MovieLens Project Report - Capstone 1

Klim Popov

5/16/2020

Abstract

This project aims to analyze the provided MovieLens dataset and to compare various types of approaches for a recommendation system, as well as various reasons for the limitation of such analysis.

This paper consists of the following parts:

Overview section describes the dataset and summarizes the goal of the project and key steps that were performed;

Methods section explains the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and modeling approach;

Results section that presents the modeling results and discusses the model performance for the Linear Model and Factorization Model;

Validation section provides insight on results of validation for Linear Model;

Conclusion section that gives a summary of the report, the limitations and future research.

Key findings (RMSE):

The paper used various methods of data analysis to find the optimal RMSE value. Upon concluding the analysis, the best RMSE result for Linear Model was achieved at **0.8648177** and **0.786** for the factorization method by *recosystem*. As part of the course requirement, only the Linear model was examined with the validation dataset. Furthermore, the validation dataset was only used once at the final point of the Linear model.

Files attached to this report:

- *RMD script*
- *R script*

Overview of the project

Project Goal

The main goal of this project is to examine various methods used in Data Science to implement simple models for recommendation systems. The project requirements were specified by the EdX team as follows:

- The submission for the MovieLens project will be three files: a report in the form of an Rmd file, a report in the form of PDF document knit from Rmd file, and an R script that generates predicted movie ratings and calculates RMSE. All three files require to be submitted in the requested formats.
- The report should include all required sections (Introduction, Methods, Results, Conclusion), it should be easy to follow with good supporting detail throughout, and be insightful and innovative.
- Code should be easy to follow, consistent with the report, and well-commented
- The final RMSE value should be less than 0.86490
- It is not allowed to use the validation set for training or regularization, and the validation set should only be applied to the final RMSE.

In order to follow the requirements of this project, we will utilize the following resources, libraries, and functions:

Libraries and Environment

The specifications of the R Studio session and libraries used in the report can be found here. In this project, we utilized R Server run on Google Cloud Platform.

The reasons for choice for R Server are manifold. During the initial setup of the project, it has been realized that computing resources of the machine where initially the code was used are not nearly enough to create a sustainable environment for the project, precisely due to numerous downtime events caused by the inability of the computer to handle big data queries. Many options to resolve this issue were discovered and tested:

a) changing the memory size limit for R Studio on the local machine,
b) obtaining an account with shinyApp and RStudio.cloud,
c) installing the R Studio server via WHM on Cent OS,
d) obtaining a separate droplet through DigitalOcean service and
e) obtaining a service from Google Cloud Platform (GCP) to host R Studio Server and this project.

While a) did not improve the performance of the local machine, b) was not able to handle the amount of data processed (without purchasing additional space on the platform), c) option did not succeed due to server limitations which we had access to, d) option had, unfortunately, not authorized the account immediately after registration and paused the activity of the project. Finally, the discovery was made with GCP, which offered a free credit of USD 300 with the sign-up. The R Studio Server was set up on dedicated IP and showed tremendous improvement in the performance of this project code.

The server required additional installations in order to produce this report: all required libraries, TinyTeX. Due to time constrain, it was decided not to install/experiment with additional fonts for the server to optimize the current report visual identity; please accept our apologies for this shortcoming.

EDX and Validation sets

The project requires to preload data from MovieLens and distribute it to the edx and validation sets. The code can be seen from the .R/RMD files submitted with this report or by enrolling in the course on EdX platform. The provided code was changed slightly to adjust to the performance of R Studio version (R Studio Server) as follows:

> movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId)(removed the following: [movieId]),

This was required as otherwise, the factors produced NAs in movieID and genres. A glance look online on the same issues suggested changing server settings, however, after confirmation that the data was not in any way affected (random checks of several users and ratings was made to ensure the data was not compromised), the decision was made to keep the change and proceed with the analysis. Additionally, during the initial stage of the project on the local machine, the data was also backed-up and exported to .csv format, but this action was not used in the final project production, as the issue with big data handling was solved by utilizing the GCP.

the *edx* set was split into two parts: train (90%) and test (10%) sets as a requirement for RMSE to be validated only on *validation* set. We will explore this step in detail later in this report.

RMSE

In this project, we used RMSE (Root Mean Squared Error) function described in the textbook.

> In this section, we describe how the general approach to defining "best" in machine learning is to define a loss function, which can be applied to both categorical and continuous data.The most commonly used loss function is the squared loss function... The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set...

We calculate RMSE as the formula below:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$$

In our code, we will define RMSE as:

```
# The RMSE function in R code:
RMSE <- function(true_ratings = NULL, predicted_ratings = NULL) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Regularization

We will use Regularization concept to "tune" our linear model, as described in the textbook:

> Regularization permits us to penalize large estimates that are formed using small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions... The general idea behind regularization is to constrain the total variability of the effect sizes.

To do that, we will define the following function with the tuning parameters λ to be examined later in the project:

```r
regularization <- function(lambda, trainset, testset){

  # Mean
  mu <- mean(trainset$rating)

  # Movie effect (bi)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # User effect (bu)
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Prediction: mu + bi + bu
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}
```

Recosystem

There are different utilities and libraries available in R packages for testing and training the recommendation systems. In particular, recommenderlab and recosystem proved to be most efficient.

In this project we will utilize recosystem, as it is intuitively more clear and uses simple syntax.

recosystem is an R wrapper of the LIBMF library developed by Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin and Chih-Jen Lin, an open source library for recommender system using marix factorization.

> The main task of recommender system is to predict unknown entries in the rating matrix based on observed values. Each cell with number in it is the rating given by some user on a specific item, while those marked with question marks are unknown ratings that need to be predicted. In some other literatures, this problem may be given other names, e.g. collaborative filtering, matrix completion, matrix recovery, etc. Highlights of LIBMF and recosystem LIBMF itself is a parallelized library, meaning that users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. [@LIBMF] recosystem is a wrapper of LIBMF, hence the features of LIBMF are all included in recosystem. Also, unlike most other R packages for statistical modeling which store the whole dataset and model object in memory, LIBMF (and hence recosystem) is much hard-disk-based, for instance the constructed model which contains information for prediction can be stored in the hard disk, and prediction result can also be directly written into a file rather than kept in memory. That is to say, recosystem will have a comparatively small memory usage.

We will use this library as our second method for the project.

Steps of this project

The project will be performed in the following steps:

1. We will analyze the provided data
2. We will preprocess the provided data and split edx into two parts
3. We will build two methods for our task: Linear Model (via RMSE and regularization) & Factorization Model (via recosystem)
4. We will test Linear Model against Final Validation (as a requirement of the project, the validation set could only be used once; since the aim of the project is to practice Linear Model - we will only utilize it with the LM itself.)
5. We will record and analyze the results
6. We will conclude the discussion on both methods used with observed limitations and recommendations for future research.

Exploratory Data Analysis

Initial Datasets

We will start our analysis with the overview of available datasets generated for this report: *edx* and *validation*. The datasets have the following characteristics:

Feature	EDX	Validation
Number of Rows	9000055	999999
Number of Columns	6	6
Unique Users	69878	68534
Unique Movies	10677	9809
Variety of Genres	797	773

The dataset *validation* was generated as a sample of *edx* dataset and comprised of almost a million observations, whereas *edx* has almost 10M records. The validation dataset will be used in our analysis as a final measure to identify the RSME on the latest model. The datasets have the following columns:

Name	Comment
userId	Unique identification number for each user in the dataset
movieId	Unique identification number for each movie in the dataset
rating	A range of marks(rating) given to a movie by specific user
timestamp	A record of specific time when the rating was given by the user to a particular movie
title	Title of the movie with Release date
genres	Genre(s) of the movie

The data in both datasets appears to be not tidy enough to start our analysis. The below first seven records from the *edx* dataset demonstrate that the data structure needs to be changed:

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Specifically, for the following observations:

- timestamp - the format currently displays the number of second since Jan 1, 1970 (EPOCH) and is hard to understand
- title - Movies' titles have the year of their release, this data might be helpful in our analysis
- genres - column consists of a variety of genres divided with |-sign; it might be useful to segregate the genres for our analysis

Hence, prior to starting any analysis of the data, the dataset must be put in order.

Methods

Preprocessing of the data

We will perform preprocessing of our data in accordance with recommendations set in the textbook and will try to keep the data as tidy as possible.

> In machine learning, we often transform predictors before running the machine algorithm. We also remove predictors that are clearly not useful. We call these steps preprocessing. Examples of preprocessing include standardizing the predictors, taking the log transform of some predictors, removing predictors that are highly correlated with others, and removing predictors with very few non-unique values or close to zero variation.

In addition, we will analyse if any variables need to be changed for our analysis.

When we attempted to start with the *colSds* and *nearZero* function from the caret package to see if any features do not vary much from observation to observation, as recommended for data preprocessing, we realized that all columns in the dataset are required for our analysis, hence, there is no need to perform this function. Some rows can be eliminated (for example, 2009 data is very limited), but for the sake of order, we will proceed.

In order to be consistent in the course of our analysis, ideally, we require to make adjustments to both datasets, *edx* and *validation*. However, since the requirement of the project not to touch the validation set until the last model - we will **only** apply changes to *edx*

We will undergo several steps to optimize the dataset. We will start by converting the *timestamp* values into Year-Month-Day format from the current EPOCH. We will then display the values of year, month and day separately in order to investigate and visualize the data.

Since we noticed, that movie titles contain their respective release dates, we will extract that information too. It will help us to identify the trend for user ratings.

As currently the dataset contains a combination of genres for each movie and we assume, that this information is essential to us, there are several ways to reassemble the data: a) split the genres for distinctive values as additional rows in the dataset or b) add additional columns for each respective genre and ensure that each movie has a TRUE/FALSE value for the respective genre.

The first approach will significantly increase the number of rows in our dataset, as each distinctive movie typically has few genres. This will lead to the crash of the R Studio due to computing limitations of the used hardware. It is also not wise to misuse the resources unless necessary. The second approach might limit our ability to calculate the RMSE correctly, as additional variables-genres will be introduced. The data is not very continuous; hence there is a high probability it won't provide us any additional leverage.

An alternative solution to the above is to consider genre variability as an extension of the movie specification and treat each particular combination of genres as a unique category.

As reported by several peers in the EDX forum, the genre specificity might have a low impact on the total RMSE, and hence at this stage of the project, in order to not overwhelm the environment, we will consider the alternative solution. Once the calculations for RMSE models are made, we might reconsider this step, if the RMSE values will not meet the required low values.

Regardless of this, we will still attempt to examine the data. In the meanwhile, we will keep only the columns which we require for our analysis.

After applying the above changes, the datasets have the following structure:

userId	movieId	rating	title	genres	release	yearR	monthR	dayR
1	122	5	Boomerang	Comedy|Romance	1992	1996	8	2
1	185	5	Net, The	Action|Crime|Thriller	1995	1996	8	2
1	292	5	Outbreak	Action|Drama|Sci-Fi|Thriller	1995	1996	8	2
1	316	5	Stargate	Action|Adventure|Sci-Fi	1994	1996	8	2
1	329	5	Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	1994	1996	8	2
1	355	5	Flintstones, The	Children|Comedy|Fantasy	1994	1996	8	2

Notably, it is also important to use the provided *validation* set only on the final model, therefore we will create additional datasets from *edx* to train and test the models before the final *validation* is used. We will apply the same settings as the initial split.

As part of trial/error, we examined that if % of allocated data for training and test sets varies, the final RMSE also varies. We examined 20% and 25% split, both of which gave higher RMSE. This can be explained due to the sampling method or odd behavior of the first machine utilized in this project.

Now we are in possessions of 3 datasets as follows:

Feature	Train	Test	Validation
Number of Rows	8100065	899990	999999
Number of Columns	9	9	6
Unique Users	69878	68159	68534
Unique Movies	10677	9701	9809
Variety of Genres	797	764	773

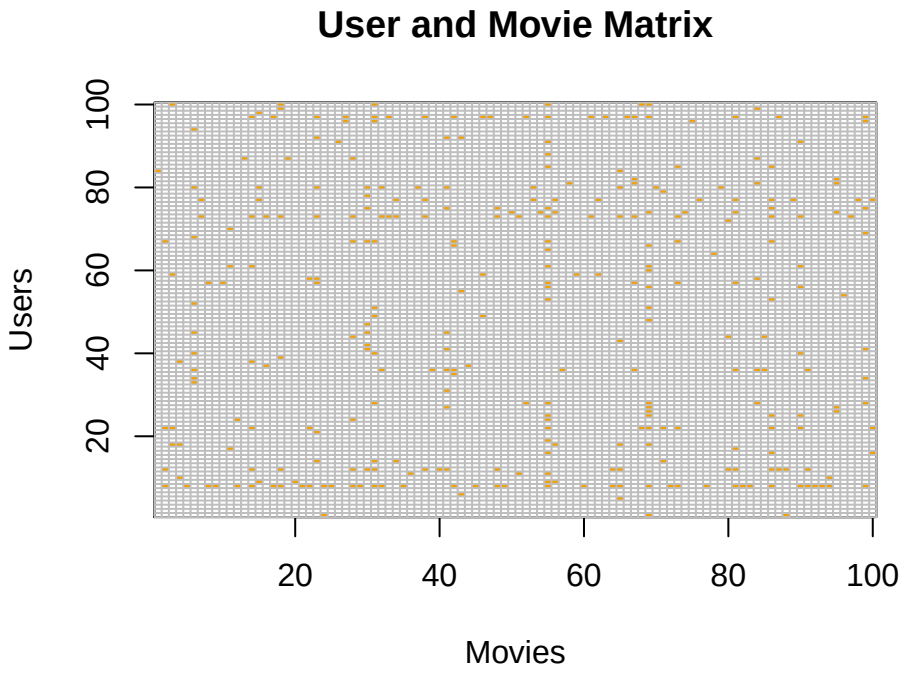
We can now explore data in our train dataset.

Dataset analysis

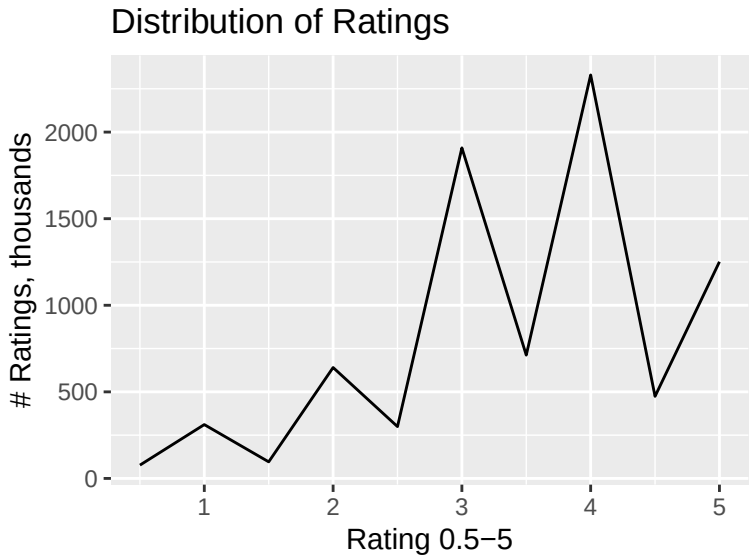
The train dataset comprises around 7.2M rows and 9 columns. Strategically, we need to understand how the data is distributed in order to utilize RMSE. The reason why we are examinig trainSet instead of EDX is because trainSet is a randomply generated sample of 90% of the EDX data - hence it is reliable to represent the patterns of data distribution.

Ratings Distribution

We will start our analysis with demonstrating a heatmap for a sample of hundred Users and Movies:

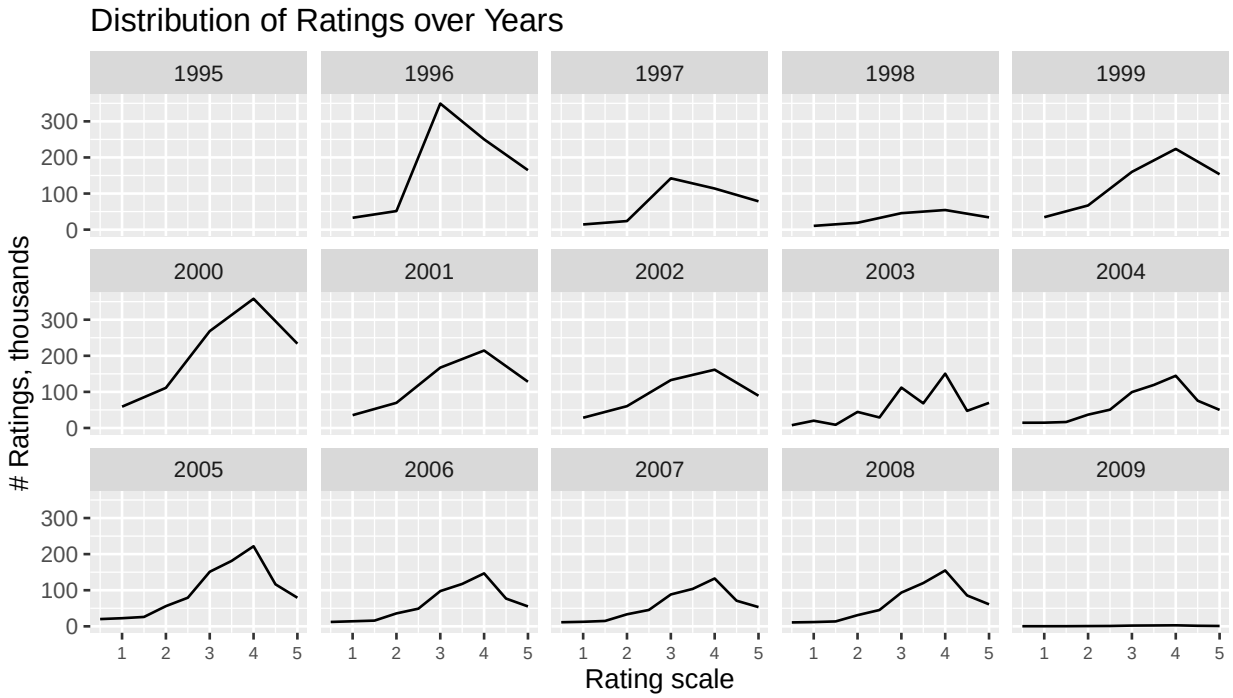


This gives us an idea how data is spreaded across the dataset.

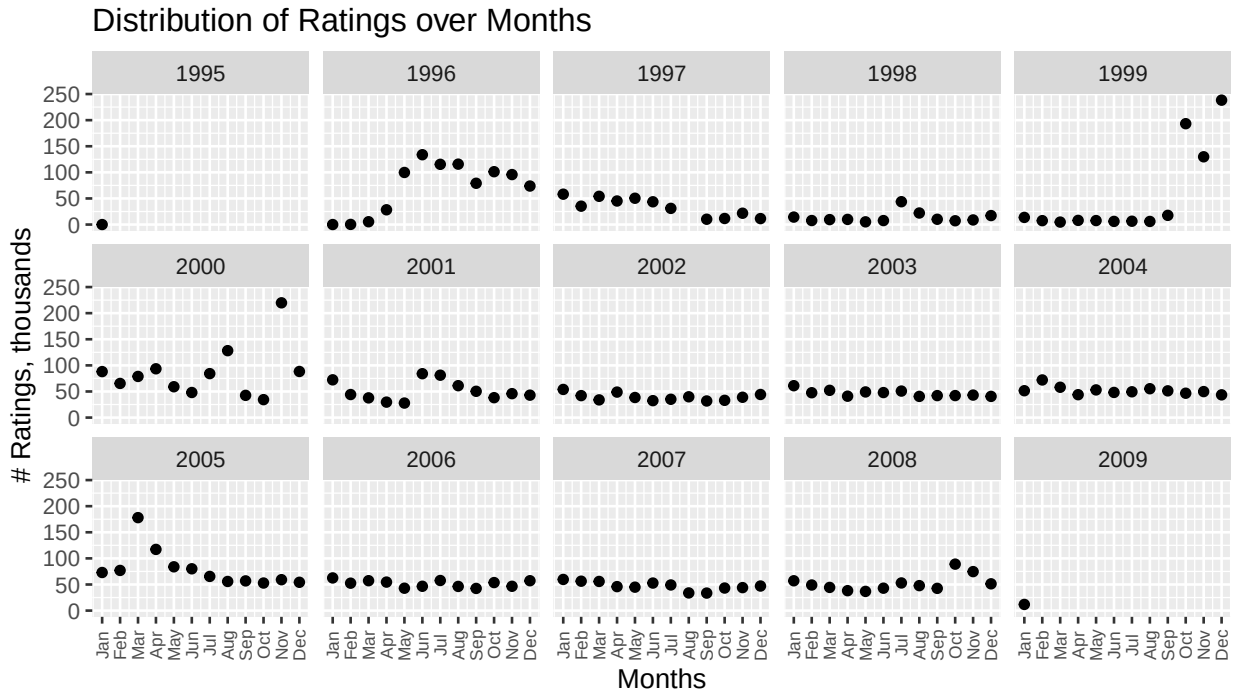


The distribution of ratings varies from user to user, movie to movie, year to year.

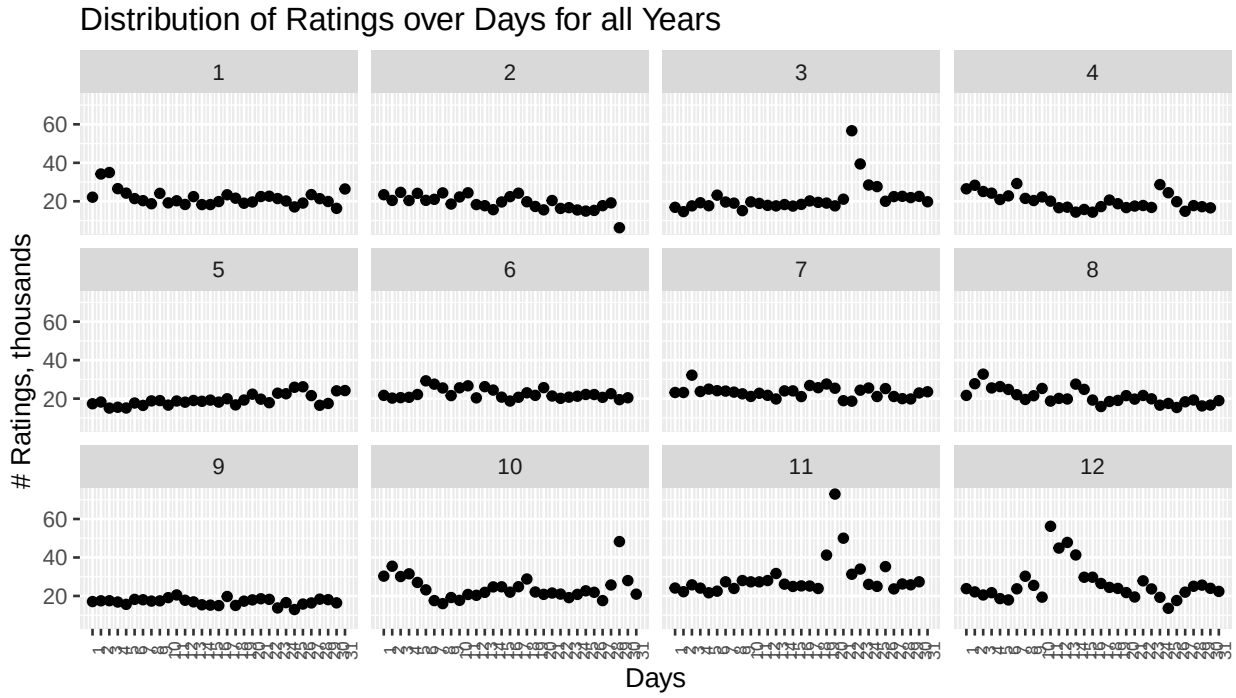
In general, half star ratings are less common than whole star ratings. In the *trainSet* we can count over 79% of whole star ratings across all users. At the same time, the avarage ratings across the dataset is 3.5.



Over the years, the trends of the rating distribution also change. If we are not taking into the account year 1995 and 2009 (as there is not enough data to compare it with other years), we can notice that in 1996, 2000 the users were very active (the majority of ratings corresponded to 3 and 4 respectively), but in 1998, the users were not active at all (hence no sharp edges over the ratings). 2001, 2004-2008 demonstrate to us how users were active, almost identically. In a real-life scenario, it might be very uncommon to see such a big range of diversity and commonality in the same dataset. This could be explained by the origin of data (the data was compromised), sampling method (probability of mistake during the sampling of edx dataset), or actual triggers/influencers within those years on users (new platforms, development of the internet, the difference between new users and old users, the origin of users - as we do not have the data on demographics).

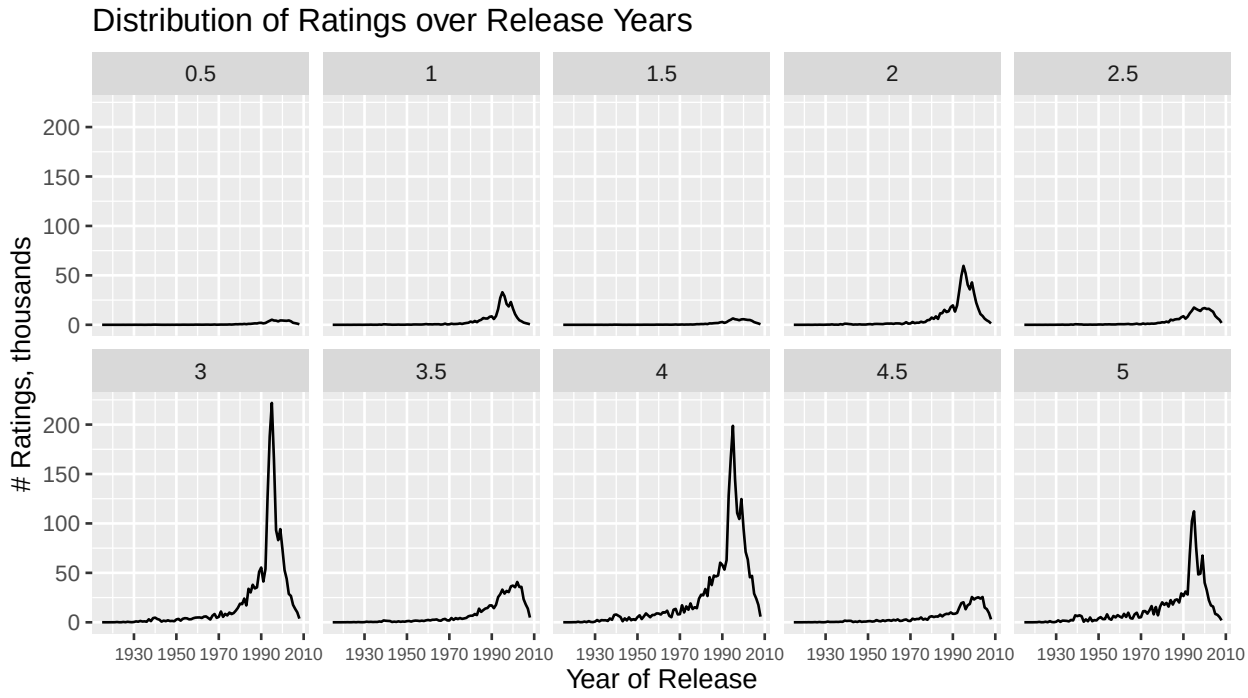


The frequency of users providing a rating for movies also varies month-to-month over the years. Notably, in 1999 we see the highest peak, whereas otherwise, from 2001 to 2008, the data changes only slightly month-to-month. Hypothetically, this could be caused by specific blockbuster movies to be released in those years (1996,1999,2000,2005), which show unregular behavior for the users.

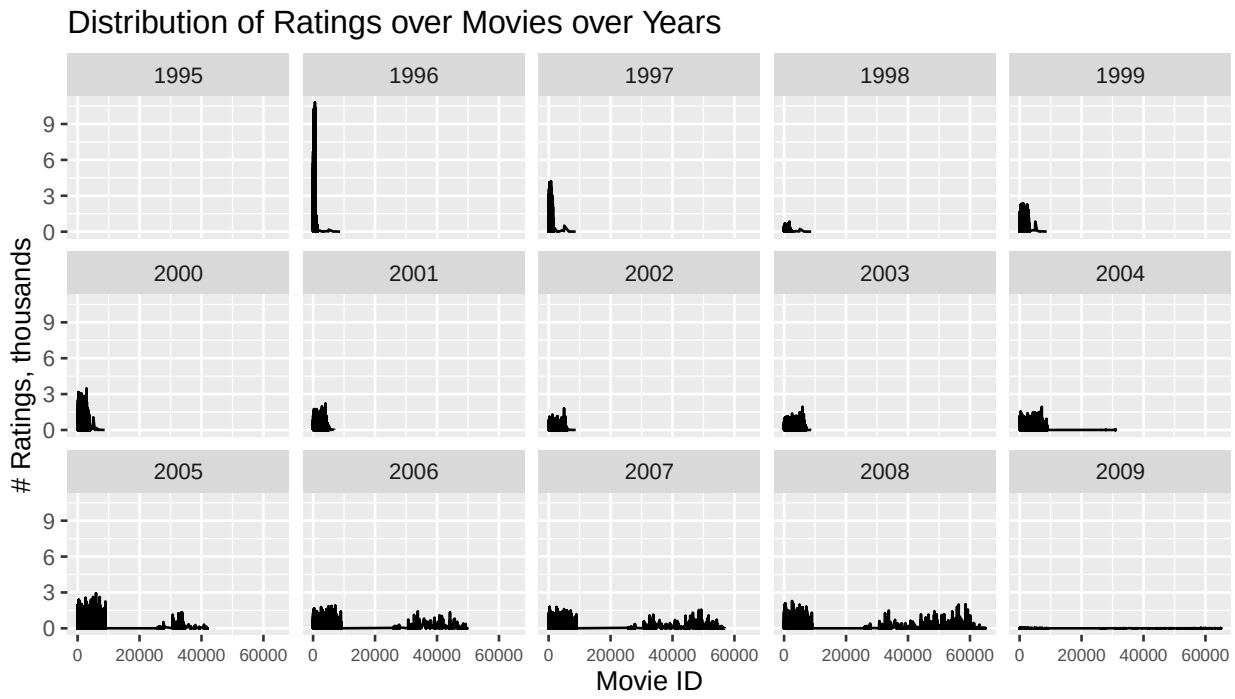


Over the years, we can identify several months and days where the number of ratings is significantly different:

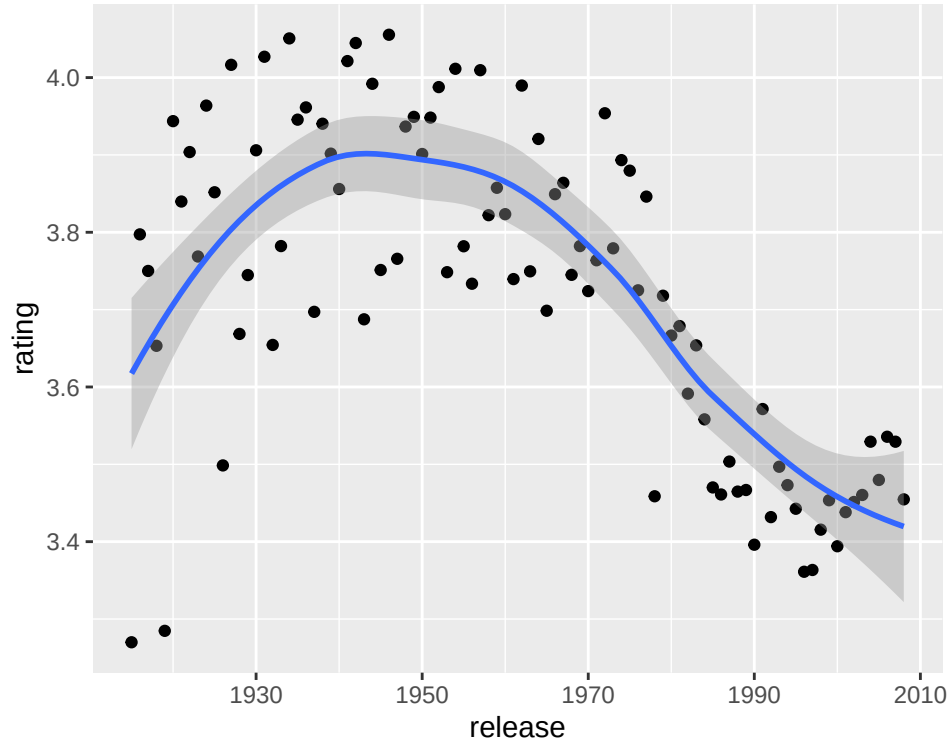
the second part of March, the second part of November. It appears that most profitable movies were released around these dates.



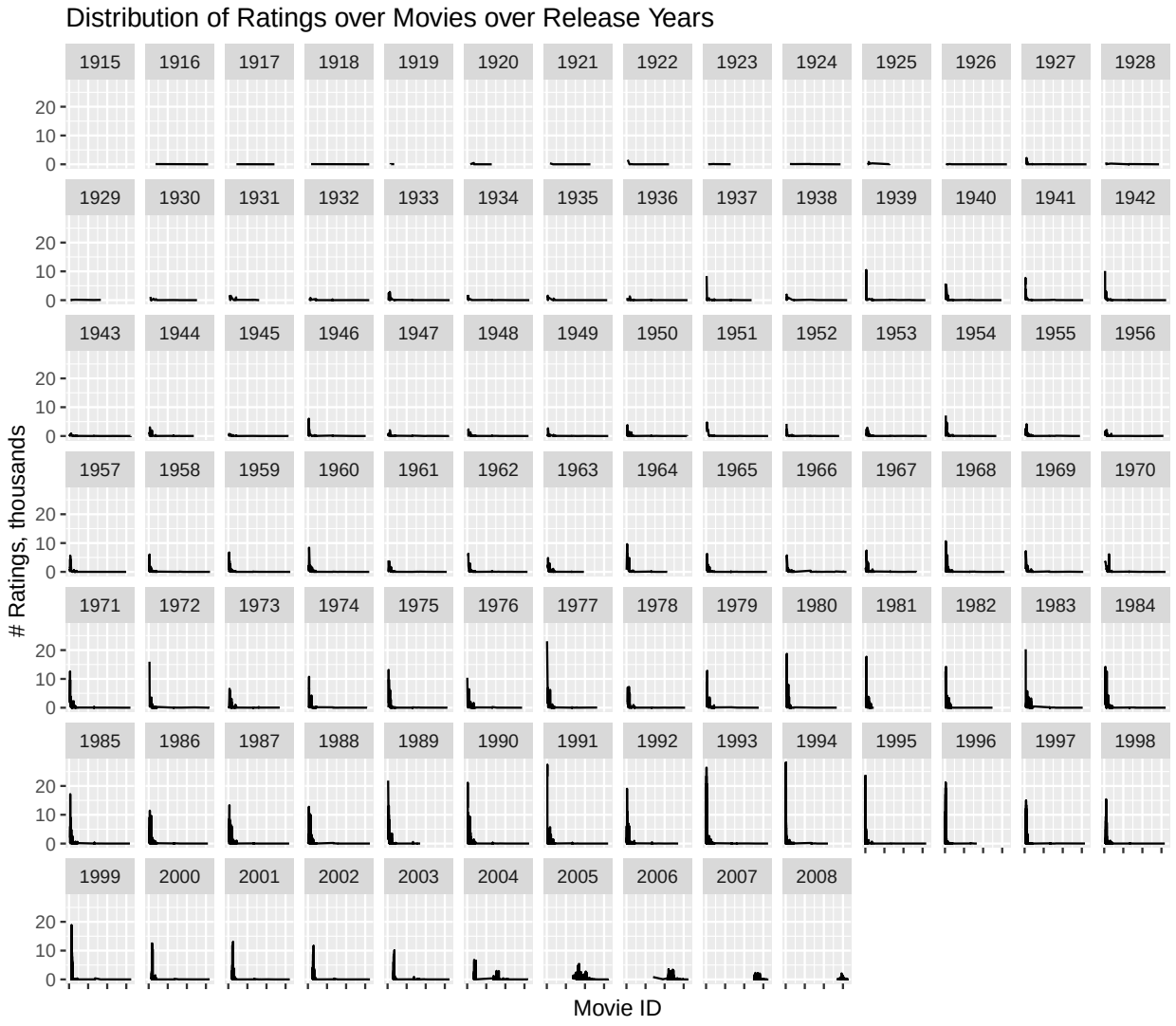
As anticipated, the year of movie release also plays a role in the distribution of ratings. For instance, ratings of 3,4,5 were awarded more for movies released after 1990. We can see that in general, movies after 1990 receive more ratings.



Over the years, some movies received particularly high or low ratings. It might be important to compare this with the year of the movie release.

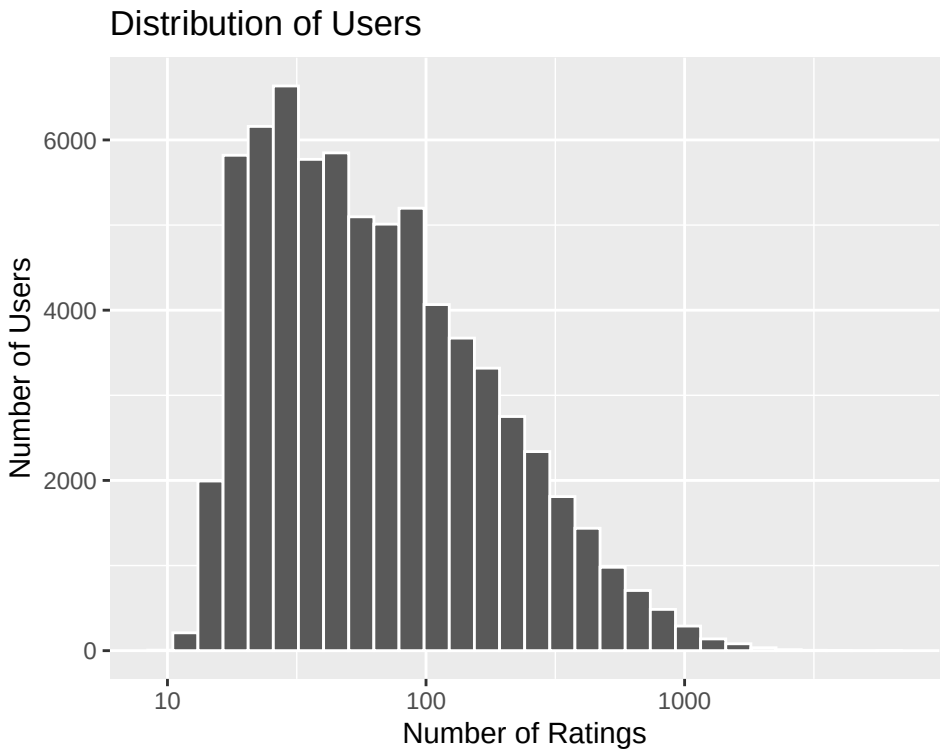


Users also tend to rate less movies which were released after 70s, A more detailed view:



Movie release year changes the behaviour of users to provide ratings. Older (1915) movies receive fewer ratings, followed by "classics" and movies from the 90s. The newest movies (00s) receive fewer ratings again.

To understand the data origin, we need to examine how users rated movies in the datasets.

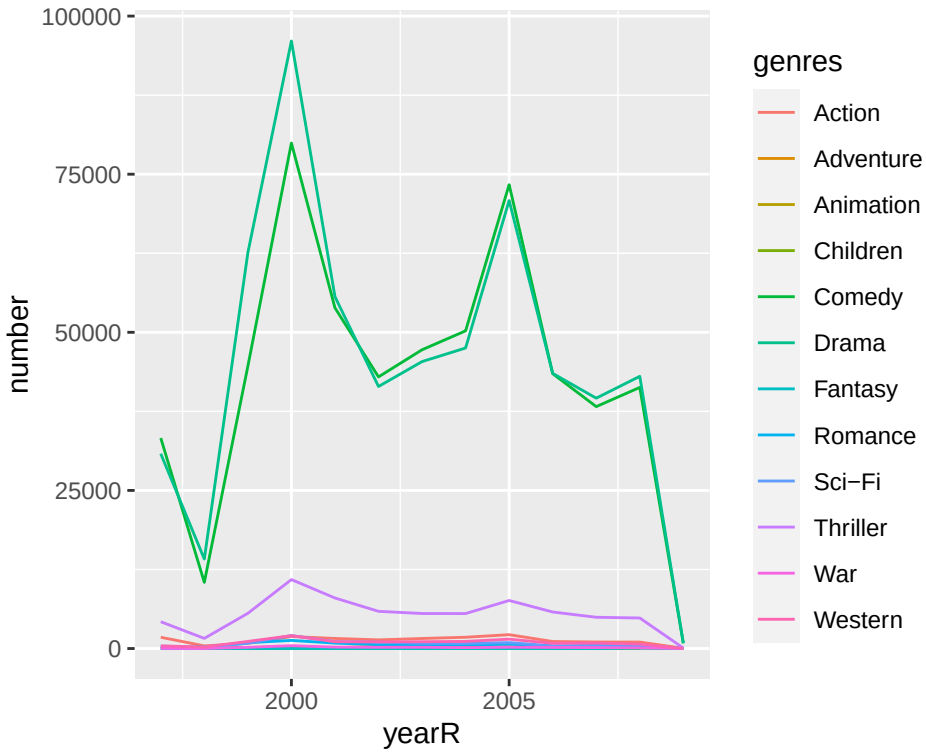


The distribution of users and their respective ratings appear to be slightly inconsistent - as anticipated by the user behavior. It is not possible to expect all users to rate the same amount of movies — few users rate few or over 1K titles. The majority lies within a hundred ratings per user.

Genres Variety: in the current datasets there are almost 800 different variety of genres as they are recorded as tags unqiue for each movie:

genres	n
(no genres listed)	7
Action	22096
Action|Adventure	61787
Action|Adventure|Animation|Children|Comedy	6705
Action|Adventure|Animation|Children|Comedy|Fantasy	165
Action|Adventure|Animation|Children|Comedy|IMAX	55

Genres, in fact, can be grouped - as we can see from the belowplot, there are similarities between the genres variety. This potentially can be used for our model, but since it is very resource-consuming, we will leave it for future research.



Now we understand more about our dataset and can use various methods to compute the RMSE.

Results

Linear Model

Mean distribution model

The first naive model approach can be used to determine the accuracy of our function to predict the star rating. We will use the train and test datasets to train this model, as this is not the final model.

A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

Using avarage of ratings for the training set we will deterimne baseline for RMSE.

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

Using this model, we will predict the rating against our training set.

Method	RMSE	Tested
Mean	1.060054	testSet

We notice that the value is too high from desireable value. We shall explore other models for rating prediction.

Movie-centered model (Adding Movie Effects)

The previous model demonstrated that just an average would not satisfy our requirements. However, we know that each movie has a diverse variety of rating scores. Let's add this component to our model. This model can be described mathematically as:

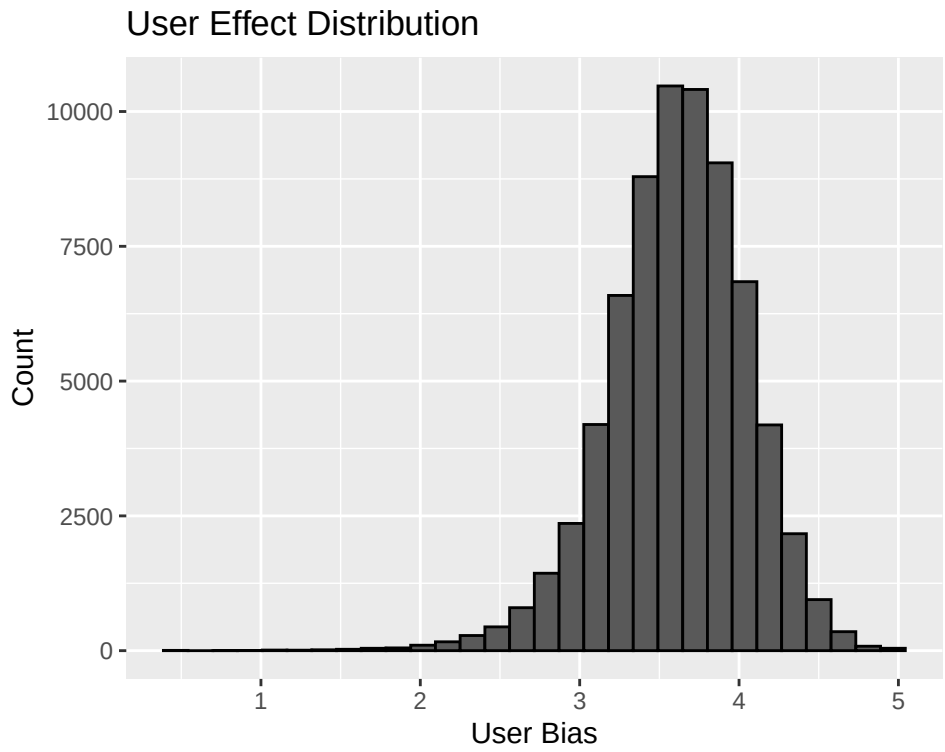
$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

The result of RSME using avarage and avarage per movie will improve our result to:

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet

Movie + User model (Adding User Effects)

Let's compute the average rating for user to examine possible effect on distribution.



As per textbook recommendation, we notice that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model may be:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

where b_u is a user-specific effect. Now if a cranky user (negative b_u) rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We can now construct predictors and see how much the RMSE improves.

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet
Mean + bi + bu	0.8646843	testSet

Movie + User + Genre model

As described in our Methods section, it is assumed that adding genre to the model will not make a significant impact on the RMSE performance, due to the fact that genres are now not separated into their groups, are repetitive by nature and consume many computing resources to be correctly calculated. We will not include this model unless absolutely necessary (if RMSE will not reach the desired value).

Regularization

As described in the textbook, regularization must be performed in order to improve the RMSE results.

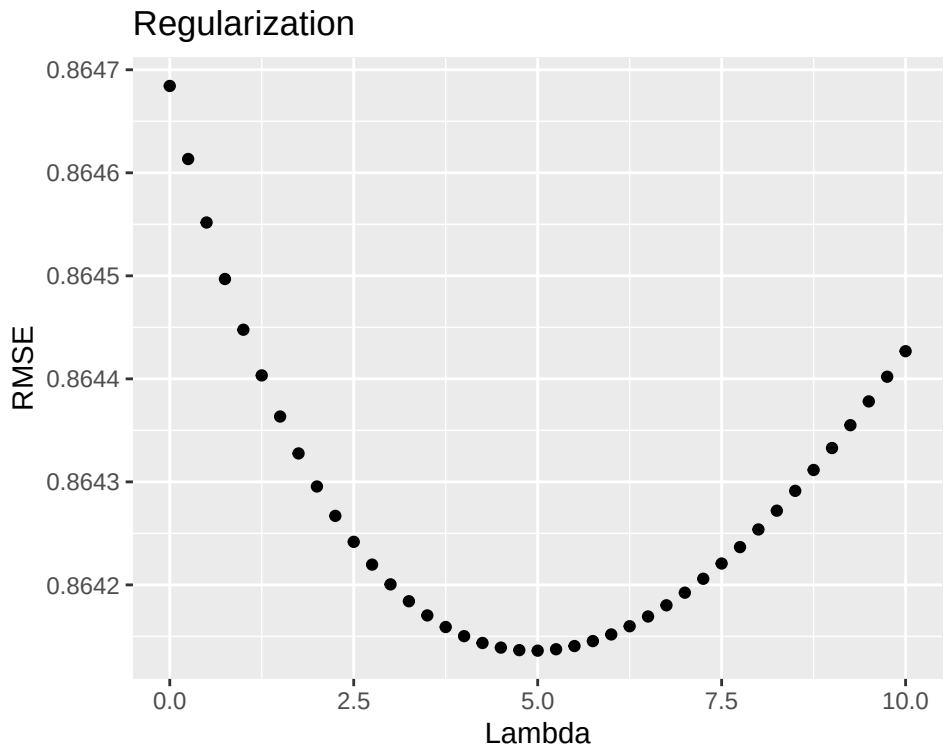
> Regularization permits us to penalize large estimates that are formed using small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions. The general idea behind regularization is to constrain the total variability of the effect sizes.

Mathematically, it can be presented as follows:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

The penalized estimates provide a large improvement over the least squares estimates. In order to find such the best λ, we will need to apply a sequence from 0 to 10 and select the best performing one (the one with the lowest RMSE value).

We can construct a plot to find out.



The best value is:

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

We will apply this parameter to our model:

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet
Mean + bi + bu	0.8646843	testSet
Regularized bi and bu	0.8641362	testSet

We can now see significant improvement and potentially meeting the requirement for the project. We should still use this model on the final validation set to approve the result.

Factorization Model

Recosystem

From the available literature review, we can gather that a comprehensive analysis was made to evaluate various methods for training the algorithms for recommendations. Specifically, we are referring to practical comparisons made by Taras Hnot, where he argues that:

> There are two main categories of collaborative filtering algorithms: memory-based and model-based methods.
>
> Memory-based methods simply memorize all ratings and make recommendations based on relation between user-item and rest of the matrix. In model-based methods predicting parametrized model firstly is needed to be fit based on rating matrix and then recommendations are issued based on fitted model.
>
> Model-based methods, on the other hand, build parametrized models and recommend items with the highest rank, returned by model.

In his research, Taras has pointed out that "the best performance was shown by Matrix Factorization techniques with Stochastic Gradient Descend".

After examining the RStudio packages as an alternative to our Linear Model: recommenderlab, SlopeOne, SVD Approximation and recosystem, the most intuitive and effortless solution was named as recosystem - a specified library which required minimum efforts to be executed.

We followed the steps described in the package documentation, and it took approximately an hour for the script to be executed. However, the results were worth waiting:

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
## iter      tr_rmse          obj
##    0       0.9820   1.1014e+07
##    1       0.8751   8.9742e+06
##    2       0.8416   8.3340e+06
##    3       0.8192   7.9408e+06
##    4       0.8031   7.6749e+06
##    5       0.7909   7.4874e+06
##    6       0.7807   7.3439e+06
##    7       0.7723   7.2299e+06
##    8       0.7650   7.1346e+06
##    9       0.7589   7.0590e+06
##   10       0.7535   6.9945e+06
##   11       0.7486   6.9375e+06
##   12       0.7443   6.8904e+06
##   13       0.7403   6.8474e+06
##   14       0.7366   6.8092e+06
##   15       0.7332   6.7750e+06
##   16       0.7303   6.7454e+06
##   17       0.7273   6.7187e+06
##   18       0.7247   6.6929e+06
##   19       0.7223   6.6695e+06
```

Results

After running the recosystem package on our trainSet, we further improved our RMSE values down to **0.786**:

We updated the table with our findings.

```
##   [1] 4.785615 3.641680 3.210669 2.973059 3.467070 3.921262 3.522733 3.169163
##   [9] 3.955036 3.142277
```

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet
Mean + bi + bu	0.8646843	testSet
Regularized bi and bu	0.8641362	testSet
Recosystem	0.7859289	testSet

As one of the requirements for the project, the validation set can only be run once. It means we have a choice which model to present as the Final and apply it to the validation set.

As Linear Model was one of the focus-points in the course, we decided to run the final validation on it, even when the results for the Factorization model have shown much improvement.

This is to exclude any possibility of potentially wrong application of this package, as our experience with Data Science thus far suggests that there are always some pitfalls awaiting for us.

Validation and Final Results

Validation

As was discussed in the textbook, validation is a very important step in our project. As per requirement fo the project, we will use validation set only once - for our Linear Model.

> Cross validation is based on the idea of imitating the theoretical setup above as best we can with the data we have. To do this, we have to generate a series of different random samples. There are several approaches we can use, but the general idea for all of them is to randomly generate smaller datasets that are not used for training, and instead used to estimate the true error.

Validation was performed on the *validation* set using the LM prediction after regularising the data:

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet
Mean + bi + bu	0.8646843	testSet
Regularized bi and bu	0.8641362	testSet
Recosystem	0.7859289	testSet
Regularized EDX's bi and bu on Validation Set	0.8648177	validation

Final Results Table

The below table summarizes the final results for the Models:

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet
Mean + bi + bu	0.8646843	testSet
Regularized bi and bu	0.8641362	testSet
Recosystem	0.7859289	testSet
Regularized EDX's bi and bu on Validation Set	0.8648177	validation

Conclusion

This sections summarizes our findings for the project.

Results

The final results for Linear Model after Validation and Factorization Model are shown below:

Method	RMSE	Tested
Mean	1.0600537	testSet
Mean + bi	0.9429615	testSet
Mean + bi + bu	0.8646843	testSet
Regularized bi and bu	0.8641362	testSet
Recosystem	0.7859289	testSet
Regularized EDX's bi and bu on Validation Set	0.8648177	validation

We met the requirements of this project as follows:

- We prepared three files for the project: PDF report, R script, and RMD script;
- In the report, we overviewed the scope of work, two models/approaches (Linear and Factorization), run the analysis on MovieLens split dataset;
- We well-commented each code section, so it will be easier to examine it ;
- We achieved the final RMSE value to be less than 0.86490: for Linear Model - **0.8648177**; for Factorization - **0.786**;
- We only used the validation set on the final Linear Model as instructed.

Limitations

This project had significant limitations summarized as follows: limited computing ability of the available machine, the quality of provided dataset (distribution of ratings for several years appeared to be too common, which might not be the case in the real datasets), recosystem settings were used "as is" and with the right tunning, the better results are expected.

Further Research

It will be beneficial to run the models on the following packages: recommenderlab, SlopeOne, SVD Approximation with various available settings, and examine how different the results are compared to Linear Model and recosystem. In addition, accuracy needs to be examined for all the above models to ensure the correct picture has been drawn.

Session Info

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04 LTS
##
## Matrix products: default
## BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=C
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] tinytex_0.22      recosystem_0.4.2  knitr_1.28        kableExtra_1.1.0
##  [5] data.table_1.12.8 caret_6.0-86      lattice_0.20-41   forcats_0.5.0
##  [9] stringr_1.4.0     dplyr_0.8.5       purrr_0.3.4       readr_1.3.1
## [13] tidyr_1.0.3       tibble_3.0.1      ggplot2_3.3.0     tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.1           viridisLite_0.3.0    jsonlite_1.6.1
##  [4] splines_4.0.0        foreach_1.5.0        prodlim_2019.11.13
##  [7] modelr_0.1.7         assertthat_0.2.1     stats4_4.0.0
## [10] cellranger_1.1.0     yaml_2.2.1           ipred_0.9-9
## [13] pillar_1.4.4         backports_1.1.7      glue_1.4.1
## [16] pROC_1.16.2          digest_0.6.25        rvest_0.3.5
## [19] colorspace_1.4-1     recipes_0.1.12       htmltools_0.4.0
## [22] Matrix_1.2-18        plyr_1.8.6           timeDate_3043.102
## [25] pkgconfig_2.0.3      broom_0.5.6          haven_2.2.0
## [28] webshot_0.5.2        scales_1.1.1         gower_0.2.1
## [31] lava_1.6.7           mgcv_1.8-31          farver_2.0.3
## [34] generics_0.0.2       ellipsis_0.3.0       withr_2.2.0
## [37] nnet_7.3-13          cli_2.0.2            survival_3.1-12
## [40] magrittr_1.5         crayon_1.3.4         readxl_1.3.1
## [43] evaluate_0.14        fs_1.4.1             fansi_0.4.1
## [46] nlme_3.1-147         MASS_7.3-51.5        xml2_1.3.2
## [49] class_7.3-16         tools_4.0.0          hms_0.5.3
## [52] lifecycle_0.2.0      munsell_0.5.0        reprex_0.3.0
## [55] compiler_4.0.0       rlang_0.4.6          grid_4.0.0
## [58] iterators_1.0.12     rstudioapi_0.11      labeling_0.3
## [61] rmarkdown_2.1        gtable_0.3.0         ModelMetrics_1.2.2.2
## [64] codetools_0.2-16     DBI_1.1.0            reshape2_1.4.4
```

```
## [67] R6_2.4.1             lubridate_1.7.8      stringi_1.4.6
## [70] Rcpp_1.0.4.6         vctrs_0.3.0          rpart_4.1-15
## [73] dbplyr_1.4.3         tidyselect_1.1.0     xfun_0.13
```