CSDA Group 1 Lab 1

Group 1

21/01/2020

Business Understanding

Business Objective

Background:

The modern consumer has been shaped and changed over-time by our ever-advancing technologically apt society. A consumer can be defined as an individual or organization that purchases or uses the services or commodities; rather than engaging in thorough and time-invested activities to find that 'one' item or service, the consumer today has a plethora of personalized and targeted content on hand. Whether they are pressed on time, shopping and browsing on-the-go, or logging in from multiple devices, companies and brands have managed to deliver rapid and on-point offers to capture interest in a competitive economy. Brands providing rapid, on-point recommendations capture the most benefits. Segment, a customer data infrastructure company, found in their 2017 report that personalization drives revenue, loyalty, and increased interactions with the company. (Segment,2017)

Machine learning systems allow the capture of vital consumer data, manipulate into meaningful insights, and display the best recommendations for users. Some examples include; highlighting different versions of your store homepage or product marketing material to different customer segments, pinpoint which inventory to showcase, scan through the product catalogue and line up the best products and even recommend the nearest locations or sites they can make the purchase, and smart systems can even capture browsing data then later target the prospect with ads.

Movie4You is an app founded in May 2015 with headquarters in Toronto, Ontario, where users can watch movies and TV shows online or stream to their smart TV, PC, mobile and tablet. They last updated their movie database in July 2017, where a wide selection of movies and TV shows can be found across several genres. In the first year of operation, they had 600,000 paid subscribers, but at the end of the second year, they only had 400,000 paid subscribers.

Business Objective:

The marketing department of Movie4You has determined that although having an extensive database of movies and TV shows, users do not know what they should watch next and eventually end their subscriptions. Group 1 has been retained to help Movie4You develop a recommendation system to predict movies for users based on their preferences and deploy it in a user-friendly interface.

Business Success Criteria:

Business objective: Development and deployment of a movie recommendation engine. Stakeholders Project Success Criteria Measurement Priority

```
Developers        Obtain and clean dataset.     1)Have all the
                                                appropriate fields
                                                of data labeled
for a supervised
                                                learning model.

                                                                       1
                                                2)Ensure adequate
                                                data is  retained
                                                after cleaning to
                                                train and test.

                  Train and test three models.  Each model should
                                                make at least 3        2
predictions.

                  Integrate best model into     User interface
user interface.              should include model   3
that makes at least
                                                3 predictions.

UI Planning &     Understand the variables,     Clear, quantifiable,           Design
inputs and outputs of the     definition of what                               model.
will be required from  4                                                       the user and what
outputs are produced.

                  Develop framework and         Develop framework for
suggest platform for          an app in R Shiny or   5
                  easiest application usage.    Dash.

User              Uses application for movie    Retention. Returns to
                  movie recommendations.        the application more   6
                                                than once for a new
                                                recommendation.
```

Proposed drivers and relationships to inputs:

To aid in developing and testing our model, we anticipate the input variables ratings, userId, movieId and movie titles to affect the outcome of our model.

Output:

There are two ways to measure our model's ability to generate movie outputs, retention and engagement. One core benefit of a recommendation system is to calibrate the user's preferences, known as "improving with use." For example, Netflix competitors would deal with the challenge of attracting their customer base due to Netflix's already 'knowing them well' by retaining the user preference. For a streaming service whose revenue relies mainly on the fixed-rate billing subscription model, users must be comfortable in their personalized content space.

The second measure is engagement. Simply encouraging activity on our platform can allow our product to gain attention and permit the opening of revenue-generating streams such as advertisement opportunities. An excellent example of such practices being employed is by the ever-popular YouTube and Facebook. The search engine results page is the right place for advertisers to place their content. Our product's movie recommendations should encourage users to return to the page and recommend our application to others. Deploying such a tool in a browser or blog page can allow advertisers to place targeted ads.

Assumptions:

1. It's important to note that our recommendation system will likely only work with enough data, dependent on the dataset after cleaning.
2. For a collaborative filtering model, we would need the input of many other users in the form of ratings to identify the right movie prediction.

Ethical Machine Learning Framework

Problem Definition and Scope

Map current business process:

Our system adds value to the addition stage of the process. The user would manually search for movies that they would enjoy watching by self-filtration of preferences such as genre or highest rated movies. However, we can turn this into a funnel type flow. An amalgamation of the user, highest movie ratings, and a dataset churns a movie recommendation depending on the machine learning model set in place. This removes the user's manual work and, in advanced applications, can make them part of the automatic system (such as human-in-the-loop systems).

Inputs, Outputs, Optimization, and Baseline Performance:

Regardless of model type, this recommender system will look at the group of movies defined by Id number and the associated ratings. We only considered who have rated at least 30 movies and movies that have been viewed at least 40 times for our matrix. As an output, we determine the top five movie recommendations for each new user. The goal is to find the best recommendations for the user by ensuring optimal data distribution in the train and test datasets. The baseline performance will be measured by the number of movies recommended. In this case, it should be 5.

Risks and Ethics:

Our system can negatively impact an individual if their movie recommendation is entirely off the expectations that were already held with the product. For example, if a user has a particular set of genres in mind, they usually avoid watching. The recommendation list suggests a genre from such categories; they might not return to use the product. No one is left vulnerable by the application of our recommender system. We can accept one wrong recommendation per model. The list is generously extensive for a single run, outputting five movies. This gives room for one movie to be an 'off-point suggestion.' The movies grouped by the ratings have the most significant influence on outputs rather than the genre of movies.

Design

The following is a snippet from contributor Rounak Banik on Kaggle. "These files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages. This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website." In this case, the user ratings dataset size is 100,000 ratings from 700 users on 9,000 movies. Our design is a fully-automated system with static data that required an initial human interaction with the application to generate a recommendation.

Derived insights, action and transparency:

The outputs, movie recommendations will generate the following insights. It will highlight some patterns in the recommended lists that are created after every click for the user. This may lead the user to agree strongly with themselves the movies they prefer watching. For the developers, it can highlight some important characteristics of the recommendation system, such as feature selection and differences in models.

Our application highlights the dataset being used for the recommendation in the ‘About’ panel. This will indicate the selection of movies, ratings, and cut off at around mid-year 2017. The dataset provided by MovieLens would have some biases in the user field. These were controlled in the data preparation set as much as possible to create an optimal training dataset. The movies were grouped by rating, but some user biases in rating the movies remain in the underlayer. The movie recommendations are strictly available for the end-user and developers to view. Our product does not loop in human input for the recommendation, thus ensuring their data’s safety. The user will be aware of launching the application that the system is a recommendation engine based on machine learning principles. Information is listed in the ‘About’ tab.

Data Collection and Retention

Procurement and ethical questions:

We have procured a single dataset only.No external expertise was obtained. Our models were trained and tested based on the developers’ features deemed most appropriate, given the dataset. The user field is de-identified, and we have taken measures to include the ratings and movie titles in our models solely. As discussed above, socially sensitive features do not influence output aside from the underlying user bias in the ratings dataset.

Data Processing

Model Prototyping and Quality

The following three models were experimented with; User-Based Collaborative Filtering, Item-Based Collaborative Filtering, and Singular Value Decomposition.

Here are some further details on the three models;

User-based collaborative filtering: No domain knowledge is required and allows users to find new interests. However, it doesn’t address the cold-start problem of a new user or item entering the system. Sparse data can make it difficult to find users that have rated the same items. It can’t deal with sparse data, meaning it’s hard to find users that have rated the same items, and it tends to recommend popular items.

Item-based collaborative filtering: It can recommend to users with unique tastes or new & unpopular items. It can further provide explanations for the recommended items by listing item-features that caused the recommendation. However, finding the appropriate features is difficult, and it does not recommend items outside a user’s content profile. Unable to exploit the quality judgments of other users and expand outside of the user’s current interests.

Singular Value Decomposition: In the context of recommendation systems, SVD is used as a collaborative filtering algorithm. It is a technique that reduces the number of features of a dataset by reducing space dimensions. (Malaeb, 2016) This method is very efficient for large sparse matrixes.

The dataset was split in the following way: 90% for training and 10% for testing to ensure the performance was not skewed beyond the training set. Our use case does not require a more interpretable algorithm. We have included all three algorithms in the application for user knowledge. Further information about the algorithms can be found in the ‘About’ tab. The only underlying bias that can be malicious is the MovieLens dataset’s nature in procuring users for rating movies. This has been discussed above. The impact of it on the training data is reduced by implementing appropriate data sparsity in the rating matrix.

Deployment, Monitoring and Maintainance

They are completed in the form of an R shiny application. Systems design will ensure only authorized members can train the model through authentication and firewall security.

Data Mining Objective

The data mining goal is to create a recommendation model that can predict movies for users/subscribers based on ratings they have provided for movies they watched.

Data Understanding

We chose to evaluate the MovieLens Dataset on Kaggle, which contains 45,000 movies released on or before July 2017. The dataset also includes a rating dataset where the ratings scale from 1-5 on 100,000 ratings from 700 users on 9,000 movies. This dataset is a combination of data collected from The Movie Database (TMDb) and GroupLens. The Ratings were collected from the official GroupLens website.

Load the datasets into R

```
library(readr)
movies <- read_csv("movies_metadata.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   adult = col_logical(),
##   budget = col_double(),
##   id = col_double(),
##   popularity = col_double(),
##   release_date = col_date(format = ""),
##   revenue = col_double(),
##   runtime = col_double(),
##   video = col_logical(),
##   vote_average = col_double(),
##   vote_count = col_double()
## )
```

```
## See spec(...) for full column specifications.
## Warning: 19 parsing failures.
##   row          col               expected                           actual                 f
ile
## 19730 NA           24 columns             10 columns                       'movies_metadata.c
sv'
## 19731 adult        1/0/T/F/TRUE/FALSE     - Written by Ørnås               'movies_metadata.c
sv'
## 19731 budget       a double               /ff9qCepilowshEtG2GYWwzt2bs4.jpg 'movies_metadata.c
sv'
## 19731 id           no trailing characters -08-20                           'movies_metadata.c
sv'
## 19731 release_date date like              1                                'movies_metadata.c
sv'
## ..... ............ ...................... ................................
....................
## See problems(...) for more details.
```

```
ratings <- read_csv("ratings_small.csv")
```

```
## Parsed with column specification:
## cols(
##   userId = col_double(),
##   movieId = col_double(),
##   rating = col_double(),
##   timestamp = col_double()
## )
```

There are 45,466 observations in the movies dataset and 24 variables which are:

- adult
- belongs_to_collection
- budget
- genres
- homepage
- id
- imdb_id
- original_language
- original_title
- overview
- popularity
- poster_path
- production_companies
- production_countries
- release_date
- revenue
- runtime
- spoken_languages
- status
- tagline
- title

- video
- vote_average
- vote_count

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
glimpse(movies)
```

```
## Observations: 45,466
## Variables: 24
## $ adult                 <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE…
## $ belongs_to_collection <chr> "{'id': 10194, 'name': 'Toy Story Collection', …
## $ budget                <dbl> 30000000, 65000000, 0, 16000000, 0, 60000000, 5…
## $ genres                <chr> "[{'id': 16, 'name': 'Animation'}, {'id': 35, '…
## $ homepage              <chr> "http://toystory.disney.com/toy-story", NA, NA,…
## $ id                    <dbl> 862, 8844, 15602, 31357, 11862, 949, 11860, 453…
## $ imdb_id               <chr> "tt0114709", "tt0113497", "tt0113228", "tt01148…
## $ original_language     <chr> "en", "en", "en", "en", "en", "en", "en", "en",…
## $ original_title        <chr> "Toy Story", "Jumanji", "Grumpier Old Men", "Wa…
## $ overview              <chr> "Led by Woody, Andy's toys live happily in his …
## $ popularity            <dbl> 21.946943, 17.015539, 11.712900, 3.859495, 8.38…
## $ poster_path           <chr> "/rhIRbceoE9lR4veEXuwCC2wARtG.jpg", "/vzmL6fP7a…
## $ production_companies  <chr> "[{'name': 'Pixar Animation Studios', 'id': 3}]…
## $ production_countries  <chr> "[{'iso_3166_1': 'US', 'name': 'United States o…
## $ release_date          <date> 1995-10-30, 1995-12-15, 1995-12-22, 1995-12-22…
## $ revenue               <dbl> 373554033, 262797249, 0, 81452156, 76578911, 18…
## $ runtime               <dbl> 81, 104, 101, 127, 106, 170, 127, 97, 106, 130,…
## $ spoken_languages      <chr> "[{'iso_639_1': 'en', 'name': 'English'}]", "[{…
## $ status                <chr> "Released", "Released", "Released", "Released",…
## $ tagline               <chr> NA, "Roll the dice and unleash the excitement!"…
## $ title                 <chr> "Toy Story", "Jumanji", "Grumpier Old Men", "Wa…
## $ video                 <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE…
## $ vote_average          <dbl> 7.7, 6.9, 6.5, 6.1, 5.7, 7.7, 6.2, 5.4, 5.5, 6.…
## $ vote_count            <dbl> 5415, 2413, 92, 34, 173, 1886, 141, 45, 174, 11…
```

```
summary(movies)
```

```
##    adult          belongs_to_collection     budget             genres         
##  Mode :logical   Length:45466          Min.   :        0   Length:45466      
##  FALSE:45454     Class :character      1st Qu.:        0   Class :character  
##  TRUE :9         Mode  :character      Median :        0   Mode  :character  
##  NA's :3                               Mean   :  4224579                     
##                                        3rd Qu.:        0                     
##                                        Max.   :380000000                     
##                                        NA's   :3                             
##    homepage               id           imdb_id          original_language 
##  Length:45466       Min.   :     2   Length:45466       Length:45466      
##  Class :character   1st Qu.: 26450   Class :character   Class :character  
##  Mode  :character   Median : 60003   Mode  :character   Mode  :character  
##                     Mean   :108360                                        
##                     3rd Qu.:157328                                        
##                     Max.   :469172                                        
##                     NA's   :3                                             
##  original_title       overview           popularity       poster_path       
##  Length:45466       Length:45466       Min.   :  0.0000   Length:45466      
##  Class :character   Class :character   1st Qu.:  0.3859   Class :character  
##  Mode  :character   Mode  :character   Median :  1.1277   Mode  :character  
##                                        Mean   :  2.9215                     
##                                        3rd Qu.:  3.6789                     
##                                        Max.   :547.4883                     
##                                        NA's   :6                            
##  production_companies production_countries  release_date       
##  Length:45466         Length:45466         Min.   :1874-12-09  
##  Class :character     Class :character     1st Qu.:1978-10-06  
##  Mode  :character     Mode  :character     Median :2001-08-30  
##                                            Mean   :1992-05-15  
##                                            3rd Qu.:2010-12-17  
##                                            Max.   :2020-12-16  
##                                            NA's   :90          
##     revenue             runtime        spoken_languages      status         
##  Min.   :0.000e+00   Min.   :   0.00   Length:45466       Length:45466      
##  1st Qu.:0.000e+00   1st Qu.:  85.00   Class :character   Class :character  
##  Median :0.000e+00   Median :  95.00   Mode  :character   Mode  :character  
##  Mean   :1.121e+07   Mean   :  94.13                                        
##  3rd Qu.:0.000e+00   3rd Qu.: 107.00                                        
##  Max.   :2.788e+09   Max.   :1256.00                                        
##  NA's   :6           NA's   :263                                            
##    tagline             title              video          vote_average   
##  Length:45466       Length:45466       Mode :logical   Min.   : 0.000  
##  Class :character   Class :character   FALSE:45367     1st Qu.: 5.000  
##  Mode  :character   Mode  :character   TRUE :93        Median : 6.000  
##                                        NA's :6         Mean   : 5.618  
##                                                        3rd Qu.: 6.800  
##                                                        Max.   :10.000  
##                                                        NA's   :6       
##    vote_count     
##  Min.   :    0.0  
##  1st Qu.:    3.0  
##  Median :   10.0  
##  Mean   :  109.9  
##  3rd Qu.:   34.0
##  Max.   :14075.0
##  NA's   :6
```

There are 100,004 observations in the ratings dataset and 4 variables which are:

- userId
- movieId
- rating
- timestamp

```
glimpse(ratings)
```

```
## Observations: 100,004
## Variables: 4
## $ userId    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,…
## $ movieId   <dbl> 31, 1029, 1061, 1129, 1172, 1263, 1287, 1293, 1339, 1343, 1…
## $ rating    <dbl> 2.5, 3.0, 3.0, 2.0, 4.0, 2.0, 2.0, 2.0, 3.5, 2.0, 2.5, 1.0,…
## $ timestamp <dbl> 1260759144, 1260759179, 1260759182, 1260759185, 1260759205,…
```

```
summary(ratings)
```

```
##      userId        movieId           rating        timestamp        
##  Min.   :  1   Min.   :     1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:182   1st Qu.:  1028   1st Qu.:3.000   1st Qu.:9.658e+08  
##  Median :367   Median :  2406   Median :4.000   Median :1.110e+09  
##  Mean   :347   Mean   : 12549   Mean   :3.544   Mean   :1.130e+09  
##  3rd Qu.:520   3rd Qu.:  5418   3rd Qu.:4.000   3rd Qu.:1.296e+09  
##  Max.   :671   Max.   :163949   Max.   :5.000   Max.   :1.477e+09
```

Highest Budget Movies

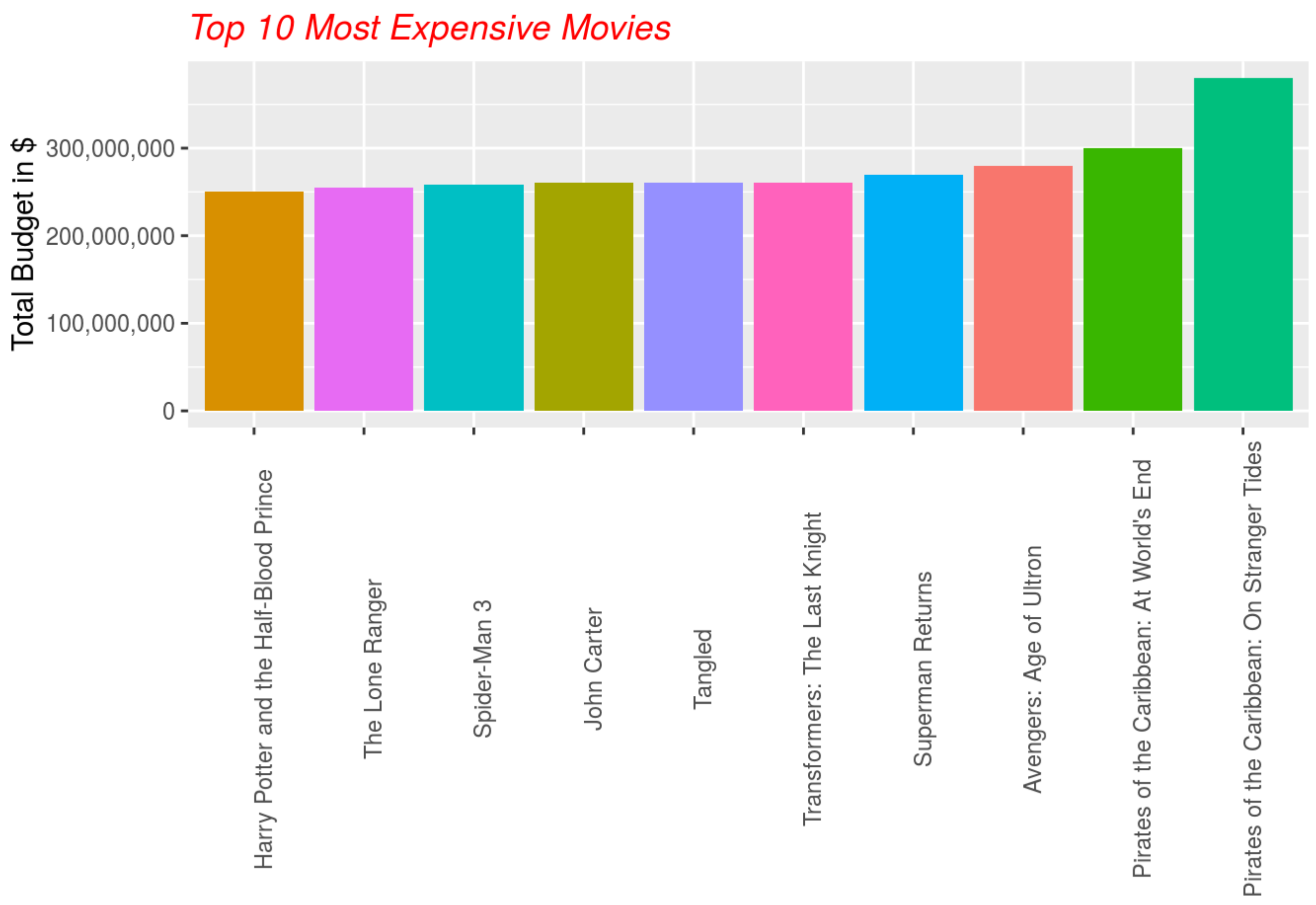
```
library(tidyverse)
```

```
## ── Attaching packages ───────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.2.1     ✓ purrr   0.3.3
## ✓ tibble  2.1.3     ✓ stringr 1.4.0
## ✓ tidyr   1.0.2     ✓ forcats 0.4.0
```

```
## ── Conflicts ──────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

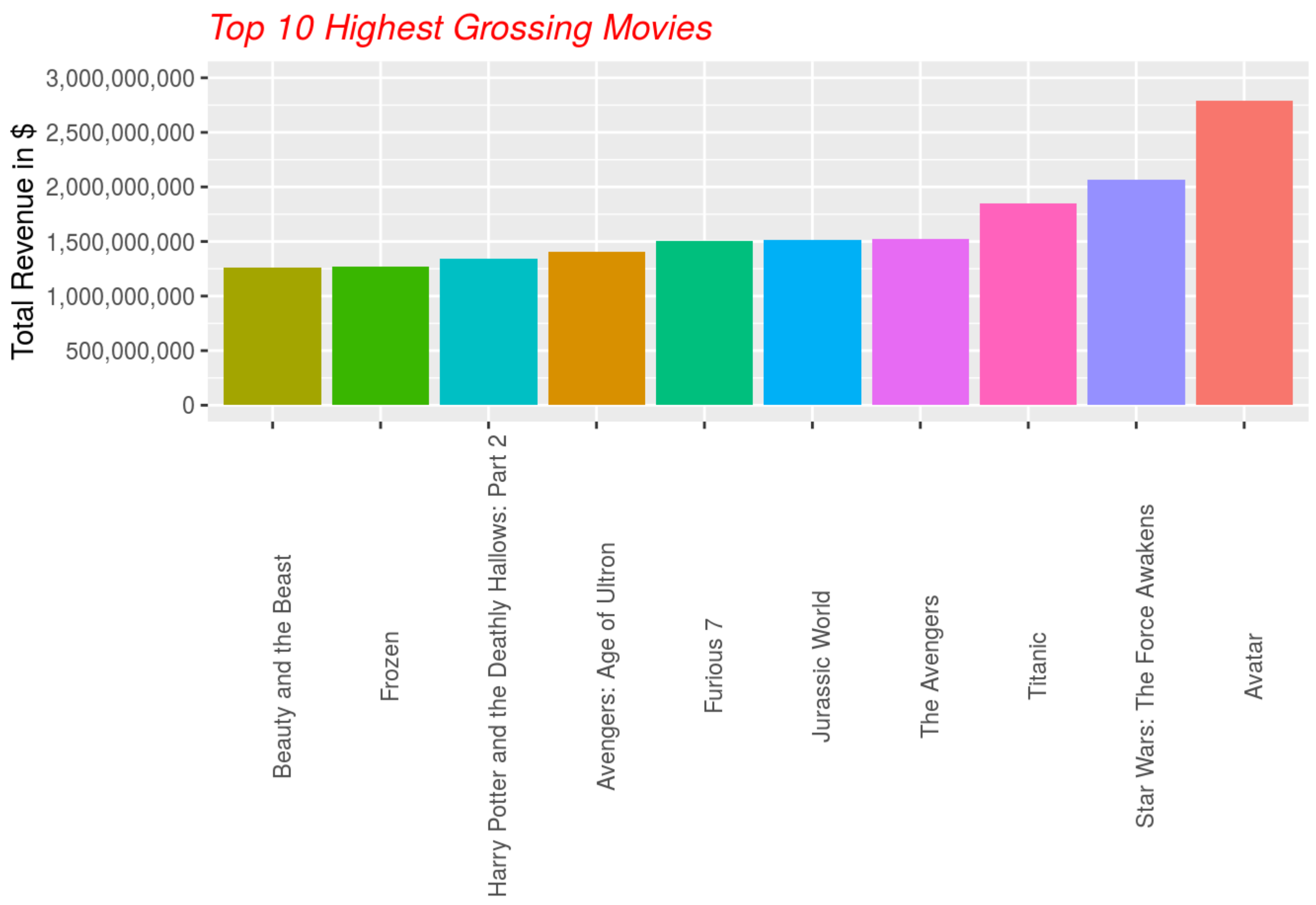
```
movies %>% select(title,budget) %>% drop_na(title)%>% arrange(desc(budget)) %>% head(10) %>%  gg
plot(aes(reorder(title,budget),budget,fill=title))+geom_bar(stat="identity")+theme(axis.text.x =
element_text(angle=90),plot.title=element_text(color="red",face="italic"),legend.position="none"
)+scale_y_continuous(labels=scales::comma)+labs(x="",y="Total Budget in $",title="Top 10 Most Ex
pensive Movies")
```



The bar graph above shows the top 10 most expensive movies to make based on its budget. These movies were all made within the last 20 years, which tells us that movies are becoming more costly with advanced technology. Also, it is interesting to note that inflation was not considered here, just the cost to make that specific movie at that time.

Highest Revenue Movies

```
movies %>% select(original_title,revenue) %>% drop_na(original_title)%>% arrange(desc(revenue))
%>% head(10)  %>% ggplot(aes(reorder(original_title,revenue),revenue,fill=original_title))+geom_
bar(stat="identity")+theme(axis.text.x = element_text(angle=90),plot.title=element_text(color="R
ed",face="italic"),legend.position="none")+scale_y_continuous(limits=c(0,3000000000),breaks=seq(
0,3000000000,500000000),labels=scales::comma)+labs(x="",y="Total Revenue in $",title="Top 10 Hig
hest Grossing Movies")
```

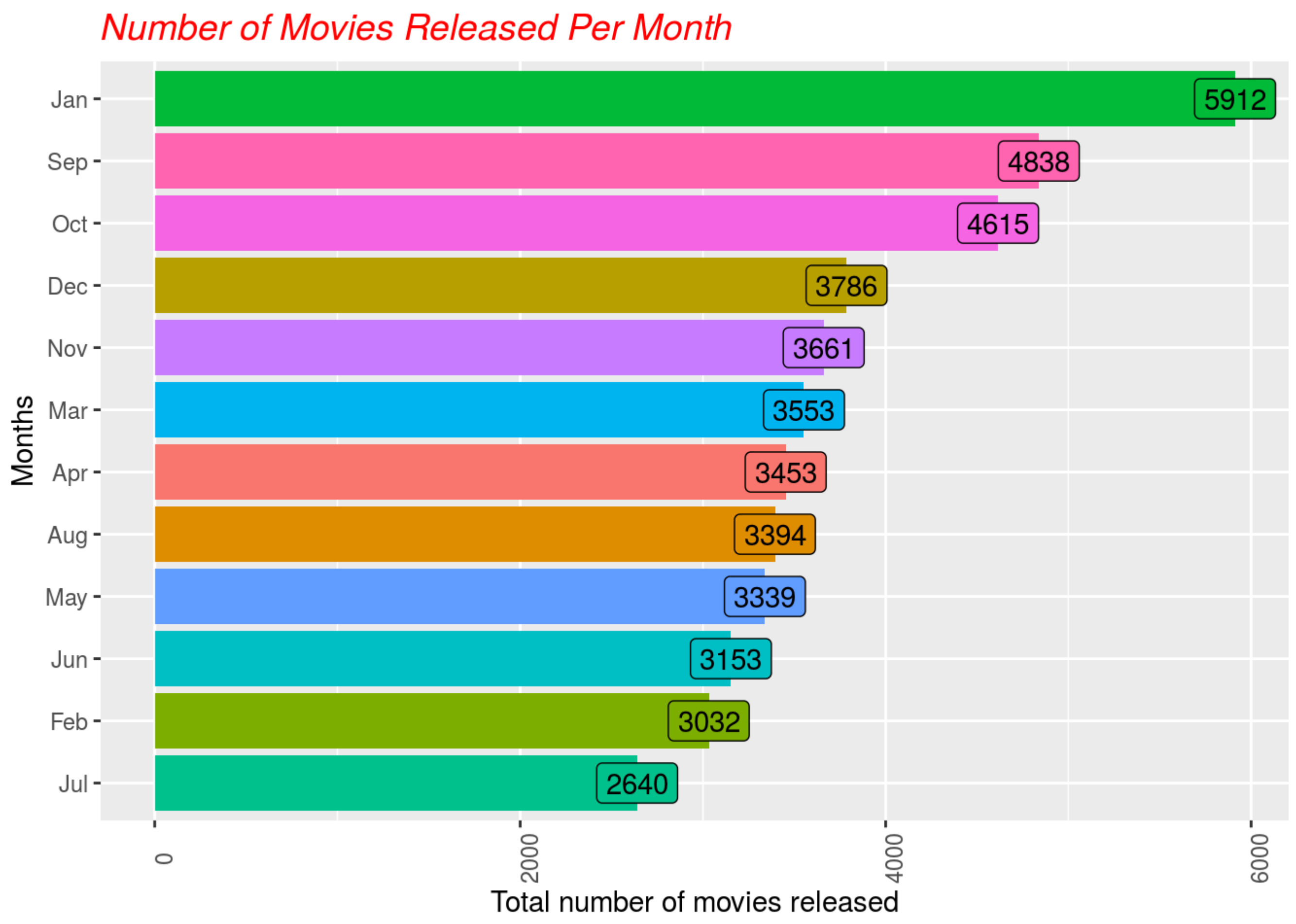


The bar graph above shows the top 10 highest-grossing movies based on its revenue acquired. These movies were all released in the last 25 years, which shows that more people are going to the cinema now than before, but we also didn't consider inflation but just the amount of revenue that money acquired at that time.

Movie release month

```
movies$month=month.abb[as.factor(format(as.Date(movies$release_date, format = "%Y-%m-%d"),"%m"
))]
movies %>% group_by(month) %>% drop_na(month) %>% summarise(count=n()) %>% arrange(desc(month))
 %>% ggplot(aes(reorder(month,count),count,fill=month))+geom_bar(stat="identity")+theme(plot.tit
le=element_text(size=14,face="italic",colour="red"),axis.text.x = element_text(angle=90),legend.
position="none")+labs(x="Months",y="Total number of movies released",title="Number of Movies Rel
eased Per Month")+coord_flip()+geom_label(aes(label=count))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



The horizontal bar graph above shows the number of movies released per month. As one can see, the highest number of movies released in a month is January with 5912 movies. This can be due to it being the first month of the new year, and also it is during the winter season where many people are indoors. July is seen as the lowest number of movie releases, with 2640 movies released. This can be because it is the summer season, where many people are occupied with outdoor activities.

Data Preparation

We eliminated duplicated entries in the movies dataset with the following R code. 17 duplicated entries were found and deleted.

```
nrow(movies)
```

```
## [1] 45466
```

```
movies <- movies[!duplicated(movies), ]
nrow(movies)
```

```
## [1] 45449
```

We also eliminated duplicated entries for the ratings dataset using the following R code. No duplicate entries were found.

```
nrow(ratings)
```

```
## [1] 100004
```

```
ratings <- ratings[!duplicated(ratings), ]
nrow(ratings)
```

```
## [1] 100004
```

Looking for missing values in each column of the movies dataset. A lot of missing values in the ‘runtime’ column but we aren’t interested in this column so we can ignore this.

```
colSums(sapply(movies, is.na))
```

```
##                 adult belongs_to_collection                budget
##                     3                 40956                     3
##                genres              homepage                    id
##                     0                 37669                     3
##               imdb_id     original_language        original_title
##                    17                    11                     0
##              overview            popularity           poster_path
##                   959                     6                   386
##  production_companies  production_countries          release_date
##                     3                     3                    90
##               revenue               runtime      spoken_languages
##                     6                   263                     6
##                status               tagline                 title
##                    87                 25043                     6
##                 video          vote_average            vote_count
##                     6                     6                     6
##                 month
##                    90
```

Looking for missing values in each column of the ratings dataset. No missing values was found so we continue our analysis.

```
colSums(sapply(ratings, is.na))
```

```
##    userId   movieId    rating timestamp
##         0         0         0         0
```

Cleaning up the genres column in the movies dataset by removing all the symbols.

```
stopwords = c("id","name")

movies$genres <- gsub("[[:punct:]]", "", gsub('[[:digit:]]+', '', gsub(paste0(stopwords,collapse
= "|"),"", movies$genres)))
```

We dropped certain variables that were deemed unimportant in the movies dataset for this recommendation system with the following codes.

```
drops <- c("adult","belongs_to_collection","budget","homepage","imdb_id","original_title","origi
nal_language","overview","popularity","poster_path","production_companies","production_countrie
s","revenue","runtime","spoken_languages","status","tagline","video","vote_average","vote_count"
,"month")

movies <- movies[,!(names(movies) %in% drops)]
```

We only drop the timestamp variable in the ratings dataset.

```
drops2 <- c("timestamp")

ratings <- ratings[,!(names(ratings) %in% drops2)]
```

Convert the release_date column in the movies dataset to only show the year, movies were released in. This is a new column which we will add permanently to our movies dataset analysis going forward.

```
movies$release_year <- format(as.Date(movies$release_date, format="%Y-%m-%d"),"%Y")
```

Next we delete the release_date column.

```
movies[,c("release_date")] <- list(NULL)
```

We are only interested in movies between 1965 and 2017 inclusive. Outliners with 2018 containing 5 movies and 2020 with 1 movie were removed since this dataset should only have movies up to 2017.

```
movies <- subset(movies,release_year >= 1965 & release_year <= 2017,)
```

```
head(movies)
```

```
## # A tibble: 6 x 4
##   genres                                 id title                  release_year
##   <chr>                               <dbl> <chr>                  <chr>
## 1 "   Animation    Comedy    Family"    862 Toy Story              1995
## 2 "   Adventure    Fantasy    Family"  8844 Jumanji                1995
## 3 "   Romance    Comedy"              15602 Grumpier Old Men       1995
## 4 "   Comedy    Drama    Romance"     31357 Waiting to Exhale      1995
## 5 "   Comedy"                         11862 Father of the Bride Pa… 1995
## 6 "   Action    Crime    Drama    Th…   949 Heat                   1995
```

Verify data quality

An unclean movie dataset and a relatively clean set of records from the ratings dataset were extracted and collectively uploaded into the Kaggle website. After examination the dataset appears to be able to cover the cases required for modeling and does not appear to contain errors.

Grouping movies by year released using the following R code

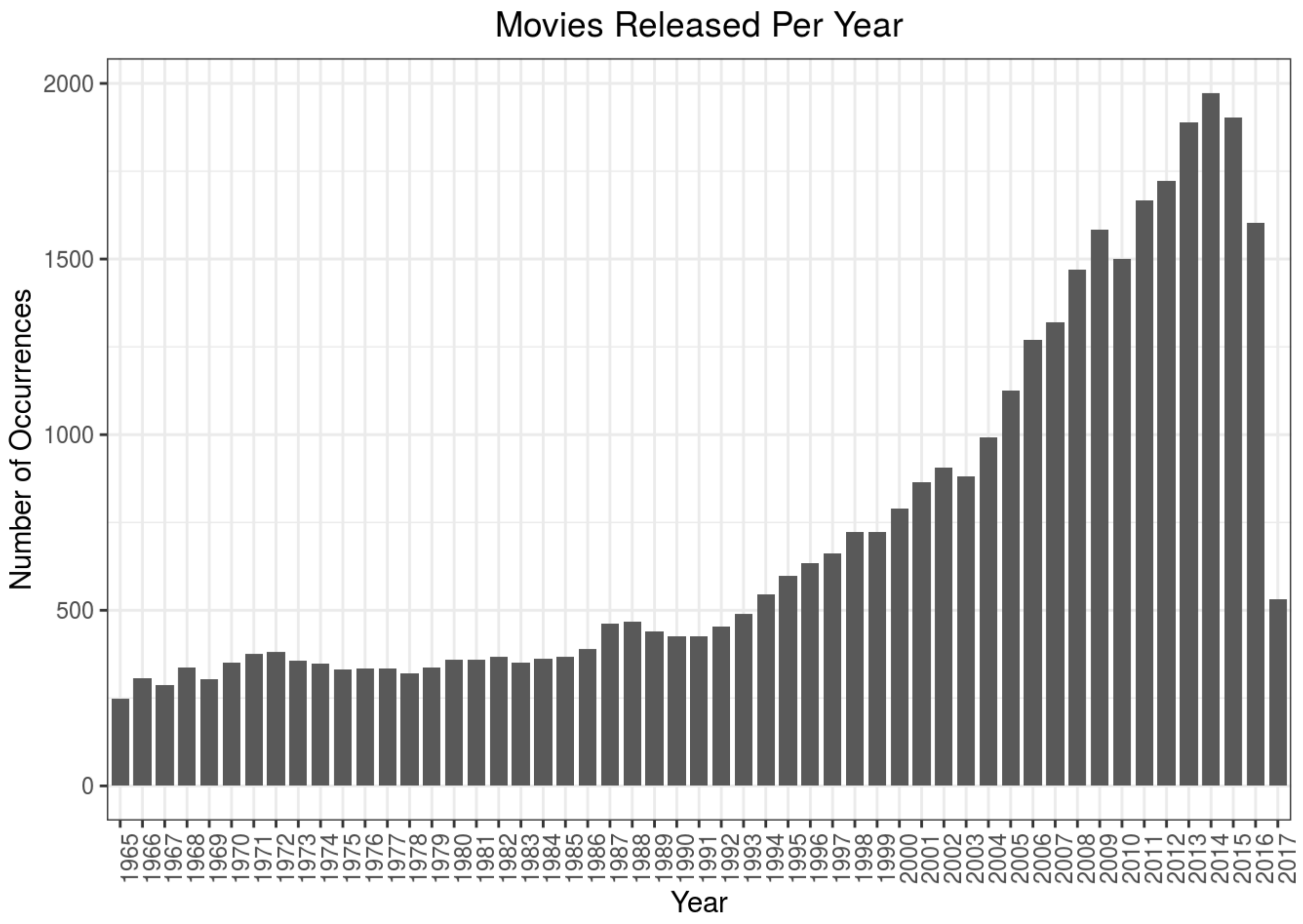
```
library(dplyr)
year_group <- group_by(movies,release_year)
movie_year <- summarise(year_group,n=n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
movie_year <- movie_year[order(movie_year$n,decreasing=TRUE),]
```

Using the ggplot2 library, we plot the results for movie_release dataframe.

```
library(ggplot2)
ggplot(aes(x=release_year,y=n),data=movie_year) +
  geom_bar(stat='identity',width=0.8) +
            xlab('Year') +
            ylab('Number of Occurrences') +
            ggtitle('Movies Released Per Year') +
            theme_bw() +
            theme(axis.text.x = element_text(angle=90),plot.title=element_text(hjust=0.5))
```



From 1992, there has been a gradual increase in movies probably due to the beginning of the digital age.The year 2017 has about 500 movie releases because the dataset ends at July, 2017.

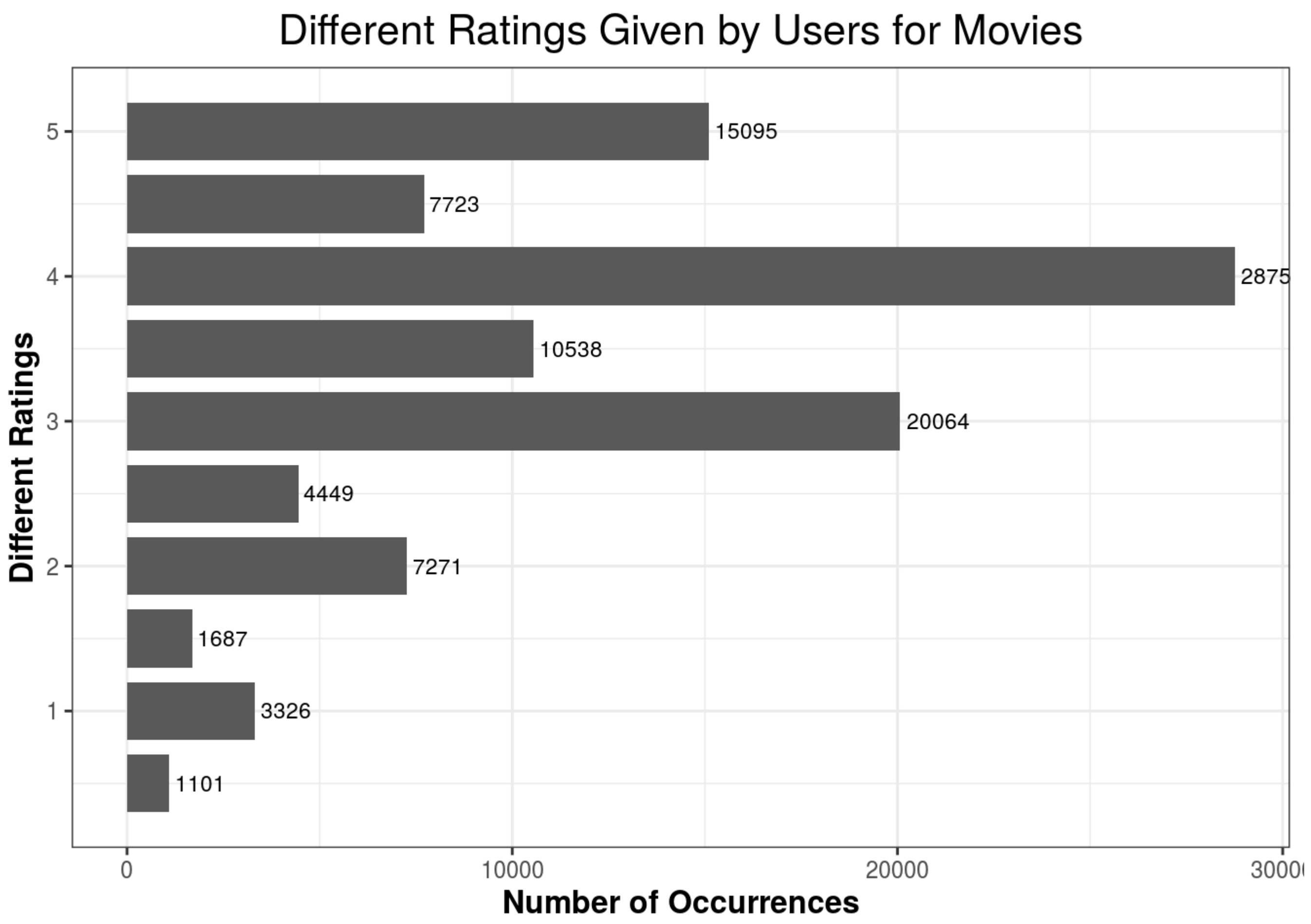
Grouping the number of stars users gave to different movies.

```
rating_group <- group_by(ratings,rating)
ratings_r <- summarise(rating_group,n=n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
ratings_r <- ratings_r[order(ratings_r$n,decreasing=TRUE),]
```

```
ggplot(aes(x=rating,y=n),data=ratings_r) +
  geom_bar(stat='identity',width=0.4) +
  geom_text(aes(label=n),stat='identity',data=ratings_r,hjust=-0.1,size=3) +
            coord_flip() +
            xlab('Different Ratings') +
            ylab('Number of Occurrences') +
            ggtitle('Different Ratings Given by Users for Movies') +
            theme_bw() +
            theme(plot.title=element_text(size=16,hjust=0.5),
            axis.title=element_text(size=12,face="bold"))
```



Once a movie was rated it appears on the histogram with the hightest rating being 5 and lowest rating being 0.5.

Modeling

Import the reshape2 and stringi library

```
library(stringi)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

Create ratings matrix with rows as userId and columns as movieId.

```
ratingmat = dcast(ratings, userId~movieId, value.var = "rating", na.rm=FALSE)
class(ratingmat)
```

```
## [1] "data.frame"
```

We can now remove userIds

```
ratingmat = as.matrix(ratingmat[,-1])
```

Import the recommenderlab library

```
library(recommenderlab)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loading required package: arules
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
## The following objects are masked from 'package:base':
## 
##     abbreviate, write
```

```
## Loading required package: proxy
```

```
## 
## Attaching package: 'proxy'
```

```
## The following object is masked from 'package:Matrix':
## 
##     as.matrix
```

```
## The following objects are masked from 'package:stats':
## 
##     as.dist, dist
```

```
## The following object is masked from 'package:base':
## 
##     as.matrix
```

```
## Loading required package: registry
```

```
## Registered S3 methods overwritten by 'registry':
##   method               from 
##   print.registry_field proxy
##   print.registry_entry proxy
```

Firstly, we would need to reduce the size of the matrix to make computation faster. Convert ratingmat matrix into realRatingMatrx. realRatingMatrix is a recommenderlab sparse-matrix like data-structure.

```
ratingmat = as(ratingmat, "realRatingMatrix")
```

ratingmat is defined as the matrix which takes into consideration of users who have rated atleast 30 movies and movies that have been viewed atleast 40 times.

```
ratingmat <- ratingmat[rowCounts(ratingmat) > 30,colCounts(ratingmat) > 40]
ratingmat
```

```
## 546 x 596 rating matrix of class 'realRatingMatrix' with 47506 ratings.
```

USING THE USER-BASED COLLABORATIVE FILTERING (UBCF) MODEL

Training

Data is split randomly with 90% train and 10% test.

```
which_train <- sample(x = c(TRUE, FALSE), size = nrow(ratingmat),replace = TRUE, prob = c(0.9,
0.1))
recc_data_train <- ratingmat[which_train,]
recc_data_test <- ratingmat[!which_train,]
```

Training parameter

Let’s take a look at the default parameters of the UBCF model. method: shows how to calculate the similarity between users nn: shows the number of similar users

```
recommender_models <- recommenderRegistry$get_entries(dataType ="realRatingMatrix")
recommender_models$UBCF_realRatingMatrix$parameters
```

```
## $method
## [1] "cosine"
##
## $nn
## [1] 25
##
## $sample
## [1] FALSE
##
## $normalize
## [1] "center"
```

We the build the UBCF Recommender model using the parameters above.

```
recc_model <- Recommender(data = recc_data_train, method = "UBCF")
recc_model
```

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 497 users.
```

Using getModel we can extract some details about the model.

```
model_details <- getModel(recc_model)
model_details
```

```
## $description
## [1] "UBCF-Real data: contains full or sample of data set"
##
## $data
## 497 x 596 rating matrix of class 'realRatingMatrix' with 43716 ratings.
## Normalized using center on rows.
##
## $method
## [1] "cosine"
##
## $nn
## [1] 25
##
## $sample
## [1] FALSE
##
## $normalize
## [1] "center"
##
## $verbose
## [1] FALSE
```

Using the R code below we can look at the components of the model.

```
names(model_details)
```

```
## [1] "description" "data"        "method"      "nn"          "sample"
## [6] "normalize"   "verbose"
```

Running the R code model_details$data object contains the rating matrix

```
model_details$data
```

```
## 497 x 596 rating matrix of class 'realRatingMatrix' with 43716 ratings.
## Normalized using center on rows.
```

Predict

Test dataset is split randomly since only 10% is used for test. We determine the top five recommendation for each new user. The type parameter is top-N-List since we want to get the Top-5 recommendation for a user.

```
n_recommended <- 5
recc_predicted <-predict(object = recc_model, newdata=recc_data_test,n = n_recommended, type="to
pNList")
recc_predicted
```

```
## Recommendations as 'topNList' with n = 5 for 49 users.
```

Top-5 movie recommendations for the different users.

```
recc_predicted <- as(recc_predicted,"list")
```

Importing the data.table library.

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:reshape2':
##
##     dcast, melt
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(dplyr)
```

Next we convert the recc_predicted list into a dataframe and assigned Top-5 Recommendations for the first user under the column movieId.

```
recc_predicted <- data.frame(recc_predicted)[1]
colnames(recc_predicted) <- "movieId"
recc_predicted
```

```
##   movieId
## 1     457
## 2     480
## 3     593
## 4     733
## 5     589
```

The movies are in number format so we want to get movies names. To achieve this we will have to use the movies dataset since it maps movie Id to movie titles.

But first let’s convert the id column in the movies dataset and also the movieId column in the recc_predicted dataframe into integer.

```
movies$id <- as.integer(movies$id)
recc_predicted$movieId <- as.integer(levels(recc_predicted$movieId))
```

```
str(movies)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    38552 obs. of  4 variables:
##  $ genres      : chr  "   Animation    Comedy    Family" "   Adventure    Fantasy    Family"
"   Romance    Comedy" "   Comedy    Drama    Romance" ...
##  $ id          : int  862 8844 15602 31357 11862 949 11860 45325 9091 710 ...
##  $ title       : chr  "Toy Story" "Jumanji" "Grumpier Old Men" "Waiting to Exhale" ...
##  $ release_year: chr  "1995" "1995" "1995" "1995" ...
##  - attr(*, "problems")=Classes 'tbl_df', 'tbl' and 'data.frame': 19 obs. of  5 variables:
##   ..$ row     : int  19730 19731 19731 19731 19731 19731 29503 29504 29504 29504 ...
##   ..$ col     : chr  NA "adult" "budget" "id" ...
##   ..$ expected: chr  "24 columns" "1/0/T/F/TRUE/FALSE" "a double" "no trailing characters"
...
##   ..$ actual  : chr  "10 columns" "- Written by Ørnås" "/ff9qCepilowshEtG2GYWwzt2bs4.jpg" "-0
8-20" ...
##   ..$ file    : chr  "'movies_metadata.csv'" "'movies_metadata.csv'" "'movies_metadata.csv'"
"'movies_metadata.csv'" ...
```

```
str(recc_predicted)
```

```
## 'data.frame':    5 obs. of  1 variable:
##  $ movieId: int  457 480 589 593 733
```

Take first user movieId recommendations and merge it with the id in the movies dataset to get the movie titles.

```
UBCF_movie_recommendations=left_join(recc_predicted, movies, by=c("movieId"="id"))
UBCF_movie_recommendations
```

```
##   movieId                                            genres           title
## 1     457                                              <NA>            <NA>
## 2     480                          Comedy    Drama    Romance Monsoon Wedding
## 3     589                                              <NA>            <NA>
## 4     593    Drama    Science Fiction    Adventure    Mystery         Solaris
## 5     733                                              <NA>            <NA>
##   release_year
## 1         <NA>
## 2         2001
## 3         <NA>
## 4         1972
## 5         <NA>
```

USING THE ITEM-BASED COLLABORATIVE FILTERING (IBCF) MODEL

k: The algorithm computes the similarities amoung each pair of items and then identifies its k most similar items and stores it method: This is the similarity function.

```
recommender_models$IBCF_realRatingMatrix$parameters
```

```
## $k
## [1] 30
##
## $method
## [1] "Cosine"
##
## $normalize
## [1] "center"
##
## $normalize_sim_matrix
## [1] FALSE
##
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE
```

Using the default parameters above, we build the IBCF Model.

```
recc_model2 <- Recommender(data = recc_data_train, method = "IBCF")
recc_model2
```

```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 497 users.
```

Predict

We want to get the Top-5 movie recommendations for a user.

```
recc_predicted2 <-predict(object = recc_model2, newdata=recc_data_test,n = n_recommended, type=
"topNList")
recc_predicted2
```

```
## Recommendations as 'topNList' with n = 5 for 49 users.
```

Top-5 movie recommendations for the different users in a list format.

```
recc_predicted2 <- as(recc_predicted2,"list")
```

Next we convert the recc_predicted2 list into a dataframe and assigned Top-5 Recommendations for the first user under the column movieId2.

```
recc_predicted2 <- data.frame(recc_predicted2)[1]
colnames(recc_predicted2) <- "movieId2"
recc_predicted2
```

```
##   movieId2
## 1       31
## 2      508
## 3      543
## 4      588
## 5      786
```

Convert movieId2 column in the recc_predicted2 dataframe into an integer.

```
recc_predicted2$movieId2 <- as.integer(levels(recc_predicted2$movieId2))
```

Take first user movieId2 recommendations and merge it with the id in the movies dataset to get the movie titles.

```
IBCF_movie_recommendations=left_join(recc_predicted2, movies, by=c("movieId2"="id"))
IBCF_movie_recommendations
```

```
##   movieId2                        genres          title release_year
## 1       31                          <NA>           <NA>         <NA>
## 2      508    Comedy    Romance    Drama Love Actually         2003
## 3      543                          <NA>           <NA>         <NA>
## 4      588              Horror    Mystery   Silent Hill         2006
## 5      786                 Drama    Music Almost Famous         2000
```

USING SINGULAR VALUE DECOMPOSITION (SVD) MODEL

Analyzing the default parameters in the SVD Model.

```
recommender_models$SVD_realRatingMatrix$parameters
```

```
## $k
## [1] 10
##
## $maxiter
## [1] 100
##
## $normalize
## [1] "center"
```

We the build the SVD Recommender model using the parameters above.

```
recc_model3 <- Recommender(data = recc_data_train, method = "SVD")
recc_model3
```

```
## Recommender of type 'SVD' for 'realRatingMatrix'
## learned using 497 users.
```

Test dataset is split randomly since only 10% is used for test. We determine the top five recommendation for each new user. The type parameter is top-N-List since we want to get the Top-5 recommendation for a user.

```
recc_predicted3 <-predict(object = recc_model3, newdata=recc_data_test,n = n_recommended, type=
"topNList")
recc_predicted3
```

```
## Recommendations as 'topNList' with n = 5 for 49 users.
```

```
recc_predicted3 <- as(recc_predicted3,"list")
```

Next we convert the recc_predicted3 list into a dataframe and assigned Top-5 Recommendations for the first user under the column movieId3.

```
recc_predicted3 <- data.frame(recc_predicted3)[1]
colnames(recc_predicted3) <- "movieId3"
recc_predicted3
```

```
##   movieId3
## 1     5952
## 2      260
## 3     4993
## 4     7153
## 5     1221
```

Convert movieId3 column in the recc_predicted3 dataframe into an integer.

```
recc_predicted3$movieId3 <- as.integer(levels(recc_predicted3$movieId3))
```

Take first user movieId3 recommendations and merge it with the id in the movies dataset to get the movie titles.

```
svd_movie_recommendations=left_join(recc_predicted3, movies, by=c("movieId3"="id"))
svd_movie_recommendations
```

```
##   movieId3                          genres       title release_year
## 1     1221                            <NA>        <NA>         <NA>
## 2      260                            <NA>        <NA>         <NA>
## 3     4993    Action    Western    Thriller 5 Card Stud         1968
## 4     5952                            <NA>        <NA>         <NA>
## 5     7153                            <NA>        <NA>         <NA>
```

Evaluation

Split the data into train and test where 90% is train. We consider a good rating as 4 or above (5). To split the data we use method = split.

```
e <- evaluationScheme(ratingmat, method="split", train=0.9, given=-1,goodRating=4)
e
```

```
## Evaluation scheme using all-but-1 items
## Method: 'split' with 1 run(s).
## Training set proportion: 0.900
## Good ratings: >=4.000000
## Data set: 546 x 596 rating matrix of class 'realRatingMatrix' with 47506 ratings.
```

Creation of UBCF Recommender Model and make predictions based on the test dataset the ratings for the given items to obtain error metrics.

```
Rec.ubcf <- Recommender(getData(e, "train"), "UBCF")
p.ubcf <- predict(Rec.ubcf, getData(e, "known"), type="ratings")
error.ubcf <- calcPredictionAccuracy(p.ubcf, getData(e, "unknown"))
```

Creation of IBCF Recommender Model and make predictions based on the test dataset the ratings for the given items to obtain error metrics.

```
Rec.ibcf <- Recommender(getData(e, "train"), "IBCF")
p.ibcf <- predict(Rec.ibcf, getData(e, "known"), type="ratings")
error.ibcf<-calcPredictionAccuracy(p.ibcf, getData(e, "unknown"))
```

Creation of SVD Recommender Model and make predictions based on the test dataset the ratings for the given items to obtain error metrics.

```
Rec.svd <- Recommender(getData(e, "train"), "SVD")
p.svd <- predict(Rec.svd, getData(e, "known"), type="ratings")
error.svd <- calcPredictionAccuracy(p.svd, getData(e, "unknown"))
```

The Root Mean Squared Error, Mean Squared Error and Mean Absolute Error for the UBCF, IBCF and SVD Model.

RMSE (Root mean squared error) is the standard deviation of the difference between the predicted and real ratings. MSE (Mean Squared Error) is the mean of the squared difference between the predicted and real ratings. RMSE is the root of MSE. MAE (Mean Absolute Error) is the mean of the absolute difference between the real and predicted ratings (Gorakala,2015).

```
error <- rbind(error.ubcf,error.ibcf,error.svd)

rownames(error) <- c("UBCF","IBCF","SVD")
error
```

```
##           RMSE       MSE       MAE
## UBCF 0.8224389 0.6764057 0.6650655
## IBCF 1.0240696 1.0487186 0.8485309
## SVD  0.7762869 0.6026214 0.6156134
```

Above results shows UBCF is slightly more accurate than IBCF and SVD is the best model to use out of the three.

```
set.seed(101)
scheme <- evaluationScheme(ratingmat, method="split", train = 0.9, k=1, given=-5, goodRating=4)
scheme
```

```
## Evaluation scheme using all-but-5 items
## Method: 'split' with 1 run(s).
## Training set proportion: 0.900
## Good ratings: >=4.000000
## Data set: 546 x 596 rating matrix of class 'realRatingMatrix' with 47506 ratings.
```

Checking our three models against each other. nn is how many neighbours to consider.

```
algorithms <- list(
 "user-based CF" = list(name="UBCF", param=list(nn=50)),
 "item-based CF" = list(name="IBCF", param=list(k=50)),
 "SVD approximation" = list(name="SVD", param=list(k = 50))
 )

results <- evaluate(scheme, algorithms, type = "topNList", n=c(1, 3, 5, 10, 15, 20))
```

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0.003sec/0.092sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [1.134sec/0.019sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.107sec/0.022sec]
```

SVD prediction time seems to shorter than UBCF with IBCF prediction time being the shortest out of the three models.

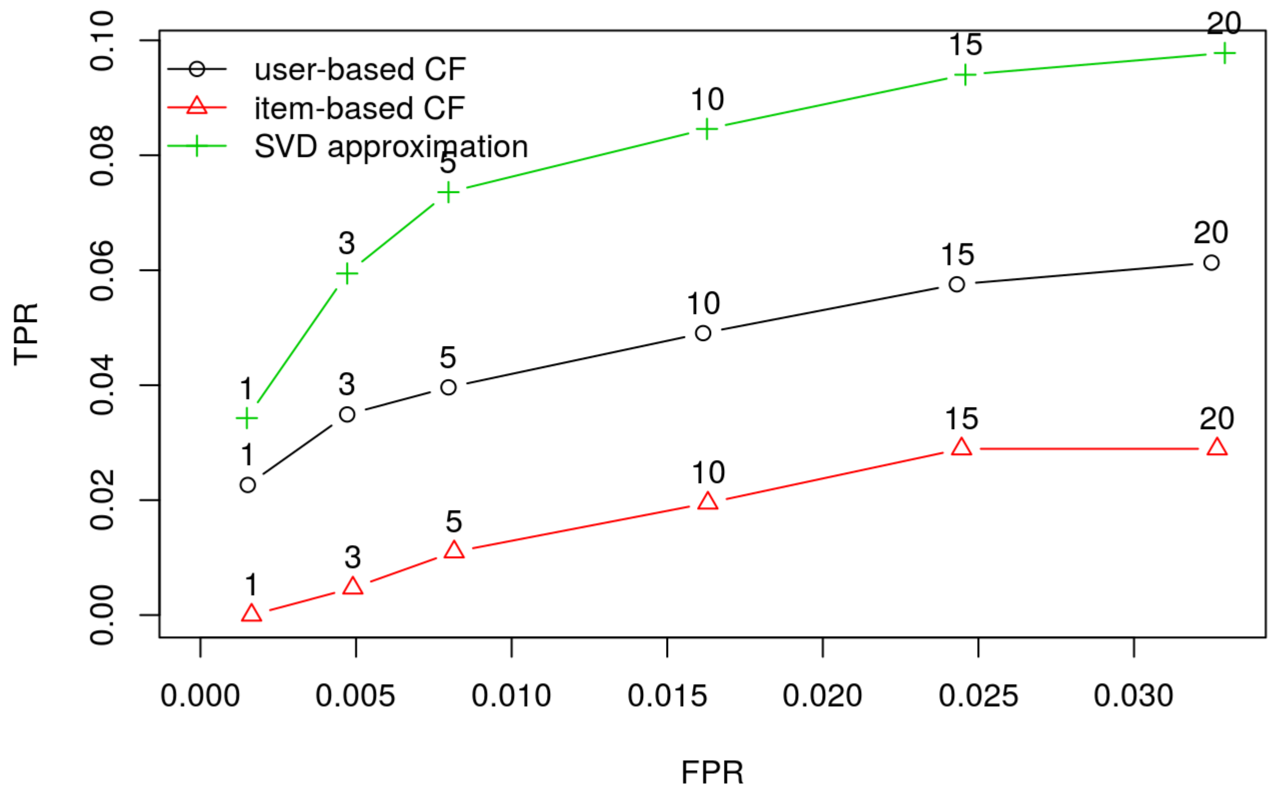
Plot ROC curve

Some background information is needed. (Gorakala,2015) True Positives (TP) are recommended items that have been purchased. False Positives (FP) are recommended items that haven’t been purchased. False Negatives(FN) are not recommended items that have been purchased. True Negatives (TN) are not recommended items that haven’t been purchased.

In our situation, items purchased refers to the movies rated.

ROC curve shows the TPR (True Positive Rate), which is the percentage of purchased items that have been recommended (It’s the number of TP (True Positive) divided by the number of purchased items (TP + FN (False Negative))) and FPR (False Positive Rate) which is the percentage of not purchased items that have been recommended (It’s the number of FP (False Positive) divided by the number of not purchased items (FP + TN (True Negative)). (Gorakala,2015)

```
plot(results, annotate=c(1,2,3), legend="topleft")
```



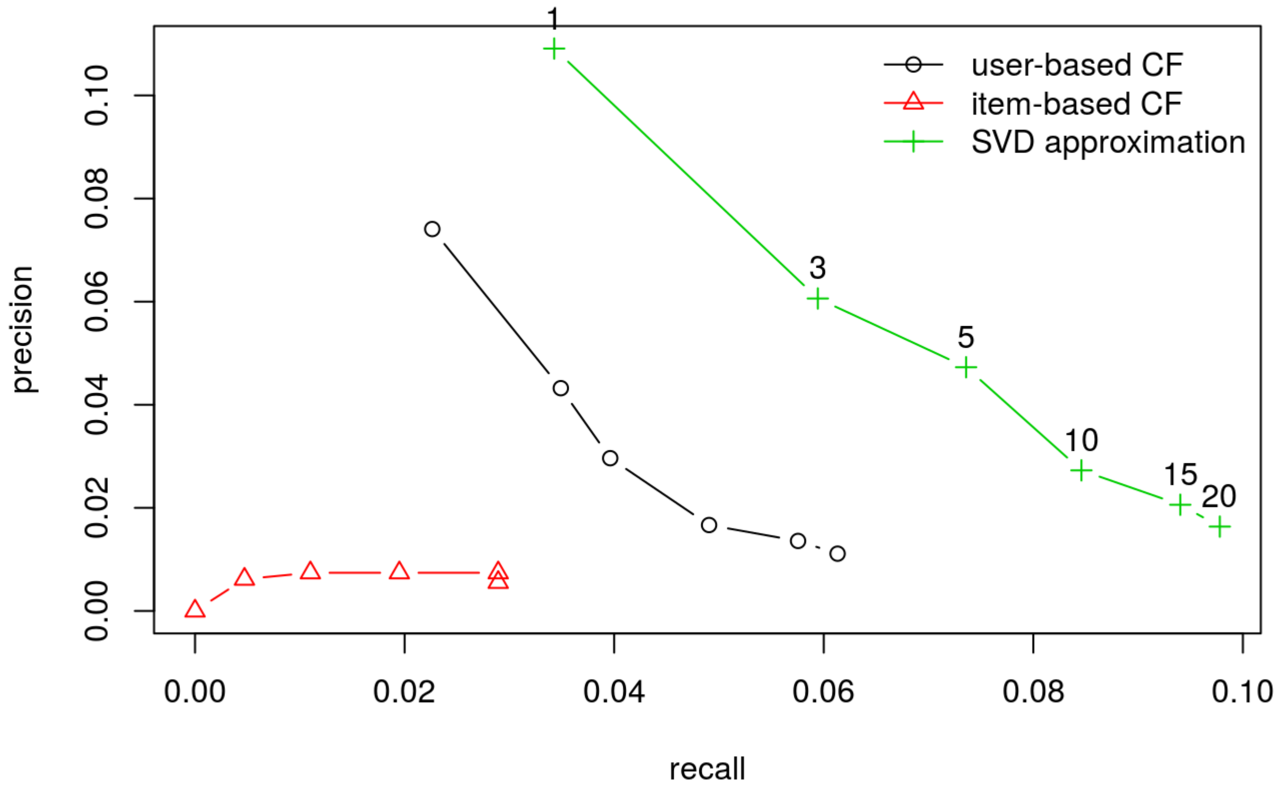
The Area Under the Curve (AUC) that is the area under the ROC curve is a good performance index. By just looking at the graph, we can see that SVD has the highest AUC so it's the best model out of the three. UBCF did better than IBCF. But IBCF is useful because it doesn't save everything but only saves k-closest items in the matrix. UBCF on the other hand saves the entire matrix and make recommendations based on the closest user (Zhao,2014).

Plot precision/recall

For a precision-recall plot, there are two accuracy metrics (Gorakala,2015) we need:

1. Precision, which is the percentage of recommended items that have been purchased. It's the number of FP (False Positive) divided by the total number of positives (TP (True Positive) + FP).
2. Recall, which is the percentage of purchased items that have been recommended. It's the number of TP (True Positive) divided by the total number of purchases (TP + FN (False Negative)). It's also equal to the True Positive Rate.

```
plot(results, "prec/rec", annotate=3, legend="topright")
```



The graph reflects the tradeoff between precision and recall, with the SVD model is still the better model than the UBCF and IBCF models. Depending on what we want to achieve, the choice of parameters might be slightly different. (Gorakala,2015)

Deployment

Our recommendation engine is beneficial for all the stakeholders mentioned in our business success criteria and the organizations that may be interested in this idea. Firstly, current subscribers/users of our platform can test and use a product to maintain their engagement with our movie services. Providing movie recommendations will enhance their experience and encourage loyalty. Secondly, the developers and deployment staff will view this as an opportunity to learn more about the users in advanced product releases with further interaction and feedback. Finally, in this competitive environment, organizations will find this use case to understand further that recommendation engines are becoming necessary to make the right choices for users interacting with their respective platforms. Our model would first be deployed with the UI's appropriate advancements to showcase the functionalities and models. Furthermore, our development team can critically analyze other extensions of this recommendation engine and integration options with interested parties looking to deploy within their platforms.

Other data that needs to be collected include; user preferences and ratings on the movies predicted to them. For example, if we can extend the product even further to allow the user to select movies that they would be most likely to watch from the predicted list and then provide an additional list of recommendations, this application will further enhance the user's interest and belief in our engines. Our model would need to be updated monthly for bugs and information collection on the predictions for maintenance purposes.

References

Banik, R. (2017). The Movies Dataset. Retrieved from https://www.kaggle.com/rounakbanik/the-movies-dataset#movies_metadata.csv (https://www.kaggle.com/rounakbanik/the-movies-dataset#movies_metadata.csv)

Segment. (2017). The 2017 State of Personalization Report (pp. 1-14). Retrieved from http://grow.segment.com/Segment-2017-Personalization-Report.pdf (http://grow.segment.com/Segment-2017-Personalization-Report.pdf)

Azzopardi J. (2018) Item-Based Vs User-Based Collaborative Recommendation Predictions. In: Szymański J., Velegrakis Y. (eds) Semantic Keyword-Based Search on Structured Data Sources. IKC 2017. Lecture Notes in Computer Science, vol 10546. Springer, Cham

Gorakala, S. K., & Usuelli, M. (2015). Building a Recommendation System with R. Packt Publishing.

Zhao, Y., & Cen, Y. (2014). Data mining applications with R. Waltham, MA, USA: Academic Press.

Malaeb,M. (Dec,2016). Singular Value decomposition (SVD) in recommender systems for Non-math-statistics-programming wizards. Retrieved from https://medium.com/@m_n_malaeb/singular-value-decomposition-svd-in-recommender-systems-for-non-math-statistics-programming-4a622de653e9 (https://medium.com/@m_n_malaeb/singular-value-decomposition-svd-in-recommender-systems-for-non-math-statistics-programming-4a622de653e9)