Data Science: Capstone Report

Louri Compain

2023-08-21

Introduction

The project

This document is my report for the Data Science : Capstone course on EDX. The goal of the project is to use a subset of the MovieLens dataset to develop a prediction of a movie's rates.

For reproducibility's sake, we will set the seed at 1, and download our libraries in a single place at the start of the project.

```
set.seed(1)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(caret)
```

```
## Le chargement a nécessité le package : lattice
##
## Attachement du package : 'caret'
##
## L'objet suivant est masqué depuis 'package:purrr':
##
##     lift
```

```
library(caTools)
library(class)
library(rpart)
library(e1071)
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
```

```
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
##
##
## Attachement du package : 'h2o'
##
## Les objets suivants sont masqués depuis 'package:lubridate':
##
##     day, hour, month, week, year
##
## Les objets suivants sont masqués depuis 'package:stats':
##
##     cor, sd, var
##
## Les objets suivants sont masqués depuis 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

Dataset

The first step was to prepare our environment by dowloading the libaries and reproducing the data for the exercise. Here is the code.

```
options(timeout = 120)
ratings_file <- "ml-10M100K/ratings.dat"
movies_file <- "ml-10M100K/movies.dat"


ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
```

```r
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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

```
## Warning in rm(dl, ratings, movies, test_index, temp, movielens, removed): objet
## 'dl' introuvable
```

This produces the two datasets we are working with, named edx and final_holdout_test. These were produced by separating the MovieLens Dataset into two. We will be using edx as our main training set, and final_holdout_test.will be used in the last step in order to calculate the RMSE.

```r
head(edx)
```

```
##   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
```

Here, we can see that EDX contains the following variables:

- user Id and movieId, to identify the user and movie
- rating, the variable we want to predict in the end
- the timestamp of the review
- the title of the film, with the date it was aired between parenthesis
- the list of its genre, as a single column, separated by vertical bars

The title itself is of little use (good title choice might impact the ratings, but using the rules that govern this phenomena, if it exists at all, is quite ambitious). The date however, could help us. Films of a certain time period might receive more love from the reviewers, or maybe a certain genre might have had a "golden age" at some point.

The genre is an interesting variable, because it is the only one that tells us what is in the film proper. Its format is quite inconvenient, as it is a string that contains several pieces of information. It will need some transformations to become useful.

The timestamp is not about the movie itself, but there is a chance that we can use it. Maybe there are trends in reviews over time.

The movieId and userId might seem like random numbers, attributed to protect the identity of reviewers, or to connect multiple reviews on the same film without dragging a clunky string variable containing the title. But we might have a chance to exploit it. Since we made sure all userId and movieId in final hold-out test set are also in edx set, we might be able to deduce information about them, maybe even the mindset of the person behind a certain Id.

The rating is an ordinal variable, with categories that have an order (they are actually non-continous numerical values). They are ranging from 0.5 to 5 with a 0.5 increment, giving a a total of 9 possible categories.

Project goals

The main goal of this project is to create a model that can predict the ratings of movie reviews from the MovieLens dataset, and to optimise it to reduce its RMSE.

We want to aim for an RMS lower than 0.9, and preferably bellow 0.86490.

Report plan

We will work using the following process:

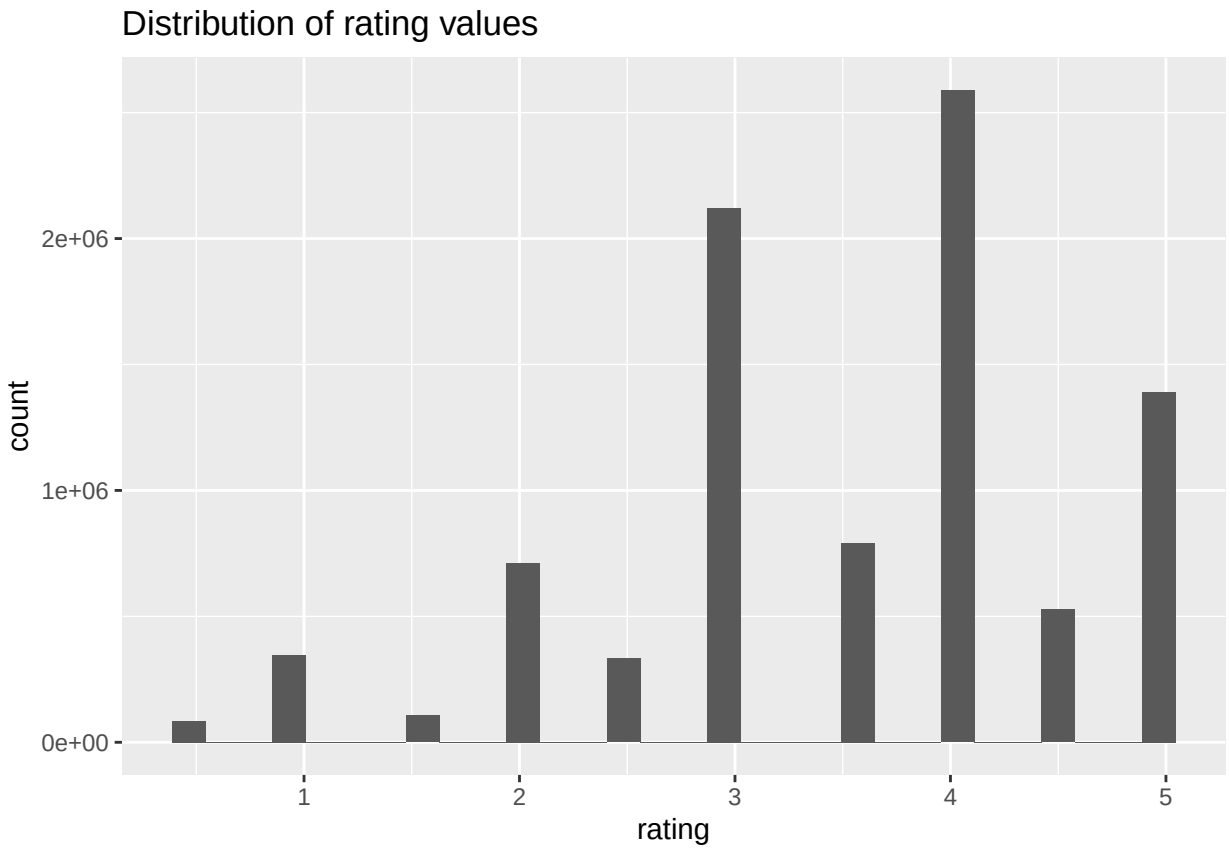
- Analyse the data
- Exctract additional information from the dataset
- Analyse the new information from the dataset
- Implement small scale models (naive prediction, linear model, treemap, naive bayes and neural network)
- Chose the most promising small scale model
- Improve our candidate
- Test it on the full scale dataset
- Implement on the final_holdout_test data # Methodology and analysis

Initial Analysis

Let's see how the ratings are distributed

```
ggplot(edx, aes(x=rating)) + geom_histogram() + ggtitle("Distribution of rating values")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can see that reviewers prefer round rating rather than giving "half-stars". There also appears to be a skew toward high values.

We can also count the number of users and movies.

```
n_distinct(edx$movieId)
```

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

```
n_distinct(edx$userId)
```

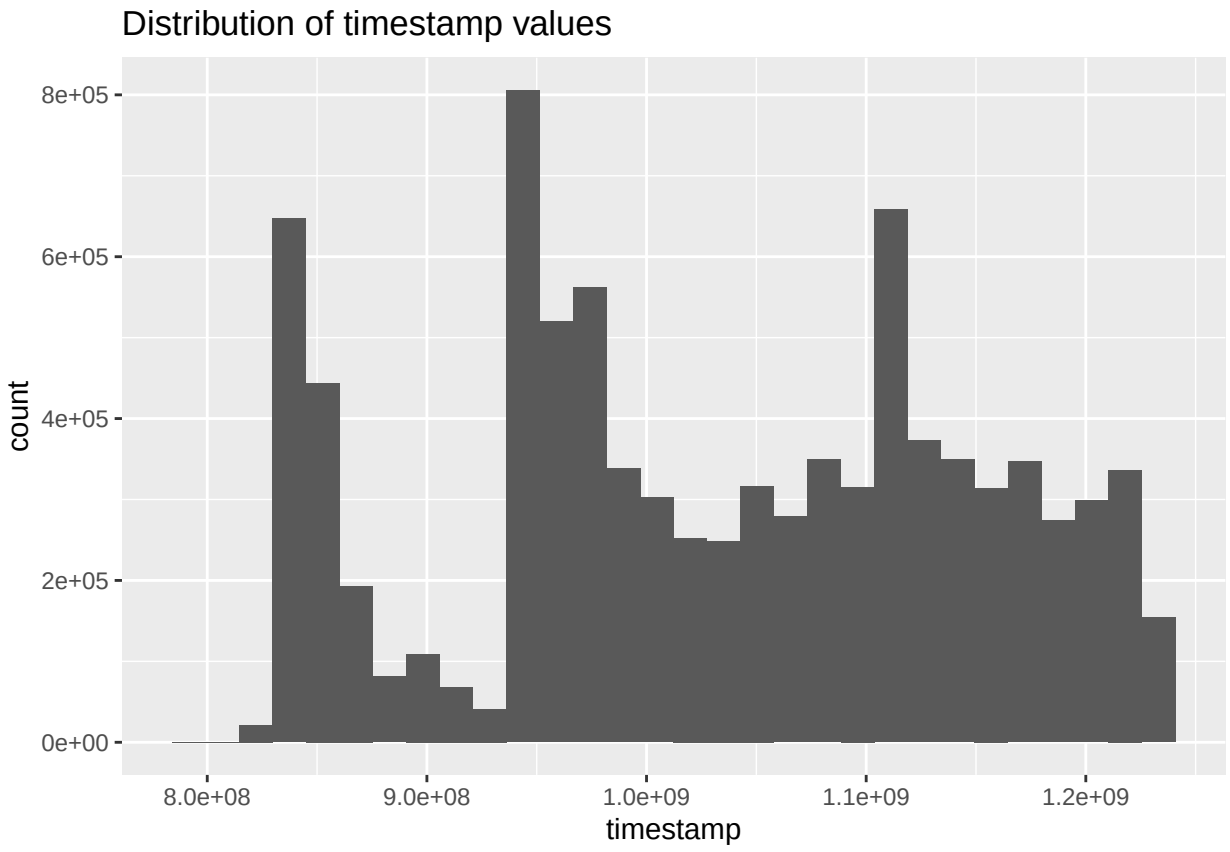
```
## [1] 69878
```

We can see a lot of users and movies. This means that we will have difficulties "profiling" specific users and movies based n their ID to gather information.

We can also look at the timestamps to see when the movies were reviewed. Due to the values of timestamps however, this will be difficult to evaluate. According to the movieLens documentation, the timestamps "represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970" (https://files.grouplens.org/datasets/movielens/ml-10m-README.html)

```
ggplot(edx, aes(x=timestamp)) + geom_histogram() + ggtitle("Distribution of timestamp values")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Dataset preparation

First we want to transform our dataset to make the most of it. It will consist of extraction and scaling.

Information extraction

There are two pieces information that might be relevant but are not directly accessible: the date and genre. We will then transform the dataset to extract these information and put them in a better format.

First, we will need to get the dates out of the title and into their own column.

```
#date extraction based on https://www.kaggle.com/code/redroy44/movielens-dataset-analysis
# use regex to get the date from the title
dat_train_1 <- edx %>%
  mutate(title = str_trim(title)) %>%
  extract(title, c("title_tmp", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F) %>%
  mutate(year = if_else(str_length(year) > 4, as.integer(str_split(year, "-", simplify = T)[1]), as.int
  mutate(title = if_else(is.na(title_tmp), title, title_tmp)) %>%
  select(-title_tmp)  %>%
  mutate(genres = if_else(genres == "(no genres listed)", `is.na<-`(genres), genres))
```

Then, we extract the different values of the genre in the form of multiple binary dummy variables. One for each possible genre.

```
# List of genres based on https://rpubs.com/outerelocarlos/MovieLens-Recommender-System and the https:/
genres <- c("Action", "Adventure", "Animation",
            "Children", "Comedy", "Crime",
            "Documentary", "Drama", "Fantasy",
```

```
            "Film-Noir", "Horror", "Musical",
            "Mystery", "Romance", "Sci-Fi",
            "Thriller", "War", "Western")

for (genre_index in 1:length(genres)){
  dat_train_1[genres[genre_index]] <- NA
  detection <-as.integer(
    str_detect(
      dat_train_1$genre,
      genres[genre_index]
      )
    )
  detection[is.na(detection)] <- 0
  dat_train_1[genres[genre_index]] <- detection
}

#rename Film-Noir and Sci-Fi to avoid issues later on with the "-"
names(dat_train_1)[names(dat_train_1) == "Film-Noir"] <- "Film_Noir"
names(dat_train_1)[names(dat_train_1) == "Sci-Fi"] <- "Sci_Fi"
```

Once these modifications have been applied, we can remove the title and the original genre columns, as they are now superfluous.

```
dat_train_2 <- dat_train_1[,c(-5, -7)]
```

We will then try to get more informations about the movies and users. For this, we can chose several methods. A tempting option might be to treat movieId and userId as categorical variables and to create a binary dummy variable for each user and movie. This option is technically possible but would add thousands of factors to our later models, rendering them unusable. Instead, we can elect to search for a deviation from the average rating based on the movie ID and the user ID.

```
avg <- mean(dat_train_2$rating)
#computing the bias
movie_avgs <- dat_train_2 %>%
  group_by(movieId) %>%
  summarize(bias_movie = mean(rating - avg))
user_avgs<- dat_train_2 %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(bias_user = mean(rating - avg - bias_movie))
```

Once the two set of bias have been created, we merge them into the training data and remove the numerical values of userId and movieId, wich pollute our dataset with meaningless numerical variables.

```
#adding the bias to the main dataset
dat_train_3 <- dat_train_2%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = avg + bias_movie + bias_user)

dat_train_3 <- dat_train_3[,c(-1,-2)]
```

Scaling

Now that e have extracted information from our dataset, we can scale it, to improve the performance of future models.

```r
#create the scae
scale <- preProcess(as.data.frame(dat_train_3[,c(-1)]), method=c("range"))
#apply it
dat_train_4 <- predict(scale, as.data.frame(dat_train_3))
```

As a result, we obtain dat_train_4, the training dataset we will be using for the rest of the project.

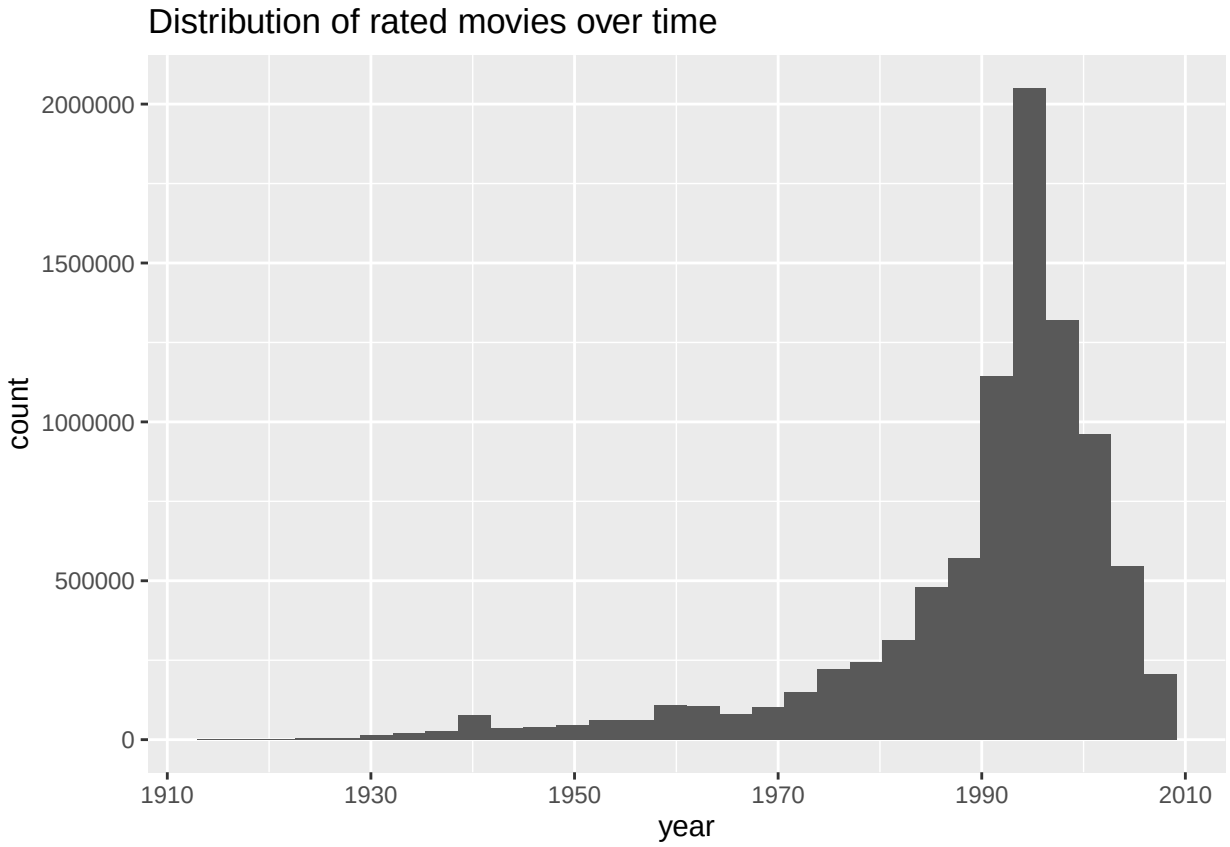
Post-preparation Analysis

Now that we adapted the dataset to our needs, we can analyse it further.

First, now that we have access to the dates we can plot them. Note that we use dat_train_3 since dat_train_4 is scaled.

```r
ggplot(dat_train_3, aes(x=year)) + geom_histogram()  + ggtitle("Distribution of rated movies over time")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

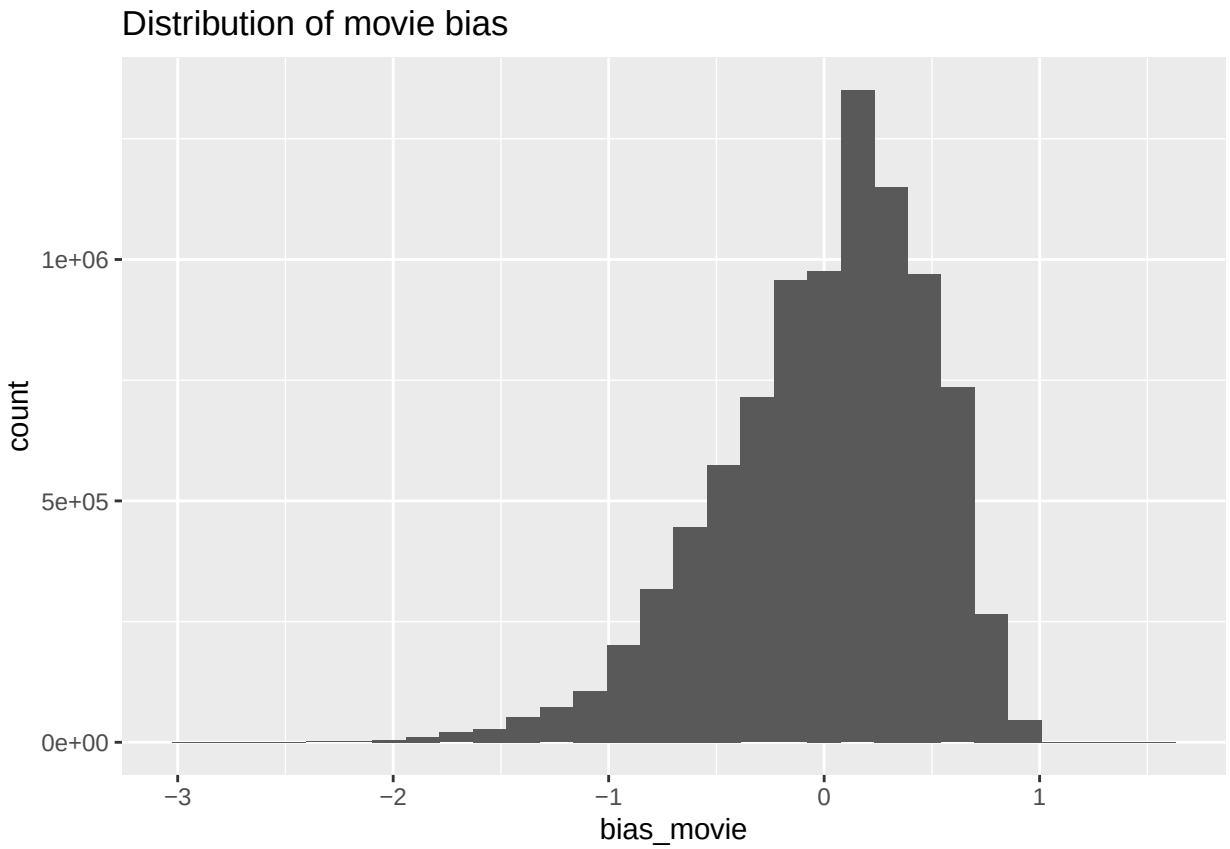


It appears that most films in the database are movies from the nineties. The data collection starts with old films, up to the late 1910s, reaches a peak in the end of the twentieth century, and drops until the data collection is cut in 2010. Older movies might have an advatage due to having more time to gain appreciation and receive reviews.

We cam also plot the movie and user bias that we computed.

```r
ggplot(dat_train_3, aes(x=bias_movie)) + geom_histogram() + ggtitle("Distribution of movie bias")
```

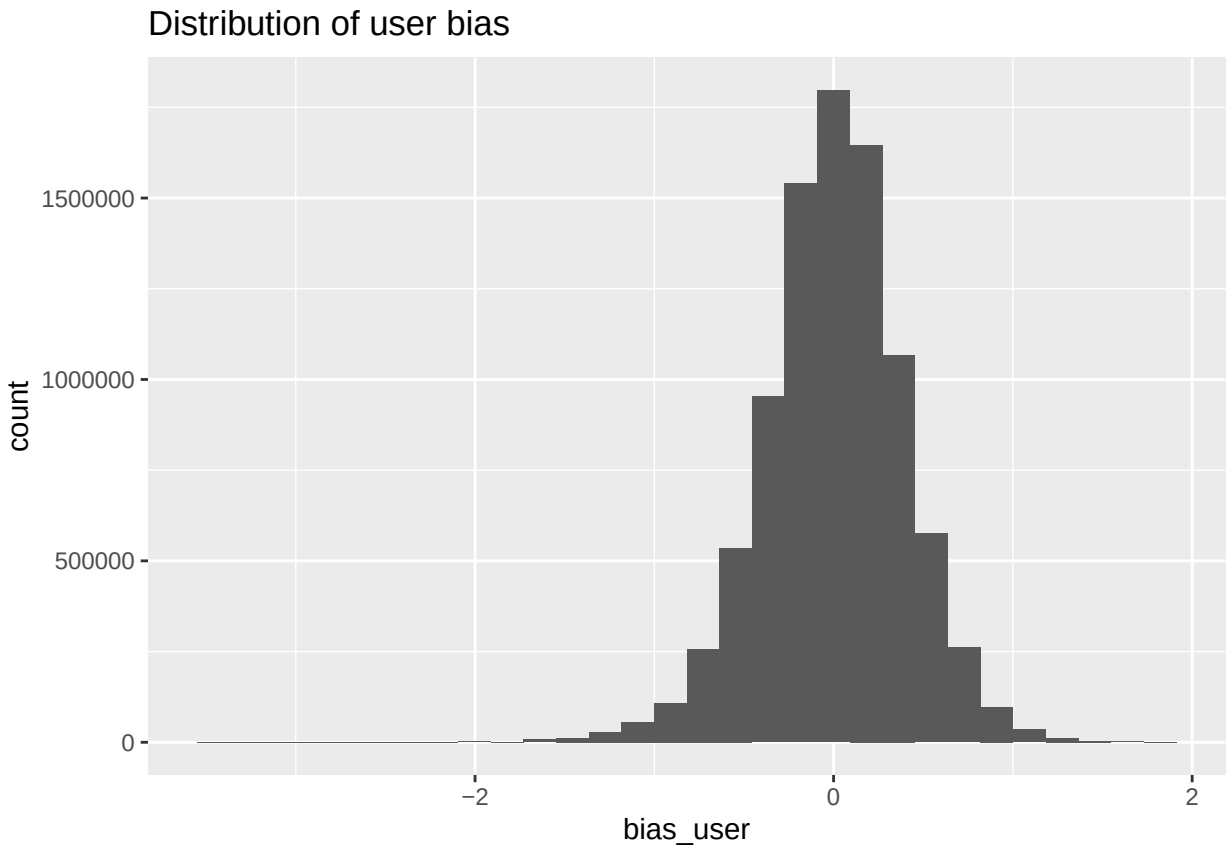
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



For the movie bias, we can observe that most movies have a bias between -1 and 1. This variation can be quite significant on a 5 stars scales. Some rare movies have bias that drop to nearly -2.5.

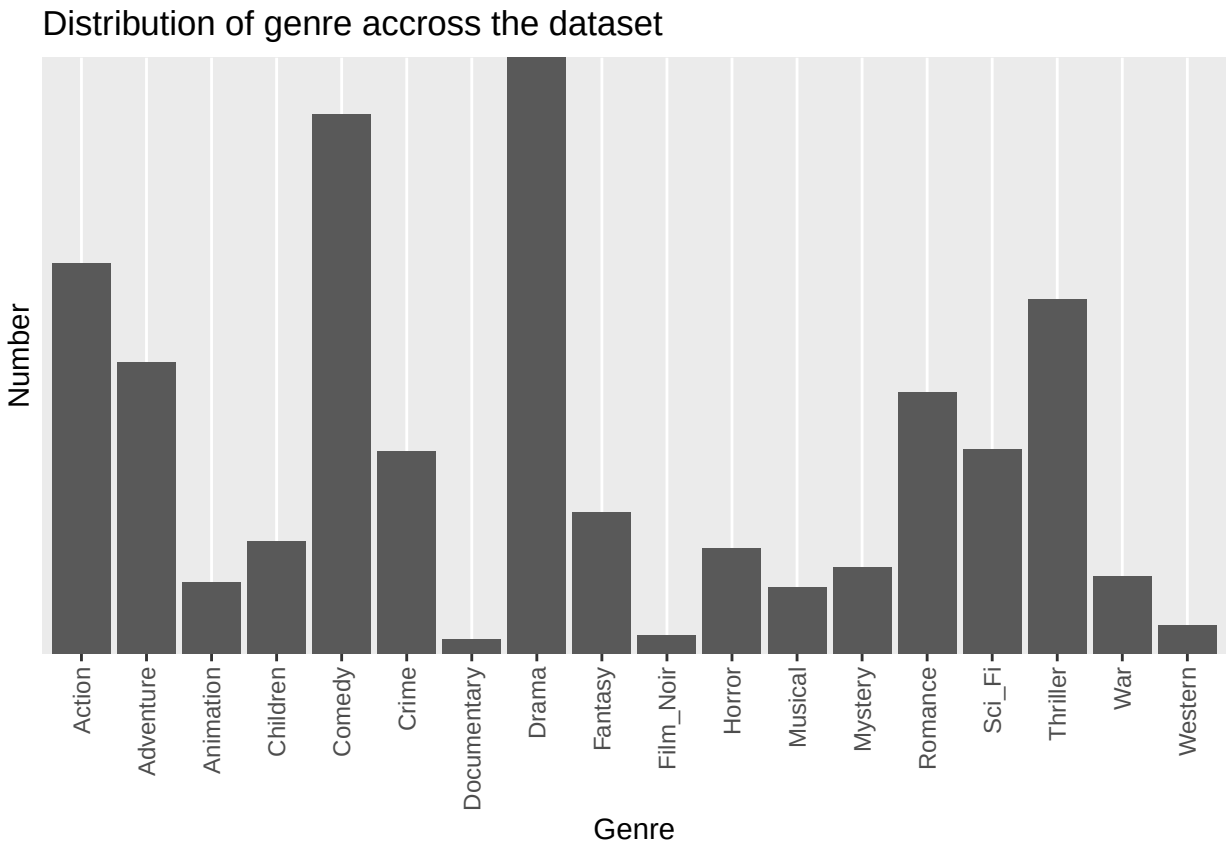
```
ggplot(dat_train_3, aes(x=bias_user)) + geom_histogram() + ggtitle("Distribution of user bias")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



For the user bias, we can see that users that reach a bias of 1 or -1 are quite rare, but that the extremes push to 2 and -2.

```
#change the genre to reflect the change from "-" to "_"
modified_genres <- c("Action", "Adventure", "Animation",
            "Children", "Comedy", "Crime",
            "Documentary", "Drama", "Fantasy",
            "Film_Noir", "Horror", "Musical",
            "Mystery", "Romance", "Sci_Fi",
            "Thriller", "War", "Western")
#count number of films with each genre
genre_counter <- data.frame(modified_genres)
genre_counter$counter <- NA
for (genre_index in 1:length(modified_genres)){
  genre_counter$counter[genre_index] <-sum(dat_train_4[modified_genres[genre_index]])
}
# display
ggplot(genre_counter, aes(x = modified_genres, y = counter)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  ggtitle("Distribution of genre accross the dataset") +
  scale_x_discrete(name ="Genre") +
  scale_y_discrete(name ="Number")
```



We can see that some genre are a lot more frequent than others.

Model

We will now test and implement models to try to fullfill our goal of predicting ratings.

Small scale tests

We create a smaller data set. It will be used to test different models and compare their performance and the time they need to train before we chose one.

```
reduction_index <- createDataPartition(
  y = dat_train_4$rating,
  times = 1,
  p = 0.01,
  list = FALSE
  )
reduced_set <- dat_train_4[reduction_index,]
print(nrow(reduced_set))
```

```
## [1] 90002
```

This set is considerably smaller, with 90002 entries instead of 9000055 and will allow us to test models and detect models that might take hours on the full set.

We then separate it between train and test sets, because we cannot use the final_holdout_test data before the end of the project.

```
train_test_index <- createDataPartition(y = reduced_set$rating, times = 1, p = 0.2, list = FALSE)
reduced_dat_train <- reduced_set[-train_test_index,]
reduced_dat_test <- reduced_set[train_test_index,]
reduced_prediction_results <- reduced_dat_test[,1]
reduced_dat_test<- reduced_dat_test[,-1]
```

Naive prediction

We start trying to predict our results with the simplest approach: averaging the ratings, and using that single value as a prediction for every review.

```
avg <- mean(reduced_dat_train$rating)
RMSE_AVG <- RMSE(reduced_prediction_results, avg)

RMSE_AVG
```

```
## [1] 1.062822
```

We get an RMSE of 1.055387. This is our baseline, and the value we will be trying to beat with our other models. It is insufficient when compared to our RMSE goal, so we will have to improve.

Linear Model

Our next model is a linear model. We use the lm function to produce a regression, and then round to the nearest value that match a possible category (0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5).

```
#create model
linear_regression_model_1 <- lm(
  rating ~ timestamp + year + Action + Adventure + Animation +
  Children + Comedy +Crime + Documentary + Drama + Fantasy +
  Film_Noir + Horror + Musical + Mystery + Romance + Sci_Fi +
  Thriller + War + Western + bias_movie + bias_user + pred,
  data = reduced_dat_train)
summary(linear_regression_model_1)
```

```
##
## Call:
## lm(formula = rating ~ timestamp + year + Action + Adventure +
##     Animation + Children + Comedy + Crime + Documentary + Drama +
##     Fantasy + Film_Noir + Horror + Musical + Mystery + Romance +
##     Sci_Fi + Thriller + War + Western + bias_movie + bias_user +
##     pred, data = reduced_dat_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2603 -0.4838  0.0612  0.5701  4.1033
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.8808123  0.0440421 -65.410  < 2e-16 ***
## timestamp   -0.0094040  0.0124037  -0.758  0.44836
## year        -0.0277252  0.0243368  -1.139  0.25461
## Action      -0.0142040  0.0090593  -1.568  0.11691
## Adventure   -0.0013889  0.0093462  -0.149  0.88186
## Animation    0.0088934  0.0198805   0.447  0.65463
## Children    -0.0377141  0.0169413  -2.226  0.02601 *
```

```
## Comedy       0.0001336  0.0080976   0.016  0.98684
## Crime        0.0015072  0.0100361   0.150  0.88063
## Documentary  0.0904764  0.0328138   2.757  0.00583 **
## Drama        0.0161053  0.0081607   1.974  0.04844 *
## Fantasy      0.0036885  0.0115183   0.320  0.74879
## Film_Noir   -0.0246954  0.0293856  -0.840  0.40069
## Horror       0.0184561  0.0129650   1.424  0.15459
## Musical      0.0106183  0.0168261   0.631  0.52800
## Mystery      0.0054220  0.0140880   0.385  0.70034
## Romance     -0.0108939  0.0086693  -1.257  0.20890
## Sci_Fi      -0.0140165  0.0100292  -1.398  0.16225
## Thriller    -0.0078947  0.0088389  -0.893  0.37176
## War         -0.0264734  0.0145053  -1.825  0.06799 .
## Western     -0.0306548  0.0222860  -1.376  0.16898
## bias_movie   4.5811173  0.0336571 136.111  < 2e-16 ***
## bias_user    5.2325222  0.0433473 120.712  < 2e-16 ***
## pred                NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8543 on 71977 degrees of freedom
## Multiple R-squared:  0.3512, Adjusted R-squared:  0.351
## F-statistic:  1771 on 22 and 71977 DF,  p-value: < 2.2e-16
```

```
#predict values
linear_regression_prediction_1 <- predict(
  linear_regression_model_1,
  newdata = reduced_dat_test
  )
linear_regression_prediction_1 <- round(linear_regression_prediction_1*2, digits = 0)/2 # round to the

#calculate RMSE
linear_regression_RMSE_1 <- RMSE(
  linear_regression_prediction_1,
  reduced_prediction_results
  )
linear_regression_RMSE_1
```

```
## [1] 0.8670271
```

We have an RMSE of 0.8670271 This is considerably better than our baseline of 1.055387. According to our goals, it is accetable but not perfect, so we want to improve more.

Treemap

We now try to compare with a treemap.

```
#create model
rpart_model_1 <- rpart(
  rating ~ timestamp + year + Action + Adventure + Animation +
  Children + Comedy +Crime + Documentary + Drama + Fantasy +
  Film_Noir + Horror + Musical + Mystery + Romance + Sci_Fi +
  Thriller + War + Western + bias_movie + bias_user + pred
  , data = reduced_dat_train)
summary(rpart_model_1)
```

```
## Call:
## rpart(formula = rating ~ timestamp + year + Action + Adventure +
##     Animation + Children + Comedy + Crime + Documentary + Drama +
##     Fantasy + Film_Noir + Horror + Musical + Mystery + Romance +
##     Sci_Fi + Thriller + War + Western + bias_movie + bias_user +
##     pred, data = reduced_dat_train)
##   n= 72000
##
##           CP nsplit rel error    xerror        xstd
## 1 0.23223760      0 1.0000000 1.0000347 0.005282866
## 2 0.04766923      1 0.7677624 0.7685690 0.004349258
## 3 0.03922652      2 0.7200932 0.7221044 0.004203420
## 4 0.01000000      3 0.6808666 0.6832386 0.004129811
##
## Variable importance
##       pred bias_movie  bias_user     Horror       year
##         58         26         15          1          1
##
## Node number 1: 72000 observations,    complexity param=0.2322376
##   mean=3.511958, MSE=1.124482
##   left son=2 (27277 obs) right son=3 (44723 obs)
##   Primary splits:
##       pred       < 0.6511255  to the left,  improve=0.23223760, (0 missing)
##       bias_movie < 0.6467532  to the left,  improve=0.14484490, (0 missing)
##       bias_user  < 0.6222759  to the left,  improve=0.08100927, (0 missing)
##       Drama      < 0.5        to the left,  improve=0.01856434, (0 missing)
##       year       < 0.672043   to the right, improve=0.01511492, (0 missing)
##   Surrogate splits:
##       bias_movie < 0.6327697  to the left,  agree=0.811, adj=0.500, (0 split)
##       bias_user  < 0.5984213  to the left,  agree=0.731, adj=0.289, (0 split)
##       Horror     < 0.5        to the right, agree=0.626, adj=0.013, (0 split)
##       timestamp  < 0.08229144 to the left,  agree=0.621, adj=0.000, (0 split)
##
## Node number 2: 27277 observations,    complexity param=0.04766923
##   mean=2.857609, MSE=1.046521
##   left son=4 (7583 obs) right son=5 (19694 obs)
##   Primary splits:
##       pred       < 0.569275   to the left,  improve=0.135200600, (0 missing)
##       bias_movie < 0.5337323  to the left,  improve=0.055040720, (0 missing)
##       bias_user  < 0.535344   to the left,  improve=0.014003730, (0 missing)
##       Drama      < 0.5        to the left,  improve=0.008228631, (0 missing)
##       timestamp  < 0.2302005  to the right, improve=0.002433942, (0 missing)
##   Surrogate splits:
##       bias_movie < 0.4917384  to the left,  agree=0.808, adj=0.309, (0 split)
##       bias_user  < 0.4932944  to the left,  agree=0.747, adj=0.091, (0 split)
##
## Node number 3: 44723 observations,    complexity param=0.03922652
##   mean=3.911052, MSE=0.751608
##   left son=6 (24655 obs) right son=7 (20068 obs)
##   Primary splits:
##       pred       < 0.7164997  to the left,  improve=0.094480560, (0 missing)
##       bias_movie < 0.7311926  to the left,  improve=0.032331760, (0 missing)
##       bias_user  < 0.7025603  to the left,  improve=0.025322740, (0 missing)
##       year       < 0.7150538  to the right, improve=0.004047365, (0 missing)
```

```
##       timestamp  < 0.5441705  to the right, improve=0.001051033, (0 missing)
##   Surrogate splits:
##       bias_movie < 0.7455654  to the left,  agree=0.709, adj=0.352, (0 split)
##       bias_user  < 0.6753389  to the left,  agree=0.687, adj=0.302, (0 split)
##       year       < 0.7150538  to the right, agree=0.585, adj=0.075, (0 split)
##       War        < 0.5        to the left,  agree=0.561, adj=0.022, (0 split)
##       Crime      < 0.5        to the left,  agree=0.560, adj=0.019, (0 split)
##
## Node number 4: 7583 observations
##   mean=2.251418, MSE=1.058476
##
## Node number 5: 19694 observations
##   mean=3.091018, MSE=0.8459481
##
## Node number 6: 24655 observations
##   mean=3.670635, MSE=0.7309527
##
## Node number 7: 20068 observations
##   mean=4.206423, MSE=0.6187284
```

```
#predict values
rpart_prediction_1 <- predict(rpart_model_1, reduced_dat_test)
#calculate RMSE
rpart_RMSE_1 <- RMSE(rpart_prediction_1, reduced_prediction_results)
rpart_RMSE_1
```

```
## [1] 0.8756083
```

we get an RMSE of 0.8756083. This is less interesting than the linear model.

Naive Bayes

```
#create model
naice_bayes_model_1 = naiveBayes(
  x = reduced_dat_train[-1],
  y = reduced_dat_train$rating
  )
#predict values
naice_bayes_prediction_1 = predict(
  naice_bayes_model_1,
  newdata = reduced_dat_test
  )
#calculate RMSE
naice_bayes_RMSE_1 <- RMSE(
  as.numeric(as.character(naice_bayes_prediction_1)),
  reduced_prediction_results
  )
naice_bayes_RMSE_1
```

```
## [1] 1.12259
```

We get an RMSE of 1.12259. This is not interesting, and actually a lower performance than our baseline.

Neural Network

Finally, we try to use an artificial neural network from the h20 library. This will be used in the same principle than the linear regression: we predict a numeric value, and round to the value of our nearest ordinal category.

```
#Initialize h2o
h2o.init(nthreads = -1)
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 28 minutes
##     H2O cluster timezone:       America/Toronto
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.42.0.2
##     H2O cluster version age:    3 months and 27 days
##     H2O cluster name:           H2O_started_from_R_Admin_voe829
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.76 GB
##     H2O cluster total cores:    16
##     H2O cluster allowed cores:  16
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.3.1 (2023-06-16 ucrt)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is (3 months and 27 days) old. There may be a newer version available.
## Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/latest
```

```
#create model
ann_model_1 = h2o.deeplearning(
  y = 'rating',
  training_frame = as.h2o(reduced_dat_train),
  activation = 'Rectifier',
  hidden = c(20,20),
  epochs = 10,
  train_samples_per_iteration = -2
  )
```

```
##   |                                                                              |
##   |                                                                              |
```

```
#predict values
ann_prediction_1 = h2o.predict(
  ann_model_1,
  newdata = as.h2o(reduced_dat_test)
  )
```

```
##   |                                                                              |
##   |                                                                              |
```

```
ann_prediction_1 = as.vector(ann_prediction_1)
ann_prediction_1 <- round(
  as.numeric(as.character(ann_prediction_1))*2,
  digits = 0
```

```
  )/2
#calculate RMSE
ann_RMSE_1 <- RMSE(ann_prediction_1, reduced_prediction_results)
ann_RMSE_1
```

```
## [1] 0.8682435
```

We get an RMSE of 0.8705437. While this is an interesting result if we can improve it, it is inferior to our current champion, the linear model.

Model selection

We now want to select the model that we will be trying to improve and extend to our entire dataset.

The naive option gave us 1.062822 The linear model gave us 0.8670271 The treemap model gave us 0.8756083 The naive bayes gave us 1.12259 The neural network gave us 0.8705437

Model Improvement.

The best option seems to be the linear model. Let's try to optimize locally with interaction terms.

```
#create model
linear_regression_model_2 <- lm(
  rating ~ .^2, #all values and all interaction terms
  data = reduced_dat_train)

#predict values
linear_regression_prediction_2 <- predict(
  linear_regression_model_2,
  newdata = reduced_dat_test
  )
```

```
## Warning in predict.lm(linear_regression_model_2, newdata = reduced_dat_test):
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
linear_regression_prediction_2 <- round(linear_regression_prediction_2*2, digits = 0)/2

#calculate RMSE
linear_regression_RMSE_2 <- RMSE(
  linear_regression_prediction_2,
  reduced_prediction_results
  )
linear_regression_RMSE_2
```

```
## [1] 0.8648299
```

We get an RMSE of 0.8648299, but the number of parameters significantly increases, ad so does the computing time. We can try to select a handful of interaction terms.

We can find the most relevant parameters with the following.

```
varImp(linear_regression_model_2, conditional=TRUE)
```

```
##                          Overall
## timestamp            0.018485364
## year                 1.505433087
## Action               1.020045511
## Adventure            1.356425865
## Animation            0.070704766
```

```
## Children                1.117019655
## Comedy                  0.536077273
## Crime                   1.505758269
## Documentary             0.089254957
## Drama                   0.058547537
## Fantasy                 1.031933381
## Film_Noir               1.197203491
## Horror                  0.068990356
## Musical                 0.041398911
## Mystery                 0.111909569
## Romance                 2.504701133
## Sci_Fi                  1.370999074
## Thriller                1.130582444
## War                     1.605489159
## Western                 1.391738109
## bias_movie             14.514589896
## bias_user              14.989667312
## timestamp:year          1.705185393
## timestamp:Action        3.448446333
## timestamp:Adventure     0.485381473
## timestamp:Animation     2.586920767
## timestamp:Children      3.217901493
## timestamp:Comedy        0.902341274
## timestamp:Crime         3.020318991
## timestamp:Documentary   0.835885743
## timestamp:Drama         0.821460815
## timestamp:Fantasy       1.638602771
## timestamp:Film_Noir     1.769363506
## timestamp:Horror        3.086811247
## timestamp:Musical       0.213473888
## timestamp:Mystery       0.655210186
## timestamp:Romance       1.185264306
## timestamp:Sci_Fi        1.122964129
## timestamp:Thriller      1.018493266
## timestamp:War           0.215032392
## timestamp:Western       1.463065083
## timestamp:bias_movie    3.563576611
## timestamp:bias_user     1.486334073
## year:Action             0.442170608
## year:Adventure          1.620408068
## year:Animation          0.882813043
## year:Children           2.068833205
## year:Comedy             0.980193305
## year:Crime              0.593482715
## year:Documentary        0.344552031
## year:Drama              0.988940786
## year:Fantasy            0.880360286
## year:Film_Noir          0.819052897
## year:Horror             0.149434127
## year:Musical            0.404302702
## year:Mystery            0.624020469
## year:Romance            1.866308934
## year:Sci_Fi             1.195830907
## year:Thriller           0.968625243
```

```
## year:War                0.042629283
## year:Western            1.397619038
## year:bias_movie         3.171157172
## year:bias_user          2.310886302
## Action:Adventure        1.616668261
## Action:Animation        2.355258078
## Action:Children         0.572093666
## Action:Comedy           0.880660294
## Action:Crime            1.474396422
## Action:Documentary      0.009587383
## Action:Drama            0.871016088
## Action:Fantasy          1.027398820
## Action:Film_Noir        0.688730455
## Action:Horror           0.124736898
## Action:Musical          1.109589269
## Action:Mystery          0.813567944
## Action:Romance          2.137167716
## Action:Sci_Fi           0.117753856
## Action:Thriller         0.437509879
## Action:War              0.236811003
## Action:Western          0.428379839
## Action:bias_movie       1.391246524
## Action:bias_user        1.733256210
## Adventure:Animation     1.114438001
## Adventure:Children      1.509845353
## Adventure:Comedy        0.317058007
## Adventure:Crime         0.612211417
## Adventure:Documentary   0.103273197
## Adventure:Drama         2.145270572
## Adventure:Fantasy       0.089837453
## Adventure:Film_Noir     0.174717356
## Adventure:Horror        1.510376949
## Adventure:Musical       0.212534173
## Adventure:Mystery       0.413055639
## Adventure:Romance       0.672095621
## Adventure:Sci_Fi        0.113236146
## Adventure:Thriller      0.056549376
## Adventure:War           0.778969108
## Adventure:Western       1.493711082
## Adventure:bias_movie    0.191437903
## Adventure:bias_user     1.357255667
## Animation:Children      0.500130943
## Animation:Comedy        1.727707970
## Animation:Crime         2.150842408
## Animation:Drama         1.131795280
## Animation:Fantasy       1.613385976
## Animation:Film_Noir     0.754440499
## Animation:Horror        1.037534094
## Animation:Musical       1.087550663
## Animation:Mystery       0.317962449
## Animation:Romance       1.689345485
## Animation:Sci_Fi        0.370299371
## Animation:Thriller      1.012078676
## Animation:War           0.901374457
```

```
## Animation:Western       0.879268559
## Animation:bias_movie    0.158443176
## Animation:bias_user     0.275043782
## Children:Comedy         1.311978025
## Children:Crime          0.512978184
## Children:Documentary    0.140079189
## Children:Drama          0.855019599
## Children:Fantasy        0.734861988
## Children:Horror         0.912652867
## Children:Musical        0.020781578
## Children:Mystery        0.809542576
## Children:Romance        0.786170066
## Children:Sci_Fi         0.303308117
## Children:Thriller       1.170071954
## Children:War            0.413646033
## Children:Western        0.556098297
## Children:bias_movie     0.317226420
## Children:bias_user      0.109289748
## Comedy:Crime            0.578327222
## Comedy:Documentary      0.805203021
## Comedy:Drama            0.601053172
## Comedy:Fantasy          0.258098798
## Comedy:Film_Noir        0.787478511
## Comedy:Horror           0.644624159
## Comedy:Musical          0.618579392
## Comedy:Mystery          0.760001122
## Comedy:Romance          2.418701431
## Comedy:Sci_Fi           0.011348245
## Comedy:Thriller         0.678831242
## Comedy:War              0.322933421
## Comedy:Western          1.366528089
## Comedy:bias_movie       0.010608548
## Comedy:bias_user        0.500550035
## Crime:Documentary       1.115624279
## Crime:Drama             1.732213887
## Crime:Fantasy           0.981208600
## Crime:Film_Noir         0.590008887
## Crime:Horror            0.037531653
## Crime:Musical           0.314890097
## Crime:Mystery           0.814523145
## Crime:Romance           1.751721720
## Crime:Sci_Fi            0.972092850
## Crime:Thriller          2.190085871
## Crime:War               0.648750596
## Crime:Western           0.448195871
## Crime:bias_movie        0.769500566
## Crime:bias_user         2.844935303
## Documentary:Drama       0.291317831
## Documentary:Fantasy     0.429908791
## Documentary:Horror      1.481264201
## Documentary:Musical     0.268611376
## Documentary:Romance     0.539686759
## Documentary:War         0.461523924
## Documentary:bias_movie  0.621612895
```

```
## Documentary:bias_user   1.144300595
## Drama:Fantasy           0.350953805
## Drama:Film_Noir         1.951120515
## Drama:Horror            0.443588762
## Drama:Musical           0.086335416
## Drama:Mystery           0.017629892
## Drama:Romance           1.388925985
## Drama:Sci_Fi            0.595052920
## Drama:Thriller          0.564414538
## Drama:War               0.547663365
## Drama:Western           1.049337866
## Drama:bias_movie        0.081724943
## Drama:bias_user         0.475541883
## Fantasy:Film_Noir       0.615079939
## Fantasy:Horror          2.170957817
## Fantasy:Musical         1.243190103
## Fantasy:Mystery         0.128814727
## Fantasy:Romance         1.090403481
## Fantasy:Sci_Fi          0.294684012
## Fantasy:Thriller        1.650004705
## Fantasy:War             0.823544707
## Fantasy:Western         0.733834775
## Fantasy:bias_movie      1.578535493
## Fantasy:bias_user       0.916325610
## Film_Noir:Horror        0.056012322
## Film_Noir:Musical       0.482406715
## Film_Noir:Mystery       0.381790203
## Film_Noir:Romance       0.704096484
## Film_Noir:Thriller      0.973228455
## Film_Noir:bias_movie    1.016523900
## Film_Noir:bias_user     0.925794653
## Horror:Musical          0.597331532
## Horror:Mystery          0.360489198
## Horror:Romance          1.171464044
## Horror:Sci_Fi           0.416291053
## Horror:Thriller         2.466062368
## Horror:War              1.456444819
## Horror:Western          3.969344016
## Horror:bias_movie       0.123850627
## Horror:bias_user        0.792580528
## Musical:Mystery         0.527205613
## Musical:Romance         0.937160442
## Musical:Sci_Fi          1.337602437
## Musical:Thriller        0.618269459
## Musical:War             1.889270565
## Musical:Western         1.092546028
## Musical:bias_movie      0.617299347
## Musical:bias_user       1.219904496
## Mystery:Romance         0.289631475
## Mystery:Sci_Fi          2.477868834
## Mystery:Thriller        0.855429854
## Mystery:War             0.574414418
## Mystery:Western         0.927091903
## Mystery:bias_movie      0.329624963
```

```
## Mystery:bias_user       0.583977578
## Romance:Sci_Fi          0.326897205
## Romance:Thriller        0.090137474
## Romance:War             1.682958566
## Romance:Western         0.895117289
## Romance:bias_movie      2.971929857
## Romance:bias_user       0.529075792
## Sci_Fi:Thriller         1.267385419
## Sci_Fi:War              0.277790756
## Sci_Fi:Western          0.301648013
## Sci_Fi:bias_movie       1.856807825
## Sci_Fi:bias_user        2.395741430
## Thriller:War            0.575304110
## Thriller:Western        0.928889932
## Thriller:bias_movie     0.114566667
## Thriller:bias_user      0.809671649
## War:Western             1.133030803
## War:bias_movie          2.377714042
## War:bias_user           0.298447619
## Western:bias_movie      0.805685081
## Western:bias_user       1.062439007
## bias_movie:bias_user   11.539175650
## bias_movie:pred         2.111696223
## bias_user:pred          2.428702463
```

This allows us to spot one particularly important interaction: bias_movie:bias_user.

```
#create model
linear_regression_model_3 <- lm(
  rating ~ timestamp + year + Action + Adventure +
    Animation + Children + Comedy + Crime + Documentary + Drama +
    Fantasy + Film_Noir + Horror + Musical + Mystery + Romance +
    Sci_Fi + Thriller + War + Western + bias_movie + bias_user + bias_movie:bias_user,
  data = reduced_dat_train)

#predict values
linear_regression_prediction_3 <- predict(
  linear_regression_model_3,
  newdata = reduced_dat_test
  )
linear_regression_prediction_3 <- round(linear_regression_prediction_3*2, digits = 0)/2

#calculate RMSE
linear_regression_RMSE_3 <- RMSE(
  linear_regression_prediction_3,
  reduced_prediction_results
  )
linear_regression_RMSE_3
```

```
## [1] 0.86548
```

We get an RMSE 0.86548. We did have a small increase in RMSE, but we still have some rather good results.

Final Implementation

We now move on to the implementation of the model on the full dataset.

Final test data

We modified the training data to fit our purpose. We must now apply the same modifications to the final_holdout_test data in order. We cannot use the final_holdout_test to train, so we must use the movie_avgs and user_avgs that we got from the training data. We will also use the scale we got from the training data, because using two different scales would be a problem.

```
# use regex to get the date from the title
final_holdout_test_2 <- final_holdout_test %>%
  mutate(title = str_trim(title)) %>%
  extract(title, c("title_tmp", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F) %>%
  mutate(year = if_else(str_length(year) > 4, as.integer(str_split(year, "-", simplify = T)[1]), as.inte
  mutate(title = if_else(is.na(title_tmp), title, title_tmp)) %>%
  select(-title_tmp)  %>%
  mutate(genres = if_else(genres == "(no genres listed)", `is.na<-`(genres), genres))


for (genre_index in 1:length(genres)){
  final_holdout_test_2[genres[genre_index]] <- NA
  final_holdout_test_2[genres[genre_index]] <- as.integer(str_detect(final_holdout_test_2$genre, genres
}

#rename Film-Noir and Sci-Fi to avoid issues later on with the "-"
names(final_holdout_test_2)[names(final_holdout_test_2) == "Film-Noir"] <- "Film_Noir"
names(final_holdout_test_2)[names(final_holdout_test_2) == "Sci-Fi"] <- "Sci_Fi"

#add the biases for movie and user, from the training data
final_holdout_test_3 <- final_holdout_test_2%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = avg + bias_movie + bias_user)

final_holdout_test_3 <- final_holdout_test_3[,c(-1,-2, -5, -7)] # remove excess columns

final_holdout_test_4 <- predict(scale, as.data.frame(final_holdout_test_3)) #set to the same scale as t
```

Model implementation and optimisation

For one last test, we will separate the full training test between a training and testing group, and see if our model maintains its interest after being applied to a greater amount of data.

```
train_test_index <- createDataPartition(y = dat_train_4$rating, times = 1, p = 0.2, list = FALSE)
check_dat_train <- dat_train_4[-train_test_index,]
check_dat_test <- dat_train_4[train_test_index,]
check_prediction_results <- check_dat_test[,1]
check_dat_test<- check_dat_test[,-1]
```

We apply this data to the model.

```
#create model
linear_regression_model_4 <- lm(
  rating ~ timestamp + year + Action + Adventure + Animation +
  Children + Comedy +Crime + Documentary + Drama + Fantasy +
  Film_Noir + Horror + Musical + Mystery + Romance + Sci_Fi +
  Thriller + War + Western + bias_movie + bias_user + bias_movie:bias_user,
  data = check_dat_train)
```

```r
#predict values
linear_regression_prediction_4 <- predict(
  linear_regression_model_4,
  newdata = check_dat_test
  )
linear_regression_prediction_4 <- round(linear_regression_prediction_4*2, digits = 0)/2

#calculate RMSE
linear_regression_RMSE_4 <- RMSE(
  linear_regression_prediction_4,
  check_prediction_results
  )
linear_regression_RMSE_4
```

```
## [1] 0.8669469
```

We get an RMSE of 0.8671712. We seem to have lost some performance, but maintain acceptable values. We can proceed to the final evaluation.

Final model

Finally, we implement on the full training dataset and compare to our final_holdout_test and see how we fare.

```r
final_model <- lm(
  rating ~ timestamp + year + Action + Adventure + Animation +
  Children + Comedy +Crime + Documentary + Drama + Fantasy +
  Film_Noir + Horror + Musical + Mystery + Romance + Sci_Fi +
  Thriller + War + Western + bias_movie + bias_user + bias_movie:bias_user,
  data= dat_train_4
  )


final_prediction <- predict(final_model, final_holdout_test_4[,-1])
final_RMSE <- RMSE(final_prediction, final_holdout_test_4[,1])
final_RMSE
```

```
## [1] 0.8638888
```

The final RMSE is 0.8638888.

Conclusion

The aim of this project was to predict movie ratings from the MovieLens dataset. We transformed the provided dataset to extract information from it, tested several possible models on a small scale, selected and improved the most promising one and trained it on the full scale dataset for the final prediction. This work led us to a prediction with an RMSE of 0.8638888, which fits the desired performance.

This project has limitations. A number of techniques, such as regularization, might have added some performance. Some of the methodologies dismissed on a small scale might have reacted better on the full dataset and resulted in better results. We also can notice that the methodology was skimming between regression and classification. While this produced results that fit the requested performance, a "pure" classification methodology might have been interesting to implement.

In the future, I would be interested in pursuing this project by exploring further the neural network model. The H2O library offers a very large amount of forms of options for optimization, and I am curious to see if it

would be possible to explore some parameters that were used with their default configuration in the test.

Bibliography

My work was informed by a variety of previous works over the web.

The MovieLens documentation

MovieLens 10M/100k Data Set README. (n.d.). Files.grouplens.org. https://files.grouplens.org/datasets/movielens/ml-10m-README.html

Other analysis project

These projects were projects using the MovieLens dataset, publicly available over the web

Bandurski, P. (2017, February 2). movieLens dataset analysis. Kaggle.com. https://www.kaggle.com/code/redroy44/movielens-dataset-analysis

Jailani, F. (2020, September 23). RPubs - Movie rating prediction in R. Api.rpubs.com. https://api.rpubs.com/fjailani/movielens

Khonje, J. (2021, June 30). RPubs - MovieLens Rating Prediction using Machine Learning Project. Rpubs.com. https://rpubs.com/Khonjeja/794882

Mineo, G. (2019, March 24). MovieLens-Rating-Prediction-Project. GitHub. https://github.com/gmineo/MovieLens-Rating-Prediction-Project

Outerelo, C. (n.d.). RPubs - MovieLens Recommender System. Rpubs.com. Retrieved November 21, 2023, from https://rpubs.com/outerelocarlos/MovieLens-Recommender-System