VIT-AP UNIVERSITY

CSE4029
Advanced Data Analytics
Slot G2

Project Report

Movie Recommendation System

Submitted To:
Prof. J. Vijaya

Team Members:

SUDHANI RUPALI - 19BCE7095
SHIVANI V - 19BCE7631
ATHRWA DESHMUKH - 19BCE7381
GNAANESHWAR - 19BCE7065

Dataset Dimensions:
Columns: 12
Rows/Samples: 7787
Dataset Link

CODE:

```
#Setting the working Directory in which the dataset is present
setwd('C:/ADA Project')
library(tidyverse)
library(tidytext)
#We will use the following features to get recommendations: Director, Cast, Country, Genre, Description
netflix = read.csv('netflix_titles.csv')

#defining functions
#Whenever we have a character vector of length >1, the elements must have their own rows in the data frame.

#These functions will enable us to do that:
spread_netflix <- function(show_id, var) {
  a <- data.frame(show_id = show_id,var = str_split(as.character(var),pattern = ', '))
  names(a) <- c('show_id','var')
  a[is.na(a$var),2] <- ''
  spread_df <<- rbind(spread_df, a)
}

desc <- data.frame()
spread_desc <- function(show_id, var) {
  a <- data.frame(show_id = show_id,desc = str_split(as.character(var), pattern = ' '))
  names(a) <- c('show_id','word')
  a$word <- gsub('[\\,.;:!?"]','',a$word)
  a$word <- a$word %>% str_to_lower()
  a <- anti_join(a, stop_words, by= 'word')
  desc <<- rbind(desc, a)
}
#These functions are taking the var and splitting it, either using a comma as a separator or a space in the case of spread_desc().

#The functions use spread() from the tidyverse to turn wide data into long and concatenate it to a blank data frame outside the function.

#The spread_desc() function does some extra cleaning of the description data before sending it to the long data frame for processing. It removes punctuation to clean the data and uses the tidytext package to remove stop words. These are common words such as 'from' and 'to' that don't provide us with any useful information about the show.
#get cast and main characters (split into its own cell as it takes a while)
```

#This code applies the spread_netflix() function we just created passing in the cast column as the variable.

#It splits off the main characters from the rest of the cast

```
spread_df <- data.frame()
mapply(netflix$show_id, FUN = spread_netflix, var = netflix$cast)
cast_members <- spread_df
cast_members$var <- as.character(cast_members$var)
cast_members$seq <- ave(cast_members$var, cast_members$show_id, FUN =
seq_along)
cast_members$type <- 'cast_member'
main_characters <- cast_members[cast_members$seq == "1",]
main_characters <- main_characters[main_characters$var != '',]
```

#To see that we have each cast member from the first show as their own row in the data frame.

```
head(cast_members)
```

```
> cast_members <- spread_df
> cast_members$var <- as.character(cast_members$var)
> cast_members$seq <- ave(cast_members$var, cast_members$show_id, FUN = seq_along)
> cast_members$type <- 'cast_member'
> main_characters <- cast_members[cast_members$seq == "1",]
> main_characters <- main_characters[main_characters$var != '',]
> head(cast_members)
  show_id              var seq        type
1      s1      João Miguel   1 cast_member
2      s1 Bianca Comparato   2 cast_member
3      s1     Michel Gomes   3 cast_member
4      s1  Rodolfo Valente   4 cast_member
5      s1  Vaneza Oliveira   5 cast_member
6      s1    Rafael Lozano   6 cast_member
>
```

#This code gets the rest of the variable data and puts it into separate data frames. We split up the execution of the code for the description because it was taking a lot of time on the system and even failing.

#getting countries

```
spread_df <- data.frame()
mapply(netflix$show_id, FUN = spread_netflix, var = netflix$country)
countries <- spread_df
countries$type <- 'country'
```

#getting genres

```
spread_df <- data.frame()
mapply(netflix$show_id, FUN = spread_netflix, var = netflix$listed_in)
genres <- spread_df
```

```
genres$type <- 'genres'

#getting directors
spread_df <- data.frame()
mapply(netflix$show_id, FUN = spread_netflix, var = netflix$director)
directors <- spread_df
directors$type <- 'director'

#get description of the show
desc <- data.frame()
mapply(netflix$show_id[1:3000], FUN = spread_desc, var = netflix$description[1:3000])
desc$type <- 'desc'
desc_all <- desc

desc <- data.frame()
mapply(netflix$show_id[3001:6000], FUN = spread_desc, var =
netflix$description[3001:6000])
desc$type <- 'desc'
desc_all <- rbind(desc_all, desc)

desc <- data.frame()
mapply(netflix$show_id[6001:nrow(netflix)], FUN = spread_desc, var =
netflix$description[6001:nrow(netflix)])
desc$type <- 'desc'
desc_all <- rbind(desc_all, desc)
names(desc_all)[2] <- 'var'
desc_all <- desc_all[desc_all$var != '-',]

#writing a function that matches a selection with the relevant shows.
This function takes three arguments: show_id, df and boost.
The show_id is the ID of the show that we have chosen. The df is one of the data
frames we have created in the steps above. The boost is an extra score to be given to
variables that we judge to be more important than others.
#The get_matches() function contains the boost argument because some variables are
more likely to get a stronger recommendation than others. Putting the boost in as an
argument means we can tweak exactly how important each variable is.
get_matches <- function(show_id, df, boost) {
  my_selection <- df[df$show_id == show_id,]
  my_selection_chr <- my_selection$var
  matching_titles <- subset(df, var %in% my_selection_chr)
  matching_titles <- matching_titles[matching_titles$show_id != show_id,]
  matching_titles <- matching_titles[matching_titles$var != '',]
```

```
  matching_titles <- matching_titles %>%
    group_by(show_id, var, type) %>%
    summarise(count = n() + boost, .groups = 'keep')
}

#writing function to suggest titles
suggest_titles <- function(title) {
  show_id <- netflix[netflix$title == title,1] %>% as.character()
  match_countries <- get_matches(show_id, df = countries, boost = 0)
  match_countries$count <- ifelse(match_countries$var == 'United States', yes =
match_countries$count, no = match_countries$count + 2)

  match_desc <- get_matches(show_id, df = desc_all, boost = 2)
  match_genres <- get_matches(show_id, df = genres, boost = 0)
  match_directors <- get_matches(show_id, df = directors, boost = 5)
  match_main_characters <- get_matches(show_id, df = main_characters, boost = 5)

  match_cast <- get_matches(show_id, df = cast_members, boost = 2)
  match_cast <- anti_join(match_cast,match_main_characters, by = 'show_id')
  all_titles <- rbind(match_countries,
                      match_genres,
                      match_directors,
                      match_main_characters,
                      match_cast,
                      match_desc)


  #check movie or TV
  #all_titles_summ <- arrange(all_titles_summ, desc(tally))
  all_titles <- merge(all_titles, netflix, by = 'show_id')


  suggested_titles <- all_titles %>%
    group_by(show_id) %>%
    summarise(tally = sum(count), .groups = 'keep')

  suggested_titles <- arrange(suggested_titles, desc(tally))
#getting top 5 suggestions
  top_picks <- suggested_titles %>% slice_max(tally, n = 5)

  top_picks <- merge(top_picks,netflix, by.x = 'show_id') %>% arrange(desc(tally))
  top_picks <- top_picks[1:5,]
```

```
  top_picks_df <- data.frame(title = paste0(top_picks$title,'\n'),
                             rank = paste0(seq(1,nrow(top_picks),1),'.'))
  top_picks_df$both <- paste(top_picks_df$rank, top_picks_df$title, sep = ' ')
  writeLines(paste0("You chose '",title,"'.\nBased on your choice we recommend: \n\n"))
  writeLines(top_picks_df$both)
}
```

#The final function takes the user's choice, filters all the data frames for matches, assigns a score to each and sums the tallies to produce a final score.

We gave a matching director or a matching main character a top boost of five. It gives a boost of two if the show is not American. This is because 42 per cent of the shows are wholly or partly American.

Knowing that the show is American doesn't tell you an awful lot. However if your selection is one of the 88 that is from Brazil then that makes the choice of country much more important.

The top matches are then written to the console!

#test out

```
suggest_titles(title = "Indiana Jones and the Temple of Doom")
```

Output:

```
> #test out
> suggest_titles(title = "Indiana Jones and the Temple of Doom")
You chose 'Indiana Jones and the Temple of Doom'.
Based on your choice we recommend:


1. Indiana Jones and the Last Crusade

2. Indiana Jones and the Raiders of the Lost Ark

3. Indiana Jones and the Kingdom of the Crystal Skull

4. Hook

5. K-19: The Widowmaker
```