IMDb Dataset Analysis

Statistical Learning Project - Predicting IMDb Score

Nicolo' Merzi, Francesco Battista

1. Introduction

Internet Movie Database (IMDb) is the world's most popular source for movies, TV and celebrity content. The goal of this project is to create statistical models using data mining algorithms to predict the ratings of movies. Such analysis can help in having a better understanding of what makes a movie succesfull.

Dataset can be found at: https://data.world/data-society/imdb-5000-movie-dataset#

2. Data Description

The dataset consists of 28 variables for 5043 movies. Our response variables is *imdb_score* and the other 27 variables are possible predictors.

Variable Name	Description
movie_title	Title of the Movie
duration	Duration (Minutes)
director_name	Name of the Director of the Movie
director_facebook_likes	Number of likes of the Director on his Facebook Page
actor_1_name	Primary actor starring in the movie
actor_1_facebook_likes	Number of likes of the Actor_1 on his/her Facebook Page

Variable Name	Description
actor_2_name	Other actor starring in the movie
actor_2_facebook_likes	Number of likes of the Actor_2 on his/her Facebook Page
actor_3_name	Other actor starring in the movie
actor_3_facebook_likes	Number of likes of the Actor_3 on his/her Facebook Page
num_user_for_reviews	Number of users who gave a review
num_critic_for_reviews	Number of critical reviews on imdb
num_voted_users	Number of people who voted for the movie
cast_total_facebook_likes	Total number of facebook likes of the entire cast of the movie
movie_facebook_likes	Number of Facebook likes in the movie page
plot_keywords	Keywords describing the movie plot
facenumber_in_poster	Number of the actor who featured in the movie poster
color	Film colorization. (‘Black and White’ or ‘Color’)
genres	Film categorization (‘Animation’, ‘Comedy’, ‘Romance’, ‘Horror’, ‘Sci-Fi’, ‘Action’, ‘Family’)
title_year	The year in which the movie is released (1916:2016)
language	Original Language of the movie
country	Country where the movie is produced
content_rating	Content rating of the movie
aspect_ratio	Aspect ratio the movie
movie_imdb_link	IMDB link of the movie
gross	USA gross earnings of the movie (Dollars)
budget	Budget of the movie (Dollars)
imdb_score	IMDB Score of the movie

3. Data Exploration

3.1 Read Data

```
## 'data.frame':    5043 obs. of  28 variables:
##  $ color                    : Factor w/ 3 levels ""," Black and White",..: 3 3 3 3 1 3 3 3 3 3 ...
##  $ director_name            : Factor w/ 2399 levels "","A. Raven Cruz",..: 927 801 2027 377 603 106
##  $ num_critic_for_reviews   : int  723 302 602 813 NA 462 392 324 635 375 ...
##  $ duration                 : int  178 169 148 164 NA 132 156 100 141 153 ...
##  $ director_facebook_likes  : int  0 563 0 22000 131 475 0 15 0 282 ...
##  $ actor_3_facebook_likes   : int  855 1000 161 23000 NA 530 4000 284 19000 10000 ...
##  $ actor_2_name             : Factor w/ 3033 levels "","50 Cent","A. Michael Baldwin",..: 1407 2218
##  $ actor_1_facebook_likes   : int  1000 40000 11000 27000 131 640 24000 799 26000 25000 ...
##  $ gross                    : int  760505847 309404152 200074175 448130642 NA 73058679 336530303 2008
##  $ genres                   : Factor w/ 914 levels "Action","Action|Adventure",..: 107 101 128 288 75
##  $ actor_1_name             : Factor w/ 2098 levels "","50 Cent","A.J. Buckley",..: 302 979 353 1968
##  $ movie_title              : Factor w/ 4917 levels "[Rec] ","[Rec] 2 ",..: 398 2731 3279 3708 3332
##  $ num_voted_users          : int  886204 471220 275868 1144337 8 212204 383056 294810 462669 321795
##  $ cast_total_facebook_likes: int  4834 48350 11700 106759 143 1873 46055 2036 92000 58753 ...
##  $ actor_3_name             : Factor w/ 3522 levels "","50 Cent","A.J. Buckley",..: 3442 1392 3134 1
##  $ facenumber_in_poster     : int  0 0 1 0 0 1 0 1 4 3 ...
##  $ plot_keywords            : Factor w/ 4761 levels "","10 year old|dog|florida|girl|supermarket",..
##  $ movie_imdb_link          : Factor w/ 4919 levels "http://www.imdb.com/title/tt0006864/?ref_=fn_tt_
##  $ num_user_for_reviews     : int  3054 1238 994 2701 NA 738 1902 387 1117 973 ...
##  $ language                 : Factor w/ 48 levels "","Aboriginal",..: 13 13 13 13 1 13 13 13 13 13 .
##  $ country                  : Factor w/ 66 levels "","Afghanistan",..: 65 65 63 65 1 65 65 65 65 63
##  $ content_rating           : Factor w/ 19 levels "","Approved",..: 10 10 10 10 1 10 10 9 10 9 ...
##  $ budget                   : num  2.37e+08 3.00e+08 2.45e+08 2.50e+08 NA ...
##  $ title_year               : int  2009 2007 2015 2012 NA 2012 2007 2010 2015 2009 ...
##  $ actor_2_facebook_likes   : int  936 5000 393 23000 12 632 11000 553 21000 11000 ...
##  $ imdb_score               : num  7.9 7.1 6.8 8.5 7.1 6.6 6.2 7.8 7.5 7.5 ...
##  $ aspect_ratio             : num  1.78 2.35 2.35 2.35 NA 2.35 2.35 1.85 2.35 2.35 ...
##  $ movie_facebook_likes     : int  33000 0 85000 164000 0 24000 0 29000 118000 10000 ...
```

We have 5043 obs. of 28 variables. Response variable *imdb_score* is a numerical variable and the other predictors are a mix of numerical and categorical variables.

4. Data Cleaning

4.1 Remove duplicate rows

Let's check for duplicate rows.

```r
sum(duplicated(IMDB))
```

```
## [1] 45
```

We have found 45, let's remove them.

```r
IMDB <- IMDB[!duplicated(IMDB), ]
```

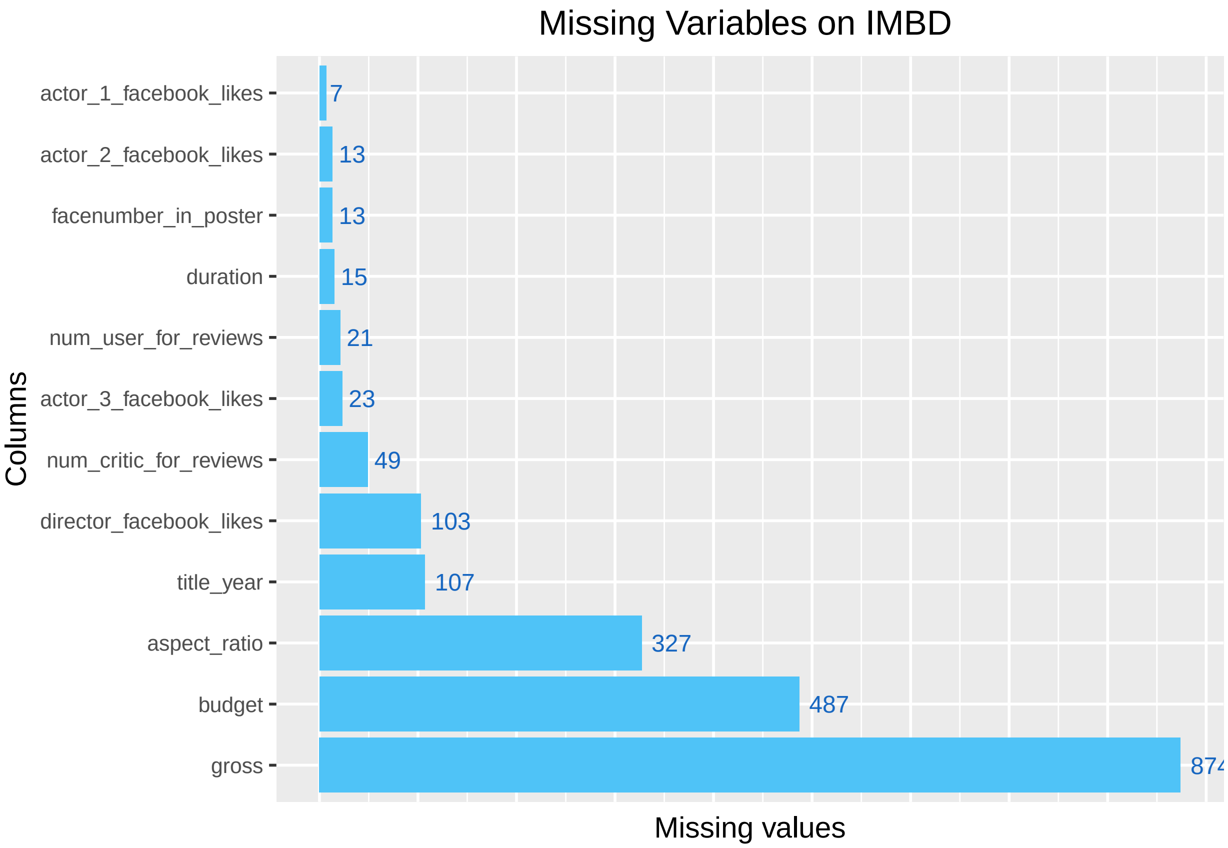
We are now left with 4998 observations.

```r
dim(IMDB)
```

```
## [1] 4998   28
```

4.2 Missing values

Check how many missing values we have for each column.



gross, *budget* and *aspect_ratio* have the most missing values with 874, 487 and 327 NA respectively. The number of missing values on those columns is quite high, for this reason we cannot replace them with reasonable data (e.g mean of the column), so let's remove those rows.

```
IMDB <- IMDB[!is.na(IMDB$gross), ]
IMDB <- IMDB[!is.na(IMDB$budget), ]
IMDB <- IMDB[!is.na(IMDB$aspect_ratio), ]
dim(IMDB)
```

```
## [1] 3783   28
```

We are left with 3783 observations.

4.3 Deal with 0s

There are some 0 values that should be treated as missing values (except for *facenumber_in_poster*). The columns containing 0 values are:

- *num_critic_for_reviews*
- *director_facebook_likes*
- *actor_3_facebook_likes*
- *actor_1_facebook_likes*
- *cast_total_facebook_likes*
- *actor_2_facebook_likes*
- *movie_facebook_likes*

First we replace missing values of *facenumber_in_poster* with the column average.

```
IMDB$facenumber_in_poster[is.na(IMDB$facenumber_in_poster)] <- round(mean(IMDB$facenumber_in_poster, na
```

After that we convert 0 values of the other predictors to NA and then we replace them with the column average.

```
#convert 0s to NA for columns containing 0 values
IMDB[,c(3,5,6,8,14,25,28)][IMDB[,c(3,5,6,8,14,25,28)] == 0] <- NA

#replace NA with columns average
IMDB$num_critic_for_reviews[is.na(IMDB$num_critic_for_reviews)] <- round(mean(IMDB$num_critic_for_revie
IMDB$director_facebook_likes[is.na(IMDB$director_facebook_likes)] <- round(mean(IMDB$director_facebook_l
IMDB$actor_3_facebook_likes[is.na(IMDB$actor_3_facebook_likes)] <- round(mean(IMDB$actor_3_facebook_lik
IMDB$actor_1_facebook_likes[is.na(IMDB$actor_1_facebook_likes)] <- round(mean(IMDB$actor_1_facebook_lik
IMDB$cast_total_facebook_likes[is.na(IMDB$cast_total_facebook_likes)] <- round(mean(IMDB$cast_total_fac
IMDB$actor_2_facebook_likes[is.na(IMDB$actor_2_facebook_likes)] <- round(mean(IMDB$actor_2_facebook_lik
IMDB$movie_facebook_likes[is.na(IMDB$movie_facebook_likes)] <- round(mean(IMDB$movie_facebook_likes, na
```

4.4 Missing values for categorical variables

There are 32 missing values for column *content_ratings* as blank strings (""). Since we cannot replace those values with reasonable data we remove those rows.

```
table(IMDB$content_rating)
```

```
##
##            Approved         G        GP         M     NC-17 Not Rated
##        32        17        89         1         2         6        35
##    Passed        PG     PG-13         R     TV-14      TV-G     TV-MA
##          3       566      1302      1697         0         0         0
##     TV-PG      TV-Y     TV-Y7   Unrated         X
##          0         0         0        23        10
```

```
#remove blank strings ("") for column content_rating
IMDB <- IMDB[!(IMDB$content_rating == ""), ]
```

We are now left with 3751 observations.

Countries have different content rating systems for movies. Some levels of *content_rating* have different names but mean the same thing, so let's group them togheter according to the most used content rating system: the Motion Picture Association of America (MPAA). According to: https://en.wikipedia.org/wiki/Motion_picture_content_rating_system:

- M = GP = PG
- X = NC-17

```
#replace M and GP with PG
IMDB$content_rating[IMDB$content_rating == 'M']   <- 'PG'
IMDB$content_rating[IMDB$content_rating == 'GP']  <- 'PG'
#replace X with NC-17
IMDB$content_rating[IMDB$content_rating == 'X']   <- 'NC-17'
```

We then replace "Approved", "Not Rated", "Passed", "Unrated" with the most common rating "R".

```
IMDB$content_rating[IMDB$content_rating == 'Approved']  <- 'R'
IMDB$content_rating[IMDB$content_rating == 'Not Rated'] <- 'R'
IMDB$content_rating[IMDB$content_rating == 'Passed']    <- 'R'
IMDB$content_rating[IMDB$content_rating == 'Unrated']   <- 'R'
```

```r
IMDB$content_rating <- factor(IMDB$content_rating)
table(IMDB$content_rating)
```

```
##
##     G NC-17    PG PG-13     R
##    89    16   569  1302  1775
```

We are left with the 5 content ratings of the Motion Piture Association of America (MPAA).

4.5 Variables Distribution

let's check how variables are distributed.

4.5.1 Color

```r
table(IMDB$color)
```

```
##
##                   Black and White            Color
##                2              123             3626
```

We see that the vast majority of movies are colored meaning that this variable is almost constant, so we can remove it.

```r
IMDB <- subset(IMDB, select = -c(color))
```

4.5.2 Country

```r
table(IMDB$country)
```

```
##
##                                   Afghanistan            Argentina
##                    0                    1                    3
##                Aruba            Australia              Bahamas
##                    1                   39                    0
##              Belgium               Brazil             Bulgaria
##                    1                    5                    0
##             Cambodia             Cameroon               Canada
##                    0                    0                   61
##                Chile                China             Colombia
##                    1                   13                    1
##       Czech Republic              Denmark   Dominican Republic
##                    3                    8                    0
##                Egypt              Finland               France
##                    0                    1                  102
##              Georgia              Germany               Greece
##                    1                   79                    1
##            Hong Kong              Hungary              Iceland
##                   13                    2                    1
##                India            Indonesia                 Iran
##                    5                    1                    4
##              Ireland               Israel                Italy
##                    7                    1                   11
##                Japan                Kenya           Kyrgyzstan
##                   15                    0                    0
##                Libya               Mexico          Netherlands
##                    0                    8                    3
##             New Line          New Zealand              Nigeria
##                    1                   11                    0
##               Norway        Official site             Pakistan
##                    4                    1                    0
##               Panama                 Peru          Philippines
##                    0                    1                    0
##               Poland              Romania               Russia
##                    1                    2                    3
##             Slovakia             Slovenia         South Africa
##                    0                    0                    3
##          South Korea         Soviet Union                Spain
##                    8                    0                   22
##               Sweden          Switzerland               Taiwan
##                    0                    0                    2
##             Thailand               Turkey                   UK
##                    4                    0                  314
## United Arab Emirates                  USA         West Germany
##                    0                 2981                    1
```

Most movies are from USA, followed by UK. In order to have fewer levels we group togheter all movies that were not made in USA or UK.

```
levels(IMDB$country) <- c(levels(IMDB$country), "Others")
IMDB$country[(IMDB$country != 'USA')&(IMDB$country != 'UK')] <- 'Others'
IMDB$country <- factor(IMDB$country)
table(IMDB$country)
```

```
## 
##     UK    USA Others 
##    314   2981    456
```

4.5.3 Language

```
table(IMDB$language)
```

```
## 
##            Aboriginal     Arabic    Aramaic    Bosnian  Cantonese
##          1          2          1          1          1          7
##    Chinese      Czech     Danish       Dari      Dutch   Dzongkha
##          0          1          3          2          3          0
##    English   Filipino     French     German      Greek     Hebrew
##       3592          1         34         11          0          1
##      Hindi  Hungarian  Icelandic Indonesian    Italian   Japanese
##          5          1          0          2          7         10
##    Kannada     Kazakh     Korean   Mandarin       Maya  Mongolian
##          0          1          5         14          1          1
##       None  Norwegian    Panjabi    Persian     Polish Portuguese
##          1          4          0          3          0          5
##   Romanian    Russian  Slovenian    Spanish    Swahili    Swedish
##          1          1          0         23          0          0
##      Tamil     Telugu       Thai       Urdu Vietnamese       Zulu
##          0          0          3          0          1          1
```

The vast majoirty of movies are in English meaning that *language* is nearly constant, so we can remove it.

```
IMDB <- subset(IMDB, select = -c(language))
```

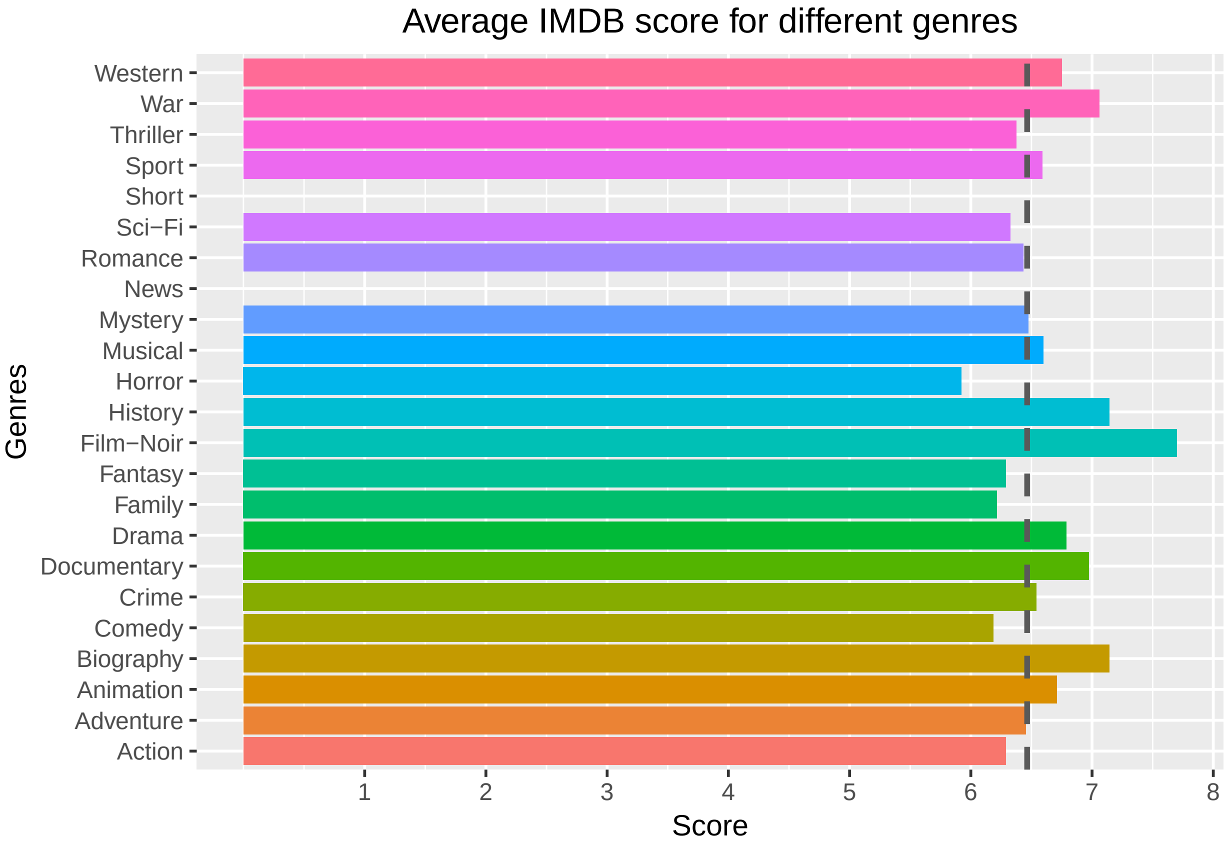
4.5.4 Split Genres

Variable *genres* contains all the genres to which the movie belongs separeted by "|". In order to see the effect of different genres on *imdb_score*, we compute the mean of the score for every genre. Some movies belong to more than one genre, in that case we count them as separate movies, but with the same score. So, for example, the movie Avatar belongs to four different genres: Action, Adventure, Fantasy and Sci-Fi, and has a score of 7.9. Here we consider it as four different movies (one for each genre) with the same score of 7.9.

```
# all different genres
names <- c("Action", "Adventure", "Animation", "Biography", "Comedy", "Crime", "Documentary", "Drama",


avg <- c()
j <- 1

#Calulate the imdb_score mean for every genre and save it in dataframe
for(i in names){
  temp <- lapply(IMDB$genre,grepl,pattern = i)
  avg[j] <- mean(IMDB$imdb_score[temp == TRUE])
  j <- j + 1
}
avg <- as.data.frame(avg)
avg$genre <- names
```



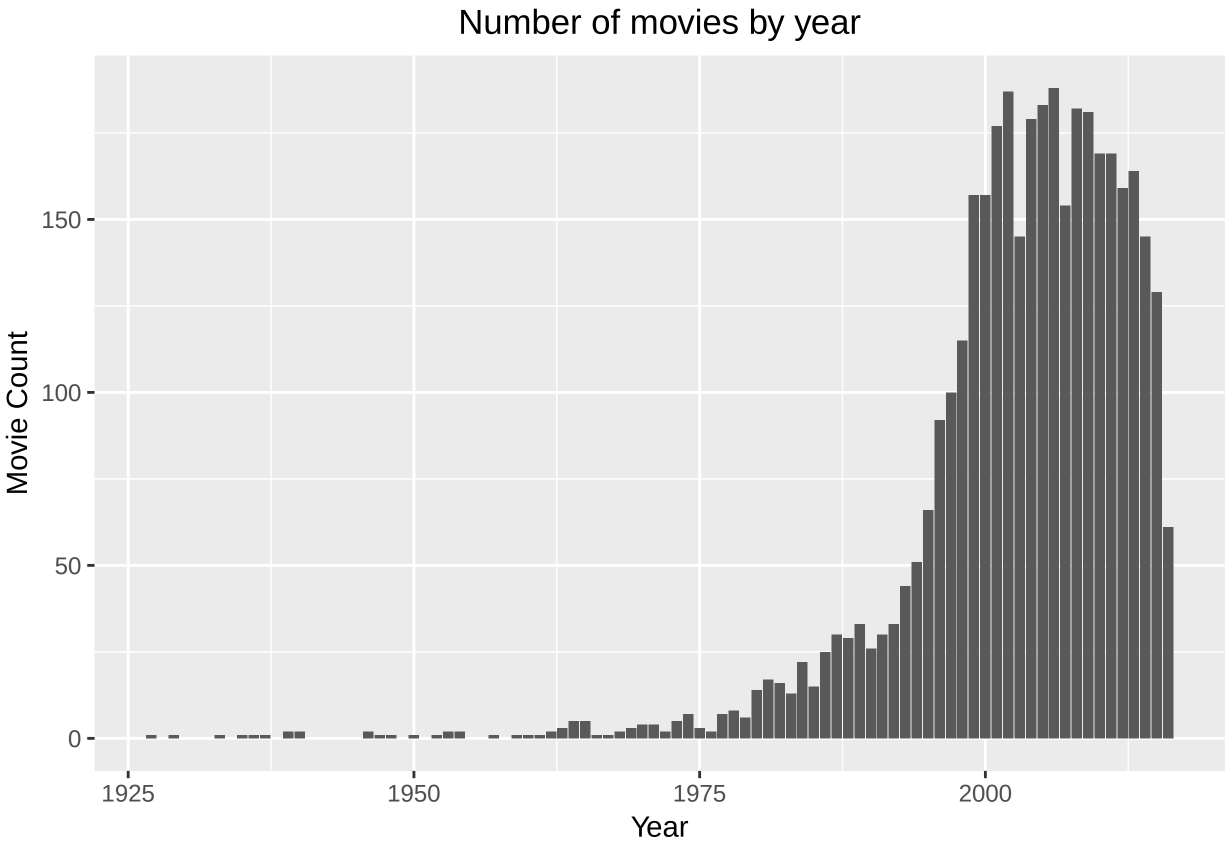
We can see that the average score for different genres is approximately the same (around 6-7), with only

“Film-Noir” being far from the average (dashed grey line). Due to the data cleaning process we are left with no movies that belong to the “Short” and “News” genre, for this reason we do not have an average for those categories. Since all genres fall approximately around the same mean we can conclude that genre does not have a significant effect on the score.

```
IMDB <- subset(IMDB, select = -c(genres))
```

5. Data Visualization

5.1 Number of movies by year

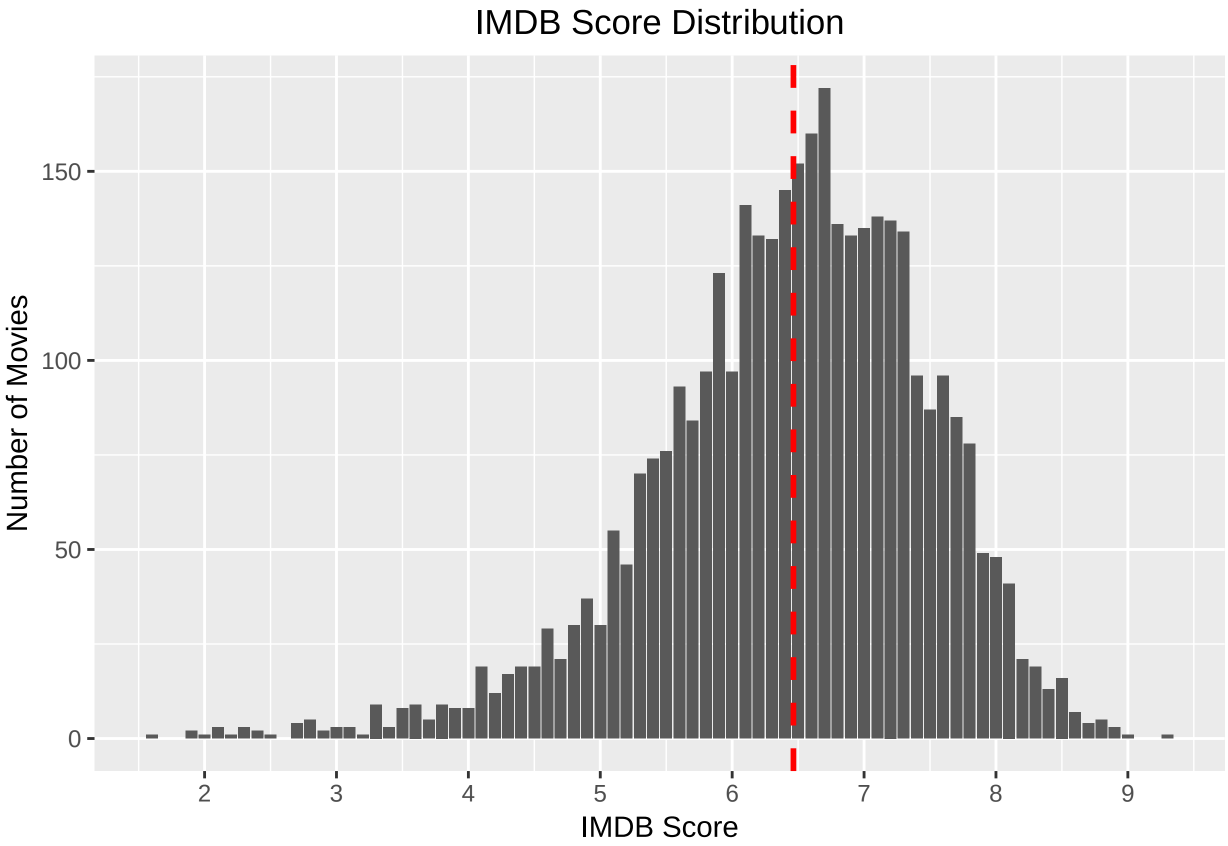


We see that most movies were released after 1980s. Since we have such few movies from before the 80’s we remove them because they might be not very representative.

```
IMDB <- IMDB[IMDB$title_year >= 1980,]
```

We are left with 3657 observations.

5.2 IMDB Score Distribution



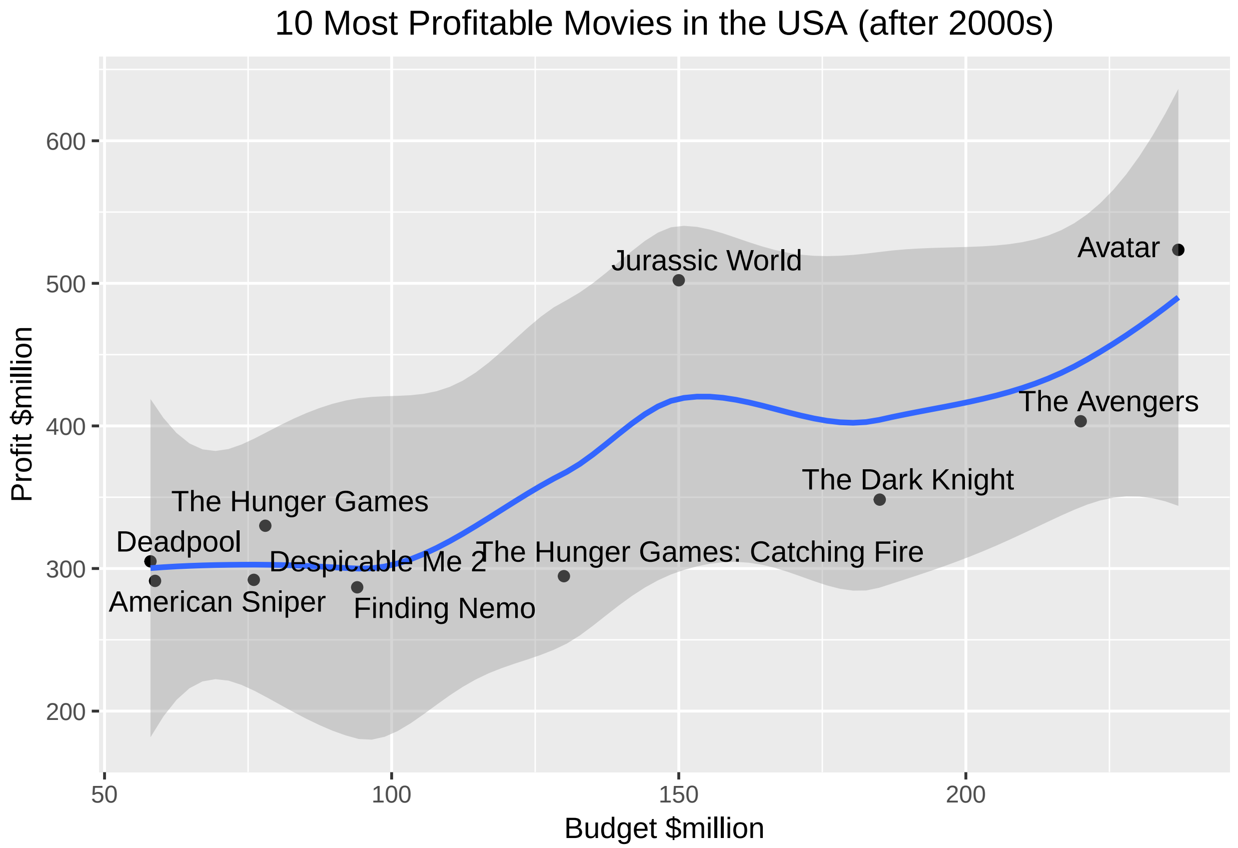
We see that the ratings are approximately normally distributed with a mean around 6.5.

5.3 Most Profitable Movies in the USA

We add a column named *profit* that is given by the difference between gross earning and the budget of the movie. **It is important to remember that *gross* referes only to USA earnings**. For example, the movie "Spectre" had a budget of 245M and grossed 200M in the USA, resulting in a loss of around 45M, but the movie made over 880M world wide.

```r
#add column profit
IMDB$profit <- (IMDB$gross - IMDB$budget)
```

```r
#order movies based on profit
IMDBprofit <- arrange(IMDB, desc(profit))
#filter by year. Consider movies after 2000
IMDBprofit <- IMDBprofit[IMDBprofit$title_year >= 2000, ]
#top10 by profit
IMDBprofit <- IMDBprofit[1:10,]
```



If we consider more recent movies (after 2000s) we can see that high budget movies tend to earn more profit.

5.4 Top20 Directors with highest average IMDb score

```
#mean of imdb_score for every director
directorAvgScore <- aggregate(IMDB$imdb_score, list(IMDB$director_name), mean)
#order
directorAvgScore <- arrange(directorAvgScore, desc(x))
#take first 20 directors
top20Directors <- directorAvgScore[1:20,]

#rename columns
names(top20Directors)[names(top20Directors) == "Group.1"] <- "director"
names(top20Directors)[names(top20Directors) == "x"] <- "avgScore"
```

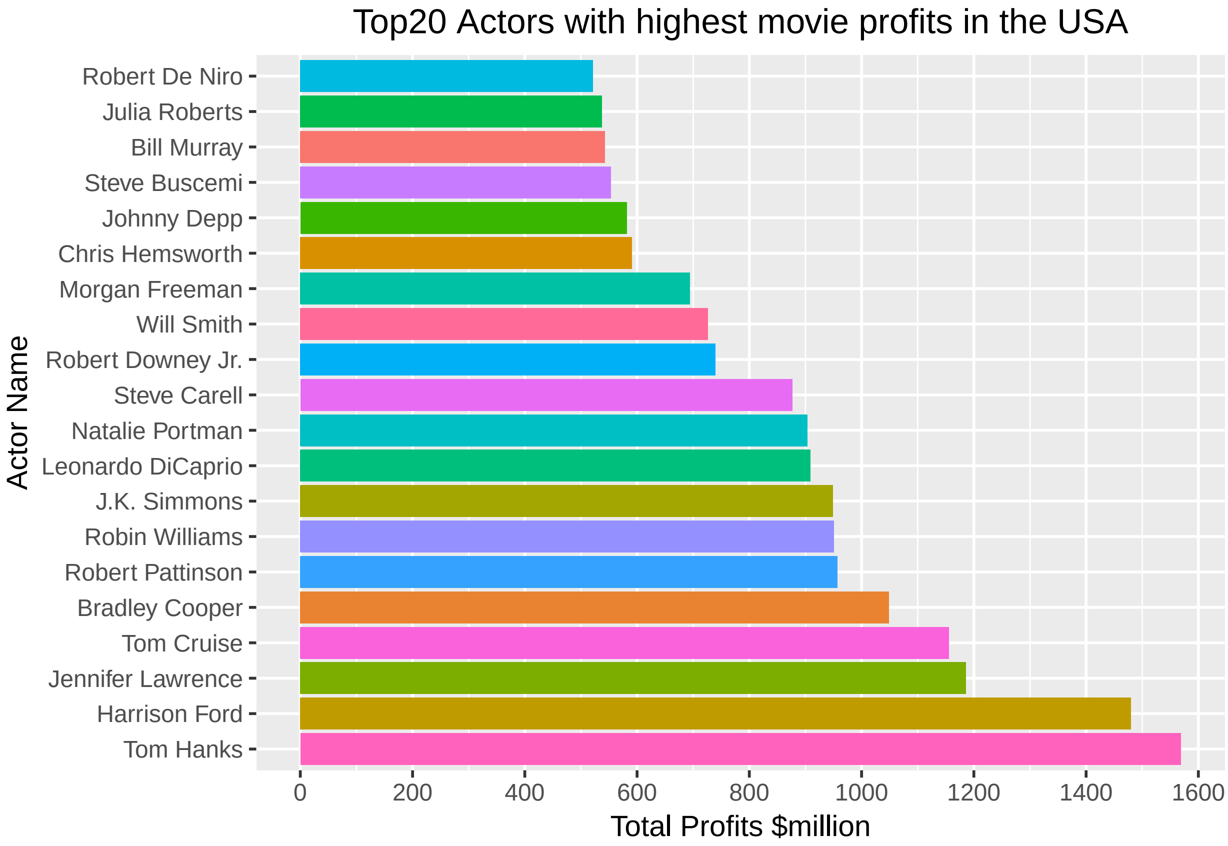
```
top20Directors
```

```
##                director avgScore
## 1             Tony Kaye 8.600000
## 2       Damien Chazelle 8.500000
## 3          Majid Majidi 8.500000
## 4            Ron Fricke 8.500000
## 5     Christopher Nolan 8.425000
## 6        Asghar Farhadi 8.400000
## 7      Richard Marquand 8.400000
## 8          Sergio Leone 8.400000
## 9           Lee Unkrich 8.300000
## 10     Lenny Abrahamson 8.300000
## 11          Pete Docter 8.233333
## 12       Hayao Miyazaki 8.225000
## 13   Joshua Oppenheimer 8.200000
## 14 Juan José Campanella 8.200000
## 15    Quentin Tarantino 8.200000
## 16        David Sington 8.100000
## 17          Je-kyu Kang 8.100000
## 18         Terry George 8.100000
## 19           Tim Miller 8.100000
## 20           Ari Folman 8.000000
```

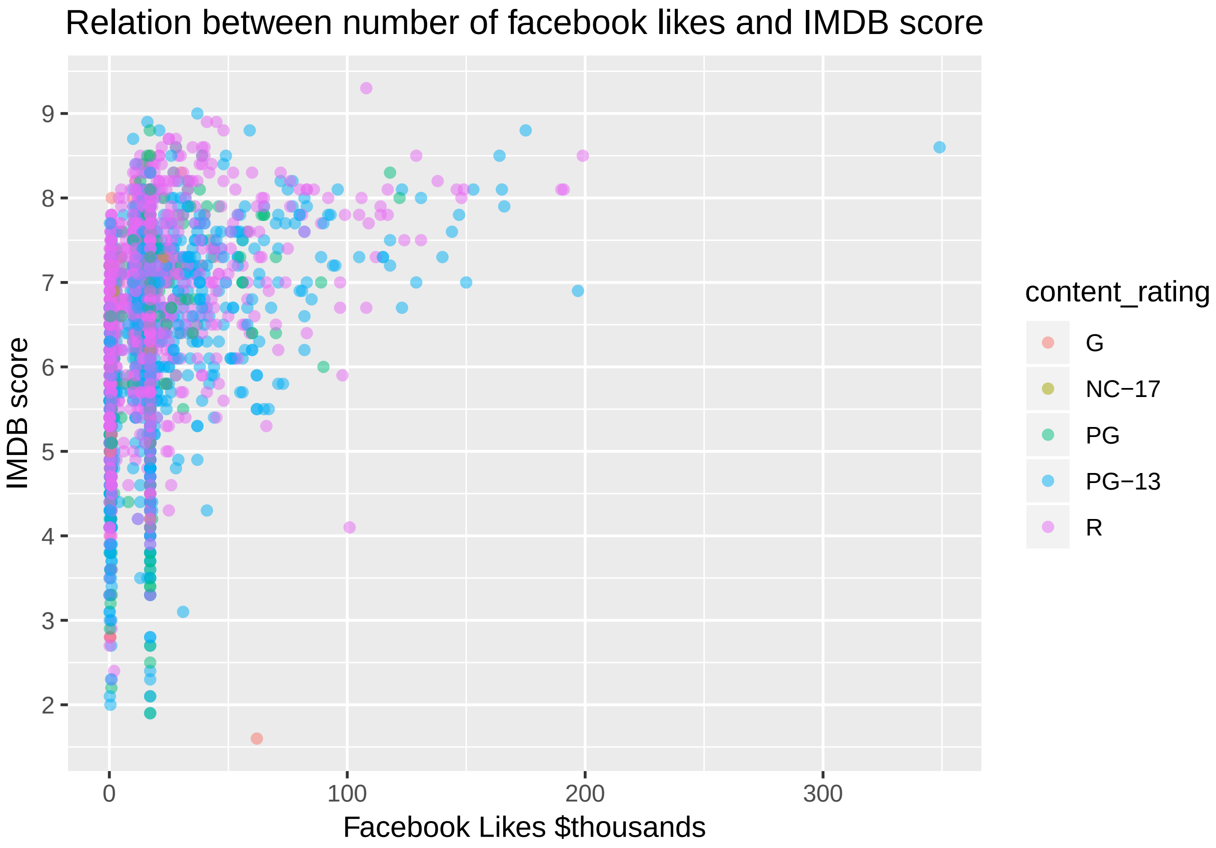
5.5 Top20 Actors with highest movie profits in the USA

```
#sum profit for every main actor
actorsProfit <- aggregate(IMDB$profit, list(IMDB$actor_1_name), sum)
#order
actorsProfit <- arrange(actorsProfit, desc(x))
#take first 20 actors
top20Actors <- actorsProfit[1:20,]

#rename columns
names(top20Actors)[names(top20Actors) == "Group.1"] <- "Actor"
names(top20Actors)[names(top20Actors) == "x"] <- "totalProfit"
```



5.6 Relation between number of facebook likes and IMDB score



Movies with very high Facebook likes (more than 100k) tend to have higher imdb score, while the score for movies with low Facebook likes vary in a very wide range.

6. Data Pre-processing

At this point we need to prepare the dataset in order to implement the algorithms. We start by removing some columns that will not be helpful in the prediction. *plot_keywords* are too diverse to be used and variables *movie_imdb_link* and *movie_title* are all unique so they will have no effect.

```
#remove columns not useful in the prediction
IMDB <- subset(IMDB, select = -c(aspect_ratio, plot_keywords, movie_imdb_link, movie_title))
```

Actors are also very different for the whole dataset. We have 3567 unique actors for 3657 movies, so we can drop the names since there are too many to be useful in the prediction.

```
#total number of unique actors
sum(uniqueN(IMDB[, c("actor_1_name", "actor_2_name", "actor_3_name")]))
```

```
## [1] 3567
```

Same for directors: they are too many unique directors to be useful in the prediction. (1624 different directors)

```
#total number of unique directors
sum(uniqueN(IMDB[, "director_name"]))
```

```
## [1] 1624
```

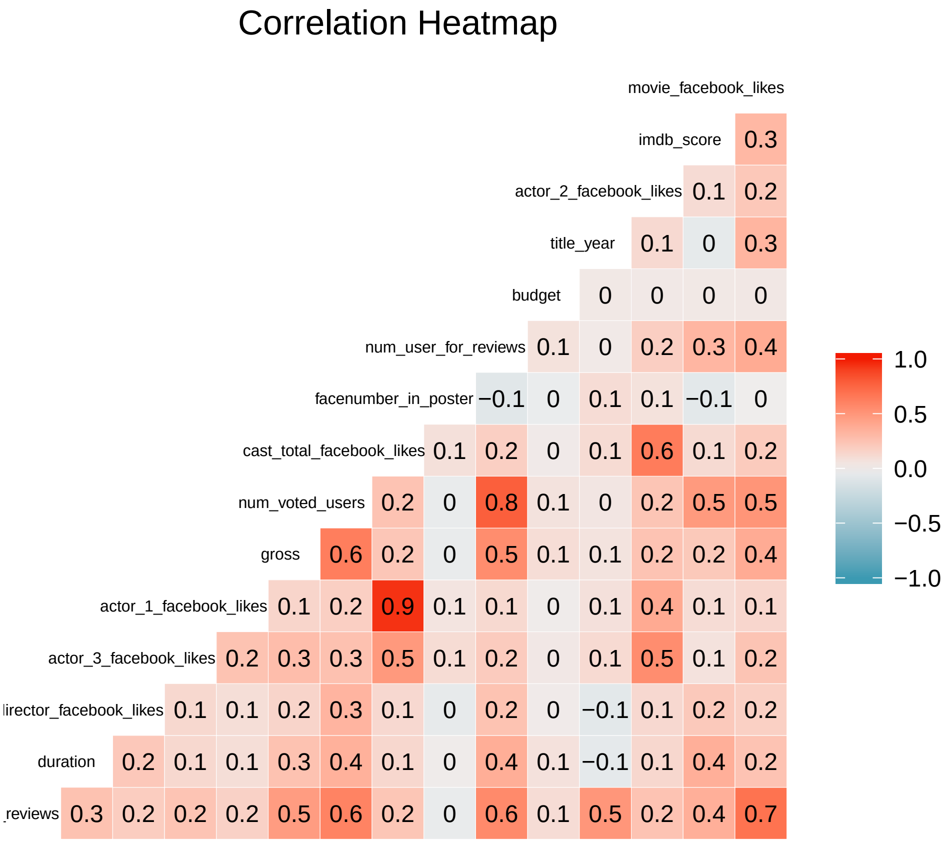
```
#remove columns not useful in the prediction
IMDB <- subset(IMDB, select= -c(actor_1_name, actor_2_name, actor_3_name, director_name))
```

We also have to remove column *profit* to avoid multicollinearity, since we obtained that information through other existing variables (*gross* and *budget*).

```
#remove column profit
IMDB <- subset(IMDB, select= -c(profit))
```

6.1 Remove highly correlated variables

Before implementing any algorithm we need to check how variables are correlated with each other.



From the graph we can see high correlation between some variables (greater than 0.7).

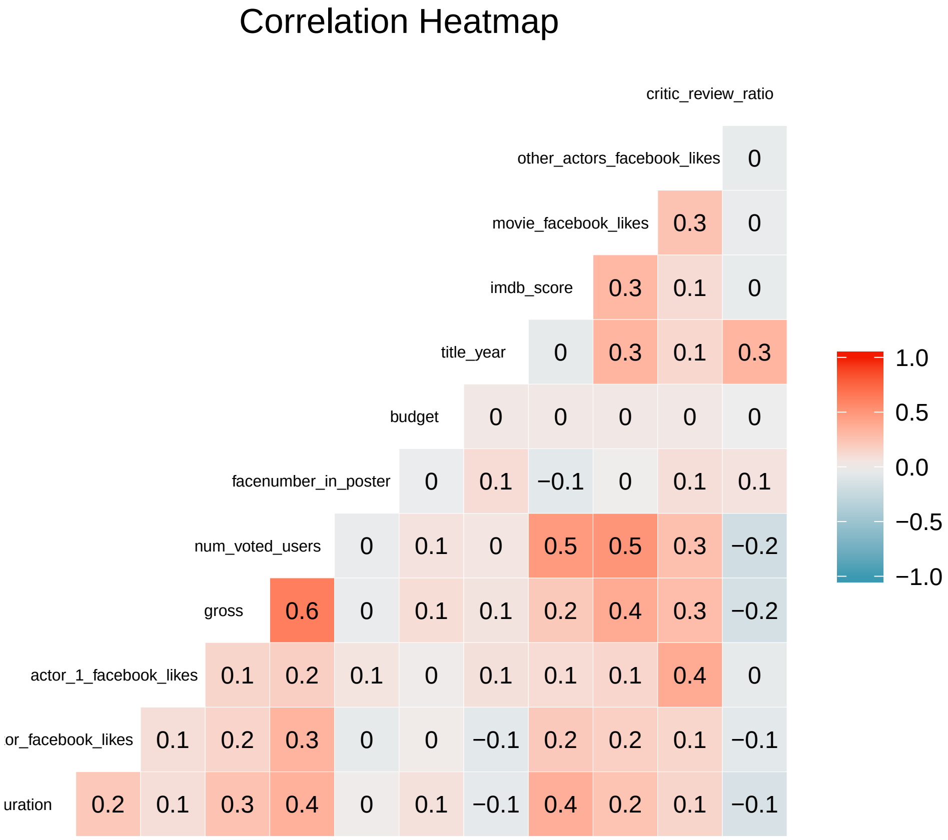
In particular, *actor_1_facebook_likes* is highly correleted with *cast_total_facebook_likes* with a value of 0.95, and *actor_2_facebook_likes* and *actor_3_facebook_likes* are somewhat correleted to *cast_total_facebook_likes*. Since the total facebook likes of the whole cast is mostly based on the sum of actor_1, actor_2 and actor_3 we can group *actor_2_facebook_likes* and *actor_3_facebook_likes* togheter and remove the total Facebook likes of the whole cast (*cast_total_facebook_likes*).

There is also a strong correlation between *num_voted_users* and *num_user_for_reviews* (corr of 0.78). To avoid this we can take the ratio between *num_critic_for_reviews* and *num_user_for_reviews* and then remove those columns.

By making these changes we keep approximately the same amount of information and avoid the high correlation between those variables.

```
#new column with sum of actor 2 and 3 facebook likes
IMDB$other_actors_facebook_likes <- IMDB$actor_2_facebook_likes + IMDB$actor_3_facebook_likes
#ratio of critic review
IMDB$critic_review_ratio <- IMDB$num_critic_for_reviews / IMDB$num_user_for_reviews
```

```
#remove columns
IMDB <- subset(IMDB, select = -c(actor_2_facebook_likes, actor_3_facebook_likes, cast_total_facebook_li
```



Now we do not have any high correlation between predictors.

6.2 Bin IMDB score

The goal of this project is trying to predict if a movie will be good or bad: we are not interested in making an exact prediction up to decimal points. We assume that a movie with a score of 8 and a movie with a score of 9 are both excellent movies, and movies with scores of 3 and 4 are both bad movies. For this reason we bin the score into buckets:

- **Bad**: score from 0 to 4
- **Mediocre**: score from 4 to 6
- **Good**: score from 6 to 8
- **Excellent**: score from 8 to 10

```
#new column with the binned_score
IMDB$binned_score <- cut(IMDB$imdb_score, breaks = c(0,4,6,8,10))
```

6.3 Reorder Dataset

We change variables name and positions to make it easier to read and understand.

```
IMDB <- IMDB[, c(7,8,10,1,6,5,14,12,2,3,13,9,4,11,15)]
colnames(IMDB) <- c("country", "content", "year", "duration", "poster", "user_vote", "critic_review_rati
```

6.4 Split dataset

We want to split the dataset into train set and test set with a 8:2 ratio.

```
set.seed(42)

sample <- sample.int(n = nrow(IMDB), size= floor(.80*nrow(IMDB)), replace = FALSE)
train <- IMDB[sample,]
test <- IMDB[-sample,]
```

7. Implement Algorithms

Dataset is now clean and ready to be used. Since we have binned the score into buckets we are now dealing with a classification problem: predict in what bucket a movie belongs to. The algorithms we are going to implement are:

- Classification Tree
- Random Forest
- Ordinal Logisic Regression
- K-Nearest Neighbors
- Support Vector Machine

7.1 Classification Tree

The first method used is classification tree, a tree-based method which divides the predictor space into a number of simple regions. The basic idea behind **decision trees** is finding a set of rules that best partition your data. This type of method is applicable to both classification problems (as in this case) and regression problems.

The task of growing a decision tree is done by recursive binary splitting: find the splitting rules that best divide the tree into two branches. If we are dealing with classification problem, the Residual Sum of Squares (RSS) cannot be used as criterion for making the binary splits. Instead we can use different methods to asses the quality of the splits, such as the Classification Error Rate: the fraction of training observation in a particular region that do not belong to the most widely occuring class.

In addition to the Classification Error Rate, there are other sensitivity measures that can be used to make the splits, the Gini Index and Cross-Entropy. Both can be used as measures of node purity (low values indicate that a node contains predominant observations from a single class).

7.1.1 Full Grown Tree

```
library(rpart)
library(rpart.plot)
```

We use rpart() function to fit a classification tree in order to predict *binned_score* using all variables but *imdb_score*.

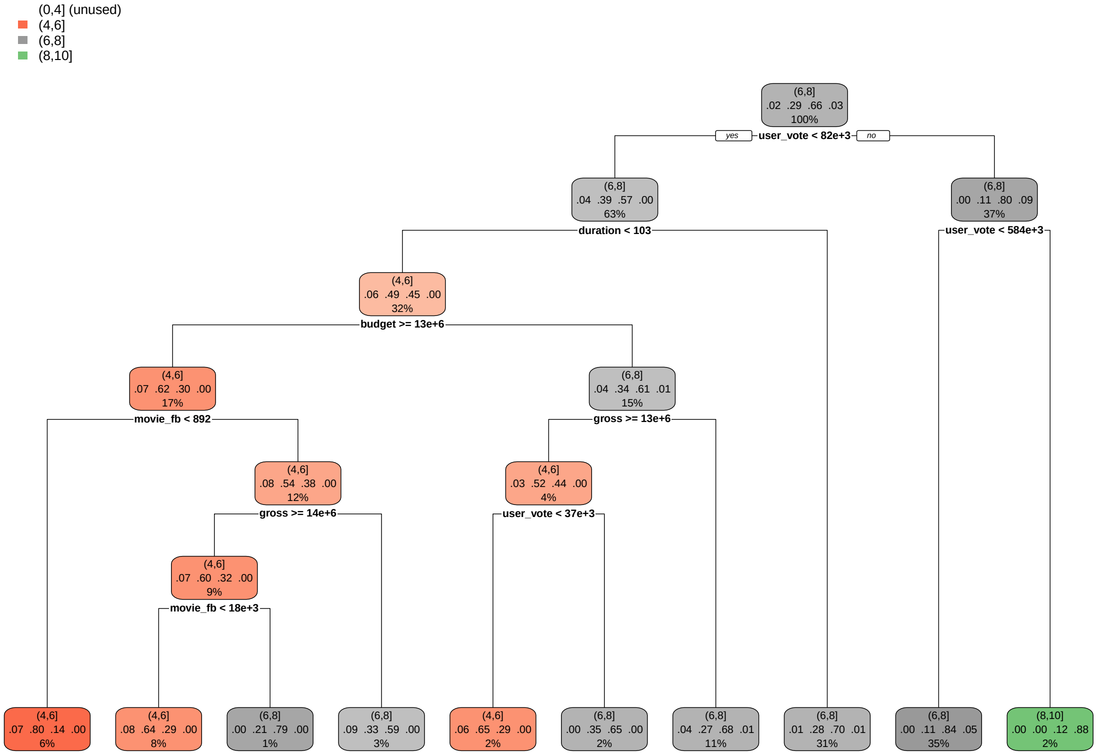
```
set.seed(42)
#classification tree model
tree.IMDB <- rpart(binned_score ~ duration+country+content+year+gross+poster+movie_fb+actor1_fb+other_ac
```

```
printcp(tree.IMDB)
```

```
##
## Classification tree:
## rpart(formula = binned_score ~ duration + country + content +
##     year + gross + poster + movie_fb + actor1_fb + other_actors_fb +
##     budget + director_fb + critic_review_ratio + user_vote, data = train,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] budget    duration  gross     movie_fb  user_vote
##
## Root node error: 1006/2925 = 0.34393
##
## n= 2925
##
##         CP nsplit rel error  xerror     xstd
## 1 0.053015      0   1.00000 1.00000 0.025537
## 2 0.038767      3   0.84095 0.88569 0.024743
## 3 0.012922      4   0.80219 0.83996 0.024367
## 4 0.010437      6   0.77634 0.83300 0.024306
## 5 0.010000      9   0.74453 0.82604 0.024245
```

With the function printcp() we can see the variables actually used to build the tree, as well as the cross validation error results. Cross-validation shows that the full grown tree (5th tree with 9 split) is the best model: the model that minimize the cross-validation error (xerror=0.82604).

```
#plot of the full_grown tree with 9 split
rpart.plot(tree.IMDB)
```



From the classification rules we can see that if a movie has over 584k user votes on IMDb it gets classified as an excellent movie. This makes sense considering that popular movies have lots of fan that vote for them. Now let's test the accuracy of our model on the *train* and *test* dataset.

```
#predict on train set
tree.pred.train <- predict(tree.IMDB, train, type = "class")
confusionMatrix(tree.pred.train, train$binned_score)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     34   331   113      0
##     (6,8]     33   508  1800     55
##     (8,10]     0     0     6     45
##
## Overall Statistics
##
##                Accuracy : 0.7439
##                  95% CI : (0.7277, 0.7597)
##     No Information Rate : 0.6561
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3831
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000      0.3945      0.9380      0.45000
## Specificity               1.00000      0.9295      0.4076      0.99788
## Pos Pred Value                NaN      0.6925      0.7513      0.88235
## Neg Pred Value            0.97709      0.7924      0.7750      0.98086
## Prevalence                0.02291      0.2868      0.6561      0.03419
## Detection Rate            0.00000      0.1132      0.6154      0.01538
## Detection Prevalence      0.00000      0.1634      0.8191      0.01744
## Balanced Accuracy         0.50000      0.6620      0.6728      0.72394
```

```
#predict on test set
tree.pred.test <- predict(tree.IMDB, test, type = "class")
confusionMatrix(tree.pred.test, test$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     11    75    38      0
##     (6,8]     13   134   429     17
##     (8,10]     0     0     1     14
##
## Overall Statistics
##
##                Accuracy : 0.7077
##                  95% CI : (0.6732, 0.7404)
##     No Information Rate : 0.6393
##     P-Value [Acc > NIR] : 5.516e-05
##
##                   Kappa : 0.3246
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000      0.3589      0.9167      0.45161
## Specificity               1.00000      0.9063      0.3788      0.99857
## Pos Pred Value                NaN      0.6048      0.7234      0.93333
## Neg Pred Value            0.96721      0.7796      0.7194      0.97629
## Prevalence                0.03279      0.2855      0.6393      0.04235
## Detection Rate            0.00000      0.1025      0.5861      0.01913
## Detection Prevalence      0.00000      0.1694      0.8101      0.02049
## Balanced Accuracy         0.50000      0.6326      0.6477      0.72509
```

The full grown tree performs with an accuracy of:

- Train set: 0.7439
- Test set: 0.7077

7.1.2 Pruned Tree

With the default complexity parameter (cp=0.01), the model only builds 5 trees and the cross-validation results showed that the full-grown tree was the best one. By lowering the complexity parameter value we can force the model to build more trees.

```
set.seed(23)
#fit new classification tree with lower cp value
tree2.IMDB <- rpart(binned_score ~ duration+country+content+year+gross+poster+movie_fb+actor1_fb+other_a
```

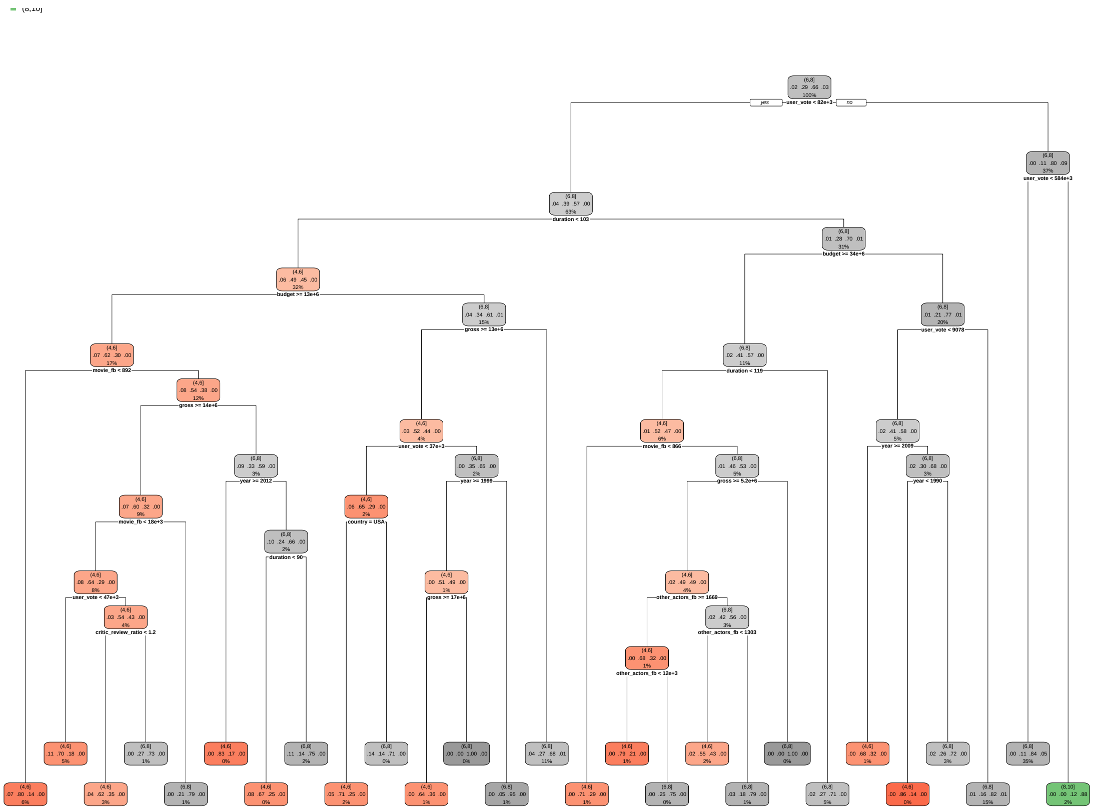
```
#show cross-validation errors results
printcp(tree2.IMDB)
```

```
##
## Classification tree:
## rpart(formula = binned_score ~ duration + country + content +
##     year + gross + poster + movie_fb + actor1_fb + other_actors_fb +
##     budget + director_fb + critic_review_ratio + user_vote, data = train,
##     method = "class", cp = 1e-06)
##
## Variables actually used in tree construction:
##  [1] actor1_fb           budget              country
##  [4] critic_review_ratio director_fb         duration
##  [7] gross               movie_fb            other_actors_fb
## [10] poster              user_vote           year
##
## Root node error: 1006/2925 = 0.34393
##
## n= 2925
##
##            CP nsplit rel error  xerror     xstd
## 1  0.05301524      0   1.00000 1.00000 0.025537
## 2  0.03876740      3   0.84095 0.89364 0.024805
## 3  0.01292247      4   0.80219 0.85984 0.024535
## 4  0.01043738      6   0.77634 0.82903 0.024272
## 5  0.00795229      9   0.74453 0.82604 0.024245
## 6  0.00616302     10   0.73658 0.82604 0.024245
## 7  0.00497018     18   0.68588 0.83101 0.024289
## 8  0.00397614     22   0.66600 0.81014 0.024102
## 9  0.00347913     26   0.65010 0.82207 0.024210
## 10 0.00331345     31   0.63221 0.83002 0.024280
## 11 0.00298211     38   0.59841 0.82207 0.024210
## 12 0.00248509     54   0.54970 0.83101 0.024289
## 13 0.00198807     57   0.54175 0.83300 0.024306
## 14 0.00165673     74   0.50596 0.84493 0.024409
## 15 0.00149105     78   0.49702 0.83698 0.024341
## 16 0.00124254     94   0.45527 0.83996 0.024367
## 17 0.00099404     98   0.45030 0.83897 0.024358
## 18 0.00049702    101   0.44732 0.82704 0.024254
## 19 0.00000100    105   0.44533 0.84095 0.024375
```

We now have 19 trees instead of 5. We can see that the 8th tree has a lower cross-validation error (xerror=0.81014) than the previously full-grown tree (5th tree, xerror=0.82604). Now we can prune this new tree with the cp value that minimize xerror (cp=0.00397614).

```
#prune tree by cp value that minimize xerror
pruned.IMDB <- prune(tree2.IMDB, cp=0.00397614)
```

```
#plot new pruned tree
rpart.plot(pruned.IMDB)
```



As we can see the pruned tree is actually longer than the previous full-grown tree but it gives a lower cross-validation error, so let's test this new "pruned" tree.

```
#predict train set
tree.pred.train <- predict(pruned.IMDB, train, type = "class")
confusionMatrix(tree.pred.train, train$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     35   476   163      0
##     (6,8]     32   363  1750     55
##     (8,10]     0     0     6     45
##
## Overall Statistics
##
##                Accuracy : 0.7764
##                  95% CI : (0.7609, 0.7914)
##     No Information Rate : 0.6561
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4917
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.5673       0.9119       0.45000
## Specificity               1.00000       0.9051       0.5527       0.99788
## Pos Pred Value                NaN       0.7062       0.7955       0.88235
## Neg Pred Value            0.97709       0.8387       0.7669       0.98086
## Prevalence                0.02291       0.2868       0.6561       0.03419
## Detection Rate            0.00000       0.1627       0.5983       0.01538
## Detection Prevalence      0.00000       0.2304       0.7521       0.01744
## Balanced Accuracy         0.50000       0.7362       0.7323       0.72394
```

```
#predict test set
tree.pred.test <- predict(pruned.IMDB, test, type = "class")
confusionMatrix(tree.pred.test, test$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     13   102    63      0
##     (6,8]     11   107   404     17
##     (8,10]     0     0     1     14
##
## Overall Statistics
##
##                Accuracy : 0.7104
##                  95% CI : (0.676, 0.743)
##     No Information Rate : 0.6393
##     P-Value [Acc > NIR] : 2.836e-05
##
##                   Kappa : 0.3689
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.4880       0.8632       0.45161
## Specificity               1.00000       0.8547       0.4886       0.99857
## Pos Pred Value                NaN       0.5730       0.7495       0.93333
## Neg Pred Value            0.96721       0.8069       0.6684       0.97629
## Prevalence                0.03279       0.2855       0.6393       0.04235
## Detection Rate            0.00000       0.1393       0.5519       0.01913
## Detection Prevalence      0.00000       0.2432       0.7363       0.02049
## Balanced Accuracy         0.50000       0.6714       0.6759       0.72509
```

The pruned tree performs with an accuracy of:

- Train set: 0.7764
- Test set: 0.7104

7.1.3 Results

Classification Tree	Train set accuracy	Test set accuracy
Full-grown tree (9 split)	0.7439	0.7077

Classification Tree	Train set accuracy	Test set accuracy
Pruned Tree (22 split)	0.7764	0.7104

The "pruned" tree (8th tree with 22 split) has slightly better performance than our first classification tree (5th tree with 9 split), but not significant enough. The first model creates a more interpretable tree and is a better generalization since it uses less variables and a lower number of splits.

7.2 Random Forest

Random Forests are an ensemble learning method for classification and regression. This model works by constructing a multitude of decision trees and outputting the most occurying class (in a classification setting).

In order to explain how a random forest works we need to briefly introduce the concepts of Bootstrapping and Bagging.

Bootstrapping is a resampling method based on repeatedly sampling observations (with replacement) from the original dataset.

Bagging is a general-purpose procedure for reducing the variance of a statistical learning method by averaging a set of observation: take many training set from the population, build a separate prediction model using each training set, and then average the resulting predictions.

Random Forest is an improvement over bagged trees. When building decision trees, Bagging considers all predictors as possible split candidate. Instead, Random Forest only consider a random subset of the predictors. By doing this we avoid the problem of having the same strong predictor in every decision tree, and thus avoid making every tree similar.

7.2.1 Build Model

Grow a random forest using function RandomForest() with *binned_score* as response variable using all predictors except *imdb_score*. In order to chose a suitable number of m variables to consider at each split, we try all of them and choose the m that maximize the accuracy.

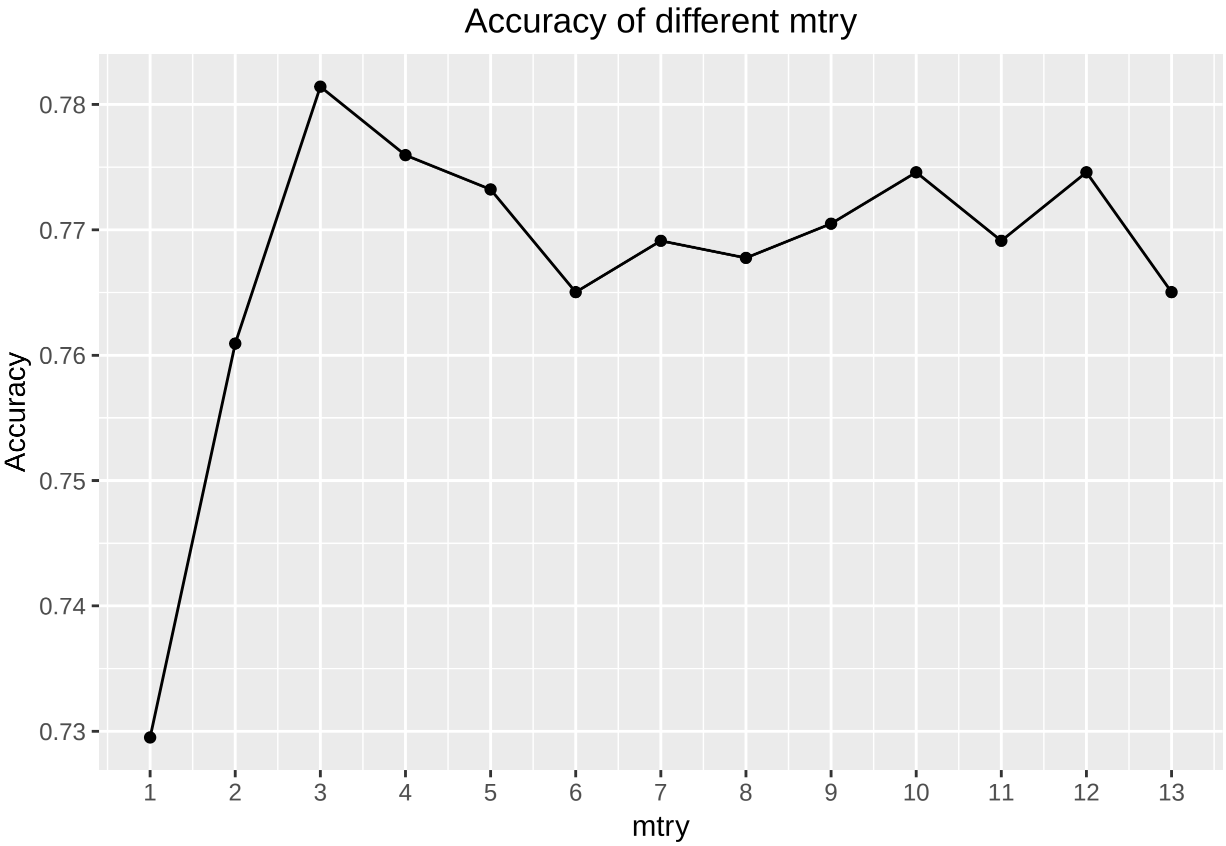
```r
library(randomForest)
set.seed(47)

acc <- c()

#Build 13 different random forest trying all possible m value (1:13)
for (i in 1:13) {
  rf <- randomForest(binned_score ~ duration+country+content+year+gross+poster+movie_fb+actor1_fb+other_
  #predict on test set
  rf.pred.test <- predict(rf, test)
  #save accuracy in a vector
  acc[i] <- confusionMatrix(rf.pred.test, test$binned_score)$overall[1]
}
```

```r
accMat <- cbind(seq(1,13,1), acc)
colnames(accMat)[colnames(accMat) == ""] <- "mtry"

#plot relation between accuracy and m
ggplot(as.data.frame(accMat), aes(x=mtry, y=acc)) +
scale_x_continuous(breaks = seq(1,13,1)) +
geom_point() +
geom_line() +
labs(x="mtry", y="Accuracy", title = "Accuracy of different mtry") +
theme(plot.title = element_text(hjust = 0.5))
```



We can see from the graph that random forests yields better results than bagging (mtry=13). Considering 3 variables at each split (mtry=3) gives the best accuracy. Now, let's test the model.

7.2.2 Apply Model

```
set.seed(47)

#grow random forest using mtry=3
rfB <- randomForest(binned_score ~ duration+country+content+year+gross+poster+movie_fb+actor1_fb+other_a

#predict on test set
rfB.pred.test <- predict(rfB, test)
confusionMatrix(rfB.pred.test, test$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     18   114    33      0
##     (6,8]      6    95   434     13
##     (8,10]     0     0     1     18
##
```

```
## Overall Statistics
##
##                Accuracy : 0.7732
##                  95% CI : (0.7411, 0.8031)
##     No Information Rate : 0.6393
##     P-Value [Acc > NIR] : 3.749e-15
##
##                   Kappa : 0.5026
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.5455       0.9274       0.58065
## Specificity               1.00000       0.9025       0.5682       0.99857
## Pos Pred Value                NaN       0.6909       0.7920       0.94737
## Neg Pred Value            0.96721       0.8325       0.8152       0.98177
## Prevalence                0.03279       0.2855       0.6393       0.04235
## Detection Rate            0.00000       0.1557       0.5929       0.02459
## Detection Prevalence      0.00000       0.2254       0.7486       0.02596
## Balanced Accuracy         0.50000       0.7240       0.7478       0.78961
```

Random Forest preforms with an accuracy of: 0.7732

```
rfB$importance
```

```
##                     MeanDecreaseGini
## duration                   152.40229
## country                     24.15845
## content                     41.78419
## year                        96.30004
## gross                      135.67222
## poster                      45.01934
## movie_fb                   123.48978
## actor1_fb                   95.89040
## other_actors_fb            103.55292
## budget                     131.79063
## director_fb                111.67782
## critic_review_ratio        120.35569
## user_vote                  239.05265
```

The interpretation of Bagging or a Random Forest is typically difficult: despite increasing the accuracy of the prediction, it loses the ease of interpretation.

One can get an overall summary of the importance of each predictor using the Gini Index. As we have already said it measures the impurity of a node where values close to zero mean that that node contains only one predominant class. It is possible to obtain an average of Gini Index calculated for each predictor of each tree created, that measure is called *variable importance*. Mean Decrease in Gini is the average (mean) of a variable's total decrease in node impurity, weighted by the proportion of samples reaching that node in each individual decision tree in the random forest. This is effectively a measure of how important a variable is for estimating the value of the target variable across all of the trees that make up the forest.

In our case we can see that *user_vote*, *duration* and *gross* are all important variables, while *country*, *content* and *poster* are less important.

7.3 Ordinal Logistic Regression

Logistic Regression models the probabilities for calssification problems with two possible outcomes. It's an extension of linear regression for classification problems.

Rather than modeling our dependant variable directly, we use the probability of our dependent variable belonging to a given category. In order to do this we use the logistic probability function and the maximum likelihood method. Since the prediction of a linear model can assume values below zero and above one, we need a way to convert this in probabilities (between 0 and 1). So, instead of fitting a straight line or hyperplane, the logistic regression model uses the logistic function to map the output of a linear equation between 0 and 1, which represent the probability of belonging to a given class.

The interpretation of this model is more difficult than that seen in previous methods and in linear regression models since there is no direct relationship between the coefficients and the dependent variable.

Usually the logistic regression model is used for two-class problems, a possible extension to multi-classification problems is given by the Ordinal Logistic Regression. Ordinal Logistic Regression is a regression model for ordinal dependent variables. In our case *binned_score* is an ordered categorical variable where *(0,4]* < *(4,6]* < *(6,8]* < *(8,10]*.

One of the assumptions underlying Ordinal Logistic Regression is that the relationship between each pair of outcome groups is the same. In other words, Ordinal Logistic Regression assumes that the coefficients that describe the relationship between, say, the lowest versus all higher categories of the response variable are the same as those that describe the relationship between the next lowest category and all higher categories, etc. This is called the *proportional odds assumption* or the *parallel regression assumption*. Because the relationship between all pairs of groups is the same, there is only one set of coefficients. If this was not the case, we would need different sets of coefficients in the model to describe the relationship between each pair of outcome groups. Thus, in order to asses the appropriateness of our model, we need to evaluate whether the proportional odds assumption is tenable. Statistical tests that do this are available in some software packages. However, these tests have been criticized for having a tendency to reject the null hypothesis (that the sets of coefficients are the same).

Therefore not having a precise way to check the proportional odds assumption, we have assumed that the relationships between our classes are the same for the whole model.

While there certainly are better suited model to perfrom classification, it is still useful to apply a simple method such as logistic regression in order to have more benchmarks to compare models.

7.3.1 Model Assumptions

- Before building our model we need to make sure that our dependent variable *binned_score* is correctly coded as an ordered factor.

```
train$binned_score <- as.ordered(train$binned_score)
```

- In order to be able to apply Ordinal Logistic Regression, one or more of the independent variables have to be either continious, categorical or ordinal. In our case we have a mix of continous and categorical so the assumption holds.
- There is no multicollinearity between predictors as we have previously seen.

7.3.2 Build Model

```
library(haven)
library(MASS)
```

Now we can build our model using polr() function with *binned_score* as response variable and using variables *country*, *duration*, *content*, *year*, *critic_review_ratio*, *director_fb* as predictors.

```
#build olr model
olr <- polr(binned_score ~ country + duration + content + year + critic_review_ratio + director_fb, data
```

```
summary(olr)
```

```
## Call:
## polr(formula = binned_score ~ country + duration + content +
##     year + critic_review_ratio + director_fb, data = train, Hess = TRUE)
##
## Coefficients:
##                          Value Std. Error  t value
## countryUSA          -0.8704713  6.843e-02  -12.720
## countryOthers       -0.5331829  5.397e-02   -9.879
## duration             0.0364572  3.146e-03   11.587
## contentNC-17        -0.3672144  4.381e-04 -838.230
## contentPG           -0.8661960  7.441e-02  -11.640
## contentPG-13        -1.0401664  6.054e-02  -17.182
## contentR            -0.2948997  5.794e-02   -5.090
## year                -0.0018815  1.830e-04  -10.279
## critic_review_ratio  0.1209720  7.135e-02    1.696
## director_fb          0.0001338  2.949e-05    4.537
##
## Intercepts:
##              Value     Std. Error t value
## (0,4]|(4,6]    -5.0584    0.0061  -823.6907
## (4,6]|(6,8]    -1.9223    0.1178   -16.3227
## (6,8]|(8,10]    3.0285    0.1863    16.2521
##
## Residual Deviance: 4365.036
## AIC: 4391.036
```

It is possible to see the value of the intercepts, which are sometimes called cutpoints. The intercepts indicate where the latent variable is cut to make the four groups that we observe in our data.

7.3.3 Calculate p-values

The summary() function does not return p-values for Ordinal Logistic Regression, but we can still calculate them by comparing the t-value against the standard normal distribution.

```
#store table
cptable <- coef(summary(olr))

#calculate and store p value
p <- pnorm(abs(cptable[, "t value"]), lower.tail = F) * 2

#combine table
cptable <- cbind(cptable, "p value" = p)
cptable
```

```
##                            Value   Std. Error     t value      p value
## countryUSA          -0.8704712917 6.843078e-02  -12.720465 4.551456e-37
## countryOthers       -0.5331828568 5.397336e-02   -9.878630 5.153257e-23
## duration             0.0364572431 3.146436e-03   11.586836 4.805565e-31
## contentNC-17        -0.3672144009 4.380828e-04 -838.230494 0.000000e+00
## contentPG           -0.8661960470 7.441275e-02  -11.640426 2.567311e-31
## contentPG-13        -1.0401664493 6.053738e-02  -17.182218 3.608452e-66
## contentR             -0.2948997496 5.794116e-02   -5.089642 3.587410e-07
## year                 -0.0018814698 1.830486e-04  -10.278525 8.806525e-25
## critic_review_ratio   0.1209719884 7.134882e-02    1.695501 8.998045e-02
## director_fb           0.0001338174 2.949232e-05    4.537363 5.696196e-06
## (0,4]|(4,6]          -5.0584178834 6.141162e-03 -823.690650 0.000000e+00
## (4,6]|(6,8]          -1.9222786389 1.177671e-01  -16.322719 6.803706e-60
## (6,8]|(8,10]          3.0285119176 1.863463e-01   16.252062 2.159752e-59
```

We can see that all predictors except *critic_review_ratio* are statistically significant. let us build a new model without it.

```
#new model without critic_review_ratio
olr2 <- polr(binned_score ~ country + duration + content + year  + director_fb, data = train, Hess = TRU
```

Again we calculate p-values.

```
#store table
cptable2 <- coef(summary(olr2))

#calculate and store p value
p <- pnorm(abs(cptable2[, "t value"]), lower.tail = F) * 2

#combine table
cptable2 <- cbind(cptable2, "p value" = p)
cptable2
```

```
##                       Value   Std. Error     t value      p value
## countryUSA    -0.8823225139 6.798969e-02  -12.977298 1.645846e-38
## countryOthers -0.5184544597 5.315499e-02   -9.753637 1.779748e-22
## duration       0.0360664582 3.000281e-03   12.021027 2.755270e-33
## contentNC-17  -0.3399124007 3.539386e-04 -960.370975 0.000000e+00
## contentPG     -0.8742270779 7.416185e-02  -11.788097 4.495830e-32
## contentPG-13  -1.0694951855 6.055824e-02  -17.660605 8.433085e-70
## contentR      -0.3025145107 5.803412e-02   -5.212701 1.861112e-07
## year           0.0032446286 1.656788e-04   19.583847 2.123598e-85
## director_fb    0.0001335529 2.948091e-05    4.530149 5.894200e-06
## (0,4]|(4,6]    5.0534861728 3.377055e-03 1496.417986 0.000000e+00
## (4,6]|(6,8]    8.1868712281 1.162061e-01   70.451325 0.000000e+00
## (6,8]|(8,10]  13.1473042591 1.865729e-01   70.467379 0.000000e+00
```

Now all predictors are statistically significant.

The coefficients from the model can be somewhat difficult to interpret because they are scaled in terms of log-odds. Another way to interpret logistic regression models is to convert the coefficients into odds ratios, making exponential log-odds.

```
#add 3 zero is usefull to have a fitted odds_ratio column
odds_ratio <- c(exp(coef(olr2)),c(0,0,0))

#combine table
cptable2 <- cbind(cptable2, "Odds ratio" = odds_ratio)
cptable2
```

```
##                       Value   Std. Error     t value      p value
## countryUSA    -0.8823225139 6.798969e-02  -12.977298 1.645846e-38
## countryOthers -0.5184544597 5.315499e-02   -9.753637 1.779748e-22
## duration       0.0360664582 3.000281e-03   12.021027 2.755270e-33
## contentNC-17  -0.3399124007 3.539386e-04 -960.370975 0.000000e+00
## contentPG     -0.8742270779 7.416185e-02  -11.788097 4.495830e-32
## contentPG-13  -1.0694951855 6.055824e-02  -17.660605 8.433085e-70
## contentR      -0.3025145107 5.803412e-02   -5.212701 1.861112e-07
## year           0.0032446286 1.656788e-04   19.583847 2.123598e-85
## director_fb    0.0001335529 2.948091e-05    4.530149 5.894200e-06
## (0,4]|(4,6]    5.0534861728 3.377055e-03 1496.417986 0.000000e+00
## (4,6]|(6,8]    8.1868712281 1.162061e-01   70.451325 0.000000e+00
## (6,8]|(8,10]  13.1473042591 1.865729e-01   70.467379 0.000000e+00
##               Odds ratio
## countryUSA     0.4138207
## countryOthers  0.5954401
## duration       1.0367247
## contentNC-17   0.7118327
## contentPG      0.4171843
## contentPG-13   0.3431817
## contentR       0.7389578
## year           1.0032499
## director_fb    1.0001336
## (0,4]|(4,6]    0.0000000
## (4,6]|(6,8]    0.0000000
## (6,8]|(8,10]   0.0000000
```

New coefficients are called proportional odds ratios and we would interpret these pretty much as we would odds ratios from a binary logistic regression. For example we will say that for a unit increase in duration, it means an increment of 1.03 of the odds that is $p(X)/1 - p(x)$, given all the other variables of the model held constant.

7.3.4 Apply Model

We can now apply our model on the *train* and *test* set.

```
#predict train set
olr2.pred <- predict(olr2, train)
confusionMatrix(olr2.pred, train$binned_score)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     27   171   114      0
##     (6,8]     40   667  1792     89
##     (8,10]     0     1    13     11
##
## Overall Statistics
##
##                 Accuracy : 0.6749
##                   95% CI : (0.6576, 0.6918)
##      No Information Rate : 0.6561
##      P-Value [Acc > NIR] : 0.01664
##
##                    Kappa : 0.1634
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000      0.20381       0.9338      0.110000
## Specificity               1.00000      0.93241       0.2087      0.995044
## Pos Pred Value                NaN      0.54808       0.6924      0.440000
## Neg Pred Value            0.97709      0.74436       0.6231      0.969310
## Prevalence                0.02291      0.28684       0.6561      0.034188
## Detection Rate            0.00000      0.05846       0.6126      0.003761
## Detection Prevalence      0.00000      0.10667       0.8848      0.008547
## Balanced Accuracy         0.50000      0.56811       0.5713      0.552522
```

```
#predict test set
olr2.pred <- predict(olr2, test)
confusionMatrix(olr2.pred, test$binned_score)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     13    42    40      0
##     (6,8]     11   167   424     30
##     (8,10]     0     0     4      1
##
## Overall Statistics
##
##                Accuracy : 0.638
##                  95% CI : (0.602, 0.6729)
##     No Information Rate : 0.6393
##     P-Value [Acc > NIR] : 0.5474
##
##                   Kappa : 0.1184
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000      0.20096       0.9060      0.032258
## Specificity               1.00000      0.89866       0.2121      0.994294
## Pos Pred Value                NaN      0.44211       0.6709      0.200000
## Neg Pred Value            0.96721      0.73783       0.5600      0.958735
## Prevalence                0.03279      0.28552       0.6393      0.042350
## Detection Rate            0.00000      0.05738       0.5792      0.001366
## Detection Prevalence      0.00000      0.12978       0.8634      0.006831
## Balanced Accuracy         0.50000      0.54981       0.5591      0.513276
```

Our Ordinal Logistic Regression model performs with an accuracy of:

- Train Set: 0.6749
- Test Set: 0.638

7.4 K-Nearest Neighbors

K-Nearest Neighbors is a non-parametric method used for classification and regression. KNN is a lazy learner, meaning that computation is deferred until the actual classification task.

Simply given a positive integer K and a test observation, the KNN classifier first identifies the K training points closest to our test observation based on the Euclidean Distance metric. Then it assign a class based on the most occuring class of its K nearest neighbors.

K represents the number of neighbors in the training set closest to the test observation. The choice of K has a drastic effect on the KNN model. K = 1 (Nearest Neighbors) leads to a too flexible decision boundary, while K too large can lead to a tightening of the decision boundaries and classyfing everything as the most occuring class in our dataset. Our approach in finding the optimal value for K is trying different values and choosing the one which maximize the predictive accuracy of the model.

7.4.1 Dummy variabels and Scaling

In order to apply KNN we have to create dummy variabels for our categorical variables: *country* and *content*.

```
#copy dataframe
IMDB2 <- IMDB
IMDB2$country <- as.factor(IMDB2$country)
IMDB2$content <- as.factor(IMDB2$content)

#dummy for country - creates a boolean vector for every level of country
IMDB2[,c("country_UK", "country_USA", "country_Others")] <- model.matrix( ~ country - 1, data = IMDB2)

#dummy for content - - creates a boolean vector for every level of content
IMDB2[,c("content_G", "content_NC17", "content_PG", "content_PG13", "content_R")] <- model.matrix( ~ co

#remove categorical variables
IMDB2 <- IMDB2[,-c(1,2,14)]
```

Then we have to create *train* and *test* set for our dataframe copy.

```
#train and test set
train2 <- IMDB2[sample,]
test2 <- IMDB2[-sample,]
```

Since KNN normally uses Euclidean distance we need to normalize our variables so that they are in the same scale.

```
train2.std <- scale(train2[,-12])
test2.std <- scale(test2[,-12])
```

7.4.2 Find best K

To find the optimal value of K we run the algorithm trying k = 1:20.

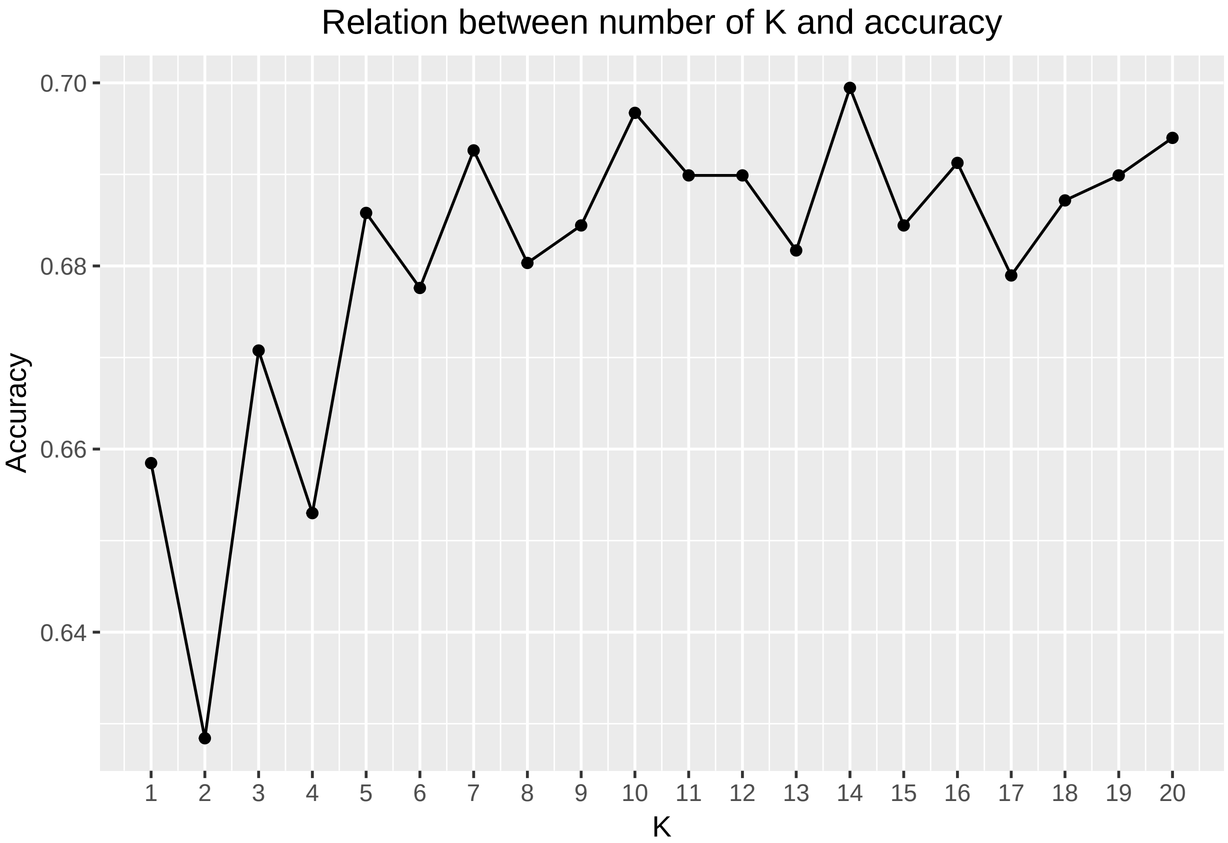
```
library(class)
set.seed(12)

#vector where we store accuracies
knn.acc <- c()

#knn with k 1:20
for (i in 1:20) {
  knn.pred <- knn(train2.std, test2.std, train2$binned_score, k = i)
  #save accuracy in a vector
  knn.acc[i] <- mean(knn.pred == test$binned_score)
}
```

```
knn.acc <- cbind(seq(1,20,1), knn.acc)
knn.acc <- as.data.frame(knn.acc)
```

```
ggplot(knn.acc, aes(x=V1, y=knn.acc)) +
  scale_x_continuous(breaks = seq(1,20,1)) +
  labs(x="K", y="Accuracy", title = "Relation between number of K and accuracy") +
  geom_point() +
  geom_line() +
  theme(plot.title = element_text(hjust = 0.5))
```



K = 14 gives best accuracy.

7.4.3 Apply model

Now that we know the value for k that maximize the accuracy we can apply our model on the *test* set.

```
set.seed(12)
library(class)
knn.pred.test <- knn(train2.std, test2.std, train2$binned_score, k=14)
```

```
confusionMatrix(knn.pred.test, test2$binned_score)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     16   102    69      2
##     (6,8]      8   107   399     21
##     (8,10]     0     0     0      8
##
## Overall Statistics
##
##                Accuracy : 0.6954
##                  95% CI : (0.6606, 0.7285)
##     No Information Rate : 0.6393
##     P-Value [Acc > NIR] : 0.0008085
##
##                   Kappa : 0.3356
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.4880       0.8526       0.25806
## Specificity               1.00000       0.8337       0.4848       1.00000
## Pos Pred Value                NaN       0.5397       0.7458       1.00000
## Neg Pred Value            0.96721       0.8029       0.6497       0.96823
## Prevalence                0.03279       0.2855       0.6393       0.04235
## Detection Rate            0.00000       0.1393       0.5451       0.01093
## Detection Prevalence      0.00000       0.2582       0.7309       0.01093
## Balanced Accuracy         0.50000       0.6608       0.6687       0.62903
```

Test set accuracy = 0.6954

7.5 Support Vector Machine

Support Vector Machine is a supervised machine learning algorithm that aims to find a N-dimensional hyperplane that distinctly classifies the data points.

To separate two classes of data points, there are many possible hyperplane that could be chosen, SVM finds the plane that has the maximum margin: the separating hyperplane farthest from the training observation of both classes. The margins are found by calculating the smallest perpendicular distance from the hyperplane to the training data. The test observation is classified on the basis of which part of the maximal margin it falls.

Support vectors are the training observations in the p-dimensional space that are closer to the hyperplane and influence the position and orientation of the hyperplane. In order to create a more robust classifier, we can allow some observations to violate the margin, this is called *Support Vector Classifier* (or *Soft Margin Classifier*).

In many cases it is not possible to linearly separete classes. SVM applies the same approach as in linear regression: modiyfing the feature space by adding high order polynomial or other transformation to achieve non-linearity. This is called *kernel trick*: a *kenrel* is a function that quantifies the similarity of two observations. The idea is that we want to enlarge our feature space in order to accomodate a non-linear boundary between the classes. Formally, a Support Vector Classifier is a Support Vector Machine with a linear Kernel. It is called Support Vector Machine only when it does not use a linear Kernel and thus does not have linear decision boundaries.

SVM can also be used for multi-class classification. The approach is to reduce a single multi-class problem into multiple binary classification problems. There are two ways to achieve this:

- *one-vs-one*: distinguish between every pair of classes
- *one-vs-all*: distinguish between one class and all of the others.

In our case we use a multiclass SVM with 10-fold cross validation approach that allows to choose the model with the best cost. The kernel that led to the best results is the radial kernel.

7.5.1 Build Model

We fit a SVM with the function *svm()* using *binned_score* as response variable and *year*, *duration*, *budget*, *gross* and *user_vote* as independent variable.

```
library(e1071)
svm.fit <- svm(binned_score ~ year + duration + budget + gross + user_vote + movie_fb, data=train,kernel
```

7.5.2 Cross-Validation & Parameter tuning

By using function *tune()* we can use cross-validation to get the best cost for our model.

```
#cross-validation
tune.out <- tune(svm, binned_score ~ year + duration + budget + gross + user_vote + movie_fb, data=train
```

```
tune.out$performances
```

```
##    cost     error dispersion
## 1 1e-03 0.3439268 0.01590773
## 2 1e-02 0.3439268 0.01590773
## 3 1e-01 0.3223865 0.01770959
## 4 1e+00 0.2707677 0.02461302
## 5 5e+00 0.2577715 0.02714355
## 6 1e+01 0.2564028 0.02929081
## 7 1e+02 0.2594874 0.02943962
```

We can see that cost=10 results in the best performance (error=0.2564028).

```
#select the best model
bestmod <- tune.out$best.model
```

7.5.3 Apply Model

We can now apply the model we just created on the *test* dataset.

```
svm.pred <- predict(bestmod, test)
confusionMatrix(svm.pred, test$binned_score)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction (0,4] (4,6] (6,8] (8,10]
##     (0,4]      0     0     0      0
##     (4,6]     15   106    47      0
##     (6,8]      9   103   419     11
##     (8,10]     0     0     2     20
##
## Overall Statistics
```

```
##
##                Accuracy : 0.7445
##                  95% CI : (0.7113, 0.7758)
##     No Information Rate : 0.6393
##     P-Value [Acc > NIR] : 7.605e-10
##
##                   Kappa : 0.4444
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: (0,4] Class: (4,6] Class: (6,8] Class: (8,10]
## Sensitivity               0.00000       0.5072       0.8953       0.64516
## Specificity               1.00000       0.8815       0.5341       0.99715
## Pos Pred Value                NaN       0.6310       0.7731       0.90909
## Neg Pred Value            0.96721       0.8174       0.7421       0.98451
## Prevalence                0.03279       0.2855       0.6393       0.04235
## Detection Rate            0.00000       0.1448       0.5724       0.02732
## Detection Prevalence      0.00000       0.2295       0.7404       0.03005
## Balanced Accuracy         0.50000       0.6943       0.7147       0.82115
```

Support Vector Machine performs with an accuracy of: 0.7445

8. Conclusions

Algorithm	Train set Accuracy	Test set Accuracy
Classification Tree	0.7764	0.7104
Random Forest		0.7732
Ordinal Logistic Regresion	0.6749	0.6380
KNN		0.6954
SVM		0.7445

We implemented five different algorithms in order to classify movies in four categories. From the confusion matrix of our models we can see that every model had problems in classifying movies belonging to the **bad** (0,4] category.

Another common misclassification that is happening in all our models is between category **Mediocre** (4,6] and **Good** (6,8]: all our models are overestimating some movies, classifying them as **Good** (6,8] while they actually are **Mediocre** (4,6].

Random Forest is the model that yields better results with an accuracy of 77%, followed by *Support Vector Machine* with an accuracy of 0.7445. *Classification Tree* and *K-Nearest Neighbors* are both at around 70% accuracy and *Ordinal Logistic Regressio*n is the worse model with an accuracy of 64%.