Movielens Recommendation Project

HarvardX PH125.9x Capstone

Octavio Rodríguez Ferreira

05/04/2021

Executive summary

This project explores different approaches to reduce the error in a movie recommendation system. Starting with a benchmark of the mere `mean`, we develop our modeling process testing different algorithms starting with a simple prediction based on the average ratings. We then explore a baseline model that can measure the "effect" of certain variables on the rating. We add a regularized model that adds a penalty to our "effects" and chooses the best from a cross-validation of a range of values of lambda. Finally, we introduce a matrix factorization method known as `recosystem` which yielded the best results and was the most efficient in processing time. Overall, both the regularized and the `recosystem` algorithms managed to bring the `root mean square error` below the initial goal of `0.8649`, however `recosystem` was the most efficient in terms of processing time and reducing the RMSE.

1. Introduction

This is one of capstone projects that concludes the HarvardX PH125.9x ninth and final course in HarvardX's Data Science Professional Certificate series. For this project it is necessary to create a movie recommendation system based on the example of the Netflix data challenge.

For this purpose we will use the MovieLens 10M https://grouplens.org/datasets/movielens/10m/, a data set compiled by GroupLens, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities. This particular data set contains 10 million ratings applied to 10,000 movies by 72,000 users.

The final goal is to build a recommendation system through machine learning algorithms with a lower margin of error. In this case the success of our algorithm will be measured by the `root mean square error` or `RMSE`, and the goal is to get a number of `0.8649` or lower.

The initial approaches taken in this analysis follow Irizarry (2019), starting with a baseline analysis of the effects of different predictors (movie, user, genre, etc.) in the movie rating. The next step, also following Irizarry, introduces regularization, to make our prediction more precise. Finally, the last step follows Chin, et al. (2016), and introduces a matrix factorization model called `recosystem`.

The final analysis shows that the baseline model used initially gives a result very close to our `0.8649` error threshold, but still a little over. The regularized model improved our algorithm significantly reaching our threshold, but the processing time was significantly high. Finally, the recosystem model improved the algorithm exponentially bringing our RMSE way below our initial threshold.

2. Data exploration

2.1. Data

The MovieLens data set consists of `10,000,054` observations and `6` variables.

	Class
userId	integer
movieId	numeric
rating	numeric
timestamp	integer
title	character
genres	character

However, to perform our machine learning analysis, it is necessary to divide our data set in two different sets, one for training our algorithm and one for testing. Our training set will consist of 90% of the original set, and the test set will be the remaining 10%.

Once we have created our train and test sets, the `edx` train set has `9,000,058` observations (rows) and `6` variables (columns), and the `validation` test set has `999,996` observations (rows) and `6` variables (columns). Variables in both sets have the same class and structure, also the same as in the original full set.

2.2. Variable exploration

After an initial exploration of we find that in our `edx` train set, there are `10,677` unique movies and `69,878` unique users.

Unique users	Unique movies
69878	10677

Also, we noticed that the most rated movies are not necessarily the best rated. In fact among the best rated are generally unknown or obscure films. This is the case since these films have very few ratings, sometimes only one.

title	rating	n_ratings
Blue Light, The (Das Blaue Licht) (1932)	5.000000	1
Fighting Elegy (Kenka erejii) (1966)	5.000000	1
Satan's Tango (Sátántangó) (1994)	5.000000	2
Shadows of Forgotten Ancestors (1964)	5.000000	1
Sun Alley (Sonnenallee) (1999)	5.000000	1
Constantine's Sword (2007)	4.750000	2
Human Condition II, The (Ningen no joken II) (1959)	4.750000	4
Human Condition III, The (Ningen no joken III) (1961)	4.750000	4
More (1998)	4.750000	8
Class, The (Entre les Murs) (2008)	4.666667	3
I'm Starting From Three (Ricomincio da Tre) (1981)	4.666667	3
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	4.666667	3

On the other hand, the most rated are very popular films that have a higher number of reviewers, bringing their average rating below those films very few reviewers.

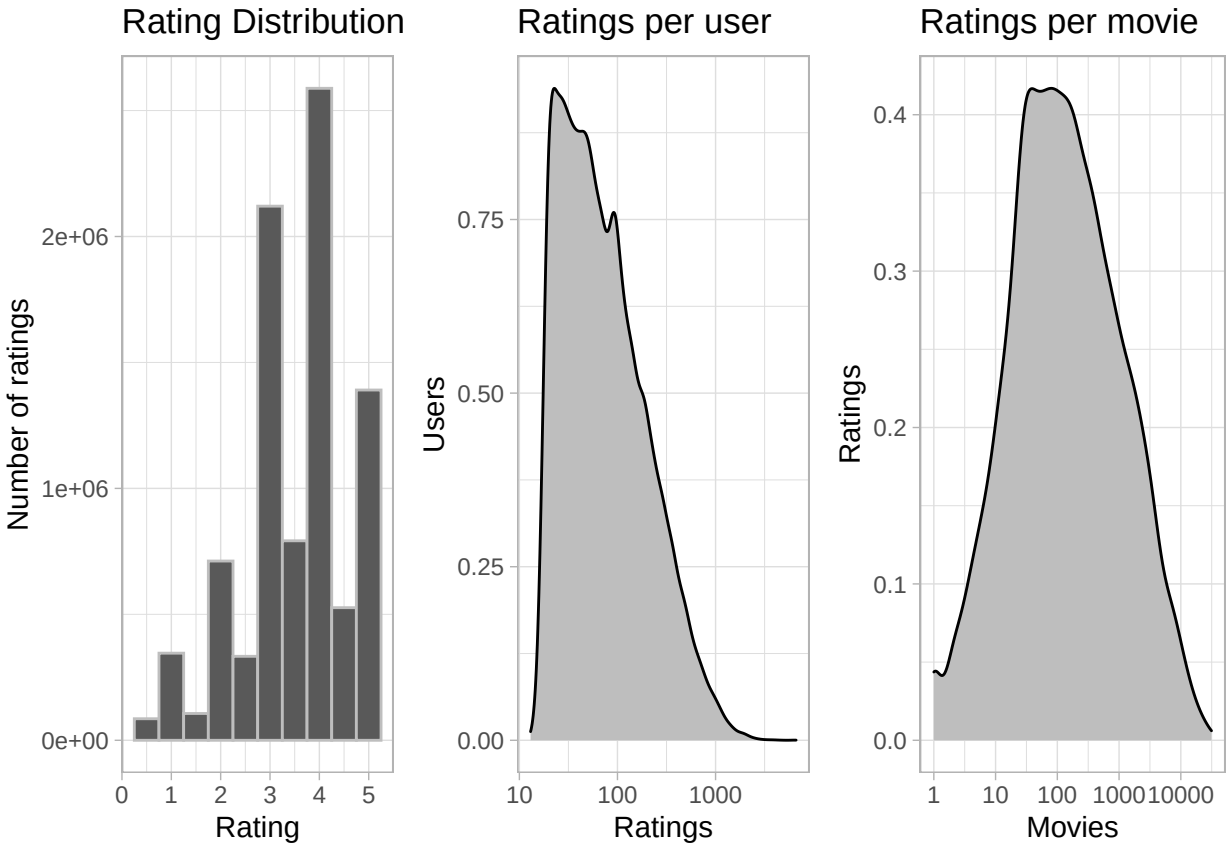
title	n_ratings	rating
Pulp Fiction (1994)	31394	4.156782
Forrest Gump (1994)	30934	4.015808
Silence of the Lambs, The (1991)	30394	4.205550
Jurassic Park (1993)	29359	3.662625
Shawshank Redemption, The (1994)	28059	4.458070
Braveheart (1995)	26292	4.081336
Terminator 2: Judgment Day (1991)	26050	3.928522
Fugitive, The (1993)	25930	4.008561
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25725	4.221380
Apollo 13 (1995)	24315	3.886243

The `ratings` variable is the most important piece of data for our purpose in this data set. The rating system consists of a numeric scale from 0 to 5. In the train set there are 10 different ratings given by users to different movies. The ratings range from a minimum of 0.5 to a maximum of 5. The most common rating is "4" (`2,588,158` ratings of "4" given by users, and the least common is "0.5" (`85,549`).

Rating	Total_ratings
4.0	2588158
3.0	2120397
5.0	1390430
3.5	791895
2.0	711627
4.5	526421
1.0	345810
2.5	333394
1.5	106377
0.5	85549

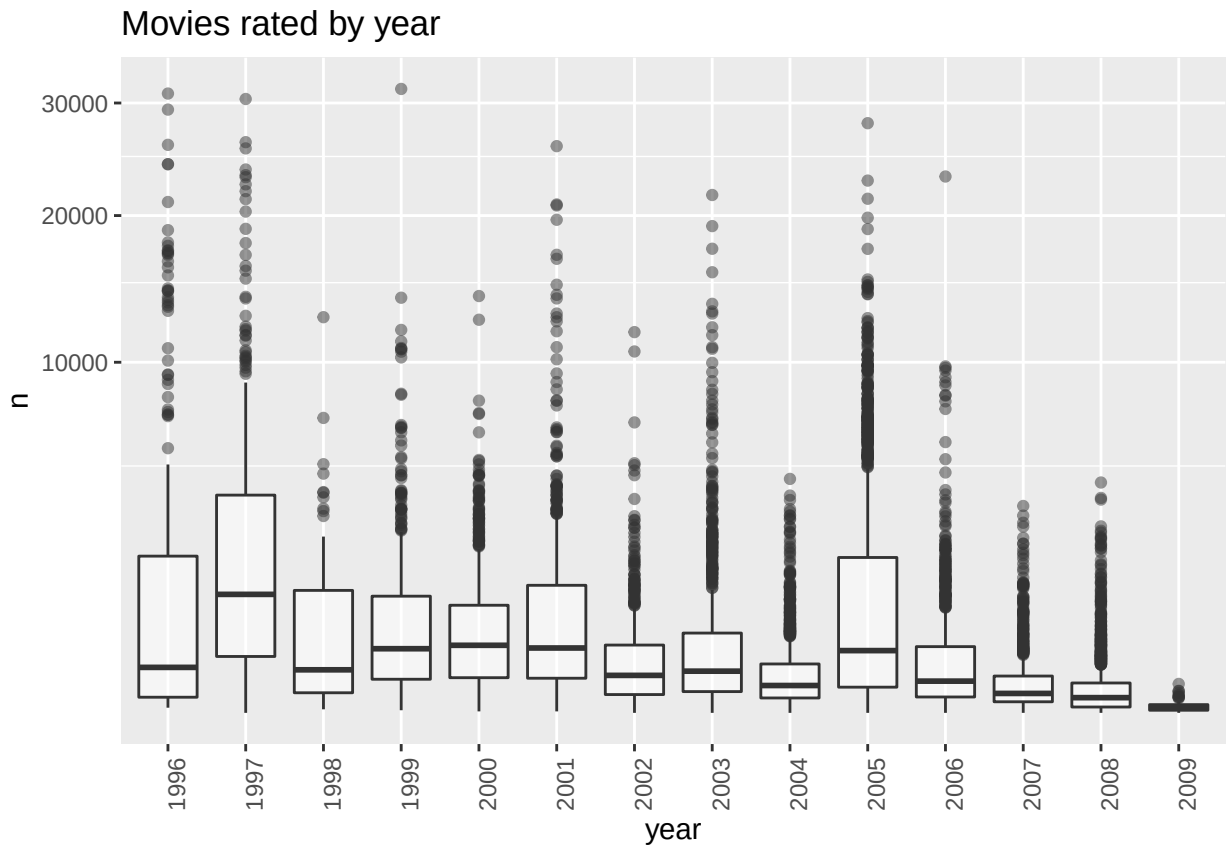
The vast majority of users have rated between 10 and 100 movies). The ratings given to each movie were varied and normally distributed, with the vast majority of movies receiving between 50 and 500 ratings.

Overall, the average rating of all movies was `3.5`. Also, it is important to mention that while not all users have rated all movies, at least every user has rated one movie. This is important to know for our analysis.



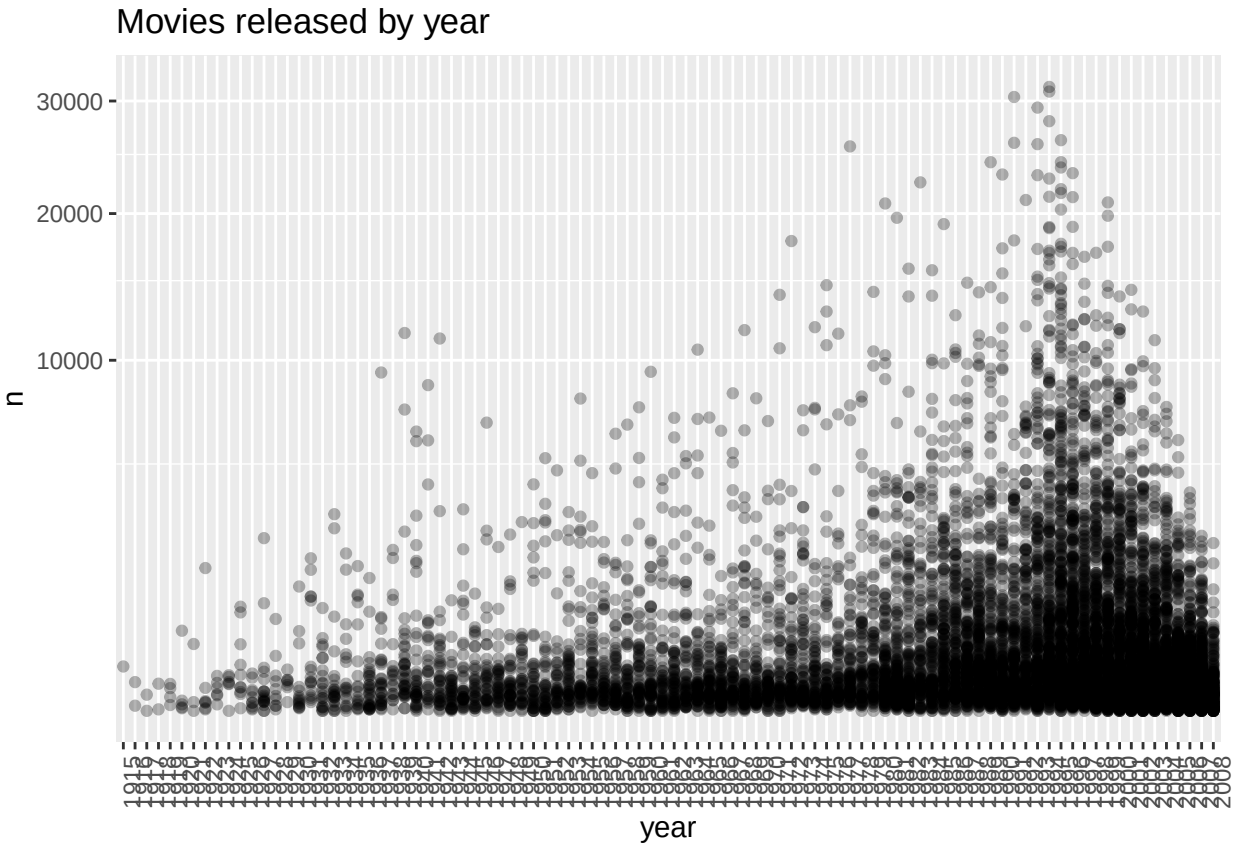
From our initial overview of the data, we noticed that the `timestamp` variable contained observations with very large numbers that did not make much sense. The reason is that `timestamp` is in the "Epoch Unix Timestamp" which counts the number of seconds since January 1, 1970(UTC), to the time of the rating. While this is a standard measure, it is not easy to understand, specially when trying to find exact dates. Therefore we transformed it in two new variables that are easier to interpret.

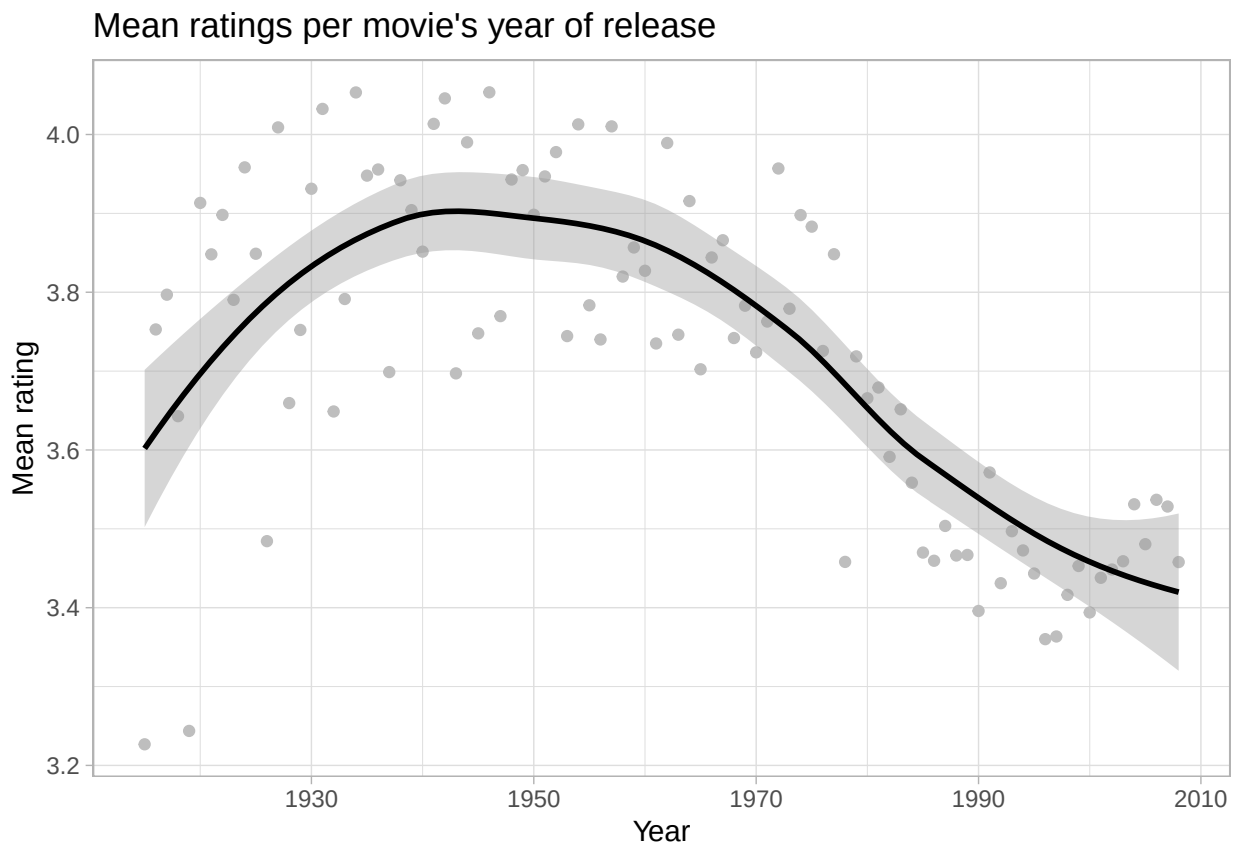
Rating year	Movies rated
1995	3
1996	1385
1997	1663
1998	2263
1999	3010
2000	3811
2001	4652
2002	5684
2003	6738
2004	7820
2005	8302
2006	8524
2007	9177
2008	9582
2009	3443



We also noticed that the `title` variable contains two different values, the title of the movie and the year of release in parenthesis. For the purposes of accuracy of our algorithm, we converted the column into two different variables: `title` and `year`.

Once the title and the year are separate values we can determine, for example, films released by year, or the mean rating per year of release.


Movies released by year
30000
20000
10000
n
year



From our initial overview of the data set we also noticed the `genres` variable sometimes contained more than one value (genre). Therefore, we proceeded to separate this column into unique values of "genre" per observation that would make our algorithm much more accurate.

Once we transform the `genre` variable into a `genres` variable with unique values of "genre" per observation, we now identify 20 unique genres, and each film is listed by every applicable genre, which means that the same film can constitute more than one observation since it can be included in more than one genre.

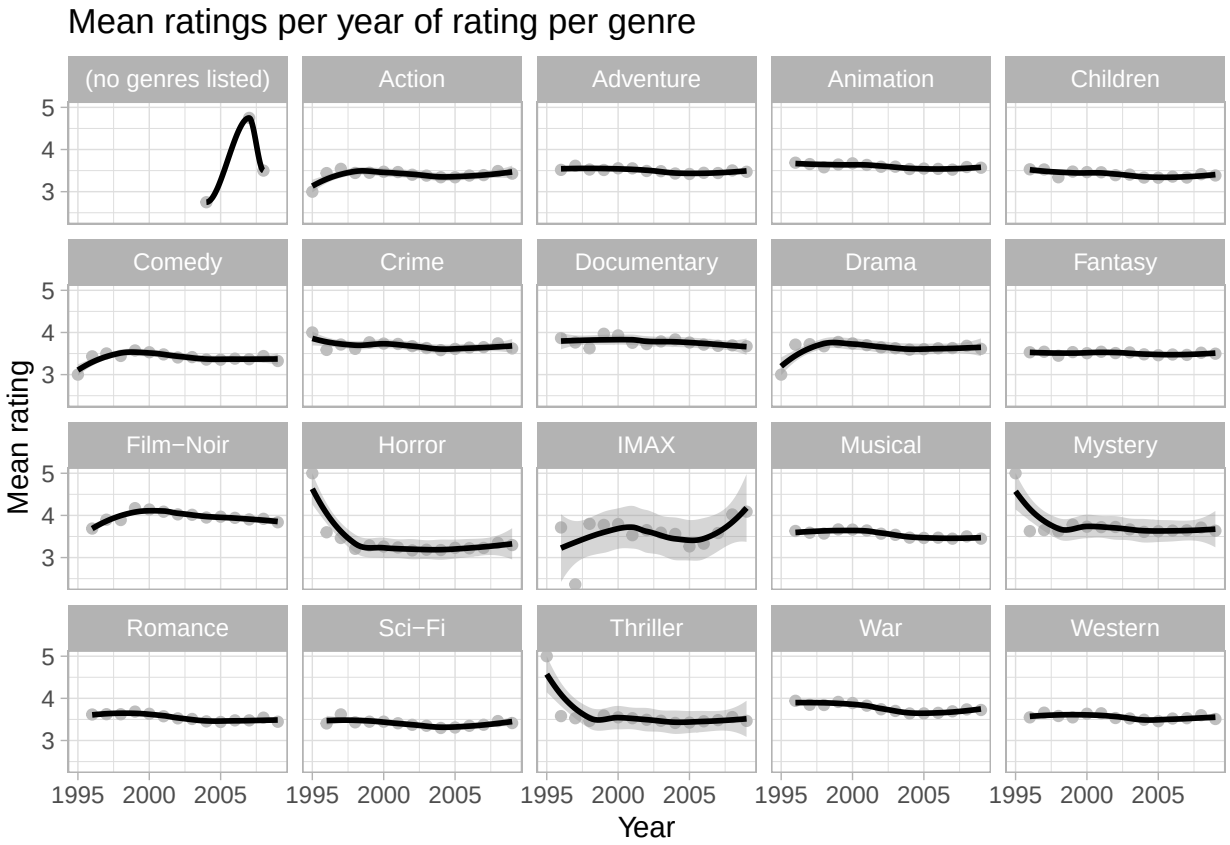
Genres	Movies_per_genre
Drama	3909752
Comedy	3540506
Action	2560942
Thriller	2325309
Adventure	1908844
Romance	1711667
Sci-Fi	1341308
Crime	1327396
Fantasy	925761
Children	737946
Horror	691175
Mystery	567497
War	511427
Animation	467135
Musical	432982
Western	189256
Film-Noir	118650

Genres	Movies_per_genre
Documentary	93231
IMAX	8179
(no genres listed)	5

We can now determine, for example, which are the most popular and less popular genres, and also what is the average rating per genre. Here, "Film Noir" appears to be the most popular genre, whereas "Horror" the least popular.

Genres	Mean_rating
Film-Noir	4.011808
Documentary	3.785093
War	3.779794
IMAX	3.760423
(no genres listed)	3.700000
Mystery	3.677424
Drama	3.673273
Crime	3.665590
Animation	3.598746
Musical	3.562466
Western	3.555636
Romance	3.553824
Thriller	3.507350
Fantasy	3.502030
Adventure	3.493519
Comedy	3.436842
Action	3.421484
Children	3.418099
Sci-Fi	3.396052
Horror	3.269118

Every single genre seems to have had a consistent popularity since 1995, but horror, mystery and thriller film popularity seems to have plateaued from an initial high popularity. IMAX films have fluctuated more, with increases and decreases, and an apparent gaining popularity over the last years recorded in our data.



With all the previous adjustments, our training data set was modified to `23,368,968` observations, and by adding `3` new variables, the data set had a total of `9`.

	Class
userId	integer
movieId	numeric
rating	numeric
timestamp	integer
title	character
year	integer
genres	character
date	POSIXct
rtg_year	integer

3. Methods and analysis

To compare the different models and algorithms, determine the typical error and define the accuracy of our models we used used the root mean squared error (RMSE).

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$

In terms of code, the RMSE is calculated through the following function

```r
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

We also made an extra partition of of our training set to train the best algorithms in a much smaller data set, which means less processing time.

3.1. Naive model

Following Irizarry (2021) our first approach was to start with a "naive model" (not related to a Naïve Bayes analysis), which is a simple prediction based on the average ratings. The simplest possible prediction we can make, then is based on the ratings mean, which we already know that is `3.5`.

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

This initial approach yielded an RMSE of `1.05` which is far from our threshold, but served as a benchmark to compare future results.

3.2. Baseline models

Following Irizarry (2021) we built a baseline model that can measure the "effect" of certain variables on the rating of the film in specific. This initial approach is the same used in the Belkor Solution to the Netflix Prize challenge https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

For example, as we saw, certain movies tend to be rated higher than others and some tend to be more rated that others. So, our first baseline model took the "Movie effect" `b_i`, along with our mean rating *mu* as our predictors. This model can be interpreted in math and code notation as follows:

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

```r
#Movie averages
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

#Predict the influence of the "Movie Effect" in the rating.
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

#We validate our prediction the test set.
rmse_bi <- RMSE(predicted_ratings, test_set$rating)
```

The initial result yielded a `0.94` RMSE on our reduced test set, which was an improvement from the naive model. Since this model seemed to be improving our RMSE we analyzed more predictors to determine how much accurate our algorithm could be.

We considered many assumptions to determine which variables to include in the algorithm. For example, that it appears logical to assume that a user who rates one film would rate similarly other movies of the same genre. Thus, calculating the "Genre Effect" would be extremely relevant. Likewise, we considered that people would like movies released in similar years. That is to say, a person that liked a classic film, might

also like other classic films. This wasn't a compelling enough hypothesis from the beginning, but we wanted to test it anyway, to see if it improved the overall accuracy of the algorithm. Based on the above, we built algorithms considering the effects of movie, user, genre, and release year.

method	RMSE
Movie	0.9414214
Movie+User	0.8589388
Movie+User+Genre	0.8588486
Movie+User+Genre+Year	0.8585346

While every model showed an improvement, after calculating the "User effect" the improvements were getting more marginal and the processing time was increasing.

3.3. Regularizaed models

As stated by Irizari (2021), we had to be aware of over-training in our data. For example, while the estimate for the first movie can be very precise, the following movies estimates will be the "observed deviation from the average rating which is an estimate based on just one number," and it will not be as precise. To avoid this we also penalized the estimates to constrain the total variability and avoid over-training. Thus we introduced regularization to our second type of models. Adding a penalty to the values of b (the different effects), minimized our model's RMSE. The model equations is represented as follows:

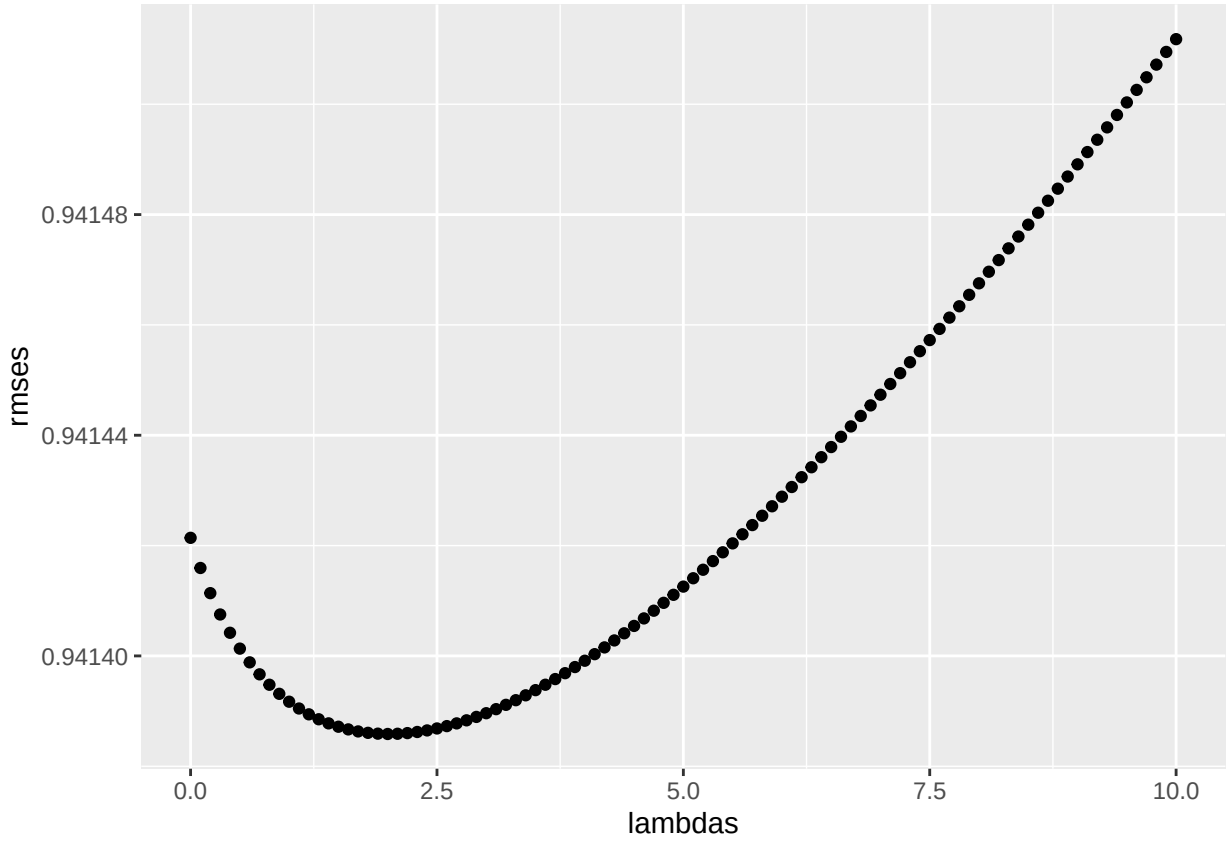
$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2+\lambda\sum_i b_i^2$$

The method for regularization used a similar approach as the baseline models, but this time we added a tuning parameter *lambda*. With cross-validation of a range of values of *lambda*, we then chose the one that generated the smaller error to be included in our model.

```
#We define our values of lambda
lambdas <- seq(0, 10, 0.1)

#Regularized Movie
rmses <- sapply(lambdas, function(l) {
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu) / (n() + l))
  predicted_ratings <- test_set %>%
    left_join(b_i, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

#Best value of lambda
lambdas[which.min(rmses)]

#Now we include this value of lambda in our model.
rmse_reg_bi <- min(rmses)

#And include the results in a table of RMSEs results
```

```
regbi_rmse_res <- tibble(method="Regularized Movie",
                                 RMSE = rmse_reg_bi)
```

For the first of our regularized models, we considered the "Movie Effect" and included the best value of *lambda*.



The model yielded very similar results to the equivalent baseline model, however, we did noticed a slight improvement.

method	RMSE
Regularized Movie	0.9413859

Following what we found on the baseline models we assumed that adding more regularized predictors to our model would increase our overall accuracy. Nevertheless, as we proceeded, we noticed that adding more than two values of b (the different effects), increased the processing time to a point that was no longer viable.

method	RMSE
Regularized Movie	0.9413859
Regularized Movie+User	0.8586647

Thus we decided to stick win only the "Movie" and "User" effects, considering that, by itself this model was already a significant improvement, to the point that it could let us at the or even below our threshold.

3.4. Matrix factorization model

According to Chin, et al. (2016) a "popular technique to solve the recommender[sic] system problem is the matrix factorization method." Chin introduces the `recosystem` package https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html, "an R wrapper of the LIBMF library developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin" (Chin, et al. 2016). According to Chin, "LIBMF (and hence recosystem) can significantly reduce memory use, for instance the constructed model that contains information for prediction can be stored in the hard disk, and output result can also be directly written into a file rather than be kept in memory." Also, according to Chin, `recosystem` provides "convenient functions" to solve "the matrices P and Q" and "additionally have functions for model exporting (outputing P and Q matrices) and prediction."

The basic idea behind `recosystem` consists in "approximate the whole rating matrix Rm×n by the product of two matrices of lower dimensions, Pk×m and Qk×n such that R P Q" with "pu be the u-th column of P, and qv be the v-th column of Q, then the rating given by user u on item v would be predicted as p uqv."

Here we apply the solution to this optimization problem given by Chin et al., where "(u,v) are locations of observed entries in R, ru,v is the observed rating, f is the loss function, and P, Q, P, Q are penalty parameters to avoid overfitting" (Chin, Zhuang, et al. 2015a, 2015b):

$$minP, Q \sum (u, v) \in R[f(pu, qv; ru, v) + \mu P||pu||1 + \mu Q||qv||1 + \lambda P2||pu||22 + \lambda Q2||qv||22]$$

The `recosystem` algorithm requires a `user_index`, an `item_index`, and the `rating`. The `user_index` is an integer vector giving the user indices of rating scores, `item_index` is an integer vector giving the item (movieId) indices of rating scores, and `rating` is a numeric vector of the observed entries in the rating matrix. If the `user index`, `item index`, and `ratings` are stored as R vectors in memory, they can be passed via function `data_memory()`.

```
#Set seed
set.seed(1982, sample.kind = "Rounding")
```

```
## Warning in set.seed(1982, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
#Create a special train set for the matrix factorization analysis
train_rec_mf <-  with(train_set,
                      data_memory(user_index = userId,
                                  item_index = movieId,
                                  rating     = rating))

#Create a special test set for the matrix factorization analysis
test_rec_mf  <-  with(test_set,
                      data_memory(user_index = userId,
                                  item_index = movieId,
                                  rating     = rating))
#Create a model object (a Reference Class object in R) by calling Reco().
r <- recosystem::Reco()

#Call the $tune() method to select best tuning parameters along a set of candidate values.
opts <- r$tune(train_rec_mf,
               opts = list(dim = 30,
                           lrate = c(0.1, 0.2),
                           costp_l2 = 0.1,
                           costq_l2 = 0.1,
```

```
                            nthread  = 4,
                            niter = 10))

#Train the model by calling the $train() method. A number of parameters can be set inside the function,
r$train(train_rec_mf,
        opts = c(opts$min, nthread = 4, niter = 30))

#Predict the rating of the matrix factorization analysis.
predicted_ratings <-  r$predict(test_rec_mf, out_memory())

#Validate prediction on the test set.
rmse_reco_mf <- RMSE(predicted_ratings, test_set$rating)

#Add results to table
reco_rmse_res <- tibble(method= "Recosystem Matrix Factorization",
                        RMSE = rmse_reco_mf)
```

The `recosystem` analysis improved our model exponentially, to the point that that we were confident of bringing RMSE way below the threshold when called into the larger data set.

method	RMSE
Recosystem Matrix Factorization	0.7896929

Up to this point we were conducting experiment with a smaller subset of the training data. Once we trained our different models, we want to see how they operate in the larger training and test sets.

4. Results

Based on our experiments we proceeded to train the different models in the larger data set. We decided to include the naive model as an initial benchmark to document improvement.

```
# Simplest possible prediction is the ratings mean
mu <- mean(edx$rating)
mu

# Computing the predicted ratings on the train set. This model only takes the average rating.
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse

# We store our results in a data frame to be able to compare the different results of our models.
rmse_results <- tibble(method = "Naive Model", RMSE = naive_rmse)
rmse_results
```

Then, we incorporated one of the baseline models. For the baseline model we decided to that included the "Movie" and "User effects". We chose only this two parameters since the margin of improvement including more parameters was not that significant.

```
#User averages
user_avgs <- edx %>%
  left_join(movie_avgs, by="movieId") %>%
```

```
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#We predict the influence of the "User Effect" in the rating.
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

#We validate our prediction the test set.
rmse_biu <- RMSE(predicted_ratings, validation$rating)

#And include the results in a table of RMSEs results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Baseline Movie+User",
                                 RMSE = rmse_biu))
```

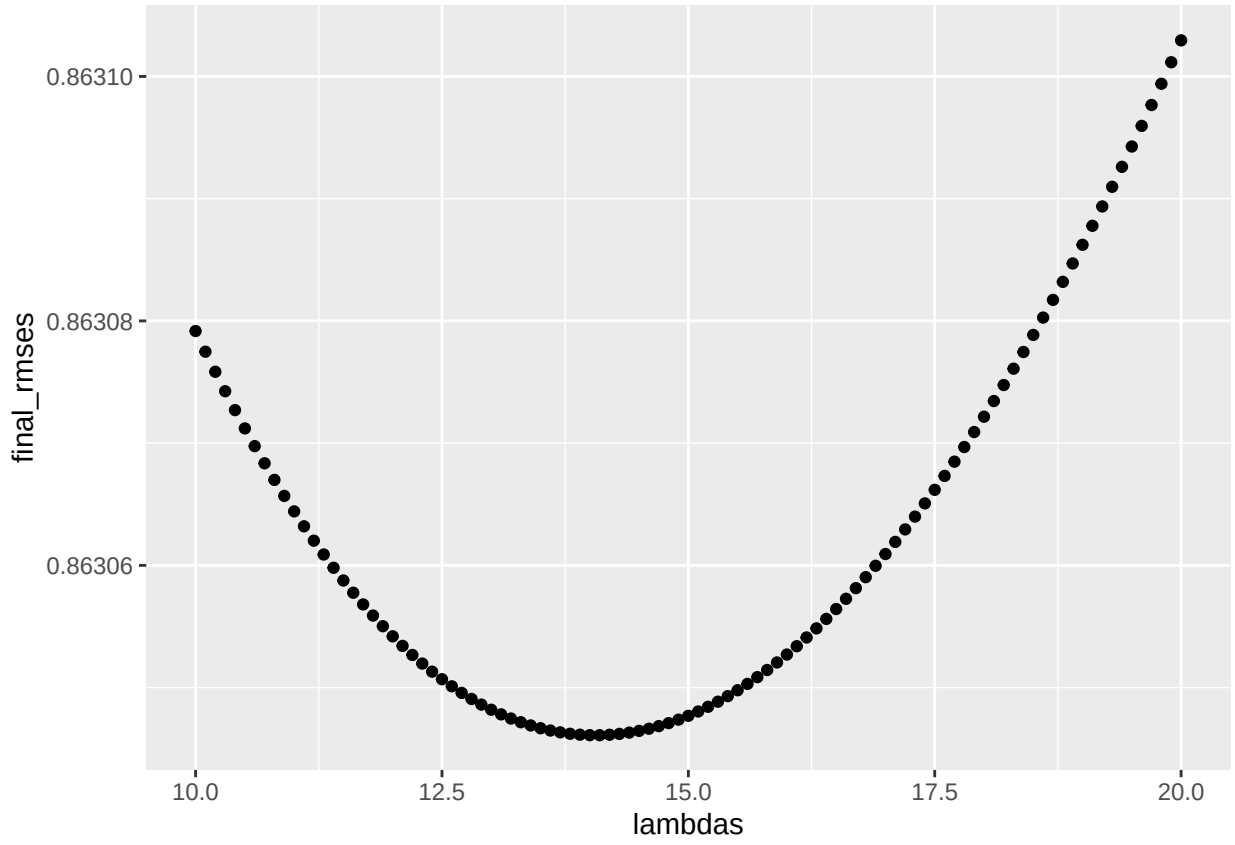
Then we incorporated the regularized model that included the penalized "Movie" and "User effects", with the the best value of *lambda*. As with the naive morel, here we also chose only two parameters since the margin of improvement including more parameters was not that significant, but most importantly because it was no longer viable due to the processing time and system capability.

```
#Set seed
set.seed(1982, sample.kind = "Rounding")
```

```
## Warning in set.seed(1982, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
#During my test I found the best lambda to be over 10, therefore I set my lambdas to be between 10 and
#We define our values of lambda.
lambdas <- seq(10, 20, 0.1)

#Regularized Movie+User
final_rmses <- sapply(lambdas, function(l) {
  b_i <- edx %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu) / (n() + l))
  b_u <- edx %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - b_i - mu) / (n() + l))
  predicted_ratings <- validation %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(predicted_ratings, validation$rating))
})

#Include the lowest value of lambda in our model.
rmse_reg_biu <- min(final_rmses)
```

```
#And include the results in a table of RMSEs results
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie+User",
                                 RMSE = rmse_reg_biu))
```



Finally, we incorporated the `recosystem` matrix factorization model.

```
#Set seed
set.seed(1982, sample.kind = "Rounding")
```

```
## Warning in set.seed(1982, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
#Create a special train set for the matrix factorization analysis
train_rec_mf <-  with(edx, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating     = rating))

#Create a special test set for the matrix factorization analysis
test_rec_mf  <-  with(validation,  data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating     = rating))

#Create a model object (a Reference Class object in R) by calling `Reco()`.
```

```
r <- recosystem::Reco()

#Call the $tune() method to select best tuning parameters along a set of candidate values.
opts <- r$tune(train_rec_mf, opts = list(dim = 30,
                                         lrate = c(0.1, 0.2),
                                         costp_l2 = 0.1,
                                         costq_l2 = 0.1,
                                         nthread  = 4,
                                         niter = 10))

#Train the model by calling the $train() method.
r$train(train_rec_mf, opts = c(opts$min, nthread = 4, niter = 30))

#Predict the rating of the matrix factorization analysis.
predicted_ratings <-  r$predict(test_rec_mf, out_memory())

#Validate prediction on the test set.
rmse_reco_mf <- RMSE(predicted_ratings, validation$rating)

#Add to results table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Recosystem Matrix Factorization",
                                 RMSE = rmse_reco_mf ))
```

The different models tested in the larger data set proved our initial assumptions.

method	RMSE
Naive Model	1.0519086
Baseline Movie+User	0.8638741
Regularized Movie+User	0.8630461
Recosystem Matrix Factorization	0.8118124

First, the regularized model improved the accuracy much better than the simple baseline models. Nevertheless, the `recosystem` matrix factorization analysis proved to be the best, and improved significantly the accuracy of our prediction, much better that we initially anticipated. Also, the processing time for this algorithm was way less than the regularized model, making it a better option four our purpose.

5. Conclusion

Recommendation systems are becoming more and more important as we increased our consumption of digital goods. Thus the importance of creating ML algorithms that reduce significantly the error in prediction. In our case the process was complicated due to the impossibility to use more common ML algorithms such as `random forests` or `logistical regression`.

Initially the only approach possible appeared to be the Belkor Solution to the Netflix Prize challenge, also replicated by Irizarry (2021). However, after considerable research, we were able to find the `recosystem` approach, which, as seen above not only yielded the best results, but was the most efficient in processing time.

Approaches like the `recosystem` analysis show how can we keep improving recommendation systems, specially as more and more data becomes available and more platforms depend on such systems to offer their digital contents to an ever-growing pool of users.

6. Sources

Chin, et al. (2016). LIBMF: A Library for Parallel Matrix Factorization inShared-memory Systems. Journal of Machine Learning Research 17, 1-5. https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/libmf_open_source.pdf

GroupLens (2021). MovieLens 10M. https://grouplens.org/datasets/movielens/10m/.

Irizarry (2021). Introduction to Data Science: Data Analysis and Prediction Algorithms with R. https://rafalab.github.io/dsbook/

Koren, Yehuda (2009). The BellKor Solution to the Netflix Grand Prize. https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.

Qiu, Yixuan (2021). recosystem: Recommender System Using Parallel Matrix Factorization. https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html.