It's in the Slope

Linear Regression of IMDb Series

Steven Kalinke
Faculty of Mathematics & Computer Science
Applied Computer Science
Digital Humanities
—Project Report—

The Internet Movie Database (IMDb) provides free access to a set of their raw data, allowing to perform various analyses. This may reveal a different perspective on the data compared to the information on their website. Based on these datasets, the current project provides a method to visualize ratings of a given series over their seasons. By using the Python programming language I wrote a program that creates a scatter plot with a linear regression line illustrating the series' "performance" over time. I argue that the slope of the linear regression reflects the general extent of dis-/satisfaction with a certain series.

Introduction

In recent years it has become a daunting task to select a TV series. First off, video-on-demand services offering a great variety of series, making it difficult to decide for one. In addition, each series entails the risk of losing the viewers' interest over time. One thinkable reason is that the series loses its initial excitement and/or appeal. A way of dealing with the just described difficulties is to conduct own research beforhand (e.g. reading reviews, watching trailers, etc.). In the current project I aim to develop an alternative approach. Specifically, I intend to develop a method that allows to visualize the viewers' dis-/satisfaction with a certain TV series over time (i.e. over one or more seasons). Rating data from IMDb are used to create a scatter plot with a linear regression line, illustrating the "performance" of a certain series. I hypothesize that the negative/positive slope of a series' linear regression represents the viewers' overall dis-/satisfaction with a TV series.

The current project report is structured as follows: First, I will highlight some major (pre-)processing steps in the methods section. Second, the section results describes three examplary outputs of the program. Finally, I will discuss the relevance of the currently developed program together with problems resulting from IMDb's data quality and current limitations of the program.

Methods

I used Python 3 as the general programming language for this project. The pandas package (McKinney, 2010) was useed for (pre-)processing of the data. To allow more advanced plotting features the package seaborn (Waskom et al., 2014), based on matplotlib (Hunter, 2007), was used.

Data Sources

Data were obtained from IMDb. Since IMDb is offering its own datasets (IMDb, n.d.) it seems to be the most intuitive starting point for the purpose of the current project. However, it should not be overlooked that there is a popular alternative to IMDb's datasets, namely IMDbPY.

IMDbPY. This Python package (Malagoli, 2013) retrieves IMDb's data directly from the website by traversing through its DOM (Document Object Model) elements. Even though the package gives the impression of being a very helpful and promising tool, it is vulnerable to changes within the IMDb website, as happened recently[1], at a time when I was starting the project (January 2018). As there was no convenient interface to access IMDb's data within the Python language[2],

[1] http://github.com/alberanid/imdbpy/issues/103
[2] The developer closed this issue on March 4th.

Table 1
Excerpt of the dataset table `title.basics.tsv` *showing wanted data (set in black type): Breaking Bad as a TV series and an episode named Pilot, and unwanted data (set in grey-italic type): Arrival as the type of movie.*

tconst	**titleType**	**primaryTitle**	**…**	**startYear**	**…**
…	…	…	…	…	…
tt0903747	tvSeries	Breaking Bad	…	2008	…
…	…	…	…	…	…
tt0959621	tvEpisode	Pilot	…	2008	…
…	…	…	…	…	…
tt2543164	*movie*	*Arrival*	…	*2017*	…
…	…	…	…	…	…

I decided to use the above mentioned intuitive path, namely IMDb datasets.

IMDb Datasets. IMDb offers six datasets, see IMDb, n.d. Yet, in order to meet my specific task of getting TV series data and their ratings, I only needed three of them. The three tables shown on this page are simplified excerpts of the original structure of the IMDb dataset. These tables can be downloaded from the URLs provided in the paragraph download. However, the files will be downloaded anyway, when running my setup.

The dataset depicted in Table 1 was required for two main concerns: (1), the matching of the title (primaryTitle) with its corresponding unique id (tconst), and (2) capturing the year. When two ore more TV series have identical titles (primaryTitle). If the latter is the case, the program prompts the available release years asking the user to input the year of the requested title. More detailed information on that topic are given in the methods subsection (2) IMDBratings.py.

Not only Table 1 but also Table 2 was essential for two aspects: (1) the assignment of episodes to their corresponding TV series, and (2) the storing of episode/season numbers for each episode (e.g., the episode *Pilot* (tt0959621, see Table 1) is the first episode in season one). The information from (2) are substantial, since their information are mapped on the time-axis.

Finally, Table 3 was fundamental to retrieve all episode ratings as well as the overall TV series rating. This information gets assigned to the y-axis of my scatter plot.

Table 2
Excerpt of the dataset table `title.episode.tsv` *showing the allocation of a single episodes to their parent TV series.* ***Note:*** *Column labels have been shortened.*

tconst	**parent**	**seasonNr**	**episodeNr**
…	…	…	…
tt0959621	tt0903747	1	1
…	…	…	…

Table 3
Excerpt of the dataset table `title.ratings.tsv` *showing wanted information (set in black type) and unwanted information (set in grey-italic type).*

tconst	**averageRating**	**numVotes**
…	…	…
tt0903747	9.5	1067074
…	…	…
tt0959621	8.9	16789
…	…	…
tt2543164	*7.9*	*436158*
…	…	…

Preprocessing

Since I intended to create an easy-to-use command line application, that is—once set up—fast and reusable, I created two single-purpose Python files: (1) `IMDBratings_setup.py` and (2) `IMDBratings.py`. In the following subsections, I will illustrate some of the steps

that the program performs when being run. If you want to follow step-by-step, please start Python script (1) in your command line[3].

(1) IMDBratings_setup.py. The purpose of this Python file is to create a helper-dataframe that gets locally stored on the hard drive of the user. Initially, the setup script checks the existence of a folder named *raw*, within the directory of the two Python files, containing a csv file named *dataframe.csv*.

Download. If the latter is not the case—which is the usual behavior for the first run—the script is downloading three compressed .gz files from the IMDb servers (corresponding to Table 1, Table 2, Table 3):

1. datasets.imdbws.com/title.basics.tsv.gz
2. datasets.imdbws.com/title.episode.tsv.gz
3. datasets.imdbws.com/title.ratings.tsv.gz

Creating Dataframes. Once the downloads are completed, three dataframes will be created, corresponding to the tables mentioned before. The script is designed to discard all information that are not required for my analyzes, leading to a smaller file size and less memory consumption for the later helper csv. A common method for data pre-selection is to drop columns while creating dataframes. As far as Table 1 is concerned, I manually selected columns that I wanted to store. For this purpose, I used the parameter `usecols`, such as in the following simplified example:

```
cols = ['tconst','type','title','year']
pd.read_csv(...,usecols=cols,...)
```

This approach ignores all columns, that are not explicitly set, such as: *originalTitle, isAdult, runTime,* etc. However, the dataframe still contains data records of all unwanted types, such as movie, short, video, etc. Thus, a boolean query selects only the desired types (i.e., *tvSeries* and *tvEpisode*). To give an example (simplified):

```
df[(df['type']=='series') | (df['type']=='episode')]
```

Since only be one variable can be mapped on the x-axis, it is fundamental to combine the columns *seasonNumber* and *episodeNumber* into one single column named *sxxexx*[4] being my "timeline". Consequently, the actual data record for tconst tt0959621 (see Table 2) is 0101. Additionally, series or episodes with no rating were removed, as these records are useless for my purpose.

Merging Dataframes. Since the goal is to create a single csv file, containing only information about series and their episodes, there are two merging steps required: (1) merging Table 1 with Table 3, and (2) merging the resulting dataframe with Table 2.

Step (1). Due to the pre-processing, Table 1 only contains information of wanted types (*tvSeries* and *tvEpisode*). Therefore, it is convenient to use pandas function `merge` with its parameter `how` set to `'inner'`[5]. This intersects Table 1 and Table 3 on the key *tconst*. This means, that the resulting dataframe only contains the data that share the unique *tconst* of both tables, everything else gets omitted.

Step (2). In the second step, the resulting dataframe from step (1) is merged with Table 2 with the parameter `how` set to `'left'`[6], leading to the final dataframe depicted in Table 4. This table gets stored on the user's hard drive as *dataframe.csv* within the *raw* folder. In the next step this file is used by the second script: `IMDBratings.py`.

(2) IMDBratings.py. The purpose of this Pyhton file is to create a scatter plot with a linear regression of a requested series by the user. To visualize the interaction between the user and the program see the following flow chart.

Since the previously created csv file is tailor-made for my specific purpose, there are just a few steps to visualize the data. Initially, as can be seen in Figure 1, the program checks the existence of a file named *dataframe.csv* within the *raw* folder located at script's path. Assuming the file is found—which is the case when the setup did run before—it will be stored within Python as a dataframe. Thereafter, a while loop is asking the user to enter a title. The loop breaks if there is at least one match. In the case of multiple matches (e.g. try to enter "Stranger Things"), the command line prompts a list of release years for each title, and asks the user to enter the year of the requested title. This helps to narrow down the results to only one, since the combination of title and release year seems to be an identifier. In the next step, the title (and the year, if nec.) gets assigned to their unique id (tconst). Having the information of tconst, the pandas' query for getting all associated episodes of the parent tconst is (simplified):

```
df[(df['type']=='epsdes') & (df['parent']==tconst)]
```

Based on this episodes dataframe, the package seaborn (as

[3] If you have not installed the Anaconda distribution, it might be required to install the following packages: pandas, matplotlib and seaborn.

[4] The range of xx is from 00 to 99.

[5] Equal to set theory's intersection or SQL's inner join.

[6] Equal to SQL's left join.

Table 4
Excerpt of the final dataframe / csv file.

tconst	type	title	year	rating	votes	parent	sxxexx
…	…	…	…	…	…	…	…
tt0903747	tvSeries	Breaking Bad	2008	9.5	1067074		
…	…	…	…	…	…	…	…
tt0959621	tvEpisode	Pilot	2008	8.9	16789	tt0903747	0101
…	…	…	…	…	…	…	…

"sns") is used to finally plot the data as follows (simplified):

```
sns.regplot('sxxexx', 'ratings', df)
```

This code plots the season/episode code (*sxxexx*) on the x-axis and the corresponding *ratings* on the y-axis, using the data from *df*. Moreover, it adds a regression line to the scatter plot visualization.

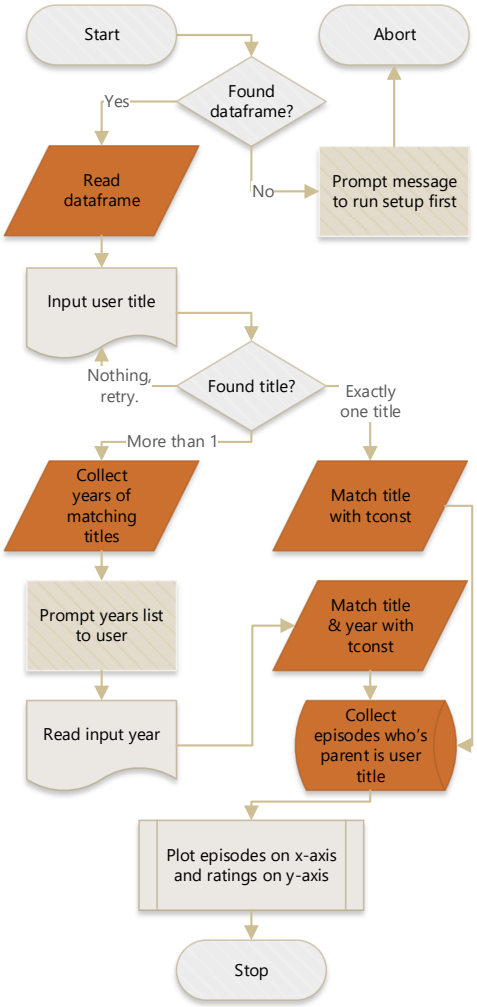

Figure 1. Flow chart of *IMDBratings.py*

Results

The scattered data and the regression line allow some interesting insights in the performance of a series over time. To illustrate the final results of the project, the current section presents data from three exemplary outputs. Note that the confidence interval of the regression line is .95, which is the default of the package seaborn.

Figure 2 shows the scattered data and regression line of the TV series "Breaking Bad". The data reveal an overall high rating and a positive regression slope. This pattern suggests that the series is perceived as overall satisfying. This seems to be a candidate for an overall satisfying, whereas the "performance" seems to be stable over the entire runtime of the entire show.

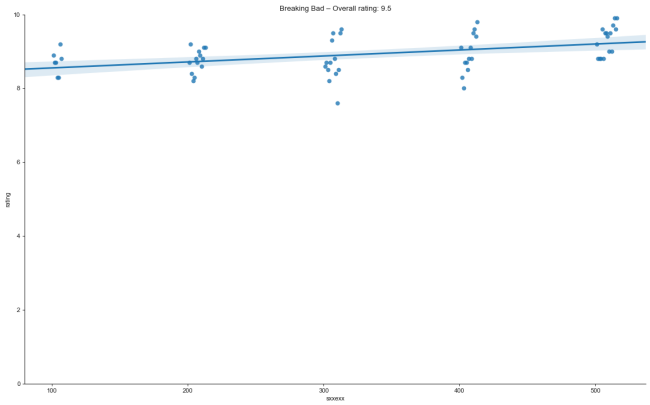

Figure 2. Output for the performance of "Breaking Bad".

An example for less smooth rating can be found in the series "Two and a Half Men" (see Figure 3). The abrupt decrease in satisfaction after season eight indicates a major incident. One likely explanation for the change is the leave of main cast Charlie Sheen[7]. Following this assumption it is plausible that the ratings get worse from season nine on, what causes the negative slope of the regression line. Nevertheless, viewers might still want to watch the series, since the first eight season are constantly satisfying. The negative slope indicates

[7]See trivia on http://www.imdb.com/title/tt0369179/

that the overall watching experience might be dissatisfying with regards to my hypotheses.

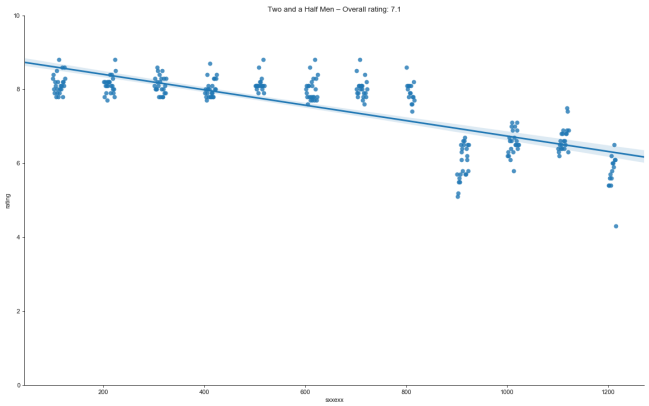

Figure 3. Output for the performance of "Two and a Half Men".

Generally, I observed, from the data I entered so far, that the series ratings mostly reveal more negative slopes than positive ones. Following my initial made assumption, one could argue that a series can lose its excitement/appeal over time. Interestingly, this seems to be the case for the majority of TV series. For example, the series "The Walking Dead" (see Figure 4) was very well rated from beginning to mid of its runtime, before the variance of ratings increased from season four on, accompanied by a general negative tendency—a typical pattern that is likely to be linked with a dissatisfying watching experience for the entire series.

Please note, that all conclusions presented in this section are hypothetical since they are based on visual inspection of the data.

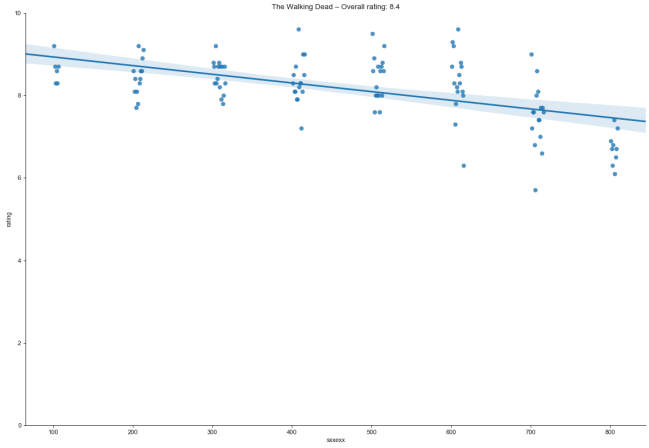

Figure 4. Performance of "The Walking Dead".

Discussion

The examples presented in the results section allow the conclusion that I successfully created a tool that visualizes the performance of a TV series over time. This tool has its relevance, since it is very helpful in times where series consumers can get easily overwhelmed by the amount of available series. The visualization of series performance, bases on IMDb data, helps to selectively focus on valuable entertainment, and avoids wasting the viewers' time and frustration.

However, the current program version contains some notable limitations. First off, the validity of IMDB's rating data is restricted by a significant gender bias skewing the data towards male preferences. Since much more men are voting—ranging from 70% to over 90% (see Hickey, 2018, Reynolds, 2017, and Beard, 2016)—the data are not representative for both sexes. According to Beard, 2016, *"[…] it turns out that men tend to look much more favourably on films with more masculine themes, or male leading actors."*.

Olteanu, 2017 addressed another interesting concern in regard to the rating of movies (also applying to series ratings): Apparently, the ratings from most movies follow the pattern of a normal distribution. In other words, there are only few movies/series that are rated as outstanding, or terrible. Most of them are rated near average (see Figure 5). In the article an evaluations was conducted about the distribution patterns of the data of IMDb, Rotten Tomatoes, Metacritic, and Fandango. The findings suggest that Metacritic's "metascore" is the most reliable one, since is distributions is closest to a normal distribution.

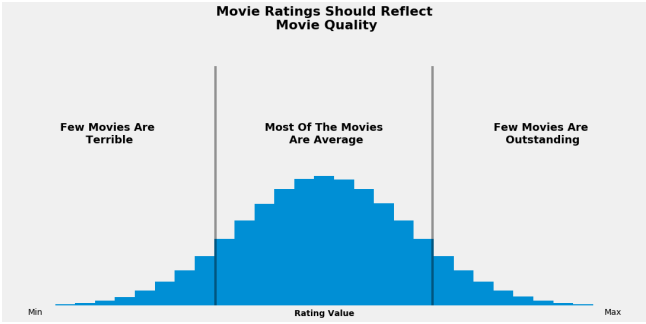

Figure 5. Ideal distribution of movie/series ratings (Olteanu, 2017).

One thinkable way to improve the data quality is the use of alternative data sources, such as Metacritc's data. Even multiple data sources could be used, however this would require the adapting and matching of different parameters.

Besides the limitations of the data I used, another major discussion point is the causation of my working hypotheses, namely that the slope of the regression line predicts overall dis-/satisfaction of a series. To validate my assumption the statement need to be investigated with statistical analysis in future experimental studies.

Further Development

Since I really enjoyed working with Python (especially, in this particular project) I plan to continue working on the project, and plan to host the source files on GitHub. At the moment I work on two main issues: Utilizing the Python package statsmodels (Seabold & Perktold, 2010) gives us direct access to the equation of the linear regression line. Thus, it enables us to get the numerical value of the slope (unfortunately, this is not possible with seaborn, yet). Since it would be interesting for series viewers to know what TV series have the steepest positive slopes, I implement a function returning a descending list of positive slope values. Additionally, I intend to validate the previously mentioned assumption that the majority of slopes (> 50%) is negative.

References

Beard, M. (2016). Imdb analysed: How do men and women's favourite films differ? Retrieved March 3, 2018, from https://oneroomwithaview.com/2016/08/10/imdb-analysed-men-womens-favourite-films-differ/

Hickey, W. (2018). What if online movie ratings weren't based almost entirely on what men think? Retrieved March 3, 2018, from https://fivethirtyeight.com/features/what-if-online-movie-ratings-werent-based-almost-entirely-on-what-men-think/

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

IMDb. (n.d.). Imdb datasets. Retrieved March 3, 2018, from https://www.imdb.com/interfaces/

Malagoli, A. (2013). Imdbpy. Retrieved March 3, 2018, from https://imdbpy.sourceforge.io/

McKinney, W. (2010). Data structures for statistical computing in python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th python in science conference* (pp. 51–56).

Olteanu, A. (2017). Whose ratings should you trust? imdb, rotten tomatoes, metacritic, or fandango? Retrieved March 3, 2018, from https://medium.freecodecamp.org/whose-reviews-should-you-trust-imdb-rotten-tomatoes-metacritic-or-fandango-7d1010c6cf19

Reynolds, M. (2017). You should ignore film ratings on imdb and rotten tomatoes. Retrieved March 3, 2018, from http://www.wired.co.uk/article/which-film-ranking-site-should-i-trust-rotten-tomatoes-imdb-metacritic

Seabold, S. & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In *9th python in science conference*.

Waskom, M., Botvinnik, O., Hobson, P., Cole, J. B., Halchenko, Y., Hoyer, S., & Miles, A. (2014). Seaborn: V0.5.0 (november 2014). doi:10.5281/zenodo.12710

Author's Note

This project report and the project itself was part of the lecture *Introduction to Digital Humanities (DH.EDH17/18)*, held by Dr Francesco Mambrini at Leipzig University. I hereby declare that I only used the literature listed before, and I had not any other help in coding or writing the report.

Steven Kalinke
steven.kalinke@outlook.com
2512638