Statisitical Consulting Final Write Up

Xunqing Zheng & Duja Michael

Extracting and Analyzing the New York Times Movie Reviews Data

Introduction

As the movie landscape continues to evolve, The New York Times movie reviews continue to offer insightful commentary, providing audiences with a nuanced perspective on the ever-changing and dynamic world of cinema. In order to get a deeper understanding of how the NYTimes is selecting what movies to review, what type of movie each of the staff critics review, and if there are patterns selecting which movies to be reviewed by whom, we decided to analyze all the movie reviews published in NYTimes from Januray 2018 until November 2023. In particular, we are trying to understand:
- Is there a pattern or a time-trend in the number of movies that the NYTimes reviews during this period?
- Is any particular genre more likely to be reviewed by NYTimes?
- Do particular reviewers/authors specialize in particular genres?
- What's the average rating of the movies reviewed each authors?
- Are "Critic's Picks" more likely to also receive higher ratings by the public?

Inital Exploration

They journey to land on this topic was a windy one. Initially, we wanted to work with a social media API where we would grab all the comments on a specific movie most and run sentiment analysis to get an idea about how the movie is received by the public. We also had a detour in our thinking in mid-October and wanted to shift our project to check the state of democracy in the USA by downloading the comments on all posts by the POTUS and Bernie Sanders accounts to see the prevalence of the calls for a ceasefire in Gaza knowing that those comments went on deaf ears. All those ideas proved to be unfit for the scope of this project due to the difficulty in extracting comments on a certain posts in the APIs of the social media platforms we wanted to use.

The final pivot we did was to go back to our initial idea of movie reviews and use the NYTimes' API.

Difficulties with the NYTimes API

Xunqing –> talk about the deprecated api, call limits, the parameters, etc

Quality, consistency, and insufficient information:

Another issue we faced with the NYTimes API was the lack of information and consistency in the data provided. For instance, data on the movie name and movie type are provided in a list of lists. Not all reviews had these two pieces of information, and in the cases where the information is provided, it is not arranged in a way that lends itself to systematic extraction. For movie names, we had to supplement the list data with information from the URL in cases where the list was missing that information. For the type of movie, the only categorizations that were consistently provided were whether the movie is a film, a documentary, or an animated film. No consistent information is provided for the genre of the movie.

From the NYTimes API data, we ended up extracting and using movie name, movie type, whether or not it was a critic's pick, the author of the review, and the date of publishing the review.

Supplementing the NYTimes data with IMDb information

Besides its paid API, IMDb has free datasets that are available for use for non-commercial purposes. We utilize this data to supplement the NYTimes information. In particular, this data adds information on the genre of the movie, the IMDb average rating, and the number of ratings.

Data Collection

Xunqing –> talk about how we got the info from the NYT api and how we saved it for later analysis;

unqing –> Also talk about the chunk of code you created to get the IMDb data directly from the web and about the two datasets (title basics and title reviews)

```r
imdbTSVfiles <- function(fileName){
  url <- paste0("https://datasets.imdbws.com/",fileName,".tsv.gz")
  tmp <- tempfile()
  download.file(url, tmp)

  assign(fileName,
         readr::read_tsv(
           file = gzfile(tmp),
           col_names = TRUE,
           quote = "",
           na = "\\N"),
         envir = .GlobalEnv)
}

imdbTSVfiles("title.basics")
```

```
## Rows: 10413389 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (5): tconst, titleType, primaryTitle, originalTitle, genres
## dbl (4): isAdult, startYear, endYear, runtimeMinutes
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
imdbTSVfiles("title.ratings")
```

```
## Rows: 1381661 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (1): tconst
## dbl (2): averageRating, numVotes
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

NYTimes data manipulation steps

1. All the files that were extracted from the API and saved as .R files were loaded and appended using the following code:

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.1     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(lubridate)
library(stringr)
library(zoo)
```

```
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(readr)
library(tidyverse)
library(splitstackshape)
library(stringi)
library(tm)
```

```
## Loading required package: NLP
##
## Attaching package: 'NLP'
##
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(ggplot2)
```

```
################
# loading API movie data
###############

# appending the data
Movie2018H1 <- load("Movie2018H1.Rdata")
Movie2018H1 <- MovieFile
Movie2018H2 <- load("Movie2018H2.Rdata")
Movie2018H2 <- MovieFile
Movie2019H1 <- load("Movie2019H1.Rdata")
Movie2019H1 <- MovieFile
Movie2019H2 <- load("Movie2019H2.Rdata")
Movie2019H2 <- MovieFile
Movie2020H1 <- load("Movie2020H1.Rdata")
Movie2020H1 <- MovieFile
Movie2020H2 <- load("Movie2020H2.Rdata")
Movie2020H2 <- MovieFile
load("Movie2021H1.Rdata")
```

```r
load("Movie2021H2.Rdata")
load("Movie2022H1.Rdata")
load("Movie2022H2.Rdata")
load("Movie2023H1.Rdata")
load("Movie2023H2.Rdata")


merged_all <- rbind(Movie2018H1, Movie2018H2,
                    Movie2019H1, Movie2019H2,
                    Movie2020H1, Movie2020H2,
                    Movie2021H1, Movie2021H2,
                    Movie2022H1, Movie2022H2,
                    Movie2023H1, Movie2023H2)
```

2. Given that the API provides movie and book reviews, we filter out all the observations that are not movie reviews using information from the URL column:

```r
# keeping only movies using the /movies/ pattern in response.docs.web_url
merged <- merged_all %>%
  filter(str_detect(response.docs.web_url, "/movies"))
# we have 4268 movies, now keep only unique ones
```

This gives a dataset with 4268 rows (movie reviews).

3. Only keep the unique reviews since some of the reviews might have been pulled more than once due to an overlap of the time period in the API call. This leaves us with 4142 unique movie reviews.

```r
merged_unique <- merged %>% distinct(response.docs.snippet, .keep_all = TRUE)
```

4. Create columns with variables of interest including review author, critic's pick, review date, movie type, and movie name:

```r
# Add author name (removing "By")
stopwords = c("By")
merged_unique$author <-gsub("By","",as.character(merged_unique$response.docs.byline.original))

# clean critic's pick column
merged_unique$criticpick <- ifelse(is.na(merged_unique$response.docs.headline.kicker),0,1)

# clean date column
merged_unique <- merged_unique %>%
  mutate (month = month(response.docs.pub_date),
          year = year(response.docs.pub_date))
merged_unique$monthyear <- as.yearmon(paste(merged_unique$year, merged_unique$month), "%Y %m")

# loop to extract movie type and movie name
merged_unique <- merged_unique %>% mutate(type=0,
                                          movie_name=0)
for (i in 1:nrow(merged_unique)) {
  typetable=as.data.frame(merged_unique$response.docs.keywords[i])
  merged_unique$type[i]=ifelse(any(typetable$value=="Documentary Films and Programs"),"Documentary Films
                             ifelse(any(typetable$value=="Movies"),"Movies",
                                    ifelse(any(typetable$value=="Animated Films"),"Animated Films",NA
  merged_unique$movie_name1[i] <- typetable[typetable$name=="creative_works", "value"][1]
}
```

```r
table(merged_unique$type, useNA = "always") #12 movies have no type information
```

```
##
##                  Animated Films Documentary Films and Programs
##                              13                           1095
##                          Movies                           <NA>
##                            3022                             12
```

```r
#Extract movie name from URL
merged_unique$movie_name2 <- str_match(merged_unique$response.docs.web_url, "/movies/\\s*(.*?)\\s*-revi

#Use name from URL if the information is not there in the response.docs.keywords list
merged_unique$movie_name <- ifelse(is.na(merged_unique$movie_name1),merged_unique$movie_name2,merged_uni

check3 <- merged_unique %>% filter(is.na(movie_name)) #only 3 have unidentifiable name from either sour
```

5. Clean up the movie names sufficiently so that they can be effectively merged with the IMDb data

```r
# remove (Movie) ; replace dashes with spaces ; remove white space ; lower case all for movie names
merged_unique$movie_name <-str_replace(merged_unique$movie_name, " \\s*\\([^\\)]+\\)", "")
merged_unique$movie_name <-str_replace_all(merged_unique$movie_name,'-',' ')
merged_unique$movie_name <- trimws(merged_unique$movie_name)
merged_unique$movie_name <- tolower(merged_unique$movie_name)
# replace & with "and", remove ":"
merged_unique$movie_name <-str_replace_all(merged_unique$movie_name,':','')
merged_unique$movie_name <-str_replace_all(merged_unique$movie_name,'&','and')
```

6. Keep relevant rows only and get the number of reviews by moth and year

```r
api_unique <- merged_unique %>% dplyr::select (response.docs.pub_date, month, year, monthyear,
                                   response.docs.news_desk, type, author, criticpick,
                                   movie_name, movie_name1, movie_name2) %>%
  group_by(monthyear) %>%
  mutate(n_month=n()) %>%
  ungroup() %>% group_by(year) %>%
  mutate(n_year=n()) %>%
  ungroup() %>%
  group_by(monthyear) %>%
  arrange(monthyear)
```

IMDb data preparation

1. Keep movies from 2017 onwards only since the NYTimes movie reviews start in 2017 and it is very unlikely for the NYTimes to review movies from over a year before and merge this information with that from the ratings dataset by movie ID (tconst).

```r
titles <- title.basics %>%
  filter(startYear>2016,
         titleType=="movie"|titleType=="tvMovie")
ratings <- title.ratings
joined <- left_join(titles,ratings, by="tconst") %>%
  rename(movie_name = primaryTitle,
         release_year=startYear) %>%
  mutate(release_year=as.numeric(release_year))
```

2. Clean up movie names in preparation for the merge with NYTimes data

```
# Clean up movie names: replace dashes with spaces ; remove white space ; lower case all for movie name
joined$movie_name <-str_replace_all(joined$movie_name,'-',' ')
joined$movie_name <- trimws(joined$movie_name)
joined$movie_name <- tolower(joined$movie_name)
joined$movie_name <-str_replace_all(joined$movie_name,':','')
joined$movie_name <-str_replace_all(joined$movie_name,'&','and')
```

Merging the two data sources

```
withratings <- left_join(api_unique, joined, by="movie_name") #5350
```

```
## Warning in left_join(api_unique, joined, by = "movie_name"): Detected an unexpected many-to-many rela
## i Row 3 of `x` matches multiple rows in `y`.
## i Row 47313 of `y` matches multiple rows in `x`.
## i If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
# keep only the joins where release date is BEFORE review date
withratings <- withratings %>% filter(release_year <= year) %>%
  group_by(movie_name) %>% mutate(n_matches=n(),
                                  unclear=ifelse(n_matches>1,1,0)) #4456 row left
table(withratings$n_matches, useNA = "always") # 3232 uniquely matched out of 4142
```

```
##
##    1    2    3    4    5    6    7    8    9   10   12 <NA>
## 3235  516  264  156  115   48   49   16   27   10   12    0
```

```
3232/4142
```

```
## [1] 0.7802994
```

```
#keep only matched movies
matched <- withratings %>% filter(n_matches==1)
```

Analysis and Visualizations

Conclusion