StreamOracle : Final Report

Capstone Project by Arjun Lokur
10th April 2023

Table of Contents:

Problem statement:

This capstone project aims to predict the quality and popularity for a movie or TV show, based on features like the cast, plot summary, genres etc. The intent is to help streaming platforms decide whether or not to greenlight/acquire a new piece of content. The quality and popularity are measured by IMDB rating and number of IMDB votes respectively. (*Note that this project is relatively unique because there's 2 target variables.)*

This project is important because streaming companies are under a lot of financial pressure in 2023. Netflix hit its all-time market cap high in Nov 2021 ($314 Bn) driven by pandemic-related lockdowns but has since plummeted to $145 Bn. It's facing subscriber loss and has announced plans to launch a lower-priced, ad-supported tier. Part of the reason is more competition in a now-crowded streaming space. Another reason is that companies (across the tech sector) bet on pandemic-era trends being more permanent, which didn't necessarily happen.

Whatever the reasons, streamers now have a much smaller margin for error in their selection of projects. This project aims to help them make better decisions.

Background on the subject matter:

Given the right data, it would be entirely possible to predict the success of a movie or TV show. Studios are already doing this - have a look at this Verge article about LA based start-up Cinelytic that is advising Hollywood on which movies to make.

Details on dataset:

The primary dataset was put together by Kaggle contributor Shivam Bansal. He scraped the website Flixable – (a content search engine of sorts) – to put together title and credits information for all the content on Netflix, HBOMax, Prime Video and Disney Plus. The Netflix dataset can be found here, and the others can be found on that page easily.

Titles dataset features and explainer:

id	title	type	description	release_year, age_certification,	runtime
Unique identifier for that content title	Name of content	Movie or TV Show	Brief plot summary	Release Year and Age Certification	Runtime in minutes. For TV shows this is episode length

seasons	production_country	imdb_score	imdb_votes
Number of seasons (for TV shows only)	Country or countries that produced the content	The rating on IMDB. One of our target columns, a measure of quality.	Number of votes on IMDB. Our other target column, a measure of popularity

Credits dataset features and explainer:

person_id	id	name	role	character
Unique identifier for that person	Unique identifier for that content title – matches the id column from titles dataset	Name of the person	Whether Actor or Director	Name of character they are playing (if actor)

Summary of cleaning:

1. **Dropping unnecessary columns**: The columns tmdb_score and tmdb_popularity were present as an alternate rating system to IMDB. After going through them, I found these columns unreliable (random scores and popularity) and dropped them.

2. **Null values:**
   a) **Target columns**: Around 11% of the dataset had null values in both our target columns IMDB score and votes. I opted to drop these rows instead of spending time scraping the data.
   b) **Age-certification**: This had 51% null values, which was too high a number to try and fill it in basis the other rows. I dropped this column.
   c) **Seasons**: The null values here implied the content was a movie, so I filled null values with 0 (as movies don't have seasons)

3. **Duplicates**: There were duplicate rows in the all_credits table because some titles were present on multiple platforms. As it wasn't important (for the purposes of this project) which platform had what title, I dropped the duplicates.

Summary of Feature Engineering:

1. **Extracting actor Value from Credits table**:
   a) I wanted to identify 'superstars' because logically, the popularity of a title increases with the presence of a superstar. To do this, I saw that the actor names in the credits table, for a given title, were listed in order of importance. I then ranked the cast for each movie based on this order, which leads us to our formula for value:

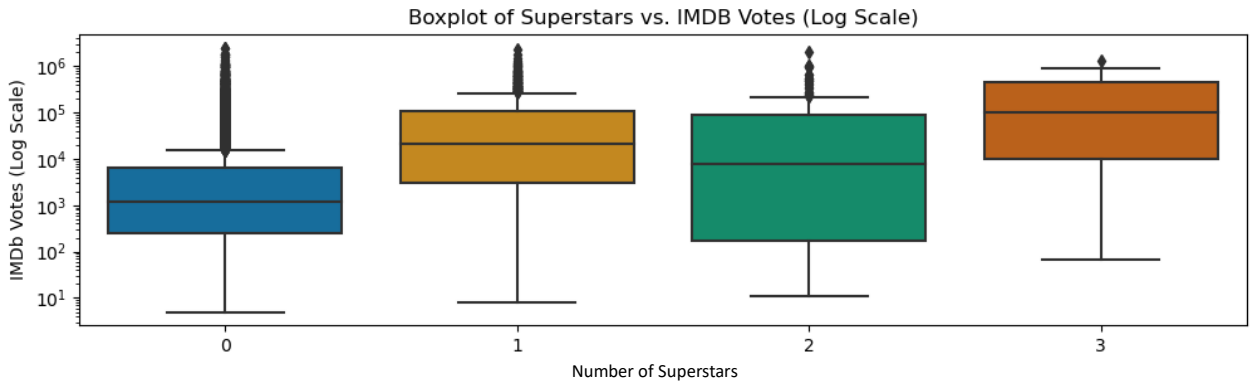
   b) **Formula for Value**:

$$\text{Value} = \frac{\text{Number of titles actor/actress was in}}{\text{Mean rank without outliers}}$$

This captures the actors and actresses who've been in a lot of movies and were the most important actors for that movie. It does a reasonable job of capturing superstars. I created cut-off values to classify the actors into 4 different categories with the following count:

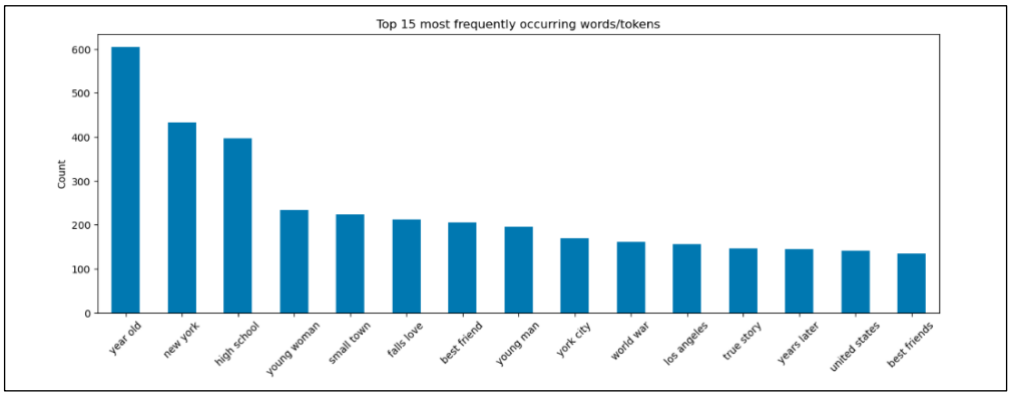
Superstars	All-Stars	Supporting_Actors	Other_Actors
71	134	629	143344

This distribution makes sense, because Superstars should be the rarest. The 'Other_Actors' is counting the number of 'extras' in a movie or TV show. It's a useful measure of the scale of a movie - a big budget production like 'Gladiator' will have many more extras than a smaller arthouse film.



The above chart tells us that in general, adding 'superstars' (as captured by my metric) to a movie/show increases its popularity. The median value of number of votes (which is the line dividing all the boxes in the middle) trends upwards with number of superstars, peaking at 3.

2. **Converting text columns to data:** The two text columns description and title were count-vectorized to find the most commonly occurring words or phrases to show up while removing common words like 'the', 'a, 'an' etc. Here are the top 15 2-word sequences for the Description column.



'New York' seems to be a popular setting for content (also shows up as 'York City'). There are clearly a lot of World War movies. 'Small town' speaks to a certain type of Hallmark movie. The most frequent one 'Year Old' refers to a lot of the plot descriptions mentioning the protagonist's age.

3. **Encoding** : The columns genres and production_countries were encoded, like the below movie example (there are actually 19 different unique genres).

genres
['fantasy', 'action', 'comedy']

drama	fantasy	thriller	action	comedy	war
0	1	0	1	1	0

Modifying the target variables:

1. **IMDB Votes**: Based on the skewed distribution of IMDB_votes (many of which were clumped between 0-3000), I converted the target into its natural log values to make it easier for the models.

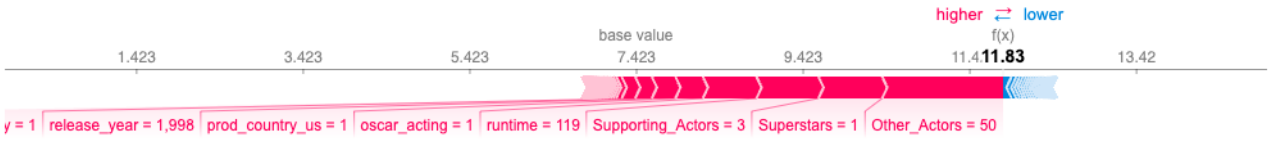
2. **IMDB Score**: After low scores initially as a regression problem, I realized it didn't matter what the exact score was, as people think of IMDB scores in categories, something like the below. So this became a classification problem.

0 - 5	**Class 0:** Never going to watch	5 - 6.5	**Class 1:** If there's nothing else on	6.5 - 7.5	**Class 2:** Might be interested	7.5 - 8.5	**Class 3:** Good to great movie	8.5+	**Class 4:** Excellent movie

Modeling:

IMDB Votes:

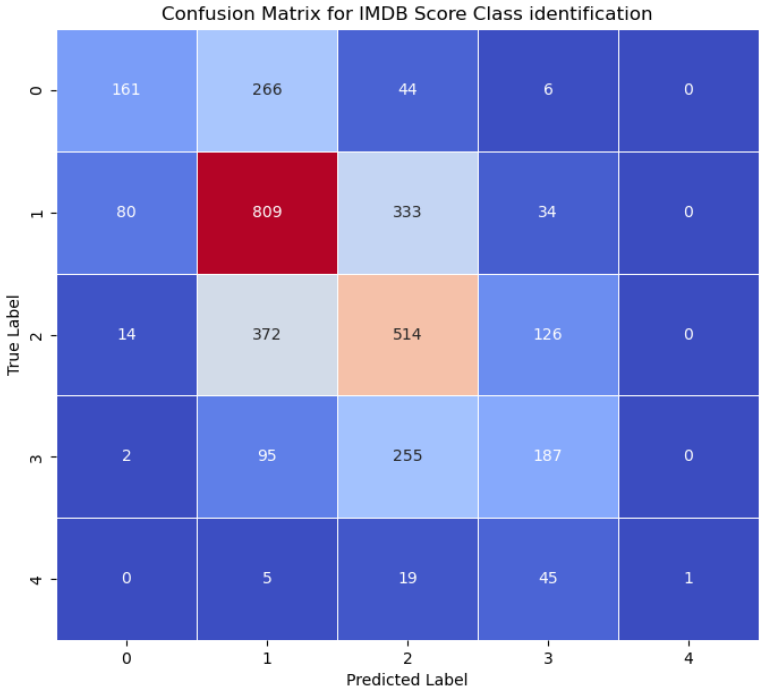
1. **Baseline** linear regression score of **0.455**.
2. **Models tried**: Lasso and Ridge linear regression, K Nearest Neighbours, Decision Trees. Then ensemble methods – Random Forests, ADABoost, and XGBoost. Finally, I tried Neural Networks with a different approach on the text columns 'Description' and 'Title' – using word embeddings.
3. **Best Scores**: The best scores in the initial testing were XGBoost and Random Forest. Tuning the XGBoost model further using Gridsearch, the best model scored **0.607**.
4. **Test score**: And finally, the model's score on the test data was **0.602**, which is quite close to the validation score.
5. **Model Interpretation**: The below visual is for the movie **'You've Got Mail'**. The model predicts a higher than average number of IMDB votes (log value) of 11.83, driven by several features including a big cast (captured by Other_Actors), the presence of a Superstar (Tom Hanks), notable supporting actors (Meg Ryan should have been captured as a superstar, but wasn't by this metric), and the presence of an acting Oscar within the cast (again, Hanks).



IMDB Score:

1. Classification problem, with a baseline accuracy of 0.37 (the frequency of the most frequent class).
2. **Models tried**: Logistic Regression, K Nearest neighbours, Support Vector Machines and finally XGBoost
3. **Best Score**: XGBoost was the best model with a score of **0.5** on the validation data after fine-tuning.
4. **Test Score**: Very close to the validation, coming in at **0.496.**
5. **Further model evaluation:** Let's look at the Confusion Matrix and the Precision/Recall scores. This tells us which how many instances of each class the model misclassified, and as what. Precision means what % of a predicted class are actually of that class, and Recall means what % of the actual class did the model correctly predict.

   What the below visual tells us is that the model was best at predicting Class 2 (IMDB rating 6.5 – 7.5) and the worst at predicting Class 4 (IMDB rating 8.5+). In fact, out of 70 titles in the test set that had an IMDB score of above 8.5, it only correctly identified 1 of them.



Class	Precision	Recall
0	0.63	0.34
1	0.52	0.64
2	0.44	0.50
3	0.47	0.35
4	1.00	0.01

Conclusion:

- The model did about as well as I expected. I did expect that the popularity would be easier to predict than the rating, which turned out to be the case.
- Ultimately, to get a higher accuracy score the data needs to be more detailed (properly capturing the value of a particular actor or director, or even a script) and we need more data for the model to train on, including movies and TV shows that are not on streaming platforms (or are on other streaming platforms outside the top 4).
- The models would probably work best if they were language/region specific – meaning there's a lot of nuance in predicting the success of a Bollywood film vs a Korean TV show vs the next Fast and Furious movie.
- Lastly, the model as it is right now would be unfairly biased towards newer actors (who don't show up in the dataset as superstars), but newer actors can create big hits – we need to adjust the model to reflect this.