R Notebook

Problem Setting and Approach

Movielens, a platform for non-commercial personalized movie recommendations has made available several movie datasets of different sizes. Throughout this kernel, we will use the 10 million rows dataset to create a movie recommender.

Predictors include userId, movieId, movie title, year released, timestamp of rating and genre. Our target variable is continuous, thus we will measure success of our algorithm by Residual Mean Square Error.

To successfully plan a strategy , we will visually explore our data and clean/engineer it whenever possible.

After experimenting with various approaches, our final prediction attempt will focus around matrix factorization.

Model Basis

We will approach the prediction of rating in the following manner:

- The **mean rating** of all movies is the easiest, yet justified, prediction we could give. By taking a look at the distribution of ratings, it becomes obvious that the mean has the highest probability to happen. If we were asked to pick a rating for a movie we don't know from a user we also don't know, the safest guess would be the mean. But since we have features available, we will try to use them as follows:
- On top of the mean rating, we can add the mean of each **specific movie** we want to predict. For example, Lord of the Rings is widely considered a good movie. Its mean rating will be way higher than the overall mean (which is shaped by both renowned and not so-acclaimed movies). As a result, a good movie's average will raise our prediction, while a bad movie's will decrease respectively. A common word in statistics used to describe such effects is bias. Thus, we'll call it movie bias and calculate is as the difference between mean rating of all movies and the mean rating of the specific movie we want to predict a rating for. For example, if we suppose the mean rating of all movies is 3.5, but Lord of the Rings has a mean rating of 4.5, then the movie bias of Lord of the Rings is 1 (= 4.5 - 3.5).
- Accordingly, the **same applies to users**. Some users are demanding and tend to rate lower than others. We also need to take this under consideration. Continuing the example above, a user with mean rating of 2 will get a -1.5 (= 2 - 3.5) user bias effect in our prediction.
- Finally, we will try to encapsulate a more abstract form of **genres**. By abstract we mean that we will not explore the presence of genres per se, but rather the presence of *concepts*. These concepts can be whichever factor describes a preference. Each user has a tendency towards a specific factor and each movie is somehow associated (more or less, positively or negatively) with this factor.

Considering the number of ratings

The fact that not all movies bear the same number of ratings creates a distortion. For movies (the same applies to users) with many ratings, it's safer for us to base our prediction on their calculated bias. But it becomes overt that movies with low ratings may lead us to assign biases they don't actually deserve. This discrepancy can be rectified by adjusting the way we compute each movie's average rating. Movie bias is calculated by computing the difference between each rating and the mean rating of all movies, summing up all these differences and diving by the number of movie ratings to make this measure average. In order to control for the number of ratings (that is, take into more consideration a movie's ratings if it has many ratings and vice versa), we will add a penalty term when dividing with the sum of all differences from the mean. This way, movies with many ratings will get almost no influenced at all by the penalty, while movies with few ratings might get their average bias halfed or more.

To make it more straightforward, we can consider a very popular movie with a sum of differences from the genereal mean of 2000, number of ratings of 2500 and a penalty term of 4. Before penalization, the popular movie would yield a movie bias of +0.8 (= 20,000/25,000). After penalization, it would return 0.799 (= 20,000 / 25,004), almost the same as before. On the other hand, a not-so-popular movie with just two ratings and a sum of (two) differences at +3, would return, before penalization, a movie bias of +1.5 (= 3 / 2) . After, penalization it would become significantly lower at .5 (= 3 / 6).

Thus, it becomes obvious that penalization minimizes biases based on a few ratings, while it leaves almost untouched biases returned by a great number of ratings. A common term for this penalty is lamba (the greek l, lambda), and that's how we'll call it throughout this analysis. Since lambda is a tuning parameter, we need to choose the best. We will cross-validate (ie try different values and compute their efficiency) to get an optimal penalty term for both the movie and the user biases.

Factorizing the residuals

So far, the formula of our prediction is formed as follows:

prediction = mean_rating_of_all_movies + movie_bias + user_bias

or

hat{y} = mu + b_i + b_u

The difference between prediction and reality is the sum of residuals (notated with the greek epsilon - epsilon)

Although this naive model is a good basis, it is not enough. We need to analyze its residuals and take more out of it. Inside the ratings matrix exist many groups of users and movies somehow corellated. Some users like thrillers and tend to rate them higher, others like Leonardo di Caprio, others dispise romance movies, etc. These trends are usually called latent factors or concepts and are almost infinite. The current state of our model ignores such factors and leaves similar residuals among them.
In order to detect and take into consideration the most important of them, we will employ a technique called matrix factorization.

The first step to factorize the residuals is to spread them in a sparse matrix. That means we convert each user into a row, each movie into a column with the corresponding ratings the values of this matrix. Its dimensions are the number of distinct users x the number of distinct movies . Since users haven't rated all movies, this matrix will be very sparse, containing NAs wherever a user has not rated a movie.

Our goal is to decompose this sparse matrix of residuals into 3 dense matrices, which, multiplied with each other, return a dense matrix very close to our initial sparse. In order to achieve it we will employ singular value decomposition (SVD for short). If we name our initial matrix of (m x n) dimensions A and the number of underlying concepts is k, we can represent A as the multiplication of a (m x k) matrix (commonly notated as U) with a (k x k) nonnegative diagonal matrix (notated as Sigma) and a (k x n) matrix (notated as V)

A = U Sigma V^T

This fairly computationally expensive process will be realized with the help of the funkSVD function from the recommenderlab package. After this transformation is achieved, we have managed to decompose our residuals epsilon. Each user-movie instance of residuals can be represented by a sum of multiplied values from the U, Sigma, and T matrices. The residual of movie i and user j is rewritten as:

epsilon_{i,j} = sum p_{i,1} q_{1,j} + p_{i,2} q_{2,j} + … + p_{i,k} q_{k,j}

Where p and q represent values from the matrices U and Q obtained by the SVD. k is the number of concepts with the variability of each term p_{i,k} q_{k,j} decreasing (thus, concept 1 has more predictive power that concept 2 etc.)

This decomposition can be embedded in our model:

hat{y} = mu + b_i + b_u + p_{i,1} q_{1,j} + p_{i,2} q_{2,j}+ … + p_{i,k} q_{k,j}

Libraries and Data Import

Let's start our journey by importing libraries.

```r
#Importing necessary libraries

#Data manipulation
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
# Train/Test split and Model Training
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
#Handle Tables
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
#SVD decomposition
if(!require(recommenderlab)) install.packages("recommenderlab",
                                         repos = "http://cran.us.r-project.org")
# to plot multiple grobs alongside
if(!require(gridExtra)) install.packages("gridExtra",
                                         repos = "http://cran.us.r-project.org")
# to separate stacked genres
if(!require(splitstackshape))install.packages("splitstackshape",
                                         repos = "http://cran.us.r-project.org")
#to transform shape of our data
if(!require(reshape2)) install.packages("reshape2",
                                         repos = "http://cran.us.r-project.org")
#For rmse function
if(!require(ModelMetrics))  install.packages("ModelMetrics",
                                      repos = "http://cran.us.r-project.org")
```

Now we need to download our data, read each file as a table and finally merge them into one.

Note: Because my pc could not run such a large dataset, I produced the report with a quarter its rows. In case you want to run on the whole dataset, disable the command that splits it.

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

#if you want to run the whole dataset, disable the following command
movielens <- movielens %>% slice(1:floor(nrow(movielens)/4))

rm( ratings, movies )
```

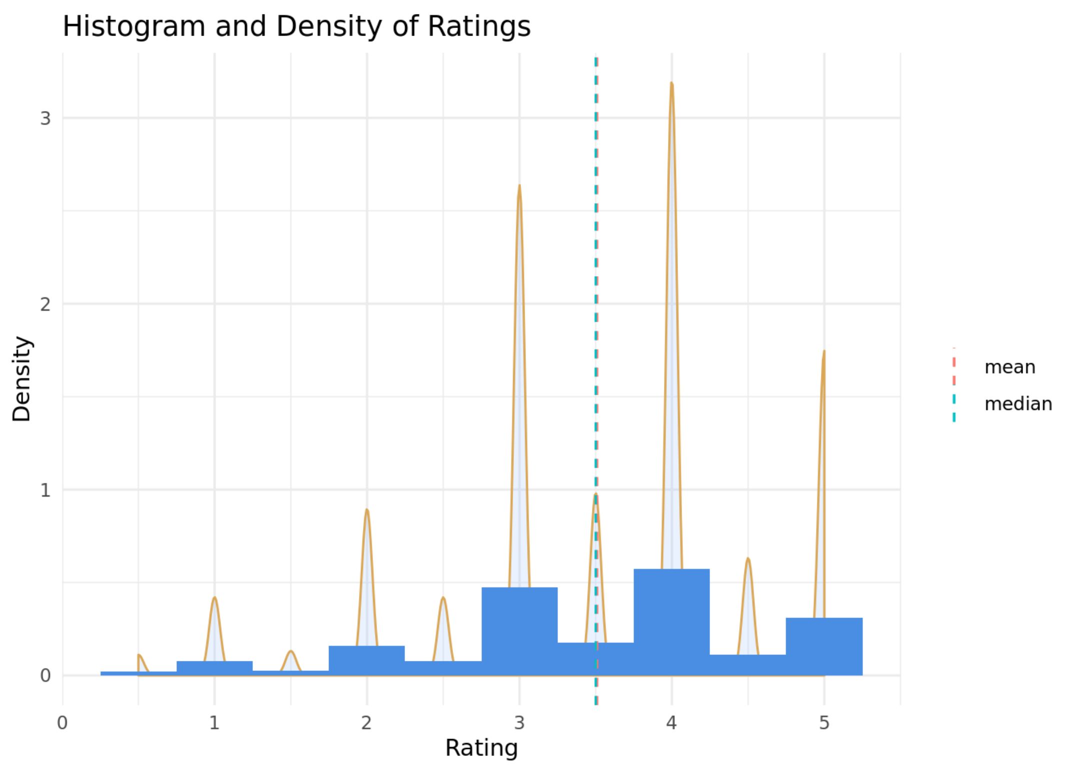
EDA and Data Cleaning

Now that we successfully imported our data, we need to start exploring it. Some quick thoughts of this first glance are:

- **movieId** and title contain almost the **same information**. At times, titles are repeated. Thus, we expect unique movieIds to be more than the unique titles. For our model, we will only keep movieId as a predictor.
- **year** can contain **little to no predictive power**. People who like space movies will rate highly “2001:A space odyssey” as well as “Interstellar”. It stays fot the EDA but we probably won’t need it in our prediction.
- **genres**, even though they are very hard to be used directly in a model, will be a good signal for the existence of underlying **concepts** in our data.
- **timestamp** probably contains **no information** at all. The time a rating is done has nothing to do with a user’s preferences or the quality of a film. However, we will briefly explore it below.
- **title** nests another feature, the **year of release**. We will unnest it to test whether it contains any power.

Distribution of Ratings

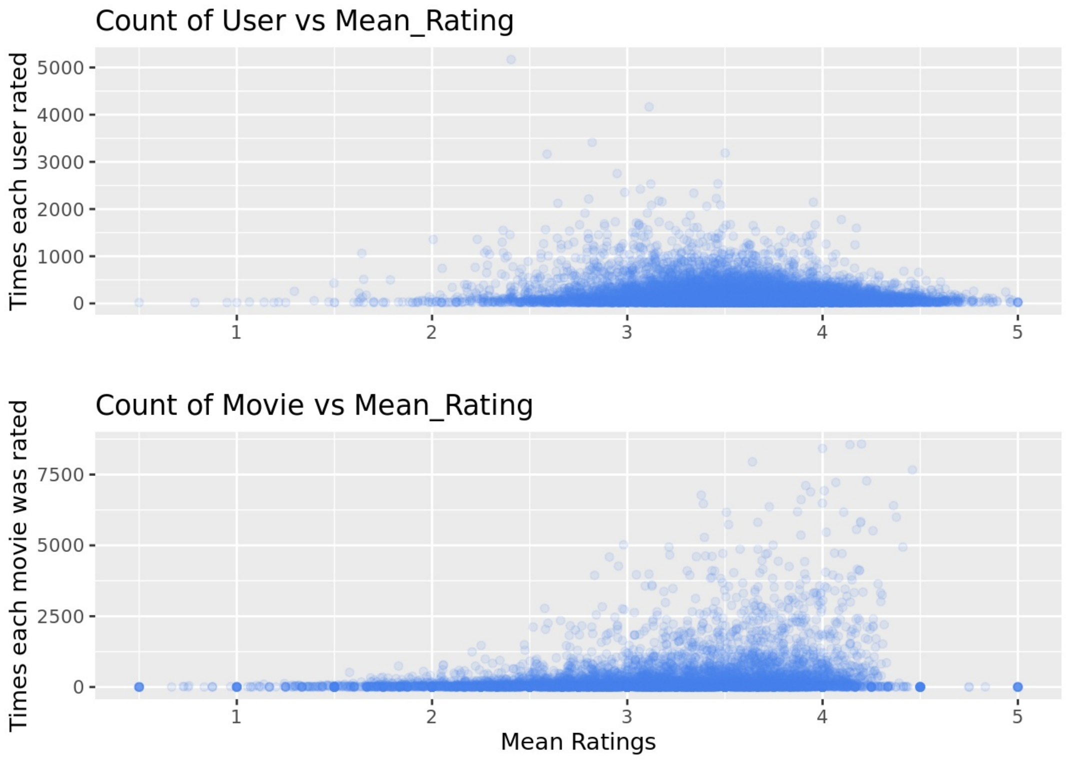
Throughout our analysis, the mean of ratings will play a significant role. It will be the base of our prediction. Let’s take a look at the distribution that shapes it.



Both the mean (3.51) and the median (3.5) are greater than half of the range (eg 2.5). That suggests a tendency towards high ratings. It looks like it is much more likely for someone to rate a movie she liked rather than one she didn’t. Also, users are lenient with movies they don’t like and seldom place ratings beneath 2.

Count vs Mean Rating

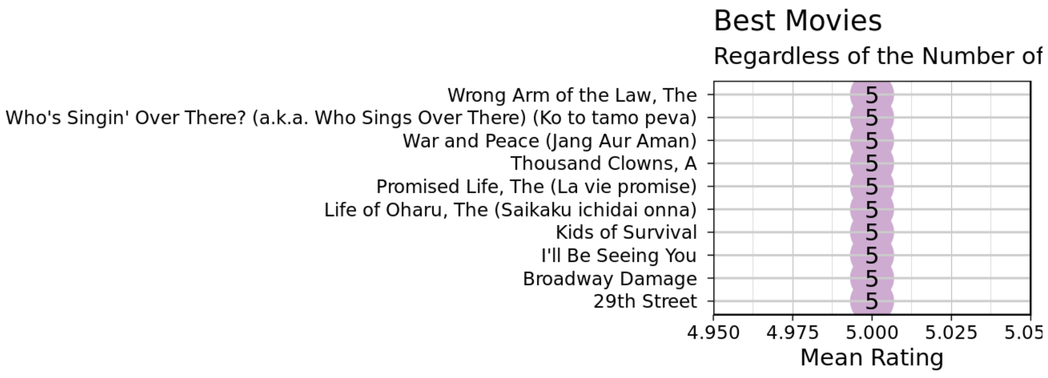
Now we want to check, both for users and movies, whether the number of ratings goes along with the mean rating. In other words, we seek to answer the question: “Do movies (users) with many ratings tend to have (give) high ratings and vice versa?”

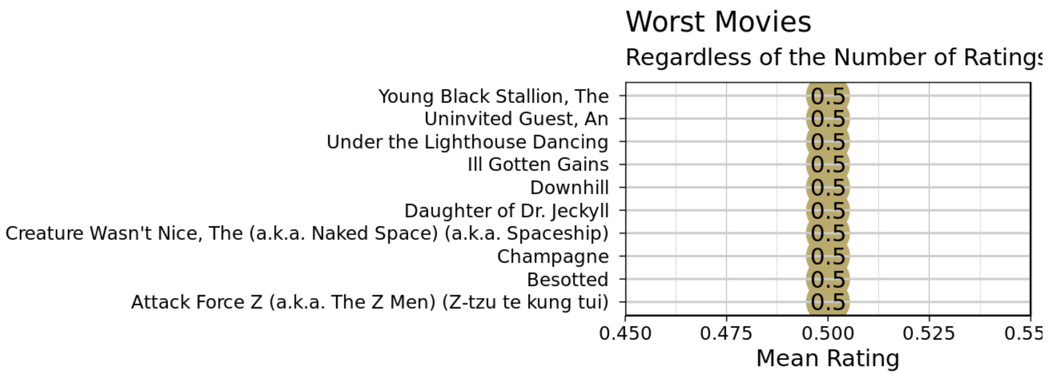


We notice a correlation for the movies. That could be a hint that popular movies tend to have high ratings. On the other hand, users seem randomly scattered not showing any strong tendency.

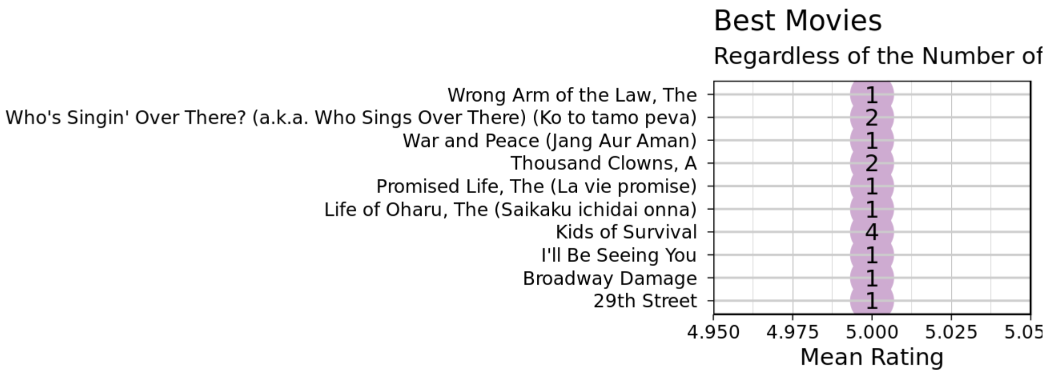
Best and Worst Movies by Rating

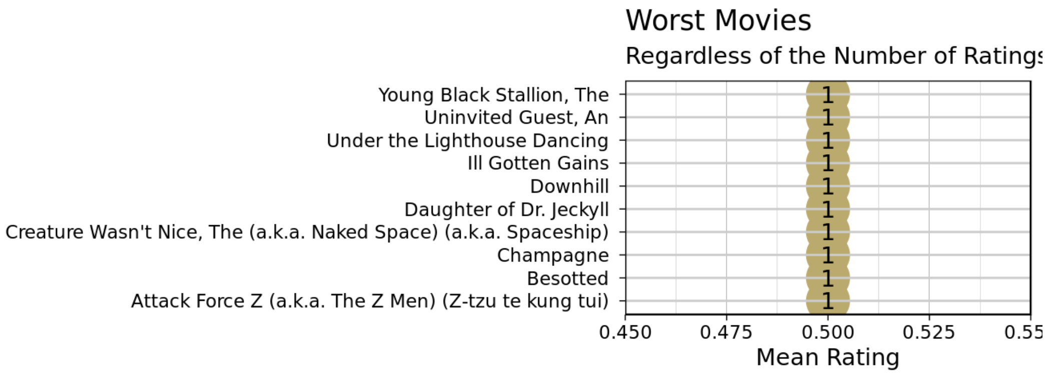
For starters, let’s take a look at the best and the worst movies, without taking into account the number of ratings they have.





Both Best and Worst movies all share the same rating. Oddly enough, they are not so recognizable. To account for their popoularity we will replot them, this time with the number of ratings as label instead of the average rating.

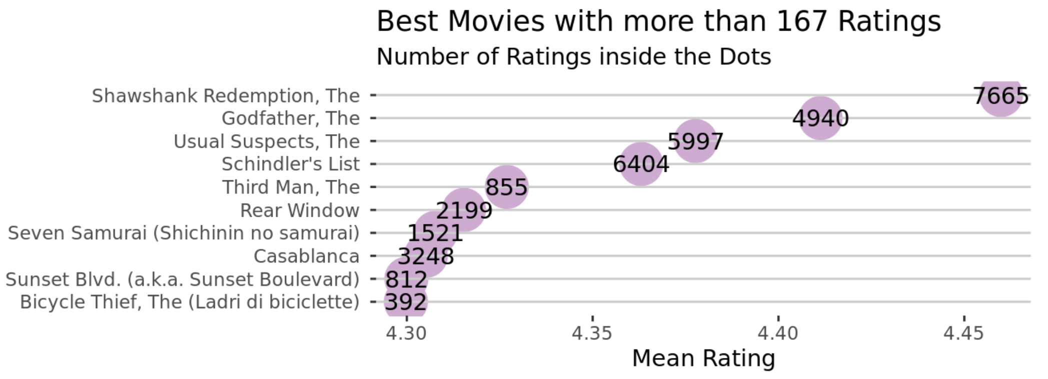


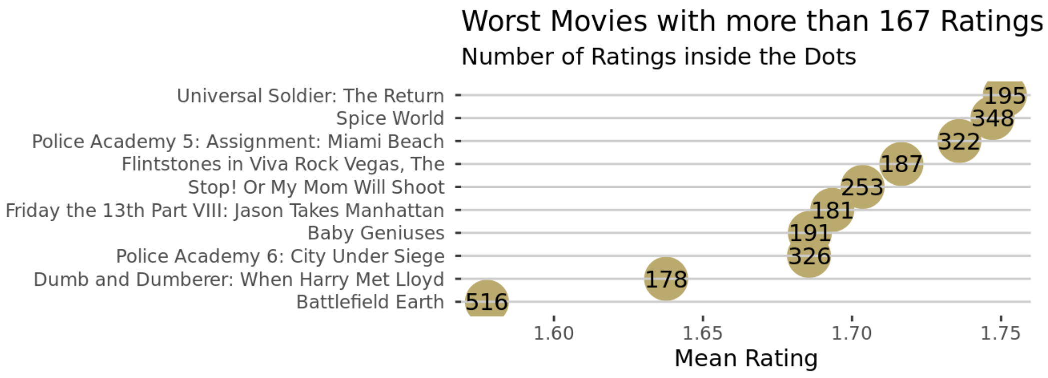


As expected, almost all plotted movies have 1 rating. That explains the extreme means.

To account for this discrepancy, we will filter the number of ratings. As a low limit we will define the first quantile of the number of ratings.

```
#pick the 1rd quantile of the movie count as a low limit
limit <- (summary(mov_df$movie_cnt))[5]
```





Now we start to recognize the movies. Among the best are some all time classics, while among the worst are many commonly critised movies.

We need to take this effect under consideration. Later, in the modelling section, we will apply a penalty term to each movie's mean rating so that it encompasses the number of ratings they were shaped from.

Genres

The genres column potentially contains great value. Unfortunately for us, many movies belong to more than one genre, and all of them are binded together in one column, separated by "|". We need to unnest them before we start exploring them.

```
#For every genre entry create a separate row containing only this entry
movielens_genres <- movielens %>%
        cSplit( "genres", #split the genres column
               sep="|", #on separator |
               direction =  "long" ) #towards a long direction
```

Now that we split genres apart, let’s look at the unique values contained in it and the frequency they appear.

```
## [1] " Genres count of distinct values : "
```

```
## .
##              Drama             Comedy             Action
##            1089529             983341             705929
##           Thriller          Adventure            Romance
##             642090             529676             477216
##             Sci-Fi              Crime            Fantasy
##             371151             367340             256601
##           Children             Horror            Mystery
##             206636             190619             158213
##                War          Animation            Musical
##             141982             130114             120953
##            Western          Film-Noir        Documentary
##              52405              33373              25368
##               IMAX (no genres listed)
##               2104                  2
```

We notice that some genres are pretty similar. In order to concentrate them a bit, we will merge:

- Horror to Thriller
- Adventure to Action
- War to Action
- Sci-Fi to Fantasy

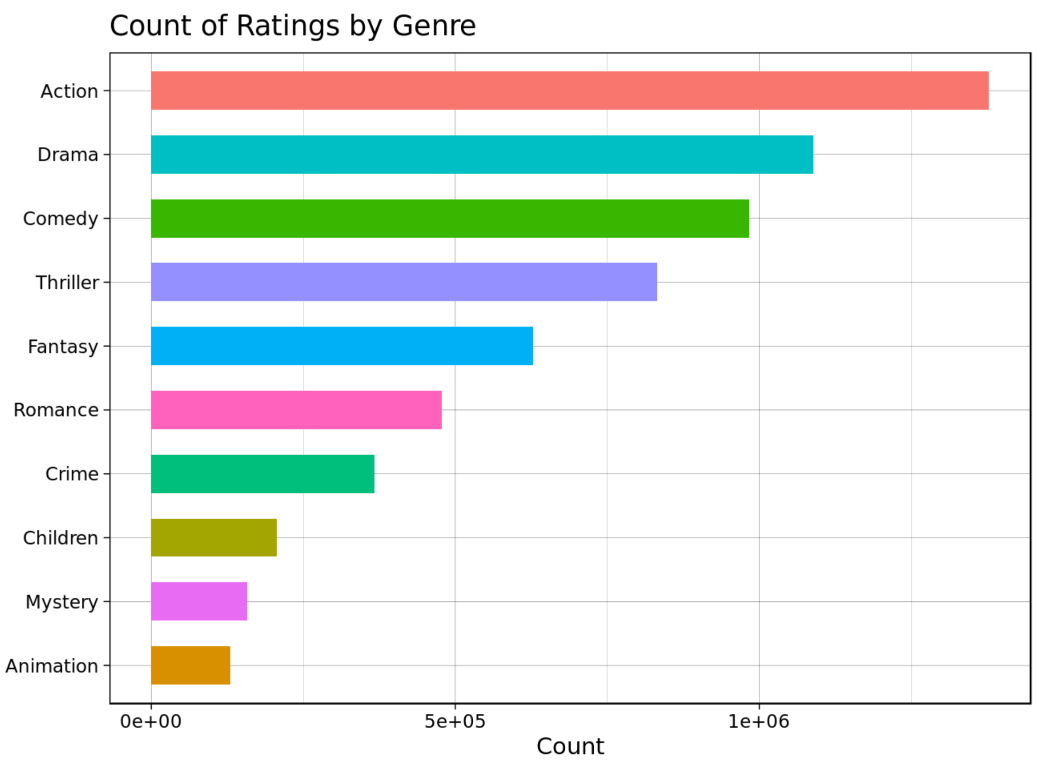
And the table of frequencies changes to:

```
## [1] " Genres count of distinct values : "
```

```
## .
##             Action              Drama             Comedy
##            1377587            1089529             983341
##           Thriller            Fantasy            Romance
##             832709             627752             477216
##              Crime           Children            Mystery
##             367340             206636             158213
##          Animation            Musical            Western
##             130114             120953              52405
##          Film-Noir        Documentary               IMAX
##              33373              25368               2104
## (no genres listed)
##                  2
```

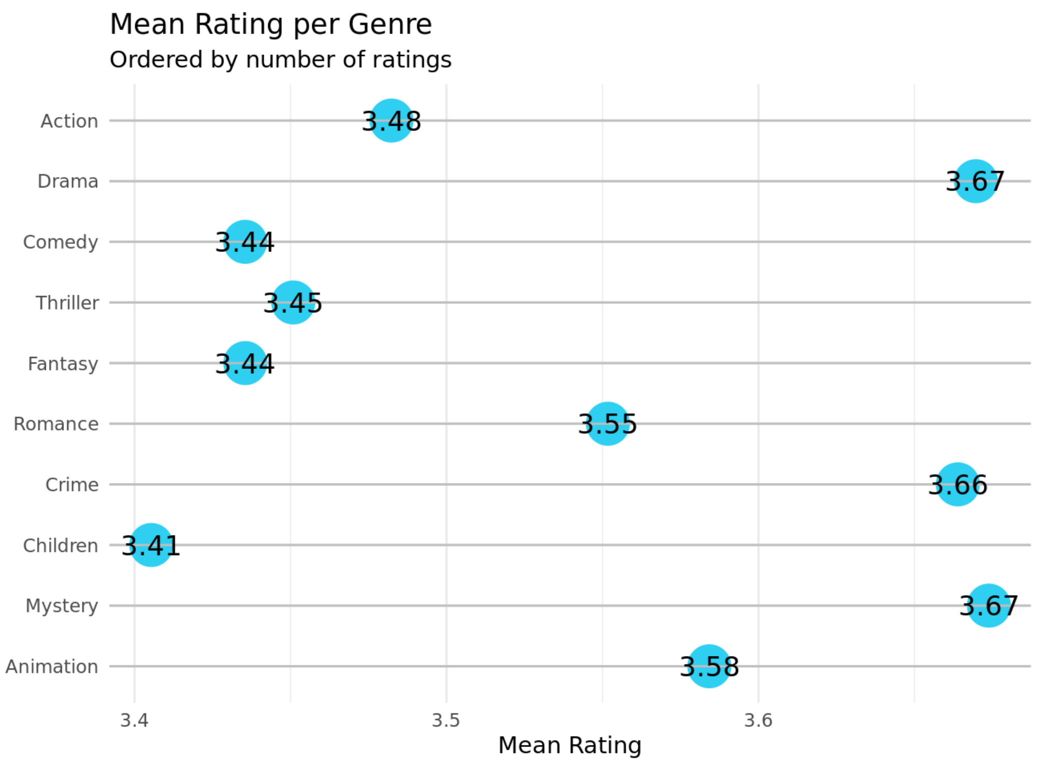
10 Most Frequent Genres

Which genres people rate most often? Let’s take a look the 10 most popular genres after the aforementioned genre merge.



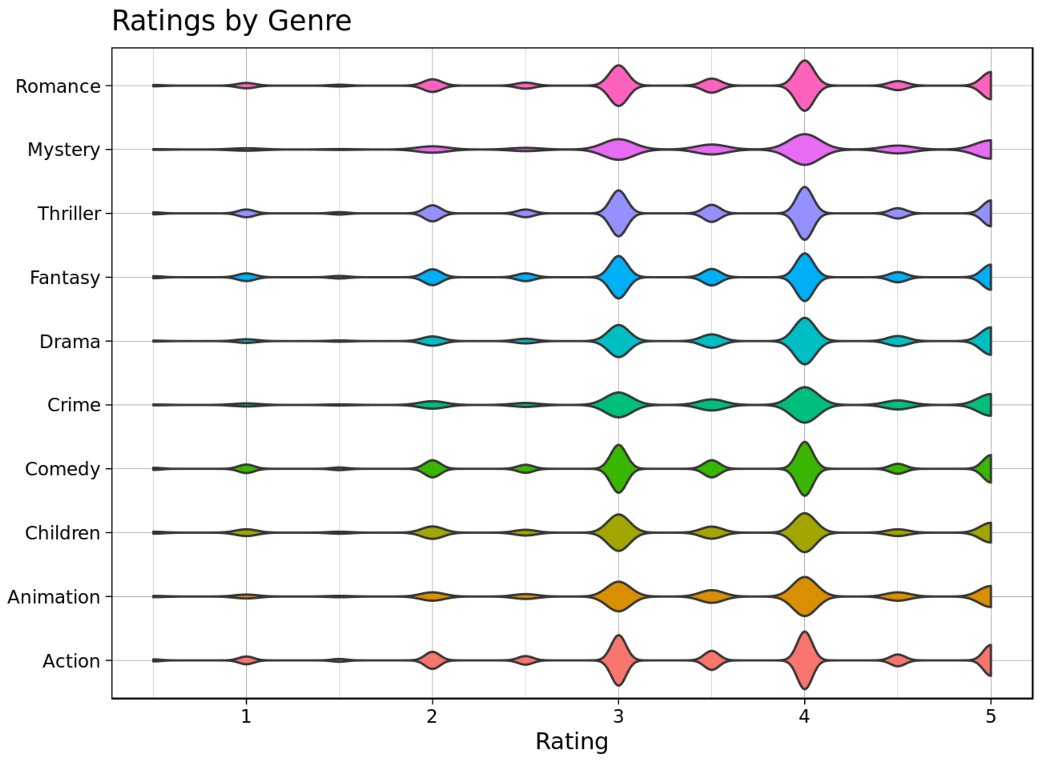
Ratings by Genre

Does popularity come with high ratings? We will create a plot where genres are in the same order, but this time we count their average rating.



We can't spot a pattern here. Popularity has not much to do with the average rating.

To further examine it, we will also make a violinplot to take a look at the distribution of ratings for each genre. This way it can become more clear which values shaped these means.

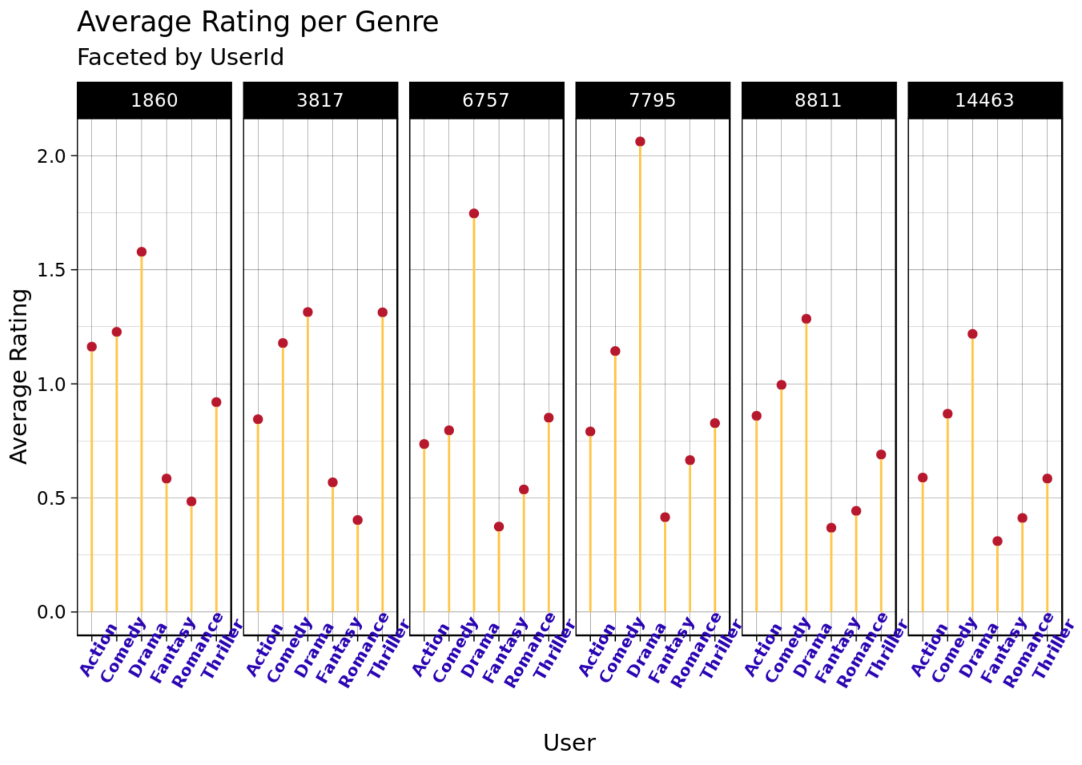


Distributions of ratings along genres look pretty identical. This comes of no surprise since means were not much different. In addition, all genres contain both good and bad movies, hence the wide distribution.

As a result, it looks like genre alone is not an efficient predictor. To know the genre of a movie seems insufficient to predict its rating.

Perhaps genres need to be examined along with the user who places the rating. To investigate it, we will employ a lollipop plot to make clear whether mean ratings per genre vary by userId.

Because genres are far too many to form informative plots we will keep the top 6 by number of ratings. The same will apply to users.



Pretty enlightening! No user has a balance in their preferences. Some love comedy, others like action, etc. That's a strong hindsight for the presence of hidden "concepts" in our data.

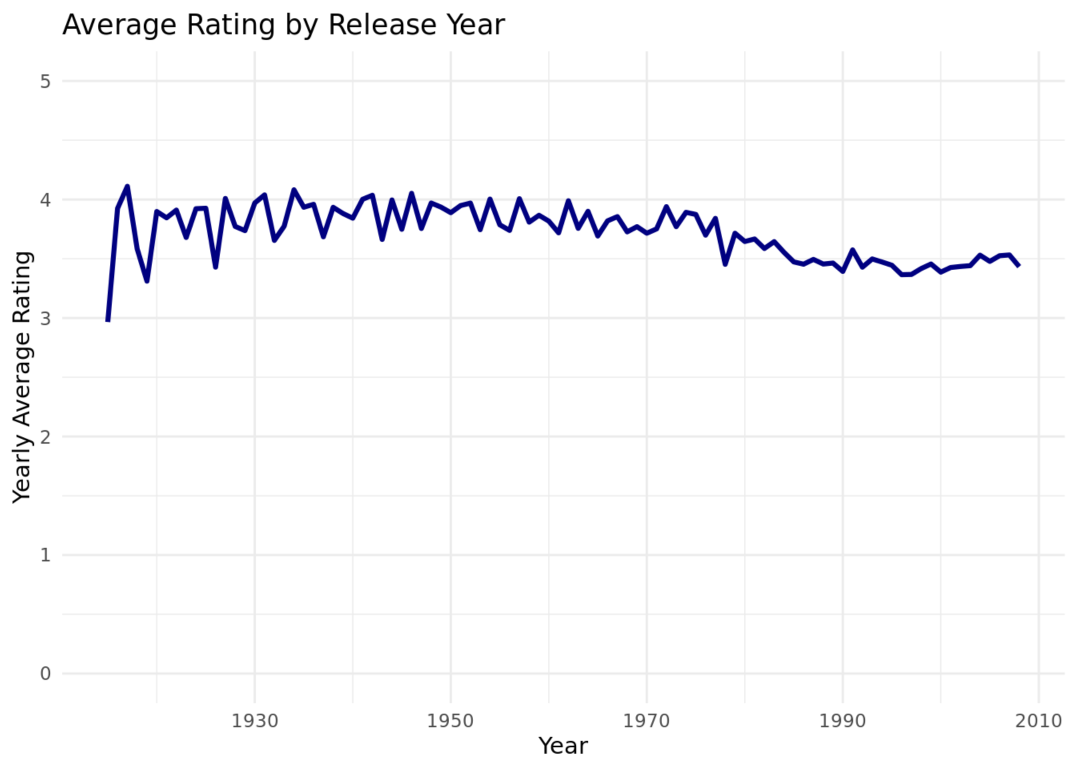
We will deploy this insight later by applying SVD in an effort to unravel such latent factors.

Time

Is time an effective predictor of rating? Does it matter when a movie is released or rated? Let's explore accordingly.

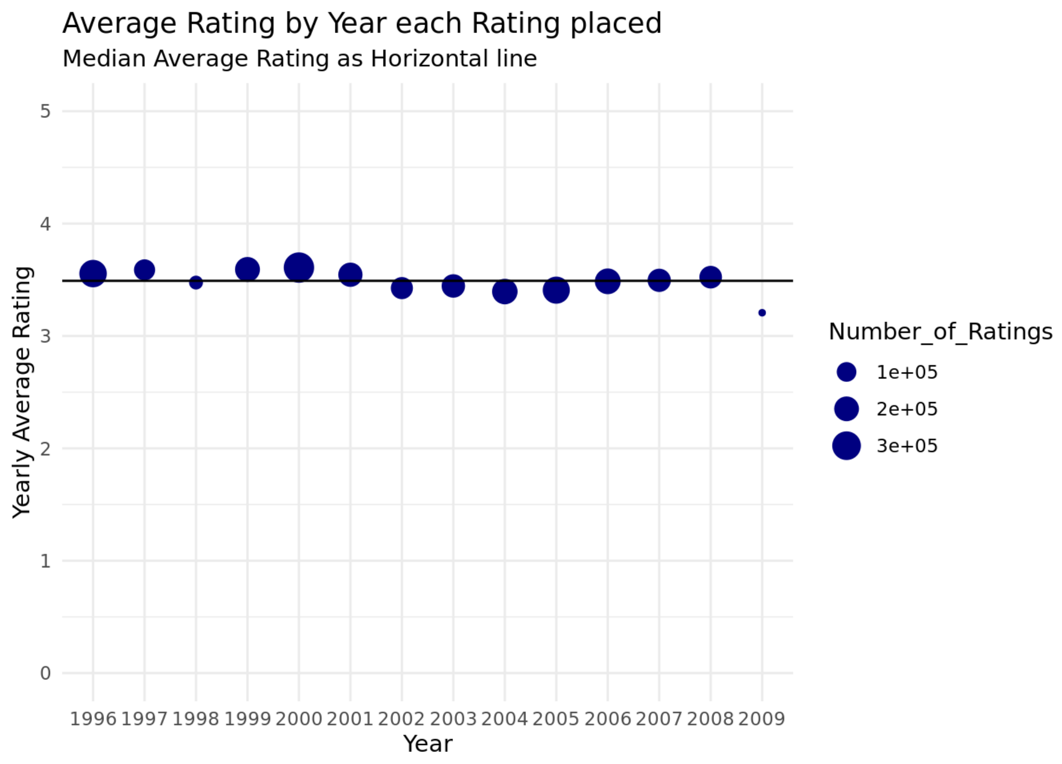
Of Release

To earn a hindsight about the release date, let's visualize the mean rating of each year



As expected, no apparent pattern.

Of Rating



Again, no apparent pattern. Seems like the time variable is not so valuable to us and it does not qualify for the modelling section.

Modelling

The time has come to start creating our predictive algorithm.

Because data is vast, regression models will take forever to train. As an alternative, we will construct a similar formula logically step-by-step.

For that, we will make use of only 3 columns: *userID*, *movieId* and *rating*.

```
movielens <- movielens %>% select( c(movieId,userId,rating) )
```

Train/Test Split

The RMSE of our model will calculated on the validation set. To avoid overfitting, we will not use it to optimize our model's parameters. Instead, we will split the initial train set to create a test set out of it that will help us optimize.

Train / Validation Split

We will perform a 90/10 split. However, the split is not enough. We need to make sure that all users and movies contained in test set exist in the train set as well. Otherwise, our algorithm will be unable to predict a rating and will return an N/A.

As a result, we need to remove some rows from test set. However, we will not let these rows get wasted. We will add them in the train set instead. Even though they won't be directly deployed, they will contribute to the unravelling of concepts.

To make this clear we will use an extremely simplified example. Suppose all rows intended for removal were thrillers. Users keen on thrillers would have given high ratings to these, while others with a preference to romance would have rated them low. This set of movies would greatly contribute to the latent factor responding to the thriller genre and would have helped our algorithm to subtly categorize each user's preference on it.

```
set.seed(1)
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
     semi_join(edx, by = "movieId") %>%
     semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

mu <- mean(edx$rating)

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

Train / Test Split

As said above, in order to optimize without using the validation set, we will conduct an 80/20 split on the existing train set. For speed of computation, we won't split the whole train set, but 200,000 rows of it.

```
set.seed(1)
# if using R 3.5 or earlier, use `set.seed(1)` instead

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.8, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test set back into train set

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

User and Movie Bias

Now we need to compute each movie's and user's bias. In order to control for the number of ratings, we need to penalize by term commonly referred to as lambda. To find the optimal lambda, we create a function that lets us that takes as input our dataframe and a vector of possible lambdas, computes the RMSE each lambda returned and outputs the lambda that yields the lowest rmse.

```
l_optimizer <- function(train_set, lambdas){

    rmses <- sapply(lambdas, function(l){

        #Calculate movie bias
        b_i <- train_set %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))

        #Calculate user bias
        b_u <- train_set %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))

        #Predict ratings
        predicted_ratings <- test_set %>%
            left_join(b_i, by='movieId') %>%
            left_join(b_u, by = "userId") %>%
            mutate(pred = mu + b_i + b_u) %>%
            pull(pred)

        #Calculate the error for the specific lambda
        return(RMSE(predicted_ratings, test_set$rating))
    })

 return(lambdas[which.min(rmses)])
}
```

```
## [1] "Optimal Lambda : 4"
```

Now that we know the optimal lambda, we will go on to create the biases that will lead us to the residuals we seek to decompose.

```
#Calculate movie bias
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

#Calculate user bias
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
```

Residuals

Now that we obtained the regularized user and movie biases, we need to calculate what is left unexplained, ie the residual values. In other words, we need to compute all differences between actual ratings and our predictions so that we can further analyze these differences.

```
#Function to calculate the residuals

resids <- function( train_set = train_set, mu = mu ){

  #This function takes training set, calculates the residuals left after subtracting mu, user and movie bias
  #from actual rating and transforms it into a sparse matrix with movieIds as columns and users as row

  df <- train_set %>%
           left_join(b_i, by = "movieId") %>%
           left_join(b_u, by = "userId") %>%
           mutate(res = rating - (mu + b_i + b_u) ) %>%
           select(userId, movieId, res) %>%
           spread( movieId, res ) %>% as.data.frame()

  #Name the rows
  rownames(df)<- df[,1]
  df <- df[,-1]
}
```

SVD

Now that we have the matrix of residuals, we de compose it in the best possible way. For that reason we will use the funkSVD function from the recommenderlab package. As its name suggests, it is not the tradiotional SVD. It uses a slightly different method, decomposing the initial matrix into two (and not three) matrices containing the latent factors for users and movies that minimize a specific objective function. For more details, refer [here](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems).

Because it contains many parameters, we will build a function that optimizes some of them. These parameters include: * train set (named tr) : the train set matrix of residuals. It will get split in in an 80/20 to create a test set out of it that will help us estimate the final RMSE * test set (named validation) : test set on which we want to predict * ks : number of features, or ranks of the approximation. In other words, how many "concepts" do we seek to unravel. A variable with high computational cost. * gammas : regularization term * lambdas : the rate by which we update our weights after each iteration. A low lambda can make our algorithm train for days, while a high one can lead us to miss the optimal weights. * min_improvement : the minimum amount of improvement derived from our objective that can be considered as improvement * min_epochs : minimum number of iterations per feature

```r
optimizer <- function(tr = tr,
                      test_set = test_set,
                      ks,gammas,lambdas,min_epochs, min_improvements){

  #Inputs: parameters of funkSVD function
  d <- data.frame(k = NULL,
                 gamma = NULL,
                 lambda = NULL,
                 min_epochs = NULL,
                 min_improvement = NULL,
                 rmse = NULL)

        for (k in ks){
        for (g in gammas){
        for (l in lambdas){
        for (e in min_epochs){
        for (imp in min_improvements) {

          #decompose residuals with funk SVD
          a <- funkSVD(tr,
                       k = k,
                       gamma = g,
                       lambda = l,
                       min_epochs = e,
                       max_epochs = 200,
                       min_improvement = imp,
                       verbose = TRUE )

           #recompose them (returns a full matrix in place of the sparse tr)
           r <- tcrossprod(a$U, a$V)

           #pass the original colnames to the new matrix
           colnames(r) <- colnames(tr)

           #create a new vector (called re) on test set
           #containing the appropriate p * q (recomposed) term
           test_set$re <- seq(0,nrow(test_set)-1) #to-be-filled vector of zeros

           #for each row of test set
           for (i in 1:nrow(test_set)){

              #fill the vector of zeros with the proper calculated p*q quantity
              test_set$re[i] <-  r[ test_set$userId[i] , which(test_set$movieId[i] == colnames(r)) ]

                }

              #calculate our prediction
              tes <- test_set %>%
              left_join(b_i, by = "movieId") %>% #bring the movie bias
              left_join(b_u, by = "userId") %>% #bring the user bias
              mutate(pred = mu + b_i + b_u + re) # calculate our prediction

              #store results in a data frame
              d <- d %>% rbind(data.frame(k = k, gamma = g, lambda = l,
                                          min_epochs = e, min_improvement = imp,
                                          rmse = ModelMetrics::rmse(tes$rating, tes$pred) ) )

              }}}}}

  return(d)
}
```

Now that we defined our function, we also define the parameters we want to experiment with.

For ease of computations, we won't use the optimizer in knitting. I've run the optimization process several times with smaller datasets. In case you want to further optimize them, re-run the script with you own preferences.

```
#Define the parameters
ks <- c(1)
gammas <-  c(.009)
lambdas <- c(.005)
min_epochs <- c(5)
min_improvements <- c(.001)
```

Results

Then, we

- run our optimizer,
- store optimizer results in dataframe
- look at the optimal parameters that gave the minimum RMSE.

```
## [1] "Optimal Parameters : "
```

```
## [1] 1.000 0.009 0.005 5.000 0.001
```

The optimizer function can also be used to predict on the validation set. We will re-run it, but this time with:

- edx as train test, (after we pass it from the function that returns the residuals)
- validation as test_set
- the rest of (optimal) parameters from the data frame we created above

```
#subtract the mean, movie and user biases
tr <- resids(train_set = edx, mu=mu)

final_RMSE <- optimizer( tr =  tr,
                         test_set = validation,
                         ks = opt_params[1],
                         gammas = opt_params[2],
                         lambdas = opt_params[3],
                         min_epochs = opt_params[4],
                         min_improvements = opt_params[5])
```

```
##
## Training feature: 1 / 1 : .....
## ->  5 epochs - final improvement was 8.461297e-05
```

```
print("Final RMSE:")
```

```
## [1] "Final RMSE:"
```

```
final_RMSE$rmse %>% print()
```

```
## [1] 0.8655681
```

As you can see, torturing data enough gave us an RMSE of 0.8655681.

Conclusion

Throughtout this report, we tried to explore movie ratings under different spectrums in order to come up with an efficient and computationally affordable movie recommender.

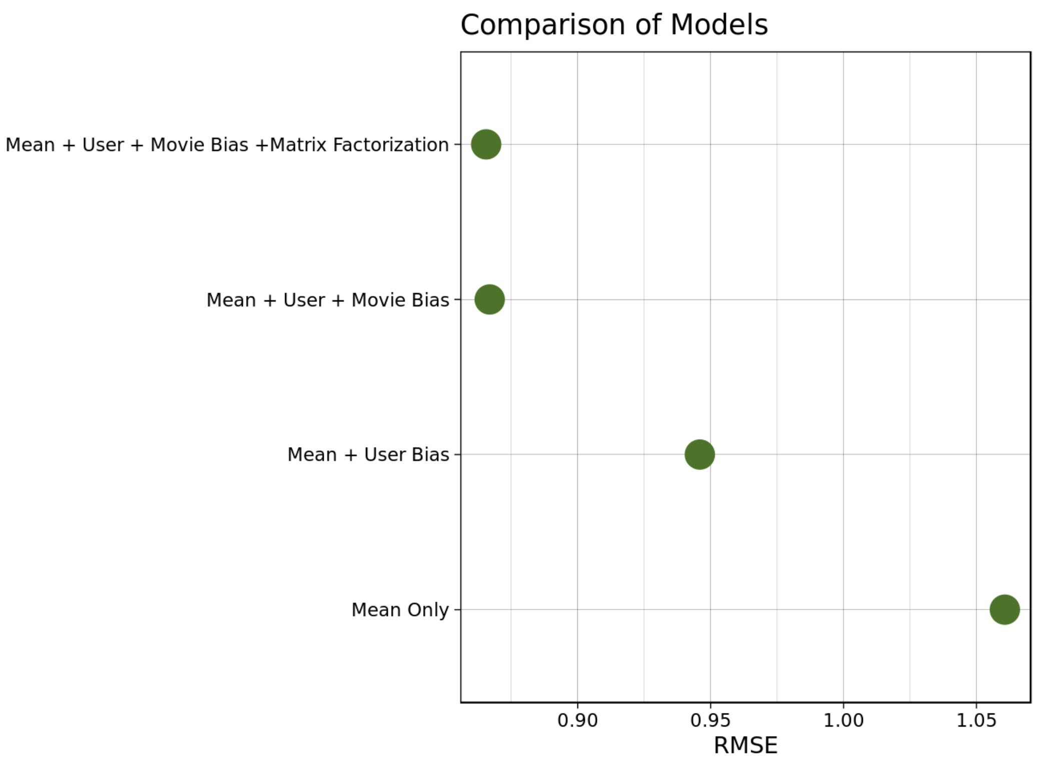
At first,we took a brief look at their **distribution** and noticed a tendency towards **high ratings**, since both mean and median rating were over 2.5.

Then, we visualised the mean rating in correlation with the number of ratings both for movies and for users. Despite users showed a randomly scattered plot, movies exhibited a slight positive tendency. That led us to explore the **mean** rating **in relation with the number of ratings** each movie had. This step made clear that we shouldn't rely on the mean rating unless it was shaped by a safe number of ratings and introduced us to the **need to regularize each movie's mean rating**. The issue was tackled later with an optimization function that detects the optimal regularization parameter.

Then, we gravitated towards the **Genres** column. After we brought it in a proper format, we started exploring the mean rating per genre in relation with its popularity. Overall, genres didn't appear so discriminating. They pretty much held the **same distribution of ratings** and their respective means were close with each other. But since we put them under the **perspective of user's preference**, they suddenly became insightful. It became clear that each user had a tendency to rate higher (lower) certain genres and preferences among users varied. This indicated that genres contained valuable information. If we knew the preferences of a user and how much each movie belonged to these preferences, we could embed these preferences in our model and substantially improve its predictive power. For that, we employed the matrix factorization, which helped us abstractly unveil such preferences.

With all this hindsight, we started creating our model. After we regularized each movie and user mean rating (aka **bias**), we focused on exploiting the residuals of this precocious model. For that, we employed the FunkSVD function, tuned by a subsplit of our original train set. Finally, we plotted each RMSE obtained by each combination of parameters and chose the optimal to train our final model.

A short description of the results is embedded in the following plot.



Appendix (Code)

```
#Importing necessary libraries

#Data manipulation
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
# Train/Test split and Model Training
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
#Handle Tables
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
#SVD decomposition
if(!require(recommenderlab)) install.packages("recommenderlab",
                                         repos = "http://cran.us.r-project.org")
# to plot multiple grobs alongside
if(!require(gridExtra)) install.packages("gridExtra",
                                         repos = "http://cran.us.r-project.org")
# to separate stacked genres
if(!require(splitstackshape))install.packages("splitstackshape",
                                         repos = "http://cran.us.r-project.org")
#to transform shape of our data
if(!require(reshape2)) install.packages("reshape2",
                                         repos = "http://cran.us.r-project.org")
#For rmse function
if(!require(ModelMetrics))  install.packages("ModelMetrics",
                                      repos = "http://cran.us.r-project.org")
```

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                          title = as.character(title),
                                          genres = as.character(genres))

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

#if you want to run the whole dataset, disable the following command
movielens <- movielens %>% slice(1:floor(nrow(movielens)/4))

rm( ratings, movies )


movielens$timestamp <- movielens$timestamp %>% as.Date.POSIXct()

movielens$timestamp_year <- movielens$timestamp %>% year()

movielens <- movielens %>%
    mutate_if(is.integer, as.factor) #integers to factors

movielens$year <- movielens %>%
  pull(title) %>%
  str_extract("\\([:digit:]{4}\\)") %>%
  str_remove_all("\\(|\\)") %>%
  as.integer()


movielens$title <-movielens$title %>% str_remove("\\([:digit:]{4}\\)")



mu <- mean(movielens$rating)
med <- median(movielens$rating)


movielens %>% ggplot( aes(x=rating)) +
   geom_density(aes(y=..density..), fill = "#99beff", col = '#d9a757', alpha = .2  )+
   geom_histogram(aes(y=..density..),binwidth = .5,fill="#498ee3")+
    geom_vline( aes( xintercept = mu, colour = "mean"), linetype ='dashed' ) +
    geom_vline(aes( xintercept = med, colour = "median"), linetype ='dashed' )   +
    theme_minimal()+
    labs(title = "Histogram and Density of Ratings", x = "Rating", y = "Density")+
    theme(legend.title = element_blank())




#Create a new dataframe containing the count of each user
cnt <- movielens %>% group_by(userId) %>% summarise( user_cnt = n() )

#average rating of each user
rat <- movielens %>% group_by(userId) %>% summarise( mean_rating = mean(rating) )

p1 <- cnt %>% left_join( rat, by = "userId" ) %>%
    ggplot( aes(y = user_cnt, x = mean_rating ) ) +
    geom_point(show.legend = F, alpha = .1,col = "#4685f2") +
    labs( title = "Count of User vs Mean_Rating", x = "", y = "Times each user rated" )

#count of each movie
cnt_mov<- movielens %>% group_by(movieId) %>% summarise( movie_cnt = n() )

#average rating of each movie
rat_mov <- movielens %>% group_by(movieId) %>% summarise( mean_rating = mean(rating) )

mov_df <- cnt_mov %>% left_join(rat_mov, by = "movieId" )
```

```r
p2 <-  mov_df %>%
    ggplot( aes(y = movie_cnt, x = mean_rating ) ) +
    geom_point(show.legend = F, alpha = .1,col = "#4685f2") +
    labs( title = "Count of Movie vs Mean_Rating", x = "Mean Ratings", y = "Times each movie was rated" )

grid.arrange(p1,p2)

rm(cnt,rat,cnt_mov,rat_mov,p1,p2)


#Add the title from original dataset
mov_df$title <- movielens[match(mov_df$movieId, movielens$movieId),"title"]

worst_movies <- mov_df %>%
    arrange((mean_rating)) %>% #arrange from highest to lowest rating
    slice(1:10) %>% #grab the first 10
    ggplot( aes(x = reorder(title,mean_rating), y = mean_rating) ) +
    geom_point(size = 9, col = "#baaa6e") +
    coord_flip() +
    geom_vline(xintercept = 1:10, col = "grey80") + #lines on which our dots "walk"
    theme_linedraw()+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = mean_rating), col = "black" ) +
    labs(title = "Worst Movies", subtitle =  "Regardless of the Number of Ratings" , y = "Mean Rating", x =
"")

best_movies <- mov_df %>%
    arrange(desc(mean_rating)) %>% #arrange from highest to lowest rating
    slice(1:10) %>% #grab the first 10
    ggplot( aes(x = reorder(title,mean_rating), y = mean_rating) ) +
    geom_point(size = 9, col = "#ceabd1") +
    coord_flip() +
    geom_vline(xintercept = 1:10, col = "grey80") + #lines on which our dots "walk"
    theme_linedraw()+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = mean_rating), col = "black" ) +
    labs(title = "Best Movies", subtitle =  "Regardless of the Number of Ratings" , y = "Mean Rating", x = "
")

grid.arrange(best_movies, worst_movies)

worst_movies <- mov_df %>%
    arrange((mean_rating)) %>% #arrange from highest to lowest rating
    slice(1:10) %>% #grab the first 10
    ggplot( aes(x = reorder(title,mean_rating), y = mean_rating) ) +
    geom_point(size = 9, col = "#baaa6e") +
    coord_flip() +
    geom_vline(xintercept = 1:10, col = "grey80") + #lines on which our dots "walk"
    theme_linedraw()+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = movie_cnt), col = "black" ) +
    labs(title = "Worst Movies", subtitle =  "Regardless of the Number of Ratings" , y = "Mean Rating", x =
"")

best_movies <- mov_df %>%
    arrange(desc(mean_rating)) %>% #arrange from highest to lowest rating
    slice(1:10) %>% #grab the first 10
    ggplot( aes(x = reorder(title,mean_rating), y = mean_rating) ) +
    geom_point(size = 9, col = "#ceabd1") +
    coord_flip() +
    geom_vline(xintercept = 1:10, col = "grey80") + #lines on which our dots "walk"
    theme_linedraw()+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = movie_cnt), col = "black" ) +
    labs(title = "Best Movies", subtitle =  "Regardless of the Number of Ratings" , y = "Mean Rating", x = "
")

grid.arrange(best_movies, worst_movies)

#pick the 1rd quantile of the movie count as a low limit
limit <- (summary(mov_df$movie_cnt))[5]
```

```r
best_movies <- mov_df %>%
    filter(movie_cnt > limit) %>% #throw away movies with few ratings
    arrange(desc(mean_rating)) %>% #arrange from highest to lowest rating
    slice(1:10) %>% #grab the first 10
    ggplot( aes(x = reorder(title,mean_rating), y = mean_rating) ) +
    geom_point(size = 9, col = "#ceabd1") + coord_flip() +
    geom_vline(xintercept = 1:10, col = "grey80") + #lines on which our dots "walk"
    theme_classic()+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = movie_cnt), col = "black" ) +
    labs(title = paste("Best Movies with more than",limit,"Ratings"), y = "Mean Rating", x = "", subtitle =
"Number of Ratings inside the Dots" )

worst_movies <- mov_df %>%
    filter(movie_cnt > limit) %>% #throw away movies with few ratings
    arrange((mean_rating)) %>% #arrange from highest to lowest rating
    slice(1:10) %>% #grab the first 10
    ggplot( aes(x = reorder(title,mean_rating), y = mean_rating) ) +
    geom_point(size = 9, col = "#baaa6e") + coord_flip() +
    geom_vline(xintercept = 1:10, col = "grey80") + #lines on which our dots "walk"
    theme_classic()+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = movie_cnt), col = "black" ) +
    labs(title = paste("Worst Movies with more than",limit,"Ratings"), y = "Mean Rating", x = "", subtitle
= "Number of Ratings inside the Dots" )

grid.arrange(best_movies, worst_movies)
rm(mov_df,best_movies,worst_movies)


#For every genre entry create a separate row containing only this entry
movielens_genres <- movielens %>%
        cSplit( "genres", #split the genres column
               sep="|", #on separator |
               direction =  "long" ) #towards a long direction

print(" Genres count of distinct values : ")
movielens_genres %>% pull(genres) %>% table() %>% sort(decreasing = TRUE) %>% print()

movielens_genres$genres <- plyr::revalue(movielens_genres$genres, c("Horror" = "Thriller",
                                                                    "Adventure" = "Action",
                                                                    "War" = "Action" ,
                                                                    "Sci-Fi" = "Fantasy"))

print(" Genres count of distinct values : ")
movielens_genres %>% pull(genres) %>% table() %>% sort(decreasing = TRUE) %>% print()

pop_genres <- movielens_genres %>%
    group_by( genres ) %>%
    summarize( n = n() ) %>%
    arrange(desc(n)) %>%
    head(10)

pop_genres %>%
    ggplot( aes(x = reorder(genres,n), y = n) ) +
    geom_bar( aes(fill = genres), stat = "identity",width = .6, show.legend = FALSE)+
    coord_flip()+
    theme_linedraw()+
    labs( title = "Count of Ratings by Genre", y = "Count", x = ""  )

movielens_genres %>%
    filter(genres %in% pop_genres$genres ) %>%
    group_by(genres) %>%
    summarize(mean_rating = mean(rating)) %>%
    mutate( n = pop_genres$n[(match(genres, pop_genres$genres))] ) %>%
    ggplot( aes( x = reorder(genres,n), y = mean_rating ) )+
    geom_point(size = 9, col = "#2ecff0") +
    geom_vline( xintercept = 1:10, col = "gray" )+
    theme(axis.line = element_blank()) +
    geom_text( aes(label = round(mean_rating,2), size = 8 ), col = "black", show.legend = FALSE ) +
    coord_flip() +
    theme_minimal() +
    labs( title = "Mean Rating per Genre", subtitle = "Ordered by number of ratings", x = "", y = "Mean Rat
ing" )

movielens_genres %>%
    filter(genres %in% pop_genres$genres ) %>%
    ggplot() +
    geom_violin( aes( x = genres, y = rating, fill = genres ), show.legend = FALSE ) +
    coord_flip() +
    labs( title = "Ratings by Genre", x = "", y = "Rating" ) +
    theme_linedraw()

#store top genres and users in variables for ease of filtering
top_genres <- movielens_genres %>%
    group_by( genres ) %>%
    summarize( n = n() ) %>%
    arrange(n%>% desc()) %>%
    head(6) %>%
    pull(genres)

top_users <- movielens_genres %>%
    group_by(userId) %>%
    summarize( user_count = n() ) %>%
    arrange(desc(user_count)) %>%
    head(6) %>%
    pull(userId)

movielens_genres <- movielens_genres %>%
    filter(genres %in% top_genres, userId %in% top_users)#keep only top genres and users


#transform data in wide format
movielens_genres <- movielens_genres %>%
    dcast(userId + movieId + title + rating ~ genres,
          fun.aggregate = length,
          value.var = "rating")  #convert to dummy columns

#Replace 1's with the actual rating
movielens_genres[,4:ncol(movielens_genres)] <- movielens_genres[,4:ncol(movielens_genres)] %>%
    mutate_all( ~case_when( . !=0 ~ rating, . ==0 ~ 0 ) ) #whenever each dummy column is not zero, place the
equivalent rating

 movielens_genres[,c(1,5:10)] %>%
   group_by(userId) %>%
   summarize_all(mean)  %>% #compute the mean of each user in each genre
    gather(key = "Genre", value = "Average_Rating" , -userId) %>% #gather it for easier plotting
    ggplot( aes(x =  Genre, y = Average_Rating)) +
    geom_point(col = "#b8182d") +
    facet_grid(~userId) +
    geom_segment(aes(x = Genre, y = 0, xend = Genre, yend = Average_Rating-.02), color = "#ffc64a") +
    labs(title = "Average Rating per Genre", subtitle = "Faceted by UserId", y = "Average Rating", x = "User
")+
    theme_linedraw()+
    theme(axis.text.x = element_text(face = "bold", color = "#2b05b3", size = 8, angle = 60))

rm(movielens_genres,top_genres,top_users, pop_genres)



movielens %>% group_by(year) %>%
  summarize( average_rating = mean(rating) ) %>%
  ggplot(aes(x = year , y = average_rating) ) +
  geom_line(lwd = 1.1, col = "navyblue")+
  #geom_point( aes(size  ) +
  theme_minimal() +
  scale_y_continuous(limits = c(0,5))+
  labs(y = "Yearly Average Rating",x = "Year", title = "Average Rating by Release Year")



#medi <- median(movielens$)

movielens %>% group_by( timestamp_year ) %>%
  summarise(avg_rating = mean(rating), Number_of_Ratings = n()) -> temp
```

```r
medi <- median(temp$avg_rating)

temp %>%
    ggplot( aes( x = timestamp_year, y = avg_rating ) ) +
  #geom_line(lwd = 1.1, col = "navyblue")+
  geom_point(aes(size = Number_of_Ratings ), col = "navyblue")+
  geom_hline(  yintercept = medi)+
  theme_minimal() +
  scale_y_continuous(limits = c(0,5))+
  labs(y = "Yearly Average Rating",x = "Year",
       title = "Average Rating by Year each Rating placed",
       subtitle = "Median Average Rating as Horizontal line")



movielens <- movielens %>% select( c(movieId,userId,rating) )

set.seed(1)
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

validation <- temp %>%
     semi_join(edx, by = "movieId") %>%
     semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

mu <- mean(edx$rating)

rm(test_index, temp, movielens, removed)

set.seed(1)
# if using R 3.5 or earlier, use `set.seed(1)` instead

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.8, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test set back into train set

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)

l_optimizer <- function(train_set, lambdas){

    rmses <- sapply(lambdas, function(l){

        #Calculate movie bias
        b_i <- train_set %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))

        #Calculate user bias
        b_u <- train_set %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))

        #Predict ratings
        predicted_ratings <- test_set %>%
            left_join(b_i, by='movieId') %>%
             left_join(b_u, by = "userId") %>%
             mutate(pred = mu + b_i + b_u) %>%
             pull(pred)

         #Calculate the error for the specific lambda
         return(RMSE(predicted_ratings, test_set$rating))
     })

 return(lambdas[which.min(rmses)])
}

#A vector of possible lambdas
lambdas <- seq(0, 5, 0.5)

#Assign the optimal lambda to a variable
lambda <- l_optimizer(train_set, lambdas)

#Take a look at it
print(paste("Optimal Lambda :",lambda))

#Calculate movie bias
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

#Calculate user bias
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

#Function to calculate the residuals

resids <- function( train_set = train_set, mu = mu ){

  #This function takes training set, calculates the residuals left after subtracting mu, user and movie bias
  #from actual rating and transforms it into a sparse matrix with movieIds as columns and users as row

  df <- train_set %>%
            left_join(b_i, by = "movieId") %>%
            left_join(b_u, by = "userId") %>%
            mutate(res = rating - (mu + b_i + b_u) ) %>%
            select(userId, movieId, res) %>%
            spread( movieId, res ) %>% as.data.frame()

  #Name the rows
  rownames(df)<- df[,1]
  df <- df[,-1]
}

#convert our train set to residuals
#tr <- resids(train_set = train_set,mu=mu)

#A quick look at the residual values
#print("Residuals of the first 10 users and movies")
#tr[1:10,1:10] %>% print()

optimizer <- function(tr = tr,
                      test_set = test_set,
                      ks,gammas,lambdas,min_epochs, min_improvements){

  #Inputs: parameters of funkSVD function
  d <- data.frame(k = NULL,
                  gamma = NULL,
                  lambda = NULL,
                  min_epochs = NULL,
                  min_improvement = NULL,
                  rmse = NULL)

        for (k in ks){
        for (g in gammas){
        for (l in lambdas){
        for (e in min_epochs){
```

```
        for (imp in min_improvements) {

          #decompose residuals with funk SVD
          a <- funkSVD(tr,
                       k = k,
                       gamma = g,
                       lambda = l,
                       min_epochs = e,
                       max_epochs = 200,
                       min_improvement = imp,
                       verbose = TRUE )

           #recompose them (returns a full matrix in place of the sparse tr)
           r <- tcrossprod(a$U, a$V)

           #pass the original colnames to the new matrix
           colnames(r) <- colnames(tr)

           #create a new vector (called re) on test set
           #containing the appropriate p * q (recomposed) term
           test_set$re <- seq(0,nrow(test_set)-1) #to-be-filled vector of zeros

           #for each row of test set
           for (i in 1:nrow(test_set)){

              #fill the vector of zeros with the proper calculated p*q quantity
              test_set$re[i] <-  r[ test_set$userId[i] , which(test_set$movieId[i] == colnames(r)) ]

                }

              #calculate our prediction
              tes <- test_set %>%
              left_join(b_i, by = "movieId") %>% #bring the movie bias
              left_join(b_u, by = "userId") %>% #bring the user bias
              mutate(pred = mu + b_i + b_u + re) # calculate our prediction

              #store results in a data frame
              d <- d %>% rbind(data.frame(k = k, gamma = g, lambda = l,
                                          min_epochs = e, min_improvement = imp,
                                          rmse = ModelMetrics::rmse(tes$rating, tes$pred) ) )

              }}}}}

  return(d)
}

#Define the parameters
ks <- c(1)
gammas <-  c(.009)
lambdas <- c(.005)
min_epochs <- c(5)
min_improvements <- c(.001)

optimization <- FALSE

if(optimization){
  results <- optimizer(tr = tr, test_set = test_set, ks,gammas,lambdas,min_epochs, min_improvements)

  results %>%
    ggplot( aes( x = reorder(k,rmse) , y = (rmse)) ) +
    geom_point(aes(col = as.factor(gamma) ), size = 9, alpha = .8) +
    coord_flip() +
    geom_vline(xintercept = 1:nrow(results), col = "grey80") + #lines on which our dots "walk"
    theme_minimal()+
    theme(axis.line = element_blank()) +
    labs(title = "RMSE results", subtitle =  "" , y = "RMSE", x = "Ranks", col = "Gamma", shape = "Lambda")

  opt_params <- results[which(results$rmse == min(results$rmse)),]


}else{

  results <- c(ks,gammas,lambdas,min_epochs,min_improvements)
  results <- c(ks,gammas,lambdas,min_epochs,min_improvements)

  opt_params <- results
}


"Optimal Parameters : " %>% print()
opt_params %>% print()
rm(train_set,test_set)

#subtract the mean, movie and user biases
tr <- resids(train_set = edx, mu=mu)

final_RMSE <- optimizer( tr =  tr,
                         test_set = validation,
                         ks = opt_params[1],
                         gammas = opt_params[2],
                         lambdas = opt_params[3],
                         min_epochs = opt_params[4],
                         min_improvements = opt_params[5])

print("Final RMSE:")
final_RMSE$rmse %>% print()


#Just the mean
mean_only <- rmse(validation$rating,rep(mu,nrow(validation)))

#Adding the user bias

pred_bu <- validation %>%
           left_join(b_i, by = "movieId") %>%
           mutate(pred = (mu + b_i) ) %>%
           pull(pred)

mean_bu <- rmse(validation$rating,pred_bu)

pred_bu_bi <- validation %>%
           left_join(b_i, by = "movieId") %>%
           left_join(b_u, by = "userId") %>%
           mutate(pred = (mu + b_i + b_u) ) %>%
           pull(pred)

mean_bu_bi <- rmse(validation$rating,pred_bu_bi)

data.frame(model = c("Mean Only","Mean + User Bias", "Mean + User + Movie Bias", "Mean + User + Movie Bias
+Matrix Factorization") ,
 rmses = c(mean_only, mean_bu, mean_bu_bi, final_RMSE$rmse)) %>%
  ggplot( aes(x = model, y = rmses) ) +
  geom_point( size = 6, col = "#4c7329" ) +
  coord_flip()+
  theme_linedraw()+
  labs( title = "Comparison of Models", y = "RMSE", x ="")
```