Movie Lens Project - Iakovos Tselentis

Iakovos Tselentis

10/17/2020

Introduction

Summary

This project is part of the Capstone course for HarvardX's *Professional Certificate in Data Science* program. The goal is to create a basic movie recommendation system that predicts how many stars a user will give to a specific movie using skills that were taught in the previous 8 courses of the program. The accuracy of the algorithm will be evaluated using the *Root-Mean-Square-Error (RMSE)*, which is defined as follows:

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

We will now also define the function above in R using the code below:

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

In the following sections we are going to:

- Explore our dataset
- Split it into the edx(Training) and validation(Test) set
- Use the edx set to create and test our models
- Implement the best performing model in the validation set
- Summarize the report, draw conclusions and outline its limitations

Before all that we make sure that we have all the packages needed to complete the project.

```
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")
library(tidyverse)
```

```r
library(caret)
library(data.table)
library(lubridate)
library(ggthemes)
library(stringr)
memory.limit(60000)
```

```
## [1] 60000
```

Dataset Exploration

Download

The dataset for this project is called Movie Lens and is provided by *GroupLens Reasearch Lab*.The dataset is free, and accessible via the following *link*. In the next code chunks we are going to load the Movie Lens dataset. Then we partition it in two sets, the edx and the final hold-out set. The edx set is going to be used in the next sections to train and test our model. Once we decide on the optimal model we are going to implement it in the validation set(final hold-out set) and compare it with the actual ratings to calculate the final RMSE.

```r
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")

# if using R 3.6 or earlier
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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# 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,]

# Make sure userId and movieId in validation set are also in edx set
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)
```

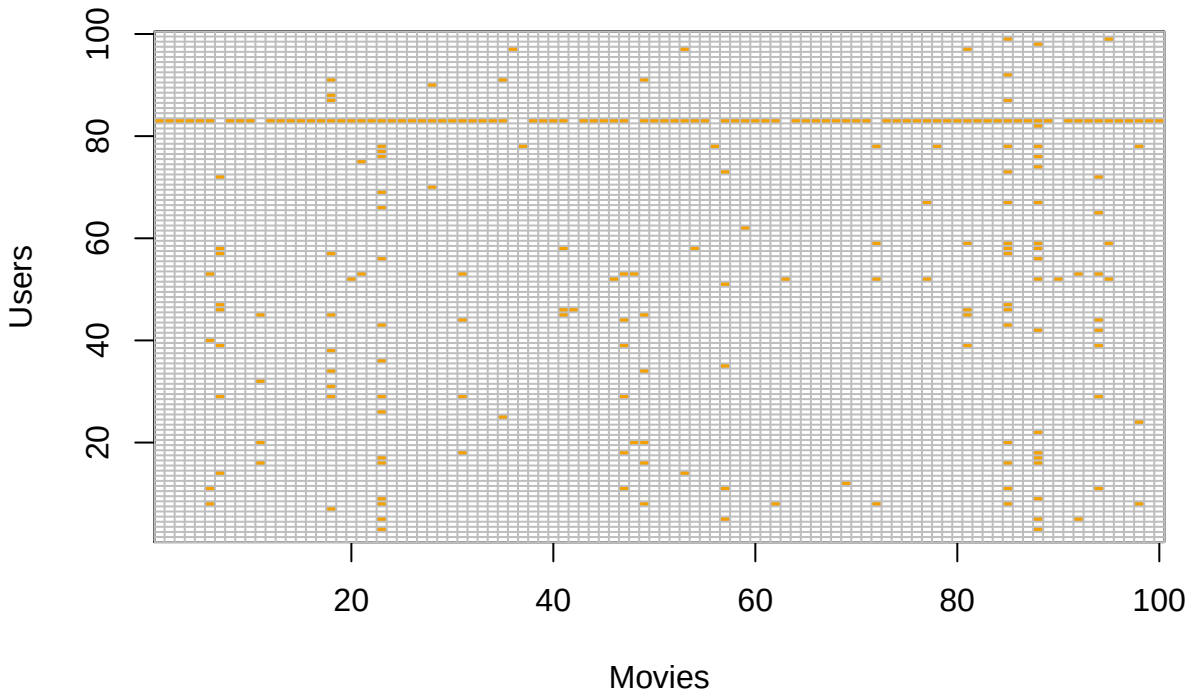
```
edx <- rbind(edx, removed)

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

Exploration

Let's first take a look at the top 6 rows of the dataset:

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

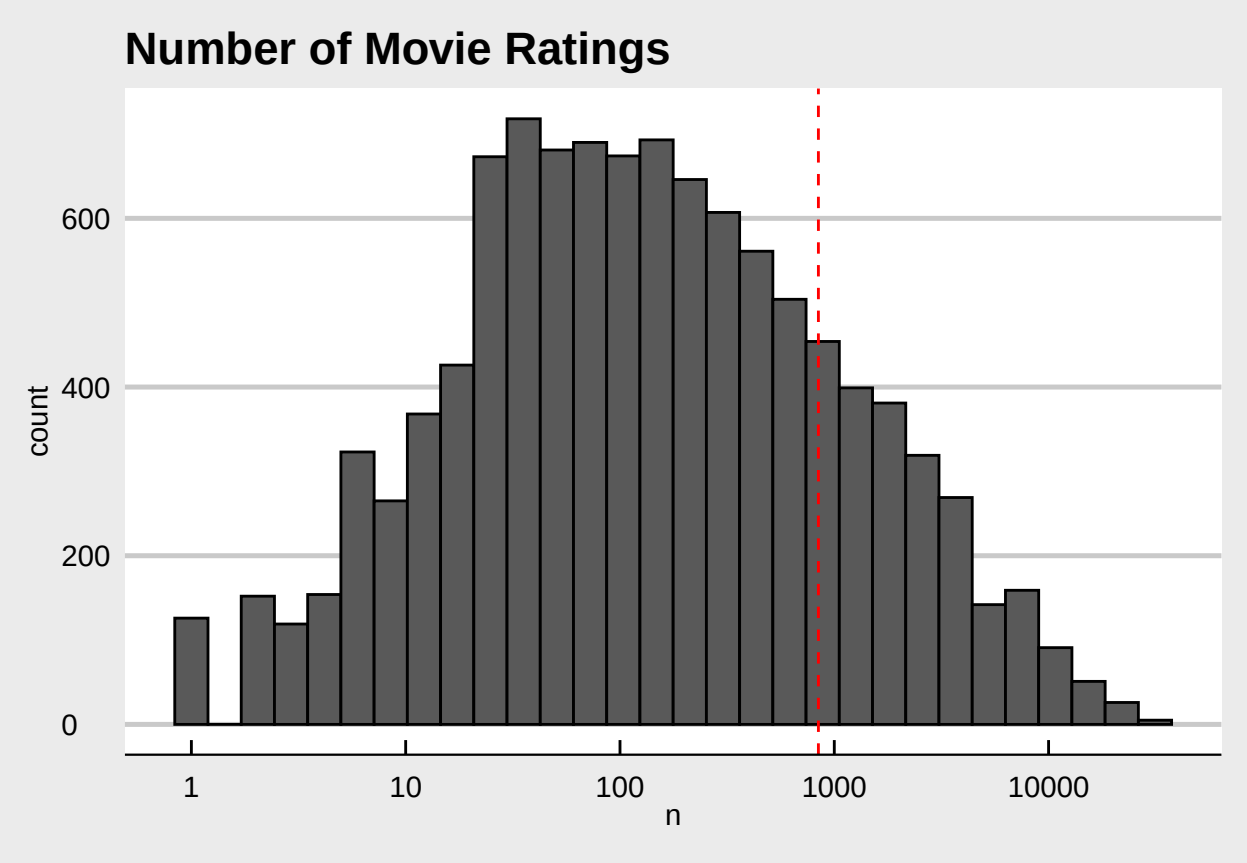
The edx dataset has 9000055 rows and 6 columns.Each row represents a rating given by one user to one movie. But did every user rate every movie? Taking a random sample of 100 users and 100 movies we see that our dataset is quite sparse.

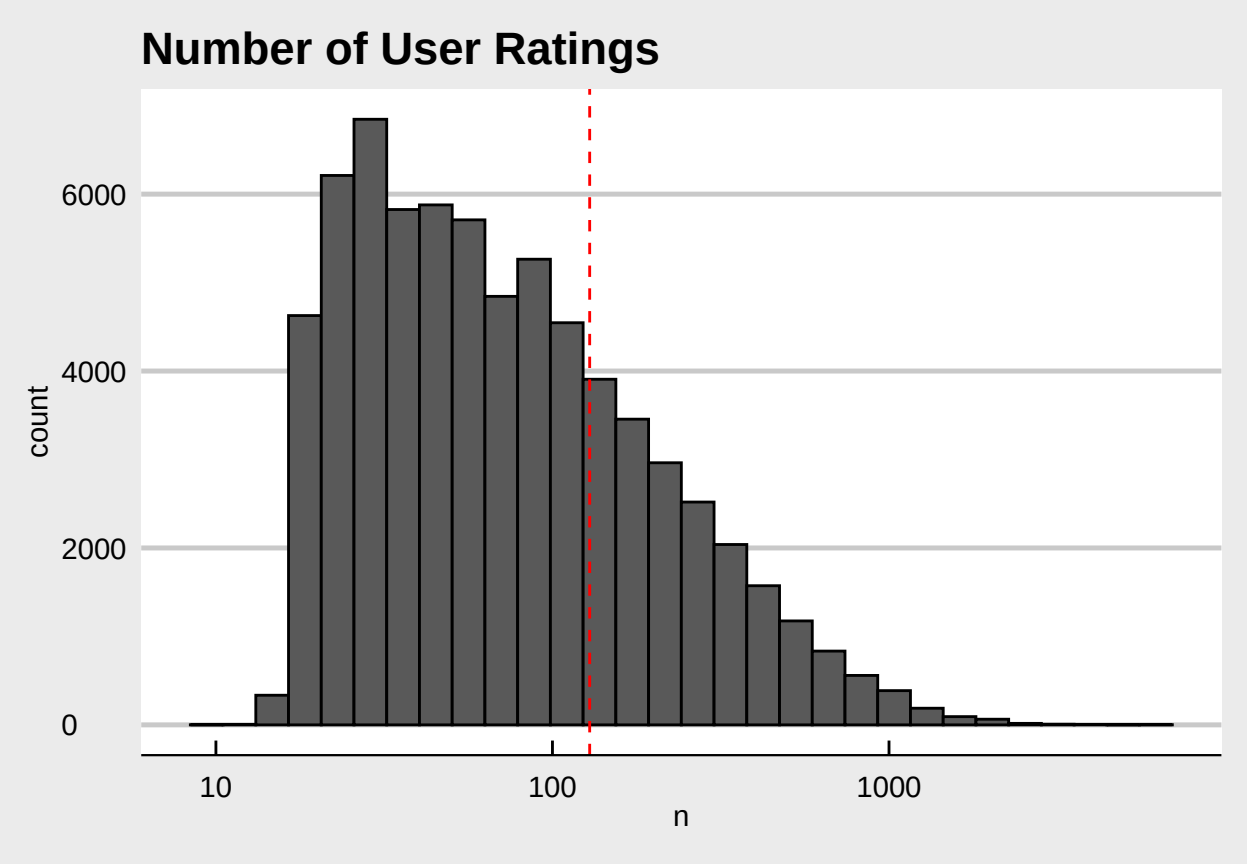


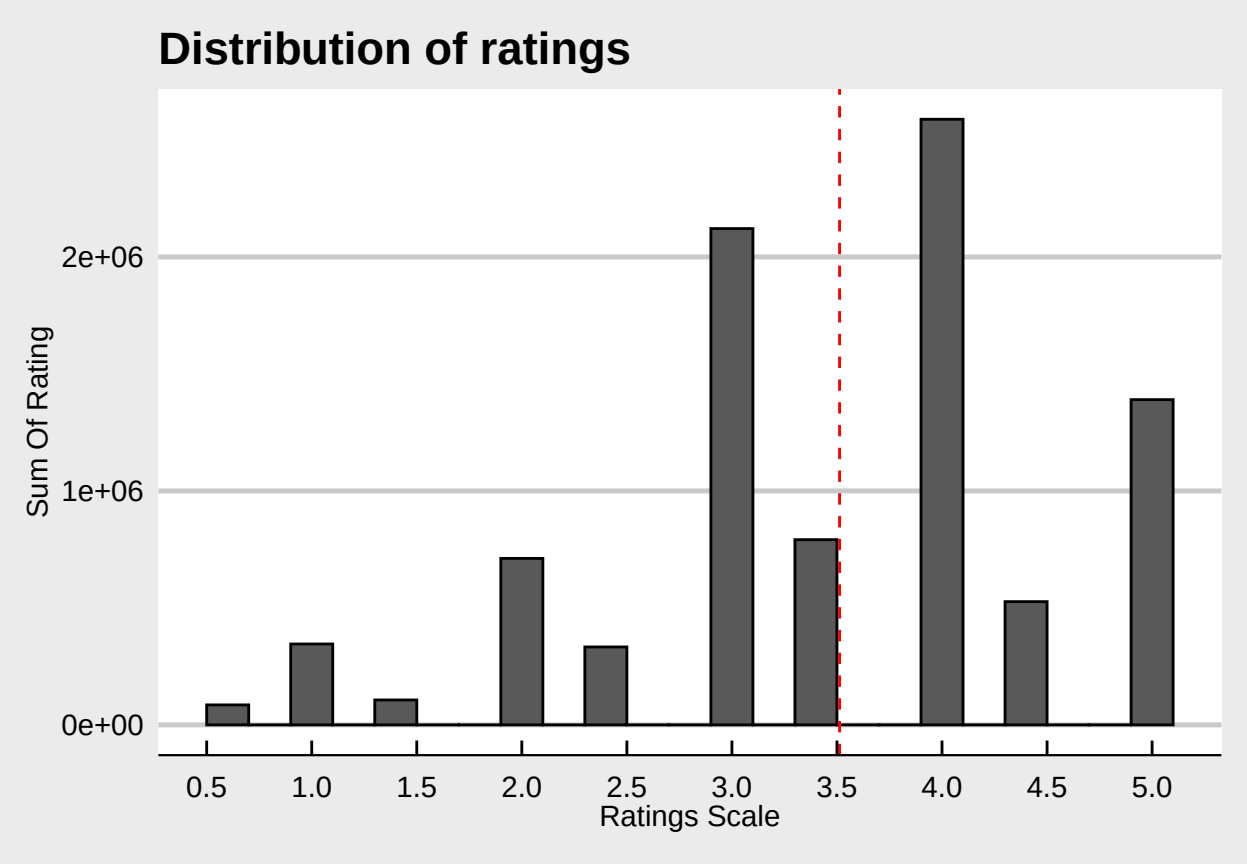
Also, we can notice that our dataset needs to be slightly manipultated in order to contain useful information for our analysis. First we are going to extract the year from the title using regex expressions, convert the timestamp to a date and extract the year of the rating, as well as create a new column with the movie age on rating.

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     292      5 838983421               Outbreak (1995)
## 4      1     316      5 838983392               Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                          genres year_released year_rated
## 1                Comedy|Romance          1992       1996
## 2         Action|Crime|Thriller          1995       1996
## 3  Action|Drama|Sci-Fi|Thriller          1995       1996
## 4       Action|Adventure|Sci-Fi          1994       1996
## 5 Action|Adventure|Drama|Sci-Fi          1994       1996
## 6       Children|Comedy|Fantasy          1994       1996
```

Graphs


Number of Movie Ratings
count
600
400
200
0
1
10
100
1000
10000
n


Number of User Ratings
6000
4000
2000
0
count
10
100
1000
n



Analysis

In this section we are going to start building our models that predict movie ratings, using the *edx* set. To predict the ratings we are gradually going to introduce new factors in each model. By adding new factors we try to reduce the ambiguity caused by the residuals of the previous model and hopefully increase the accuracy of our prediction.

Before that however, we need to partition our edx set into a train and a test set. We are going to train each model on the train set and implement the prediction on the test set to reduce the probability of overfitting our model.

```
#Data partition
set.seed(1,sample.kind="Rounding")
partindex<- createDataPartition(edx_A$rating,p=0.1,list=FALSE)
train_set<-edx_A[-partindex,]
test_set<-edx_A[partindex,]
#we also remove all the movies from the test set that do not appear in the train set
test_set<- test_set %>%  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Model 1: Just the Average

Our first model predicts that each user will give each movie the average movie rating. This rating according to our edx train set is at 3.5 stars.

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

```
mu <- mean(train_set$rating)
naive_rmse<-RMSE(test_set$rating,mu)
RMSE_by_method <- data_frame(method = "Just the average", RMSE = round(naive_rmse,digits=3))
RMSE_by_method%>%knitr::kable()
```

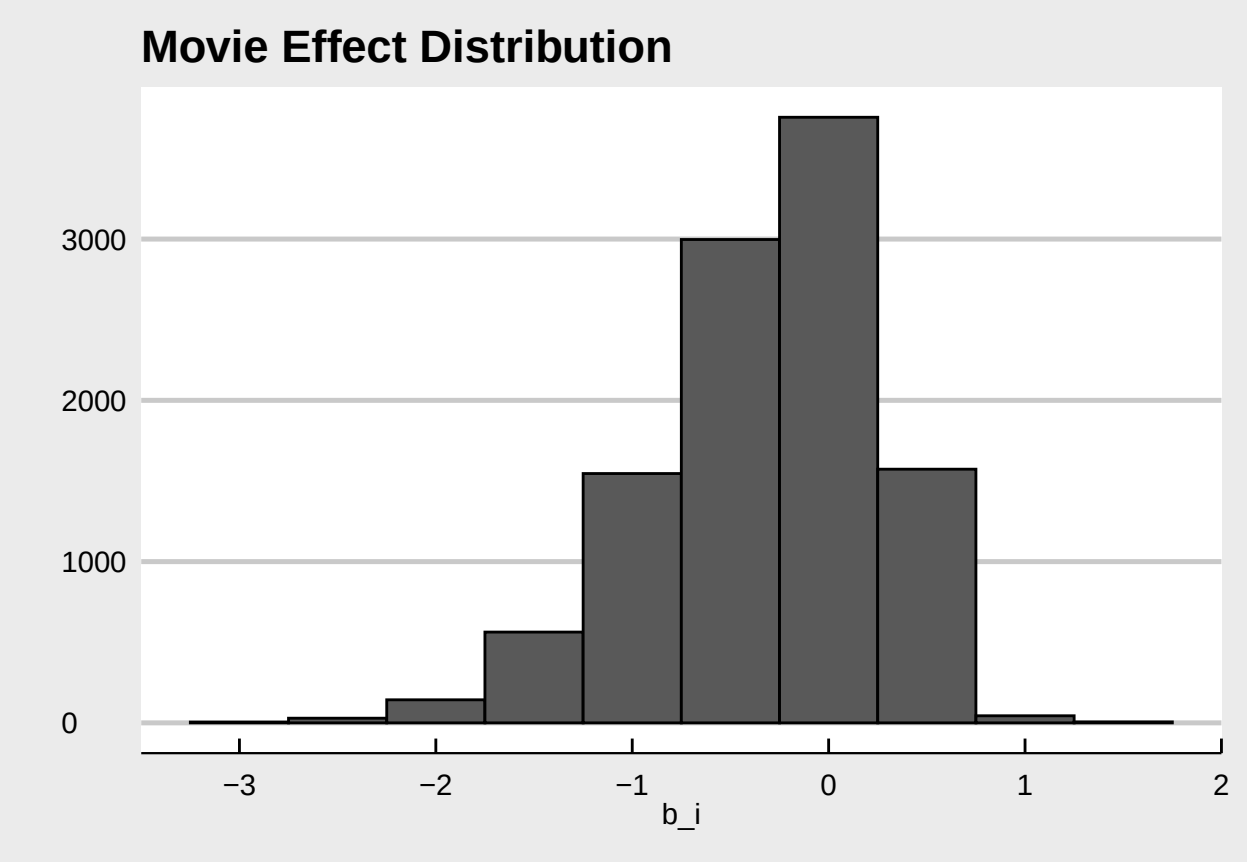
method	RMSE
Just the average	1.06

Model 2: Movie Effects

Our Previous model's predictions deviate from the actual ratings by more than one star on average. For this reason we are going to introduce one more factor that will hopefully explain the residuals of the 'Just the Average' Model. This factor is going to explain the effect of each movie.

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

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))


qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))+
  theme_economist_white()+
  ggtitle("Movie Effect Distribution")
```




```
test_set<- test_set %>%
  left_join(movie_avgs, by='movieId')


predicted_ratings <- mu + test_set %>%
  pull(b_i)



model_2_rmse <- RMSE(test_set$rating,predicted_ratings)
RMSE_by_method<-bind_rows(RMSE_by_method,data_frame(method="Movie Effect Model",RMSE=model_2_rmse))
RMSE_by_method%>%knitr::kable()
```

method	RMSE
Just the average	1.0600000
Movie Effect Model	0.9429615

From the graph above we see that most of the movies, as expected revolve around 0 deviation from the mean (3.5 stars), but there are certain movies that have on average much better or much worse rating. The movie effects (b_i) factor helps us reduce the RMSE to 0.94

Model 3: Movie Effects + User Effects

Let's see if we can further decrease RMSE by adding the user effects. This factor will show the tendency that users have to rate movies higher or lower than the average rating.

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

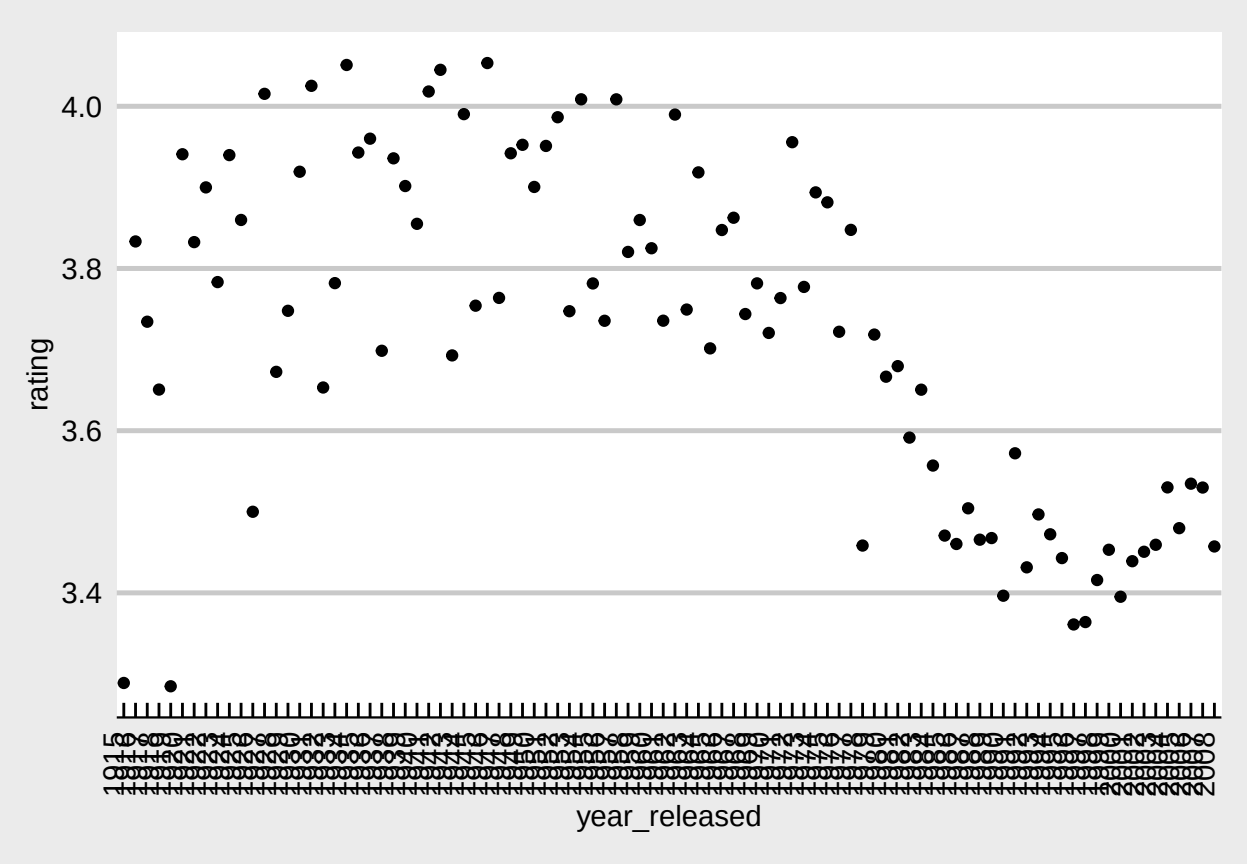
```
train_set<- train_set%>% mutate(mu= mu) %>% left_join(movie_avgs,by='movieId')
user_avgs <-train_set %>% group_by(userId)%>%
  summarise(b_u= mean(rating-mu-b_i))

test_set<- test_set %>%
  left_join(user_avgs, by='userId')


predicted_ratings <- mu + test_set %>%
  pull(b_i) + test_set %>%
  pull(b_u)


model_3_rmse <- RMSE(test_set$rating,predicted_ratings)
RMSE_by_method<-bind_rows(RMSE_by_method,data_frame(method="Movie Effect & User Effect Model",RMSE=model
RMSE_by_method%>%knitr::kable()
```

method	RMSE
Just the average	1.0600000
Movie Effect Model	0.9429615
Movie Effect & User Effect Model	0.8646844

Our prediction has significantly improved again, with the RMSE reduced to 0.865

Model 4: Movie Effects + User Effects + Release Year Effect

In the next model we are adding the release year effect. By adding a new factor in our model, we will explore if movies released in a certain year had a better average rating, capturing the 'golden eras' of cinema in our model. First, let's graph the average ratings for each year.



It seems that the data follow a pattern, movies after the 80s have received a lower average rating. While, movies released in the 30s and 70s have a better average rating. So we are now going to implement the new factor in our model as described in the equation below:

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

```
train_set<- train_set%>%  left_join(user_avgs,by='userId')

release_year_avgs <-train_set %>% group_by(year_released)%>%
  summarise(b_ry= mean(rating-mu-b_i-b_u))


#implementing year averages in the test set

test_set<- test_set%>%
  left_join(release_year_avgs, by='year_released')

predicted_ratings <- mu + test_set %>%
  pull(b_i) + test_set %>%
  pull(b_u) + test_set %>%pull(b_ry)

model_4_rmse <- RMSE(test_set$rating,predicted_ratings)
RMSE_by_method<-bind_rows(RMSE_by_method,data_frame(method="Movie Effect & User Effect & Release Year E
train_set<- train_set%>% mutate(mu= mu) %>% left_join(release_year_avgs,by='year_released')

RMSE_by_method%>%knitr::kable()
```

method	RMSE
Just the average	1.0600000
Movie Effect Model	0.9429615
Movie Effect & User Effect Model	0.8646844
Movie Effect & User Effect & Release Year Effect	0.8643302

The new model has marginally improved our prediction, reducing the RMSE to 0.864

Regularization

Our previous models have yielded good results, however we can still improve the predictions. Let's see where are the biggest errors.

title	residual	prediction	rating
Gigli (2003)	5.200030	-0.2000301	5
Smokey and the Bandit III (1983)	4.564385	0.4356149	5
From Justin to Kelly (2003)	4.353827	0.6461730	5
Cruel Intentions (1999)	4.334896	0.6651039	5
Cavalcade (1933)	4.320612	0.6793879	5
Country Bears, The (2002)	4.123017	0.8769828	5
Bulletproof (1996)	3.924575	1.0754250	5
Haunting, The (1999)	3.900081	1.0999187	5
House of the Dead, The (2003)	3.832838	1.1671622	5
Lizzie McGuire Movie, The (2003)	3.823266	1.1767335	5

Our biggest errors are for movies that are largely unpopular.

Let's see the top 10 worst and best movies using the movie according to our movie bias factor and their respective ratings

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487543	1
Satan's Tango (SÎ¨tÎ¨ntangÎ³) (1994)	1.487543	1
Shadows of Forgotten Ancestors (1964)	1.487543	1
Fighting Elegy (Kenka erejii) (1966)	1.487543	1
Sun Alley (Sonnenallee) (1999)	1.487543	1
Blue Light, The (Das Blaue Licht) (1932)	1.487543	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237543	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237543	2
Human Condition II, The (Ningen no joken II) (1959)	1.237543	4
Human Condition III, The (Ningen no joken III) (1961)	1.237543	4

```
## Joining, by = "movieId"
```

title	b_i	n
Besotted (2001)	-3.012457	1
Hi-Line, The (1999)	-3.012457	1
Confessions of a Superhero (2007)	-3.012457	1
War of the Worlds 2: The Next Wave (2008)	-3.012457	2
SuperBabies: Baby Geniuses 2 (2004)	-2.767776	47
Disaster Movie (2008)	-2.745790	30
From Justin to Kelly (2003)	-2.638140	183
Hip Hop Witch, Da (2000)	-2.603366	11
Criminals (1996)	-2.512457	1
Mountain Eagle, The (1926)	-2.512457	2

We see that the number of ratings for the top 10 and worst 10 movies is quite lower than the average number of ratings in our sample which is 759.783134790357.

Following the same process for our other user effect we see that the outliers are users that have given less movie ratings that the average which is 115.916998196857.

```
## Joining, by = "userId"
```

userId	b_u	n
13496	-3.418827	15
48146	-3.233854	21
49862	-3.163456	16
63381	-3.020395	16
62815	-2.928849	19
6322	-2.847928	16
15515	-2.654739	28
42019	-2.638786	25
19059	-2.599914	17
6907	-2.572646	20

```
## Joining, by = "userId"
```

userId	b_u	n
13524	1.894167	19
18591	1.863840	18
46484	1.804013	22
45895	1.781367	16
52749	1.765072	79
36896	1.752568	133
1943	1.744700	15
7999	1.730383	34
36022	1.686395	77
1	1.668560	18

Similarly, the Release Years that have the best average rating according to our model are years with very few observations.

```
## Joining, by = "year_released"
```

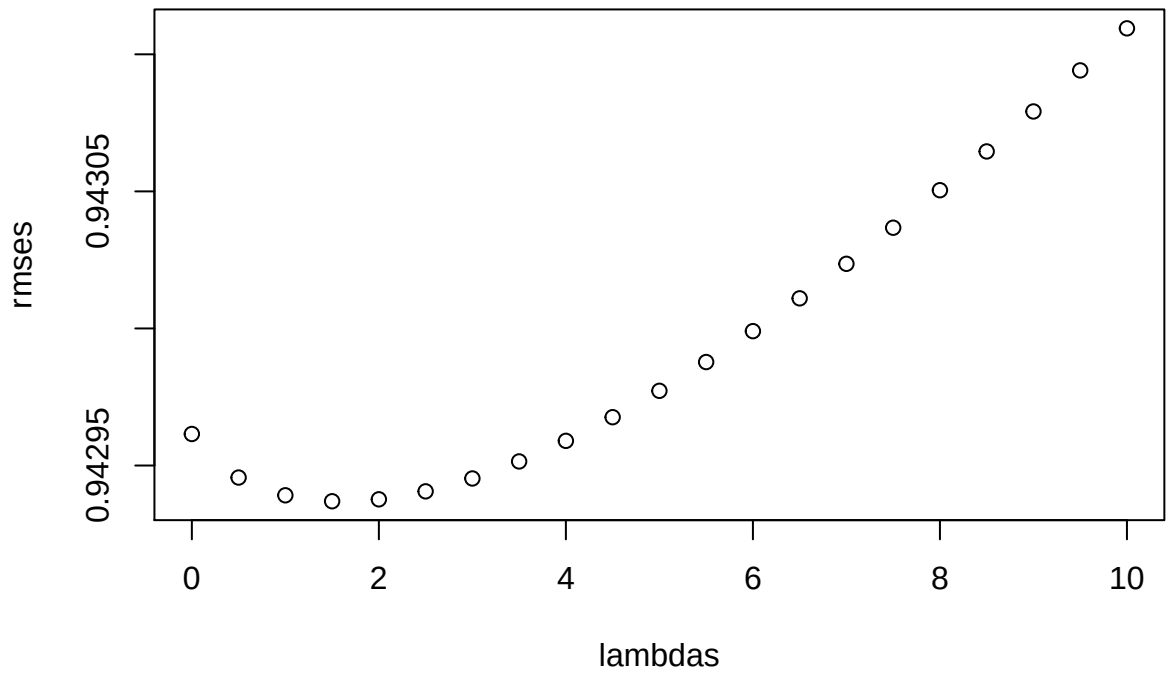
year_released	b_ry	n
1996	-0.0392964	534246
1995	-0.0379376	707979
1994	-0.0319427	604423
1993	-0.0183397	432917
1937	-0.0073147	12115
1977	-0.0030896	53989
1999	-0.0021340	440618
1990	-0.0010597	207482
1991	-0.0004616	177213
1983	0.0028923	84858

```
## Joining, by = "year_released"
```

year_released	b_ry	n
1917	0.1732288	30
1919	0.1556957	144
1923	0.1320376	281
1920	0.1229887	515
1929	0.1143143	519
1924	0.1094592	414
1926	0.1001389	352
1915	0.0883440	163
1921	0.0876467	362
1922	0.0755381	1646

Penalty term

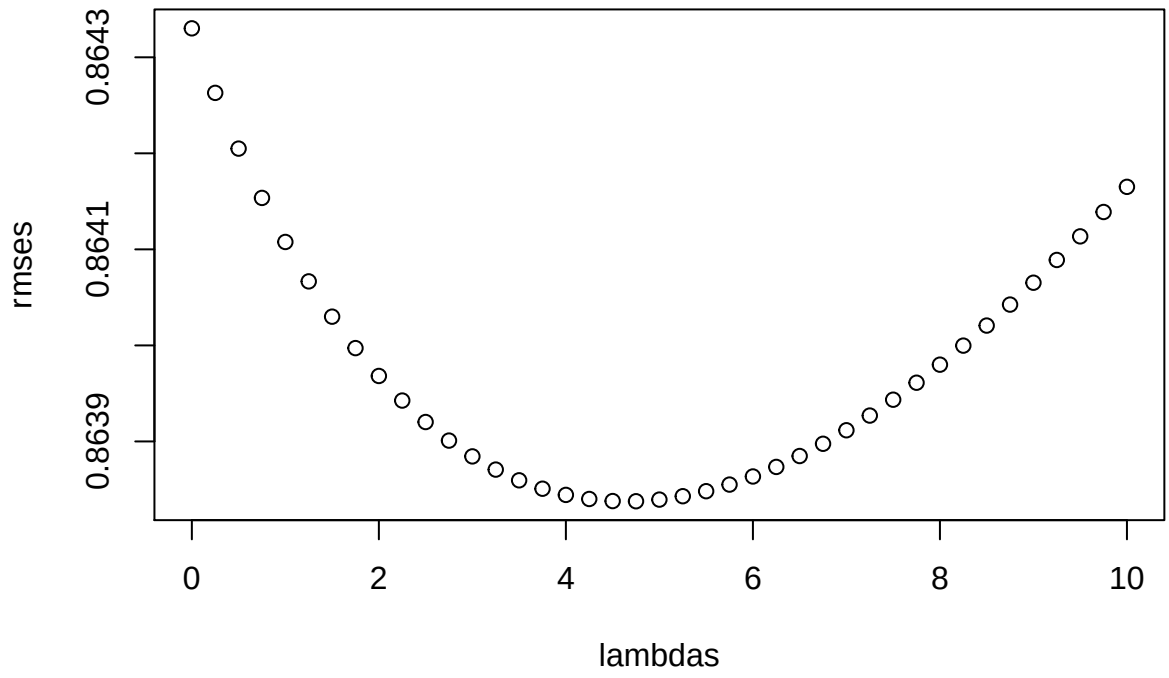
We see that the biggest errors are generate by factors with too few observations. For this reason we are going to introduce a term λ that will penalize large estimates from small samples. All factors(biases) will be re-estimated. This time instead of using the simple mean for each factor we are going to divide the sum of each group of observations with the sample size plus the λ term. This way, groups with large sample size are not going to be affected, but groups with small sample sizes are going to be pulled towards 0 deviation.




```
## # A tibble: 1 x 2
##   lambdas rmses
##     <dbl> <dbl>
## 1     1.5 0.943
```

```
## [1] 0.942937
```

```
## Error in eval(expr, envir, enclos): object 'model_1_rmse' not found
```



It seems that the best RMSE is return when the penatly term lambda is set to 4.75.

Now, let's compare all models.

method	RMSE
Just the average	1.0600000
Movie Effect Model	0.9429615
Movie Effect & User Effect Model	0.8646844
Movie Effect & User Effect & Release Year Effect	0.8643302
Regularized Movie Effect & User Effect & Release Year Effect	0.8638377

Results

After comparing all models we reach to the conclusion that the 'Regularized Movie Effect & User Effect & Release Year Effect' model with the penalty error set at 4.75 yields the best RMSE from all previous models. Now we can implement it in our validation set and calculate the final RMSE.

```
l<- 4.75


mu <- mean(edx_A$rating)

b_i_r <- edx_A %>%
  group_by(movieId) %>%
  summarize(b_i_r = sum(rating - mu)/(n()+l))
```

```
b_u_r <- edx_A %>% left_join(b_i_r,by="movieId")%>%
  group_by(userId) %>%
  summarize(b_u_r = sum(rating - mu-b_i_r)/(n()+l))

b_ry_r <- edx_A %>% left_join(b_u_r,by="userId")%>%
  left_join(b_i_r,by="movieId")%>%
  group_by(year_released) %>%
  summarize(b_ry_r = sum(rating - mu-b_i_r-b_u_r)/(n()+l))


predicted_ratings <- validation_A %>%
  left_join(b_i_r,by="movieId") %>%
  left_join(b_u_r,by="userId") %>%
  left_join(b_ry_r,by="year_released")%>%
  mutate(pred = mu + b_i_r+b_u_r+b_ry_r)%>% .$pred



FinalRMSE <- RMSE(validation$rating,predicted_ratings)
```

Our final RMSE calculation is 0.8645223.

Conclusion

In this short report we have demonstrated how using big data and simple models can help us create strong algorithms for predicting results and reccommending content. We have touched upon the basic principles of data science. We loaded a dataset from an online source, we cleaned it and used basic text mining functions to extract important information, we graphed our data and introduced gradually 3 factors and a penalty error that manage to predict the ratings of users for movies to a satisfactory level.

The analysis and the prediction could be significantly more accurate if we had used machine learning techniques to factorize the matrix, such as singular value decomposition or principal component analysis. These techniques are also what used by the famous reccommendation algorithms of companies such as Netflix, Amazon, Google and Facebook.However, this would require significantly more time and computing power.