horizontal line

Movie Analysis Project

**2nd August 2019**

# **OVERVIEW**

We decided to do analysis on movie data for the final project in CMPT 353 because of our interest in movies and curiosity of trends and relationships between fans and movies.

# **GOALS**

* Use statistical tools to answer questions we have about movies.
* Use machine learning tools to predict movie sentiments, and explore trends within the industry.

**Parts:**

Part 1: Do people like the first movie better than the second one in a series?

Part 2: This task is to predict the polarity of a movie based on cast members and movie ratings.

Part 3: This part is to predict the polarity of a movie based on the plot that the movie has got.

Part 4: This part focuses over analyzing the behaviour between movie ratings and movie awards through visual interpretation

Part 1: Do people like the first movie better than the second one in a series?

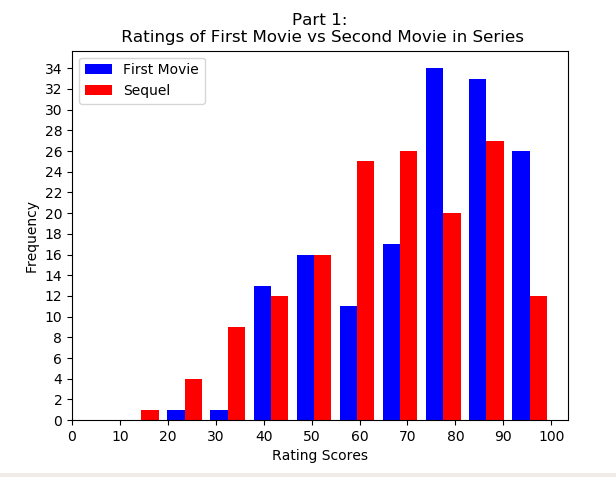
This part of the project stemmed from the curiosity of whether it is true that the audience likes the first movie in a series better than the rest. We thought it would be difficult to compare the first movie in a series with all the other movies in a series and that it would not be a fair evaluation. Therefore, we decided to only compare the first movie in the series to the second, because they are more likely to have the same quality, production, and teams.

The movie data was downloaded from the sfu computing cluster. We needed both rotten tomato data as well as data about the movies in order to find the first and second movies in a series. The initial cleaning was done in spark to filter out unnecessary data, then output into a folder. From there we used pandas data frame for the rest of the calculations. We grouped movies by series then converted dates to date time objects in order to sort. Once sorted the movies by date published we were able to extract the first and second movie in each series. From there we were able to create a new dataframe that combined the movie titles with their respective rotten tomato scores, which is now ready for statistical analysis.

I performed a normal test and Levene’s test in order to see if the data is viable for a regular t-test. The Levene's test outputted a p-value above the level of significance of 0.05, therefore we could proceed as if the data has equal variance. The p-values from the normal test were not above the level of significance of 0.05, therefore could not conclude that the datasets were normal. Because we were not able to normalize the data we resorted to the Mann-Whitney U-test. I also needed to make sure I did a one-sided t-test, and I was able to do that by passing in ‘alternative=greater’ parameter into the Mann-Whitney U-test function. The null hypothesis was that the first movie in a series received lower than or equal ratings than the second movie, and the alternative hypothesis was that first movie received ratings greater than the second movie in the series.

The p-value from the Mann-Whitney U-test was 0.00037. Since the p-value is below the level of significance of 5%, there is a high probability that the null hypothesis is false. Therefore, we can reject it the null hypothesis and accept the alternative hypothesis that first movie rating receives better ratings than the second movie in a series.

As you can see below, my conclusion that the first movie in a series receives better ratings that the second movie in a series is represented in the double bar graph below. There were more first movies than sequels in the upper percentiles of the ratings.



Some things that we could improve on is that we could have resolved the warnings in the code, but it would have complicated the functions. Also, could have avoided the use of for loops, but since we reduced the size of the data considerably with spark when cleaning the data loops did not affect the time too much. Another problem we encountered was that the data was not normal, therefore we could not do a regular t-test. We attempted to normalize the data, but couldn't, so we had to resort to the Mann-Whitney U test.

Part 2: This task is to predict the polarity of a movie based on cast members and movie ratings.

Our motive for this task is to predict the polarity (-1 to 1) i.e. negative score means that the movie’s sentiment has negative emotions such as tragedy, sadness, conflict whereas positive polarity means movie contains positive sentiments such as comedy, jokes, success and excitement based on the cast members and the critic ratings that a movie has received. All the data used here is downloaded from the sfu computing cluster. We needed to combine the right data (plot, ratings, cast) so we used merge. We merged Wikipedia data, OMDB data and rotten tomatoes data to create a new pandas data frame which consists of a plot summary of each movie. The plot column is passed into a text\_process function which is created by us to get away with all punctuation, stop words and get a cleaned version of the plot. This cleaned plot is passed into a Textblob library function that calculates and stores the polarities based on the plot from -1 to 1. We classified polarity into 4 nature categories: Highly positive, positive, negative and highly negative using the give\_sentiment function created by us that takes input as polarity and returns the respective nature. The new column movie\_nature is then introduced to store the nature of the movie. This took about 3 minutes of computing time to calculate every time we ran the program so we stored the results in the form of a CSV file: omdb\_clean and used it thereafter to save time and effort. (The previous code is commented out in ml1.py if you wish to have a look)

We extracted the cast members of all movies in the data frame and dropped any movies with less than 3 cast members. First three cast members were taken into account and saved in their respective new columns. We merged the required data into a movies data frame and put it in a CSV to better understand and analyze the data and figure out what data classification would look decent. Ended up using a support vector classifier model with a pipeline to use StandardScaler() to normalize the distribution. Overall both testing and training scores are around 0.67 which clearly means we are not overfitting or underfitting the data.

I think I could have managed to get away with some warnings that show up while running the program. Also, I was expecting the model score to be at least a little over 0.70 which is achievable if I reduce the categories for polarity but that will not be very ethical since we would be biasing our test or prediction only to get the results we want to see. Some of the rows in our final data frame were messed up so dropped them. If I had more time and relevant data I would also take in the money that movie has made relative to the money spent on making it analyze the success of it.

Part 3: This part is to predict the polarity of a movie based on the plot that the movie has got.

Our main goal for this task was to calculate the sentiment value, i.e polarity which we had categorized into 5 different fields, from the omdb movie plot. All the data used here is downloaded from the sfu computing cluster. We used omdb-data. Specifically, we used the column named omdb\_plot to calculate the polarity using natural language processing tool, Textblob. Later on, we divided the movies into 5 different categories as per their polarities. Since, we had decided to work on natural language processing therefore, taking plot as input seemed to be the best fit because movie\_plot being a string can be very resourceful in extracting a lot of features.

Text Pre-processing

As a first step, we wrote a function that will split a message into its individual words and return a list. We'll also remove very common words, ('the', 'a', etc..). To do this we will take advantage of the NLTK library. It's pretty much the standard library in Python for processing text and has a lot of useful features. We'll only use some of the basic ones here. We created a function, text\_process, that will process the string in the omdb\_plot column. First of all, we took the whole string/plot and deleted all the punctuations from it and separate all remaining characters as individuals

Later we joined all those characters to make strings which are now without any punctuation

Since we are done with the punctuation we need to remove all the stop words from those substrings or the bigger string

What are stopWords?

In a very generalized way, those words which are used as helpers in common English language **NLTK** module contains a list of stop words that we are quite common.

After removing all the punctuations and the stop words we separate those substrings and make them distribute in a list. This process is known as tokenization and all those substrings which are separated are known as tokens.

## Vectorization

Currently, we have the messages as string of tokens and now we need to convert each of those plots into a vector, the SciKit Learn's algorithm models can work with. Therefore, we converted each plot, represented as a list of tokens, into a vector that machine learning models can understand.

We did that in three steps, We first used SciKit Learn's **CountVectorizer**. This model converted a collection of text documents to a matrix of token counts. Scikit learn will output this matrix as a sparse matrix, which we can imagine as a 2-D matrix. Where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents. After the counting, the term weighting and normalization can be done with TF-IDF, using scikit-learn's TfidfTransformer.

TF-IDF

The tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document.

The tf-ifd weight is composed of two terms Term frequency (TF) and Inverse document frequency(IDF).

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

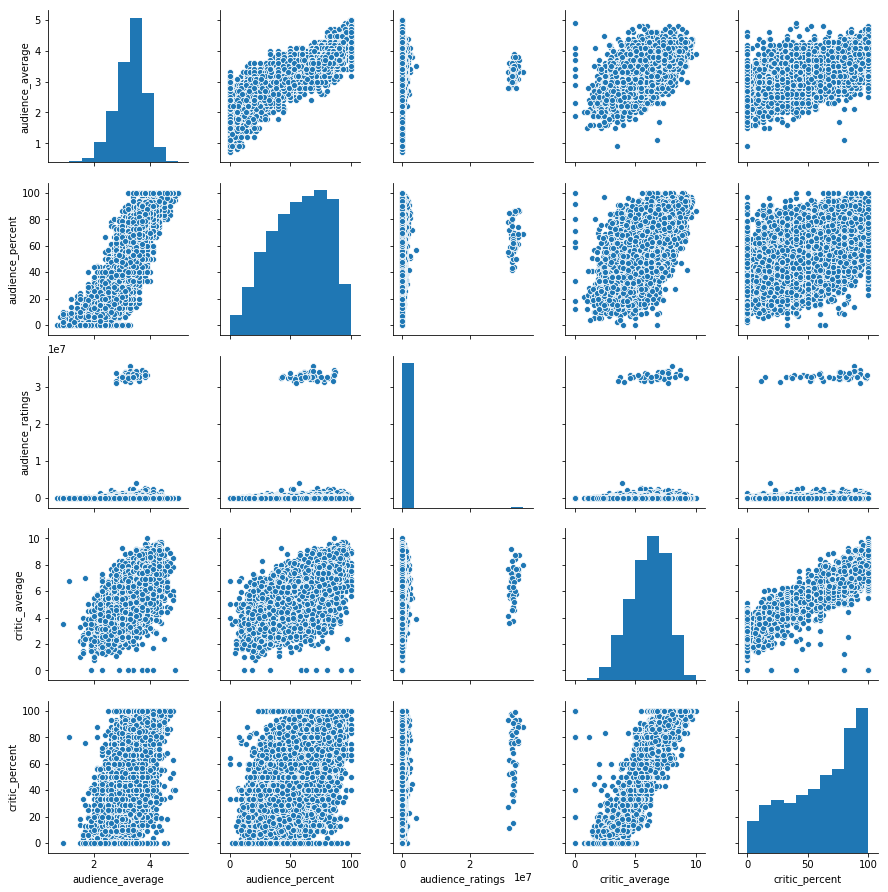
IDF(t) = log\_e(Total number of documents / Number of documents with term t in it).

With movie plots represented as vectors, we could finally train our polarity classifier. We have used SVC classifier model in order to carry out our workflow. We decided to use SVC model as it can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes, which we thought would be the most suitable fit for our ml program Overall both testing and training scores are around 0.67 which clearly means we are not overfitting or underfitting the data.

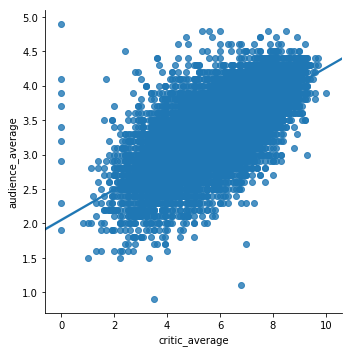
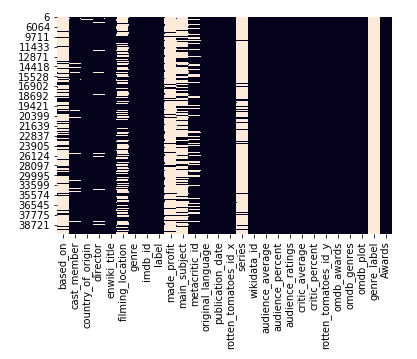
We could have improved this model in terms of time consumption and score. It takes an overall time of 5-8 minutes. We tried to save all the tokens and sparse matrix in a CSV to save some time but that approach didn’t turn out to be a success. In terms of score, we were expecting a score of at least a little over 0.70 which is achievable if we reduce the categories for polarity but that will not be very ethical since we would be biasing our test or prediction only to get the results we want to see. We could have better x values into the model to help predict the polarity. if we had time maybe the season could help depict the polarity of the movies better.

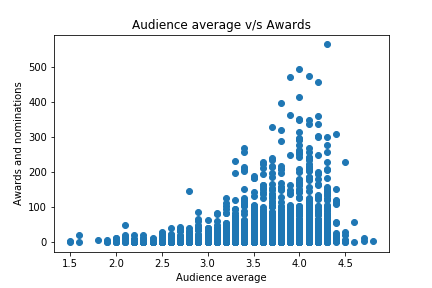
Part 4: This part focuses over analyzing the behaviour between movie ratings and movie awards through visual interpretation

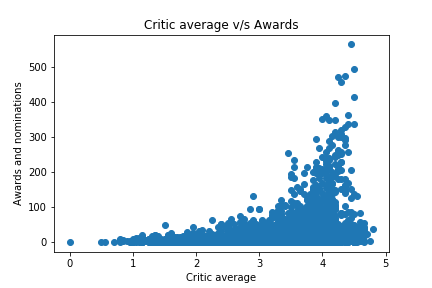
Firstly we merged all the data frames in order to get familiar with the provided data. For this question, we used rotten-tomatoes and omdb data frames. Using seaborn pairplot we were able to decide which feature of rotten tomatoes should we use to analyze the behavior of awards.



Hence we were able to conclude that since critic average and audience average are quite related to each other we decided to use those two columns. The figure below shows the linear regressed line between the data points of the critic and audience average. Later, we made a new column named ‘awards’ which extracts the total number of awards from each movies using regex expression. In order to clean the data, we simply dropped all the null and NaN values. To verify that we don’t have any null values in the required fields, we used seaborn heatmaps. The figure below to the bottom of this page shows that the null values were successfully dropped.





After extracting and cleaning all the required data we plotted the scatter plots of audience and critic rating with the number of awards won. It can be effectively depicted that since critic average are more strictly skewed towards the left, therefore, it is reasonable to conclude that critic average has more effect over the winning and nomination of awards.

## Project Experience Summary

Sarb

- Led brainstorming sessions to come up with ideas for the project

- Organized remote repository to keep files consistent among team members

- Cleaned and transformed data in spark for part 1 of the project

- Plotted the data using matplot in order to visualize the differences between ratings of first and second movies in a series

- Performed a t-test and recorded the conclusion from the results in part 1

- Contributed to the final report

Sidharth

- Scheduled meeting times for the group and made a timeline to follow throughout

- Organized git repository with other team members

- Merged all the given data and wrote it to CSV to better understand the domain of the problem

- Cleaned and transformed the data in pandas for part-2

- Used heatmap from the seaborn library to visualize data

- Built a machine learning model to predict the polarity category of a movie based on cast members and critic ratings

- Normalized the data in part4

- Contributed to Project report

Ronit

- Cleaned the data in part 2,3,4 of the project

- Plotted the data in part4 using seaborn’s heatmaps, pair plots and line plot

- Calculated the polarity and extracted the desired cast members in part 2

- Tokenized and the vectorized the movie plots

- Built a machine learning model to predict the polarity category of a movie based on the tokenized plots

- Organised git repository for effective team workflow

- Used natural language processing to calculate the sentiments and TF-IDF weight of movie plots from rotten tomatoes in part4