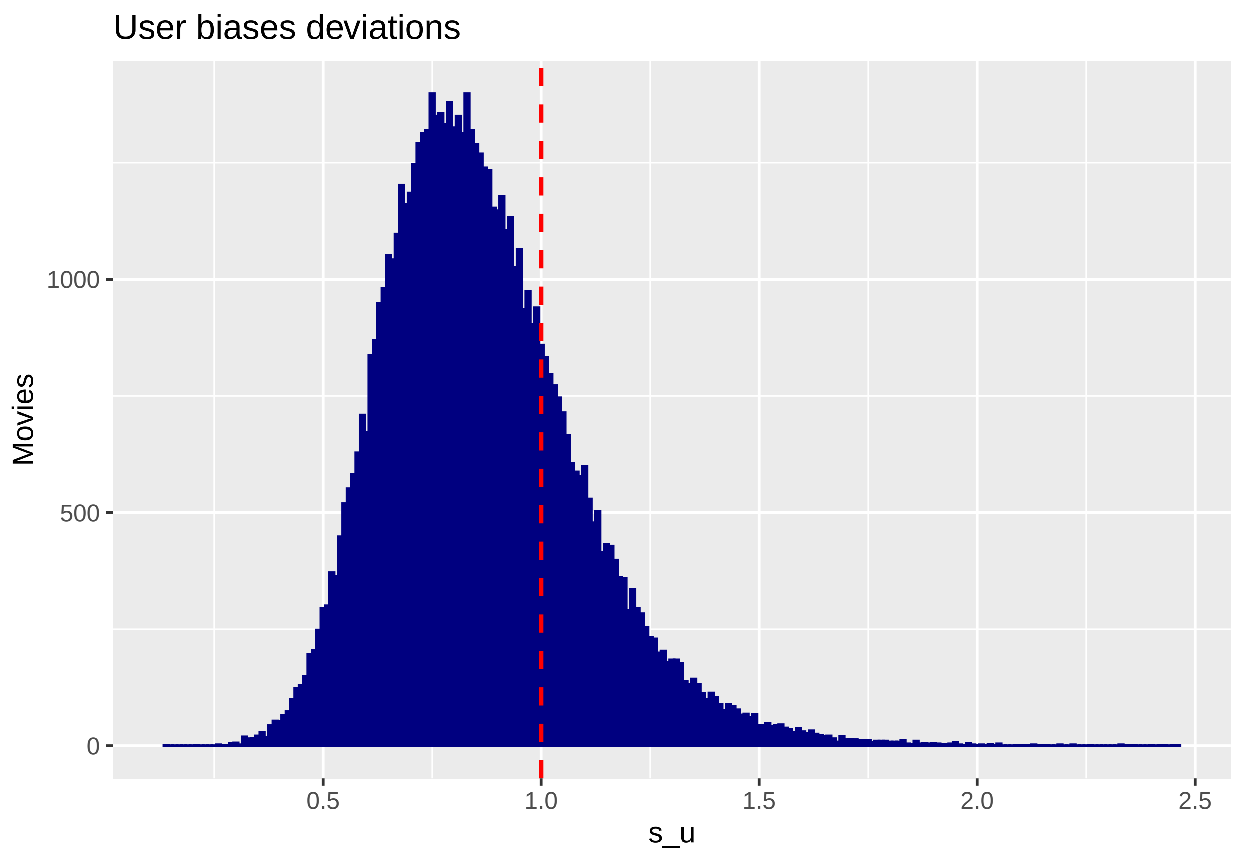
Model	RMSE	Below Target
Textbook	0.8663964	FALSE

As expected, the RMSE is higher than the target value. After thinking about the possible causes of this, part of this may be related to the fact that this model is based on averages. Although this is not necessarily a bad approach and works for many cases, it also has some limitations, namely:

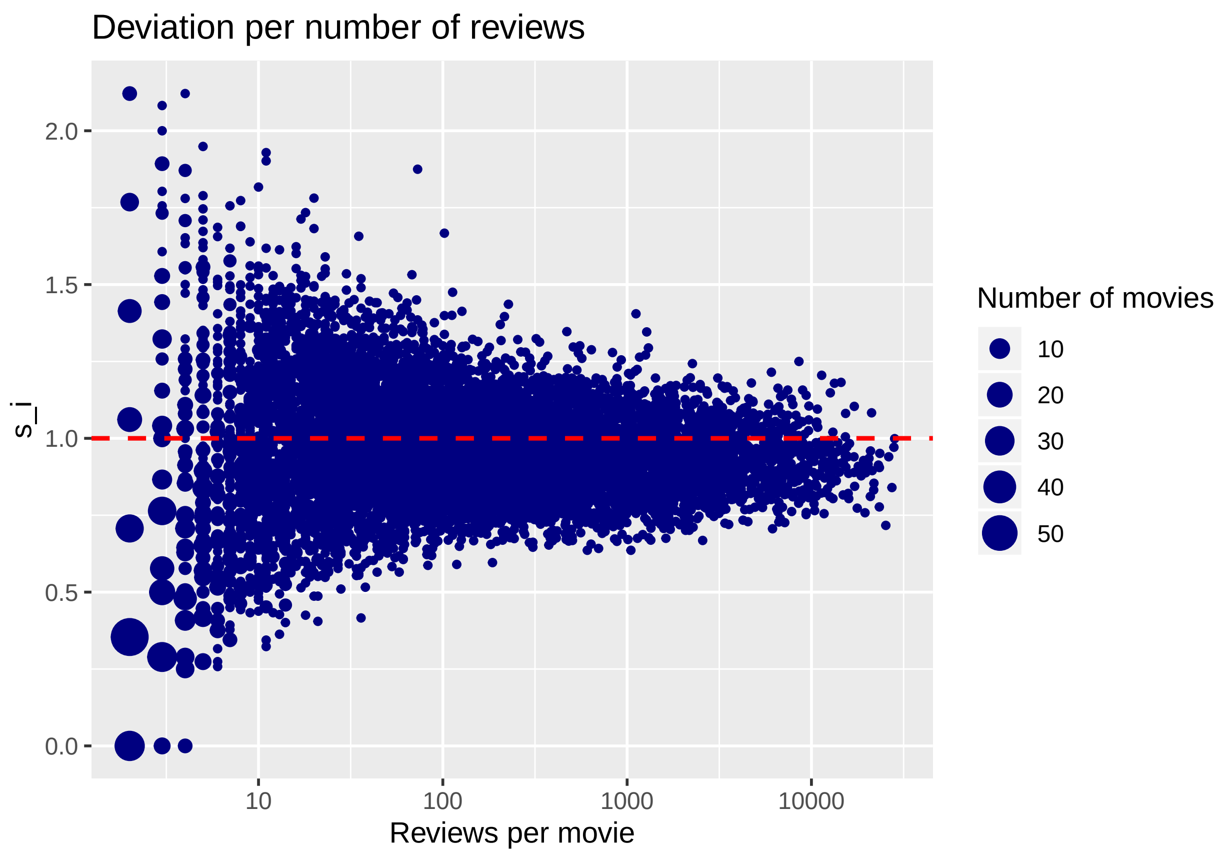
- it won't perfom well with *divisive* movies. For example a niche film will obtain 5 stars from the fans, but will be many users which will disklike. This won't be reflected by an average bias (and the user bias may not necessarily compensate this).
- There will be users with extrem ratings (i.e. who like family movies and dislike horror films). The user bias won't reflect this.
- Already mentioned in the textbook, when the number of reviews for a user or a film is low , this model won't predict accurately.

Taking this into account, the *General_Biases* function also calculates the standard deviation of the movie and user biases (named *s_i* and *s_u* respectively). The below charts show their results

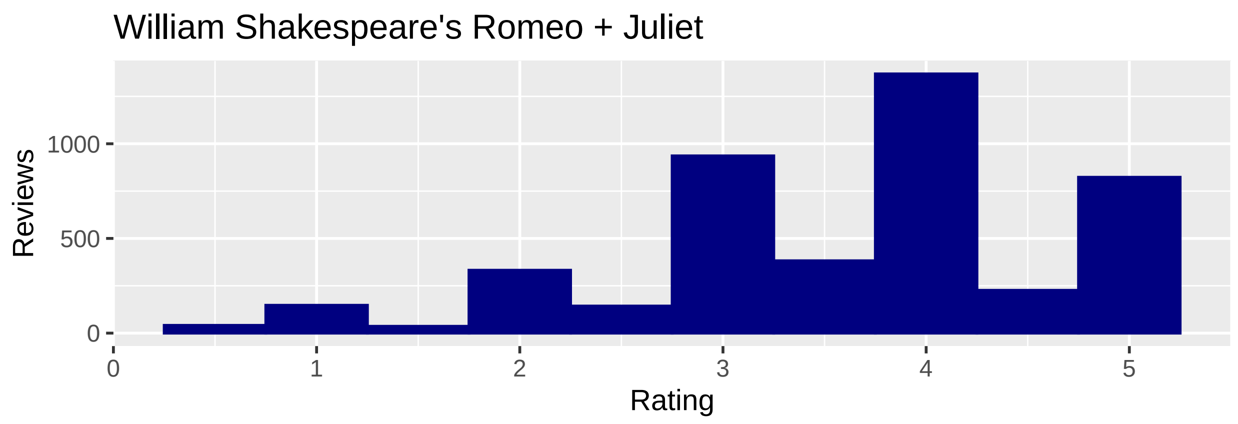


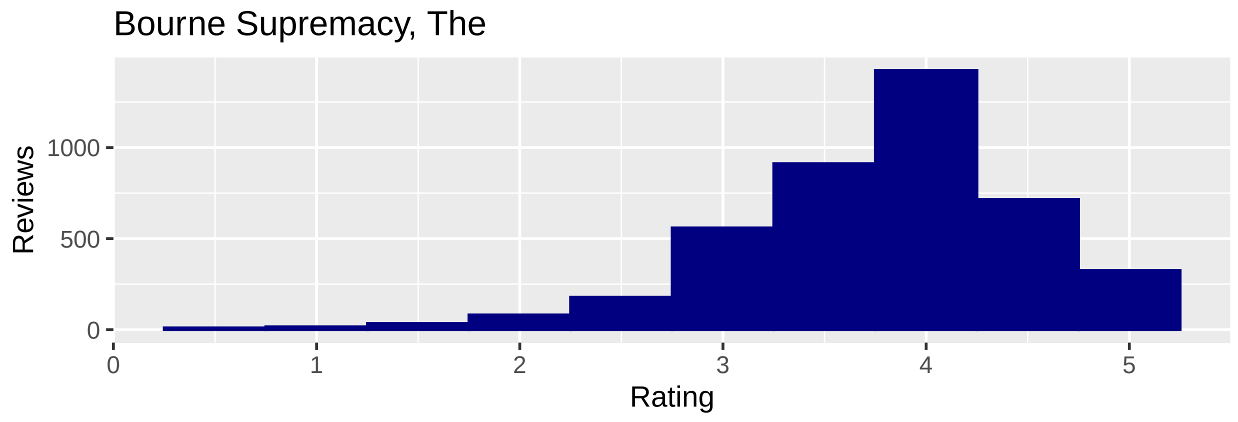


As shown, there is a significant percentage of movie (37.84 %) and user (24.03 %) biases equal or larger than 1. This means there will be many cases where this model with values contributing to a larger RMSE. Although movies/user with lower number of reviews have higher deviations, this is not always the case, as illustrated in the below graph.



Perhaps, a way to illustrate this is to look at two films with similar number of reviews and average rating (and close b_i) but different standard deviations for their ratings.





Looking at the charts above, it looks like *Romeo+Juliet* has a wider range of opinions, compared to a more concentrated range of ratings in the case of the *Bourne Supremacy*, which in turns means the model won't be as effective, especially for the 7 908 users who have rated both films and may be in all sections of liking/disliking this particular combination of films, which is not captured in the existing model.

2.3 An Improved Model : User Clustering

All the previous observations notwithstanding, it is still worth pointing that the above model has merit, given its relative simplicity and ease to calculate. In addition, in the textbook, this model shows a good performance with a dataset considerably smaller than 8+ million observations.

Perhaps, 8 million reviews is indeed a large number of reviews and it is possible to generate ratings based on smaller groups. This gives the opportunity to segment the data into smaller groups, where deviations are smaller and therefore have a lower prediction error.

One approach is to cluster the users according their similar movie tastes (i.e split the ones who like Shakespearean romances from the fans of action thrillers). The way used in this report classify the users based on the genres of the films they have rated.

As mentioned before, each film has a set of genres they belong (from one to many). Collectively, there are 20 genres, listed below.

Table 3: Genres in training set, ordered by ocurrences

genres	n_t
Drama	3 519 116
Comedy	3 187 491
Action	2 303 879
Thriller	2 091 981
Adventure	1 717 390
Romance	1 541 444
Sci-Fi	1 206 991
Crime	1 194 987
Fantasy	833 007
Children	664 073
Horror	622 441
Mystery	511 345
War	460 314
Animation	420 486
Musical	389 793
Western	170 421
Film-Noir	106 798
Documentary	83 754
IMAX	7 357
(no genres listed)	6

In order to profile each user, this report has used the genre in two ways:

- The average rating each user gives to each genre, for each film rated (i.e. how much each user likes each genre).
- The percentage of each genre compared to the overall reviews per user (i.e. how much each user watches each genre).

This will result in a *user_vector*, profiling each user in the training set. This algorithm, has been written int the *User_Vectoriser* function, present in the attached R file. The below table shows a sample of the user vectors created by this code.

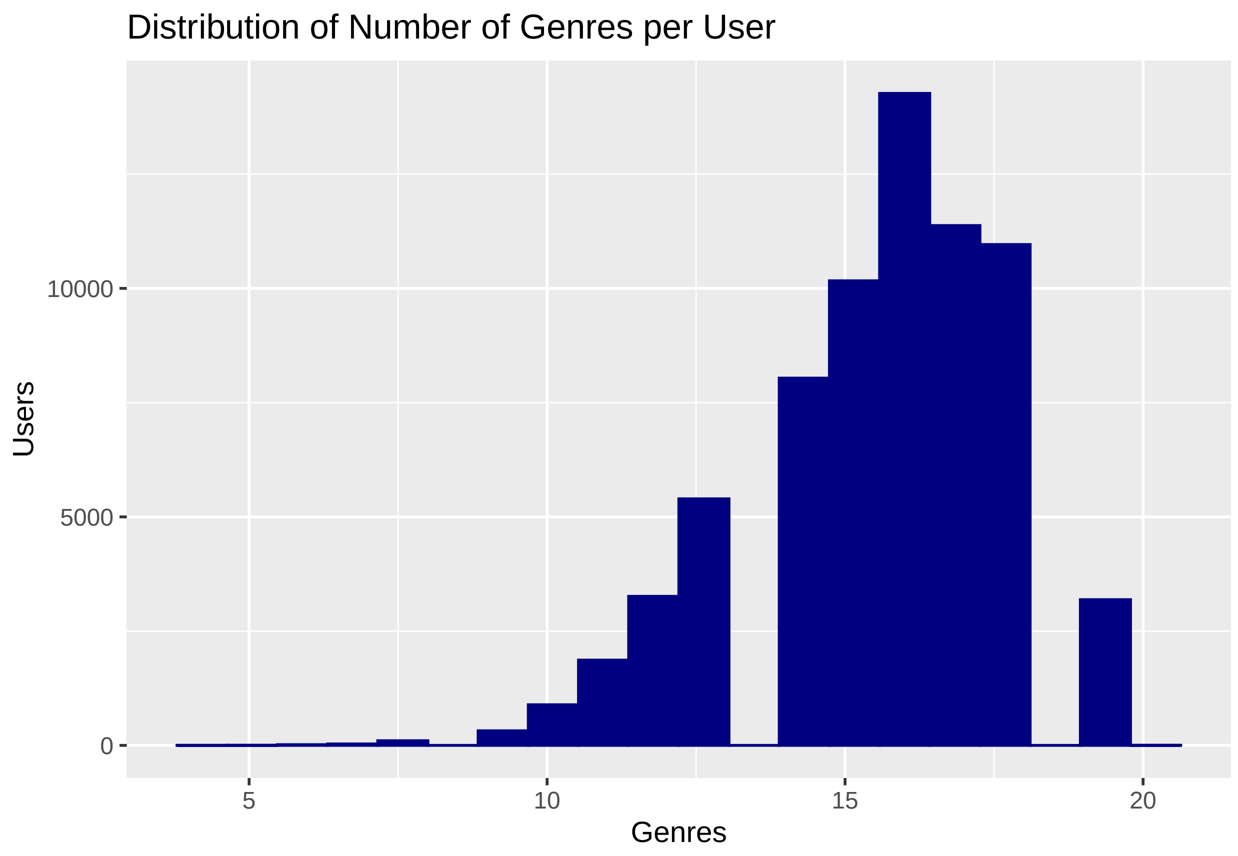
Table 4: User Vectors - Averages

userId	Drama	Comedy	Action	Thriller	Adventure	Romance	Sci-Fi	Crime	Fantasy	Children	Horror	Mystery	War	Animation	Musical	Western	Film-Noir	Documentary	IMAX	(no genres listed)
1	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	0.000	0.000	5.000	5.000	5.000	0.000	0.000	0	0	0
2	3.250	3.000	3.333	2.833	3.250	2.667	3.600	0.000	3.000	3.000	0.000	2.000	4.000	0.000	3.000	0.000	0.000	0	0	0
3	4.176	3.600	3.857	4.083	3.700	4.150	0.000	4.000	4.167	4.000	0.000	4.500	4.000	4.000	3.000	3.750	4.000	0	0	0
4	4.667	3.846	4.143	4.111	4.778	3.857	4.667	4.200	4.500	5.000	0.000	0.000	5.000	5.000	5.000	5.000	0.000	0	0	0
5	4.098	3.609	1.500	4.091	3.000	3.737	3.000	4.500	3.333	3.333	4.400	4.250	3.800	1.000	4.000	0.000	5.000	0	0	0
6	4.429	3.909	3.731	3.643	3.857	4.333	4.154	3.500	4.000	0.000	1.000	4.000	4.000	3.000	0.000	5.000	4.000	0	0	0
7	4.000	3.479	3.385	3.972	3.733	3.607	3.000	4.214	2.750	3.375	4.000	4.342	5.000	3.333	4.750	3.462	4.367	3	0	0
8	3.409	3.405	3.475	3.491	3.474	3.221	3.522	3.539	3.382	3.243	3.387	3.595	3.543	3.543	3.133	3.000	3.500	4	0	0
9	4.136	4.100	4.500	4.833	3.000	3.875	4.500	4.917	3.500	0.000	0.000	5.000	3.250	0.000	3.500	0.000	5.000	0	0	0
10	3.940	4.053	3.857	3.538	3.400	3.943	5.000	3.500	4.000	3.667	2.500	3.778	4.333	3.000	0.000	4.000	4.000	0	0	0

Table 5: User Vectors - Percentages

userId	Drama	Comedy	Action	Thriller	Adventure	Romance	Sci-Fi	Crime	Fantasy	Children	Horror	Mystery	War	Animation	Musical	Western	Film-Noir	Documentary	IMAX	(no genres listed)
1	0.11321	0.15094	0.15094	0.05660	0.07547	0.07547	0.075472	0.03774	0.037736	0.09434	0.000000	0.000000	0.037736	0.056604	0.037736	0.000000	0.000000	0.0000000	0	0
2	0.08333	0.08333	0.25000	0.12500	0.16667	0.06250	0.104167	0.00000	0.020833	0.02083	0.000000	0.020833	0.041667	0.000000	0.020833	0.000000	0.000000	0.0000000	0	0
3	0.22368	0.13158	0.09211	0.07895	0.06579	0.13158	0.000000	0.02632	0.039474	0.02632	0.000000	0.039474	0.065789	0.026316	0.013158	0.026316	0.013158	0.0000000	0	0
4	0.12903	0.13978	0.15054	0.09677	0.09677	0.07527	0.064516	0.05376	0.043011	0.03226	0.000000	0.000000	0.032258	0.032258	0.032258	0.021505	0.000000	0.0000000	0	0
5	0.33333	0.15033	0.02614	0.07190	0.05229	0.12418	0.026144	0.05229	0.019608	0.01961	0.032680	0.026144	0.032680	0.006536	0.019608	0.000000	0.006536	0.0000000	0	0
6	0.12500	0.09821	0.23214	0.12500	0.12500	0.05357	0.116071	0.01786	0.044643	0.00000	0.008929	0.008929	0.017857	0.008929	0.000000	0.008929	0.008929	0.0000000	0	0
7	0.14103	0.10256	0.05556	0.15385	0.06410	0.05983	0.051282	0.08974	0.008547	0.01709	0.025641	0.081197	0.004274	0.012821	0.008547	0.055556	0.064103	0.0042735	0	0
8	0.10479	0.16056	0.15887	0.12732	0.08789	0.03944	0.078310	0.06535	0.038310	0.01972	0.052394	0.023662	0.012958	0.012958	0.008451	0.005634	0.002817	0.0005634	0	0
9	0.20000	0.18182	0.03636	0.10909	0.03636	0.07273	0.054545	0.10909	0.036364	0.00000	0.000000	0.072727	0.036364	0.000000	0.036364	0.000000	0.018182	0.0000000	0	0
10	0.40097	0.09179	0.03382	0.06280	0.02415	0.16908	0.004831	0.06763	0.028986	0.01449	0.009662	0.043478	0.028986	0.004831	0.000000	0.004831	0.009662	0.0000000	0	0

The average number of genres per user is 15.559.



Using these vectors, it is possible to use existing clustering algorithms such as k-means (presented in section 34 of the textbook). However after an initial test, **Gaussian Mixture Model clustering** was chosen instead. This link offers and explanation of the algorithm and why can be a better method for to account variances in this particular case. In this excercise, the GMM implementation the **ClusterR** package is used. The code below shows a sample of how GMM is used in this project.

```
#First we need to combine weights and averages, and remove the first columns
user_profiles <- user_vector$user_averages %>%
      left_join(user_vector$user_weights, by = "userId")
us <- user_profiles %>% select(-userId)

## Then we need to specified the number of clusters we want the users to be classified into,eg
```

```r
cluster_n <- 5

###The, we run GMM as per the manual
fit <- GMM(us, cluster_n, dist_mode = "maha_dist",
           seed_mode = "random_subset", km_iter = 10,
           em_iter = 10, verbose = F)

pr <- predict_GMM(us, fit$centroids, fit$covariance_matrices, fit$weights)
user_profiles$group <- pr$cluster_labels

###Show user profiles

user_profiles
```

Before running the above code, it is worth noticing that:

- The combined averages and weights vectors have 40 dimensions, which is large. In addition, most of these vector don't have a presence across all these dimensions (in this particular training set, none has). Potentially there are some vector without any common dimensions (action&adventure fans opposite documentaries&film-noir followers).
- Although no analysis has been done, it is reasonable to assume that some genres have a big overlap, thus not all dimensions are independent from each other.
- There will be a significant number of vectors(users) defined by the most popular dimensions (genres).
- Taking this into account, using all dimensions may not be the best method of clustering - and it is interesting to assess how this impacts the RMSE.
- In addition, the number of clusters - which will certainly have an impact on the RMSE - is an input parametre.

Considering all the points above and after some initial tests, this document proposes a clustering algorithm based on GMM which follows the below steps:

1. User vector is created.
2. The number of genres used for clustering (n) is set.
3. Using the genre percentages, the relative weight of the n most popular genres is calculated.
4. The user vector is filtered to only keep the cases where the previously calculated relative weight meets/exceeds a treshold; This filters out the users for which the selected genres are not relevant.
5. The filtered vector is fed into the GMM functions and users are group into an initial set of clusters.
6. The remaining users are sorted, and filtered by their n most popular genres. Again, they are filtered by the predifined treshold and fit into a new set of clusters.
7. Step 6 repeated as many times as desired or until there are either no users left or there is no enough data to cluster them.
8. All remaining users (if any) and classified in a residual group.

This algorithm as been written into the **User_Classifier** function, available in the R file. It is worth noticiing this process has four *arbitrary* parametres that will change the number of resulting clusters and therefore the may have an impact on the RMSE - this will be discussed later in the document.

Once the users have been segmented, it is possible to use the *textbook* model to calculate the average and biases for each of these groups and then predict ratings (the objective of this entire process!). The first part of this step is straightforward (just add the group to the dataset and use this parametre to group prior to summarising) and it has been written into a function called **Group_Biases**.

However, for predicting algorithm needs to consider that potentially some user weren't not grouped into a meaningful cluster and thus this method makes no sense. Additionally, there maybe situations where a particular (group,movie) was absent from the training data and this method is not valid. To cover those situations, the prediction algorithm needs to steps:

1. A prediction method using group averages and biases for all qualifyng reviews - just like the textbook model but split per group.
2. A *backstop* method for all reviews don't fit the above - for this we can use the 'textbook' method, which is better than just guessing.

Please note that both segments have tuning parametres (λ_{1-4}).

This prediction algorithm has been written into a function called **Rating_Predicter**.

2.4 Tuning Step 1 - Optimise Clustering.

The first optimisation step consists in finding the optimal clustering parametres to minimise the RMSE. These parametres are:

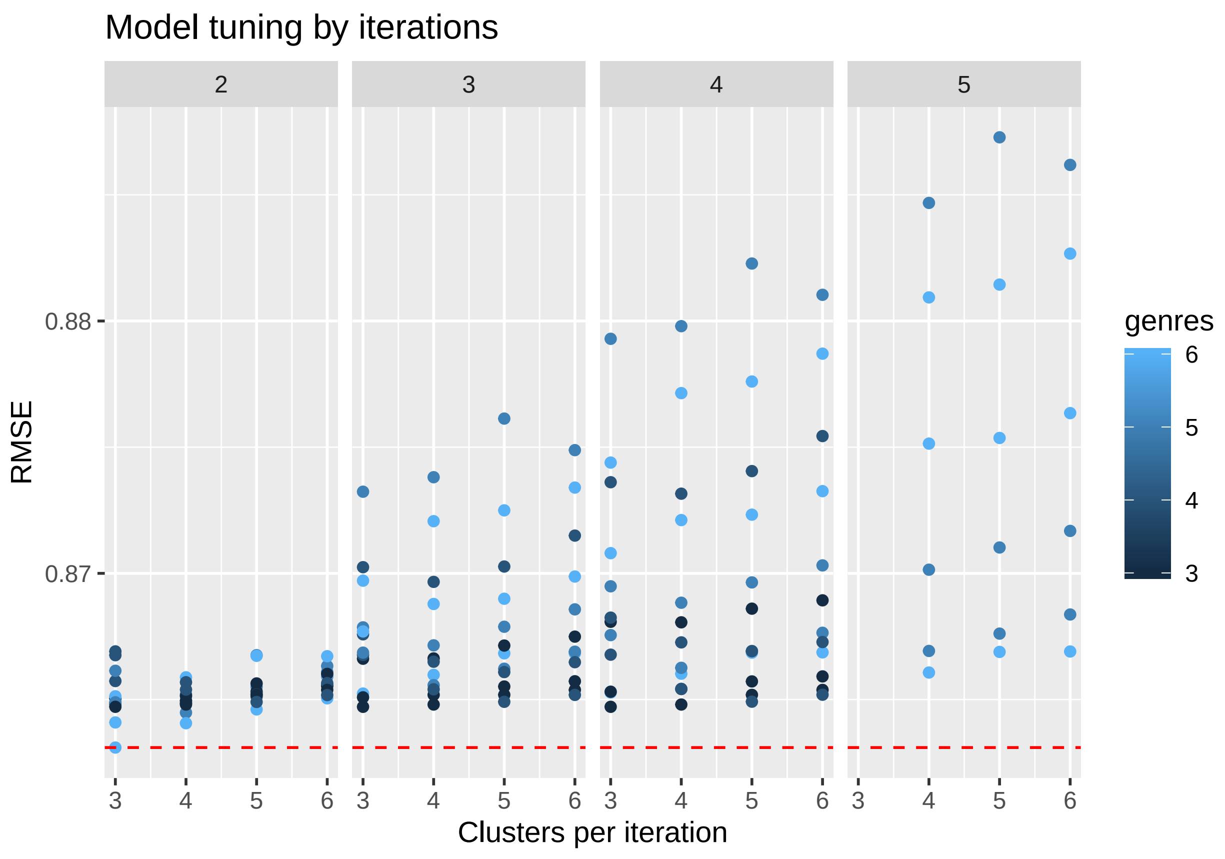
1. the number of genres (*genres*) to consider in each clustering round.
2. the number of target clusters per round (*cluster_n*).
3. the cutoff value to filter the users to feed into each clustering round (*cutoff*).
4. the number of time clustering will be attempted (*iterations*).

This process has been executed through the below code.

```
#create tibble to record stats

stats <- tibble(clustering=character(),iterations=double(),cutoff=double(),
                genres=double(),cluster_n=double(),RMSE = double(),
                RMSE_1 = double(),RMSE_2 = double(),
                RMSE_3 = double(),
                method_1=double(),method_2=double(),
                method_3=double())

#define ranges to iterate
```

```r
genres_n<-3:6
cluster_n <- 3:6
cutoff <- c(0.4,0.5,0.6)
iterations <-2:4


#iterate predictions
for(g in iterations){
  for(h in cutoff){
    for (i in genres_n){
      for(j in cluster_n){

        #create groups, biases  and predict

        user_classification  <- User_Classifier(edx_train_test$train,user_vector,
                                                genres=i,step_cutoff=h,cluster_n=j,
                                                cluster_type="GMM",iterations=g)

        biases_by_group <- Group_Biases(edx_train_test$train,user_classification)

        results<-Rating_Predicter(edx_train_test$test,user_classification,
                                  biases_by_group,general_biases)

        # store results in stats tibble
        stats <- add_row(stats,clustering="GMM",iterations=g,
                         cutoff=h,genres=i,cluster_n=j,
                         RMSE = results$RMSE$overall,
                         RMSE_1=results$RMSE$`1`,RMSE_2=results$RMSE$`2`,
                         RMSE_3=results$RMSE$`3`,
                         method_1=results$distribution_percentage$`1`,
                         method_2=results$distribution_percentage$`2`,
                         method_3=results$distribution_percentage$`3`)
     }
   }
 }
}
#cleanup
clustering_stats <- stats
rm(g,h,i,j,results,stats)
```

This code in turn produces the results represented in the below chart.



The optimal solution has the below parametre, which shows also the resulting overall RMSE plus the RMSE for the clustered (RMSE_1) and non-clusters (RMSE_2) groups, along with the proportion of reviews in each method.

Table 6: Optimal Clustering Parametres

clustering	iterations	cutoff	genres	cluster_n	RMSE	RMSE_1	RMSE_2	method_1	method_2	method_3
GMM	2	0.5	6	3	0.8630969	0.8629808	1.050733	0.9994422	0.0005578	0

When compared with the *texbook* method, this procedure has better results:

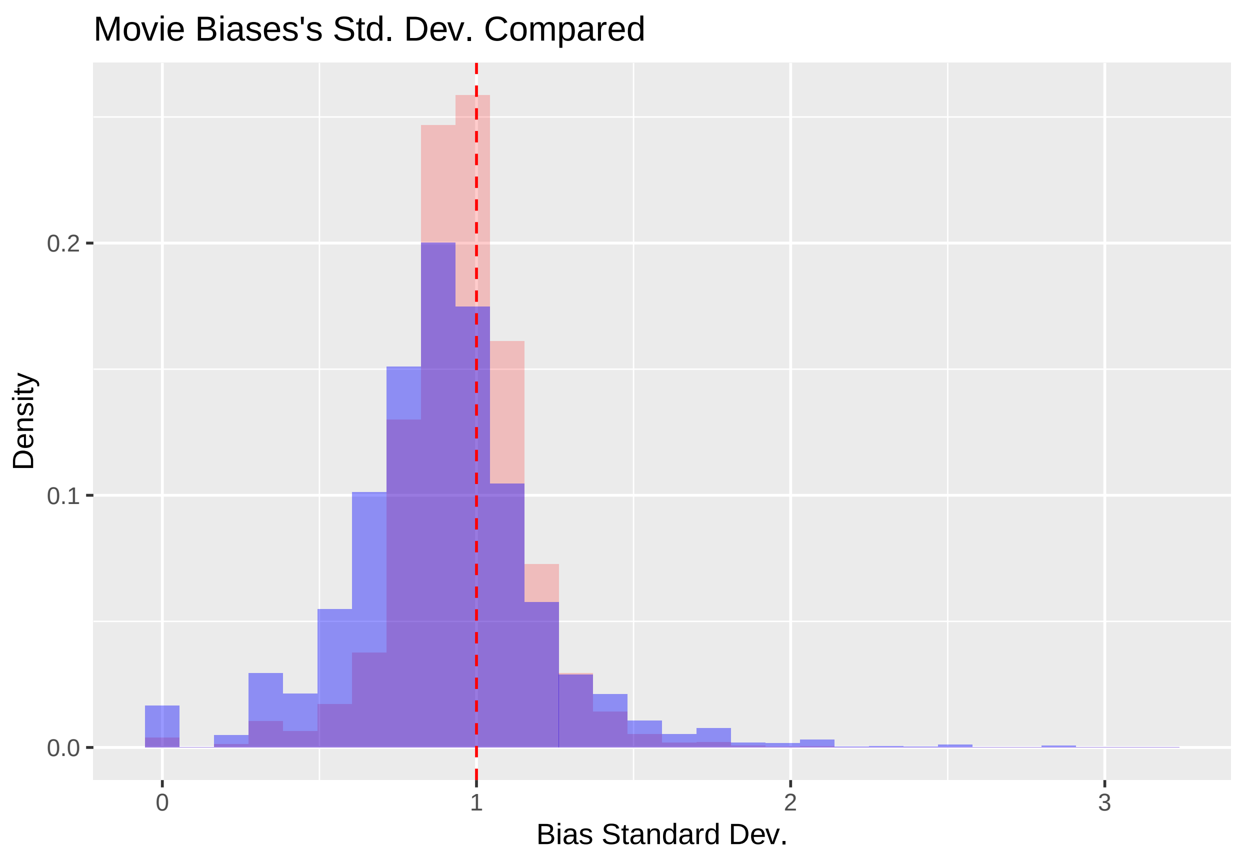
Table 7: First and Second Predictions

Model	RMSE	Below Target
Textbook	0.8663964	FALSE
Optimal Clusters	0.8630969	TRUE

What are reasons this method has a better performance? Some explanations are:

- In this cases, there are different averages for different groups, having more accurate stating points.

- Also, grouping provides movie biases with smaller deviations, contributing to a smaller RMSE. In this case, 69.035 % of the deviations are lower than 1, compared to 62.161 % for the *textbook* case. This can be visualised in the below chart:



It is worth noticing this solution generates splits the training set into 6 groups of users, with the below characteristics:

Table 8: Optimal User Groups

group	users	movies	top_movies
0	6140	10450	Jurassic Park Pulp Fiction Shawshank Redemption, The Silence of the Lambs, The Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)
1	9287	9649	Matrix, The Pulp Fiction Shawshank Redemption, The Silence of the Lambs, The Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)
2	7944	9768	Fargo Pulp Fiction Schindler's List Shawshank Redemption, The Silence of the Lambs, The
200	9622	8903	Forrest Gump Four Weddings and a Funeral Pretty Woman Pulp Fiction Sleepless in Seattle
201	15308	9921	Batman Forrest Gump Jurassic Park Pulp Fiction True Lies
202	21577	9327	Forrest Gump Fugitive, The Jurassic Park Terminator 2: Judgment Day True Lies

Looking at this and the compared deviation histogram's, it looks like there are groups that maybe quite similar and there improvement in accuracy seems smaller, yet the result improves. Although not covered in there report, perhaps exploring better clustering mechanisms will deliver significantly improved results.

2.5 Tuning λ parametres

Having settled on a set of optimal clustering parametres, there is stil room to attempt improving the RMSE but tuning the λ parametres in the movie and user bias equations:

$$b_i = \frac{1}{\lambda_1 + m}\sum_{j=1}^{m}(rating_j - \hat{\mu})$$

$$b_u = \frac{1}{\lambda_2 + n}\sum_{j=1}^{n}(rating_j - \hat{\mu} - b_{i,j})$$

In the proposed model, the are four lambda parametres: two for the clustering biases (from now on, λ_1 and λ_2) and two for the backstop model (λ_3 and λ_4). However, given the previous result shows that over 99% of the dataset fit in the cluster method, thus no tuning of (λ_3 and λ_4) will be done at this stage.

In addition, at this stage it would be good to see how much the RMSE naturally varies due to being a random variable - in order to have a level of confidence how well the method will perform against the validation set. For these two reasons, k-fold cross validation will be also done at this point.

There is one problem with the method : since there are three variables to tune and each iteration requires data partition, user clustering, calculating biases and predictions. As experienced in the previous optimisation, this will take **several** hours to complete. As a workaround, tuning will proceed the following way:

1. First, we will run iterations to run $lambda_1$ and $lambda_2$.
2. From this first set, we will choose the twenty combinations with the lowest RMSE.
3. Then, we will run k-fold validation for those twenty combinations.
4. Finally, the optimal set of $lambda_1$ and $lambda_2$ will be chosen picking the lowest average RMSE (from all k iterations).

The first iteration will be implemented with the below code:

```
stats <- tibble(k=double(),lambda_1=double(),lambda_2=double(),
                RMSE = double(),RMSE_1 = double(),RMSE_2 = double(),
                RMSE_3 =double(),
                method_1=double(),method_2=double(),method_3=double())

lambda_1 <- seq(0,8,0.5)
lambda_2 <- seq(0,8,0.5)


for(k in 1:1){

  edx_cross_validation <- Train_Test(edx_train_test$train)

  user_vector <-User_Vectoriser(edx_cross_validation$train)
  general_biases <- General_Biases(edx_cross_validation$train)
  user_classification  <- User_Classifier(edx_cross_validation$train,user_vector,
                                              genres=genres,step_cutoff=step_cutoff,
                                        cluster_n=cluster_n,cluster_type=cluster_type,
                                        iterations=clustering_iterations)


  for (i in lambda_1){
    for(j in lambda_2){


            biases_by_group<-Group_Biases(edx_cross_validation$train,
                                            user_classification,
                                            lambda_1=i,lambda_2=j)

            results<-Rating_Predicter(edx_cross_validation$test,
                                        user_classification,
                                        biases_by_group,general_biases)

            stats <- add_row(stats, k=k, lambda_1=i,lambda_2=j,
                                RMSE = results$RMSE$overall,
                          RMSE_1=results$RMSE$`1`,
                          RMSE_2=results$RMSE$`2`,
                          RMSE_3=results$RMSE$`3`,
                          method_1=results$distribution_percentage$`1`,
                          method_2=results$distribution_percentage$`2`,
                          method_3=results$distribution_percentage$`3`)
  }
  }

  rm(edx_cross_validation)

}

group_lambda_stats_1 <- stats
rm(i,j,k)
```

The 20 lowest RMSE combinations are presented in the below table:

Table 9: Lambda Optimisation - Step 1

lambda_1	lambda_2	RMSE	RMSE_1	RMSE_2	method_1	method_2
2.0	4.0	0.8618505	0.8616856	1.096348	0.9993815	0.0006185
2.0	4.5	0.8618516	0.8616867	1.096348	0.9993815	0.0006185
2.0	3.5	0.8618548	0.8616900	1.096348	0.9993815	0.0006185
2.5	4.0	0.8618559	0.8616910	1.096348	0.9993815	0.0006185
2.5	4.5	0.8618569	0.8616920	1.096348	0.9993815	0.0006185
2.0	5.0	0.8618576	0.8616927	1.096348	0.9993815	0.0006185
2.5	3.5	0.8618604	0.8616955	1.096348	0.9993815	0.0006185
2.5	5.0	0.8618627	0.8616978	1.096348	0.9993815	0.0006185
2.0	3.0	0.8618652	0.8617004	1.096348	0.9993815	0.0006185
2.0	5.5	0.8618680	0.8617031	1.096348	0.9993815	0.0006185
2.5	3.0	0.8618710	0.8617061	1.096348	0.9993815	0.0006185
2.5	5.5	0.8618729	0.8617081	1.096348	0.9993815	0.0006185
2.0	6.0	0.8618824	0.8617175	1.096348	0.9993815	0.0006185
2.0	2.5	0.8618824	0.8617176	1.096348	0.9993815	0.0006185
2.5	6.0	0.8618872	0.8617223	1.096348	0.9993815	0.0006185
2.5	2.5	0.8618883	0.8617235	1.096348	0.9993815	0.0006185
1.5	4.0	0.8618892	0.8617243	1.096348	0.9993815	0.0006185
3.0	4.0	0.8618899	0.8617250	1.096348	0.9993815	0.0006185
1.5	4.5	0.8618905	0.8617256	1.096348	0.9993815	0.0006185
3.0	4.5	0.8618906	0.8617258	1.096348	0.9993815	0.0006185

Then, using a modified version of the previous code, 5 more iterations of those 20 sets of *lambdas* are generated and average RMSEs are calculated. The best 5 results are presented in the below table.

Table 10: Lambda Optisimation - Step 2

lambda_1	lambda_2	Avg_RMSE
2.0	4.5	0.8623377
2.0	4.0	0.8623385
2.5	4.5	0.8623390
2.5	4.0	0.8623400
2.0	5.0	0.8623419

Using these parametres, the model is retrained using the whole training set and RMSE is calculated against the test set for comparison with the previous two cases.

Table 11: Result with Optimal Lambdas on Test Set

RMSE	RMSE_1	RMSE_2	RMSE_3	method_1	method_2	method_3
0.8620962	0.8619794	1.050733	NaN	0.9994422	0.0005578	0

Table 12: First and Second Predictions

Model	RMSE	Below Target
Textbook	0.8663964	FALSE
Optimal Clusters	0.8630969	TRUE
Optimal Clusters + Optimised Lambdas	0.8620962	TRUE

2.6 Checks prior to prediction

Before moving to assess the model against the validation set, there are two points to inspect. The first point was mentioned before and is the RMSE variance when k-fold validation was done. Taking the 5 best results, it is possible to build the below boxplot. This graph show that all combinations are well below the RMSE threshold!

The second inspection is to have a look at the films that couldn't be rated by the clustering mechanism. The table below shows the top 20 films in that category. After reviewing this list, those films seem to be mostly obscure and seems reasonable that couldn't be found in the clustering movie biases. Even if the user is not new, this result would be expected for rare films, which are not going to have many reviews. This is also reassuring since it minimises the risk of popular movies falling into this category and risking a higher prediction error.

Table 13: Non Clusterable Reviews - Movies

title	n
Dying of Laughter (Muertos de risa)	4
Bitter Sugar (Azúcar amarga)	3
Evil Aliens	3
Reflections In A Golden Eye	3
All Night Long	2
Anna	2
As in Heaven (Så som i Himmelen)	2
Baby Doll	2
Battle for Haditha	2
Biker Boyz	2
Bordertown	2
Burning, The	2
Comandante	2
Dakota	2
DarkBlueAlmostBlack (Azuloscurocasinegro)	2
Demonlover	2
Fathom	2
Free Zone	2
Grisbi (Touchez pas au grisbi)	2
Happily Ever After	2

3 Results

With all optimsed paramatres, it is time now to try to predict the ratings for the validation set. This will be done in two parts:

- First, the model will be re-train using all optimal parametres and the whole combined training+test dataset.
- Then the model will be used to predict the ratings for the validation set and the RMSE will be calculated.

The code to achieve this is presented below

```
# Prediction against validation dataset

##re-join training test tests
edx <- rbind(edx_train_test$train,edx_train_test$test)

### Retrain : reclustering and re-calculation of biases with new dataset
```

```
user_vector_prediction <-User_Vectoriser(edx)
general_biases_prediction <- General_Biases(edx)
user_classification_prediction  <- User_Classifier(edx,user_vector_prediction,
                                                genres=genres,step_cutoff=step_cutoff,
                                         cluster_n=cluster_n,cluster_type=cluster_type,
                                         iterations=clustering_iterations)
biases_by_group_prediction <- Group_Biases(edx,
                                            user_classification_prediction,
                                            lambda_1=lambda_1,
                                            lambda_2=lambda_2)

### tidy up validation table

validation<-Tidy_Up(validation)

# Predict ratings

prediction <- Rating_Predicter(validation,
                               user_classification_prediction,
                               biases_by_group_prediction,
                               general_biases_prediction)


##Calcuate method split and RMSE

stats <- tibble(RMSE = double(),RMSE_1 = double(),
                RMSE_2 = double(),
                method_1=double(),method_2=double())

stats <- tibble(RMSE = prediction$RMSE$overall,
                         RMSE_1=prediction$RMSE$`1`,
                         RMSE_2=prediction$RMSE$`2`,
                         method_1=prediction$distribution_percentage$`1`,
                         method_2=prediction$distribution_percentage$`2`)
prediction_stats <-stats

rm("stats")
```

The RMSE results are shown in the following table:

Table 14: RMSE for validation set

RMSE	RMSE_1	RMSE_2	method_1	method_2
0.8614071	0.8612884	1.083573	0.999527	0.000473

4 Conclusion