MovieLens Project: Movie recommendation system

HarvardX PH125.9x Data Science: Capstone

Priscila Trevino Aguilar

December, 2020

Acknowledgement

I would like to thank professor Irrizary and the whole HarvardX staff for their superb work on this course, I was very pleased with the content and assignments and I consider that the capstone project did an excellent job of summarizing the skills learned and encouraging practice of the most relevant aspects of the course. Completing this capstone project was an exciting and challenging experience in my path to becoming a data scientist.

1. Introduction

The aim of this capstone project was to employ the R-programming and statistics skills obtained throughout the course as well as applying the learned machine learning techniques to create a movie recommendation system. A machine learning algorithm was trained to predict user movie ratings (0 to 5 stars) in a validation set using the inputs of a provided subset.

The basic purpose of a recommendation system is to find and recommend items that a user is most likely to be interested in, they are commonly classified into two categories: content based methods, which are based on similarity of item attributes and collaborative methods, which calculate similarity from interactions. Modern systems usually combine both approaches. Recommendation systems have proven to be valuable means for online users to cope with the information overload and have become one of the most successful and widespread applications of machine learning technologies in business, their development is a multi-disciplinary effort which involves experts from other fields such as Human Computer Interaction, Information Technology, Data Mining, Statistics, Adaptive User Interfaces, Decision Support Systems, Marketing, and Consumer Behavior (Gorakala et al., 2015) (Ricci et al., 2011).

The data set employed in the present project is a publicly available and extensive (around 10 million) movie ratings version of the MovieLens data set developed by GroupLens, a research lab specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

The challenge presented in this capstone project was inspired by the Netflix challenge. In October 2006, Netflix offered a challenge to the data science community to improve their recommendation algorithm by 10% and win a million dollars. The approach taken by the challenge's winning team was collaborative filtering along with an ensemble of complex techniques and models (Irrizary, 2019), however, the methods employed in this project are more simple. First, pertinent data wrangling and exploratory data analysis were performed, followed by the creation of a simple naive model that assumed the same rating for all movies and users with all differences explained by random variation, the model was then modified by adding terms to account for the bias or effect introduced by the movie, user, genre and year variables. Regularization was also employed to come up with a final optimal model. The evaluation metric of choice for the models in this project was the root mean squared error (RMSE), a frequently used measure of the differences between values predicted by a model and the values observed. Finally, the random forest algorithm was also fit to data to evaluate its performance with such an extensive data set and its effectiveness compared to the generated final model.

2. Methods

Exploratory data analysis

The structure of the data set showed that it contained 9000055 observations of both numeric and character variables and that it’s multivariate.

```
str(edx, width=80, strict.width="cut")
```

```
## Classes ’data.table’ and ’data.frame’:   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474..
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "St"..
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

A look at summary and class of the variables gave a more clear idea of their properties and values. It was also observed that the data set contains 6 variables of interest: userId, the movieId, the movie rating, the timestamp, the movie title (which also contains the release year), and the movie genre(s). The movie rating was treated as the output variable in this case.

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
sapply(edx, class)
```

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "numeric"   "numeric"   "integer" "character" "character"
```

The number of distinct users and movies rated were computed.

```
n_distinct(edx$userId)  # users
```

```
## [1] 69878
```

```r
n_distinct(edx$movieId)  # movies
```

```
## [1] 10677
```

The data set did not contain any NA values.

```r
any(is.na(edx))
```

```
## [1] FALSE
```

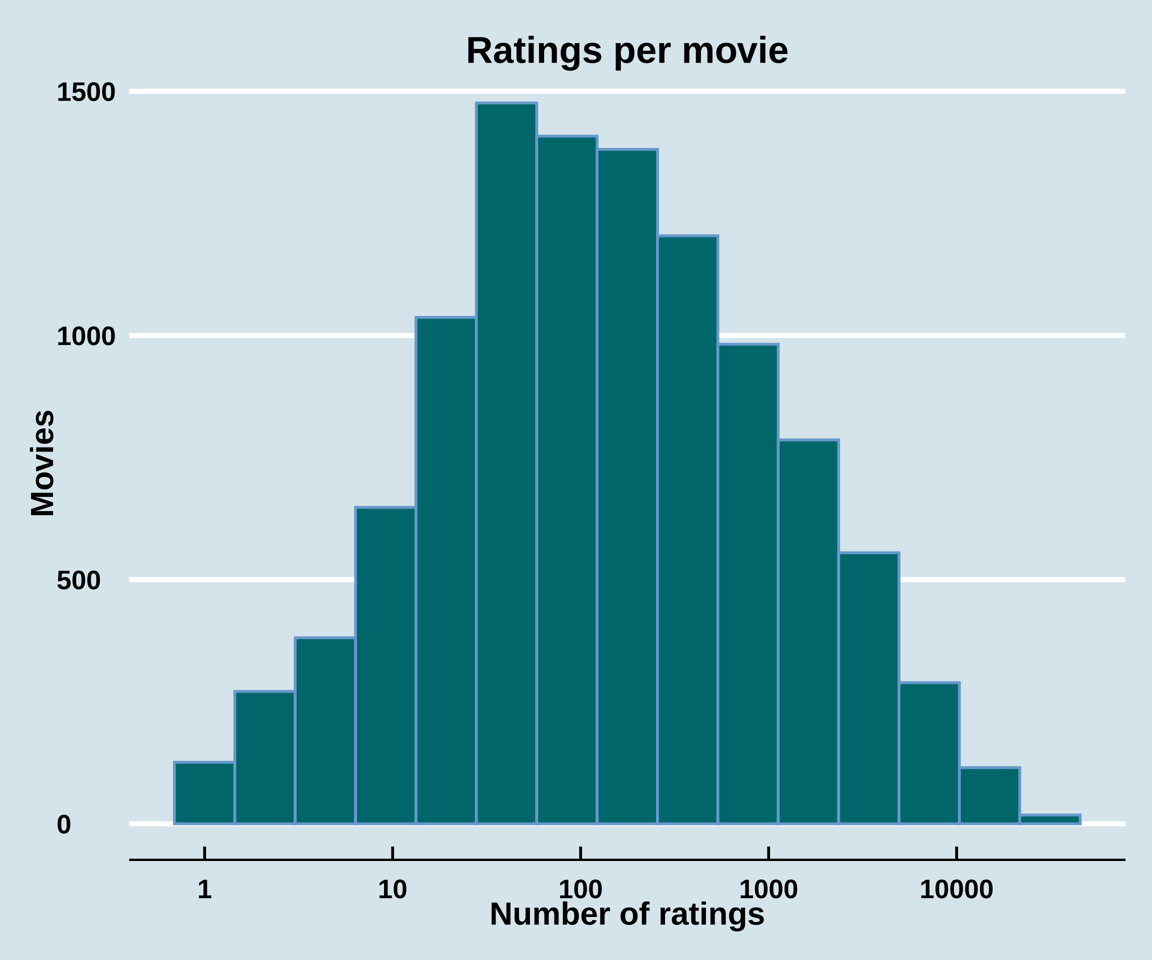
There can be many different biases affecting movie ratings, however, on this project it was attempted to tackle the major ones according to the variables available in the data set.

Table 1: Biases description

Bias/effect	Description
Movie bias	Movies might have extremely high or low ratings, and in general some movies have higher ratings than others
User bias	Some users might be more critical when rating movies than others
Genre bias	The movie genre is significant because some movies might be more popular or considered more consequential and thus receive higher ratings than others
Year bias	The release year of the movies is relevant because audiences perception and judgment changes over time

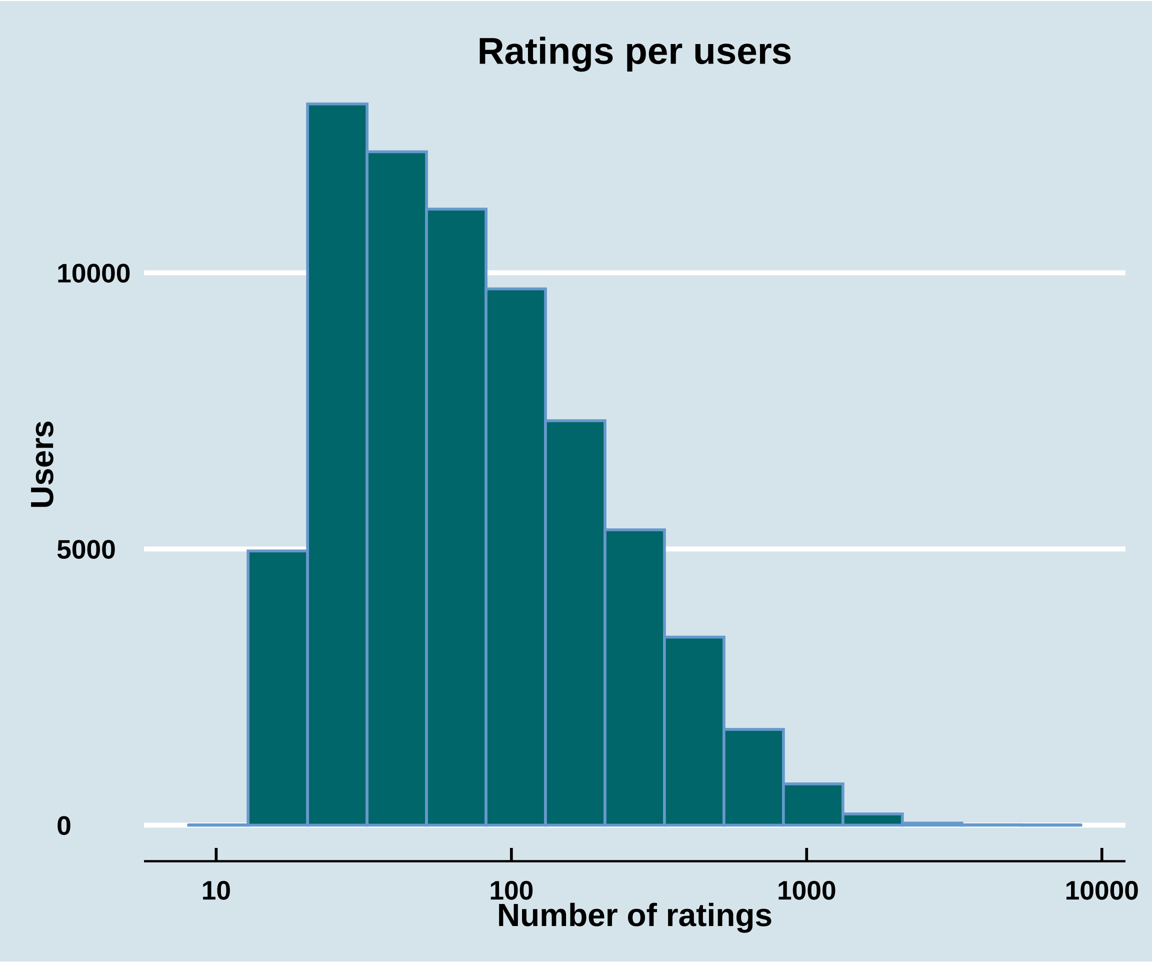
Data visualization was performed to observe how the variables of interest could introduce bias into the model and thus justify their incorporation into the model.

The number of ratings per movie distribution was plotted below. It was observed that the distribution is approximately normal and a large portion of the movies are rated between 50 to 500 times. There are a few extremely low values, which effectively could add a disproportionate effect to movie rating prediction.



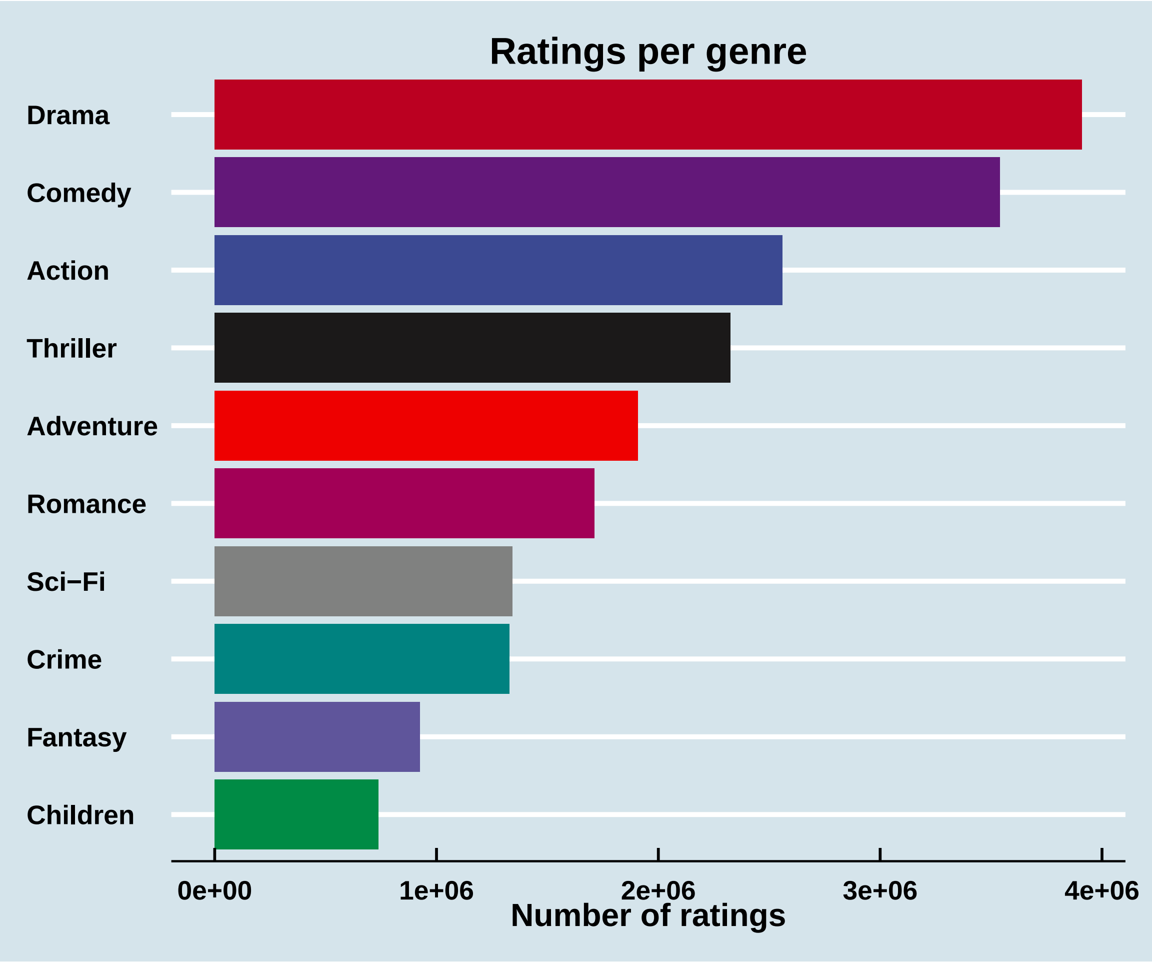
Many recommender systems focus on the content items rather than focusing on the users, although both are important factors that may influence rating prediction.

The number of movie ratings given per user plot below showed that some users are more active than others and the ratings of those less active may bias the prediction.



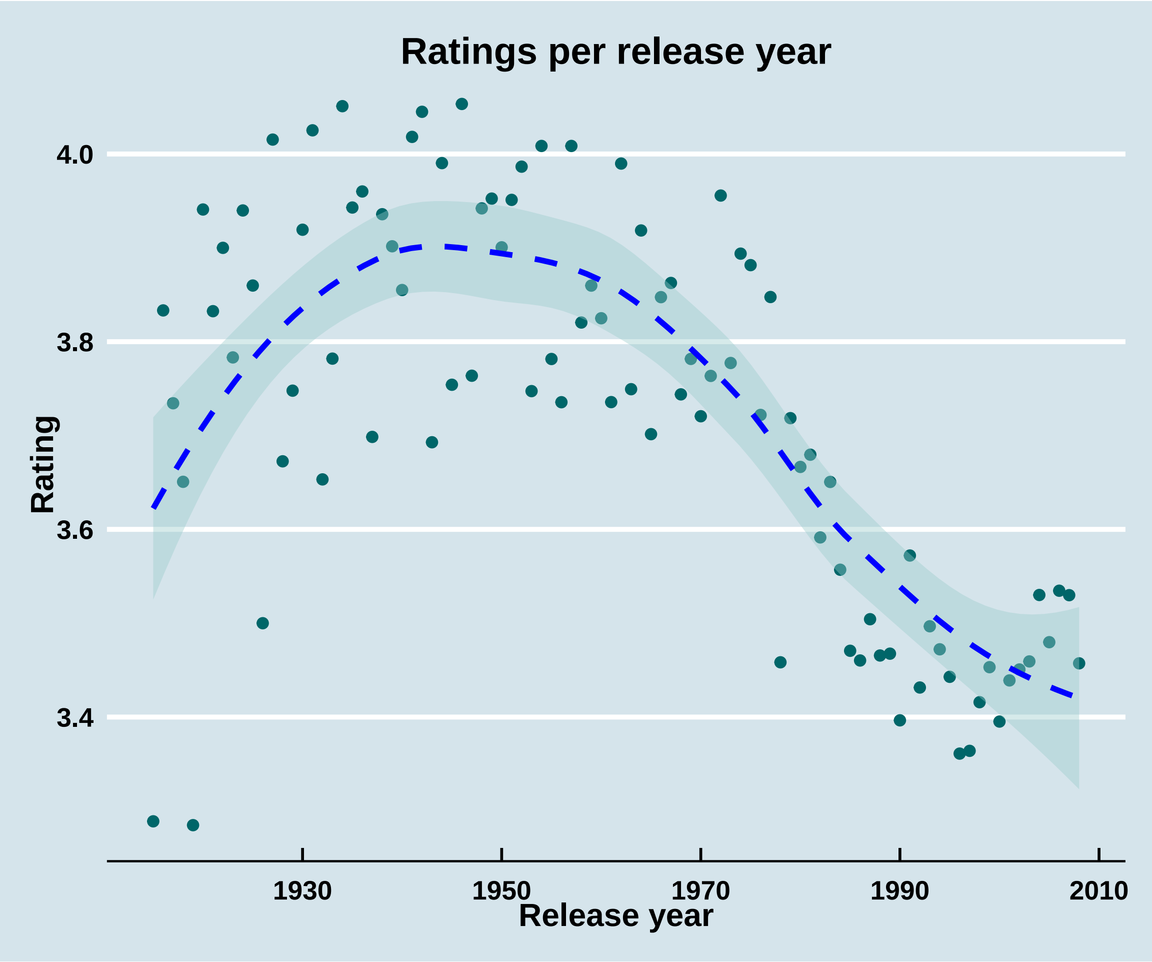
Including this variable made sense since several recommender systems rely on user behaviour to make predictions. However, each type of system has its strengths and weaknesses, in some ocassions a large amount of information about a user is needed to make accurate recommendations.

The top ten most rated movie genres plotted below demonstrated that their popularity (number of ratings) is variable and that the genre may also introduce bias to the model.



The movie genre is relevant as it is provided by movie experts and directors, so an argument could be made that it is even more reliable than user ratings.

Rating given per movie release year was plotted below. It was observed that, in general, audiences have become more critical by giving lower ratings to movies, which could also have an effect on prediction.



The model building and evaluation performed afterwards helped to elucidate whether these variables really had an effect on the movie rating prediction and how significant it was.

Data wrangling

While the movieId and userId variables were ready for analysis, it was necessary to separate the movie genres from each other and also extract the movie release year from the *title* column. A new column, *releaseYear* was created to store the year variable. Functions shown below were created for this purpose.

```
# Function to extract the year variable
year_sep <- function(z){

  z <- z %>% mutate(releaseYear = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
                                                         regex("\\d{4}"))),
                    title = str_remove(title, "[/(]\\d{4}[/)]$"))
}

edx <- year_sep(edx)  # the function was applied to the edx and validation sets
validation <- year_sep(validation)
```

```
# Function to separate the movie genres
genre_sep <- function(x){

  x <- x %>% separate_rows(genres, sep = "\\|")
}

edx <- genre_sep(edx)  # the function was applied to the edx and validation sets
validation <- genre_sep(validation)


# The sets were transformed back to data frames to facilitate analysis
edx <- as.data.frame(edx)
validation <- as.data.frame(validation)
```

The head of the new data set ready for analysis is shown below.

```
head(edx)
```

```
##   userId movieId rating timestamp      title   genres releaseYear
## 1      1     122      5 838985046 Boomerang    Comedy        1992
## 2      1     122      5 838985046 Boomerang   Romance        1992
## 3      1     185      5 838983525  Net, The    Action        1995
## 4      1     185      5 838983525  Net, The     Crime        1995
## 5      1     185      5 838983525  Net, The  Thriller        1995
## 6      1     292      5 838983421  Outbreak    Action        1995
```

Modeling approach

The evaluation metric chosen was the root mean squared error (RMSE), a commonly used metric that indicates the absolute fit of the model to the data, that is how close the observed data points are to the model's predicted values. RMSE is a negatively-oriented score, which means that lower values are better, and it expresses average model prediction error in units of the variable of interest. The following code chunk shows a function to compute the RMSE of the models.

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

Data partition to obtain train and test sets was performed.

```r
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
test_set <- edx[test_index,]
train_set <- edx[-test_index,]

# Ensuring that the corresponding movieId and userIds are included in all sets
test_set <- test_set %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")

validation <- validation %>%
    semi_join(train_set, by = "movieId") %>%
    semi_join(train_set, by = "userId")
```

In machine learning, a naive model is a simple classification model that assumes little to nothing about the problem and the performance of which provides a baseline by which all other models evaluated on a data set can be compared.

A naive model was computed for this issue (shown below), it simply predicted the same rating for all movies regardless of which movie or user.

```r
mu <- mean(train_set$rating)  # just the average

result_mu <- RMSE(test_set$rating, mu)
print(result_mu)  # result was computed and saved
```

```
## [1] 1.051984
```

As expected, its RMSE value is quite high, a little over 1 star away from the real rating, this certainly must be improved to a more accurate prediction.

For a first model, the movie bias was taken into account by computing the estimated deviation of the movies' mean rating from the global mean rating (shown below). The same approach was taken with all the following effects considered to build the model.

```r
mu <- mean(train_set$rating)
movie_effect <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu))  # added movie effect

model_1 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  mutate(pred = mu + b_m) %>%
  pull(pred)

result_1 <- RMSE(test_set$rating, model_1)
print(result_1)  # result was computed and saved
```

```
## [1] 0.9409964
```

To further improve the model's RMSE, the user bias was accounted for along with the movie effect as shown in the following code chunk.

```r
user_effect <- train_set %>%
  left_join(movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_m))  # added user effect

model_2 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  mutate(pred = mu + b_m + b_u) %>%
  pull(pred)

result_2 <- RMSE(test_set$rating, model_2)
print(result_2)  # result was computed and saved
```

```
## [1] 0.8574998
```

To continue improving the model, the movie genre was accounted for along with the movie and user effect.

```r
genre_effect <- train_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_m - b_u))  # added genre effect

model_3 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(genre_effect, by="genres") %>%
  mutate(pred = mu + b_m + b_u + b_g) %>%
  pull(pred)

result_3 <- RMSE(test_set$rating, model_3)
print(result_3)  # result was computed and saved
```

```
## [1] 0.8574106
```

Finally, as shown in the following code chunk, the movie release year bias was added to the model along with the movie, user and genre bias model.

```r
year_effect <- train_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(genre_effect, by="genres") %>%
  group_by(releaseYear) %>%
  summarize(b_y = mean(rating - mu - b_m - b_u - b_g))  # added release year effect

model_4 <- test_set %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(genre_effect, by="genres") %>%
  left_join(year_effect, by="releaseYear") %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
  pull(pred)

result_4 <- RMSE(test_set$rating, model_4)
print(result_4)  # result was computed and saved
```

```
## [1] 0.8570634
```

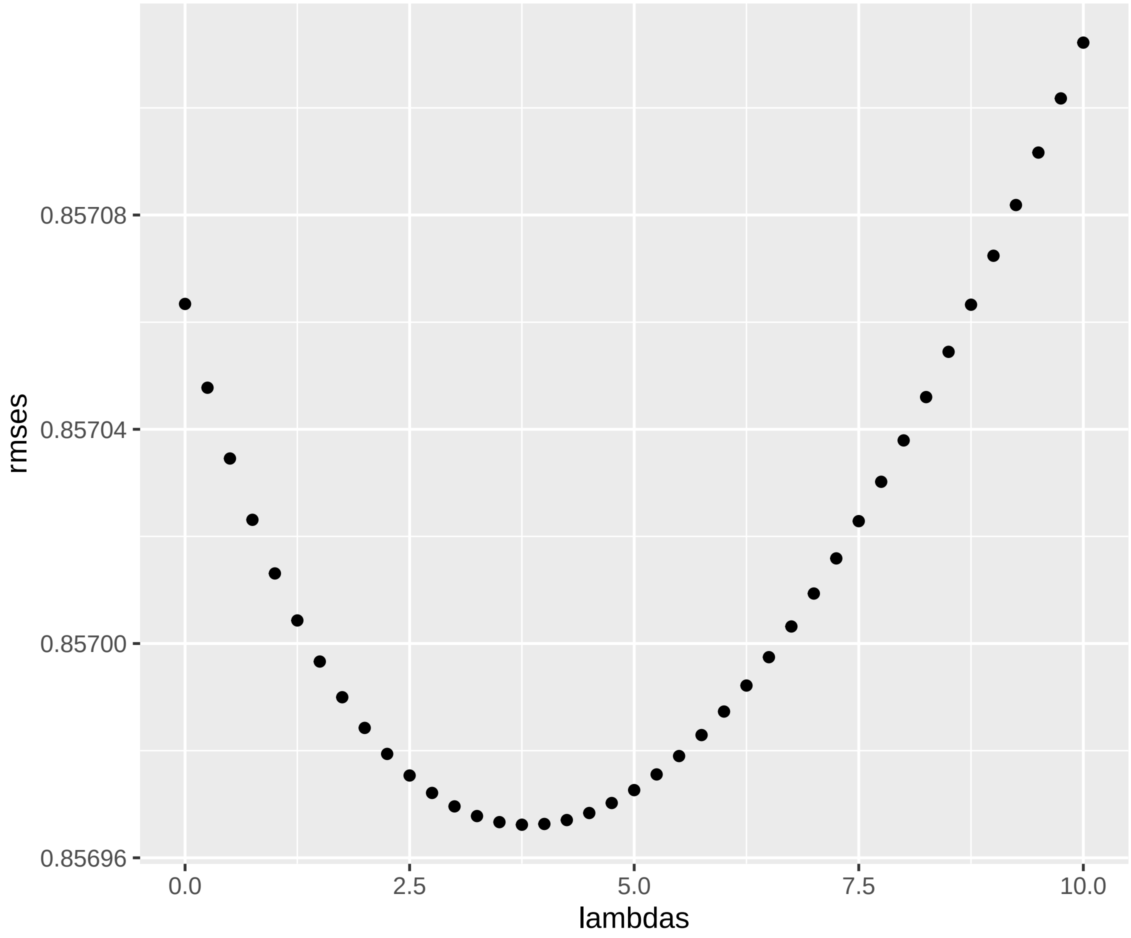
To further improve the model, regularization was performed. Regularization is the process of adding information (a regularization term or penalty) to impose a cost on the optimization function to prevent overfitting and finding an optimal solution. In this case, regularization permitted to penalize the possible unequal aspects of the variables taken into account for the model (e.g. movies with very few ratings) which means it penalized large or noisy estimates that came from small sample sizes.

In the following chunk code, a function was created to perform regularization on the test set first while simultaneously performing cross validation to obtain the optimal *lambda* value, which is a tuning parameter that determines the degree of regularization and is key in reducing the model's RMSE.

```
lambdas <- seq(0, 10, 0.25) # tuning parameter range

mu <- mean(train_set$rating)

rmses <- sapply(lambdas, function(l){  # function to perform cross validation

  b_m <- train_set %>%  # adjust mean by penalizing movie effect
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+l))

  b_u <- train_set %>%  # adjust mean by penalizing movie and user effect
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_m - mu)/(n()+l))

  b_g <- train_set %>%  # adjust mean by penalizing movie, user and genre effect
    left_join(b_m, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_m - b_u - mu)/(n()+l))

  b_y <- train_set %>%  # adjust mean by penalizing movie, user, genre and year effect
    left_join(b_m, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(releaseYear) %>%
    summarize(b_y = sum(rating - b_m - b_u - b_g - mu)/(n()+l))

  predicted_ratings <- test_set %>%  # computed predictions
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "releaseYear") %>%
    mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
    pull(pred)

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

A plot of the lambdas vs RMSEs values was computed.

```r
qplot(lambdas, rmses)
```



The exact optimal lambda value was obtained through its index.

```r
lambda <- lambdas[which.min(rmses)]  # optimal lambda value
print(lambda)
```

```
## [1] 3.75
```

The regularization test and cross validation result was computed and saved.

```r
result_reg <- min(rmses)
print(result_reg) # cross validation test result was computed and saved
```

```
## [1] 0.8569662
```

The final hold-out test set (validation) using the optimal lambda value obtained was performed (shown below).

```
lambda <- 3.75 # optimal lambda value obtained in previous cross validation

b_m <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = sum(rating - mu)/(n()+lambda))

b_u <- train_set %>%
  left_join(b_m, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_m - mu)/(n()+lambda))

b_g <- train_set %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_m - b_u - mu)/(n()+lambda))

b_y <- train_set %>%
  left_join(b_m, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(releaseYear) %>%
  summarize(b_y = sum(rating - b_m - b_u - b_g - mu)/(n()+lambda))

predicted_ratings <- validation %>%  # only call for validation set, final test
  left_join(b_m, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_y, by = "releaseYear") %>%
  mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
  pull(pred)

result_5 <- RMSE(validation$rating, predicted_ratings)
print(result_5) # final model result was computed and saved
```

```
## [1] 0.8629447
```

Finally, the random forest algorithm was fit to data to test another common machine learning approach. Random forest is a supervised learning algorithm that builds an ensemble of regression or classification trees, usually trained with the bagging method. The general idea of the bagging method is that a combination of learning models increases the overall result, in this case, the multiple trees obtained are merged to get a more accurate and stable prediction. The *randomForest* package was used.

A smaller random sample subset of the data set (length of 10 thousand) was used, this approach was also attempted with the original data set as well as with larger subsets but it was simply impossible to compute given the RAM capacity of the equipment used.

A random sample without replacement of the edx data set was created.

```
mysample <- edx[sample(1:nrow(edx), 10000,  # a random sample sample of length 10 thousand...
                       replace=FALSE),]     #... is taken from the edx set
```

A data partition of the sample was performed to create pertinent train and test sets for this analysis.

```
set.seed(1)
test_index <- createDataPartition(y = mysample$rating, times = 1, p = 0.2,
                                  list = FALSE)  # data partition
test_rf <- mysample[test_index,]
train_rf <-mysample[-test_index,]

test_rf <- test_rf %>%
  semi_join(train_rf, by = "movieId") %>% # ensuring that the corresponding movieId and ...
  semi_join(train_rf, by = "userId")      # ... userIds are included in the test set
```

The *randomForest* function was used with the regression method as only the movieId and usedId variables were considered for prediction and the model's RMSE was computed.

```
rf_model <- randomForest(rating ~ movieId + userId,  # random forest fit
                           data = train_rf,
                           type = "regression",
                           proximity = TRUE)

rf_pred <- predict(rf_model, test_rf)
result_rf <- RMSE(test_rf$rating, rf_pred)
print(result_rf)  # result was computed and saved
```

```
## [1] 1.047713
```

Results

Model	RMSE
Naive model	1.0519843
Movie effect	0.9409964
Movie + user effect	0.8574998
Movie + user + genre effect	0.8574106
Movie + user + genre + year effect	0.8570634
Regularized movie + user + genre + year cv test	0.8569662
Regularized movie + user + genre + year validation final	0.8629447
Random forest subset test	1.0477127

The results showed that the movie and user effects greatly improved the RMSE compared to the naive model. The genre and year effects, however, only had a slight impact. The random forest attempt only slightly improved the RMSE of the naive model. Given the size of the data set and its multivariate nature, fitting other models would be a more extensive process and likely inefficient in some cases. It was also observed that its variable importance was consistent with the one observed in the algorithm that was built, the movie and user variables were practically equally important for prediction. The final model generated with all the included biases plus regularization successfully predicted movie ratings in the **validation set**, achieving **a final RMSE of 0.8629447**.

(Gorakala et al., 2015) (Ricci et al., 2011) (Irrizary, 2019) (Kane, 2018)

Conclusion

This project's goal was quite a challenge, movie recommender systems are difficult because movie ratings can be influenced by several biases including subjective and social ones. The assignment of this project strongly established frequent machine learning issues like overfitting and the tradeoff between prediction accuracy and computational cost. Usually common machine learning algorithms are used for this problem, however, the obstacle in this case was the large size of the data set. Therefore, the real task at hand was to build a suitable algorithm that could break the data set into less extensive and more manageable subsets and rejoin them to create a model that mitigated the major biases. The final model generated on this project successfully predicted movie ratings in the validation set. The approach taken on this project was in accordance with the hardware and expertise limitations, however, further improvement could be achieved through more extensive model trial and error, data wrangling, different machine learning models fitting and other ensemble techniques.

References

Gorakala K and usuelli M. 2015. Building a recommendation system with R. Packt Publishing.

Ricci F, Rokach L, Shapira B, Kantor P.B. 2011. Recommender Systems Handbook. Springer.

Kane F. 2018. Building recommender systems with Machine Learning and AI. DBA Sundog Education.

Irizarry R.A. 2019. Introduction to data science. CRC Press.

Appendix

```
# Environment
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.3
## year           2020
## month          10
## day            10
## svn rev        79318
## language       R
## version.string R version 4.0.3 (2020-10-10)
## nickname       Bunny-Wunnies Freak Out
```